Pixel: Multi-signatures for Consensus

Manu Drijvers
DFINITY
manu@dfinity.org

Sergey Gorbunov
Algorand and University of Waterloo
sergey@algorand.com

Gregory Neven
DFINITY
gregory@dfinity.org

Hoeteck Wee
Algorand and CNRS, ENS, PSL
hoeteck@algorand.com

ABSTRACT

Multi-signatures enable a group of signers to jointly generate a short and efficiently verifiable signature on a common message. They are commonly used in proof-of-stake and permissioned blockchains, where reaching consensus usually involves a committee of nodes signing the next block. Adaptive corruptions, however, pose a common threat to such designs, because the adversary can corrupt committee members after they certified a block (and possibly after they sold their stake) and use their signing keys to fork the chain by certifying a different block, thereby undermining the main security goal of a blockchain. Forward-secure signatures protect against such attacks by letting signers evolve their keys over time, while keeping the verification key constant. We present Pixel, a pairing-based forward-secure multi-signature scheme optimized for use in blockchains, that achieves substantial savings in bandwidth, storage requirements, and verification effort. Pixel signatures consist of two group elements, regardless of the number of signers, and can be verified using three pairings and one exponentiation; they also support non-interactive aggregation of individual signatures into a multi-signature. We prove our scheme secure in the random-oracle model under a suitable variant of the bilinear Diffie-Hellman inversion problem.

1 INTRODUCTION

Blockchain technologies are quickly gaining popularity for payments, financial applications, and other distributed applications. A blockchain is an append-only public ledger to which anyone can write and read. At the core of the blockchains is a consensus mechanism that allows nodes to agree on changes to the ledger, while ensuring that changes once confirmed cannot be altered; we refer to the latter as the safety requirement. The key question in any blockchain design is: "How to choose and agree on the next block?"

In the first generation of blockchain implementations, such as Bitcoin, Ethereum, Litecoin, the nodes with the largest computational resources choose the next block. These implementations suffer from large computational waste, high transaction costs, low throughput, high latency, and centralization due to the formation of mining pools [11, 19, 37]. To overcome these problems, the current generation of blockchain implementations such as Algorand, Cardano, Ethereum Casper and Dfinity turn to proofs of stake (PoS), where nodes with larger stakes in the system —as measured for instance by the amount of money in their account— are more likely to participate in choosing the next block [14, 17, 21, 24, 26, 29, 35].

At a high level, PoS-based blockchains share the following structure: (a) a committee of selected users runs a consensus sub-protocol to agree on what block B to be added next, (b) each committee member then signs that block B, and (c) each node then appends a block B to their view of the ledger if it sees sufficiently many committee member signatures on the block B. We refer to this collection of committee signatures on the block B as the *block certificate*. The way in which the committees are selected and the consensus sub-protocol varies quite substantially amongst the various designs.

This work. In this work, we focus on the common cryptographic core of all PoS-based blockchains, namely the signature scheme used by the committee, and how we can simultaneously meet the requirements for efficiency and security.

In terms of efficiency, a major cost of PoS protocols are bandwidth and space needed to propagate committee signatures and to store the block certificate, as well as the computational resources needed for signature verification of these certificates. The former can be mitigated with the use multi-signatures [1, 4, 6, 23, 27, 31, 32, 36, 38], where a single short signature validates that a message m was signed by N different parties. Multi-signatures based on the BLS signature scheme [6, 9, 10, 39] are particularly well-suited to the distributed setting of PoS blockchains as no communication is required between the signers; anybody can aggregate individual signatures into a multi-signature.

In terms of security, we require that the signatures be *forward-secure*. That is, each signature is associated with the current time period in addition to the signed message, and after each time interval, a user's secret key can be updated in such a way that it can only be used to sign messages for future time periods, but not previous ones. The use of forward-secure signatures prevents *adaptive* attacks on a PoS-based blockchain, where an adversary waits until the agreement on a block B is reached for a round r, and then at some time in the future, it corrupts all the committee members that signed the block to obtain their signing keys.[1] Using the keys, the adversary can produce a valid certificate for a different block B' for the same round r. Note that this attack is prevented if committee members use a forward-secure signature scheme and update their keys as soon as they sign a block B.

[1] In a typical PoS protocol, a committee is a tiny fraction of the total number of users in the system so that an adaptive adversary can corrupt an entire committee while controlling only a tiny fraction of the total stake. Also, the stakes of the committee members may decrease significantly over time.

scheme	key update	sign	verify	$\|\sigma\|$	$\|pk\|$	$\|sk\|$	forward security
BLS multi-signatures [6, 9, 39]	–	1 exp	2 pair	1	1	$O(1)$	no
Pixel multi-signatures (this work)	2 exp	4 exp	3 pair + 1 exp	2	1	$O((\log T)^2)$	yes

Figure 1: Comparing our scheme with BLS signatures. Here, "exp" and "pair" refer to number of exponentiations and pairings respectively. T denotes the maximum number of time periods. We omit additive overheads of $O(\log T)$ multiplications. The column "key update" refers to amortized cost of updating the key for time t to $t+1$. The columns $|\sigma|$, $|pk|$, and $|sk|$ denote the sizes of signatures, public keys, and secret keys, respectively, in terms of group elements. Aggregate verification for N signatures requires an additional $N-1$ multiplications over basic verification.

1.1 Our Results

We present the Pixel signature scheme, which is a pairing-based forward-secure multi-signature scheme for use in PoS-based blockchains that achieves substantial savings in bandwidth and storage requirements. To support a total of T time periods and a committee of size N, the block certificate comprises just two group elements (in addition to the identities of the committee members), whereas verifying each committee member's signature as well as the block certificate requires only three pairings plus one exponentiation. Pixel signatures are almost as efficient as BLS multi-signatures, as depicted in Figure 1, while also preventing adaptive attacks; moreover, like in BLS multi-signatures, anybody can non-interactively, aggregate individual signatures into a multi-signature. In contrast, using existing forward-secure signature yields much larger block certificate of size $O(N)$ to $O(N \log T)$ group elements [3, 12, 28, 30, 33]; this is the case even if we were to instantiate the tree-based constructions with aggregatable BLS signatures.

Our construction builds on prior forward-secure signatures based on hierarchical identity-based encryption (HIBE) [7, 12, 15, 18] and adds the ability to securely aggregate signatures on the same message. We achieve security under a standard q-type assumption in the random oracle model.

Overview of our scheme. Starting with a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ with $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ of prime order q and generators g_1, g_2 for $\mathbb{G}_1, \mathbb{G}_2$ respectively, a signature on $M \in \mathbb{Z}_q$ at time t under public key g_2^x is of the form:

$$\sigma = (\sigma', \sigma'') = (h^x \cdot F(t, M)^r, g_2^r) \in \mathbb{G}_1 \times \mathbb{G}_2$$

where the function $F(t, M)$ can be computed with some public parameters (two group elements in $\mathbb{G}_1$ in addition to $h \in \mathbb{G}_1$) and r is fresh randomness used for signing. Verification relies on the relation:

$$e(\sigma', g_2) = e(h, y) \cdot e(F(t, M), \sigma'')$$

and completeness follows directly:

$$\begin{aligned} e(\sigma', g_2) &= e(h^x \cdot F(t, M)^r, g_2) \\ &= e(h^x, g_2) \cdot e(F(t, M)^r, g_2) \\ &= e(h, g_2^x) \cdot e(F(t, M), g_2^r) \\ &= e(h, y) \cdot e(F(t, M), \sigma'') \ . \end{aligned}$$

Note that $e(h, y)$ can be precomputed to save verification computation.

Given N signatures $\sigma_1, \ldots, \sigma_N \in \mathbb{G}_1 \times \mathbb{G}_2$ on the same message M at time t under N public keys $g_2^{x_1}, \ldots, g_2^{x_N}$, we can produce a multi-signature Σ on M by computing the coordinate-wise product of $\sigma_1, \ldots, \sigma_N$. Concretely, if $\sigma_i = (h^{x_i} \cdot F(t, M)^{r_i}, g_2^{r_i})$, then

$$\Sigma = (h^{x_1 + \cdots + x_N} \cdot F(t, M)^{r'}, g_2^{r'})$$

where $r' = r_1 + \cdots + r_N$. To verify Σ, we first compute a single *aggregate* public key that is a compressed version of all N individual public keys

$$apk \leftarrow y_1 \cdot \ldots \cdot y_N \ ,$$

and verify Σ against apk using the standard verification equation.

How to generate and update keys. To complete this overview, we describe a simplified version of the secret keys and update mechanism, where the secret keys are of size $O(T)$ instead of $O((\log T)^2)$. The construction exploits the fact that the function F satisfies

$$F(t, M) = F(t, 0) \cdot F'^M$$

for some constant F'. This means that in order to sign messages at time t, it suffices to know

$$\widetilde{sk}_t = \{h^x \cdot F(t, 0)^r, F'^r, g_2^r\}$$

from which we can compute $(h^x \cdot F(t, M)^r, g_2^r)$.

The secret key sk_t for time t is given by:

$$\widetilde{sk}_t, \widetilde{sk}_{t+1}, \cdots, \widetilde{sk}_T$$

generated using independent randomness. To update from the key sk_t to sk_{t+1}, we simply erase $\widetilde{sk}_t$. Forward security follows from the fact that an adversary who corrupts a signer at time t only learns sk_t and, in particular, does not learn $\widetilde{sk}_{t'}$ for $t' < t$, and is unable to create signatures for past time slots.

To compress the secret keys down to $O((\log T)^2)$ without increasing the signature size, we combine the tree-based approach in [15] with the compact HIBE in [7]. Roughly speaking, each sk_t now contains $\log T$ sub-keys, each of which contains $O(\log T)$ group elements and looks like an "expanded" version of $\widetilde{sk}_t$. (In the simplified scheme, each sk_t contains $T - t + 1$ sub-keys, each of which contains three group elements.)

Security against rogue-key attacks. A well-known challenge in constructing secure multi-signature schemes is to avoid rogue-key attacks, where an adversary forges a multi-signature by providing specially crafted public keys that are correlated with the public keys of the honest parties. We achieve security against rogue-key attacks by having users provide a proof of possession of their secret key [6, 39]; it suffices here for each user to provide a standard BLS signature y' on its public key y.

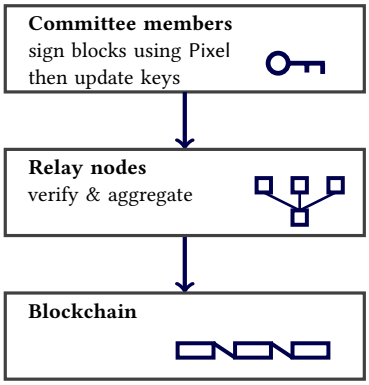

Figure 2: Using Pixel in PoS-based blockchains

1.2 Pixel in PoS-based Blockchains

Next, we describe how to integrate Pixel during consensus in a PoS-based blockchain. The consensus sub-protocol begins by selecting a committee members, each of whom votes on a block B by signing the block B using Pixel with the current block number, and consensus is reached when we see a collection of N committee member signatures $\sigma_1, \ldots, \sigma_N$ on the *same* block B, where N is some fixed threshold, typically in the range of 1000 to 5000. Note that to tolerate malicious committee members, we will need to start with a committee of size much larger than N. Finally, we will aggregate these N signatures into a single multi-signature Σ, and the pair (B, Σ) constitute a so-called block certificate and the block B is appended to the blockchain.

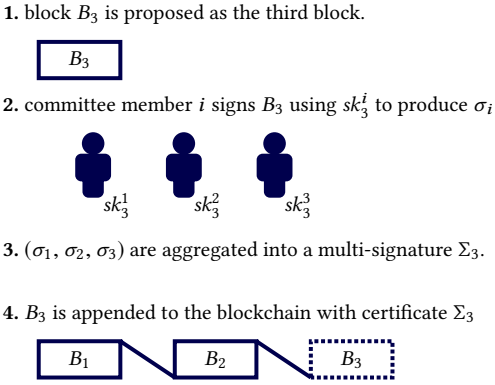

Figure 3: Using Pixel to sign the third block

Propagating signatures. Individual committee signatures will be propagated through the network via so-called relay nodes, until we see N committee member signatures on the same block B. Note that Pixel supports non-interactive and incremental aggregation: the former means that signatures can be aggregated by any party after broadcast without communicating with the original signers, and the latter means that we can incorporate a new signature to an aggregate signature to obtain a new aggregate signature. In practice, this means that relay nodes can perform intermediate aggregation on any number of committee signatures and propagate the result, until the block certificate is formed. In order to speed up verification of individual committee member signatures, a relay node could pre-compute $e(h, y)$ for the y's corresponding to the users with the highest stakes.

Registering public keys. In order to be eligible for selection as a committee member, a user needs to first register a public key y for Pixel along with the corresponding y' for the proof of possession. Registration here means that y must appear in a block B on the blockchain. The committee member that signs the block B is responsible for verifying the proof of possession y' and we stress that only y and not y' appears in the block B, and therefore only y but not y' will stored on the blockchain.

Tweaking the scheme. The blockchain stores Pixel public keys of all eligible committee members, as well as multi-signatures on each block. It is easy to see that we can tweak the Pixel scheme so that public keys live in the group $\mathbb{G}_1$ (which has a more compact description) instead of $\mathbb{G}_2$; this way, we can minimize the size of the blockchain as well as the cost of aggregate verification, which is dominated by the cost of multiplying N public keys for large N. This change does come at a small cost since signing is performed over the slower $\mathbb{G}_2$ instead of $\mathbb{G}_1$. When instantiated with the BLS12-381 pairing-friendly curve, each public key is 48 bytes, and each multi-signature is 48+96=144 bytes independent of N. Moreover, we estimate signing to take less than 3 ms, and signature verification less than 5 ms for $T = 2^{30}$. More details are provided in Section 5.

Key updates. When using Pixel in block-chains, time corresponds to the block number. Naively, this means that all eligible committee members should update their Pixel secret keys for each time a new block is formed and the round number is updated. As it turns out, we can in fact support less frequent updates while still guaranteeing security. Assume for simplicity that each committee member signs at most one block (if not, simply append a counter to the block number and use that as the time). If a user is selected to be on the committee at block number t, it should first update its key to sk_t (Pixel supports "fast-forward" key updates from sk_t to $sk_{t'}$ for any $t' > t$), and as soon as it signs a block, updates its key to sk_{t+1} and then propagates the signature. In particular, there is no need for key updates when a user is not selected to be on the committee, which yields significant savings in computation for users with small stakes. With the infrequent updates, it is possible that an adversary corrupts a user at block number 999 and obtains her Pixel secret key for block number 997, but the adversary is still unable to rewrite block 997 if the user was last selected to be on the committee for block number 996.

Avoiding trusted set-up. Note that the common parameters contain uniformly random group elements $h, h_0, \ldots, h_{\log T}$ in $\mathbb{G}_2$ which are used to define the function F. These elements can be generated using a indifferentiable hash-to-curve algorithm [13, 40] evaluated on some fixed sequence of inputs (e.g. determined by the digits of pi), thereby avoiding any trusted set-up.

1.3 Discussion

Related works. The use of HIBE schemes for forward secrecy originates in the context of encryption [15] and has been used in signatures [12, 18], key exchange [25] and proxy re-encryption [22]. Our signature scheme is quite similar to the forward-secure signatures of Boyen et al. [12] and achieves the same asymptotic complexity; their construction is more complex in order to achieve security against untrusted updates. The way we achieve aggregation is similar to the multi-signatures in [31].

Alternative approaches to adaptive security. There are two variants of the adaptive attack: (i) a short-range variant, where an adversary tries to corrupt a committee member prior to completion of the consensus sub-protocol, and (ii) a long-range variant as explained earlier. Dfinity [26], Ouroboros [29] and Casper [14] cope with the short-range attacks by assuming a delay in attacks that is longer than the running time of the consensus sub-protocol. For long-range attacks, Casper adopts a fork choice rule to never revert a finalized block, and in addition, assumes that clients log on with sufficient regularity to gain a complete update-to-date view of the chain. We note that forward-secure signatures provide a clean solution against both attacks, without the need for fork choice rules or additional assumptions about the adversary and the clients.

Application to permissioned blockchains. Consensus protocols, such as PBFT, are also at the core of many permissioned blockchains (e.g. Hyperledger), where only approved parties may join the network. Our signature scheme can similarly be applied to this setting to achieve forward secrecy, reduce communication bandwidth, and produce compact block certificates.

2 PRELIMINARIES

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ be multiplicative groups of prime order q with an admissible pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

In analogy with the weak bilinear Diffie-Hellman inversion problem ℓ-wBDHI* [8], which was originally defined for Type-1 pairings (i.e., symmetric pairings where we have $\mathbb{G}_1 = \mathbb{G}_2$), we define the following variant for Type-3 pairings denoted ℓ-wBDHI$^*_3$.

$$\begin{aligned}\text{Input: } & A_1 = g_1^{\alpha}\ ,\ A_2 = g_1^{(\alpha^2)}\ ,\ \ldots\ ,\ A_\ell = g_1^{(\alpha^\ell)}\ ,\\ & B_1 = g_2^{\alpha}\ ,\ B_2 = g_2^{(\alpha^2)}\ ,\ \ldots\ ,\ B_\ell = g_2^{(\alpha^\ell)}\ ,\\ & C_1 = g_1^{\gamma}\ ,\ C_2 = g_2^{\gamma}\\ & \text{for } \alpha, \gamma \xleftarrow{\$} \mathbb{Z}_q\\ \text{Compute: } & e(g_1, g_2)^{(\gamma \cdot \alpha^{\ell+1})}\end{aligned}$$

The advantage $\mathbf{Adv}^{\ell\text{-wBDHI}^*_3}_{\mathbb{G}_1 \times \mathbb{G}_2}(\mathcal{A})$ of an adversary $\mathcal{A}$ is defined as its probability in solving this problem.

Alternatively, our scheme could be proved secure under a variant of the above assumption where the adversary has to output $g_1^{(\alpha^{\ell+1})}$ given as input $A_1, \ldots, A_\ell, B_1, \ldots, B_\ell$ and given access to an oracle $\psi : g_2^x \mapsto g_1^x$. Because of the ψ oracle, this assumption is incomparable to the ℓ-wBDHI assumption described above.

3 FORWARD-SECURE SIGNATURES

We begin by describing a forward-secure signature scheme, and then extend the construction to a multi-signature scheme in Section 4.

3.1 Definition

We use the Bellare-Miner model [3] to define syntax and security of a forward-secure signature scheme. A forward-secure signature scheme $\mathcal{FS}$ for a message space $\mathcal{M}$ consists of the following algorithms:

Setup: $pp \xleftarrow{\$} \mathsf{Setup}(T)$. All parties agree on the public parameters pp. The setup algorithm mainly fixes the distribution of the parameters given the maximum number of time periods T. The parameters may be generated by a trusted third party, through a distributed protocol, or set to "nothing-up-my-sleeve" numbers. The public parameters are taken to be an implicit input to all of the following algorithms.

Key generation: $(pk, sk_1) \xleftarrow{\$} \mathsf{Kg}$. The signer runs the key generation algorithm on input the maximum number of time periods T to generate a public verification key pk and an initial secret signing key sk_1 for the first time period.

Key update: $sk_{t+1} \xleftarrow{\$} \mathsf{Upd}(sk_t)$. The signer updates its secret key sk_t for time period t to sk_{t+1} for the next period using the key update algorithm. The scheme could also offer a "fast-forward" update algorithm $sk_{t'} \xleftarrow{\$} \mathsf{Upd}'(sk_t, t')$ for any $t' > t$ that is more efficient than repetitively applying Upd.

Signing: $\sigma \xleftarrow{\$} \mathsf{Sign}(sk_t, M)$. On input the current signing key sk_t and message $M \in \mathcal{M}$, the signer uses this algorithm to compute a signature σ.

Verification. $b \leftarrow \mathsf{Vf}(pk, t, M, \sigma)$. Anyone can verify a signature σ for on message M for time period t under public key pk by running the verification algorithm, which returns 1 to indicate that the signature is valid and 0 otherwise.

Correctness. Correctness requires that for all messages $M \in \mathcal{M}$ and for all time periods $t \in [T]$ it holds that

$$\Pr[\mathsf{Vf}(pk, t, M, \mathsf{Sign}(sk_t, M)) = 1] = 1$$

where the coin tosses are over $pp \xleftarrow{\$} \mathsf{Setup}(T)$, $(pk, sk_1) \xleftarrow{\$} \mathsf{Kg}$, and $sk_i \leftarrow \mathsf{Upd}(sk_{i-1})$ for $i = 2, \ldots, t$.

Moreover, if the scheme has a fast-forward update algorithm, then the keys it produces must be distributed identically to those produced by repetitive application of the regular update algorithm. Meaning, for all $t, t' \in [T]$ with $t < t' \leq T$ and for all sk_t it holds that $sk'_{t'} \xleftarrow{\$} \mathsf{Upd}'(sk_t, t')$ follows the same distribution as sk_t produced as $sk_i \xleftarrow{\$} \mathsf{Upd}(sk_{i-1})$ for $i = t+1, \ldots, t'$.

Security. Unforgeability under chosen-message attack for forward-secure signatures is defined through the following game. The experiment generates a fresh key pair (pk, sk_1) and hands the public key pk to the adversary $\mathcal{A}$. The adversary is given access to the following oracles:

Key update. If the current time period t (initially set to $t = 1$) is less than T, then this oracle updates the key sk_t to sk_{t+1} and increases t.

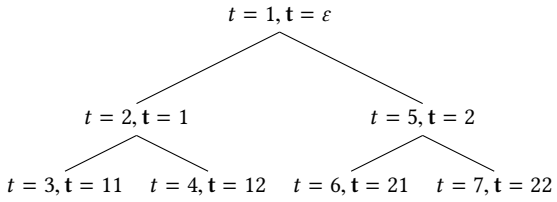

Figure 4: Tree structure illustrating bijection between $t \in [2^\ell]$ and $\mathbf{t} \in \{1,2\}^{\leq \ell-1}$ for $\ell = 3$.

Signing. On input a message M, this oracle runs the signing oracle with the current secret key sk_t and message M, and returns the resulting signature σ.

Break in. The experiment records the break-in time $\bar{t} \leftarrow t$ and hands the current signing key $sk_{\bar{t}}$ to the adversary. This oracle can only be queried once, and after it has been queried, the adversary can make no further queries to the key update or signing oracles.

At the end of the game, the adversary outputs its forgery (t^*, M^*, σ^*). It wins the game if σ^* verifies correctly under pk for time period t^* and message M^*, if it never queried the signing oracle on M^* during time period t^*, and if it queried the break-in oracle, then it did so in a time period $\bar{t} > t^*$. We define $\mathcal{A}$'s advantage $\mathbf{Adv}^{\text{fu-cma}}_{\mathcal{FS}}(\mathcal{A})$ as its probability in winning the above game.

We also define a selective variant of the above notion, referred to as sfu-cma, where the adversary first has to commit to $\bar{t}$, t^*, and M^*. More specifically, $\mathcal{A}$ first outputs $(\bar{t}, t^*, M^*)$, then receives the public key pk, is allowed to make signature and key update queries until time period $t = \bar{t}$ is reached, at which point it is given $sk_{\bar{t}}$ and outputs its forgery σ^*.

3.2 Encoding time periods

We describe a bijection between $\mathbf{t} = t_1 \| t_2 \| \ldots \in \{1,2\}^{\leq \ell-1}$ and $t \in [2^\ell - 1]$ for any integer ℓ given by

$$t(\mathbf{t}) = 1 + \sum_{i=1}^{|\mathbf{t}|} (1 + 2^{\ell-i}(t_i - 1)) \,.$$

For instance, for $\ell = 3$, this maps ε, 1, 11, 12, 2, 21, 22 to 1, 2, 3, 4, 5, 6, 7. The inverse of the bijection can be described as

$$\begin{aligned} \mathbf{t}(1) &= \varepsilon \\ \mathbf{t}(t) &= \mathbf{t}(t-1)\|1 && \text{if } |\mathbf{t}(t-1)| < \ell - 1 \\ \mathbf{t}(t) &= \bar{\mathbf{t}}\|2 && \text{if } |\mathbf{t}(t-1)| = \ell - 1 \end{aligned}$$

where $\bar{\mathbf{t}}$ is the longest string such that $\bar{\mathbf{t}}\|1$ is a prefix of $\mathbf{t}(t-1)$.

The bijection induces a natural precedence relation over $\{1,2\}^{\leq \ell-1}$ where $\mathbf{t} \preceq \mathbf{t}'$ iff either $\mathbf{t}$ is a prefix of $\mathbf{t}'$ or exists $\bar{\mathbf{t}}$ s.t. $\bar{\mathbf{t}}\|1$ is a prefix of $\mathbf{t}$ and $\bar{\mathbf{t}}\|2$ is a prefix of $\mathbf{t}'$. We also write $\mathbf{t}, \mathbf{t}+1$ corresponding to $t, t+1$.

When interpreting $\mathbf{t} \in \{1,2\}^{\leq \ell-1}$ as a path to a node in a binary tree, where 1 denotes taking the left branch and 2 denotes taking the right branch, then this precedence relation corresponds to a pre-order traversal of the tree. An example of such a tree is depicted in Figure 4.

Next, we associate any $\mathbf{t} \in \{1,2\}^{\leq \ell-1}$ with a set $\Gamma_{\mathbf{t}} \subset \{1,2\}^{\leq \ell-1}$ given by

$$\Gamma_{\mathbf{t}} := \{\, \mathbf{t} \,\} \cup \{\, \bar{\mathbf{t}}\|2 : \bar{\mathbf{t}}\|1 \text{ prefix of } \mathbf{t} \,\}$$

that corresponds to the set containing $\mathbf{t}$ and all the right-hand siblings of nodes on the path from $\mathbf{t}$ to the root, which also happens to be the smallest set of nodes that includes a prefix of all $\mathbf{t}' \succeq \mathbf{t}$. For instance, for $\ell = 3$, we have

$$\Gamma_1 = \{1,2\}, \Gamma_{11} = \{11,12,2\}, \Gamma_{12} = \{12,2\} \,.$$

The sets $\Gamma_{\mathbf{t}}$ satisfy the following properties:

- $\mathbf{t}' \succeq \mathbf{t}$ iff there exists $\mathbf{u} \in \Gamma_{\mathbf{t}}$ s.t. $\mathbf{u}$ is a prefix of $\mathbf{t}'$;
- For all $\mathbf{t}$, we have $\Gamma_{\mathbf{t}+1} = \Gamma_{\mathbf{t}} \setminus \{\mathbf{t}\}$ if $|\mathbf{t}| = \ell - 1$ or $\Gamma_{\mathbf{t}+1} = (\Gamma_{\mathbf{t}} \setminus \{\mathbf{t}\}) \cup \{\, \mathbf{t}\|1, \mathbf{t}\|2 \,\}$ otherwise;
- For all $\mathbf{t}' \succ \mathbf{t}$, we have that for all $\mathbf{u}' \in \Gamma_{\mathbf{t}'}$, there exists $\mathbf{u} \in \Gamma_{\mathbf{t}}$ such that $\mathbf{u}$ is a prefix of $\mathbf{u}'$.

The first property is used for verification and for reasoning about security; the second and third properties are used for key updates.

3.3 Construction

We assume the bound T is of the form $2^\ell - 1$. We use the above bijection so that the algorithms take input $\mathbf{t} \in \{1,2\}^{\leq \ell-1}$ instead of $t \in [T]$. The following scheme is roughly the result of applying the Canetti-Halevi-Katz technique to obtain forward security from hierarchical identity-based encryption (HIBE) [16] to the signature scheme determined by the key structure of the Boneh-Boyen-Goh HIBE scheme [7]; we describe the differences at the end of this subsection.

Setup. Let $\mathcal{M}$ be the message space of the scheme and let $\mathsf{H}_q : \mathcal{M} \to \{0,1\}^\kappa$ be a hash function that maps messages to bit strings of length κ such that $2^\kappa < q$. Apart from the description of the groups, the common system parameters also contain the maximum number of time slots $T = 2^\ell - 1$ and random group elements $h, h_0, \ldots, h_\ell \xleftarrow{\$} \mathbb{G}_1$. These parameters could, for example, be generated as the output of a hash function modeled as a random oracle.

Key generation. Each signer chooses $x \xleftarrow{\$} \mathbb{Z}_q$ and computes $y \leftarrow g_2^x$. It sets its public to $pk = y$ and computes its initial secret key $sk_1 \leftarrow \{\widetilde{sk}_\varepsilon\}$ where

$$\widetilde{sk}_\varepsilon = (g_2^r \,,\, h^x h_0^r \,,\, h_1^r, \ldots, h_\ell^r)$$

for $r \xleftarrow{\$} \mathbb{Z}_q$.

Key update. We associate with each $\mathbf{w} \in \{1,2\}^k$ with $k \leq \ell - 1$ a key $\widetilde{sk}_{\mathbf{w}}$ of the form

$$\begin{aligned} \widetilde{sk}_{\mathbf{w}} &= (c, d, e_{k+1}, \ldots, e_\ell) \\ &= \left(g_2^r \,,\, h^x (h_0 \prod_{j=1}^{k} h_j^{w_j})^r \,,\, h_{k+1}^r \,,\, \ldots \,,\, h_\ell^r \right) \end{aligned} \quad (1)$$

for $r \xleftarrow{\$} \mathbb{Z}_q$. Given $\widetilde{sk}_{\mathbf{w}}$, one can derive a key for any $\mathbf{w}' \in \{1,2\}^{k'}$ which contains $\mathbf{w}$ as a prefix as

$$(c', d', e'_{k'+1}, \ldots, e'_\ell) = \left(c \cdot g_2^{r'} \,,\, d \cdot \prod_{j=k+1}^{k'} e_j^{w_j} \cdot (h_0 \prod_{j=1}^{k'} h_j^{w_j})^{r'} \,,\, e_{k'+1} \cdot h_{k'+1}^{r'} \,,\, \ldots \,,\, e_\ell \cdot h_\ell^{r'} \right) \quad (2)$$

for $r' \xleftarrow{\$} \mathbb{Z}_q$.

The secret key $sk_{\mathbf{t}}$ at time period $\mathbf{t}$ is given by

$$sk_{\mathbf{t}} = \{\widetilde{sk}_{\mathbf{w}} : \mathbf{w} \in \Gamma_{\mathbf{t}}\}\,,$$

which, by the first property of $\Gamma_{\mathbf{t}}$, contains a key $\widetilde{sk}_{\mathbf{w}}$ for a prefix $\mathbf{w}$ of all nodes $\mathbf{t}' \geq \mathbf{t}$.

To perform a regular update of $sk_{\mathbf{t}}$ to $sk_{\mathbf{t}+1}$, the signer users the second property of $\Gamma_{\mathbf{t}}$. Namely, if $|\mathbf{t}| < \ell - 1$, then the signer looks up $\widetilde{sk}_{\mathbf{t}} = (c, d, e_{|\mathbf{t}|+1}, \ldots, e_\ell) \in sk_{\mathbf{t}}$, computes

$$\widetilde{sk}_{\mathbf{t}\|1} \leftarrow (c, d \cdot e_{|\mathbf{t}|+1}, e_{|\mathbf{t}|+2}, \ldots, e_\ell)\,,$$

and derives $\widetilde{sk}_{\mathbf{t}\|2}$ from $\widetilde{sk}_{\mathbf{t}}$ using Equation (2). The signer then sets $sk_{\mathbf{t}+1} \leftarrow (sk_{\mathbf{t}} \setminus \widetilde{sk}_{\mathbf{t}}) \cup \{\widetilde{sk}_{\mathbf{t}\|1}, \widetilde{sk}_{\mathbf{t}\|2}\}$ and securely deletes $sk_{\mathbf{t}}$ as well as the re-randomization exponent r' used in the derivation of $\widetilde{sk}_{\mathbf{t}\|2}$.

If $|\mathbf{t}| = \ell - 1$, then the signer simply sets $sk_{\mathbf{t}+1} \leftarrow sk_{\mathbf{t}} \setminus \{\widetilde{sk}_{\mathbf{t}}\}$ and securely deletes $sk_{\mathbf{t}}$.

To perform a fast-forward update of its key to any time $\mathbf{t}' \geq \mathbf{t}$, the signer derives keys $\widetilde{sk}_{\mathbf{w}'}$ for all nodes $\mathbf{w}' \in \Gamma_{\mathbf{t}'} \setminus \Gamma_{\mathbf{t}}$ by applying Equation (2) to the key $\widetilde{sk}_{\mathbf{w}} \in sk_{\mathbf{t}}$ such that $\mathbf{w}$ is a prefix of $\mathbf{w}'$, which must exist due to the third property of $\Gamma_{\mathbf{t}}$. The signer then sets $sk_{\mathbf{t}'} \leftarrow \{\widetilde{sk}_{\mathbf{w}'} : \mathbf{w}' \in \Gamma_{\mathbf{t}'}\}$ and securely deletes $sk_{\mathbf{t}}$ as well as all re-randomization exponents used in the key derivations.

Signing. To generate a signature on message $M \in \mathcal{M}$ in time period $\mathbf{t} \in \{1,2\}^{\leq \ell-1}$, the signer looks up

$$\widetilde{sk}_{\mathbf{t}} = (c, d, e_{|\mathbf{t}|+1}, \ldots, e_\ell) \in sk_{\mathbf{t}}\,,$$

chooses $r' \xleftarrow{\$} \mathbb{Z}_q$, and outputs

$$(\sigma_1, \sigma_2) \;=\; \left(d \cdot e_\ell^{\mathsf{H}_q(M)} \cdot \big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}|} h_j^{t_j} \cdot h_\ell^{\mathsf{H}_q(M)}\big)^{r'},\; c \cdot g_2^{r'}\right).$$

Verification. Anyone can verify a signature $(\sigma_1, \sigma_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ on message M under public key $pk = y$ in time period $\mathbf{t}$ by checking whether

$$\mathrm{e}(\sigma_1, g_2) \;=\; \mathrm{e}(h, y) \cdot \mathrm{e}\big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}|} h_j^{t_j} \cdot h_\ell^{\mathsf{H}_q(M)},\; \sigma_2\big)\,.$$

Note that the pairing $\mathrm{e}(h, y)$ can be pre-computed from the public key ahead of time, so that verification only requires two pairing computations.

Differences from prior works. We highlight the differences between our scheme and those in [7, 12, 15], assuming some familiarity with these prior constructions.

- We rely on asymmetric bilinear groups for efficiency, and our signature sits in $\mathbb{G}_2 \times \mathbb{G}_1$ instead of $\mathbb{G}_2^2$. This way, it is sufficient to give out the public parameters $h_0, \ldots, h_\ell$ in $\mathbb{G}_1$ (which we can then instantiate using hash-to-curve without trusted set-up) instead of having to generate "consistent" public parameters $(h_i, h_i') = (g_1^{x_i}, g_2^{x_i}) \in \mathbb{G}_1 \times \mathbb{G}_2$.
- Our key-generation algorithm also deviates from that in the Boneh-Boyen-Goh HIBE, which would set

$$pk = \mathrm{e}(g_1, g_2)^x, h = g_1, \widetilde{sk}_\varepsilon = \big(g_2^r, g_1^x h_0^r, h_1^r, \ldots, h_\ell^r\big).$$

3.4 Correctness

We say that a secret key $sk_{\mathbf{t}}$ for time period $\mathbf{t}$ is *well-formed* if $sk_{\mathbf{t}} = \{\widetilde{sk}_{\mathbf{w}} : \mathbf{w} \in \Gamma_{\mathbf{t}}\}$, where each $\widetilde{sk}_{\mathbf{w}}$ is of the form of Equation (1) for an independent uniformly distributed exponent $r \xleftarrow{\$} \mathbb{Z}_q$. We first show that all honestly generated and updated secret keys are well-formed, and then proceed to the verification of signatures.

The key $sk_{\mathbf{t}}$ is trivially well-formed for $t = 1$, i.e., $\mathbf{t} = \varepsilon$, as can be seen from the key generation algorithm. We now show that $sk_{\mathbf{t}}$ is also well-formed after a regular update from time $\mathbf{t}$ to $\mathbf{t} + 1$ and after a fast-forward update from $\mathbf{t}$ to $\mathbf{t}' > \mathbf{t}$.

In a regular update, assume that $sk_{\mathbf{t}}$ is well-formed. If $|\mathbf{t}| = \ell - 1$, then the update procedure sets $sk_{\mathbf{t}+1} \leftarrow sk_{\mathbf{t}} \setminus \{\widetilde{sk}_{\mathbf{t}}\}$, which by the second property of $\Gamma_{\mathbf{t}}$ and the induction hypothesis means that $sk_{\mathbf{t}+1}$ is also well-formed. If $|\mathbf{t}| < \ell - 1$, the update procedure adds keys $\widetilde{sk}_{\mathbf{t}\|1}$ and $\widetilde{sk}_{\mathbf{t}\|2}$ and removes $\widetilde{sk}_{\mathbf{t}}$ from $sk_{\mathbf{t}}$, which by the second property of $\Gamma_{\mathbf{t}}$ indeed corresponds to $\{\mathbf{w} : \mathbf{w} \in \Gamma_{\mathbf{t}+1}\}$. Moreover, $\widetilde{sk}_{\mathbf{t}\|1}$ is derived from $\widetilde{sk}_{\mathbf{t}} = \widetilde{sk}_{\mathbf{t}\|1} \leftarrow (c, d, e_{|\mathbf{t}|+1}, \ldots, e_\ell)$ as $\widetilde{sk}_{\mathbf{t}\|1} \leftarrow (c, d \cdot e_{|\mathbf{t}|+1}, e_{|\mathbf{t}|+2}, \ldots, e_\ell)$, which satisfies Equation (1) with randomness r that is independent from all other keys in $sk_{\mathbf{t}+1}$ because $\widetilde{sk}_{\mathbf{t}} \notin sk_{\mathbf{t}+1}$. Similarly, $\widetilde{sk}_{\mathbf{t}\|2}$ satisfies Equation (1) because it is generated as

$$\begin{aligned}
c' &= c \cdot g_2^{r'} = g_2^{r+r'} \\
d' &= d \cdot e_{k+1} \cdot (h_0 \prod_{j=1}^{k} h_j^{t_j} \cdot h_{k+1}^{w_{k+1}})^{r'} \\
&= h^x (h_0 \prod_{j=1}^{k} h_j^{t_j} \cdot h_{k+1}^2)^{r+r'} \\
e'_{k+2} &= e_{k+2} \cdot h_{k+2}^{r'} = h_{k+2}^{r+r'} \\
&\vdots \\
e'_\ell &= e_\ell \cdot h_\ell^{r'} = h_\ell^{r+r'}
\end{aligned}$$

satisfying Equation (1) with randomness $r + r'$, which is independent of the randomness of other keys in $sk_{\mathbf{t}+1}$ due to the uniform choice of r'.

For the fast-forward update procedure, one can see that if $sk_{\mathbf{t}}$ is well-formed, then the updated key $sk_{\mathbf{t}'}$ for $\mathbf{t}' > \mathbf{t}$ is well-formed as well. Indeed, by adding the keys for nodes in $\Gamma_{\mathbf{t}'} \setminus \Gamma_{\mathbf{t}}$ and removing those for $\Gamma_{\mathbf{t}} \setminus \Gamma_{\mathbf{t}'}$, we have that $sk_{\mathbf{t}'}$ contains keys $\widetilde{sk}_{\mathbf{w}}$ for all $\mathbf{w} \in \Gamma_{\mathbf{t}'}$. The randomness independence is guaranteed by the random choice of r' in Equation (2). In the optimized variant, all keys still have independent randomness because one key $\widetilde{sk}_{\mathbf{w}'} \in sk_{\mathbf{t}'}$ will have the same randomness r as some key $\widetilde{sk}_{\mathbf{w}} \in sk_{\mathbf{t}}$ where $\mathbf{w}$ is a prefix of $\mathbf{w}'$. That randomness is independent from all other keys in $sk_{\mathbf{t}'}$, however, because the key $\widetilde{sk}_{\mathbf{w}}$ does not occur in $sk_{\mathbf{t}'}$. Indeed, by the definition of $\Gamma_{\mathbf{t}'}$, one can see that $\Gamma_{\mathbf{t}'}$ cannot have elements $\mathbf{w} \neq \mathbf{w}'$ with $\mathbf{w}$ a prefix of $\mathbf{w}'$.

To see why signature verification works, observe that a signature for time period $\mathbf{t}$ and message M is computed from a key $\widetilde{sk}_{\mathbf{t}} = (c, d, e_{|\mathbf{t}|+1}, \ldots, e_\ell)$ in a well-formed key $sk_{\mathbf{t}}$. The left-hand side of

the verification equation is therefore

$$\begin{aligned}
\mathrm{e}(\sigma_1, g_2) &= \mathrm{e}\Big(d \cdot e_\ell^{\mathsf{H}_q(M)} \cdot \big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}|} h_j^{t_j} \cdot h_\ell^{\mathsf{H}_q(M)}\big)^{r'} \ , \ g_2\Big) \\
&= \mathrm{e}\Big(h^x \big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}|} h_j^{t_j} \cdot h_\ell^{\mathsf{H}_q(M)}\big)^{r+r'} \ , \ g_2\Big) \\
&= \mathrm{e}(h^x, g_2) \cdot \mathrm{e}\Big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}|} h_j^{t_j} \cdot h_\ell^{\mathsf{H}_q(M)} \ , \ g_2\Big)^{r+r'} \\
&= \mathrm{e}(h, y) \cdot \mathrm{e}\Big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}|} h_j^{t_j} \cdot h_\ell^{\mathsf{H}_q(M)} \ , \ \sigma_2\Big) \ .
\end{aligned}$$

3.5 Security

Theorem 3.1. *For any* fu-cma *adversary $\mathcal{A}$ against the above forward-secure signature scheme in the random-oracle model for $T = 2^\ell - 1$ time periods, there exists an adversary $\mathcal{B}$ with essentially the same running time and advantage in solving the ℓ-wBDHI$^*_3$ problem*

$$\mathbf{Adv}^{\ell\text{-wBDHI}^*_3}_{\mathbb{G}_1 \times \mathbb{G}_2}(\mathcal{B}) \ \geq \ \frac{1}{T \cdot q_{\mathsf{H}}} \cdot \mathbf{Adv}^{\text{fu-cma}}_{\mathcal{FS}}(\mathcal{A}) - \frac{q_{\mathsf{H}}^2}{2^\kappa} \ ,$$

where q_{H} is the number of random-oracle queries made by $\mathcal{A}$.

Proof. We prove the theorem in two steps. First, we show that the scheme is selectively secure when the message space $\mathcal{M} = \{0,1\}^\kappa$ and H_q is the identity function, meaning, interpreting a κ-bit string as an integer in $\mathbb{Z}_q$.

Step 1: sfu-cma. We show that the above scheme with message space $\mathcal{M} = \{0,1\}^\kappa$ and H_q the identity function is sfu-cma-secure under the ℓ-wBDHI$^*_3$ assumption by describing an algorithm $\mathcal{B}$ that, given a successful sfu-cma forger $\mathcal{A}'$, solves the ℓ-wBDHI$^*_3$ problem. On input $(A_1 = g_1^\alpha, A_2 = g_1^{(\alpha^2)}, \ldots, A_\ell = g_1^{(\alpha^\ell)}, B_1 = g_2^\alpha, \ldots, \mathcal{B}_\ell = g_2^{(\alpha^\ell)}, C)$, algorithm $\mathcal{B}$ proceeds as follows.

It first runs $\mathcal{A}$ to obtain $(\bar{\mathbf{t}}, \mathbf{t}^*, M^*)$. That is, $\mathcal{A}$ receives $sk_{\bar{\mathbf{t}}}$ and produces a forgery on $\mathbf{t}^*, M^*$. Let $\mathbf{w}^* \in \{0,1,2\}^{\ell-1}$ such that $\mathbf{w}^* = w_1^* \| \ldots \| w_{\ell-1}^* = \mathbf{t}^* \| 0^{\ell-1-|\mathbf{t}^*|}$. It then sets the public key and public parameters as

$$\begin{aligned}
y &\leftarrow B_1 \\
h &\leftarrow g_1^\gamma \cdot A_\ell \\
h_0 &\leftarrow g_1^{\gamma_0} \cdot \prod_{i=1}^{\ell-1} A_{\ell-i+1}^{-w_i^*} \cdot A_1^{-M^*} \\
h_i &\leftarrow g_1^{\gamma_i} \cdot A_{\ell-i+1} \qquad\qquad \text{for } i = 1, \ldots, \ell \ ,
\end{aligned}$$

where $\gamma, \gamma_0, \ldots, \gamma_\ell \xleftarrow{\$} \mathbb{Z}_q$.

By setting the parameters as such, $\mathcal{B}$ implicitly sets $x = \alpha$ and $h^x = A_1^\gamma \cdot g_1^{(\alpha^{\ell+1})}$. The reduction allows us to achieve two goals:

- extract the value of h^x from a forgery on $\mathbf{t}^*, M^*$ (provided by $\mathcal{A}'$), allowing $\mathcal{B}$ to easily compute its ℓ-wBDHI$^*_3$ solution $\mathrm{e}(g_1, C)^{(\alpha^{\ell+1})}$;
- simulate $\widetilde{sk}_{\mathbf{w}'}$ for all $\mathbf{w}' \in \{0,1,2\}^{\leq \ell-1}$ which are not a prefix of $\mathbf{w}^*$; this would be useful for simulating both the signing and the break-in oracle.

Algorithm $\mathcal{B}$ responds to $\mathcal{A}'$'s oracle queries as follows.

Key update. There is no need for $\mathcal{B}$ to simulate anything beyond keeping track of the current time period $\mathbf{t}$.

Signing. We first describe how to answer a signing query for a message M in time period $\mathbf{t} \neq \mathbf{t}^*$, and then describe the case that $\mathbf{t} = \mathbf{t}^*$ and $M \neq M^*$. Let $\mathbf{w} \in \{0,1,2\}^{\ell-1}$ be such that $\mathbf{w} = \mathbf{t} \| 0^{\ell-1-|\mathbf{t}|}$.

Case 1: $\mathbf{t} \neq \mathbf{t}^*$. It is easy to see that

$$\mathbf{t} \neq \mathbf{t}^* \Rightarrow \mathbf{w} \neq \mathbf{w}^* .$$

(This crucially uses the fact that $\mathbf{t}, \mathbf{t}^* \in \{1,2\}^*$.) Then, let $\mathbf{w}' = w_1 \| \cdots \| w_k$ denote the shortest prefix of $\mathbf{w}$ which is not a prefix of $\mathbf{w}^*$. Extending the notation of $\widetilde{sk}_{\mathbf{w}'}$ to $\mathbf{w}' \in \{0,1,2\}^{\leq \ell-1}$, we describe how $\mathcal{B}$ can derive a valid key $\widetilde{sk}_{\mathbf{w}'}$, from which it is straight-forward to derive both $\widetilde{sk}_{\mathbf{w}}$ and a signature for $\mathbf{t}, M$. Recall that $\widetilde{sk}_{\mathbf{w}'}$ has the structure

$$\begin{aligned}
&(c, d, e_{k+1}, \ldots, e_\ell) \ = \\
&\Big(g_2^r \ , \ h^x \big(h_0 \prod_{i=1}^{k} h_i^{w_i}\big)^r \ , \ h_{k+1}^r \ , \ \ldots \ , \ h_\ell^r\Big)
\end{aligned}$$

for a uniformly distributed value of r. Focusing on the second component d first, we have that

$$\begin{aligned}
d &= h^x \cdot \left(h_0 \cdot \prod_{i=1}^{k} h_i^{w_i}\right)^r \\
&= \left(g_1^\gamma A_\ell\right)^\alpha \cdot \left(\left(g_1^{\gamma_0} \prod_{i=1}^{\ell-1} A_{\ell-i+1}^{-w_i^*} A_1^{-M^*}\right) \cdot \prod_{i=1}^{k} \left(g_1^{\gamma_i} A_{\ell-i+1}\right)^{w_i}\right)^r \\
&= A_1^\gamma g_1^{(\alpha^{\ell+1})} \cdot \left(g_1^{\gamma_0 + \sum_{i=1}^k \gamma_i w_i} A_{\ell-k+1}^{w_k - w_k^*} \cdot \prod_{i=k+1}^{\ell-1} A_{\ell-i+1}^{-w_i^*} A_1^{-M^*}\right)^r \ ,
\end{aligned}$$

where the third equality holds because $w_i = w_i^*$ for $1 \leq i < k$ and $w_k \neq w_k^*$. (Note that in the product notation $\prod_{i=k+1}^{\ell-1}$ above, we let the result of the product simply be the unity element if $k+1 > \ell-1$.) Let us denote the four factors between parentheses in the last equation as F_1, F_2, F_3, and F_4, and denote their product as F. If we let

$$r \leftarrow r' + \frac{\alpha^k}{w_k^* - w_k} \bmod q$$

for a random $r' \xleftarrow{\$} \mathbb{Z}_q$, then we have that

$$d \ = \ A_1^\gamma \cdot g_1^{(\alpha^{\ell+1})} \cdot F^{r'} \cdot F^{\frac{\alpha^k}{w_k^* - w_k}} \ .$$

The first and third factors in this product are easy to compute. The second factor would allow $\mathcal{B}$ to compute the solution its ℓ-wBDHI$^*_3$ problem as $\mathrm{e}(g_1^{(\alpha^{\ell+1})}, C)$, so $\mathcal{B}$ cannot simply compute it. The last factor $F^{\frac{\alpha^k}{w_k^* - w_k}}$ can be written as the product

of

$$
\begin{aligned}
F_1^{\frac{\alpha^k}{w_k^*-w_k}} &= A_k^{\frac{\gamma_0+\sum_{i=1}^{k}\gamma_i w_i}{w_k^*-w_k}} \\
F_2^{\frac{\alpha^k}{w_k^*-w_k}} &= A_{\ell-k+1}^{-\alpha^k} = g_1^{-(\alpha^{\ell+1})} \\
F_3^{\frac{\alpha^k}{w_k^*-w_k}} &= \prod_{i=k+1}^{\ell-1} A_{\ell+k-i+1}^{\frac{-w_i^*}{w_k^*-w_k}} = \prod_{i=0}^{\ell-k-2} A_{\ell-i}^{\frac{-w_{k+2+i}^*}{w_k^*-w_k}} \\
F_4^{\frac{\alpha^k}{w_k^*-w_k}} &= A_{k+1}^{\frac{-M^*}{w_k^*-w_k}} .
\end{aligned}
$$

Because $1 \le k \le \ell-1$, it is clear that all but the second of these can be computed from $\mathcal{B}$'s inputs, and that the second cancels out with the factor $g_1^{(\alpha^{\ell+1})}$ in d, so that it can indeed compute d this way. The other components of the key are also efficiently computable as

$$
\begin{aligned}
c &= g_2^{r'} \cdot B_k^{\frac{1}{w_k^*-w_k}} \\
e_i &= h_i^{r'} \cdot A_{\ell+k-i+1} \quad \text{for } i = k+1, \ldots, \ell \\
&= h_{k+i}^{r'} \cdot A_{\ell-i} \quad \text{for } i = 0, \ldots, \ell-k-1 .
\end{aligned}
$$

From this key $(c, d, e_{k+1}, \ldots, e_\ell)$ for $\mathbf{w}'$, $\mathcal{B}$ can derive a key for $\mathbf{w}$ and compute a signature as in the real signing algorithm.

Case 2: $\mathbf{t} = \mathbf{t}^*, M \neq M^*$. For a signing query with $\mathbf{t} = \mathbf{t}^*$ but $M \neq M^*$, $\mathcal{B}$ proceeds in a similar way, but derives the signature (σ_1, σ_2) directly. Algorithm $\mathcal{B}$ can generate a valid signature using a similar approach as above, but using the fact that $M \neq M^*$ instead of $w_k \neq w_k^*$. Namely, letting $\mathbf{w} = \mathbf{t} \| 0^{\ell-1-|\mathbf{t}|}$, $\mathcal{B}$ computes a signature

$$
\begin{aligned}
\sigma_1 &= h^x \cdot \left(h_0 \cdot \prod_{i=1}^{\ell-1} h_i^{w_i} \cdot h_\ell^M \right)^r \\
&= \left(g_1^\gamma A_\ell \right)^\alpha \cdot \left(\left(g_1^{\gamma_0} \cdot \prod_{i=1}^{\ell-1} A_{\ell-i+1}^{-w_i^*} \cdot A_1^{-M^*} \right) \cdot \right. \\
&\quad \left. \prod_{i=1}^{\ell-1} \left(g_1^{\gamma_i} \cdot A_{\ell-i+1} \right)^{w_i} \cdot (g_1^{\gamma_\ell} \cdot A_1)^M \right)^r \\
&= A_1^\gamma \cdot g_1^{(\alpha^{\ell+1})} \cdot \left(g_1^{\gamma_0 + \sum_{i=1}^{\ell-1} \gamma_i w_i + \gamma_\ell M} \cdot A_1^{M-M^*} \right)^r \\
\sigma_2 &= g_2^r
\end{aligned}
$$

by setting

$$
r \leftarrow r' + \frac{\alpha^\ell}{M^* - M} \bmod q
$$

for $r' \xleftarrow{\$} \mathbb{Z}_q$, so that $\mathcal{B}$ can compute (σ_1, σ_2) from its inputs $A_1, \ldots, A_\ell, B_1, \ldots, B_\ell$ similarly to the case that $\mathbf{t} \neq \mathbf{t}^*$.

Break in. Here, $\mathcal{B}$ needs to simulate $sk_{\bar{\mathbf{t}}}$ where $\mathbf{t}^* \prec \bar{\mathbf{t}}$. This in turn requires simulating $\widetilde{sk}_{\mathbf{w}}$ for all $\mathbf{w} \in \Gamma_{\bar{\mathbf{t}}}$. By the first property of $\Gamma_{\bar{\mathbf{t}}}$ (described in Section 3.2), all of these $\mathbf{w}$ are not prefixes of $\mathbf{t}^*$ and also not prefixes of $\mathbf{w}^*$, and we can therefore simulate $\widetilde{sk}_{\mathbf{w}}$ exactly as before.

Forgery. When $\mathcal{A}'$ outputs a forgery (σ_1^*, σ_2^*) that satisfies the verification equation

$$
e(\sigma_1^*, g_2) = e(h, y) \cdot e\Big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}^*|} h_j^{t_j^*} \cdot h_\ell^{M^*} ,\ \sigma_2^*\Big) ,
$$

then there exists an $r \in \mathbb{Z}_q$ such that

$$
\begin{aligned}
\sigma_1^* &= h^\alpha \cdot \Big(h_0 \cdot \prod_{i=1}^{|\mathbf{t}^*|} h_i^{t_i^*} \cdot h_\ell^{M^*}\Big)^r \\
\sigma_2^* &= g_2^r .
\end{aligned}
$$

From the way that $\mathcal{B}$ chose the parameters $h, h_0, \ldots, h_\ell$, one can see that

$$
\sigma_1^* = A_1^\gamma \cdot g_1^{(\alpha^{\ell+1})} \cdot (g_1^r)^{\gamma_0 + \sum_{i=1}^{|\mathbf{t}^*|} \gamma_i t_i^* + \gamma_\ell M^*}
$$

Note that we do not know g_1^r, so we cannot directly extract $g_1^{(\alpha^{\ell+1})}$ from σ_1^*. Instead, observe that we have

$$
\begin{aligned}
e(\sigma_1^*, C_2) &= e(A_1^\gamma, C_2) \cdot e(g_1^{(\alpha^{\ell+1})}, C_2) \\
&\quad \cdot e(C_1, \sigma_2^*)^{\gamma_0 + \sum_{i=1}^{|\mathbf{t}^*|} \gamma_i t_i^* + \gamma_\ell M^*} ,
\end{aligned}
$$

from which $\mathcal{B}$ can easily compute its output $e(g_1^{(\alpha^{\ell+1})}, C_2) = e(g_1, g_2)^{(\gamma \cdot \alpha^{\ell+1})}$. It does so whenever $\mathcal{A}'$ is successful, so that

$$
\mathbf{Adv}^{\ell\text{-wBDHI}^*_3}_{\mathbb{G}_1 \times \mathbb{G}_2}(\mathcal{B}) \ \ge\ \mathbf{Adv}^{\text{sfu-cma}}_{\mathcal{FS}}(\mathcal{A}') .
$$

Step 2: fu-cma. Full fu-cma security for $\mathcal{M} = \{0,1\}^*$ and with $\mathsf{H}_q : \mathcal{M} \to \{0,1\}^\kappa$ modeled as a random oracle then follows because, given an fu-cma adversary $\mathcal{A}$ in the random-oracle model, one can build a sfu-cma adversary $\mathcal{A}'$ that guesses the time period t^* and the index of $\mathcal{A}$'s random-oracle query for $\mathsf{H}_q(M^*)$, and sets $\bar{t} \leftarrow t^* + 1$. If $\mathcal{A}'$ correctly guesses t^*, then it can use $sk_{\bar{t}}$ to simulate $\mathcal{A}$'s signature, key update, and break-in queries after time $\bar{t}$ until $\mathcal{A}$'s choice of break-in time $\bar{t}'$, at which point it can hand over $sk_{\bar{t}'}$.

If $\mathcal{A}'$ moreover correctly guessed the index of $\mathsf{H}_q(M^*)$, and if $\mathcal{A}$ never made colliding queries $\mathsf{H}_q(M) = \mathsf{H}_q(M')$ for $M \neq M'$, then $\mathcal{A}$'s forgery is also a valid forgery for $\mathcal{A}'$. Note that for $\mathcal{A}$ to be successful, it must hold that $\bar{t}' > t^*$, so it must hold that $\bar{t}' \ge \bar{t}$. The advantage of $\mathcal{A}'$ is given by

$$
\mathbf{Adv}^{\text{sfu-cma}}_{\mathcal{FS}}(\mathcal{A}') \ \ge\ \frac{1}{T \cdot q_{\mathrm{H}}} \cdot \mathbf{Adv}^{\text{fu-cma}}_{\mathcal{FS}}(\mathcal{A}) - \frac{q_{\mathrm{H}}^2}{2^\kappa} , \qquad (3)
$$

where q_{H} is an upper bound on $\mathcal{A}$'s number of random-oracle queries. Together with Equation (3), we obtain the inequality of the theorem statement. □

4 FORWARD-SECURE MULTI-SIGNATURES

To obtain a multi-signature scheme, we observe that the component-wise product $(\Sigma_1, \Sigma_2) = (\prod_{i=1}^n \sigma_{i,1}, \prod_{i=1}^n \sigma_{i,2})$ of a number of signatures $(\sigma_{1,1}, \sigma_{1,2}), \ldots, (\sigma_{n,1}, \sigma_{n,2})$ satisfies the verification equation with respect of the product of public keys $Y = y_1 \cdot \ldots \cdot y_n$. This method of combining signatures is vulnerable to a rogue-key attack, however, where a malicious signer chooses his public key based on that of an honest signer, so that the malicious signer can compute valid signatures for their aggregated public key. The scheme below

borrows a technique due to Ristenpart and Yilek [39] using proofs of possession to prevent against these types of attack.

4.1 Definitions

In addition to the algorithms of a forward-secure signature scheme in Section 3.1, a forward-secure multi-signature scheme $\mathcal{FMS}$ in the key verification model has a key genreation that additionally outputs a proof π for the public key:

Key generation: $(pk, \pi, sk_1) \xleftarrow{\$} \mathsf{Kg}$. The key generation algorithm generates a public verification key pk, a proof π, and an initial secret signing key sk_1 for the first time period.

and additionally has the following algorithms:

Key verification: $b \leftarrow \mathsf{KVf}(pk, \pi)$. The key verification algorithm returns 1 if the proof pk is valid for pk and returns 0 otherwise.

Key aggregation: $apk \xleftarrow{\$} \mathsf{KAgg}(pk_1, \ldots, pk_n)$. On input a list of individual public keys $(pk_1, \ldots, pk_n)$, the key aggregation returns an aggregate public key apk, or $\perp$ to indicate that key aggregation failed.

Signature aggregation. $\Sigma \xleftarrow{\$} \mathsf{SAgg}((pk_1, \sigma_1), \ldots, (pk_n, \sigma_n), t, M)$. Anyone can aggregate a given list of individual signatures $(\sigma_1, \ldots, \sigma_n)$ by different signers with public keys $(pk_1, \ldots, pk_n)$ on the same message M and for the same period t into a single multi-signature Σ.

Aggregate verification. $b \leftarrow \mathsf{AVf}(apk, t, M, \Sigma)$. Given an aggregate public key apk, a message M, a time period t, and an aggregate signature Σ, the verification algorithm returns 1 to indicate that all signers in apk signed M in period t, or 0 to indicate that verification failed.

Correctness requires that $\mathsf{KVf}(pk, \pi) = 1$ with probability one if $(pk, \pi, sk_1) \xleftarrow{\$} \mathsf{Kg}$ and that for all messages $M \in \mathcal{M}$, for all $n \in \mathbb{Z}$, and for all time periods $t \in \{0, \ldots, T-1\}$, it holds that $\mathsf{AVf}(apk, t, M, \Sigma) = 1$ with probability one if $(pk_i, \pi_i, sk_{i,1}) \xleftarrow{\$} \mathsf{Kg}$, $apk \xleftarrow{\$} \mathsf{KAgg}(pk_1, \ldots, pk_n)$, $sk_{i,j} \xleftarrow{\$} \mathsf{Upd}(sk_{i,j-1})$ for $i = 1, \ldots, n$ and $j = 2, \ldots, t$, $\sigma_i \xleftarrow{\$} \mathsf{Sign}(sk_{i,t}, M)$ for $i = 1, \ldots, n$, and $\Sigma \xleftarrow{\$} \mathsf{SAgg}((pk_1, \sigma_1), \ldots, (pk_n, \sigma_n), t, M)$.

Unforgeability (fu-cma) is defined through a game that is similar to that described in Section 3.1. The adversary is given the public key pk and proof π of an honest signer and access to the same key update, signing, and break-in oracles. However, at the end of the game, the adversary's forgery consists of a list of public keys and proofs $(pk_1^*, \pi_1^*, \ldots, pk_n^*, \pi_n^*)$, a message M^*, a time period t^*, and a multi-signature Σ^*. The forgery is considered valid if

- $pk \in \{pk_1^*, \ldots, pk_n^*\}$,
- the proofs $\pi_1^*, \ldots, \pi_n^*$ are valid for public keys $pk_1^*, \ldots, pk_n^*$ according to KVf,
- Σ^* is valid with respect to the aggregate public key apk^* of $(pk_1^*, \ldots, pk_n^*)$, message M^*, and time period t^*,
- $\bar{t} > t^*$,
- and $\mathcal{A}$ never made a signing query for M^* during time period t^*.

4.2 Construction

Let $\mathsf{H}_{\mathbb{G}_1} : \{0,1\}^* \to \mathbb{G}_1^*$ be a hash function. The multi-signature scheme reuses the key update and signature algorithms from the scheme from Section 3.3, but uses different key generation and verification algorithms, and adds signature and key aggregation.

Key generation. Each signer chooses $x \xleftarrow{\$} \mathbb{Z}_q$ and computes $y \leftarrow g_2^x$ and $y' \leftarrow \mathsf{H}_{\mathbb{G}_1}(\mathsf{PoP}, y)$, where PoP is a fixed string used as a prefix for domain separation. It sets its public key to $pk = y$, the proof to $\pi = y'$, and computes its initial secret key as $sk_1 \leftarrow h^x$.

Key verification. Given a public key $pk = y$ with proof $\pi = y'$, the key verification algorithm validates the proof of possession by returning 1 if

$$\mathrm{e}(y', g_2) \;=\; \mathrm{e}(\mathsf{H}_{\mathbb{G}_1}(\mathsf{PoP}, y), y)$$

and returning 0 otherwise.

Key aggregation. Given public keys $pk_1 = y_1, \ldots, pk_n = y_n$, the key aggregation algorithm computes $Y \leftarrow \prod_{i=1}^{n} y_i$ and returns the aggregate public key $apk = Y$.

Signature aggregation. Given signatures $\sigma_1 = (\sigma_{1,1}, \sigma_{1,2}), \ldots, \sigma_n = (\sigma_{n,1}, \sigma_{n,2}) \in \mathbb{G}_1 \times \mathbb{G}_2$ on the same message M, the signature aggregation algorithm outputs

$$\Sigma \;=\; (\Sigma_1, \Sigma_2) \;=\; \Big(\prod_{i=1}^{n} \sigma_{i,1}\,,\; \prod_{i=1}^{n} \sigma_{i,2}\Big)\,.$$

Aggregate verification. Aggregate signatures are verified with respect to aggregate public keys in exactly the same way as individual signatures with respect to individual public keys. Namely, given an aggregate signature $(\Sigma_1, \Sigma_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ on message M under aggregate public key $apk = Y$ in time period $\mathbf{t}$, the verifier accepts if and only if $apk \neq \perp$ and

$$\mathrm{e}(\Sigma_1, g_2) \;=\; \mathrm{e}(h, Y) \cdot \mathrm{e}\Big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}|} h_j^{t_j} \cdot h_{\ell+1}^{\mathsf{H}_q(M)}\,,\; \Sigma_2\Big)\,.$$

4.3 Security

Theorem 4.1. *For any* fu-cma *adversary $\mathcal{A}$ against the above forward-secure multi-signature scheme for $T = 2^\ell - 1$ time periods in the random-oracle model, there exists an adversary $\mathcal{B}$ with essentially the same running time that solves the ℓ-wBDHI$_3^*$ problem with advantage*

$$\mathbf{Adv}^{\ell\text{-wBDHI}_3^*}_{\mathbb{G}_1 \times \mathbb{G}_2}(\mathcal{B}) \;\geq\; \frac{1}{T \cdot q_{\mathrm{H}}} \cdot \mathbf{Adv}^{\text{fu-cma}}_{\mathcal{FMS}}(\mathcal{A}) - \frac{q_{\mathrm{H}}^2}{2^\kappa}\,,$$

where q_{H} is the number of random-oracle queries made by $\mathcal{A}$.

Proof. We prove the theorem by showing that a forger $\mathcal{A}$ for the multi-signature scheme yields a forger $\mathcal{A}'$ for the single-signer scheme of Section 3.3 such that

$$\mathbf{Adv}^{\text{fu-cma}}_{\mathcal{FS}}(\mathcal{A}') \;\geq\; \mathbf{Adv}^{\text{fu-cma}}_{\mathcal{FS}}(\mathcal{A})\,.$$

The theorem then follows from Theorem 3.1.

The key idea following [39] is to program $\mathsf{H}_{\mathbb{G}_1}$ in such a way that we can "extract" a valid forgery for the single-signer scheme starting from that for the multi-signature scheme. In particular,

- given a rogue public key $pk_i^* = y_i$ with proof $\pi_i^* = y_i'$ where $y_i = g_2^{x_i}$, we can extract the corresponding secret key h^{x_i} from y_i' by programming $\mathsf{H}_{\mathbb{G}_1}(\mathsf{PoP}, y_i) = h^{r_i}$.

- given h^{x_i} for all $y_i \neq y$ along with a valid forgery for the multi-signature scheme, we can extract a forgery for the single-signer scheme.

Step 1: simulating $\mathcal{A}$'s view. On input the parameters $(T, h, h_0, \ldots, h_\ell)$ and a public key y for the single-signer scheme, the single-signer forger $\mathcal{A}'$ chooses $r \xleftarrow{\$} \mathbb{Z}_q^*$ and stores $(y, \perp, g_1^r)$ in a list L. It computes $y' \leftarrow y^r$ and runs $\mathcal{A}$ on the same common parameters and target public key $pk = y$ and proof $\pi = y'$. Observe that π is indeed a valid proof for pk since $e(y', g_2) = e(\mathsf{H}_{\mathbb{G}_1}(\mathsf{PoP}, y), y)$.

Algorithm $\mathcal{A}'$ answers all of $\mathcal{A}$'s key update, signing, and break-in oracle queries, as well as random-oracle queries for H_q, by simply relaying queries and responses to and from $\mathcal{A}'$'s own oracles. Queries to the random oracle for $\mathsf{H}_{\mathbb{G}_1}$ are answered as follows.

Random oracle $\mathsf{H}_{\mathbb{G}_1}$**.** On input (PoP, z), $\mathcal{A}'$ checks whether there already exists a tuple $(z, \cdot, v) \in L$. If so, it returns v. If not, it chooses $r \xleftarrow{\$} \mathbb{Z}_q^*$, computes $v \leftarrow h^r$, adds a tuple (z, r, v) to L and returns v.y

Step 2: extracting a forgery. When $\mathcal{A}$ outputs its forgery

$$(pk_1^*, \pi_1^*, \ldots, pk_n^*, \pi_n^*), M^*, \mathbf{t}^*, \Sigma^*,$$

algorithm $\mathcal{A}'$ first verifies the proofs $\pi_1^*, \ldots, \pi_n^*$ for public keys $pk_1^*, \ldots, pk_n^*$ and computes the aggregate public key apk^*, creating additional entries in L if necessary. Let $pk_i^* = y_i = g_2^{x_i}$ and $\pi_i^* = y_i'$. Looking ahead, if pk_i^* passes key verification, then we have $y_i' = (h^{x_i})^{r_i}$ and since we know r_i, we will be able to "extract" $h^{x_i} \in \mathbb{G}_1$.

If all keys are valid, then it holds that $y_i' = \mathsf{H}_{\mathbb{G}_1}(\mathsf{PoP}, y_i)^{x_i}$ for all $i = 1, \ldots, n$. Let $apk^* = Y$ be the aggregate public key. From the aggregate verification equation

$$\mathrm{e}(\Sigma_1^*, g_2) \;=\; \mathrm{e}(h, Y) \cdot \mathrm{e}\big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}^*|} h_j^{t_j^*} \cdot h_\ell^{\mathsf{H}_q(M^*)},\ \Sigma_2^*\big)$$

and the fact that $Y = \prod_{i=1}^n y_i = y \cdot g_2^{\sum_{i=1, y_i \neq y}^n x_i}$, we have that

$$\begin{aligned} \mathrm{e}(\Sigma_1^*, g_2) \;&=\; \mathrm{e}(h, y) \cdot \mathrm{e}(h, g_2)^{\sum_{i=1, y_i \neq y}^n x_i} \cdot \\ &\qquad \mathrm{e}\big(h_0 \cdot \prod_{j=1}^{\ell} h_j^{t_j^*} \cdot h_{\ell+1}^{\mathsf{H}_q(M^*)},\ \Sigma_2^*\big) \\ \Leftrightarrow\ \mathrm{e}(\Sigma_1^* \cdot h^{-\sum_{i=1, y_i \neq y}^n x_i}, g_2) \;&=\; \mathrm{e}(h, y) \cdot \\ &\qquad \mathrm{e}\big(h_0 \cdot \prod_{j=1}^{|\mathbf{t}^*|} h_j^{t_j^*} \cdot h_\ell^{\mathsf{H}_q(M^*)},\ \Sigma_2^*\big)\ . \end{aligned}$$

For all $y_i \neq y$, $\mathcal{A}'$ looks up the tuple (y_i, r_i, v_i) in L. We know that $v_i = h^{r_i}$, and hence that $y_i' = h^{r_i x_i}$. By comparing the last equation above to the verification equation of the single-signer scheme, and by observing that $y_i' = h^{r_i x_i}$, we know that the pair

$$\begin{aligned} \sigma_1^* \;&\leftarrow\; \Sigma_1^* \cdot \prod_{i=1, y_i \neq y}^{n} y_i'^{-1/r_i} \\ \sigma_2^* \;&\leftarrow\; \Sigma_2^* \end{aligned}$$

is a valid forgery for the single-signer scheme, so $\mathcal{A}'$ can output $M^*, \mathbf{t}^*, (\sigma_1^*, \sigma_2^*)$ as its forgery. □

5 EFFICIENCY ANALYSIS

In section we analyze the efficiency of the Pixel scheme by analyzing the computational work and the size of objects, and present estimated performance figures. We let $T = 2^\ell - 1$ denote the maximum number of time periods.

Computational Efficiency. The main operations are key generation, updating the key, signing, aggregating public keys, and verifying signatures.

- Key generation requires 1 exponentiation in each of $\mathbb{G}_1$ and $\mathbb{G}_2$.
- Key verification requires 2 pairings.
- Key update for an arbitrary number of time steps requires ℓ^2 exponentiations and $2\ell^2$ multiplications in $\mathbb{G}_2$ and ℓ exponentiations in $\mathbb{G}_1$; key updates can of course be entirely precomputed, if necessary. Key updates from $\mathbf{t}$ to $\mathbf{t}+1$ require $\ell - |\mathbf{t}|$ exponentiation in $\mathbb{G}_1$ and 1 exponentiation in $\mathbb{G}_2$ (ignoring multiplications) if $|\mathbf{t}| < \ell - 1$, and no group operations if $|\mathbf{t}| = \ell - 1$. On average, this only requires

  $$1/2 \cdot 0 + 1/4 \cdot 2 + 1/8 \cdot 3 + 1/16 \cdot 4 + \cdots \leq 1.5$$

  exponentiations in $\mathbb{G}_1$ and

  $$1/2 \cdot 0 + 1/4 \cdot 1 + 1/8 \cdot 1 + 1/16 \cdot 1 + \cdots \leq 0.5$$

  exponentiation in $\mathbb{G}_2$. That is, irrespective of the maximum number of time periods T, the average work for updating the key does not exceed 1.5 and 0.5 exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- Signing requires 3 exponentiations and 4ℓ multiplications in $\mathbb{G}_1$ and 1 exponentiation in $\mathbb{G}_2$. By precomputing

  $$\begin{aligned} \sigma_{1,1} \;&\leftarrow\; d \cdot \big(h_0 \cdot \prod_{j=1}^{\ell-1} h_j^{t_j}\big)^{r'} \\ \sigma_{1,2} \;&\leftarrow\; e_\ell \cdot h_\ell^{r'} \\ \sigma_2 \;&\leftarrow\; c \cdot g_2^{r'}\ , \end{aligned}$$

  the signature can be computed as $\sigma_1 \leftarrow \sigma_{1,1} \cdot \sigma_{1,2}^{\mathsf{H}_q(M)}$ once the message M is known, bringing the online computation down to a single exponentiation.
- Aggregating N public keys together costs $N - 1$ multiplications in $\mathbb{G}_2$. Here, we ignore the cost of verifying proofs of possession, which should only be performed once per public key.
- Verification of a signature requires 3 pairings (or one 3-multi-pairing) and ℓ multiplications and 1 exponentiation in $\mathbb{G}_1$, plus subgroup membership checks for $\mathbb{G}_1$ and $\mathbb{G}_2$.

Space Efficiency. We are mainly concerned with the size of the public parameters, public keys, secret keys, and signatures.

- The public parameters consist of $\ell + 2$ elements of $\mathbb{G}_1$.
- Every public key is a single element of $\mathbb{G}_2$.
- The size of sk_t is $\ell(\ell-1)/2$ elements in $\mathbb{G}_1$ and ℓ elements in $\mathbb{G}_2$.
- A signature consists of one element in $\mathbb{G}_1$ and one element in $\mathbb{G}_2$.

	keygen	key update	sign	key agg. ($N = 1000$)	key agg. ($N = 5000$)	verify	$\lvert pk\rvert$	$\lvert\sigma\rvert$	$\lvert sk_t\rvert$
$pk \in \mathbb{G}_2$, $T = 2^{20} - 1$	1.03 ms	0.75 ms	1.56 ms	3.04 ms	15.22 ms	4.66 ms	96 B	144 B	11 kB
$pk \in \mathbb{G}_2$, $T = 2^{30} - 1$	1.03 ms	0.75 ms	1.59 ms	3.04 ms	15.22 ms	4.68 ms	96 B	144 B	23 kB
$pk \in \mathbb{G}_1$, $T = 2^{20} - 1$	1.03 ms	1.31 ms	2.85 ms	0.83 ms	4.13 ms	4.19 ms	48 B	144 B	19 kB
$pk \in \mathbb{G}_1$, $T = 2^{30} - 1$	1.03 ms	1.31 ms	2.98 ms	0.83 ms	4.13 ms	4.25 ms	48 B	144 B	43 kB

Figure 5: Estimated performance figures of the Pixel **signature scheme algorithms, and the size of public keys, signatures, and secret keys when using a BLS12-381 curve. N denotes the amount of keys aggregated in key aggregation. Note that the estimated running time of key update procedure is based on the average cost of updating from time t to $t + 1$.**

Estimated Performance. Figure 5 shows estimated performance figures[2] and object sizes for the BLS12-381 curve [2]. Note that $\mathbb{G}_1$ and $\mathbb{G}_2$ are not equally efficient, $\mathbb{G}_1$ has a more efficient group operation and elements can be represented in half the space. We can choose whether we let place public keys in $\mathbb{G}_1$ or $\mathbb{G}_2$ represent the more efficient group. Setting up the scheme with $pk \in \mathbb{G}_2$ yields smaller secret keys and faster signing time, while choosing $pk \in \mathbb{G}_1$ results in smaller public keys and faster key aggregation.

6 VARIANTS AND EXTENSIONS

Deterministic signatures. In practice, it is helpful to implement deterministic signing and key updates in order to protect against attacks arising from bad randomness. We can achieve using the standard technique [5] of deriving randomness from a random oracle. More precisely, we assume a random oracle H' that maps to $\mathbb{Z}_q$, and when signing M at time t, we use $r \leftarrow \mathsf{H}'(\texttt{rand-sign}, \tilde{sk}_t, M, t)$. When updating the key from time t to $t+1$, we may need to compute $\widetilde{sk}_{\mathbf{t}\|1}$ and $\widetilde{sk}_{\mathbf{t}\|2}$ from $\widetilde{sk}_{\mathbf{t}}$. The required randomness r can be computed with $\mathsf{H}(\texttt{rand-update}, \widetilde{sk}_{\mathbf{t}})$. Alternatively, if we wish to avoid additional use of a random oracle, we can rely on prior "forward-secure PRG techniques" [30] as follows: during set-up, we sample a random seed s for a length-doubling pseudorandom generator whose outputs are $G_0(s), G_1(s)$. Whenever we need fresh randomness r for signing or key updates, we set $r = G_1(s)$ and replace s with $G_0(s)$.

Non-binary trees. One could try to reduce the key size by using b-ary trees instead of binary trees. A larger value of b reduces the depth of the tree, but increases the amount of key material that must be kept at each level of the tree. To support T time periods, one needs a b-ary tree of depth $\ell = \lceil \log_b T \rceil$. A node key at level d, however, can now take up to $b - 1$ keys of one element in $\mathbb{G}_2$ and $(\ell + d - 2)$ elements of $\mathbb{G}_1$.

The savings effect is quite limited, however, because the disadvantage of needing more keys per level quickly starts dominating the advantage of having less levels. For practical values of T, the maximum size of the secret key will usually be minimal for $b = 3$.

Parallel key timelines. In some applications, a signer may want to maintain several parallel timelines for different usages of a signing key. For example, in a sharded blockchain, the shards may be running in parallel at different speeds, without strict synchronization between the shards. If a time frame of the forward-secure signature scheme corresponds to the block height of a blockchain, for example, then the signer needs to maintain a different key schedule for the different shards.

A trivial approach is to run a separate instance of Pixel per timeline, and certify each public key with one root signing key. A more efficient approach for our particular scheme is to replace the fixed common parameter h with the output of a hash function $\mathsf{H}_{\mathbb{G}_1}(\texttt{scope}, \textit{scope})$. Meaning, during key generation, the signer generates $sk_{scope,1} \leftarrow \mathsf{H}_{\mathbb{G}_1}(\texttt{scope}, \textit{scope})^x$ for all relevant scopes *scope* and deletes the master key x. It can then update, sign, and aggregate signatures for each scope separately in the same way as before, but substituting $\mathsf{H}_{\mathbb{G}_1}(\texttt{scope}, \textit{scope})$ for h. Verification of individual signatures and of multi-signatures is also the same as before, substituting $\mathsf{H}_{\mathbb{G}_1}(\texttt{scope}, \textit{scope})$ for h.

Tighter security. The loss in tightness in Equation (3) of $T \cdot q_{\mathsf{H}}$ can be brought down to $T \cdot q_{\mathsf{S}}$ using Coron's technique [20], where q_{S} is the number of signing queries made by the adversary $\mathcal{A}$, by hashing the message into $\mathbb{G}_1$ instead of into $\mathbb{Z}_q$. Namely, a multi-signature would be a tuple $(\Sigma_1, \Sigma_2, \Sigma_3)$ satisfying

$$\mathrm{e}(\Sigma_1, g_2) \;=\; \mathrm{e}(h, Y) \cdot \mathrm{e}\big(h_0 \cdot \prod_{j=1}^{\ell} h_j^{t_j} \;,\; \Sigma_2\big) \cdot \mathrm{e}\big(\mathsf{H}_{\mathbb{G}_1}(\texttt{msg}, M), \Sigma_3\big) \;.$$

This scheme has the additional advantage of saving up to ℓ elements of $\mathbb{G}_1$ in secret key size, but signatures are one element of $\mathbb{G}_2$ longer than the base scheme. We leave details to the reader.

Avoiding proofs-of-possession. In situations where proofs-of-possession are not desirable, one could alternatively reuse techniques from [9, 34] to avoid rogue-key attacks. Signers' public keys are simply given by $pk_i = y_i = g_2^{x_i}$, but the aggregate public key is computed as $apk \leftarrow \prod_{i=1}^{n} pk_i^{\mathsf{H}_q(\{pk_1,\ldots,pk_n\},pk_i)}$. Individual signatures $(\sigma_{1,1}, \sigma_{1,2}), \ldots, (\sigma_{n,1}, \sigma_{n,2})$ are aggregated as

$$(\Sigma_1, \Sigma_2) \leftarrow \Big(\prod_{i=1}^{n} \sigma_{i,1}^{\mathsf{H}_q(\{pk_1,\ldots,pk_n\},pk_i)}, \prod_{i=1}^{n} \sigma_{i,2}^{\mathsf{H}_q(\{pk_1,\ldots,pk_n\},pk_i)}\Big) \,,$$

so that verification can be performed as usual.

Partial aggregation of multi-signatures. Further savings in terms of signature length can be obtained by partially aggregating multi-signatures. Multi-signatures $(\Sigma_{i,1}, \Sigma_{i,2})$ under aggregate public keys $apk_i = Y_i$ on messages M_i for time periods t_i for $i = 1, \ldots, n$, can be compressed into an aggregate multi-signature $(\Sigma_1, \Sigma_{1,2}, \ldots,$

[2] Computed by measuring the speed of the individual group operations (using the rust implementation of BLS12-381 used by zcash on an Intel i7 CPU) and summing the measured cost of all operations of each algorithm.

$\Sigma_{n,2}$) where $\Sigma_1 \leftarrow \prod_{i=1}^{n} \Sigma_{i,1}$, which can be verified by checking that

$$\mathrm{e}(\Sigma_1, g_2) \;=\; \mathrm{e}(h, \prod_{i=1}^{n} Y_i) \cdot \prod_{i=1}^{n} \mathrm{e}\big(h_0 \cdot \prod_{j=1}^{\ell} h_j^{t_{i,j}} \cdot h_{\ell+1}^{\mathrm{H}_q(M_i)} \;,\; \Sigma_{i,2}\big) \;.$$

Care must be taken, however, that either the messages M_i are all different, or that all aggregate public keys $apk_1, \ldots, apk_n$ are "trusted", in the sense that the verifier checks that they are composed of individual public keys with valid proofs of possession. One could enforce the messages M_i to be all different by including the aggregate public key in the message $M_i = apk_i \| M_i'$, but this has the disadvantage that the aggregate public key (and hence, the set of signers in the aggregate) must be known at the time of signing. Failure to follow these precautions makes the scheme insecure, because for a given aggregate public key apk_1 it is easy to come up with a "rogue" key $apk_2 = g_2^x / apk_1$ that allows an adversary to forge an aggregate signature on any message under apk_1 and apk_2.

REFERENCES

[1] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. 2008. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *ACM CCS 2008: 15th Conference on Computer and Communications Security*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM Press, Alexandria, Virginia, USA, 449–458. https://doi.org/10.1145/1455770.1455827

[2] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. 2003. Constructing Elliptic Curves with Prescribed Embedding Degrees. In *SCN 02: 3rd International Conference on Security in Communication Networks (Lecture Notes in Computer Science)*, Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano (Eds.), Vol. 2576. Springer, Heidelberg, Germany, Amalfi, Italy, 257–267. https://doi.org/10.1007/3-540-36413-7_19

[3] Mihir Bellare and Sara K. Miner. 1999. A Forward-Secure Digital Signature Scheme. In *Advances in Cryptology – CRYPTO'99 (Lecture Notes in Computer Science)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 431–448. https://doi.org/10.1007/3-540-48405-1_28

[4] Mihir Bellare and Gregory Neven. 2006. Multi-signatures in the plain public-Key model and a general forking lemma. In *ACM CCS 2006: 13th Conference on Computer and Communications Security*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM Press, Alexandria, Virginia, USA, 390–399. https://doi.org/10.1145/1180405.1180453

[5] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (Sept. 2012), 77–89. https://doi.org/10.1007/s13389-012-0027-1

[6] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography (Lecture Notes in Computer Science)*, Yvo Desmedt (Ed.), Vol. 2567. Springer, Heidelberg, Germany, Miami, FL, USA, 31–46. https://doi.org/10.1007/3-540-36288-6_3

[7] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. 2005. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In *Advances in Cryptology – EUROCRYPT 2005 (Lecture Notes in Computer Science)*, Ronald Cramer (Ed.), Vol. 3494. Springer, Heidelberg, Germany, Aarhus, Denmark, 440–456. https://doi.org/10.1007/11426639_26

[8] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. 2005. Hierarchical Identity Based Encryption with Constant Size Ciphertext. Cryptology ePrint Archive, Report 2005/015. (2005). http://eprint.iacr.org/2005/015.

[9] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *Advances in Cryptology – ASIACRYPT 2018, Part II (Lecture Notes in Computer Science)*, Thomas Peyrin and Steven Galbraith (Eds.), Vol. 11273. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia, 435–464. https://doi.org/10.1007/978-3-030-03329-3_15

[10] Dan Boneh, Ben Lynn, and Hovav Shacham. 2004. Short Signatures from the Weil Pairing. *Journal of Cryptology* 17, 4 (Sept. 2004), 297–319. https://doi.org/10.1007/s00145-004-0314-9

[11] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 104–121. https://doi.org/10.1109/SP.2015.14

[12] Xavier Boyen, Hovav Shacham, Emily Shen, and Brent Waters. 2006. Forward-secure signatures with untrusted update. In *ACM CCS 2006: 13th Conference on Computer and Communications Security*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM Press, Alexandria, Virginia, USA, 191–200. https://doi.org/10.1145/1180405.1180430

[13] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. 2010. Efficient Indifferentiable Hashing into Ordinary Elliptic Curves. In *Advances in Cryptology – CRYPTO 2010 (Lecture Notes in Computer Science)*, Tal Rabin (Ed.), Vol. 6223. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 237–254. https://doi.org/10.1007/978-3-642-14623-7_13

[14] Vitalik Buterin and Virgil Griffith. 2017. Casper the Friendly Finality Gadget. *CoRR* abs/1710.09437 (2017). arXiv:1710.09437 http://arxiv.org/abs/1710.09437

[15] Ran Canetti, Shai Halevi, and Jonathan Katz. 2003. A Forward-Secure Public-Key Encryption Scheme. In *Advances in Cryptology – EUROCRYPT 2003 (Lecture Notes in Computer Science)*, Eli Biham (Ed.), Vol. 2656. Springer, Heidelberg, Germany, Warsaw, Poland, 255–271. https://doi.org/10.1007/3-540-39200-9_16

[16] Ran Canetti, Shai Halevi, and Jonathan Katz. 2007. A Forward-Secure Public-Key Encryption Scheme. *Journal of Cryptology* 20, 3 (July 2007), 265–294. https://doi.org/10.1007/s00145-006-0442-5

[17] Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. 2018. ALGORAND AGREEMENT: Super Fast and Partition Resilient Byzantine Agreement. Cryptology ePrint Archive, Report 2018/377. (2018).

[18] Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu, and K. P. Chow. 2004. Secure Hierarchical Identity Based Signature and Its Application. In *ICICS 04: 6th International Conference on Information and Communication Security (Lecture Notes in Computer Science)*, Javier López, Sihan Qing, and Eiji Okamoto (Eds.), Vol. 3269. Springer, Heidelberg, Germany, Malaga, Spain, 480–494.

[19] M. Conti, E. Sandeep Kumar, C. Lal, and S. Ruj. 2018. A Survey on Security and Privacy Issues of Bitcoin. *IEEE Communications Surveys Tutorials* 20, 4 (Fourthquarter 2018), 3416–3452. https://doi.org/10.1109/COMST.2018.2842460

[20] Jean-Sébastien Coron. 2000. On the Exact Security of Full Domain Hash. In *Advances in Cryptology – CRYPTO 2000 (Lecture Notes in Computer Science)*, Mihir Bellare (Ed.), Vol. 1880. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 229–235. https://doi.org/10.1007/3-540-44598-6_14

[21] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *Advances in Cryptology – EUROCRYPT 2018, Part II (Lecture Notes in Computer Science)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.), Vol. 10821. Springer, Heidelberg, Germany, Tel Aviv, Israel, 66–98. https://doi.org/10.1007/978-3-319-78375-8_3

[22] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. 2018. Revisiting Proxy Re-encryption: Forward Secrecy, Improved Security, and Applications. In *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I (Lecture Notes in Computer Science)*, Michel Abdalla and Ricardo Dahab (Eds.), Vol. 10769. Springer, Heidelberg, Germany, Rio de Janeiro, Brazil, 219–250. https://doi.org/10.1007/978-3-319-76578-5_8

[23] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. 2019. On the Security of Two-Round Multi-Signatures. In *2019 IEEE Symposium on Security and Privacy, SP 2019*. IEEE Computer Society.

[24] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 51–68. https://doi.org/10.1145/3132747.3132757

[25] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 2017. 0-RTT Key Exchange with Full Forward Secrecy. In *Advances in Cryptology – EUROCRYPT 2017, Part III (Lecture Notes in Computer Science)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.), Vol. 10212. Springer, Heidelberg, Germany, Paris, France, 519–548. https://doi.org/10.1007/978-3-319-56617-7_18

[26] Timo Hanke, Mahnush Movahedi, and Dominic Williams. 2018. DFINITY Technology Overview Series, Consensus System. (2018). https://dfinity.org/pdf-viewer/library/dfinity-consensus.pdf

[27] K. Itakura and K. Nakamura. 1983. *A public-key cryptosystem suitable for digital multisignatures.* Technical Report. NEC Research and Development.

[28] Gene Itkis and Leonid Reyzin. 2001. Forward-Secure Signatures with Optimal Signing and Verifying. In *Advances in Cryptology – CRYPTO 2001 (Lecture Notes in Computer Science)*, Joe Kilian (Ed.), Vol. 2139. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 332–354. https://doi.org/10.1007/3-540-44647-8_20

[29] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Advances in Cryptology – CRYPTO 2017, Part I (Lecture Notes in Computer Science)*, Jonathan Katz and Hovav Shacham (Eds.), Vol. 10401. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[30] Hugo Krawczyk. 2000. Simple Forward-Secure Signatures From Any Signature Scheme. In *ACM CCS 2000: 7th Conference on Computer and Communications Security*, Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati (Eds.). ACM Press, Athens, Greece, 108–115. https://doi.org/10.1145/352600.352617

[31] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. 2006. Sequential Aggregate Signatures and Multisignatures Without Random Oracles.

In *Advances in Cryptology – EUROCRYPT 2006 (Lecture Notes in Computer Science)*, Serge Vaudenay (Ed.), Vol. 4004. Springer, Heidelberg, Germany, St. Petersburg, Russia, 465–485. https://doi.org/10.1007/11761679_28

[32] Changshe Ma, Jian Weng, Yingjiu Li, and Robert H. Deng. 2010. Efficient discrete logarithm based multi-signature scheme in the plain public key model. *Des. Codes Cryptography* 54, 2 (2010), 121–133.

[33] Tal Malkin, Daniele Micciancio, and Sara K. Miner. 2002. Efficient Generic Forward-Secure Signatures with an Unbounded Number Of Time Periods. In *Advances in Cryptology – EUROCRYPT 2002 (Lecture Notes in Computer Science)*, Lars R. Knudsen (Ed.), Vol. 2332. Springer, Heidelberg, Germany, Amsterdam, The Netherlands, 400–417. https://doi.org/10.1007/3-540-46035-7_27

[34] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. 2019. Simple Schnorr multi-signatures with applications to Bitcoin. *Des. Codes Cryptography* (2019).

[35] Silvio Micali. 2016. ALGORAND: The Efficient and Democratic Ledger. *CoRR* abs/1607.01341 (2016). arXiv:1607.01341 http://arxiv.org/abs/1607.01341

[36] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. 2001. Accountable-Subgroup Multisignatures: Extended Abstract. In *ACM CCS 2001: 8th Conference on Computer and Communications Security*, Michael K. Reiter and Pierangela Samarati (Eds.). ACM Press, Philadelphia, PA, USA, 245–254. https://doi.org/10.1145/501983.502017

[37] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf. (2008).

[38] Kazuo Ohta and Tatsuaki Okamoto. 1993. A Digital Multisignature Scheme Based on the Fiat-Shamir Scheme. In *Advances in Cryptology – ASIACRYPT'91 (Lecture Notes in Computer Science)*, Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto (Eds.), Vol. 739. Springer, Heidelberg, Germany, Fujiyoshida, Japan, 139–148. https://doi.org/10.1007/3-540-57332-1_11

[39] Thomas Ristenpart and Scott Yilek. 2007. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In *Advances in Cryptology – EUROCRYPT 2007 (Lecture Notes in Computer Science)*, Moni Naor (Ed.), Vol. 4515. Springer, Heidelberg, Germany, Barcelona, Spain, 228–245. https://doi.org/10.1007/978-3-540-72540-4_13

[40] Riad S. Wahby and Dan Boneh. 2019. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. Cryptology ePrint Archive, Report 2019/403. (2019). https://eprint.iacr.org/2019/403.