Pixel: Multi-signatures for Consensus

Manu Drijvers
DFINITY

Sergey Gorbunov
Algorand and University of Waterloo

Gregory Neven
DFINITY

Hoeteck Wee*
Algorand and CNRS, ENS, PSL

Abstract

In Proof-of-Stake (PoS) and permissioned blockchains, a committee of verifiers agrees and sign every new block of transactions. These blocks are validated, propagated, and stored by all users in the network. However, posterior corruptions pose a common threat to these designs, because the adversary can corrupt committee verifiers after they certified a block and use their signing keys to certify a different block. Designing efficient and secure digital signatures for use in PoS blockchains can substantially reduce bandwidth, storage and computing requirements from nodes, thereby enabling more efficient applications.

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, can be verified using three pairings and one exponentiation, and support non-interactive aggregation of individual signatures into a multi-signature. Pixel signatures are also forward-secure and let signers evolve their keys over time, such that new keys cannot be used to sign on old blocks, protecting against posterior corruptions attacks on blockchains. We show how to integrate Pixel into any PoS blockchain. Next, we evaluate Pixel in a real-world PoS blockchain implementation, showing that it yields notable savings in storage, bandwidth, and block verification time. In particular, Pixel signatures reduce the size of blocks with 1500 transactions by 35% and reduce block verification time by 38%.

1 Introduction

Blockchain technologies are quickly gaining popularity for payments, financial applications, and other distributed applications. A blockchain is an append-only public ledger that is maintained and verified by distributed nodes. At the core of the blockchain is a consensus mechanism that allows nodes to agree on changes to the ledger, while ensuring that changes once confirmed cannot be altered.

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 many known inefficiencies, low throughput, and high transaction latency [18, 28, 52]. 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 [22, 25, 31, 34, 36, 41, 50]. Permissioned blockchains such as Ripple [57] and Hyperledger Fabric [4] take yet another approach, sacrificing openness for efficiency by limiting participation in the network to a selected set of nodes.

All PoS-based blockchains, as well as permissioned ones, have a common structure where the nodes run a consensus sub-protocol to agree on the next block to be added to the ledger. Such a consensus protocol usually requires nodes to inspect block proposals and express their agreement by digitally signing acceptable proposals. When a node sees sufficiently many signatures from other nodes on a particular block, it appends the block to its view of the ledger.

Because the consensus protocol often involves thousands of nodes working together to reach consensus, efficiency of the signature scheme is of paramount importance. Moreover, to enable outsiders to efficiently verify the validity of the chain, signatures should be compact to transmit and fast to verify. Multi-signatures [37] have been found particularly useful for this task, as they enable many signers to create a compact and efficiently verifiable signature on a common message [16, 42, 61, 62].

The Problem of Posterior Corruptions. Chain integrity in a PoS blockchain relies on the assumption that the adversary controls less than a certain threshold (e.g., a third) of the total stake; an adversary controlling more than that fraction

*Authors are listed alphabetically.

may be able to fork the chain, i.e., present two different but equally valid versions of the ledger. Because the distribution of stake changes over time, however, the real assumption behind chain integrity is not just that the adversary *currently* controls less than a threshold of the stake, but that he never did so *at any time in the past*.

This assumption becomes particularly problematic if stake control is demonstrated through possession of signature keys, as is the case in many PoS and permissioned blockchains. Indeed, one could expect current stakeholders to properly protect their stake-holding keys, but they may not continue to do so forever, especially after selling their stake. Nevertheless, without additional precautions, an adversary who obtains keys that represent a substantial fraction of stake at *some* point in the past can compromise the ledger at *any* point in the future. The problem is further aggravated in efficient blockchains that delegate signing rights to a small committee of stakeholders, because the adversary can gain control of the chain after corrupting a majority of the committee members.

Referred to by different authors as *long-range attacks* [21], *costless simulation* [55], and *posterior corruptions* [12], this problem is best addressed through the use of *forward-secure* signatures [3,9,43,48]. Here, each signature is associated with the current time period, and 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, not previous ones. An adversary that corrupts an honest node can therefore not use the compromised key material to create forks in the past of the chain.

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 multi-signature comprises just two group elements and verification requires only three pairings, one exponentiation, and $N-1$ multiplications. Pixel signatures are almost as efficient as BLS multi-signatures, as depicted in Figure 1, but also satisfy forward-security; moreover, like in BLS multi-signatures, anybody can non-interactively aggregate individual signatures into a multi-signature.

Our construction builds on prior forward-secure signatures based on hierarchical identity-based encryption (HIBE) [15, 19,23,27] and adds the ability to securely aggregate signatures on the same message as well as to generate public parameters without trusted set-up.

We achieve security in the random oracle model under a variant of the bilinear Diffie-Hellman inversion assumption [11, 15]. At a very high level, the use of HIBE techniques allows us to compress $O(\log T)$ group elements in a tree-based forward-secure signature into two group elements, and secure aggregation allows us to compress N signatures under N public keys into a single multi-signature of the same size as a single signature.

To validate Pixel's design, we compared the performance of a Rust implementation [2] of Pixel with previous forward-secure tree-based solutions. We show how to integrate Pixel into any PoS blockchain. Next, we evaluate Pixel on the Algorand blockchain, showing that it yields notable savings in storage, bandwidth, and block verification time. Our experimental results show that Pixel is efficient as a stand-alone primitive and in use in blockchains. For instance, compared to a set of $N = 1500$ tree-based forward-secure signatures (for $T = 2^{32}$) at 128-bit security level, a single Pixel signature that can authenticate the entire set is 2667x smaller and can be verified 40x faster (c.f. Tables 1 and 3). Pixel signatures reduce the size of Algorand blocks with 1500 transactions by $\approx 35\%$ and reduce block verification time by $\approx 38\%$ (c.f. Figures 4 and 5).

1.2 Related Work

Multi-signatures can be used to generate a single short signature validates that a message m was signed by N different parties [6, 10, 14, 33, 37, 45, 46, 51, 53], Multi-signatures based on the BLS signature scheme [14, 16, 17, 56] 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. However, these signatures are not forward-secure.

Tree-based forward-secure signatures [9, 38, 43, 48] can be used to meet the security requirements, but they are not very efficient in an N-signer setting because all existing constructions have signature size at least $O(N \log T)$ group elements, where T is an upper bound on the number of time periods. Some schemes derived from hierarchical identity-based encryption (HIBE) [15, 19, 23] can bring that down to $O(N)$ group elements, which is still linear in the number of signers.

The only forward-secure multi-signature schemes that appeared in the literature so far have public key length linear in the number of time periods T [47] or require interaction between the signers to produce a multi-signature [58], neither of which is desirable in a blockchain scenario. The forward-secure multi-signature scheme of Yu et al. [64] has signature length linear in the number of signers, so is not really a multi-signature scheme.

Combining the generic tree-based forward-secure signature scheme of Bellare-Miner [9] with BLS multi-signatures [14, 17] gives some savings, but still requires $O(T)$ "certificates" to be included in each multi-signature. Batch verification [8] can be used to speed up verification of the certificates to some extent, but does not give us any space savings. Compared with existing tree-based forward-secure signatures in [9,38,43,48], our savings are two-fold:

- we reduce the size of the signature set for N commit-

scheme	key update	sign	verify	$\|\sigma\|$	$\|pk\|$	$\|sk\|$	forward security
BLS multi-signatures [14, 16, 56]	–	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.

tee members from $O(N \log T)$ group elements[1] to $O(1)$ group elements; and

- we reduce the verification time from $O(N)$ exponentiations to $O(1)$ exponentiation and $O(N)$ multiplications.

1.3 Paper Organization

The rest of this paper is organized as follows:

- In Section 2, we give a high level technical description of our new pairing-based forward-secure multi-signature scheme.
- In Sections 4 and 5, we describe the scheme in details. We prove the security of the construction in the random oracle model under a variant of a bilinear Diffie-Hellman inversion problem.
- In Section 6, we explain how to apply Pixel to PoS blockchains to solve posterior corruptions.
- In Section 7, we evaluate the efficiency savings for storage, bandwidth, and block verification time from using Pixel on the Algorand PoS blockchain.
- In Section 8, we describe various extensions to the scheme.
- In Appendix C, we perform theoretical efficiency analysis of Pixel.

2 Technical Overview

Our construction builds on prior forward-secure signatures based on hierarchical identity-based encryption (HIBE) [15, 19,23,27] and adds the ability to securely aggregate signatures on the same message as well as to generate public parameters without trusted set-up.

[1] Each tree-based signature comprise $O(\log T)$ group elements corresponding to a path in a tree of depth $\log T$ (see Section 7 for details), and there are N such signatures, one for which committee member.

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 [23] with the compact HIBE in [15]. 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. The design of multi-signature schemes must take into account 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 [14, 56]; it suffices here for each user to provide a standard BLS signature y' on its public key y (cf. the proof π in the key generation and verification algorithms in Section 5.2).

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 [20,63] evaluated on some fixed sequence of inputs (e.g. determined by the digits of pi), thereby avoiding any trusted set-up.

2.1 Discussion

Related works. The use of HIBE schemes for forward secrecy originates in the context of encryption [23] and has been used in signatures [19,27], key exchange [35] and proxy re-encryption [32]. Our signature scheme is quite similar to the forward-secure signatures of Boyen et al. [19] 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 [45].

Alternative approaches to posterior security. There are two variants of the posterior 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 [36], Ouroboros [41] and Casper [22] 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.

3 Preliminaries

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t$ be multiplicative groups of prime order q with a non-degenerate 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* [15], 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.

As shown in [15], the assumption holds in the generic bilinear group model, with a lower bound of $\Omega(\sqrt{q/\ell})$ (with a matching attack in [26]). Concretely, for the BLS12-381

pairing-friendly curve with $\ell = 32$, the best attack has complexity roughly 2^{125}.

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.

4 Forward-Secure Signatures

We begin by describing a forward-secure signature scheme, and then extend the construction to a multi-signature scheme in Section 5.

4.1 Definition

We use the Bellare-Miner model [9] 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.

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^*.

4.2 Encoding time periods

Following [23], we associate time periods with all nodes of the tree according to a pre-order traversal. Prior tree-based forward-secure signatures [9, 48] associate time periods with the only leaf nodes; using all nodes allows us to reduce the amortized complexity of key updates from $O(\log T)$ exponentiations to $O(1)$ exponentiations.

Recall that a tree of depth $\ell - 1$ has $2^\ell - 1$ nodes, which then correspond to time periods in $[2^\ell - 1]$. We will identify

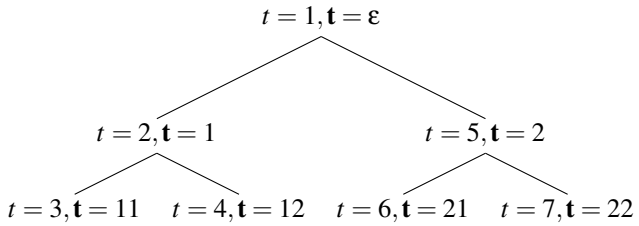

Figure 2: Tree structure illustrating bijection between $t \in [2^\ell - 1]$ and $\mathbf{t} \in \{1,2\}^{\leq \ell-1}$ for $\ell = 3$.

the nodes of the tree of depth $\ell-1$ with strings in $\{1,2\}^{\leq \ell-1}$ where 1 denotes taking the left branch and 2 denotes taking the right branch. An example of such a tree is depicted in Figure 2. We work with $\{1,2\}$ instead of $\{0,1\}$ for technical reasons: roughly speaking, in the scheme, we need to work with strings of length exactly $\ell-1$, which we obtain by padding strings in $\{1,2\}^{\leq \ell-1}$ with zeroes.

We can also describe the association explicitly as 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 $\varepsilon, 1, 11, 12, 2, 21, 22$ to $1, 2, 3, 4, 5, 6, 7$. The inverse of the bijection can be described as

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

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 $t, t+1$.

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.

4.3 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) [24] to the signature scheme determined by the key structure of the Boneh-Boyen-Goh HIBE scheme [15]; 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 $\widetilde{sk}_\varepsilon = \left(g_2^r \, , \, h^x h_0^r \, , \, h_1^r, \ldots, h_\ell^r\right)$ 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

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

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

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

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 $\mathbf{t}' \succeq \mathbf{t}$.

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 $\mathbf{t}' \succeq \mathbf{t}$, 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 [15, 19, 23], 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

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

  In our scheme, $pk = g_2^x$ lies in $\mathbb{G}_2$ instead of $\mathbb{G}_t$ and is therefore smaller. Setting h to be random instead of g_1 also allows us to achieve security under weaker assumptions. In fact, setting $h = g_1$ and $pk = g_2^x$ would yield an insecure scheme in symmetric pairing groups where $g_1 = g_2$, since $h^x = g_1^x = g_2^x = pk$.

4.4 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 $\mathbf{t}' \succ \mathbf{t}$.

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 $\mathbf{t}' \succ \mathbf{t}$ 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

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

4.5 Security

Theorem 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*

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

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

We refer the interested reader to Appendix A for the full proof of security.

5 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 [56] using proofs of possession (denote by π below) to prevent against these types of attack.

5.1 Definitions

In addition to the algorithms of a forward-secure signature scheme in Section 4.1, a forward-secure multi-signature scheme $\mathcal{FMS}$ in the key verification model has a key generation 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 a multi-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. 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 $\Sigma \xleftarrow{\$} \mathsf{SAgg}((pk_1, \sigma_1), \ldots, (pk_n, \sigma_n), t, M)$.

Security. Unforgeability (fu-cma) is defined through a game that is similar to that described in Section 4.1. 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^*.

Our security model covers rogue-key attacks because the adversary first receives the target public key pk, and only then outputs the list of public keys $pk_1^*,...,pk_n^*$ involved in its forgery. The only condition on these public keys is that they are accompanied by valid proofs $\pi_1^*,...,\pi_n^*$.

5.2 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 4.3, 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 $y' \leftarrow \mathsf{H}_{\mathbb{G}_1}(\mathtt{PoP}, y)$, where $\mathtt{PoP}$ 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

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

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. Multi-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 a multi-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

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

5.3 Security

Theorem 2. *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 $\mathcal{A}$.

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 4.3 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 1.

The key idea for the proof of security following [56] 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 $\mathsf{H}_{\mathbb{G}_1}(\mathtt{PoP}, y_i) = h^{r_i}$.
- 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. Here, we use the proofs $\pi_i^* = (h_i^x)_i^r$ to extract h_i^x for all adversarial keys $pk_i^* = g_2^{x_i}$.

We defer the interested reader to Appendix B for proof details.

6 Pixel in PoS-based Blockchains

In this section, we describe how to integrate Pixel into PoS-based blockchains that rely on forward-secure signatures to achieve security against posterior corruptions. We summarize systems that rely on forward-secure signatures, abstract how signatures are used in these systems, and explain how to apply Pixel.

PoS Blockchains Secure under Posterior Corruptions. Ouroboros Genesis and Praos rely on forward-secure signatures to protect against posterior corruptions [5,31,40]. These blockchains require users to rotate key and assume secure erasures. Thuderella is a blockchain with fast optimistic instant confirmation [54]. The blockchain is secure against posterior corruptions assuming that a majority of the computing power is controlled by honest players. Similarly, the protocol relies on forward-secure signatures. Pixel can be applied in all these blockchains to protect against posterior attacks and potentially reduce bandwidth, storage, and computation costs in instances where many users propagate many signatures on

the same message (e.g., a block of transactions). Ouroboros Crypsinous uses forward-secure encryption to protect against the same attack [39]. Snow White shows that under a mild setup assumption, when nodes join the system they can access a set of online nodes the majority of whom are honest, the system can defend against posterior corruption attack [30]. The system does not rely on forward-secure signatures.

Background on PoS Blockchains. A blockchain is an append-only public ledger to which anyone can write and read. The fundamental problem in blockchains is to agree on a block of transactions between users. In Proof-of-Stake protocols, users map the stake or tokens they own in the system to "voting power" in the agreement protocol. Various types of PoS systems exist that use different formulas for determining the weight of each vote. For instance, in bounded PoS protocols, users must explicitly lock some amount A of their tokens to participate in the agreement. The weight of each vote is A/Q, where Q is the total number of locked tokens who's users wish to participate in the agreement. Users that misbehave are punished by a penalty applied to their locked tokens.

To tolerate malicious users, all PoS protocols run a Byzantine sub-protocol to agree on a block of transactions. The system is secure, assuming that that majority (often $2/3$) of the tokens participating in the consensus is honest. Each block is valid if a majority of committee members, weighted by their stake, approved it.

Pixel **Integration.** In order to vote on a block B, each member of the sub-protocol signs B using Pixel with the current block number. The 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. 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.

Registering public keys. Each user who wishes to participate in consensus needs to register a participation signing key. A user first samples a Pixel key pair and generates a corresponding PoP. The user then issues a special transaction (signed under her spending key) registering the new participation key. The transaction includes PoP. PoS verifiers who are selected to run an agreement at round r, check (a) validity of the special transaction, and (b) validity of PoP. If both checks pass, the user's account is updated with the new participation key. From this point, if selected, the user signs on blocks using Pixel.

Vote generation. To generate a vote on a block number t, users first update their keys to correspond to the round number. Subsequently, they sign the block using the correct secret key and propagate the signature to the network.

Propagating and aggregating signatures. Individual committee signatures will be propagated through the network 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 add a new signature to a multi-signature to obtain a new multi-signature. In practice, this means that propagating nodes can perform intermediate aggregation on any number of committee signatures and propagate the result, until the block certificate is formed. Alternatively, nodes can aggregate all signatures just before writing a block to the disk. That is, upon receiving enough certifying votes for a block, a node can aggregate N committee members' signatures into a multi-signature and then write the block and the certificate to the disk. To speed up verification of individual committee member signatures, a node could pre-compute $e(h, y)$ for the y's corresponding to the users with the highest stakes.

Key updates. When using Pixel in block-chains, time corresponds to the block number or sub-steps in consensus protocols. Naively, when associating time with block numbers, 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. 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.

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 C.

7 Evaluation on Algorand Blockchain

In order to measure the concrete efficiency gains of Pixel, we evaluate it on the Algorand blockchain [59, 60].

Algorand Overview. Algorand is a Pure PoS (PPOS) system, where each token is mapped to a single vote in the consensus without any explicit bonding [59, 60]. Some users may opt-out from participation, in which case their tokens are excluded from the total number of participating tokens (i.e., the denominator in the weight). Each user maintains an *account state* on-chain that specifies her spending key, balance, consensus participation status, participation key, and other auxiliary information. A user wishing to perform a transaction must sign it with her corresponding secret key. Users run a Byzantine consensus algorithm to agree on a block of transactions following the high-level structure we outlined in the previous section. We call a *block certificate* to denote a collection of votes above a certain threshold approving a block. All users in the network validate and store block certificate (and the corresponding transactions) on disk. We refer to a *node* as a computer system running Algorand client software on the user's behalf.

Verifier Vote Structure and Block Certificates. In Algorand, each valid vote for a block proposal includes (a) a proof that the verifier was indeed selected to participate in the consensus at round r, and (b) a signature on the block proposal. In more detail, each vote includes the following fields:

- Sender identifier which is represented by a unique public key registered on-chain (32 bytes).
- Round and sub-step identifiers (8 bytes).
- Block header proposal (32 bytes).
- A seed used as an input to a VRF function for cryptographic sortition (32 bytes).
- VRF credential that proves that the sender was indeed chosen to sign on the block (96 bytes).
- Forward-secure signature authenticating the vote (256 bytes).

Overall, each vote is about 500 bytes (including some additional auxiliary information), half of which is for the forward-secure signature.

Algorand has two voting sub-steps for each round. In the first sub-step, a *supporting* set (of expected size 3000) of verifiers is chosen to vote on a block proposal. In the second sub-step, a *certifying* set (of expected size 1500) of verifiers is chosen to finalize the block proposal. All verifiers' votes propagate in the network during the agreement, but only the *certifying* votes are stored long-term and sufficient to validate a block in the future. Larger *recovering* set (of expected size 10000) is chosen during a network partition for recovery.

Sig. set size	BM-Ed25519	BM-BLS	Pixel
1	256 B	192 Bytes	144 B
1500	375 KB	141 KB	144 B
3000	750 KB	281 KB	144 B
10000	2.4 MB	938 KB	144 B

Table 1: Total size of signature sets using various forward-secure signature schemes for 2^{32} time periods. BM-Ed25519 is instantiated using Algorand's parameters with $10,000$-ary tree of depth 2. BM-BLS is instantiated using the same parameters with public keys in $\mathbb{G}_1$ and signatures in $\mathbb{G}_2$.

Algorand's Existing Solution to Posterior Corruptions. Algorand solves posterior corruptions using forward-secure signatures instantiated with a d-ary certificate tree [9], which we call BM-Ed25519 for convenience. The root public key of an Ed25519 signature scheme is registered on-chain, and keys associated with the leaves (and subsequently used to sign at each round) are stored locally by the potential verifiers. For each block at round r a verifier must (a) produce a valid certificate chain from the root public key to the leaf associated with r, and (b) signature of the vote under the leaf key. Algorand assumes secure erasures and that users delete old keys from their nodes. BM-Ed25519 is instantiated with 10000-ary tree and depth 2 (supporting approximately $2^{26.6}$ time periods). Ed25519-based signatures have public keys of 32 bytes and 64 bytes signatures. Hence, since a valid certificate chain must include the intermediary public keys, the resulting size of each forward-secure signature is $3 \times 64 + 2 \times 32 = 256$ bytes.

7.1 Efficiency Evaluation

Pixel signatures can serve as a replacement of BM-Ed25519 in Algorand following the same design as outlined Section 6.

Setup. Our experiments are performed on a MacBook Pro, 3.5 GHz Intel Core i7 with 16 GB DDR3. We use Alogrand's open-source implentations of Pixel signatures, VRF functions, Ed25519 signing, and verification [1, 2]. For blockchain applications, since the public key must live on-chain, we choose to place Pixel public keys in $\mathbb{G}_1$, obtaining smaller public keys and faster key aggregation during verification. We set the maximum time epoch to $T = 2^{32} - 1$, which is sufficient to rotate a key every second for 136 years.

Figure 3 shows the runtime of individual Pixel algorithms, aggregation, and object sizes for the BLS12-381 curve [7]. Next, we measure quantities that affect all nodes participating in the system: the size of signature sets, bandwidth, and block verification time. In Pixel, the signature set corresponds to a single multi-signature.

	keygen	key update	sign	aggregate ($N = 1500$)	verify agg. ($N = 1500$)	aggregate ($N = 1500$)	verify agg. ($N = 3000$)	$\|pk\|$	$\|\sigma\|$	$\|sk_t\|$
$pk \in \mathbb{G}_1$	1.03 ms	1.8 ms	2.8 ms	7.2 ms	6.7 ms	13.9 ms	8.3 ms	48 B	144 B	43 kB

Figure 3: 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 signatures and keys aggregated, respectively. Maximum number of time periods is $T = 2^{32} - 1$.

Storage Savings. In Table 1, we compare the sizes of signature sets that are propagated (for supporting and verifying votes) and stored (for verifying votes) by all participating nodes. We instantiate BM-Ed25519 with Algorand parameters of 10000-ary and depth 2. For BM-BLS we place the public key in $\mathbb{G}_1$ and signatures in $\mathbb{G}_2$. Since BLS supports aggregation of signatures, we can compress all signatures in a certificate chain and the signature of the block into 96 B (note that the public key in the certificate chain cannot be compressed and adds an additional 96 B per signature). Furthermore, we can compress all signatures across votes. Pixel signatures authenticating a block with 1500 signatures are 2667x and 1003x times smaller than signature sets using BM-Ed25519 and BM-BLS, respectively.

In Figure 4, we show long-term blockchain storage improvements using Pixel signatures. We evaluate storage assuming various number of transactions in each block. Each transaction in Algorand is about 232 bytes. We also assume that the entire expected number of certifying verifiers (1500) are selected for each block. Given today's block confirmation time of just under 4.3 seconds per block, Algorand blockchain should produce 10^6 blocks every ≈ 50 days and 10^8 blocks every ≈ 13 years. Pixel signatures improve blockchain size by about 40% and 20% on blocks packed with 1500 and 5000 transactions, respectively. This improvement translates to smaller overall storage requirements and faster catch-up speed for new nodes.

We clarify that the savings we obtain from Pixel are complementary to those of Vault [44], which is another system built on top of Algorand to improve storage and catch-up speed. In particular, Vault can be used in conjunction with Pixel to obtain further storage savings. Vault creates "jumps" between blocks so that the system can confirm block r knowing only block $r - k$ for some parameter k (e.g., $k = 100$). Instead of downloading every block, a catch-up node in Vault only needs to download every kth block. Even using Vault, users would need to download and store about 10^6 blocks for every ≈ 13 years of blockchain operation.

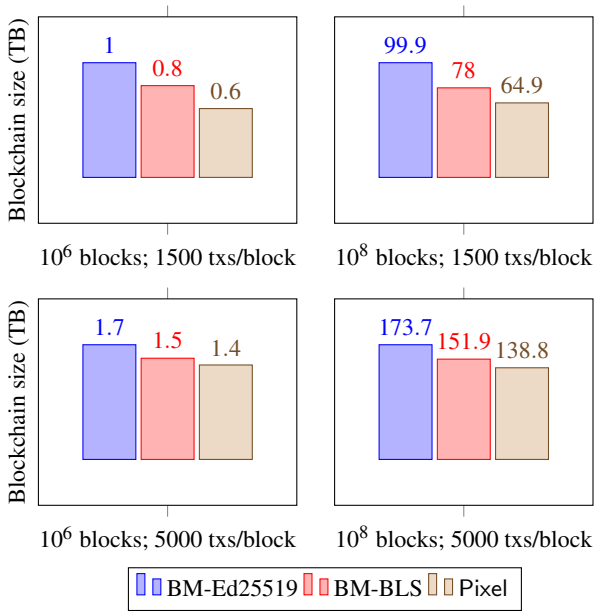

Figure 4: Size of blockchain measured for different total number of blocks. The top two plots assume average of 1500 transactions per block and the bottom plots assume 5000 transactions per block. All plots assume average of 1500 certifying votes per block.

Bandwidth Savings. Algorand uses a relay-based propagation model where users' nodes connect to a network of relays (nodes with more resources). Without aggregation during propagation, Pixel savings for the bandwidth for both relays

Number of connections	BM-Ed25519	Pixel
4	4.4 MB	2.5 MB
10	11 MB	6.2 MB
100	109.9 MB	61.8 MB

Table 2: Total bandwidth to propagate a set of 4500 signatures during consensus to agree on a block of transactions.

Sig. set size	BM-Ed25519	Pixel	Improvement
1	0.18 ms	4.9 ms	27x slower
1500	270 ms	6.7 ms	40x faster
3000	540 ms	8.3 ms	65x faster
10000	1.8 sec	15.6 ms	115x faster

Table 3: Total runtime to verify signature sets authenticating a block. Pixel verification includes the time to aggregate public keys.

and regular nodes come from smaller signatures sizes. Each relay can serve dozens or hundreds of nodes, depending on the resources it makes available. A relay must propagate a block of transactions and the corresponding certificate (with 1500 votes) to each node that it serves. During consensus, however, an additional 3000 *supporting votes* are propagated for every block. Each node connects to 4 randomly chosen relays. Every vote that the node receives from a relay, it propagates to the remaining 3 relays. Duplicate votes are dropped, so each vote propagates once on each connection. In Table 2, we summarize savings for 4500 votes propagated during consensus for each block. From the table, we see that a relay with 10 connections saves about 44% of bandwidth. Bandwidth can be improved even further if Algorand relays were to aggregate multiple votes before propagating them to the users.

Block Verification Time Savings. Since verifying a Pixel multi-signature requires only 3 pairings in addition to multiplying all the public keys in the signature set, they are faster to verify than BM-Ed25519 signatures sets. Table 3 shows that a set of 3000 signatures can be verified about 65x faster. In Figure 5, we measure the overall savings on block verification time. Block verification time is broken into three main intervals: (a) time to verify vote signatures, (b) time to verify vote VRF credentials, and (c) time to verify transactions. In each interval, signature verification dramatically exceeds the time of any additional checks (e.g., check that the transaction amount is higher than the user's balance). Blocks with 1500 and 5000 transactions can be verified 38% and 29% faster, respectively.

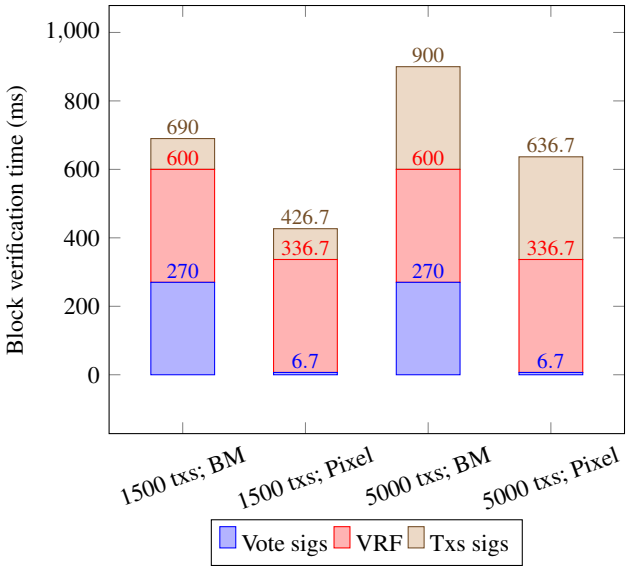

Figure 5: Overall Algorand block verification time using BM-Ed25519 and Pixel signatures. Each block is assumed to contain 1500 certifying votes. The two plots on the left assume 1500 txs/block; whereas the two plots on the right assume 5000 txs/block.

8 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 [13] 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" [43].

Using internal nodes. For ease of exposition, our scheme only assigns time periods to leaf nodes in the tree. Alternatively, one could follow the approach of [24] to use all nodes in the tree, in a pre-order traversal, as time periods, which will improve the efficiency of the key update algorithm. Time periods are then identified by bit strings of length at most ℓ, rather than exactly ℓ bits, and a signature in time period $t = t_1 \ldots t_d$ is a tuple of the form

$$(\sigma_1, \sigma_2) \;=\; \left(h^x \cdot \left(h_0 \prod_{j=1}^{d} h_j^{t_j} \cdot h_{\ell+1}^{\mathsf{H}_q(M)} \right)^r \,,\; g_2^r \right) .$$

Details are left to the reader.

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 $sk_{\textit{scope},1} \leftarrow \mathsf{H}_{\mathbb{G}_1}(\texttt{scope}, \textit{scope})^x$ 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 $T \cdot q_{\mathrm{H}}$ can be brought down to $T \cdot q_{\mathrm{S}}$ using Coron's technique [29], 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 [16, 49] 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.

9 Conclusion

In this work, we focus on improving the speed and security of PoS consensus mechanisms via optimizing its core building block – digital signature scheme. We design a new pairing-based forward-secure multi-signature scheme, Pixel. We prove that Pixel is secure in the random oracle model under a variant of Diffie-Hellman inversion problem over bilinear groups. Pixel is efficient as a stand-alone primitive and results in significant performance and size reduction compared to the previous forward-secure signatures applied in settings where multiple users sign the same message (block). For instance, compared to a set of 1500 tree-based forward-secure signatures, a single Pixel signature that can authenticate the entire set is 2667x smaller and can be verified 40x faster. We explained how to integrate Pixel to any PoS blockchains to solve posterior corruptions problem. We also demonstrate that Pixel provides significant efficiency gains when applied to Algorand blockchain. Pixel signatures reduce the size of Algorand blocks with 1500 transactions by $\approx 35\%$ and reduce block verification time by $\approx 38\%$.

Acknowledgments

We would like to thank Zhenfei Zhang for implementing Pixel as well as his help with Section 7. In addition, we thank Jens Groth, Nickolai Zeldovich, our shepherd Ari Juels, and the anonymous reviewers for useful feedback.

References

[1] Algorand's official implementation in go. `https://github.com/algorand/go-algorand`, 2019.

[2] Algorand's official pixel implementation. `https://github.com/algorand/pixel`, 2019.

[3] Ross Anderson. Two remarks on public-key cryptology. Manuscript. Relevant material presented by the author in an invited lecture at the 4th ACM Conference on Computer and Communications Security, CCS 1997, Zurich, Switzerland, April 1–4, 1997, September 2000.

[4] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Rui Oliveira, Pascal Felber, and Y. Charlie Hu, editors, *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018*, pages 30:1–30:15. ACM, 2018.

[5] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 913–930, New York, NY, USA, 2018. ACM.

[6] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008.

[7] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Heidelberg, September 2003.

[8] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 236–250. Springer, Heidelberg, May / June 1998.

[9] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999.

[10] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

[11] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[12] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. `http://eprint.iacr.org/2016/919`.

[13] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012.

[14] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003.

[15] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, Heidelberg, May 2005.

[16] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Heidelberg, December 2018.

[17] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[18] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121. IEEE Computer Society Press, May 2015.

[19] Xavier Boyen, Hovav Shacham, Emily Shen, and Brent Waters. Forward-secure signatures with untrusted update. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 191–200. ACM Press, October / November 2006.

[20] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010.

[21] Vitalik Buterin. Long-range attacks: The serious problem with adaptive proof of work. `https://blog.ethereum.org`, 2014.

[22] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[23] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, Heidelberg, May 2003.

[24] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, July 2007.

[25] Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. Algorand agreement: Super fast and partition resilient byzantine agreement. Cryptology ePrint Archive, Report 2018/377, 2018.

[26] Jung Hee Cheon. Security analysis of the strong Diffie-Hellman problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 1–11. Springer, Heidelberg, May / June 2006.

[27] Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu, and K. P. Chow. Secure hierarchical identity based signature and its application. In Javier López, Sihan Qing, and Eiji Okamoto, editors, *ICICS 04*, volume 3269 of *LNCS*, pages 480–494. Springer, Heidelberg, October 2004.

[28] M. Conti, E. Sandeep Kumar, C. Lal, and S. Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys Tutorials*, 20(4):3416–3452, Fourthquarter 2018.

[29] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Heidelberg, August 2000.

[30] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security FC*, 2019.

[31] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.

[32] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 219–250. Springer, Heidelberg, March 2018.

[33] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019.

[34] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. ACM.

[35] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 519–548. Springer, Heidelberg, April / May 2017.

[36] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system, 2018.

[37] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. Technical report, NEC Research and Development, 1983.

[38] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 332–354. Springer, Heidelberg, August 2001.

[39] Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *IEEE Symposium on Security and Privacy SP*, pages 157–174, 2019.

[40] Thomas Kerber, Markulf Kohlweiss, Aggelos Kiayias, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. Cryptology ePrint Archive, Report 2018/1132, 2018. `https://eprint.iacr.org/2018/1132`.

[41] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

[42] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 279–296. USENIX Association, August 2016.

[43] Hugo Krawczyk. Simple forward-secure signatures from any signature scheme. In Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati, editors, *ACM CCS 2000*, pages 108–115. ACM Press, November 2000.

[44] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. Vault: Fast bootstrapping for the algorand cryptocurrency. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, 2019.

[45] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Heidelberg, May / June 2006.

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

[47] Di Ma and Gene Tsudik. Forward-secure sequential aggregate authentication. In *2007 IEEE Symposium on Security and Privacy (S&P 2007)*, pages 86–91. IEEE Computer Society, 2007.

[48] Tal Malkin, Daniele Micciancio, and Sara K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 400–417. Springer, Heidelberg, April / May 2002.

[49] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptography*, 2019.

[50] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[51] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, November 2001.

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

[53] Kazuo Ohta and Tatsuaki Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 139–148. Springer, Heidelberg, November 1993.

[54] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT*, pages 3–33, 2018.

[55] Andrew Poelstra. On stake and consensus. `https://download.wpsoftware.net/bitcoin/pos.pdf`, 2015.

[56] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer, Heidelberg, May 2007.

[57] David Schwartz, Noah Youngs, and Arthur Britto. The Ripple protocol consensus algorithm. Ripple Labs Inc White Paper, `https://ripple.com/files/ripple_consensus_whitepaper.pdf`, 2014.

[58] N. R. Sunitha and B. B. Amberker. Forward-secure multi-signatures. In Manish Parashar and Sanjeev K. Aggarwal, editors, *Distributed Computing and Internet Technology, 5th International Conference, ICDCIT 2008*, volume 5375 of *Lecture Notes in Computer Science*. Springer, 2009.

[59] Algorand Team. Algorand blockchain features specification version 1.0. Github, 2019.

[60] Algorand Team. Algorand byzantine fault tolerance protocol specification. Github, 2019.

[61] The Elrond Team. Elrond: A highly scalable public blockchain via adaptive state sharding and secure proof of stake. `https://elrond.com/files/Elrond_Whitepaper_EN.pdf`, 2019.

[62] The ZILLIQA Team. The zilliqa technical whitepaper, 2017. `http://docs.zilliqa.com/whitepaper.pdf`.

[63] Riad S. Wahby and Dan Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR TCHES*, 2019(4):154–179, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8348`.

[64] Jia Yu, Fanyu Kong, Xiangguo Cheng, Rong Hao, Yangkui Chen, Xuliang Li, and Guowen Li. Forward-secure multisignature, threshold signature and blind signature schemes. *Journal of Networks*, 5(6):634–641, 2010.

A Security Proof of Forward-Secure Signatures

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

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

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

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

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

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

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

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

Because $1 \leq k \leq \ell - 1$, 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

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

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

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

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 4.2), 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 $e(g_1^{(\alpha^{\ell+1})}, C_2) = e(g_1, g_2)^{(\gamma \alpha^{\ell+1})}$. It does so whenever $\mathcal{A}'$ is successful, so that

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

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 $\bar{t}' \geq \bar{t}$. The advantage of $\mathcal{A}'$ is given by

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

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. □

B Security Proof of Forward-Secure Multi-signatures

Proof. We show how to construct a forger $\mathcal{A}$ for the multi-signature scheme yields a forger $\mathcal{A}'$ for the single-signer scheme of Section 4.3 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 1.

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

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 $(\mathtt{PoP}, z)$, $\mathcal{A}'$ 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 $y_i' = \mathsf{H}_{\mathbb{G}_1}(\mathtt{P} \circ \mathtt{P}, y_i)^{x_i}$ 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. □

C Efficiency Analysis

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 $\mathbb{G}_2$.