Pixel+ and Pixel++: Compact and Efficient Forward-Secure Multi-Signatures for PoS Blockchain Consensus

Jianghong Wei[1, 2] Guohua Tian[1] Ding Wang[3] Fuchun Guo[4] Willy Susilo[4] Xiaofeng Chen[1*]

[1] *State Key Laboratory of Integrated Service Networks (ISN), Xidian University*
[2] *State Key Laboratory of Mathematical Engineering and Advanced Computing*
[3] *College of Cyber Science, Nankai University*
[4] *School of Computing and Information Technology, University of Wollongong*

Abstract

Multi-signature schemes have attracted considerable attention in recent years due to their popular applications in PoS blockchains. However, the use of general multi-signature schemes poses a critical threat to the security of PoS blockchains once signing keys get corrupted. That is, after an adversary obtains enough signing keys, it can break the immutable nature of PoS blockchains by forking the chain and modifying the history from some point in the past. Forward-secure multi-signature (FS-MS) schemes can overcome this issue by periodically updating signing keys. The only FS-MS construction currently available is Drijvers et al's Pixel, which builds on pairing groups and only achieves forward security at the time period level.

In this work, we present new FS-MS constructions that either are free from pairing or capture forward security at the individual message level (i.e., fine-grained forward security). Our first construction Pixel+ works for a maximum number of time periods T. Pixel+ signatures consist of only one group element, and can be verified using two exponentiations. It is the first FS-MS from RSA assumption, and has 3.5x and 22.8x faster signing and verification than Pixel, respectively. Our second FS-MS construction Pixel++ is a pairing-based one. It immediately revokes the signing key's capacity of re-signing the message after creating a signature on this message, rather than at the end of the current time period. Thus, it provides more practical forward security than Pixel. On the other hand, Pixel++ is almost as efficient as Pixel in terms of signing and verification. Both Pixel+ and Pixel++ allow for non-interactive aggregation of signatures from independent signers and are proven to be secure in the random oracle model. In addition, they also support the aggregation of public keys, significantly reducing the storage overhead on PoS blockchains.

We demonstrate how to integrate Pixel+ and Pixel++ into PoS blockchains. As a proof-of-concept, we provide implementations of Pixel+ and Pixel++, and conduct several representative experiments to show that Pixel+ and Pixel++ have good concrete efficiency and are practical.

*Corresponding author (xfchen@xidian.edu.cn).

1 Introduction

The proof-of-stake (PoS) consensus protocol achieves the decentralization and immutability of blockchain in an energy-efficient way, and has been widely studied both in the industry (e.g., DFINITY and Algorand[1]) and academia [16, 31, 54]. In most PoS-based blockchain consensus protocols, all selected block proposers need to check the validity of block proposals and state their own agreement by generating digital signatures on acceptable block proposals. When the number of received signatures on the same block proposal reaches a pre-specified threshold, a participant appends the block proposal to the local blockchain. Due to the large number of protocol participants, improving the efficiency of the underlying signature scheme becomes extremely important to the usability of PoS blockchain consensus protocols.

Multi-signature schemes [13, 63] allow multiple signers to jointly generate a space and time-efficient signature on a single message so that a verifier can be convinced that all participants signed this message. Although this cryptographic primitive has been introduced and studied for decades, it has recently been drawing new attention due to it perfectly fulfilling the requirements of the above scenario. Specifically, given n participants indexed by their own public/secret key pairs $\{(pk_i, sk_i)\}_{i=1}^{n}$, each participant i computes a signature σ_i on a message m using sk_i. Then, a third party (e.g., a miner) can compress these signatures $\{\sigma_i\}_{i=1}^{n}$ into a short multi-signature σ. Consequently, a verifier can determine whether every participant approved the message m by just checking the correctness of σ. This significantly reduces not only the time to check that a consensus on the next block has been reached, but also the corresponding block storage.

Despite the obvious advantages of applying multi-signature to PoS blockchain settings, they are vulnerable to long-range attacks (a.k.a. posterior corruptions and costless simulation) [26, 42] due to the leakage of signing keys. More concretely, if the stake ownership is shown through the possession of the corresponding signing key, then an adversary can fork

[1] https://dfinity.org/, https://algorand.com/.

the chain and modify the history from some point in the past when he obtained substantial signing keys. On the other hand, from a practical point of view, those users with relatively small stakes may not protect their accounts well compared to active users. Moreover, after users sell their stakes, they may also no longer need to maintain their signing keys. These realistic human factors further exacerbate this attack.

A well-studied approach to mitigating the damage of signing key exposure is to use forward-secure signatures [3, 25]. Roughly, in the setting of such a cryptographic primitive, the user's signing key evolves unidirectionally over time periods, and each signature is associated with the corresponding time period in which the user generated it. As a consequence, given the exposed current signing key, the adversary cannot use it to forge signatures from previous time periods. Focusing on the scenario of PoS blockchains, this means that an adversary will fail to create any forks in the past of the blockchain, even if he has controlled a total stake above a certain threshold.

To date, we are aware of only one forward-secure multi-signature (FS-MS) scheme Pixel due to Drijvers et al. [35]. By using a new manner of encoding time periods, Pixel achieves substantial savings in bandwidth and verification effort. On the other hand, as in previous forward-secure signatures, it still follows the key delegation mechanism of hierarchical identity-based encryption (HIBE) [20], and thus inevitably has logarithmic cost. In addition, as demonstrated by Green and Miers [45], this traditional forward security is coarse-grained, that is, an adversary can still utilize the corrupted signing key to create signatures during the current time period. Conversely, fine-grained forward security is more desirable since it allows a user to individually revoke the signing key's capacity of re-signing a message, without waiting for a time period to elapse. Moreover, Pixel builds its security on a non-standard q-type complexity assumption over bilinear groups, and involves computation-heavy pairing operations.

1.1 Our Results

Given the significant advantage of the primitive of *forward-secure multi-signatures* to efficiently remedy long-range attacks in PoS blockchains, it is desirable to study different FS-MS constructions that provide various performance and security trade-offs, giving users application-specific choices.

In this work, we propose two new FS-MS constructions under different cryptographic assumptions. Our first construction Pixel+ is based on the RSA assumption, and achieves more efficient signing and verification than Pixel. In more detail, Pixel+ works for a bounded number of time periods T, which is specified in the system setup phase. The main technical challenge is how to come up with an efficient solution for an T value large enough to be practical. For example, under the demand that a PoS blockchain needs to be maintained for fifty years and the time to generate a block is ten seconds, then we need to assign $T = 2^{28}$. To this end, we use the RSA sequencer [51] as the core building block, which is an abstraction of pebbling techniques of Itkis and Reyzin [53] and provides a trade-off between computation cost and storage overhead. Specifically, in the scenario of T time periods and n participants, a multi-signature of Pixel+ comprises just one group element, and the verification algorithm only requires two hash operations and two exponentiations, and the size of both public parameters and signing key scales with $\log T$.

Our second construction, Pixel++, is designed to achieve more practical (i.e., fine-grained) forward security and efficient key evolution. In more detail, inspired by Green and Miers' [45] elegant work of puncturable encryption, we introduce the notion of puncturable multi-signature scheme[2], where a signing key is updated by repeatedly and sequentially puncturing it with tags associated with signed messages. As a consequence, the updated (or punctured) signing key cannot be used to create valid signatures on previous pairs of tags and messages. Pixel++ builds on bilinear groups, and uses the probabilistic data structure Bloom filter to maintain signing keys. A multi-signature of Pixel++ is comprised of two group elements and an integer, the verification requires four pairings and two exponentiations. Particularly, the key update only needs a few deletion operations, and thus is very efficient. This comes at the cost of additional storage overhead and a non-negligible probability of failing to sign.

We prove the security of Pixel+ and Pixel++ in random oracle models under the RSA assumption and a q-type assumption, respectively. Our forward-secure multi-signature constructions provide attractive solutions to the problem of long-range attacks in PoS blockchain consensus protocols. As depicted in Table 1, Pixel+ is the only one that builds on the RSA assumption. Its signing and verification algorithms require one and two exponentiations respectively, and are more efficient than Pixel. Similar to Pixel, our second construction, Pixel++, also builds on bilinear groups. But it provides more practical forward security and more efficient key update. Compared with multi-signature schemes based on discrete logarithm assumptions [60, 61], ours eliminate the interaction among independent signers.

We provide implementations of Pixel+ and Pixel++ using Python. We conduct representative experiments that evaluate their computation cost and storage/communication overhead under different usage scenarios, and compare their performance with previous forward-secure multi-signature schemes. The experimental results indicate that Pixel+ and Pixel++ are efficient for practical applications. For instance, in the setting of 100 signers and $T = 2^{32}$ time periods, the signature sizes of Pixel+ and Pixel++ are 0.90 KB and 0.92 KB, respectively. At the same time, the signing time of Pixel+ and Pixel++ are 0.07 s and 0.2 s respectively, and the verification time is 0.13 s and 2.91 s respectively. We also demonstrate how to integrate them into PoS blockchain to resist long-range attacks.

[2] We consider puncturable multi-signature schemes as FS-MS schemes.

Table 1: The comparison of existing FS-MS schemes with ours. Here, $|\mathbb{G}_i|$ $(i = 1,2)$ and $|\mathbb{Z}_N|$ denote the sizes of an element of the bilinear group and RSA group. $\mathbb{G}$ is a group of order p. E and P are the computation cost of one exponentiation and one pairing. We omit lightweight operations like addition, multiplication and hash. T refers to the maximum number of time periods. ℓ denotes the parameter of a Bloom filter. FS-I and FS-II represent traditional forward security and fine-grained forward security.

Scheme	Sig. size	pk size	sk size	Sign	Verify	Key Update	Assumption	Rounds	FS-I	FS-II	Trusted Setup
MuSig [60]	1 $(\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p\rvert)$	1 $\lvert\mathbb{G}\rvert$	1 $\lvert\mathbb{Z}_p\rvert$	1 E	2 E	–	DL	3	✗	✗	✗
MuSig2 [61]	1 $(\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p\rvert)$	1 $\lvert\mathbb{G}\rvert$	1 $\lvert\mathbb{Z}_p\rvert$	7 E	2 E	–	AOMDL	2	✗	✗	✗
Boneh et al. [21]	1 $\lvert\mathbb{G}_1\rvert$	1 $\lvert\mathbb{G}_2\rvert$	1 $\lvert\mathbb{Z}_p\rvert$	1 E	2 P	–	co-CDH	0	✗	✗	✗
Pixel [35]	1 $(\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert)$	1 $\lvert\mathbb{G}_2\rvert$	$O((\log T)^2)$ $\lvert\mathbb{G}_1\rvert$ + $O(\log T)$ $\lvert\mathbb{G}_2\rvert$	4 E	1 E + 3 P	2 E	q-wBDHI$_3^*$	0	✓	✗	✗
Pixel+ (Section 4.2)	1 $\lvert\mathbb{Z}_N\rvert$	1 $\lvert\mathbb{Z}_N\rvert$	$O(\log T)$ $\lvert\mathbb{Z}_N\rvert$	1 E	2 E	$O(\log T)$ E	RSA	0	✓	✗	✓
Pixel++ (Section 5.2)	1 $(\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert)$	1 $\lvert\mathbb{G}_2\rvert$	$O(\ell)$ $(\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert)$	5 E	2 E + 3 P	0	q-BDHE$_3^*$	0	✓	✓	✗

1.2 Related Work

1.2.1 Multi-signature Schemes

The notion of multi-signatures was first put forth by Itakura and Nakamura [52] 40 years ago, and has received renewed attention in recent years, mainly motivated by novel real-world applications like blockchains and cryptocurrencies. We can acquire a trivial multi-signature scheme from any signature by just concatenating each signer's individual signature. Therefore, a multi-signature only makes sense if its length is independent of the number of signers.

To produce compact multi-signatures, most of available multi-signature schemes build on Schnorr signature. Bellare and Neven [12] proposed the first multi-signature with provable security in the plain public-key model, where each signer is required to generate a proof of possession of the secret key [66], so as to resist rogue public-key attacks. The signing algorithm of their scheme needs three rounds of interaction among signers. After that, many two-round multi-signature schemes were proposed [8, 58, 74]. However, Drijvers et al. [34] found that these schemes suffer from concurrent attacks, and put forth a secure scheme based on Bagherzandi et al.'s multi-signature scheme [8]. In recent years, a larger number of two-round multi-signature schemes against concurrent attacks have been proposed, such as MulSig-DN [62], HMBS [10], MuSig2 [61] and MuSig2-H [75]. To further compress the storage overhead of multi-signatures, Maxwell et al. [60] proposed a Schnorr-based multi-signature scheme that supports public key aggregation.

Although computationally expensive, pairing-based multi-signatures avoid the requirement of communication among signers, and thus are more suitable for the setting of PoS blockchains. Based on Boneh-Lynn-Shacham (BLS) signature [23], Boldyreva [18] constructed a multi-signature scheme in gap Diffie-Hellman groups. Ristenpart and Yilek [66] extended Boldyreva's scheme to be free from rogue public-key attacks. Boneh et al. [21] introduced a BLS-based multi-signature scheme with public-key aggregation for blockchains. Another important research line of eliminating signing interactions is to restrict the signing setting to be synchronized [5, 43, 50]. That is, all signers share a global value each time they sign, such as the current time period.

All of the above multi-signature schemes (including our proposed schemes) are built upon traditional complexity assumptions, e.g., discrete logarithm assumptions, RSA assumptions, and pairings-based assumptions, and thus are vulnerable to quantum attacks [70]. Lattice-based cryptography is considered to be resilient against quantum attacks, and there has been a spark of interest in building multi-signature schemes from lattice in recent years. Those early lattice-based multi-signature schemes [9, 40, 41] follow the idea of Bellare and Neven's [12] multi-signature, and have three rounds of signing interaction. Damgård et al. [30] employed the structure of Drijvers et al.'s [34] scheme, and proposed a two-round lattice-based multi-signature. Boschini et al. [24] proposed a lattice-based multi-signature MuSig-L with single-round online phase and key aggregation. Chen [28] further put forward a lattice-based two-round multi-signature with smaller public keys and signatures than MuSig-L. Fleischhacker et al. [37, 38] constructed synchronized multi-signature schemes from lattices. We note that none of the above lattice-based multi-signature schemes consider forward security. By combining our techniques with the approach of achieving forward security in the setting of lattices [4, 36], it might be possible to extend them to be forward secure.

1.2.2 Forward-secure Signature Schemes

Anderson [6] first introduced the notion of forward security, which was originally studied in the context of key exchange, into the setting of digital signature, so as to mitigate the damage of signing key compromise. Bellare and Miner [11] later formally defined the primitive of forward-secure digital signature and presented the first concrete construction based on the hardness of factoring. Numerous subsequently proposed forward-secure signature schemes [3, 25, 53] optimize the previous one in terms of security and efficiency.

Given a maximum number T of total time periods, in order to reduce the computation cost or storage/communication overhead to $O(\log T)$, these constructions either follow the binary tree structure of hierarchical identity-based encryption [20], or use the pebbling techniques due to Itkis and Reyzin [53]. The idea of forward-secure signatures has been extended in several aspects, e.g., forward-secure signatures in untrusted update environments [57], tightly forward-secure signature [1], forward-secure threshold signature schemes [2], forward-secure identity/attribute-based signatures [80, 81], forward-secure aggregate signature schemes [59, 68, 79].

Early FS-MS schemes [59,72] either have $O(T)$-size public key or require an interactive signing protocol, and thus are not suitable for PoS blockchain consensus protocols. Drijvers et al. [35] recently presented a pairing-based forward-secure multi-signature scheme Pixel, which is specially optimized for use in PoS blockchains and achieves significant savings in terms of bandwidth overhead and verification cost. On the other hand, although combing tree-based forward-secure signature schemes [3, 11] with multi-signature schemes [19, 21] might also yield forward-secure multi-signature schemes, their bandwidth cost would scale linearly with T.

Green and Miers [45] pointed out that the original notion of forward security in the literature of public-key encryption is relatively blunt, and thus introduced the notion of puncturable public-key encryption that provides fine-grained forward security. Their work motivates various new constructions and extensions of puncturable public-key encryption [32, 71, 76, 77], and also has found many interesting applications in the design of other cryptographic primitives [33, 47, 78]. In the setting of digital signature, Bellare et al. [14] introduced the concept of puncturable signature, and provided a construction based on iO and one-way function. Halevi et al . [48] defined a puncturable signature scheme that needs to update signers' public keys repeatedly. Li et al. [56] put forth a puncturable signature scheme based on the idea of bloom filter encryption.

2 Technical Overview

This paper aims to build forward-secure multi-signature schemes under assumptions different from previous ones. Our first construction Pixel+ is based on the RSA assumption, and uses the RSA sequencer [51] to maintain a user's signing key. Our second construction Pixel++ is a paring-based one, and employs the bloom filter data structure to achieve efficient key update and fine-grained forward security.

Overview of Pixel+. The starting point of Pixel+ is Itkis and Reyzin [53]'s pebbling technique for constructing forward-secure signature schemes with fast signing. Specifically, given a maximum number T of time periods and an RSA modulus N, this technique enables a signer to successively compute the e_i-th root U^{1/e_i} of a group element $U \in \mathbb{Z}_N$ for $i = 1, \ldots, T$, and captures the forward security by erasing U^{1/e_i} at the end of the i-th time period. One of its major limitations is that the computation cost of generating a user's signing key is linear in T. The RSA sequencer [51], as the abstraction of the pebbling technique, overcomes this issue by additionally storing roots of a user's public key U of size $\log T$, and can immediately provide U^{1/e_i} at the time period i.

In more detail, given an RSA sequencer comprised of algorithms $(\mathsf{Setup}, \mathsf{Update}, \mathsf{Current}, \mathsf{Shift}, \mathsf{Program})$, we perform the Setup algorithm to produce an initial state $\mathsf{state}_{\mathsf{pp}}$ that contains a global public parameter $U = g^{\prod_{i\in[T]} e_i}$, where $g \in \mathbb{Z}_N$, $e_i = \mathsf{H}_{\mathsf{Primes}}(i)$ and $\mathsf{H}_{\mathsf{Primes}}(\cdot)$ is a hash function that maps an integer to a prime with a fixed size. Then, from $\mathsf{state}_{\mathsf{pp}}$ and a random integer $x \in \mathbb{Z}_N^*$, a user can run the Shift algorithm to efficiently compute his initial signing key sk_1 consisting of group elements of size $\log T$ and public key $pk = U^x$. In fact, sk_i, the secret key for the i-th time period, already contains the e_i-th root u_i of U^x that is used as the actual signing key throughout the time period i. Thus, the Current algorithm directly reads it from sk_i. To achieve forward security, the Update algorithm evolves sk_i to sk_{i+1} by erasing those auxiliary group elements that can be used to compute u_i, while generating new auxiliary group elements for computing subsequent roots.

The structure of RSA sequencer yields general forward-secure signature schemes. However, in the setting of multi-signature, it becomes challenging since we need to aggregate multiple signatures on the same message m into one. Luckily for us, we observe that signatures from different signers are generated at the same time period t. Therefore, our setting is similar to the synchronized setting of multi-signature [38, 50]. This enables us to derive a shared random number from the same message m and time period t. Roughly, for n signers indexed by public keys $\{pk_1, \ldots, pk_n\}$ and current signing keys $\{sk_{1,t}, \ldots, sk_{n,t}\}$, each signer first obtains the e_i-th root $u_{i,t}$ of pk_i, and then creates the signature $\sigma_{i,t} = u_{i,t}^{r\cdot a_{i,t}}$, where $a_{i,t} = \mathsf{H}_\lambda(pk_i, pk_1, \ldots, pk_n)$ and $r = \mathsf{H}_\lambda(m,t)$ that are computed using a hash function $\mathsf{H}_\lambda(\cdot)$. The shared r makes it possible to aggregate $\{\sigma_{1,t}, \ldots, \sigma_{n,t}\}$ into $\sigma_t = \prod_{i=1}^n \sigma_{i,t}$. The aggregated signature can be verified through checking $\sigma_t^{e_t} = apk^r$, where $apk = \prod_{i=1}^n pk_i^{\mathsf{H}_\lambda(pk_i, pk_1, \ldots, pk_n)}$ is the aggregated public key.

Note that Pixel+ only allows a user to sign once using the signing key of each time period. Such a restriction can be removed by combining the above construction with a general signature scheme [51]. However, it seems unnecessary in the context of PoS blockchain consensus, since each user only uses the current signing key to sign the block to be packaged, and then updates the secret key for the next block.

Overview of Pixel++. Our second construction Pixel++ aims to achieve more efficient key update and fine-grained forward security. To this end, we adopt the idea of Bloom filter encryption due to Deler et al. [32], and introduce the notion of puncturable multi-signature. Pixel++ builds upon

the BLS signature scheme in the setting of bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e)$, and uses a Bloom filter to manage all signing key materials.

A Bloom filter provides a succinct representation T of a set S, and can answer membership queries with a non-negligible false positive probability f. That is, for a membership query about an element s, if $s \in S$, it always answers "yes", and if $s \notin S$, it might also answer "yes" with probability f. Initially, the representation is assigned as $T = 0^\ell$. Adding an element s to the set S requires to compute k hash functions $\{H_j\}_{j\in[k]}$ on input s, and set corresponding positions of T to $T[H_j(s)] = 1$. Then, when querying the Bloom filter on an element, it outputs "yes" if it holds $T[H_j(s)] = 1$ for all $j \in [k]$.

In the construction of Pixel++, we associate each position l of T with a key component sk_l that binds the position l and a common secret key x. When a user intends to create a signature on a message m, he first chooses a valid key component $sk_{l'}$ from $\{sk_l | l \in \{H_j(m)\}_{j\in[k]}\}$, and then uses it to produce a BLS-style signature. After that, the signer adds m to Bloom filter, and also assigns $sk_l = \perp$ for each $l \in \{H_j(m)\}_{j\in[k]}$. As a result, even if the current signing key gets exposed, it cannot be used to create signatures for the previous message m. This captures the desirable forward security. The aggregation nature of BLS signature enables us to aggregate signatures on the same message from different users to one.

Since the key update (or puncture) in Pixel++ only involves erasure operations, it achieves more efficient key update than Pixel and Pixel+, which scale with $\log T$. Moreover, it evolves the signing key at the level of individual messages (vs. time periods in both Pixel and Pixel+), and thus also provides more practical forward security. Of course, these merits come at the cost of more storage overhead and signing error probability. As we discuss in Section 6.1, they are tolerated in the context of PoS blockchain consensus protocols.

3 Preliminaries

3.1 Hashing to Primes

In Pixel+, we need a hash function that maps an integer to a random prime with fixed size $\lambda + 1$. We make use of the hash function $\mathsf{H}_{\mathsf{Primes}} : [T] \to \{0,1\}^{\lambda+1}$ introduced in [49].

Specifically, $\mathsf{H}_{\mathsf{Primes}}$ maps an integer $x \in [T]$ to a random prime of size $\lambda + 1$ by rejection sampling. In more detail, it first chooses a PRF function $F : [T] \times [\lambda \cdot (\lambda^2 + \lambda)] \to \{0,1\}^\lambda$, and randomly picks a key k' for F. Then, it selects a random string $c \in \{0,1\}^\lambda$ and a default prime $e_{\text{default}} \in [2^\lambda, 2^{\lambda+1}]$, and assigns $k \leftarrow (k', c, e_{\text{default}})$. After that, an integer $x \in [T]$ is mapped to a prime as follows. For $i = 1$ to $\lambda \cdot (\lambda^2 + \lambda)$, compute $y_i = c \oplus F_{k'}(x, i)$. If $2^\lambda + y_i$ is a prime, then return it. Otherwise, increment i and repeat the above procedure. Eventually, if there does not exist an integer $i \in [\lambda \cdot (\lambda^2 + \lambda)]$ such that $2^\lambda + y_i$ is a prime, then return e_{default}. For each $x \in [T]$, we denote by $e_x = \mathsf{H}_{\mathsf{Primes}}(x)$.

3.2 RSA Sequencer

The notion of RSA sequencers was recently introduced by Hohenberger and Waters [51], as an abstraction of the pebbling technique. Informally, the goal of an RSA sequencer is to maintain and manage roots of a public key $U \in \mathbb{Z}_N$ such that given a specific integer t, it can immediately provide U^{1/e_t}, where $e_t = H(t)$ is an RSA exponent computed from t via a certain function $H(\cdot)$. It is formally defined as follows:

Definition 1 (RSA Sequencer [51])**.** *An RSA sequencer* RSASeq *is comprised of a 5-tuple of deterministic algorithms* $(\mathsf{Setup}, \mathsf{Update}, \mathsf{Current}, \mathsf{Shift}, \mathsf{Program})$ *that are specified as follows:*

- $\mathsf{Setup}(N, T, H(\cdot), \ell, (v_1, \ldots, v_\ell))$*: This algorithm takes as input an integer* $N \in \mathbb{Z}$*, a maximum number of allowed tags* T*, a function* $H(\cdot) : [T] \to \mathbb{Z}$*, an integer* $\ell \in \mathbb{Z}$ *and a* ℓ*-length vector* $(v_1, \ldots, v_\ell)$ *sampled from* $\mathbb{Z}_N^\ell$*. It outputs a state value* state.
- $\mathsf{Update}(\mathsf{state}, t)$*: This algorithm takes as input a state value* state *and an integer* $t \in [T]$*, and outputs an updated state value* state'.
- $\mathsf{Current}(\mathsf{state})$*: This algorithm takes as input a state value* state*, and outputs a* ℓ*-length vector* $(s_1, \ldots, s_\ell) \in \mathbb{Z}_N^\ell$*.*
- $\mathsf{Shift}(\mathsf{state}, (z_1, \ldots, z_\ell))$*: This algorithm takes as input a state value* state *and a vector* $(z_1, \ldots, z_\ell) \in \mathbb{Z}^\ell$*. It outputs another state value* state'.
- $\mathsf{Program}(N, T, H(\cdot), \ell, (v'_1, \ldots, v'_\ell), \mathsf{start})$*: On input of an integer* $N \in \mathbb{Z}$*, a maximum number of allowed tags* T*, a function* $H(\cdot) : [T] \to \mathbb{Z}$*, an integer* $\ell \in \mathbb{Z}$*, a* ℓ*-length vector* $(v_1, \ldots, v_\ell) \in \mathbb{Z}_N^\ell$ *and an integer* $\mathsf{start} \in [T]$*, this algorithm outputs a state value* state.

Update/Output Correctness. For arbitrarily selected parameters $N, T, \ell \in \mathbb{Z}, (v_1, \ldots, v_\ell) \in \mathbb{Z}_N^\ell$ and $H(\cdot) : [T] \to \mathbb{Z}$, assign $\mathsf{state}_1 = \mathsf{Setup}(N, T, H(\cdot), \ell, (v_1, \ldots, v_\ell))$ and $\mathsf{state}_t = \mathsf{Update}(\mathsf{state}_{t-1})$ for all $t \in [2, T]$. Then, for any $t \in [T]$, it must hold that

$$\mathsf{Current}(\mathsf{state}_t) = \left(v_1^{\prod_{i\in[T]\setminus\{t\}} e_i}, \ldots, v_\ell^{\prod_{i\in[T]\setminus\{t\}} e_i} \right),$$

where $e_t = H(t)$, and the arithmetic is conducted in $\mathbb{Z}_N$.

Shift Correctness. Let $\mathsf{state} = \mathsf{Setup}(N, T, H(\cdot), \ell, (v_1, \ldots, v_\ell))$ and $\mathsf{state}' = \mathsf{Setup}(N, T, H(\cdot), \ell, (v'_1, \ldots, v'_\ell))$, where $v'_j = v_j^{z_j}$ for $j \in [\ell]$. Then, it must hold that

$$\mathsf{state}' = \mathsf{Shift}(\mathsf{state}, (z_1, \ldots, z_\ell)).$$

Program Correctness. Let $\mathsf{state}_1 = \mathsf{Setup}(N, T, H(\cdot), \ell, (v_1, \ldots, v_\ell))$ and $\mathsf{state}_t = \mathsf{Update}(\mathsf{state}_{t-1})$ for $t \in [2, \mathsf{start}]$

and $\mathsf{start} \in [T+1]$. Furthermore, let $v_j^{\prod_{i\in[T]\setminus\{t\}} e_i}$ for $j \in [\ell]$, and $\mathsf{state}' = \mathsf{Program}(N, T, H(\cdot), \ell, (v'_1, \dots, v'_\ell), \mathsf{start})$. Then, it must hold that $\mathsf{state}_{\mathsf{start}} = \mathsf{state}'$.

3.3 Bloom Filter

The Bloom filter (BF) [17] is a widely used probabilistic data structure that supports membership query. Denote by $S = \{x_1, \dots, x_c\}$ a set of c elements from a universal set U (i.e., $S \subseteq U$). BF succinctly represents the set S using a ℓ-length binary string T, which is initialized as $T = 0^\ell$. To insert an element x, a group of k hash functions $\{H_j\}_{j\in[k]}$ are employed to randomly map x to k positions $\{H_j(x) \in [\ell]\}_{j\in[k]}$ of T. Then, the bits in these positions are all assigned to 1. Conversely, an element y is assumed to be a member of S if $T[H_j(y)] = 1$ holds for all $j \in [k]$. Formally, a Bloom filter is defined by the following algorithms.

Definition 2 (Bloom filter)**.** *A Bloom filter* BF *for a universal set U is comprised of three algorithms* $\mathsf{BF} = (\mathsf{Gen}, \mathsf{Update}, \mathsf{Check})$*, which are specified as follows.*

- $\mathsf{Gen}(\ell, k)$*: The generation algorithm takes as input two parameters $\ell, k \in \mathbb{Z}$. It first selects k independent hash functions $\{H_j\}_{j\in[k]}$, where $H_j : U \to [\ell]$. Then, it initializes $T = 0^\ell$, and output $(\{H_j\}_{j\in[k]}, T)$.*
- $\mathsf{Update}(\{H_j\}_{j\in[k]}, T, x)$*: The update algorithm takes as input the current state of the bloom filter $(\{H_j\}_{j\in[k]}, T)$ and an element $x \in U$. It first assigns $T' \leftarrow T$, and then sets $T'[H_j(x)] = 1$ for all $j \in [k]$. Finally, it returns T'.*
- $\mathsf{Check}(\{H_j\}_{j\in[k]}, T, y)$*: The check algorithm takes as input the current state of the bloom filter $(\{H_j\}_{j\in[k]}, T)$ and an element $y \in U$. If $T[H_j(y)] = 1$ holds for all $j \in [k]$, then it outputs 1 indicating that y has been inserted into* BF*, and 0 otherwise.*

The membership query based on the Bloom filter may suffer from false positive errors, that is, outputting 1 for an element y that has not been inserted into the Bloom filter. Given the maximum number c of elements allowed to be inserted into the set S, the false positive probability f can be arbitrarily tuned by adequately choosing the parameters ℓ and k. More precisely, the value of f can be calculated as $f \approx \left(1 - e^{-\frac{k\cdot c}{\ell}}\right)^k$.

4 Forward-Secure Multi-Signature from RSA

We first formally define the syntax and security notion of forward-secure multi-signature schemes, and then present the concrete construction of Pixel+, a forward-secure multi-signature scheme from RSA assumption.

4.1 Syntax and Security Definitions

We follow the syntax and security notion of forward-secure multi-signature scheme due to Drijvers et al. [35]. Formally, a forward-secure multi-signature scheme consists of the following algorithms:

$\mathsf{Setup}(1^\lambda, T) \to \mathsf{pp}$: Given a security parameter λ and the maximum number of time periods T, a third party runs the setup algorithm to generate the public parameter pp for all signers.

$\mathsf{KeyGen}(\mathsf{pp}) \to (pk, sk_1)$: Each signer performs the key generation algorithm on input the public parameter pp to produce a public verification key pk and an initial secret signing key sk_1.

$\mathsf{Update}(\mathsf{pp}, sk_t)$: At the end of each time period t, by running the key update algorithm, each signer updates its current secret key sk_t to sk_{t+1} used throughout the next time period $t+1$. After that, the previous secret key $sk_{i,t}$ is immediately erased.

$\mathsf{KeyAgg}(\mathsf{pp}, \{pk_1, \dots, pk_n\}) \to apk$: Given a set of public keys $\{pk_1, \dots, pk_n\}$, the key aggregation algorithm returns a single aggregate public key apk.

$\mathsf{Sign}(\mathsf{pp}, sk_t, \{pk_1, \dots, pk_n\}, m) \to \sigma_t$: All signers can independently sign a message m by each invoking the signing algorithm on input its current secret key sk_t and the set of all signers' pubic verification keys $\{pk_1, \dots, pk_n\}$.

$\mathsf{SigAgg}(\sigma_{1,t}, \dots, \sigma_{n,t}) \to \sigma_t$: Given signatures $\sigma_{1,t}, \dots, \sigma_{n,t}$ from different signers on the same message m and for the same time period t, anyone can run the signature aggregation algorithm to aggregate them into a single one σ_t.

$\mathsf{Verify}(\mathsf{pp}, apk, \sigma_t, m, t) \to b$: A verifier can check the validity of an aggregate signature σ_t on a message m and a time period t under an aggregate public key apk by calling the verification algorithm, which outputs 1 or 0 indicating that the signature is valid or invalid, respectively.

Correctness. The correctness of a forward-secure multi-signature scheme requires that, for any integer $n \in \mathbb{Z}$, for any message m and for any time period $t \in [T]$, if we have $(pk_i, sk_{i,1}) \leftarrow \mathsf{KeyGen}(pp)$ for $i \in [n]$, $apk \leftarrow \mathsf{KeyAgg}(\mathsf{pp}, \{pk_1, \dots, pk_n\})$, $sk_{i,j} \leftarrow \mathsf{Update}(\mathsf{pp}, sk_{i,j-1})$ for $i \in [n]$ and $j = 2, \dots, t$, $\sigma_{i,t} \leftarrow \mathsf{Sign}(\mathsf{pp}, sk_{i,t}, m)$ for $i \in [n]$ and $\sigma_t \leftarrow \mathsf{SigAgg}(\sigma_{1,t}, \dots, \sigma_{n,t})$, then it holds that

$$\Pr[\mathsf{Verify}(\mathsf{pp}, apk, \sigma_t, m, t) = 1] = 1.$$

Security. The unforgeability under chosen message attacks for a forward-secure multi-signature scheme is defined through the following security game consisting of three stages.

Setup. The challenger $\mathcal{C}$ first generates the public parameter $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T)$ and a public/secret key pair $(pk^*, sk_1^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, then sends pp and pk^* to the adversary $\mathcal{A}$.

Queries. The adversary $\mathcal{A}$ is allowed to adaptively issue the following queries.

- Secret key update: For such a query, when the current secret signing key is sk_t^* $(t < T)$, the challenger $\mathcal{C}$ updates it to sk_{t+1}^* for the next time period $t+1$.
- Signing: Given any message m and any set of public keys $\mathcal{PK} = \{pk_1, \ldots, pk_n\}$ with $pk^* \in \mathcal{PK}$, the challenger $\mathcal{C}$ uses the current secret key sk_t^* to generate a signature σ_t with respect to $(m, t, \mathcal{PK})$, and returns σ_t to $\mathcal{A}$.
- Corruption: At some point, the adversary $\mathcal{A}$ can issue a corruption query to obtain the current secret key sk_t^* held by the challenger $\mathcal{C}$. After this query, the adversary $\mathcal{A}$ is not allowed to issue any other queries. Here we record the corruption time as $\hat{t} \leftarrow t$.

Output. Finally, the adversary $\mathcal{A}$ outputs a multi-signature forgery tuple $(t^*, m^*, \sigma_{t^*}, \mathcal{PK}^*)$. The adversary $\mathcal{A}$ wins the security game provided that the following conditions are fulfilled:

- The adversary $\mathcal{A}$ made no signing queries on m^* during the time period t^*.
- $pk^* \in \mathcal{PK}^*$.
- $\hat{t} > t^*$.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{KeyAgg}(\mathsf{pp}, \mathcal{PK}^*), \sigma_{t^*}, m^*, t^*) = 1$.

Definition 3. *Denote by* $\mathsf{Adv}_{\mathcal{A}}^{\text{fu-cma}}(\lambda)$ *the probability of* $\mathcal{A}$ *winning the above security game. A forward-secure multi-signature scheme is said to be unforgeable under chosen message attacks provided that for any PPT adversary* $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\text{fu-cma}}(\lambda)$ *is negligible in the security parameter* λ.

4.2 The Pixel+ Construction

Pixel+ follows the framework of GQ signature [46], and builds on a particular instance of RSA sequencer that only produces one global RSA group. Given such an RSA sequencer $\mathsf{RSASeq} = (\mathsf{Setup}, \mathsf{Update}, \mathsf{Current}, \mathsf{Shift}, \mathsf{Program})$, Pixel+ works as follows.

$\mathsf{Setup}(1^\lambda, T)$: The setup algorithm takes as input a security parameter λ and the total number $T = 2^{\ell+1} - 2$ of time periods. It first chooses two safe primes $p = 2p' + 1$ and $q = 2q' + 1$, and assigns the RSA modulus $N = p \cdot q$ such that $2^\lambda < \varphi(N) < 2^{\lambda+1}$. Denote by QR_N the group of quadratic residues, and let g be its generator with order $p'q'$. Next, this algorithm defines two independent hash functions $\mathsf{H}_\lambda : \{0,1\}^* \rightarrow [0, 2^\lambda - 1]$ and $\mathsf{H}_{\mathsf{Primes}} : [T] \rightarrow \mathsf{Primes}(\lambda)$. Furthermore, it generates an initial state of the RSA sequencer $\mathsf{state}_{\mathsf{pp}} \leftarrow \mathsf{RSASeq.Setup}(N, T, \mathsf{H}_{\mathsf{Primes}}, g)$, and computes $U = g^{\prod_{i \in [T]} e_i} \mod N$, where $e_i = \mathsf{H}_{\mathsf{Primes}}(i)$. Finally, it outputs the public parameter as $\mathsf{pp} = \{T, N, U, \mathsf{H}_{\mathsf{Primes}}, \mathsf{H}_\lambda, \mathsf{state}_{\mathsf{pp}}\}$.

$\mathsf{KeyGen}(\mathsf{pp})$: Given the public parameter pp, each signer performs the key generation algorithm to produce his/her own public and secret key pair. For the i-th signer, this algorithm first picks a random integer $s_i \in [N]$, and then computes $\mathsf{state}_{i,1} \leftarrow \mathsf{RSASeq.Shift}(\mathsf{state}_{\mathsf{pp}}, s_i)$. Next, it publishes the public key as $pk_i = U^{s_i} \mod N$, and assigns the initial secret key as $sk_{i,1} = \{\mathsf{state}_{i,1}, e_1, 1\}$, where $e_1 = \mathsf{H}_{\mathsf{Primes}}(1)$.

$\mathsf{Update}(\mathsf{pp}, sk_{i,t}, t+1)$: For the i-th signer, the secret key update algorithm takes as input the public parameter pp, the current secret key $sk_{i,t}$ of the form $\{\mathsf{state}_{i,t}, e_t, t\}$ and the next time period $t+1$. It computes $\mathsf{state}_{i,t+1} \leftarrow \mathsf{RSASeq.Update}(\mathsf{state}_{i,t})$ and $e_{t+1} = \mathsf{H}_{\mathsf{Primes}}(t+1)$. It outputs the secret key for the next time period $t+1$ as $sk_{i,t+1} = \{\mathsf{state}_{i,t+1}, e_{t+1}, t+1\}$. At the same time, the previous secret key $sk_{i,t}$ is erased.

$\mathsf{KeyAgg}(\mathsf{pp}, \{pk_1, \ldots, pk_n\})$: Given the public parameter pp and public keys $pk_1, \ldots, pk_n$, the public key aggregation algorithm outputs the aggregated public key

$$apk = \prod_{i=1}^{n} pk_i^{\mathsf{H}_\lambda(pk_i, pk_1, \ldots, pk_n)} \mod N.$$

$\mathsf{Sign}(\mathsf{pp}, sk_{i,t}, \{pk_1, \ldots, pk_n\}, m)$: For the i-th signer, the signing algorithm takes as input the public parameter pp, the current secret key $sk_{i,t} = \{\mathsf{state}_{i,t}, e_t, t\}$, all signers' public keys $\{pk_1, \ldots, pk_n\}$ and the message m to be signed. It first lets $u_{i,t} \leftarrow \mathsf{RSASeq.Current}(\mathsf{state}_{i,t})$, and then computes $a_{i,t} = \mathsf{H}_\lambda(pk_i, pk_1, \ldots, pk_n)$ as well as $r = \mathsf{H}_\lambda(m, t)$. Next, it outputs the corresponding signature $\sigma_{i,t} = (u_{i,t})^{r \cdot a_{i,t}} \mod N$. Finally, it sends $\sigma_{i,t}$ to a designated combiner.

$\mathsf{SigAgg}(\sigma_{1,t}, \ldots, \sigma_{n,t})$: Given signatures $\sigma_{1,t}, \ldots, \sigma_{n,t}$ on the message m and time period t, the signature aggregation algorithm, which is run by the designated combiner, outputs the aggregated signature $\sigma_t = \prod_{i=1}^{n} \sigma_{i,t}$.

$\mathsf{Verify}(\mathsf{pp}, apk, \sigma_t, m, t)$: The verification algorithm takes as input the public parameter pp, the aggregated public key apk, the aggregated signature σ_t and the corresponding message m as well as time period t. It first computes the hash prime $e_t = \mathsf{H}_{\mathsf{Primes}}(t)$ and $r = \mathsf{H}_\lambda(m, t)$. Then, it outputs 1 to accept if

$$\sigma_t^{e_t} = apk^r,$$

and returns 0 otherwise.

Correctness. Given the initial state $\mathsf{state}_{\mathsf{pp}}$ of the RSA sequencer, we have that the i-th signer's initial secret key $sk_{i,1}$ is the same as the output of $\mathsf{RSASeq.Setup}(N, T, \mathsf{H}_{\mathsf{Primes}}, g^{s_i})$, due to the shift correctness of the RSA sequencer. Accordingly, his/her public key is assigned as

$$pk_i = U^{s_i} = g^{s_i \cdot \prod_{i \in [T]} e_i} \mod N.$$

After $t-1$ secret key updates, the i-th signer's secret key on the time period t is $sk_{i,t} = \{\mathsf{state}_{i,t}, e_t, t\}$, and it holds that

$$\mathsf{RSASeq.Current}(\mathsf{state}_t) = u_{i,t} = U^{s_i/e_t} = pk_i^{1/e_t} \mod N,$$

due to the output correctness of the RSA sequencer. Thus, for the message m and the time period t, the signature from the i-th signer is computed as

$$\sigma_{i,t} = (u_{i,t})^{r \cdot a_{i,t}} = pk_i^{\mathsf{H}_{\mathsf{Primes}}(m,t) \cdot \mathsf{H}_\lambda(pk_i, pk_1, \ldots, pk_n)/e_t} \mod N.$$

Furthermore, the aggregated signature is formed as

$$\sigma_t = \prod_{i=1}^{n} \sigma_{i,t} = \big(\prod_{i=1}^{n} pk_i^{\mathsf{H}_\lambda(pk_i, pk_1, \ldots, pk_n)}\big)^{\mathsf{H}_{\mathsf{Primes}}(m,t)/e_t}.$$

Consequently, during the verification procedure, we have that

$$\sigma_t^{e_t} = \big(\prod_{i=1}^{n} pk_i^{\mathsf{H}_\lambda(pk_i, pk_1, \ldots, pk_n)}\big)^{\mathsf{H}_{\mathsf{Primes}}(m,t)} = apk^r.$$

Security. The security of Pixel+ is guaranteed by the following theorem.

Theorem 1. *If F is a secure pseudorandom function and the RSA assumption holds, then the proposed forward-secure multi-signature scheme Pixel+ is unforgeable under chose message attacks in the random oracle model.*

The security proof uses the general forking lemma [8], and also is similar to that of Hohenberger and Waters' signature scheme. Due to space constraints, we refer the readers to the full version of this work for the proof.

Discussions. Note that, in the above construction, we need a trusted third party to run the global setup algorithm to generate a secure RSA modulus. In the case that the third party is untrusted, we can use the distributed manner of securely producing RSA modulus [27,29,39]. On the other hand, if the third party is believed to be fully trusted, then it also knows $\varphi(N)$. This enables it to compute $E = \prod_{i \in [T]} e_i \mod \varphi(N)$ and $U = g^E \mod N$, significantly reducing the computation cost of the setup algorithm. Moreover, observe that Pixel+ allows to sign only once during each time period, since the shared random r is completely determined by the corresponding message m and time period t. We can transform it to support multiple signatures per time period by using a common idea in the literature. That is, we use Pixel+ to sign a public key of a standard signature scheme.

5 Puncturable Multi-Signature from Pairing

We first formally define the syntax and security notion of puncturable multi-signature schemes, and then present our second construction Pixel++, a pairing-based forward-secure multi-signature scheme.

5.1 Syntax and Security Definitions

We follow the models of Ristenpart-Yilek's multi-signature scheme [66] and Derler et al.'s Bloom filter encryption scheme [32] to define the syntax and security notion of puncturable multi-signature. By using proofs of possession of secret keys, this combination prevents the puncturable multi-signature scheme from being vulnerable to the rogue public-key attack. Specifically, a puncturable multi-signature scheme consists of the following algorithms.

$\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, the setup algorithm generates the global public parameter pp. This algorithm may be run by a trusted third party through a distributed protocol. We assume that all the algorithms described below implicitly take pp as input.

$\mathsf{KeyGen}(\ell, k)$: On input parameters $\ell, k \in \mathbb{N}$ for the Bloom filter, each signer runs this key generation algorithm to generate a public key pk and a secret key sk as well as a proof of possession π.

$\mathsf{KeyVerify}(pk, \pi)$: On input a public key pk and the corresponding proof of possession π, the public key verification algorithm outputs 1 if pk is valid under the proof π and 0 otherwise.

$\mathsf{Punc}(sk, m)$: On input a secret key sk and a message m, the secret key puncturing algorithm outputs an updated (a.k.a. punctured) secret key sk'. After that, we also say that sk' has been punctured with m.

$\mathsf{Sign}(sk, m)$: On input a secret key sk and a message m, a signer runs this signing algorithm to output a signature σ, and further sends it to a designated combiner.

$\mathsf{KeyAgg}(\mathcal{PK})$: On input a set of public keys $\mathcal{PK} = \{pk_1, \ldots, pk_n\}$, the public key aggregation algorithm outputs an aggregate public key apk, or $\perp$ to indicate a fail of aggregating these public keys.

$\mathsf{SigAgg}(\sigma_1, \ldots, \sigma_n)$: On input signatures $\sigma_1, \ldots, \sigma_n$ on the same message m from different signers, the designated combiner runs this signature aggregation algorithm to output an aggregate signature σ.

$\mathsf{Verify}(apk, \sigma, m)$: On input an aggregate public key apk, an aggregate signature σ and a message m, the verification algorithm outputs 1 to indicate that all singers in apk correctly signed the message m, and 0 otherwise.

Correctness. Intuitively, the correctness of puncturable multi-signature scheme requires that, if the secret key has not been punctured with a message m, then the probability of correctly signing m is bounded by some non-negligible function $\mu(\ell,k)$ in parameters ℓ,k of the Bloom filter. On the other hand, if the secret key has been punctured with a message m, then the signing would definitely fail. Formally, for any $\lambda,\ell,k,n \in \mathbb{N}$, any message m, any $(pk_i,\pi_i,sk_i) \leftarrow \mathsf{KeyGen}(\ell,k)$ and any signature $\sigma_i \leftarrow \mathsf{Sign}(sk_i',m)$, it holds that

$$\Pr[\mathsf{KeyVerify}(pk_i,\pi_i)=1]=1, \Pr[\mathsf{Verify}(apk,\sigma,m)=1]=1,$$

where $i \in [n]$ and $apk = \mathsf{KeyAgg}(\{pk_1,\ldots,pk_n\})$ as well as $\sigma = \mathsf{SigAgg}(\sigma_1,\ldots,\sigma_n)$. Moreover, for any arbitrary interleaved sequence of invocations of $sk_i' \leftarrow \mathsf{Punc}(sk_i,m')$ for $m' \neq m$, it holds that

$$\Pr[\mathsf{Verify}(apk,\sigma,m)=1] \geq 1-\mu(\ell,k).$$

Remark 1. *As suggested by Li et al. [56], we can split the message into several blocks $m = m_1||\ldots||m_b$, and choose to run the puncturing algorithm on any block m_i according to the concrete scenarios of specific applications. For example, in the setting of PoS blockchains, we can think of the block number as part of a block message and conduct a puncture operation on the block number. This enables the signer to create signatures on the same block message at different blocks while capturing fine-grained forward security.*

Security. The definition of unforgeability under chosen message attacks (fu-cma) for a puncturable multi-signature scheme is similar to that described in Section 4.1. Here we emphasize that the fu-cma definition implicitly includes forward security. Essentially, this is achieved by allowing the adversary to obtain the signing key that has been punctured with the challenge message m^*. Specifically, it is captured through the following security experiment consisting of three stages.

Setup. The challenger $\mathcal{C}$ first generates the global public parameter $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. Then, it runs the key generation algorithm $\mathsf{KeyGen}(\ell,l) \rightarrow (pk^*,\pi^*,sk^*)$, and initializes two sets $\mathcal{S} \leftarrow \emptyset$ and $\mathcal{P} \leftarrow \emptyset$. Finally, it forwards the public parameter pp and the public key pk^* as well as the proof of possession π^* to the adversary $\mathcal{A}$.

Queries. The adversary $\mathcal{A}$ is allowed to adaptively issue the following queries.

- Secrete key puncturing: Given a message m, the challenger $\mathcal{C}$ punctures the secret key sk^* by running the algorithm $\mathsf{Punc}(sk^*,m) \rightarrow sk^{*\prime}$. After that, the challenger $\mathcal{C}$ updates the set $\mathcal{P} \leftarrow \mathcal{P} \cup \{m\}$.
- Signing: Given a message m, the challenger $\mathcal{C}$ runs the algorithm $\mathsf{Sign}(sk^*,m) \rightarrow \sigma$, and returns σ to the adversary $\mathcal{A}$. It also updates the set $\mathcal{S} \leftarrow \mathcal{S} \cup \{m\}$.
- Corruption: At some point, the adversary $\mathcal{A}$ can issue a corruption query to obtain the current secret key sk^* held by the challenger $\mathcal{C}$. After this query, the adversary $\mathcal{A}$ is not allowed to issue any other queries.

Output. Finally, the adversary $\mathcal{A}$ outputs a multi-signature forgery tuple $(m^*,\sigma^*,\mathcal{PK}^*)$ such that $pk^* \in \mathcal{PK}^*$. Denote by $\mathcal{PK}^* = \{pk_1,\ldots,pk_n\}$, and let the corresponding proofs of possession be $\{\pi_1,\ldots,\pi_n\}$. The adversary $\mathcal{A}$ wins the security game provided that the following conditions are fulfilled:

- For each $i \in [n]$, $\mathsf{KeyVerify}(pk_i,\pi_i)=1$.
- $\mathcal{A}$ made no signing queries on m^*, i.e., $m^* \notin \mathcal{S}$.
- If $\mathcal{A}$ made the corruption query, then it must be $m^* \in \mathcal{P}$.
- $\mathsf{Verify}(\mathsf{KeyAgg}(\mathcal{PK}^*),\sigma^*,m^*)=1$.

Definition 4. *Denote by $\mathsf{Adv}^{\text{fu-cma}}_{\mathsf{PMS},\mathcal{A}}(\lambda)$ the probability of $\mathcal{A}$ winning the above security experiment. A puncturable multi-signature scheme is said to be unforgeable under chosen message attacks provided that for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\text{fu-cma}}_{\mathsf{PMS},\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.*

5.2 The Pixel++ Construction

We build Pixel++ upon the BLS signature scheme, and employ a Bloom filter to maintain signing key components. Given a bloom filter $\mathsf{BF} = (\mathsf{Gen},\mathsf{Update},\mathsf{Check})$, the details of Pixel++ are specified as follows.

$\mathsf{Setup}(1^\lambda)$: This algorithm first generates Type-3 bilinear groups $(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,g_1,g_2,p,e) \leftarrow \mathsf{BilGen}(\lambda)$, and then picks three independent hash functions $\mathsf{H}_1 : \mathbb{N} \rightarrow \mathbb{Z}_p^*$, $\mathsf{H}_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $\mathsf{H}_3 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$. Furthermore, it chooses random group elements $h,h_0,h_1,h_2 \in \mathbb{G}_1$. We assume that all the algorithms described below implicitly take the above global public parameters as input.

$\mathsf{KeyGen}(\ell,k)$: For the i-th user, this algorithm first initializes a Bloom filter instance $(\{H_j\}_{j\in[k]},T) \leftarrow \mathsf{BF.Gen}(\ell,k)$. Then, it picks a random integers $x_i \in \mathbb{Z}_p^*$, and assigns the public key as $pk_i = g_2^{x_i}$. Furthermore, for each $l \in [\ell]$, it chooses a random integer $r_{i,l} \in \mathbb{Z}_p$, and computes the secret key $sk_i = \{T,\{(sk_{i,l,0},sk_{i,l,1},sk_{i,l,2})\}_{l\in[\ell]}\}$, where

$$sk_{i,l,0} = g_2^{r_{i,l}},\ sk_{i,l,1} = h_2^{r_{i,l}},$$
$$sk_{i,l,2} = h^{x_i} \cdot \left(h_0 \cdot h_1^{a_l}\right)^{r_{i,l}},\ a_l = \mathsf{H}_1(l).$$

In addition, this algorithm computes a proof of possession $\pi_i = \mathsf{H}_2(pk_i)^{x_i}$.

$\mathsf{KeyVerify}(pk_i,\pi_i)$: This algorithm outputs 1 validating the proof of possession if it holds that

$$e(\mathsf{H}_2(pk_i),pk_i) = e(\pi_i,g_2).$$

$\mathsf{Punc}(sk_i, m)$: Given a message m and a secret key $sk_i = \{T, \{(sk_{i,l,0}, sk_{i,l,1}, sk_{i,l,2})\}_{l\in[\ell]}\}$, this algorithm first computes an array $T' = \mathsf{BF.Update}(\{H_j\}_{j\in[k]}, T, m)$. Then, for each $l \in [\ell]$, it defines

$$(sk'_{i,l,0}, sk'_{i,l,0}, sk'_{i,l,2}) = \begin{cases} (sk_{i,l,0}, sk_{i,l,1}, sk_{i,l,2}), & T'[l] = 0 \\ \perp, & T'[l] = 1 \end{cases}$$

where $T'[l]$ is the l-th bit of T'. Finally, this algorithm outputs $sk'_i = \{T, \{(sk'_{i,l,0}, sk'_{i,l,0}, sk'_{i,l,2})\}_{l\in[\ell]}\}$.

$\mathsf{Sign}(sk_i, m)$: Given a message m with the prefix m' and a secret key $sk_i = \{T, \{(sk_{i,l,0}, sk_{i,l,0}, sk_{i,l,2})\}_{l\in[\ell]}\}$, this algorithm first checks whether $\mathsf{BF.Check}(\{H_j\}_{j\in[k]}, T, m) = 1$ and outputs $\perp$ in this case. Otherwise, it picks the smallest index $l' \in [\ell]$ such that $(sk_{i,l',0}, sk_{i,l',0}, sk_{i,l',2}) \neq \perp$. Then, it chooses a random integer $r'_i \in \mathbb{Z}_p$, and computes

$$\sigma_{i,1} = sk_{i,l',2} \cdot (sk_{i,l',1})^{\mathsf{H}_3(m)} \cdot (h_0 h_1^{a_{l'}} h_2^{\mathsf{H}_3(m)})^{r'_i},$$
$$\sigma_{i,2} = sk_{i,l',0} \cdot g_2^{r'_i},$$

where $a_{l'} = \mathsf{H}_1(l')$. Finally, it sends $(\sigma_{i,1}, \sigma_{i,2}, l')$ to a designated combiner[3].

$\mathsf{KeyAgg}(\mathcal{PK})$: Given a set of public keys $\mathcal{PK} = \{pk_1, \ldots, pk_n\}$, if $\mathsf{KeyVerify}(pk_i, \pi_i) = 1$ for each index $i \in [n]$, then this algorithm outputs an aggregate public key $apk = \prod_{i=1}^n pk_i$. Otherwise, it outputs $\perp$.

$\mathsf{SigAgg}(\sigma_1, \ldots, \sigma_n)$: Given n signatures $\sigma_1 = (\sigma_{1,1}, \sigma_{1,2}, l')$, $\ldots$, $\sigma_n = (\sigma_{n,1}, \sigma_{n,2}, l')$ on the same message m from independent signers, this algorithm computes the aggregate signature $\sigma = (\sigma_1, \sigma_2, l')$, where

$$\sigma_1 = \prod_{i=1}^n \sigma_{i,1}, \quad \sigma_2 = \prod_{i=1}^n \sigma_{i,2}.$$

$\mathsf{Verify}(apk, \sigma, m)$: Given the aggregate public key and signature $\sigma = (\sigma_1, \sigma_2, l')$ on the message m, this algorithm outputs 1 if and only if $apk \neq \perp$ and

$$e(\sigma_1, g_2) = e(h, apk) \cdot e(h_0 \cdot h_1^{\mathsf{H}_1(l')} \cdot h_2^{\mathsf{H}_3(m)}, \sigma_2).$$

Correctness. Given public keys $\mathcal{PK} = \{pk_1, \ldots, pk_n\}$ and signatures $\sigma_1 = (\sigma_{1,1}, \sigma_{1,2}, l'), \ldots, \sigma_n = (\sigma_{n,1}, \sigma_{n,2}, l')$ on the same message m under these public keys, the aggregate public key apk and $\sigma = (\sigma_1, \sigma_2, l')$ are the forms of

$$\begin{aligned} apk &= \prod_{i=1}^n pk_i = \prod_{i=1}^n g_2^{x_i} = g_2^{\sum_{i=1}^n x_i}, \\ \sigma_1 &= \prod_{i=1}^n \sigma_{i,1} = \prod_{i=1}^n sk_{i,l',2} \cdot (sk_{i,l',1})^{\mathsf{H}_3(m)} \cdot (h_0 h_1^{a_{l'}} h_2^{\mathsf{H}_3(m)})^{r'_i} \\ &= h^{\sum_{i=1}^n x_i} \cdot (h_0 \cdot h_1^{a_{l'}} \cdot h_2^{\mathsf{H}_3(m)})^{\sum_{i=1}^n (r_{i,l'} + r'_i)}, \\ \sigma_2 &= \prod_{i=1}^n \sigma_{i,2} = \prod_{i=1}^n sk_{i,l',0} \cdot g_2^{r'_i} = g_2^{\sum_{i=1}^n (r_{i,l'} + r'_i)}, \end{aligned}$$

where $a_{l'} = \mathsf{H}_1(l')$.

Then, the left-hand of the verification equation holds

$$\begin{aligned} e(\sigma_1, g_2) &= e\left(h^{\sum_{i=1}^n x_i} \cdot (h_0 \cdot h_1^{a_{l'}} \cdot h_2^{\mathsf{H}_3(m)})^{\sum_{i=1}^n (r_{i,l'} + r'_i)}, g_2\right) \\ &= e\left(h^{\sum_{i=1}^n x_i}, g_2\right) \cdot e\left((h_0 \cdot h_1^{a_{l'}} \cdot h_2^{\mathsf{H}_3(m)})^{\sum_{i=1}^n (r_{i,l'} + r'_i)}, g_2\right) \\ &= e\left(h, g_2^{\sum_{i=1}^n x_i}\right) \cdot e\left(h_0 \cdot h_1^{a_{l'}} \cdot h_2^{\mathsf{H}_3(m)}, g_2^{\sum_{i=1}^n (r_{i,l'} + r'_i)}\right) \\ &= e(h, apk) \cdot e(h_0 \cdot h_1^{a_{l'}} \cdot h_2^{\mathsf{H}_3(m)}, \sigma_2). \end{aligned}$$

During the above signature verification process, we assume that the message m has not been punctured. This means that the secret key component $sk_{i,l'}$ has not been deleted, and enables the signer to correctly generate the signature σ_i. On the other hand, if $\mathsf{BF.Check}(\{H_j\}_{j\in[k]}, T, m) = 1$, then signing the message m fails.

Security. The security of Pixel++ is guaranteed by the following theorem.

Theorem 2. *If the 2-BDHE$_3^*$ assumption holds, then the proposed puncturable multi-signature scheme is unforgeable under chosen message attacks. Formally, given an adversary $\mathcal{A}$ against the unforgeability of the proposed puncturable multi-signature scheme, we can construct an algorithm $\mathcal{C}$ against the 2-BDHE$_3^*$ problem such that*

$$\mathsf{Adv}_{\mathcal{C}}^{\text{2-BDHE}_3^*}(\lambda) \geq \frac{1}{\ell \cdot q_{\mathsf{H}_3}} \cdot \mathsf{Adv}_{\mathsf{PMS},\mathcal{A}}^{\text{fu-cma}}(\lambda),$$

where q_{H_3} is the number of $\mathcal{A}$'s queries to random oracle H_3, and ℓ is the length of the binary string in the Bloom filter.

Due to space constraints, we refer interested readers to the full version of this work for the proof.

Discussions. The key update procedure in Pixel++ only involves deletion operations, and thus is much more efficient than Pixel and Pixel+. The cost of this gain is that the signing may fail with a non-negligible probability. However, as we discuss in the next section, this is tolerable for PoS blockchain consensus protocols. Moreover, when the number of puncture operations reaches the upper bound of the Bloom filter's capacity, the false-positive probability (i.e., the probability of failing to sign) would exceed an acceptable bound. At

[3] To ensure l' is the same for all signers, we require every user to puncture his key after generating a new block, even if he does not participate in the packaging process.

this point, to maintain such a bound, we have to generate a new pair of public and secret keys. As suggested by Gree and Miers [45] and Derler et al. [32], we can avoid this by combining puncturable multi-signature with traditional FS-MS (e.g., Pixel). More precisely, we split the lifetime of the system into time periods, during each of which we can conduct puncture operations as in Pixel++. At the end of each time period, we can update the current secret key to a new unpunctured one for the next time period, as in Pixel. This ensures the signing error probability never exceeds the acceptable bound, while avoiding the requirement of generating new public keys.

6 Applications

In this section, we mainly demonstrate how to apply Pixel+ and Pixel++ to the setting of PoS blockchains that use forward-secure signature schemes to mitigate the damage of posterior corruptions. Of course, we can also use our schemes to improve the performance and security of those systems based on multi-signature schemes, such as improving the scalability of Bitcoin [60] and distributed randomness protocols [73].

6.1 Background on PoS Blockchains

How to securely and efficiently achieve a consensus is a basic design goal of blockchain systems. In PoS blockchains [15, 16, 44, 54, 55], each user (a.k.a stakeholder) possesses a certain amount of stakes or tokens that determine his/her voting power in the consensus procedure. In general, this is measured via the fraction of the user's stakes out of the total ones. The time is divided into a sequence of units called slots $\{sl_1, sl_2, \ldots, sl_n\}$. In each slot, all users have a probability approximately proportional to their relative stakes to be assigned as a leader. Such leaders, organized as a committee, are permitted to collectively generate a new block.

A block generated at a slot sl_j is structured in the form of $B_j = (sl_j, st_j, d_j, \sigma_j)$, where $st_j \in \{0,1\}^\lambda$ is the state of the previous block B_{j-1}, $d_j \in \{0,1\}^*$ is the packaged transaction data, and σ_j is a signature on (sl_j, st_j, d_j) produced by the committee. Then, the final blockchain is organized as a sequence of blocks $\{B_1, B_2, \ldots, B_n\}$, which strictly corresponds to the increasing sequence of slots $\{sl_1, sl_2, \ldots, sl_n\}$. During the generation of a block in each slot, most PoS protocols run a Byzantine fault-tolerant sub-protocol to tolerate malicious users. It is usually assumed that the majority (greater than $2/3$) of stakes involved in each consensus process is held by honest users. We say that each block is valid provided that a majority of committee members approved it.

6.2 Applications to PoS blockchains

. As we have discussed earlier, most of the existing PoS protocols suffer from the issue of posterior corruptions. In PoS protocols Ouroboros Praos [31] and Ouroboros Genesis [7] as well as Thunderella [64], to resist posterior corruption attacks, the signature on each block is generated with traditional forward-secure signature schemes, which mainly build on hierarchical identity-based encryption. We can apply Piexl+ or Piexl++ in these PoS protocols to achieve the same security goal, while possibly reducing the storage overhead and communication cost in some special scenarios where many users broadcast a large number of signatures on the same message. In addition, there are a few PoS protocols like Snow White [16] and Ouroboros [54] that use ordinary signature schemes to sign each block. Therefore, Piexl+ and Piexl++ can also be deployed in these PoS protocols to enable them to resist posterior corruption attacks. Below we provide the details of integrating our proposals into general PoS blockchains.

Registering public keys. When a user intends to participate in the consensus, he needs to register the public key of his voting keys. To this end, a user runs the Setup algorithm of Pixel+ or Pixel++ to generate a key pair. Then, the user issues a key registration transaction that includes the public key and the corresponding proof of possession. Furthermore, those verifiers participating in the Byzantine fault-tolerant protocol check the validity of both the key registration transaction and the proof of possession. If the check passes, the newly generated voting keys are added to the user's account. After this, if the user is eligible to participate in the consensus protocol, he employs either Pixel+ or Pixel++ to sign blocks.

Voting on blocks. The stake ownership is generally shown through the possession of corresponding signing keys. So, the number of signing keys is proportional to the stake. Each committee member participating in the consensus is required to vote on a new block B_j to be generated. This is conducted by letting participants sign blocks with their voting keys and propagating signatures to the blockchain network. The voting details are slightly different when we use Pixel+ or Pixel++, respectively.

In the case of Pixel+, we treat the block number j as the current time period. Thus, participants first need to update their secret keys to the current time period by repeatedly invoking the update algorithm in sequence. Then, using the current voting keys, they independently perform the signing algorithm to produce corresponding signatures on the same block B_j. As in Pixel+, the block number j is necessary for checking B_j's validity.

Recall that in Pixel++, each user can produce a correct signature on a message only if the secret key has not been punctured with this message. Therefore, when we use Pixel++ to authenticate the voting, each participant should make sure that his voting key has not been punctured with the block to be signed. Furthermore, during the voting process, each participant freely chooses a key component corresponding to a position l of the Bloom filter's string T such that $T[l] = 0$, and then directly runs the signing algorithm to generate a vote on the block B_j.

Particularly, Pixel++ has a non-negligible probability of signing failure due to the use of the Bloom filter. That is, it might be possible that even if the user's secret key has not been punctured with the block to be signed, signing the block still yields a failure symbol $\perp$. This failure probability can be as small as possible by adjusting the parameters of the Bloom filter, which implies a trade-off between the failure probability and the cost of generating and storing the signing key. Moreover, for those committee-based PoS consensus protocols, we can exclude participants who fail to sign. In fact, in exchange for the vast efficiency gain, most PoS blockchain protocols like Ouroboros and Snow White tolerate a non-negligible signing error.

Propagating and aggregating signatures. After the voting is done, the corresponding signature $\sigma_{i,j}$ on the block B_j from the i-th committee member would be broadcast through the blockchain network. When a designated aggregation node receives N signatures $\{\sigma_{1,j},\ldots,\sigma_{N,j}\}$ on the same block B_j, where N is a predefined threshold, it terminates the propagation, and invokes the aggregation algorithm to aggregate all signatures into a single multi-signature σ_j. Then, after σ_j is verified to be correct, B_j is written into the blockchain.

Note that the aggregation process in both Pixel+ and Pixel++ is non-interactive and incremental. This means that, on the one hand, those signatures propagated through the network can be aggregated by any party without interacting with the corresponding committee members (i.e., those original signers). On the other hand, it allows propagating nodes to just broadcast the aggregation result of several committee members' signatures, since adding a new signature to the current multi-signature immediately yields a new multi-signature. These two features bring huge communication overhead benefits.

Updating or puncturing secret keys. To achieve forward security, the voting key should be immediately updated or punctured after each new block is signed. In more detail, when we use Pixel+ in PoS blockchains, its entire lifetime is divided into T time periods, each of which corresponds to a block number in a natural order. During each time period t, the i-th participant's voting key $sk_{i,t}$ can only be used to sign the corresponding block B_t once. Although this is an inherent limitation of Pixel+, it just fits the actual application scenario of the blockchain. After that, the current voting key $sk_{i,t}$ is updated to the next time period $sk_{i,t+1}$, while $sk_{i,t}$ is erased securely [65]. For those users who have not participated in the consensus, they should first update their keys to the current time period before voting on a new block.

In the case of using Pixel++ in PoS blockchains, the forward security is captured by letting each participant puncture his voting key with the signed block. Since Pixel++ enables a user to conduct puncture operations on any position of a message, and each block B_j is uniquely identified by the corresponding slot number sl_j, so we can succinctly run the puncture algorithm with the slot number, instead of the entire block. Pixel++ allows a user to participate in the consensus on-the-fly. That is, he can directly participate in a new consensus without updating his voting key, even if he has not been elected to the committee for a long time.

7 Performance Evaluation

As a proof-of-concept, we implement Pixel+ and Pixel++ using Python[4], and conduct several representative experiments.

Experimental setup. For Pixel+, we use a 3072-bit RSA modulus N that provides 128-bit security. The hash function $\mathsf{H}_{\mathsf{Primes}}(\cdot)$ maps an integer to a 128-bit prime, and is implemented with SHA256 and Rabin-Miller primality test. We consider the maximum number of time periods for $T \in \{2^{16},2^{20},2^{20},2^{24},2^{28},2^{32}\}$. As an exemplary scenario, for $T = 2^{28}$, if the generation of each block takes 10 seconds, then the lifetime of the public key is around 85 years.

For Pixel++, we use the BLS12-381 curve that also provides 128-bit security, and builds its implementation upon Python implementation of BLS12-381[5]. The underlying Bloom filter is implemented with Python package `pybloom`, and takes into account three false positive probability $f \in \{0.1, 0.01, 0.001\}$ and capacity $c \in \{100, 1000, 10000\}$. The assignments of f and c also determine the length ℓ of the binary string of the Bloom filter and the number k of hash functions.

When comparing Pixel+ and Pixel++ with Pixel [35], we use Pixel's Python implementation [6]. All experiments are conducted on a PC with 3.00GHz Intel Core i7-9700 CPU and 40.0 GB DDR3 memory.

Table 2: Comparison of the cost of generating and aggregating public key and signature.

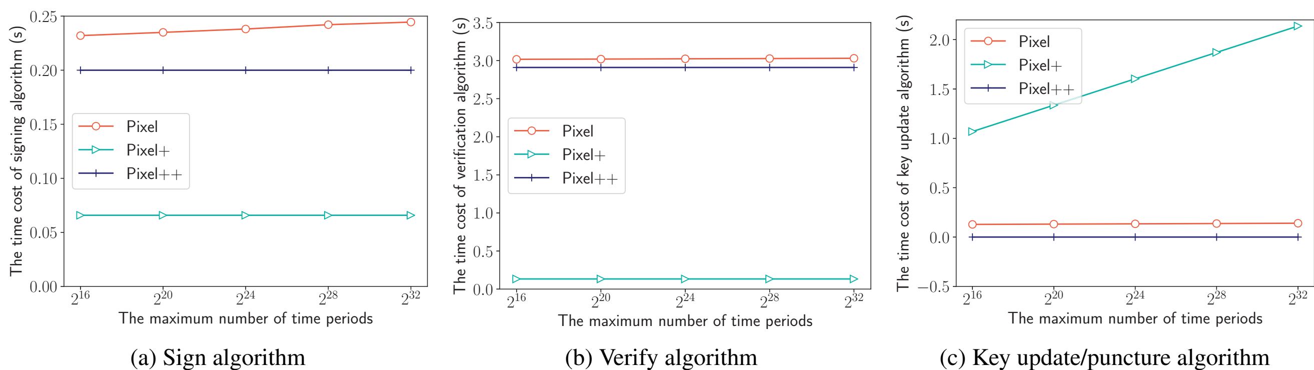
Scheme	Sig. size	pk size	SigAgg	KeyAgg
Pixel [35]	0.92 KB	0.62 KB	47.41 ms	42.25 ms
Pixel+ (Section 4.2)	0.90 KB	0.90 KB	0.42 ms	523.42 ms
Pixel++ (Section 5.2)	0.92 KB	0.62 KB	46.84 ms	41.46 ms

Storage cost. We evaluate the space complexity of these schemes in terms of the sizes of signature, public key and secret key, and obtain their specific values by storing the corresponding data in a text file. From Table 2 we can see that our constructions are almost as efficient as previous schemes from the perspective of blockchain data storage overhead. Figure 2 indicates that Pixel+ has smaller space complexity on the user side than Pixel. In the setting of $T = 2^{32}$, it is less than 60 KB. As indicated in Table 3, at the cost of achieving fine-grained forward security and more efficient key update,

[4]https://github.com/Crypto4hub/Pixel-signatures
[5]https://github.com/kwantam/bls_sigs_ref
[6]https://github.com/algorand/pixel.

Table 3: Performance evaluation of Pixel++ under different parameter settings of Bloom filter

f	c	ℓ	k	KeyGen	Sign	Verify	KeyAgg	SigAgg	Puncture	$\lvert pk \rvert$	$\lvert \sigma \rvert$	$\lvert sk \rvert$
0.1	100	480	4	1.57 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	0.58 MB
	1000	4796	4	15.51 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	5.78 MB
	10000	47928	4	154.45 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	57.73 MB
0.01	100	959	7	3.08 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	1.16 MB
	1000	9590	7	31.45 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	11.55 MB
	10000	95851	7	311.63 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	115.46 MB
0.001	100	1440	10	4.65 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	1.73 MB
	1000	14380	10	46.15 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	17.32 MB
	10000	143780	10	1038.24 min	0.20 s	2.91 s	41.46 ms	46.84 ms	0.05 ms	0.62 KB	0.92 KB	173.20 MB



(a) Sign algorithm (b) Verify algorithm (c) Key update/puncture algorithm

Figure 1: The time cost of main algorithms in Pixel [35] and our Pixel+ and Pixel++ under different numbers of time periods.

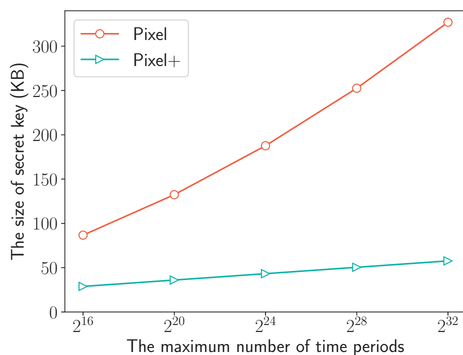


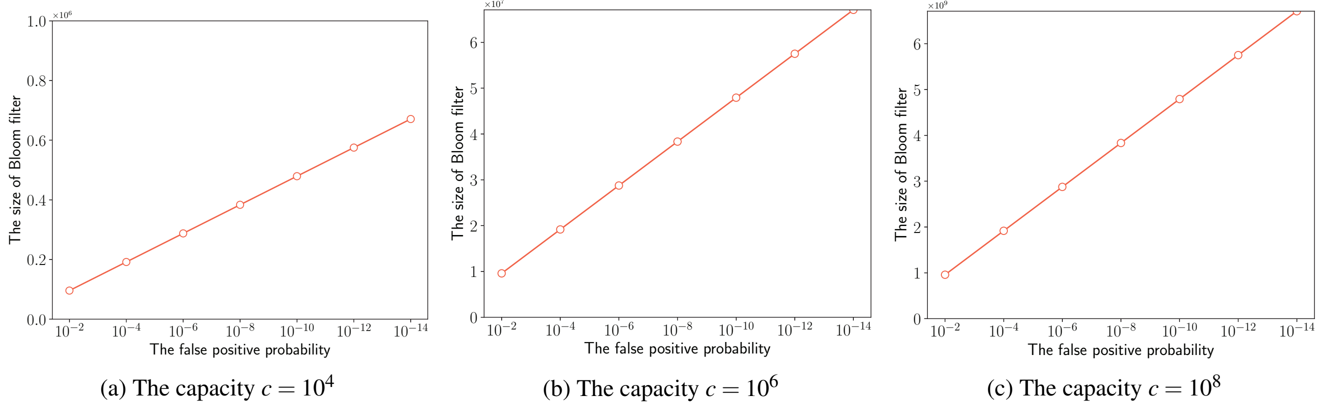
Figure 2: The secret key size under different maximum numbers of time periods.

the secret key size of Pixel++ is several orders of magnitude more expensive than the other two schemes.

Computation cost. In Figure 1 we provide the timing results of main algorithms (in the case of a single signer or verifier) in Pixel and Pixel+ as well as Pixel++. Observe that the signing and verification algorithms of Pixel+ are more efficient than that of the other two, and only take 0.07 s and 0.13 s, respectively. Pixel++ achieves the most efficient key update, and only takes 0.05 ms which is 1000x faster than Pixel. Also, note that the time cost of these three algorithms in Pixel time cost is slightly dependent on the maximum number of time periods. Moreover, as shown in Table 2, Pixel++ and Pixel are almost as efficient in generating and aggregating public keys and signatures.

Signing capacity of Pixel+ and Pixel++. The parameters T and c state the maximum number of messages that can be signed with Pixel+ and Pixel++, respectively. In our experiments, we use a practical assignment of $T = 2^{32}$ and assign the largest $c = 10000$ that is much less than T. This is mainly because the cost of KeyGen in Pixel++ is roughly linear in c, while Pixel+'s cost is logarithmic in T. As shown in Table 3, given $f = 0.001$ and $n = 1000$, the running time of KeyGen is roughly 17 hours. Thus, setting $c = 2^{32}$ is unpractical. On the other hand, a small c is indeed not practical. Fortunately, as suggested by Green and Miers [45] and Derler et al. [32], we can increase Pixel++'s signing capacity by combining it with a traditional FS-MS scheme, without sacrificing fine-grained forward security. Of course, this will additionally bring logarithmic computation cost and storage overhead.

The impact of Bloom filter parameters on Pixel++. The performance of Pixel++ partially depends on the parameters of the underlying Bloom filter. More precisely, the failure probability of signing (i.e., false positive probability) can be roughly computed as $f \approx \left(1 - e^{-\frac{kn}{\ell}}\right)^k$, where ℓ is the size



(a) The capacity $c = 10^4$ (b) The capacity $c = 10^6$ (c) The capacity $c = 10^8$

Figure 3: The relationship between the false-positive probability and the size of Bloom filter.

of the bloom filter, k is the number of hash functions, c is the maximum number of elements that tolerates the failure probability of signing f. As indicated in Table 1 and Table 3, the parameters (f, c, ℓ, k) of the Bloom filter *do not* affect Pixel++'s signature size, public-key size and the computation cost of signing, verification, key aggregation and puncturing. On the other hand, the smaller f is, the better, and c are larger. As demonstrated in Figure 3, for such desirable assignments of f and c, we need to increase the size ℓ of the Bloom filter, which brings more cost of generating and storing the signing key. This is a trade-off between usability and performance.

8 Conclusion

In this work, we presented Pixel+ and Pixel++, two forward-secure multi-signature schemes from RSA and pairing, respectively. Pixel+ is based on the GQ signature and uses the RSA sequencer to maintain signing keys. It outperforms previous FS-MS constructions in terms of signing and verification efficiency. Pixel++ provides more practical forward security (i.e., fine-grained forward security) and more efficient key update by using the Bloom filter, but at the cost of a non-negligible probability of signing failure. We proved the security of Pixel+ and Pixel++ in the random oracle model, and demonstrated how to integrate them into PoS blockchains. We also implemented our proposals, and conducted several representative experiments that show its practicability.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their comments and suggestions. This work was supported by the National Natural Science Foundation of China (Nos. 62172434, 61960206014 and 62121001), China 111 Project (No. B16037), Xi'an International Science and Technology Cooperation Base, Natural Science Foundation of Henan (No. 232300421099), China Postdoctoral Science Foundation (No. 2021T140531), and Natural Science Foundation of Tianjin, China (No. 21JCZXJC00100). W. Susilo is supported by the Australian Research Council (ARC) Laureate Fellowship FL230100033 and ARC Discovery Project DP240100017. F. Guo is supported by ARC Future Fellow FT220100046.

References

[1] Michel Abdalla, Fabrice Ben Hamouda, and David Pointcheval. Tighter reductions for forward-secure signature schemes. In *Public-Key Cryptography–PKC 2013*, pages 292–311, 2013.

[2] Michel Abdalla, Sara Miner, and Chanathip Namprempre. Forward-secure threshold signature schemes. In *Topics in Cryptology–CT-RSA 2001*, pages 441–456, 2001.

[3] Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In *Advances in Cryptology–ASIACRYPT 2000*, pages 116–129. Springer, 2000.

[4] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *Advances in Cryptology–EUROCRYPT 2010*, pages 553–572. Springer, 2010.

[5] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *Proceedings of the 17th ACM conference on Computer and Communications Security–CCS 2010*, pages 473–484, 2010.

[6] Ross Anderson. Invited lecture. In *Fourth Annual Conference on Computer and Communications Security*, 1997.

[7] Christian Badertscher, Peter Gazi, 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, pages 913–930. ACM, 2018.

[8] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *CCS 2008*, pages 449–458. ACM, 2008.

[9] Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In *15th International Conference on Cryptology and Network Security–CANS 2016*, pages 140–155, 2016.

[10] Mihir Bellare and Wei Dai. Chain reductions for multi-signatures and the HBMS scheme. In *Advances in Cryptology–ASIACRYPT 2021*, pages 650–678. Springer, 2021.

[11] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 431–448. Springer, 1999.

[12] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security–CCS 2006*, pages 390–399. ACM, 2006.

[13] Mihir Bellare and Gregory Neven. Identity-based multi-signatures from RSA. In *Topics in Cryptology–CT-RSA 2007*, pages 145–162. Springer, 2007.

[14] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In *Advances in Cryptology–EUROCRYPT 2016*, volume 9666, pages 792–821. Springer, 2016.

[15] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Financial Cryptography and Data Security–FC 2016*, pages 142–157. Springer, 2016.

[16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. https://allquantor.at/blockchainbib/pdf/bentov2016snow.pdf, 2016.

[17] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[18] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography–PKC 2002*, pages 31–46, 2002.

[19] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography–PKC 2003*, pages 31–46. Springer, 2003.

[20] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology–EUROCRYPT 2005*, pages 440–456. Springer, 2005.

[21] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology–ASIACRYPT 2018*, pages 435–464. Springer, 2018.

[22] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology–CRYPTO 2005*, pages 258–275. Springer, 2005.

[23] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology–ASIACRYPT 2001*, pages 514–532, 2001.

[24] Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. Musig-l: Lattice-based multi-signature with single-round online phase. In *Advances in Cryptology–CRYPTO 2022*, pages 276–305. Springer, 2022.

[25] Xavier Boyen, Hovav Shacham, Emily Shen, and Brent Waters. Forward-secure signatures with untrusted update. In *Proceedings of the 2006 ACM SIGSAC Conference on Computer and Communications Security*, pages 191–200. ACM, 2006.

[26] Vitalik Buterin. Long-range attacks: The serious problem with adaptive proof of work. https://blog.ethereum.org/2014/05/15, 2014.

[27] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and Abhi Shelat. Multiparty generation of an RSA modulus. In *Advances in Cryptology–CRYPTO 2020*, pages 64–93, 2020.

[28] Yanbo Chen. Dualms: Efficient lattice-based two-round multi-signature with trapdoor-free simulation. In *Advances in Cryptology–CRYPTO 2023*, pages 716–747, 2023.

[29] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. In *Theory of Cryptography–TCC 2010*, pages 183–200. Springer, 2010.

[30] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. *Journal of Cryptology*, 35(2):14, 2022.

[31] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology–EUROCRYPT 2018*, pages 66–98. Springer, 2018.

[32] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-rtt key exchange. In *Advances in Cryptology–EUROCRYPT 2018*, pages 425–455. Springer, 2018.

[33] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In *Public-Key Cryptography–PKC 2018*, pages 219–250. Springer, 2018.

[34] 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–SP 2019*, pages 1084–1101. IEEE, 2019.

[35] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In *29th USENIX Security Symposium*, pages 2093–2110. USENIX Association, 2020.

[36] Priyanka Dutta, Mei Jiang, Dung Hoang Duong, Willy Susilo, Kazuhide Fukushima, and Shinsaku Kiyomoto. Hierarchical identity-based puncturable encryption from lattices with application to forward security. In *ACM Asia Conference on Computer and Communications Security–ASIACCS 2022*, pages 408–422, 2022.

[37] Nils Fleischhacker, Gottfried Herold, Mark Simkin, and Zhenfei Zhang. Chipmunk: Better synchronized multi-signatures from lattices. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security–CCS 2023*, pages 386–400, 2023.

[38] Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. Squirrel: Efficient synchronized multi-signatures from lattices. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security–CCS 2022*, pages 1109–1123. ACM, 2022.

[39] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed rsa key generation for semi-honest and malicious adversaries. In *Advances in Cryptology–CRYPTO 2018*, pages 331–361. Springer, 2018.

[40] Masayuki Fukumitsu and Shingo Hasegawa. A tightly-secure lattice-based multisignature. In *Proceedings of the 6th on ASIA Public-Key Cryptography Workshop*, pages 3–11, 2019.

[41] Masayuki Fukumitsu and Shingo Hasegawa. A lattice-based provably secure multisignature scheme in quantum random oracle model. In *14th International Conference on Provable and Practical Security–ProvSec 2020*, pages 45–64, 2020.

[42] Peter Gazi, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. In *Crypto Valley Conference on Blockchain Technology–CVCBT 2018*, pages 85–92. IEEE, 2018.

[43] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *International Workshop on Public Key Cryptography-PKC 2006*, pages 257–273, 2006.

[44] 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*, pages 51–68, 2017.

[45] Matthew D. Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy–SP 2015*, pages 305–320. IEEE, 2015.

[46] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology–CRYPTO 1988*, pages 216–231. Springer, 1988.

[47] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-rtt key exchange with full forward secrecy. In *Advances in Cryptology–EUROCRYPT 2017*, pages 519–548, 2017.

[48] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In *Advances in Cryptology–ASIACRYPT 2017*, pages 181–211. Springer, 2017.

[49] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In *Advances in Cryptology–CRYPTO 2009*, pages 654–670. Springer, 2009.

[50] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In *Advances in Cryptology–EUROCRYPT 2018*, pages 197–229, 2018.

[51] Susan Hohenberger and Brent Waters. New methods and abstractions for rsa-based forward secure signatures. In *Applied Cryptography and Network Security–ACNS 2020*, pages 292–312. Springer, 2020.

[52] Kazuharu Itakura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research and Development*, 71:1–8, 1983.

[53] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In *Advances in Cryptology–CRYPTO 2001*, pages 332–354. Springer, 2001.

[54] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology–CRYPTO 2017*, pages 357–388. Springer, 2017.

[55] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, 2012.

[56] Xinyu Li, Jing Xu, Xiong Fan, Yuchen Wang, and Zhenfeng Zhang. Puncturable signatures and applications in proof-of-stake blockchain protocols. *IEEE Transactions on Information Forensics and Security*, 15:3872–3885, 2020.

[57] Benoît Libert, Jean-Jacques Quisquater, and Moti Yung. Forward-secure signatures in untrusted update environments: efficient and generic constructions. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security–CCS 2007*, pages 266–275. ACM, 2007.

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

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

[60] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[61] Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. In *Advances in Cryptology–CRYPTO 2021*, pages 189–221. Springer, 2021.

[62] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. In *2020 ACM SIGSAC Conference on Computer and Communications Security–CCS 2020*, pages 1717–1731. ACM, 2020.

[63] Kazuo Ohta and Tatsuaki Okamoto. Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 82(1):21–31, 1999.

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

[65] Joel Reardon, David A. Basin, and Srdjan Capkun. Sok: Secure data deletion. In *2013 IEEE Symposium on Security and Privacy*, pages 301–315, 2013.

[66] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *Advances in Cryptology–EUROCRYPT 2007*, pages 228–245. Springer, 2007.

[67] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[68] Efe UA Seyitoglu, Attila A Yavuz, and Muslum Ozgur Ozmen. Compact and resilient cryptographic tools for digital forensics. In *2020 IEEE Conference on Communications and Network Security*, pages 1–9, 2020.

[69] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1(1):38–44, 1983.

[70] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science–FOCS 1994*, pages 124–134, 1994.

[71] Shifeng Sun, Amin Sakzad, Ron Steinfeld, Joseph K. Liu, and Dawu Gu. Public-key puncturable encryption: Modular and compact constructions. In *Public-Key Cryptography–PKC 2020*, pages 309–338. Springer, 2020.

[72] N. R. Sunitha and B. B. Amberker. Forward-secure multi-signatures. In *5th International Conference on Distributed Computing and Internet Technology–ICDCIT 2008*, pages 89–99. Springer, 2008.

[73] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy–SP 2017*, pages 444–460, 2017.

[74] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Symposium on Security and Privacy–SP 2016*, pages 526–545, 2016.

[75] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In *Advances in Cryptology–EUROCRYPT 2023*, 2023.

[76] Jianghong Wei, Xiaofeng Chen, Jianfeng Ma, Xuexian Hu, and Kui Ren. Communication-efficient and fine-grained forward-secure asynchronous messaging. *IEEE/ACM Transactions on Networking*, 29(5):2242–2253, 2021.

[77] Jianghong Wei, Xiaofeng Chen, Jianfeng Wang, Xinyi Huang, and Willy Susilo. Securing fine-grained data sharing and erasure in outsourced storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 34(2):552–566, 2023.

[78] Jianghong Wei, Xiaofeng Chen, Jianfeng Wang, Willy Susilo, and Ilsun You. Towards secure asynchronous messaging with forward secrecy and mutual authentication. *Information Sciences*, 626:114–132, 2023.

[79] Attila A Yavuz, Peng Ning, and Michael K Reiter. Baf and fi-baf: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems. *ACM Transactions on Information and System Security*, 15(2):1–28, 2012.

[80] Jia Yu, Rong Hao, Fanyu Kong, Xiangguo Cheng, Jianxi Fan, and Yangkui Chen. Forward-secure identity-based signature: security notions and construction. *Information Sciences*, 181(3):648–660, 2011.

[81] Tsz Hon Yuen, Joseph K Liu, Xinyi Huang, Man Ho Au, Willy Susilo, and Jianying Zhou. Forward secure attribute-based signatures. In *14th International Conference on Information and Communications Security–ICICS 2012*, pages 167–177, 2012.

A Complexity Assumptions

A.1 Number Theoretic Assumptions

The security of our first FS-MS construction Pixel+ is built upon the following RSA assumption.

Definition 5 (RSA Assumption [67]). *Given a security parameter λ, let $p = 2p'+1$ and $q = 2q'+1$ be two safe primes of size λ, and $N = p \cdot q$. Denote by e a prime randomly sampled from $[2^{\lambda}, 2^{\lambda+1}-1]$ and QR_N the $p'q'$-order group of quadratic residues in $\mathbb{Z}_N^*$. Randomly select $x \in QR_N$ and assign $y \equiv x^e \bmod N$. Then, given (N, e, y), the RSA problem is required to compute x such that $y \equiv x^e \bmod N$. We define the advantage of an algorithm $\mathcal{B}$ solving this problem as*

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{RSA}}(\lambda) = \Pr[x \leftarrow \mathcal{B}(N, e, y) : y \equiv x^e \bmod N].$$

We say that the RSA assumption holds provided that for any PPT algorithm $\mathcal{B}$, its advantage $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{RSA}}(\lambda)$ is negligible in the security parameter λ.

We will also use the following lemma in the security proof of Pixel+ and another theorem about the number of primes.

Lemma 1 (Shamir's Trick [69]). *Given $x, y \in \mathbb{Z}_N$ and $a, b \in \mathbb{Z}$ that satisfy $\gcd(a,b) = 1$ and $x^a \equiv y^b \bmod N$, then there exists an algorithm of efficiently computing $z \in \mathbb{Z}_N$ such that $z^a \equiv y \bmod N$.*

Theorem 3 (Prime Number Theorem). *Let $\pi(x)$ be the number of primes less than x. Then, for $x > 1$, we have that*

$$\frac{7}{8} \cdot \frac{x}{\ln x} < \pi(x) < \frac{9}{8} \cdot \frac{x}{\ln x}.$$

A.2 Bilinear Groups and q-BDHE Assumption

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of prime order p with a non-degenerate bilinear map pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. For simplicity, we assume that there exists an efficient algorithm BilGen that can generate bilinear groups for the given security parameter λ, i.e., $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e) \leftarrow \mathsf{BilGen}(\lambda)$. In this paper, we use the Type-3 pairing. That is, it holds that $\mathbb{G}_1 \neq \mathbb{G}_2$, and there exists a public and efficient isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ such that $\psi(g_2^x) = g_1^x$ for any $x \in \mathbb{Z}_p$.

The security of our second forward-secure multi-signature construction Pixel++ is built upon the weak q-bilinear Diffie-Hellman exponent (q-BDHE)[7] assumption [22], which was originally defined in the setting of Type-1 pairing. We utilize its variant for Type-3 pairing (denoted by ℓ-BDHE$_3^*$), and define it as follows.

Definition 6. *Given bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e) \leftarrow \mathsf{BilGen}(\lambda)$, let $\{g_i^{\alpha}, \ldots, g_i^{\alpha^q}, g_i^{\alpha^{q+2}}, \ldots, g_i^{\alpha^{2q}}, g_i^{\gamma}\}_{i=1,2}$ be random group elements, where $\alpha, \gamma \in \mathbb{Z}_p$ are random integers. Let $\mathsf{Adv}_{\mathcal{A}}^{q\text{-BDHE}_3^*}(\lambda)$ be the probability (a.k.a advantage) of an algorithm $\mathcal{A}$ successfully outputting $e(g_1, g_2)^{\alpha^{q+1}\gamma}$. We say ℓ-BDHE$_3^*$ assumption holds if for any PPT algorithm $\mathcal{A}$, its advantage $\mathsf{Adv}_{\mathcal{A}}^{q\text{-BDHE}_3^*}(\lambda)$ is negligible in the security parameter λ.*

[7] We use a specific case of this assumption in our scheme, i.e., $q = 2$.