That's not my signature! Fail-stop signatures for a post-quantum world

Cecilia Boschini[1], Hila Dahari[2], Moni Naor[2], and Eyal Ronen[3]

[1] ETH Zürich, Switzerland
cecilia.boschini@inf.ethz.ch
[2] Weizmann Institute, Israel
{hila.dahari, moni.naor}@weizmann.ac.il
Tel-Aviv University, Israel
eyal.ronen@cs.tau.ac.il

Abstract. The Snowden's revelations kick-started a community-wide effort to develop cryptographic tools against mass surveillance. In this work, we propose to add another primitive to that toolbox: Fail-Stop Signatures (FSS) [EC'89]. FSS are digital signatures enhanced with a forgery-detection mechanism that can protect a PPT signer from more powerful attackers. Despite the fascinating concept, research in this area stalled after the '90s. However, the ongoing transition to post-quantum cryptography, with its hiccups due to the novelty of underlying assumptions, has become the perfect use case for FSS. This paper aims to reboot research on FSS with practical use in mind: Our framework for FSS includes "fine-grained" security definitions (that assume a powerful, but bounded adversary e.g: can break 128-bit of security, but not 256-bit). As an application, we show new FSS constructions for the post-quantum setting. We show that FSS are equivalent to standard, provably secure digital signatures that do not require rewinding or programming random oracles, and that this implies lattice-based FSS. Our main construction is an FSS version of SPHINCS$^+$, which required building FSS versions of all its building blocks: WOTS$^+$, XMSS, and FORS. In the process, we identify and provide generic solutions for two fundamental issues arising when deriving a large number of private keys from a single seed, and when building FSS for Hash-and-Sign-based signatures.

Keywords: Fail-stop signature, foundations, hash-based signature, SPHINCS+

1 Introduction

In Asiacrypt 2015, Phillip Rogaway gave a thought-provoking IACR Distinguished Lecture titled "The Moral Character of Cryptographic Work" [51]. In light of the Snowden revelations, Rogaway called for a "community-wide effort to develop more effective means to resist mass surveillance". Threat actors such as nation-states have many possible approaches when trying to build such mass surveillance capabilities. Of these possible approaches, the most sought-after is arguably discovering and using a secret cryptographic attack that is not publicly known. More specifically, the ability to forge digital signatures and thus subvert trust mechanisms such as Public Key Infrastructures (PKIs) can lead to devastating results. In this work, we explore the use of Fail-Stop Signature (FSS) [63] to resist such a mass surveillance attempt. By using FSSs, we can allow honest signers to prove that a powerful adversary has forged a signature. Moreover, it will allow the signer to prove to the world that a previously unknown cryptographic attack has indeed been exploited and provably pinpoint the security assumption that was broken. Based on this proof, schemes based on the broken security assumption can be promptly deprecated and replaced with secure ones.

Digital Signatures. In recent years, digital signatures have received significant attention from the academic community due to the possible threat of quantum computers. Today, all widely deployed digital signature schemes are vulnerable to quantum attacks. This motivates research into new and post-quantum secure digital schemes (e.g., the NIST post-quantum standardization efforts [3,46]). Researchers are trying to develop new schemes and gain better understanding on both their classical and post-quantum security. This leads to a fast paced research cycle of proposed schemes and attacks that either completely break the underlying assumptions [11], or show that larger security parameters are needed [41]. Adding to the uncertainty is the fact that the academic community might not be aware of advances in both quantum computers and cryptanalysis achieved by nation-states and kept secret to be weaponized (e.g., the exploit of MD5 collisions for Flame [65]). This leads to the following natural question:

Can we build a mechanism to expose a previously unknown attack on a digital signature scheme that has been exploited in practice?

Although such a mechanism should not replace ongoing research on the security of the different schemes, it can serve as a "canary in the coal mine", and alert us if a scheme is practically broken and should be deprecated immediately. This can also act as a major deterrence against wide usage of exploits: such attacks are usually very costly to both develop and implement, so it would be a major loss (for the attacker) if one is detected and the affected scheme deprecated.

"Believably Hard" Cryptographic Assumptions. In cryptography, all practical (and most impractical) schemes are built based on one or more *security assumptions*. The trust placed in security assumptions is based on the fact that the cryptographic community has not yet been able to break them. However, there is no guarantee that these assumptions will remain secure in the future, as numerous schemes have been broken over time (e.g., the recent key-recovery attacks on the Rainbow signature scheme [11] and SIDH [19,39]). The case of security assumptions is no different, as seen in the case of SHA-1, whose first theoretical attack appeared in 2005 [64] and became practical in 2017 [55]. Additionally, the advent of quantum computing presents a potential scenario in which many security assumptions (e.g., factoring [54]) will no longer hold. Finally, expecting new attacks to be published is not a foolproof method, as there could be parties who choose to keep their exploits private (e.g., nation-states).
Furthermore, in modern cryptography the security of a scheme is typically based on the assumption that *a chosen security parameter is strong enough*. History shows that it is crucial to consider the possibility that the security parameter may not be sufficient in practice: In 1986 Fiat and Shamir assumed that a 512-bit length factoring challenge would be hard enough for virtually any application, while in 2020, a 829-bit length factoring challenge was broken [24,23]. This raises the question of how to determine if the chosen security parameter is adequate in the face of ever-evolving attack methods.
The implications of a *broken assumption* can be severe, as seen in the Logjam attack, where the parameters previously deemed strong enough (512-bit) were found to be insecure due to advancements in computing

power [1]. The attack broke a version of Diffie-Hellman that at the time was used by most instantiations of the TLS protocol, thus requiring an immediate, extensive update of the protocol [1]. As we cannot get rid of cryptographic assumptions, it is fair to wonder which *mitigation measures* can be put in place to reduce the impact of a (possibly secret) exploit, and how much would they cost. The ideal solution would be a "magic box" that could alert us when a security assumption has been broken. Such a mechanism might require compromises, e.g., in term of signature length. Moreover, it might not be an one-size-fits-all kind of countermeasure, but would require a clearly define adversarial scenario (e.g., assuming that some assumptions would be harder to break than others). In the context of digital signatures, it would be enough if one could detect forged signatures, and convince a verifier that a contested signature is in fact a forgery. Thus, we ask:

> *Given a digital signature based on some cryptographic assumption, is there a way to prove that a (maliciously generated) signature could only have been generated by an entity that has broken the assumption?*

Fail Stop Signatures achieve exactly this.

Fail-Stop Signatures. A Fail-Stop Signature (FSS) is a signature scheme that incorporates this kind of "canary in the coal mine" mechanism. It allows a *computationally bounded* signer to prove whether an *unbounded* adversary managed to forge signatures [63]. Hence the origin of "Fail-Stop", in case the signature scheme *fails* (there exists an adversary who can break the scheme), the honest signer can produce a publicly verifiable proof to the world to *stop* using the scheme, due to its state of insecurity. Surprisingly, only a handful of works on the topic exist [61,62,21,8,52], which do not go much further than laying the foundations. The original model [49,47] includes a (potentially malicious) authority that interacts with the signer during key generation. The rest of the protocol is essentially a standard signature enhanced with a "forgery detection" procedure. Security enhances standard unforgeability (even against a malicious authority) with two additional properties: security for the signer and security for the recipient. **Security for the signer** guarantees that if an unbounded attacker can generate a valid forgery, then a signer can generate a publicly verifiable "proof of forgery" showing that the hardness assumption underlying the scheme's security has been broken. **Security for the recipient** prevents the possibility of a malicious signer being able to sign a message and subsequently disown it by producing a proof of non-authorship. We note that if the adversary can break the scheme, but *does nothing actively*, nothing can be proven (but nothing is gained by the adversary either). However, one forgery is enough to prove to the world that the scheme is broken. Thus we believe that the "forgery detection" procedure may deter wide exploits of unknown vulnerabilities.

Can we trust the authority? FSS for the real world. Parameters of real-life implementations of cryptographic protocols have to be set by an authority (e.g., NIST). This implies a certain level of trust: even when choosing the hash function, there are cases (e.g., SNARKs) that require different hashes [9] than the tried and tested SHA-256. The role of such a party is usually limited, and in many cases one assumes it exists without even formally model it. As our work focuses on real world uses of FSS, we adapt the FSS framework to explicitly include this and similar "common practices". In particular we require the key generation to be non-interactive, and we assume the authority that produces the common reference string for the proof of forgeries to be *trusted*. One could argue that the latter is contradictory with the goal of protecting against powerful adversaries: why would we trust e.g., a standardization body, if we do not trust nation-states? The answer is that the contribution of the trusted authority is limited to the setup/key generation phase, and can in fact be distributed, e.g., using MPC. *Non-interactive key generation*, when coupled with a trusted setup, eliminates the need for real-time interaction. This allows to drop the requirement of real-time communication between parties, thus eradicating the possibility of attackers exploiting the communication channel, e.g., with network attacks, timing attacks, and other forms of interception. Finally, while in the original model the adversary is computationally unbounded, we allow for *fine-grained* security definitions: we assume that the adversary might be more powerful that the signer, but will still be *computationally bounded*. For example, the adversary might be powerful enough to break 128-bit of security, but not 256-bit of security. All of these framework changes allow us to construct *practical post-quantum FSSs*.

Table 1. Size comparison between SPHINCS$^+$ and FSS.SPHINCS implemented with either a message log, or with parallel-signing. Parameters are given for a security level $\lambda_r = 128$, where the fine-grained fail-stop mechanism ensures $\lambda_s = 256$ bits of security, reachable with expansion factor $c_s = 2$ and compression factor $c = 8$ (cf. Section 10.1).

	pk (bytes)	sk (bytes)	sig (bytes)	Bit Sec Recipient	Bit Sec Signer
SPHINCS$^+$-128s	32	64	7856	128	128
FSS.SPHINCS-128s,$c_s = 2$ (log)	32	96	11,790	128	256
FSS.SPHINCS-128s,$c_s = 2$ (c_s sigs)	32	96	23,580	128	256
SPHINCS$^+$-256s	64	128	29,792	256	256

Related works. The first attempt to deal with accountability in digital signatures schemes was in the electronic cash protocol of Chaum, Fiat and Naor, that allowed the bank to prove double-spending of coins [20]. Later, Pfitzmann and Waidner suggested the notion of fail-stop signatures [63,49,47]. Several constructions of FSS exist in the literature, based on various assumptions such as the factoring assumption [58,40,66], the RSA assumption [60,57] and a generic construction based on one-way functions [21]. A good overview can be found in [47]. There are works on making the scheme more efficient in terms of length of one signature [56,59,53,62], and how to compress efficiently many FSS [8]. Lower bounds for the length of public key and signatures in FSS appear in [61].

1.1 Our Results

The aim of this work is to lay the foundations for future research on practical FSS. Hence, our first contribution is a **restriction of the FSS framework** [47] that will be conducive to practical schemes. The key differences are a *non-interactive* key-generation protocol, *fine-grained security* definitions, and that the authority is now *trusted*. All come from observing how these kind of protocols are implemented in practice. For completeness, we also recap and extend existing results on minimal assumptions to construct FSS, and we prove the equivalence of FSS with (a subset of) digital signatures. The latter result extends a result by Pedersen and Pfitzman [47], that only proved that FSS imply signatures. Finally, we show two FSS instantiations from post-quantum computational assumptions (from lattices and from hash functions). The first is a theoretical **Lattice-Based FSS** construction that follows from our equivalence result. The second is a practical **FSS augmentation of** SPHINCS$^+$, a hash-based signature chosen for standardization by NIST. As stepping stones towards the latter, we include three different independent FSS variants of tree other signatures, WOTS$^+$, FORS, and XMSS, which are the building blocks of SPHINCS$^+$. In doing so we highlight and overcome some difficulties inherent to adding a fail-stop mechanism to any Hash-and-Sign signature. Thus we expect our approach to be applicable to a wider range of signatures.

Solidifying the foundations of FSS. The original FSS definition seems to imply a non-interactive key generation. Accordingly, all known *non-black-box* constructions of FSS do not require interaction between the signer and a trusted authority. However, the only known black-box construction of FSS from one-way functions relies on statistically hiding commitments [21,27] (cf. Lemma 3), which inherently require multiple rounds of interaction between the signer and the authority [26]. We propose a classification of FSS schemes in three classes, depending on the level of participation of the authority in the key generation, and review the minimal assumptions needed for each of the three types (see Table 2). Details are deferred to Section 4 due to length constraints. While constructing FSS from OWF remains a non-trivial open problem, we make a first step in that direction, by showing an equivalence result between FSS and digital signatures under some constraints (e.g., that the secret key has high entropy). Our new compiler from a signature to a FSS allows to build two post-quantum FSS from lattices in the standard model. As these schemes are the least practical, we defer them to Appendix C, and focus on our main result: FSS.SPHINCS.

Augmenting SPHINCS$^+$ to an FSS. Our main contribution is augmenting SPHINCS$^+$ to an FSS. SPHINCS$^+$ [10] is a stateless hash-based signature scheme, which is the only post-quantum signatures standardized by NIST [30] that is not based on lattices. Adding a fail-stop mechanism to SPHINCS$^+$ [10] turned out to be quite complicated, and yielded techniques that will have broader impact on future instatiations of FSS. In fact, SPHINCS$^+$ has three main components: WOTS$^+$ (a one-time signature), XMSS (i.e., multiple WOTS$^+$ instances compressed with a Merkle tree), and FORS, which is a "few-time" signature scheme based on binary trees similar to XMSS. As the details of the construction are quite intricate, we opt for building the FSS-variants of all three of them, to then show how to combine them to obtain FSS.SPHINCS. In doing so, we identify (and propose mitigation for) two interesting issues that arise when building FSS in presence of pseudorandom functions (PRF) and of the Hash-and-Sign paradigm. These result is in a very specific fine-grained security model, where we increase the size of the security parameters only for specific primitives (such as PRFs), while keeping the security parameters of other primitives (such as hash functions) the same as the original scheme. The security proof relies on the reductions for SPHINCS$^+$ included in the NIST specification [6]. In terms of efficiency, the main impact of adding a fail-stop mechanism is on the signature size, which increases additively by a factor linear in a constant c (as a direct result of the fail-stop mechanism). However, Table 1 shows that augmenting to FSS.SPHINCS still results in a smaller signature compared to simply using SPHINCS$^+$ with a higher security level. Indeed, consider FSS.SPHINCS with $\lambda_r = 128$, that is, that guarantees 128 bits of security for standard unforgeability (and security for the recipient), and $\lambda_s = 256$ (that is, it guarantees 256 bits of security for the signer). Table 1 shows the size comparison with SPHINCS$^+$ for the two security levels 128 and 256. FSS.SPHINCS provides a significantly shorter signature, even in its less efficient variant, while the signature size of SPHINCS$^+$ increases by a factor 4 when going from 128-bit security to 256-bit security. Further details on the computations can be found in Section 10.1.

Hash-and-Sign paradigm. In most practical digital signatures, signing requires to first hash the message to a fixed length, and then sign the hash value (e.g., RSA, DSA, ECDSA, and SPHINCS$^+$). Unfortunately, naïvely augmenting such signatures with a fail-stop mechanism yields a vulnerability: An unbounded adversary can find a collision on the hash, that is, two messages m and m' such that $h(m) = h(m')$. In this case, the adversary can query a signature σ on m and use it to authenticate m'. The honest signer cannot generate a proof that (m', σ) is a forgery, because the adversary did not break the scheme, but the hash function. Section 8 illustrates two workarounds: keeping a log, or parallel-signing.

1.2 Open Problems and Conclusions.

Our systematic study of the notions of FSS lays the foundations to re-open an exciting line of research. In fact, several natural questions stem from our work. From a more theoretical point of view, an exciting direction is to investigate whether OWF are enough to construct all types of FSS. Regarding applications, the field is even more open. Can we generalize our construction to a generic compiler for hash-based signatures? Can we augment Schnorr signature to a FSS? Can we extend more NIST-PQ signature candidates with a fail-stop mechanism (e.g., extend the approach in Appendix C to Falcon [50])? Can we better handle the Hash-and-Sign issue? Finally, it is a natural question to ask whether FSS.SPHINCS can be proved secure in a weaker fine-grained security model, and whether fine-grained assumptions are inherently necessary for hash-based FSS.

2 Technical Overview

We start by explaining the main idea behind the definition of FSS, and then the intuition behind lattice-based FSS, FSS.WOTS, FSS.XMSS, FSS.FORS, and FSS.SPHINCS. The addition of a fail-stop mechanism requires to address different challenges for each of the primitives, thus we discuss each of them separately.

2.1 Warm-up: A Toy Example

The intuition behind FSS is that any forgery generated without knowing the secret key sk contains information that could not have been generated by a polynomial-time machine (i.e., the signer), but that can be extracted using sk. This information constitutes the "proof of forgery". Thus, as long as the public key pk does not correspond to a single sk, but allows for multiple valid sk's, even an unbounded adversary has to generate forgeries containing such information. To give an intuition behind our choice for *practical* model, we start from a toy example: augmenting Lamport signature [34] on 1-bit messages to a FSS. The secret key is composed by two 128-random bits $\mathsf{sk} = (r_0, r_1)$, and the public key is their hashes $\mathsf{pk} = (h(r_0), h(r_1))$. A signature on $b \in \{0,1\}$ is simply the string r_b. Security relies on the hardness of finding collisions for the underlying hash function $h : \{0,1\}^{128} \to \{0,1\}^{128}$. Observe that if $h(r_b)$ has only *one* preimage, an *unbounded* adversary can recover the private key r_b and generate the same signature as the honest signer, thus no proof of forgery can be generated. This leads us to a *necessary condition for FSS*: the secret key must have enough entropy [61]. To augment Lamport signature to an FSS, one has to change the assumption on the hash function, which is now required to be a *compressing* collision resistance hash function $h : \{0,1\}^{128+c} \to \{0,1\}^{128}$, where c is such that $\Pr[|\{h^{-1}(h(r))\}| > 1]$ is high [31]. In this case an unbounded adversary cannot verify which of the possible preimages of the public keys $(h(r_0), h(r_1))$ is the secret key. Thus, it cannot do better then guessing. In case of a forgery, the honest signer can present the two signatures, the real one r_b and the forgery $r' \in \{h^{-1}(h(r_b))\}$, as a proof of forgery: a *collision* for h. Accordingly, *security for signer* follows from compression: if there exist at least two preimages for $h(r_b)$, even an unbounded adversary cannot find r_b with probability greater than $1/2$. *Security for recipient* follows from (computational) collision-resistance: a PPT malicious signer cannot find a collision for h with high probability. In case of widespread attacks, constant probability is enough to catch w.h.p that the scheme is broken (and attack has occurred).

Observe that every public key of a *computationally* secure digital signature reveals some information about the secret key. This implies that an unbounded adversary can find the real secret key with some probability ϵ. Hence we model the security for signer with respect to an "information theoretic" parameter ϵ. Finally, the notion of FSS requires a trusted party in the setup phase. Intuitively, the reason is to prevent the signer (or the adversary) from placing a trapdoor in the public parameters. For example, in Lamport signature, if the signer could choose the public CRH by itself, it can choose a function with a hard-coded collision, and use such information to reject its own signature as forgery. Nevertheless, our model require to put very little trust in the "trusted authority", as its contribution is limited to the setup/key generation phase. For example, in $\mathsf{FSS.SPHINCS}$ the authority only chooses the hash function (SHA-256). In real life, this authority can be the NIST (or comparable standardization bodies).

2.2 Augmenting $\mathsf{SPHINCS}^+$ to $\mathsf{FSS.SPHINCS}$

In the following we give an informal description of $\mathsf{SPHINCS}^+$, and we show the main issues that we encountered when converting it (and its building blocks) to an FSS. Throughout this work, we assume the reader to be familiar with $\mathsf{SPHINCS}^+$; for more information we refer to Appendix E and [10].

Overview of $\mathsf{SPHINCS}^+$. $\mathsf{SPHINCS}^+$ (cf. Fig. 3) is a hash-based signature obtained as an interesting mixture of Lamport and Merkle signatures [34,42]. Its core structure is a *hypertree*, that is, a modified Merkle tree where every leaf can be extended into a tree itself to allow to sign very long messages with short signing keys. The "glue" between the base tree and the next tree layer is a signature: Each leaf of the base tree is in fact a public key of a one-time signature (OTS) called WOTS^+. To add another tree, one can generate fresh WOTS^+ public keys, use them as the leaves of the new tree, and sign its root with the WOTS^+ public key contained in the leaf of the base tree. The signature resulting from extending the one-time signature WOTS^+ to a multiple-time signature through a Merkle tree is called XMSS. However, $\mathsf{SPHINCS}^+$ is not just a hypertree of XMSS instances: for efficiency reasons, the very last of the trees in the hypertree is connected to a similar tree-based few-times signature called FORS, which is used to actually sign the message digest.

The signature protocol is of the Hash-and-Sign kind: A message msg is first hashed to obtain a leaf index idx and a message digest MD. Then, the signer generates only the trees on the path from the root pk to the

leaf idx. The WOTS^+ pk contained in the last leaf idx of the path is used to sign a FORS pk, which is in turn used to sign the message digest MD. To recap, a signature on msg contains the authentication path from the root pk to the leaf idx (including the WOTS^+ signatures that connect the tree layers), the FORS signature on MD, and the randomness used to generate the message digest.

It remains to explain how WOTS^+ and FORS work. FORS utilizes k trees of depth d, where the leaves in each tree are hashes of random strings (the sk's). The roots of the k trees form the public key. To sign a message msg^*, the message is split into k blocks, each signed individually. The i-th block is treated as an integer $\mathsf{idx}_i \in [1, d]$. Signing the i-th block requires revealing the authentication path from the root of the i-th tree to its idx_i-th leaf, and a preimage of that leaf, that is, one of the secret keys. WOTS^+ is a one-time signature that relies on ℓ hash chains. The starting point of each chain is a block of the secret key (a random bit string), and each subsequent chain element is obtained by hashing the previous one. The public keys are the output of the last hash evaluation. Signing a message implies revealing an intermediate value in each chain, where the level depends on the message. To avoid easy forgeries, a signature is generated on both the message and its checksum. Verification requires to iterate the chaining function on the signature until one obtains the public key. Observe that WOTS^+ is deterministic: in $\mathsf{SPHINCS}^+$ this ensures the uniqueness of the entire hypertree structure given the first level. This implies that the key generation is efficient, as the signer only has to generate the first level of the hypertree.

Before we describe our challenges in the construction, we state our result.

Theorem 1 (Informal).

- *Assuming* $\mathsf{SPHINCS}^+$ *is unforgeable under adaptive CPA, and* PRF, $\mathsf{PRF}_{\mathsf{msg}}$ *are secure PRFs, then* FSS.SPHINCS *is unforgeable under adaptive CPA.*
- *Assuming* $\mathsf{SPHINCS}^+$ *is unforgeable under adaptive CPA, and* PRF, $\mathsf{PRF}_{\mathsf{msg}}$ *are secure PRFs, then* FSS.SPHINCS *is secure for signer against an adversary* $\mathcal{A}$ *with running time at most* $2^{c_s \lambda_s / 2}$ *(in the QROM).*
- *Assuming* $\mathsf{SPHINCS}^+$ *is unforgeable under adaptive CPA, then* FSS.SPHINCS *is secure for the recipient.*

Hiding the sk: *Compressing* Hash Chains. Adding a fail-stop mechanism to WOTS^+ can be done with an approach similar to the one used by Kiktenko et al. [33]. In this work the authors identify the possibility of producing a proof-of-forgery in some special cases, in particular, as long as the adversary could not guess what is the correct preimage from the set of possible ones. Intuitively, WOTS^+ is based on a *chaining function* $c^{j,k}$ that applies a hash function i times on a (randomly chosen, secret) input x. If the adversary is not able to correctly guess one of the hidden values in the chain, a forged signature implies that the adversary has found a collision somewhere in the chain, which can be easily recovered by a honest signer and presented as proof-of-forgery. For this to happen, the chaining function has to be compressing (so that points can have multiple preimages), and behave like a random oracle (so that the probability that a random evaluation of the function has many preimages is high). The latter assumption is not new, as $\mathsf{SPHINCS}^+$ implicitly relies on it too[3]. However, the security analysis in [33] is not complete: what is proved is that w.h.p. the adversary cannot guess the preimage of one of the chains elements. This is not enough in a scenario where the adversary is unbounded: as long as there is even one element in the chain with exactly one preimage, the adversary can find it (through brute-force) and produce the exact same signature as the signer would have. Moreover, [33] lacks the FSS framework, and as a result, does not introduce a trusted authority, nor prove security for the recipient, meaning that their scheme does not include any guarantee in case the signer is dishonest. Thus our analysis extends and improve [33].

Dealing with PRFs: Fine-Grained Assumption. When implementing a signature it is common to generate the random strings that compose the secret key by evaluating a PRF over a counter. This way the signer has to store only the seed of the PRF, which is much shorter than the collection of secret random strings. This is done in XMSS and FORS too, as they require an exponential number of secret keys. Observe

[3] The QROM assumption is necessary to construct practical, secure tweakable hash functions from known hash functions, cf. [10, Section 6 and Appendix F].

that the hypertree is essentially public, as it is completely revealed after a number of signatures have been generated. Thus, information-theoretically the tree leaks the secret Seed: an unbounded adversary could recover it completely and break our FSS requirements. The trivial solution to this challenge is to increase the security parameters. However, even this solution is not always "trivial": increasing the security parameters of a hash function requires years of cryptanalysis. In addition, this will significantly increase the *length* of the signature, which is a heavy price to pay. Ideally, an elegant solution would achieve succinct secret key *and* signatures. Unfortunately, succinct secret keys cannot yield secure FSS: van Heijst et al. [61] show that the size of the secret key has to be at least *linear* in the number of messages to be signed. To work around this lower bound and achieve succinctness for both the secret key and signatures, we extend the original FSS framework to allow for a more "fine-grained" adversarial model. We assume that the adversary is much more powerful than the signer, to the point that it can forge the signature, but it is still somewhat bounded: despite its ability to forge, some computational tasks remain out of its reach. In particular, we assume that, if we increase only the size of the seed Seed of the PRF, the adversary cannot break its security. In practice, expanding the size of the seed Seed does not effect the length of the signature. To motivate our assumption, observe that the size of the key to the PRF increases linearly, the run time of generic brute force attack increases exponentially even when using quantum attacks such as the Grover algorithm [25]. Thus, in our FSS.XMSS and FSS.FORS constructions, we expand the seed of the PRF by a factor c_s, and we assume that an adversary that successfully returns a forgery is still not able to break the pseudorandomness of the PRF. There is still only one possible PRF's seed Seed, but enumerating over all possible values of Seed is much harder task to the powerful but bounded adversary. A more detailed explanation can be found in Section 6.1. Observe that we do not need this trick to hide the preimages of the nodes of the Merkle trees. By construction, SPHINCS$^+$ does not hide the (hyper)tree nodes, as the only *undetectable* way to break the scheme is to find all the correct preimages both for the tree and for the chains. Thus, hiding the preimages of the first element of the chains (and of the leaves of FORS) is enough to prevent an undetectable forgery, even if the adversary can reconstruct the whole tree.

The Hash-And-Sign Workaround. The final step to construct FSS.SPHINCS (and FSS.FORS) requires handling a generic forgery attack that applies to any digital signature based on the Hash-and-Sign paradigm. Recall that in Hash-and-Sign signatures, the message msg can be of arbitrary length and is first hashed into a fixed-length digest that is then signed. Therefore, instead of targeting the signature itself, an unbounded adversary can find another message msg* such that $HASH(\mathsf{msg}^*) = HASH(\mathsf{msg})$, where msg is a previously signed message: a honest signature for msg can then be re-used as a signature on msg*, and no proof of forgery can be generated. To the best of our knowledge, no previous FSS solution addressed this specific problem. We propose two possible solutions. The first one is to rely on a log file storing all the signed messages, so that the signer can produce the collision $(\mathsf{msg}, \mathsf{msg}^*)$ for $HASH$ as a proof of forgery in case such an attack is performed. We remark that adding the log file does not make it a stateful signature, as the log does not have to be secret, and it is only used when generating a proof of forgery. As such, the loss/publication of the state does not impact the original security of the signature (the standard unforgeability) against PPT adversaries. In addition, usually when several agents/server want to sign with the same secret key using a stateful signature, they need to keep syncing their state for security to hold. This is not needed in our case, where the signers have to just combine their logs when a proof of forgery has to be generated. The second solution is based on a fine-grained assumption on the adversary, and uses multiple signatures on multiple unique hashes of the message to prevent the possibility of generating a collision. More details on the attack can be found in Section 8. Section 10 explains how to leverage our solutions to transform SPHINCS$^+$ in FSS.SPHINCS.

3 Definition of FSS in the non-interactive model

A Fail-Stop Signature (FSS) is essentially a standard signature enhanced with a "forgery detection" procedure: a trusted authority computes the public parameters so that, in case of a disputed signature, a PPT signer can convince the authority that generating such a signature from the public parameters required solving a task impossible for a bounded-time machine.

Definition 1. *A* Fail-Stop signature (FSS) *in the non-interactive model consists of six PPT algorithms* (GenCh, GenKey, Sign, VrfySig, PoF, VrfyPoF) *such that:*

$\mathsf{ch} \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon})$**:** *the* challenge-generation algorithm *takes as input the security parameters* $(1^{\lambda_r}, 1^{\varepsilon})$*, and outputs a challenge* ch*.*

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{GenKey}(\mathsf{ch})$**:** *the* key-generation algorithm *takes as input the challenge, and outputs a (secret) signing key* sk *and a (public) verification key* pk*.*

$\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$**:** *the* signing algorithm *takes as input a signing key* sk *and a message* m *from the message space* $\mathcal{M}$*, and returns a signature* σ*.*

$b \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$**:** *the* verification algorithm *takes as input a public key* pk*, and a message-signature pair* (m, σ)*, and returns 1 if* σ *is valid, 0 otherwise.*

$\pi \leftarrow \mathsf{PoF}_{\mathsf{sk}}((m, \sigma), \mathsf{ch})$**:** *the* proof of forgery algorithm *takes as input the secret key* sk*, a message* m*, and a signature* σ*. If* $\mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma_m) = 0$*, it aborts (i.e., it returns* $\perp$*). Otherwise, it returns a proof* π*.*

$b \leftarrow \mathsf{VrfyPoF}(\mathsf{ch}, \pi)$**:** *the* proof of forgery verification algorithm *takes as input a challenge* ch*, and a proof* π*. It outputs 1 if* π *is valid, and 0 otherwise.*

Correctness is defined analogously as for digital signatures, so we omit it here.

At a high-level, an FSS is secure if it is *secure for recipient*, that is, if it guarantees to a verifier that a signer cannot repudiate its own signature, and *secure for the signer*, i.e., it ensures that a signer can explain forgeries, even if generated by unbounded adversaries. However, security for the signer is meaningful only if unbounded adversaries are the only threat to the scheme [21]. This seems counter-intuitive, as the notion of unforgeability by a PPT adversary is weaker than security for the signer: if a signature can be successfully forged by a PPT adversary, then it can also be forged by an unbounded one. At the same time, security for the signer only guarantees that forgeries can be disputed: without unforgeability it could still happen that every PPT adversary could generate a forgery, thus exposing the signer to the risk of having to constantly generate proof of forgeries. Essentially, unforgeability gives the additional guarantee that the need for a proof of forgery only arises in *exceptional cases*, i.e., when the corrupted recipient is not a PPT machine, but an unbounded one.

In light of these observations, the security of an FSS has two security parameters: a "computational" one, λ_r, that bounds the probability that a PPT adversary breaks security (be it a signer trying to disavow its own signature, or a PPT malicious verifier attempting an impersonation attack), and a "information-theoretical" one, ε, that bounds the probability that an unbounded adversary can produce a forgery for which no proof of forgery can be generated. To be able to easily integrate our fine-grained assumption on the adversary in the FSS framework, we define security for the signer a bit differently than in the literature: we only require that the success probability of $\mathcal{A}$ is bounded by some ε, without requiring it to be negligible.

Definition 2 (Security for the Signer). *An FSS* Σ *is* ε-secure for the signer *for* $0 < \varepsilon \leq 1$ *if, fixed* $\lambda_r > 0$*, and for all unbounded adversary* $\mathcal{A}$ *it holds:*

$$\Pr[\mathsf{Exp}^{\mathsf{ss}}_{\mathcal{A},\Sigma}(\varepsilon, \lambda_r) = 1] \leq \varepsilon$$

where the security experiment is Experiment 1 that returns 1 *if the signer fails to provide a valid proof of forgery. This probability is over the random coins of* GenKey*,* $\mathcal{A}$ *and* PoF*.*

Remark 1. Definition 2 only allows a signer to prove that a forgery has occurred. However, in practice one might want to be able to prove that a *specific* signature was forged, e.g., to avoid liabilities due to a forged signature on a contract. One way to do it is for the signer to reveal the secret key alongside the disputed signature, to show that by using this specific signature one can generate a proof of forgery from the secret key. As a valid proof of forgery implies there is a successful attack against the signature, publishing the secret key, for the honest signer, is not an issue since in any case we must stop using the scheme. Every secure FSS allows for this: in the rest of the paper we focus on the more basic task of proving that a forgery has occurred.

Experiment 1: $\mathsf{Exp}^{\mathsf{ss}}_{\mathcal{A},\Sigma}(\varepsilon,\lambda_r)$

1: $\mathsf{ch} \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon})$
2: $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenKey}(\mathsf{ch})$
3: $(m^*,\sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}(\cdot)}(\mathsf{ch},\mathsf{pk})$
4: $b_1 \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(m^*,\sigma^*)$
5: $\pi \leftarrow \mathsf{PoF}_{\mathsf{sk}}((m^*,\sigma^*),\mathsf{ch})$
6: $b_2 \leftarrow \mathsf{VrfyPoF}(\mathsf{ch},\pi)$
7: **if** $m^* \notin \mathcal{Q}$ **then**
8: **return** $b_1 \wedge \neg b_2$
9: **else return** 0

Experiment 2: $\mathsf{Exp}^{\mathsf{unf}}_{\mathcal{A},\Sigma}(\varepsilon,\lambda_r)$

1: $\mathsf{ch} \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon})$
2: $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenKey}(\mathsf{ch})$
3: $(m^*,\sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}(\cdot)}(\mathsf{ch},\mathsf{pk})$
4: $b \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(m^*,\sigma^*)$
5: **if** $(m^* \notin \mathcal{Q})$ **then**
6: **return** b
7: **else return** 0

Fig. 1. Security experiments for security for the signer and unforgeability. $\mathcal{O}_{\mathsf{sk}}$ is the signing oracle (which aborts if the maximum number of signatures that the scheme allows is reached), and $\mathcal{Q}$ is the list of the queries it receives.

Definition 3 (Security for the Recipient). *An FSS scheme is* secure for the recipient *iff for all $\lambda_r > 0$ and PPT adversary $\mathcal{A}$ there exists a negligible function* negl *such that:*

$$\Pr\left[\mathsf{VrfyPoF}(\mathsf{ch},\pi^*) = 1 \mid \begin{array}{c}\mathsf{ch} \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon}),\\ \pi^* \leftarrow \mathcal{A}(\mathsf{ch})\end{array}\right] < \mathsf{negl}(\lambda_r)\ .$$

The probability is computed on the random coins of GenCh*, and $\mathcal{A}$.*

Definition 4 (Unforgeability). *An FSS signature scheme Σ is* existentially unforgeable under adaptive chosen-message attack*, if fixed $\lambda_r > 0$, for all $0 < \varepsilon \leq 1$ and PPT adversaries $\mathcal{A}$ there is a negligible function* negl *such that:*

$$\Pr[\mathsf{Exp}^{\mathsf{unf}}_{\mathcal{A},\Sigma}(\varepsilon,\lambda_r) = 1] \leq \mathsf{negl}(\lambda_r)$$

where the security experiment is Experiment 2.

Van Hejist et al. [61] identified the following necessary condition for a secure FSS, which we decide to include as it constitutes an easy "rule of thumb" to check whether a candidate FSS is trivially broken. Let H be the Shannon entropy.

Lemma 1 (Necessary Condition for security of FSS). *Every FSS that satisfies Definition 3 and Definition 2 for $\varepsilon = \mathsf{negl}(\lambda)$, also satisfies the following property :*

$$\mathsf{H}(\mathsf{sk} \mid \mathsf{pk}, Hist) \geq (N+1)(\min\{\lambda_r,\lambda\} - 1) \tag{1}$$

where $Hist$ is the list of the first N signatures generated by the honest signer (cf. [61, Lemma 1]). If signing is deterministic, the requirement reduces to $\mathsf{H}(\mathsf{sk} \mid \mathsf{pk}) \geq (N+1)(\min\{\lambda_r,\lambda\} - 1)$ (cf. [61, Theorem 5]).

Lemma 1 is information-theoretical, in the sense that it is a necessary condition for security against an *unbounded* $\mathcal{A}$. However, in practice it is not always possible to guarantee such a high level of security while maintaining the usability of a primitive. This can be seen for theoretical results too: we can show that FSSs are equivalent to signatures (cf. Appendix B), but only assuming that Eq. (1) holds for the signature too. Thus, we introduce a relaxation of Lemma 1, to allow for a more fine-grained security model. Instead of assuming that $\mathcal{A}$ is unbounded, we fix a third parameter λ_s, and assume that $\mathcal{A}$ is more powerful than PPT, but still *somewhat* bounded in λ_s: it can break unforgeability for security parameter λ_r, but it cannot extract information about the secret key from the public key and signatures for a *large enough* secret key. This implies that when dealing with fine-grained assumptions on the adversary, we need to substitute Lemma 1 with the following assumption.

Definition 5 (Fine-Grained Necessary Condition for security of FSS). *Let $\lambda_r,\ \lambda_s, c_s \in \mathbb{N}$, $0 < \varepsilon \leq 1$. Let $\mathcal{A}$ be an adversary that breaks unforgeability of the FSS with security parameter λ_r, and consider an*

	Minimal assumption	References
Interactive key generation (Type 1)	OWF	[21]+[27]
One private message key generation (Type 2)	OWF	This work
Non-interactive key generation (Type 3)	CRH	This work (implicit in [28])
	CIH	[21]

Table 2. Summary of minimal assumptions for FSS (definitions recap in Appendix A). CRH stands for collision-resistant hash functions, and CIH for collision-intractable hash functions. Both are included as the relation between them is still an open question.

FSS that is secure for the recipient and $(\varepsilon + \mathsf{negl}(\lambda_s))$*-secure for the signer against* $\mathcal{A}$*. For such an FSS there exists a function* $f : \{0,1\}^{c_s \cdot \lambda_r} \times \{0,1\}^{\lambda_r} \to \{0,1\}^{\lambda_r}$ *such that for* $\forall x$*:*

$$|\Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}}(\mathsf{pk}, f(\mathsf{sk}, x)) \ : \ \mathsf{sk} \xleftarrow{\$} \{0,1\}^{c_s \cdot \lambda_r}] + \\ - \Pr[1 \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}}(\mathsf{pk}, r) \ : \ r \xleftarrow{\$} \{0,1\}^{\lambda_r}]| \leq \mathsf{negl}(\lambda_s)$$

where $\mathcal{O}_{\mathsf{sk}}$ *is the signing oracle.*

The requirement on the security for the signer changes as well as the adversary now has a probability $\leq \mathsf{negl}(\lambda_s)$ of recovering the secret key. In Section 6.1 we show how to use this threat model in the case of XMSS, FORS, and SPHINCS$^+$.

4 Minimal Assumptions for FSS

4.1 Background: Communication Channel Assumptions for GenKey

As already mentioned in Section 2, understanding which are the minimal assumptions for FSS requires us to first clarify the communication requirements. To this end, we classify FSS schemes in three classes, depending on the level of participation of the authority in the key generation.

1. **Interactive Key Generation:** this assumes two-ways (possibly insecure) communication channel between the trusted authority and the signer, that jointly generate the signer's keys. The original model falls under this type, thus making it susceptible to man-in-the-middle attack.
2. **One (private) Message Key Generation:** these schemes require a *secure* one-time one-way channel from the authority to the signer. This includes schemes where the trusted authority generates part of the secret key, while the rest of the key is generated by the signer alone.
3. **Non-Interactive Key Generation:** These schemes only assume that the trusted authority has a broadcast channel, i.e., it does not participate in the key generation. This is the model we focus in our work. At the beginning, $\mathcal{T}$ broadcasts the challenge. Upon receiving it, the signer generates its keys and broadcasts the verification key as well. For example, the Authority can choose a hash function, and then the signer can use the hash in the key generation.

Lemma 2. *Type 3 FSS* $\Rightarrow$ *Type 2 FSS* $\Rightarrow$ *Type 1 FSS (with a secure channel, or with an insecure channel assuming KEM+SKE).*

Proof. The first implication is trivial. The second follows considering that a secure channel can be implemented on an insecure one through key-exchange combined with symmetric key encryption. □

4.2 Minimal Assumptions for FSS

Given this classification, it is natural to ask whether different communication requirements in the key generation require different minimal assumptions to build FSS. Table 2 summarizes the state of the art. Throughout this section we assume $\varepsilon = \mathsf{negl}(\lambda)$ for some $\lambda \in \mathbb{N}$, as all the constructions assume an unbounded adversary against security for the signer.

Lemma 3 (OWF ⇒ Type 1 FSS, Informal). *Assuming the existence of OWFs, then there exists a secure FSS with interactive key generation.*

Proof (sketch). This fact follows combining two results: a black-box construction of FSS from statistically hiding commitments by Damgård et al. [21], and the black-box construction of statistically hiding commitments from OWFs by Haitner and Reingold [27]. The idea behind the commitment-based FSS is that the signer has two one-time keys, C_0 and C_1. Each key contains 2λ bit-commitments $C_b = \{Com_b(r_i^b)\}_{i=1,\ldots,2\lambda}$ with distinct opening random strings r_i, generated by interacting with the authority (which does not know the openings). To sign a bit b, the signer reveals $\{r_i^b\}_{i=1,\ldots,2\lambda}$ (i.e., opens the corresponding commitments).

This construction is not enough to obtain a black-box construction of Type 2 FSS from OWF. Indeed, statistically hiding commitments require at least $\Omega(\frac{\lambda}{\log \lambda})$ rounds of interaction [26]. Thus the previous construction requires multiple interaction with the trusted authority to generate the commitments in the public key of the signer. In the following, we present our new proposal.

Lemma 4 (OWF ⇒ Type 2 FSS, Informal). *Assuming the existence of OWFs, then there exists a secure FSS with one message key generation.*

Proof. The FSS is constructed as follows: the authority chooses a random function, and two sets of n random points each U^0 and U^1. It sets the public key to be the sets V^0, V^1 of evaluations of f on the elements of U^0, U^1 respectively. Then it sends in the private channel two random subsets $S^b \subseteq U^b$, $|S^b| = n' < n$, $b = 0, 1$ to the signer, which constitute the secret key. Signing a bit b requires returning a random element in S^b, that is, inverting a *random* element in V^b. As an unbounded adversary cannot know which preimages of the elements in V^0, V^1 the signer has, w.h.p. a forged signature on b^* will contain a preimage outside of S^{b^*}. Such a preimage constitutes the proof of forgery, and security trivially follows from one-wayness. Security for the signer requires that the probability that $\mathcal{A}$ guesses correctly is negligible, that is, that $n'/n \leq \varepsilon$.

To prove security for recipient, assume there exists a PPT malicious signer $\mathcal{A}$ who breaks security for recipient with probability $\varepsilon_{\mathcal{A}}$. We construct a PPT adversary $\mathcal{B}$ that breaks the security of OWF with probability $\frac{\varepsilon_{\mathcal{A}}}{2(n-n')}$ exploiting $\mathcal{A}$. Given a challenge z on which $\mathcal{B}$ needs to invert f, $\mathcal{B}$ simulates the authority by randomly setting an element of $V^0 \cup V^1$ as z and generating the rest of the elements as evaluations of the OWF on random points. Then, it sends n' preimages to $\mathcal{A}$. A successful forgery requires $\mathcal{A}$ to invert one more element in $V^0 \cup V^1$, thus $\mathcal{A}$ returns a preimage of z with probability $\frac{1}{2(n-n')}$, and $\mathcal{B}$ wins with probability $\frac{\varepsilon_{\mathcal{A}}}{2(n-n')}$. Finally, unforgeability trivially follows from one-wayness. □

This can be extended to multiple use with standard techniques, that is, compressing multiple instance of the one-time signatures with a Merkle tree (cf. [21] or our FSS.XMSS construction in Section 6).

Finally, a construction of Type 3 FSS can be obtained from the statistically hiding commitment scheme by Damgård et al. [21] by instantiating the commitment scheme with the construction by Halevi and Micali [28].

Lemma 5 (CRH ⇒ Type 3 FSS, Informal). *Assuming the existence of CRHs, then there exists a secure FSS with non-interactive key generation.*

Proof. Protocol 2 is obtained taking the construction of FSS from a statistically hiding commitment scheme by Damgård et al. [21] and instantiating the commitment scheme with the construction by Halevi and Micali [28]. The step in GenKey, Line 6 can be done efficiently as described in [28, Footnote 5].

Protocol 1: Type 2 FSS from OWF

Let $\mathcal{F} = \{f : \{0,1\}^{\lambda_r} \to \{0,1\}^{\lambda_r}\}$ be a family of OWF, and $\mathcal{M} = \{0,1\}$ be the message space.

$\mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon})$:

1: Choose $n', n \in \mathbb{N} \setminus \{0\}$ such that $n'/n \leq \varepsilon$
2: $f \xleftarrow{\$} \mathcal{F}$
3: **for** $b = 0, 1$ **do**
4: **for** $j = 1, \ldots, n$ **do**
5: $u_j^b \xleftarrow{\$} \{0,1\}^{\lambda_r}$
6: $v_j^b \leftarrow f(u_j^b)$
7: $U^b \leftarrow \{u_1^b, \ldots, u_n^b\}$
8: $V^b \leftarrow \{v_1^b, \ldots, v_n^b\}$
9: $I_b \xleftarrow{\$} [n], |I_b| = n'$ // I_b is a random subset of indexes
10: $S^b \leftarrow \{(i, u_i^b) \mid i \in I_b\}$
11: $\mathsf{ch} \leftarrow (n, n', \lambda_r, f, V^0, V^1)$
12: **return**
Private channel: (S^0, S^1)
Public channel: ch

$\mathsf{GenKey}(\mathsf{ch}, (S^0, S^1))$:

1: Parse $\mathsf{ch} = (n, n', \lambda_r, f, V^0, V^1)$
2: $\mathsf{pk} \leftarrow (V^0, V^1)$
3: $\mathsf{sk} \leftarrow (S^0, S^1)$
4: **return** $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}_{\mathsf{sk}}(b)$:

1: Parse $\mathsf{sk} = (S^0, S^1)$
2: $(j, x) \xleftarrow{\$} S^b$
3: $\sigma \leftarrow (j, x)$
4: **return** σ.

$\mathsf{VrfySig}_{\mathsf{pk}}(b, \sigma)$:

1: Parse $\mathsf{pk} = (\mathsf{pk}^0, \mathsf{pk}^1)$
2: Parse $\sigma = (j, x)$
3: **if** $\mathsf{pk}_j^b = f(x)$ **then return** 1.
4: **else return** 0.

$\mathsf{PoF}_{\mathsf{sk}}((b, \sigma), \mathsf{ch})$:

1: **if** $\mathsf{VrfySig}_{\mathsf{pk}}(b, \sigma) = 0$ **then return** $\perp$
2: Parse $\mathsf{ch} = (n, n', \lambda_r, f, V^0, V^1)$
3: Parse $\sigma = (j, x)$
4: **if** $\sigma \notin S^b$ **then**
5: $S^b = S^b \cup \{\sigma\}$
6: **return** (b, S^b)
7: **else return** $\perp$.

$\mathsf{VrfyPoF}(\mathsf{ch}, \pi)$:

1: Parse $\pi = (b, S^b)$
2: Parse $\mathsf{ch} = (n, n', \lambda_r, f, V^0, V^1)$
3: **if** $[|S^b| = n' + 1] \ \wedge \ [\forall (j, x) \in S^b : \mathsf{pk}_j^b = f(x)]$ **then return** 1
4: **else return** 0

Security follows combining [21, Theorem 3.1] with [28, Theorem 1]. The only difference is that Damgård et al. used a bit-commitment, while the construction by Halevi and Micali allows to commit to a λ-bits long string. Thus the public key for the bit b can be a single commitment to a 2λ-bits long random string, instead of 2λ commitments to distinct random strings in order to have $\varepsilon = \mathsf{negl}(\lambda)$. □

The question of whether one can build Type 3 FSS black-box from OWF remains, to the best of our knowledge, open. Nevertheless, in Appendix B we shed more light in that direction, by showing an equivalence result between FSS and standard digital signature. The intuition is quite easy: One can see the fail-stop mechanism as a security reduction to a computationally hard problem. As long as Eq. (1) holds, a forgery allows a PPT signer to generate a solution of the instance of the problem chosen by the trusted authority.

5 One-time FSS (or from WOTS$^+$ to an FSS)

In this section, we show how to build a one-time hash-based FSS. We first present the high-level idea of WOTS$^+$, and then how to augment it to an FSS.

WOTS$^+$ structure. WOTS$^+$ [29] is a hash-based one-time signature that is built on the Winternitz signature [42]. The latter is preferable to Lamport signatures [34], as it reduces the length of signatures and keys by signing the representation of a message $m \in \{0,1\}^h$ in base w, for some $w \in \mathbb{N}$ (WOTS$^+$ with $w = 2$ is essentially Lamport signatures). The construction relies on a *chaining function* c^k, that is, a function that

Protocol 2: Type 3 FSS from CRH

Let $\mathcal{H} = \{h : \{0,1\}^L \to \{0,1\}^k\}$ be a family of CRH.
Let $\mathcal{U} = \{u : \{0,1\}^L \to \{0,1\}^n\}$ be a family of universal hash functions.
Let $\mathcal{M} = \{0,1\}$ be the message space.

$\mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon})$:
1: Let $\varepsilon = 2^{-\lambda}$
2: Choose $k \geq \max\{\lambda_r, 2\lambda + 4\}$
3: $n \leftarrow 2\lambda$
4: $L \leftarrow 4k + 2n + 4$
5: **for** $b = 0, 1$ **do**
6: $\quad h_b \xleftarrow{\$} \mathcal{H}$
7: **return** $\mathsf{ch} \leftarrow (n, k, L, h_0, h_1)$

$\mathsf{GenKey}(\mathsf{ch})$:
1: Parse $\mathsf{ch} = (n, k, L, h_0, h_1)$
2: **for** $b = 0, 1$ **do**
3: $\quad x_b \xleftarrow{\$} \{0,1\}^n$
4: $\quad r_b \xleftarrow{\$} \{0,1\}^L$
5: $\quad y_b \leftarrow h_b(r_b)$
6: $\quad$ Choose $u_b \in \mathcal{U}$ such that $x_b \leftarrow u_b(r_b)$
7: $\quad \mathsf{pk}_b \leftarrow (y_b, u_b)$
8: $\quad \mathsf{sk}_b \leftarrow (x_b, r_b)$
9: $\mathsf{sk} \leftarrow (\mathsf{sk}_0, \mathsf{sk}_1)$
10: $\mathsf{pk} \leftarrow (\mathsf{pk}_0, \mathsf{pk}_1)$
11: **return** $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}_{\mathsf{sk}}(b)$:
1: Parse $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1)$
2: **return** $\sigma = \mathsf{sk}_b$.

$\mathsf{VrfySig}_{\mathsf{pk}}(b, \sigma)$:
1: Parse $\mathsf{pk}_b = (y_b, u_b)$
2: Parse $\sigma = (x, r)$
3: **if** $(x = u_b(r)) \ \wedge \ (y_b = h_b(r))$ **then return** 1.
4: **else return** 0.

$\mathsf{PoF}_{\mathsf{sk}}((b, \sigma), \mathsf{ch})$:
1: **if** $0 \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(b, \sigma)$ **then return** $\perp$
2: Parse $\sigma = (x, r)$
3: Parse $\mathsf{ch} = (n, k, L, h_0, h_1)$
4: **if** $(r \neq r_b) \ \wedge \ (h_b(r) = h_b(r_b))$ **then**
5: $\quad \pi \leftarrow (b, r_b, r)$
6: $\quad$ **return** π
7: **else return** $\perp$.

$\mathsf{VrfyPoF}(\mathsf{ch}, \pi)$:
1: Parse $\pi = (b, r_b, r)$
2: Parse $\mathsf{ch} = (n, k, L, h_0, h_1)$
3: **if** $(r \neq x_b) \ \wedge \ (h_b(r) = h_b(r_b))$ **then return** 1
4: **else return** 0

applies a second-preimage resistant, somewhat pseudorandom one-way function f for $w - 1$ times to each secret key x:

$$c^k(x) = \begin{cases} x & \text{if } k = 0 \\ f(c^{k-1}(x)) & \text{otherwise} \end{cases}$$

where f is a fixed public function chosen at random from a family $F_n := \{f : \{0,1\}^n \to \{0,1\}^n\}$. The chaining function c takes as input some randomness too, but we ignore it here for simplicity (cf. Appendix E.2 for details). The construction yields ℓ_1 chains, where $\ell_1 = \lceil h / \log w \rceil$, i.e., one chain per component of the representation of m in base w (denoted by $[m]_w$ from now on). Let m_i be the i^{th} component of $[m]_w$: the i^{th} component of the signature would be $c^{m_i}(\mathsf{sk}_i)$.

This is not enough to guarantee unforgeability though. For example, by querying a signature on a message m such that $[m]_w = (0, \ldots, 0)$ the adversary gets all the secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_{\ell_1})$, thus in this case an adversary can perfectly impersonate the signer. To avoid this, the message digest that is signed includes both the message m and a *checksum* $C = \sum_{i=1}^{\ell_2} (w - 1 - m_i)$. This increases the length of keys and signature by $\ell_2 = \lfloor \log_w(\ell_1(w-1)) \rfloor + 1$, but now guarantees unforgeability under some special assumptions on f. The algorithms of WOTS^+ are formally described in Appendix E.2.

Augmenting WOTS^+ to an FSS. Kiktenko et al. [33] observed that, if an adversary $\mathcal{A}$ is not able to correctly guess one of the hidden values in the chain, a forged signature implies that $\mathcal{A}$ has found a *collision* somewhere in the chain. An honest signer could easily recover the collision using its secret key, and use it

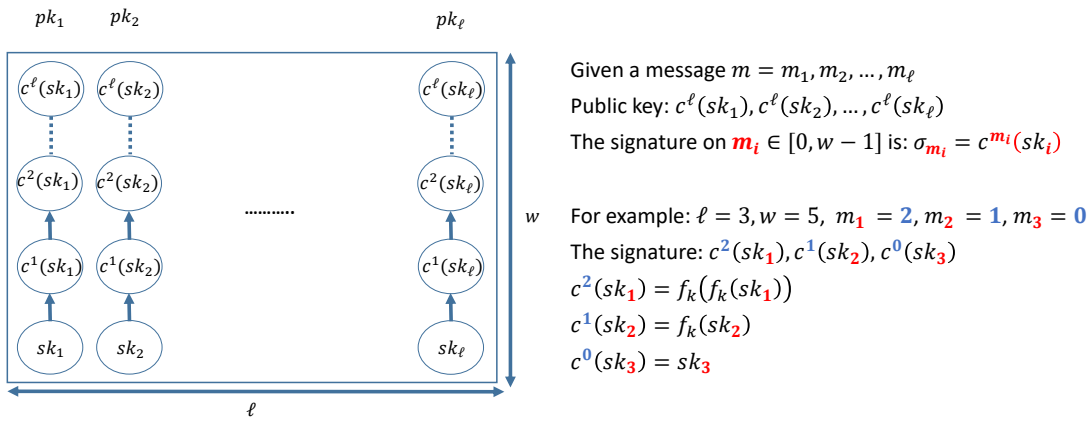

Fig. 2. Key generation of WOTS^+ using the chaining function $c^k(\cdot)$, and a toy example of the singing algorithm (where we leave out the checksum for the sake of simplicity). Recall that a message m is composed by ℓ elements $m_1, \dots, m_\ell$, where $m_i \in [0, w-1]$.

as a proof that a more powerful machine generated the disputed signature: the signer being a PPT machine could not have broken collision-resistance. A necessary but not sufficient condition to prevent an unbounded adversary from recovering a preimage in the chain is that an evaluation $y = c^k(\mathsf{sk}_i)$ statistically hides sk_i (cf. Lemma 1), and that multiple choices of sk_i can correspond to the same y. However, this is not true when WOTS^+ is instantiated with a standard hash function. In [33], the authors propose as workaround to change the chaining function to a *compressing* one-way function, and assume it to be a random oracle[4] (so that the number of preimages is distributed in the same way as for random functions). Inspired by this approach, our FSS.WOTS relies on compressing hash functions. As the only difference between FSS.WOTS and WOTS^+ is that the hash function is compressing, we defer the formal protocol description to appendix (cf. Protocol 3), and only focus on describing the hash.

On the Use of Compressing, Tweakable Hash Functions (THF). In FSS.WOTS hash chains are built using compressing hash functions, so that even if an adversary recovers a possible sk^*, there is still a chance that it is not the preimage chosen by the honest signer. This technique increases the size of the sk by a *compression factor*[5] c, but the size is still linear in w. Modeling the security of this function requires some care. Usually, in the security experiment the adversary has oracle access to the function, thus it can query the evaluation of the function on any input x of its choice. This is not the case in the unforgeability game of WOTS^+ (and, analogously, of FSS.WOTS): here the adversary can only choose the *position* in the chain that will be opened, not its input (which is given by the iterative application of the function on the secret key). In [10], they model the index of the chain as a *tweak* of the function: in the security game the input of the function is uniformly chosen (and different for every chain), while the adversary can query the oracle on tweaks of its choice. To make sure that the public key hides the secret key one has to assume undetectability of the THF (*SM-UD security*): informally it means that $\mathbf{Th}(p, t, x)$ is computationally indistinguishable from x, where x is uniformly sampled. Due to this "pseudorandomness", a successful forgery σ' must contain at least one intermediate value x of a chain whose preimage is not the one computed during key generation.

[4] This assumption is already present in $\mathsf{SPHINCS}^+$. Indeed, in [10] the authors propose three constructions of THF from hash functions to instantiate $\mathsf{SPHINCS}^+$. Among these, the only one proved secure in the standard model (i.e., [10, Construction 5]) is not used in practice, as it would imply exponentially-sized public parameters. The other two rely on the QROM, thus $\mathsf{SPHINCS}^+$ implicitly assumes it too.

[5] One can assume that the description of the function is public, and the signer only has to publish the parameters of the function it has selected.

Security requires second-preimage resistance of the THF (*SM-PRE security*), that is, the adversary not be able to find a *second preimage* for a target from the set of its queries Q given an oracle access to $\mathbf{Th}(p, \cdot, x)$ for random x, and collision resistance (*SM-TCR security*), i.e., that the adversary cannot find a *collision* for a target from the set of its queries Q given oracle access to $\mathbf{Th}(p, \cdot, \cdot)$. Definitions of THF and their security are in Appendix D.1.

Chaining Function. Let $w \in \mathbb{N}$ a base, ℓ the total number of chains, and n the security parameter (for the FSS constructions, $n = \lambda_r$). Let

$$\mathbf{Th} = \{\mathbf{Th}_i : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{n+c\cdot(w-i)} \to \{0,1\}^{n+c\cdot(w-i-1)} \text{ for } i = 1, \ldots, w-1\}$$

be a family of tweakable hash functions with parameter set $\mathcal{P} := \{0,1\}^n$ and tweaks[6] $\mathcal{T} = \{0,1\}^{256}$. The chaining function of FSS.WOTS takes as inputs an iteration counter $k \in \{0, \ldots, w-1\}$, a start index $j \in \{0, \ldots, w-1\}$, a chain index $i \in \{1, \ldots, w-1\}$, a message $x \in \{0,1\}^{n+c\cdot(w-j-1)}$ (which length varies due to the ongoing compression in the chain), and a public parameter Seed and behaves as follows:

$$c^{j,k}(x, i, \mathsf{Seed}) = \begin{cases} x & \text{if } k = 0 \\ \mathbf{Th}_{j+k}(\mathsf{Seed}, T_{i,j+k-1}, c^{j,k-1}(x, i, \mathsf{Seed})) & \text{if } k > 0 \end{cases}$$

Observe that j and k mean that the chaining function assumes that the input x is the value of the chain after j iterations, and starts from computing $\mathbf{Th}_{j+1}$ on the input x, iterating for k times (until $\mathbf{Th}_{j+k}$). Tweaks are defined as the output of a (deterministic) encoding function $T(\mathsf{typ}, \mathsf{adrs}) = T_{a,b}$ that associates a distinct tweak $T_{a,b}$ with the b^{th} function call in the a^{th} chain generated with the THF $\mathbf{Th}_b(\mathsf{Seed}, \cdot, \cdot)$. The value of typ and adrs are deterministically generated to uniquely identify the position of the call to **Th** in the SPHINCS$^+$ structure: typ has different values depending on what the THF is used for, and adrs is the "address" of the point where **Th** is called. For the purpose of the proofs, we just need to know that the encoding function T is *injective*. We refer the reader to [6, Section 2.7.3] for a formal definition.

Due to its use in SPHINCS$^+$, it is enough to prove non-adaptive security of FSS.WOTS, both for unforgeability and for security for the signer.

Lemma 6 (Non-Adaptive Unforgeability). *If* **Th** *is a family of THF that is SM-UD secure,* non-adaptive *SM-TCR secure, and SM-PRE secure, then* FSS.WOTS *is unforgeable under non-adaptive CPA.*

The proof is similar to [31, Theorem 1], thus it is deferred to Appendix F.

On the Adversarial Model. Relying on the FSS framework allows our model of the adversary (and our security analysis) to be more precise than the analysis by Kiktenko et al. [33]. In the FSS framework only extremely powerful adversaries can forge, as the unforgeability property stops the attempt of PPT ones. Kiktenko et al. [33] implicitly assumed that the adversary cannot *on avarage* find the right preimage of a point in the chain. However, we assume that having an unbounded adversary means that $\mathcal{A}$ can enumerate *all* preimages of a given point: assuming that finding a preimage is a probabilistic algorithm, the adversary can simply repeat it a logarithmic number of times to find all of the preimages. This can be done for all the points in all of the chains. So, if there is even a single point in one chain with only one preimage, the adversary can find it and break the FSS. Thus, we need to make sure that the probability that *every point in every chain* has only a single preimage is negligible, not just the expected probability of choosing the correct preimage over all of the points. Lemma 7 shows our analysis. Observe that even though the adversary can enumerate all the preimages in the chains, it cannot win with overwhelming probability as long as the output of **Th** does not leak information about the preimage used by the signer.

[6] This is the specific choice for SPHINCS$^+$. In general, we need $|\mathcal{T}| \geq w\ell$.

Protocol 3: FSS.WOTS

Let $[m]_w$ denote the representation of m base w, and $\mathcal{M} = \{0,1\}^l$ be the message space.

Let $\mathbf{Th} = \{\mathbf{Th}_i : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{n+c\cdot(w-i)} \to \{0,1\}^{n+c\cdot(w-i-1)}$ for $i = 1,\dots,w-1\}$ be a family of THF.

The public parameters ch are implicitly given as input to all the algorithms (excluding GenCh).

$\mathsf{GenCh}(1^{\lambda_r}, 1^{\lambda_s})$:

$n \leftarrow \max\{\lambda_r, \lambda_s\}$
Choose w.
$\ell_1 \leftarrow \lceil \frac{l}{\log w} \rceil$
$\ell_2 \leftarrow \lfloor \frac{\log(\ell_1(w-1))}{\log w} \rfloor + 1$
$\ell \leftarrow \ell_1 + \ell_2$
Set c such that $2\ell(w-1)\exp(-2^c) \leq 2^{-\lambda_s}$
$\mathsf{Seed} \xleftarrow{\$} \{0,1\}^n$
return $\mathsf{ch} = (\lambda_r, \lambda_s w, \ell_1, \ell_2, c, \mathsf{Seed})$

$\mathsf{GenKey}(\mathsf{ch})$:

Parse $\mathsf{ch} = (\lambda_r, \lambda_s w, \ell_1, \ell_2, c, \mathsf{Seed})$
$\ell \leftarrow \ell_1 + \ell_2$
for $i = 1,\dots,\ell$ **do**
  $\mathsf{sk}_i \xleftarrow{\$} \{0,1\}^{n+c(w-1)}$
  $\mathsf{pk}_i \leftarrow c^{0,w-1}(\mathsf{sk}_i, i, \mathsf{Seed})$
$\mathsf{sk} \leftarrow (\mathsf{sk}_1, \dots, \mathsf{sk}_\ell)$
$\mathsf{pk} \leftarrow (\mathsf{Seed}, \mathsf{pk}_1, \dots, \mathsf{pk}_\ell)$
return $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}_{\mathsf{sk}}(m)$:

$\ell \leftarrow \ell_1 + \ell_2$
$(m_1, \dots, m_{\ell_1}) \leftarrow [m]_w$, $m_i \in [0, w-1]$.
$C \leftarrow \sum_{i=1}^{\ell_1}(w-1-m_i)$ // checksum
$(c_1, \dots, c_{\ell_2}) \leftarrow [C]_w$, $c_{i'} \in [0, w-1]$
$(b_1, \dots, b_\ell) \leftarrow (m_1, \dots m_{\ell_1}, c_1, \dots, c_{\ell_2})$
for $i = 1,\dots,\ell$ **do**
  Compute $\sigma_i = c^{0,b_i}(\mathsf{sk}_i, i, \mathsf{Seed})$
$\sigma \leftarrow (\sigma_1, \dots, \sigma_\ell)$
return σ.

$\mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$:

$(\sigma_1, \dots, \sigma_\ell) \leftarrow \sigma$
$(m_1, \dots, m_{\ell_1}) \leftarrow [m]_w$, $m_i \in [0, w-1]$.
$C \leftarrow \sum_{i=1}^{\ell_1}(w-1-m_i)$ // checksum
$(c_1, \dots, c_{\ell_2}) \leftarrow [C]_w$, $c_{i'} \in [0, w-1]$
$(b_1, \dots, b_\ell) \leftarrow (m_1, \dots m_{\ell_1}, c_1, \dots, c_{\ell_2})$
for $i = 1,\dots,\ell$ **do**
  $\sigma'_i = c^{b_i, w-1-b_i}(\sigma_i, i, \mathsf{Seed})$
if $\mathsf{pk}_i = \sigma'_i$, $\forall\, i \in [1, w-1]$ **then**
  return 1.
else return 0.

$\mathsf{PoF}_{\mathsf{sk}}((m, \sigma), \mathsf{ch})$:

$(\sigma_1, \dots, \sigma_\ell) \leftarrow \sigma$
if $0 \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$ **then return** $\perp$
if $\exists\, i, j \in [0, w-1]$: $c^{0,b_i+j}(\mathsf{sk}_i, i, \mathsf{Seed}) \neq c^{b_i,j}(\sigma_i, i, \mathsf{Seed})$ **then**
  $\pi \leftarrow (b_i, i, j, \mathsf{sk}_i, \sigma_i)$
  return π
else return $\perp$.

$\mathsf{VrfyPoF}(\mathsf{ch}, \pi)$:

if $(c^{0,w-1}(\mathsf{sk}_i, i, \mathsf{Seed}) = c^{b_i, w-1-b_i}(\sigma_i, i, \mathsf{Seed})) \wedge (c^{0,b_i+j}(\mathsf{sk}_i, i, \mathsf{Seed}) \neq c^{b_i,j}(\sigma_i, i, \mathsf{Seed}))$ **then**
  return 1
else return 0

Lemma 7 (implicit in [33, Lemma 1]). *Let $f : \{0,1\}^{n+\delta} \to \{0,1\}^n$, $n \gg 1$, $\delta > 0$ be chosen uniformly at random from the set of all functions from $\{0,1\}^{n+\delta}$ to $\{0,1\}^n$. Then, the probability that a point y* in the image *has strictly more than one preimage can be bounded as*

$$\Pr[\exists\ \mathcal{S} \subseteq \{0,1\}^{n+\delta}, |\mathcal{S}| > 1\ :\ f(x) = y\ \forall\ x \in \mathcal{S}] \geq 1 - 2\exp(-2^\delta)\ .$$

The proof is similar to [33], and can be found in Appendix E.3.

To use Lemma 7 in our analysis we still need the QROM assumption, to ensure that the output of a tweakable hash function on a randomly sampled input is indistinguishable from random, even by a powerful adversary.

Theorem 2 (Security for the Signer). *If* $\mathbf{Th}$ *is a family of compressing THF then there exist a constant value for the compression factor c such that* FSS.WOTS *is ε-secure for the signer in the QROM for $\varepsilon = 1/2$.*

Proof. Let $\mathcal{A}$ be an adversary in Experiment 1. Then its success probability is

$$\Pr[\mathsf{Exp}^{\mathsf{ss}}_{\mathcal{A},\Sigma}(\lambda_s, \lambda_r) = 1] = \Pr[\sigma^* = \sigma' \mid \sigma' \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m^*)] \tag{2}$$

i.e., the adversary wins if it has guessed correctly all the ℓ preimages in σ^*. As we assume the adversary to be unbounded, if there is *any* unopened point in any chain that has exactly one preimage, then $\mathcal{A}$ is always successful (as inverting the hash on one chain is enough to generate a forgery). Lemma 7 combined with the observation that the outputs of $c^{0,j}(\mathsf{sk}_i, i, \mathsf{Seed})$ are uniformly random and independent (by the QROM

assumption on the construction of the THF from hash functions) yields that this happens with probability

$$\begin{aligned}&\Pr[\exists\ i \in \{1,\dots,\ell\},\ j \in \{1,\dots,w-1\} \mid c^{0,j}(\mathsf{sk}_i, i, \mathsf{Seed}) \text{ has only one preimage}]\\ &\quad \leq \ell(w-1)2\exp(-2^c)\end{aligned} \tag{3}$$

which for $n = l = 128$ and $w = 16$ (the usual choice for $\mathsf{SPHINCS}^+$) is already less than 2^{-81} for $c = 6$. Combining (2) and (3) yields that $\mathcal{A}$ looses with probability

$$\begin{aligned}&\Pr[\mathsf{Exp}^{\mathsf{ss}}_{\mathcal{A},\Sigma}(\lambda_s, \lambda_r) = 0] =\\ &\quad \geq \left(1 - \ell(w-1)\frac{2^n}{2^n-1}\exp(-2^c)\right) \cdot \Pr[\mathcal{A} \text{ wrongly guesses at least once}]\\ &\quad \geq (1 - \ell(w-1)2\exp(-2^c)) \cdot \frac{1}{2} \approx \frac{1}{2} - \mathsf{negl}(\lambda_s)\end{aligned}$$

that is, the signer can generate a proof of forgery essentially 50% of the times. The probability of a correct guess by $\mathcal{A}$ follows observing that the THF behaves as a QROM, thus it does not leak any information about the input. Given that each evaluation $\mathcal{A}$ sees has multiple preimages with overwhelming probability, the adversary cannot win with probability much larger than $1/2$. Analogously, if c is chosen so that the probability of strictly more than 2 preimages is overwhelming, then the adversary's winning probability becomes $1/3 + \mathsf{negl}(\lambda_s)$. □

Second Preimage vs. Collisions. Observe that unforgeability requires SM-PRE security, that is, something analogous to second preimage resistance, but proving a forgery just requires breaking SM-TCR security, that is, a property similar to collision-resistance. This is because the latter is a weaker property implied by the former: a forgery breaking SM-PRE security implies that SM-TCR security is broken too, but the vice versa does not hold.

Theorem 3 (Security for the Recipient). *If* **Th** *is a family of* adaptively *SM-TCR secure THF, then* FSS.WOTS *is secure for the recipient.*

Proof (sketch). We sketch the proof in the following. A formal reduction can be obtained through a series of hybrid games analogously to the proof of Lemma 6.

Assume that there exists a malicious PPT signer $\mathcal{A}$ that break the security for the recipient of FSS.WOTS with probability ε. We show that one can construct a PPT algorithm $\mathcal{B}$ that breaks the adaptive SM-TCR security of an element of the family **Th**. At the beginning of the *adaptive* SM-TCR experiment, $\mathcal{B}$ receives the public parameter Seed from the challenger. From these, it computes the rest of ch , and runs $\mathcal{A}$ on ch. Upon receiving $\pi^* = (b, i, j, \mathsf{sk}_i, \sigma_i)$ from $\mathcal{A}$, $\mathcal{B}$ finds $k \in [j+1, w-1]$ for which it holds $c^{0,b_i+k}(\mathsf{sk}_i, i, \mathsf{Seed}) \neq c^{b_i,k}(\sigma_i, i, \mathsf{Seed})$ and $c^{0,b_i+k+1}(\mathsf{sk}_i, i, \mathsf{Seed}) = c^{b_i,k+1}(\sigma_i, i, \mathsf{Seed}))$. Then it computes $M_1 = c^{0,b_i+k}(\mathsf{sk}_i, i, \mathsf{Seed})$ and $M_2 = c^{b_i,k}(\sigma_i, i, \mathsf{Seed}))$, the index $\bar{i} = b_i + k + 1$, and the tweak $T = T_{i,\bar{i}}$, and returns $(M_1, M_2, \bar{i}, T)$ to the game. As $\mathcal{B}$ perfectly simulates the authority, its success probability is equal to the success probability of $\mathcal{A}$ in breaking security for the recipient. □

6 Augmenting XMSS to an FSS

Informally, XMSS is a set of N WOTS^+ instances whose public keys are compressed using a Merkle tree. Each signature includes a WOTS^+ signature and an authentication path that allows the verifier to recompute the root. As each WOTS^+ key pair can only be used once, the state of the XMSS scheme needs keep track of the used pair, thus it is *stateful*. To reduce the total size of the private key (we need for *every* WOTS^+ signature ℓ independent secret keys), all unique private keys used in the WOTS^+ signatures are generated from a single private $\mathsf{Seed}_{\mathsf{sk}}$ using a PRF, so the i^{th} secret key of the j^{th} leaf (a WOTS^+ signature) is $\mathsf{sk}_i^j = PRF(\mathsf{Seed}_{\mathsf{sk}}, i, j)$. We augment XMSS to an FSS scheme we call FSS.XMSS, which is essentially XMSS where we replace WOTS^+ with our new FSS.WOTS and a slightly larger private $\mathsf{Seed}_{\mathsf{sk}}$ is used.

6.1 Fine-grained Assumption on Adversary Capabilities

Our FSS.WOTS variant of the WOTS^+ signature assumes an unbounded adversary that is able to break the one-wayness property of the hash function and find *all* possible preimages for some target value y. We show that as long as w.h.p. *all* hash values in *all* the chains in the WOTS^+ signature have more than one preimage, the honest signer can prove with some constant (non-zero) probability that the forged signature is indeed a forgery even assuming that the adversary is unbounded. Our proof relies on the fact that due to the compression parameter, there are many different private keys that will result in the same signature and public key. This means that even if an adversary can recover all possible private keys that are consistent with the signature, it will not be able to know which is the correct one. However, this is not the case for signature schemes such as XMSS. Each XMSS signature includes an exponential number of WOTS^+ signatures. Each WOTS^+ signature requires its own unique set of private keys. Not to store an exponential size secret, the private keys used as the start of each chain are computed using a PRF and a small private key. This means that unless we allow an exponentially large private key (which is not practical), even a small fraction of the total supported number of signatures of the XMSS scheme will result in just a single possible private key that is consistent with the observed signatures. Thus an unbounded adversary can recover the correct private key and use it to forge signatures. In such a scenario the honest server will not be able to prove a signature was forged.

Fortunately, all is not lost! Instead of assuming a completely unbounded adversary, we can use the more fine-grained but natural assumption of Definition 5. We assume a very powerful (exponential-time) adversary able to break the security assumptions of the XMSS scheme with security parameter λ_r. Under this assumption, if XMSS uses a PRF with a key of size λ_r to generate the private keys, we assume that the attacker can recover the single possible $\mathsf{Seed}_{\mathsf{sk}}$ after seeing enough outputs and perfectly impersonate a honest signer. However, we assume that if we increase the size of the PRF's key even by a small constant factor $c_s > 1$, we make such a key recovery attack exponentially harder. Following Definition 5 with our PRF as the function f, we assume that with a when using a a secret key of size $c_s \cdot \lambda_r$ our adversary is unable to distinguish between the output of the PRF and random samples.

To motivate our assumption, consider the generic classical brute force attack, with an attacker that runs in time $O(2^{\lambda_r})$ and can enumerate all possible keys and find the correct one. However, when we increase the key size to $c_s \cdot \lambda_r$, the attacker is required to run in time $O(2^{c_s \cdot \lambda_r})$. Even for a quantum adversary, the runtime of the generic attack using Grover algorithm [25] will increase exponentially from $O(2^{\frac{\lambda_r}{2}})$ to $O(2^{\frac{c_s \cdot \lambda_r}{2}})$.[7] To conclude, we assume a powerful adversary, able to break at least one of the security assumptions of the XMSS scheme with non-negligible property for security parameter λ_r but that has only a negligible probability of breaking the security of our PRF when using a larger security parameter $c_s \cdot \lambda_r$; c_s is treated as third security parameter, alongside λ_r and λ_s.

6.2 Binary Trees in FSS

Both XMSS and FORS and their FSS counterparts rely on binary trees [42], thus we include in this section summarizes an informal description of the standard tree-related algorithms. No algorithm to compute leaves is included, as XMSS and FORS compute leaves differently. Let $H : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$ be a (compressing) THF.

$\mathsf{Tree}_{\mathsf{H}}$**:** on input the leaves and the seeds, it returns the root (using H).

$\mathsf{TreeGenAuth}_{\mathsf{H}}$**:** on input the leaves, the address of a leaf, and the seed, it generates the authentication path of the leaf.

$\mathsf{VrfyAuth}_{\mathsf{H}}$**:** on input an authentication path, the root, and the seed, it verifies the correctness of the path w.r.t. the root.

[7] Note that in a real-world instantiation of the PRF the function's state size must be large enough to accommodate the entire $\mathsf{Seed}_{\mathsf{sk}}$ to avoid analogous exploits to [48].

$\mathsf{CTree}_{\mathsf{H}}$**:** on input a root, two authentication paths and the position of the starting leaf, and the seed, it checks if there are collisions in the paths. It returns the two pairs of nodes and tweaks $T^0, (n_0^0, n_1^0)$ and $T^1, (n_0^1, n_1^1)$ that yield the collision, or $\perp$ if no collision is found.

Remark that the THF used to build the tree can just be compressing with a (standard) factor $c = 2$. This is because the constructions of the signatures do not hide the inputs of the hash from the adversary, thus there is no need for the hash to hide preimages.

6.3 Construction of FSS.XMSS

Let $\mathbf{H} = \{\mathbf{H}_i\}_i$ be a family of (compressing) THF, where $H_i : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{\lambda_r \cdot i} \to \{0,1\}^{\lambda_r}$.

Analogously to XMSS (cf. Appendix E.4), FSS.XMSS (see formal description in Protocol 4) allows to sign

Protocol 4: FSS.XMSS

Let $H : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{2\lambda_r} \to \{0,1\}^{\lambda_r}$ and $F : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{\ell\lambda_r} \to \{0,1\}^{\lambda_r}$ be THFs.

Let $\mathsf{PRF}_1 : \{0,1\}^{c_s\lambda_r} \times \mathcal{T} \to \{0,1\}^{\lambda_r + c(w-1)}$ be a PRF and $\mathcal{M} = \{0,1\}^l$ the message space.

Let $\mathsf{FSS.WOTS} = (\mathsf{GenCh}', \mathsf{GenKey}', \mathsf{Sign}', \mathsf{VrfySig}'.\mathsf{PoF}', \mathsf{VrfyPoF}')$ and $N = 2^h$.

$\mathsf{GenCh}(1^{\lambda_r}, 1^{\lambda_s}, 1^{c_s})$:
1: $\mathsf{ch}' \leftarrow \mathsf{GenCh}'(1^{\lambda_r}, 1^{\lambda_s})$
2: $\ell \leftarrow \ell_1 + \ell_2$
3: Parse $\mathsf{ch}' = (\lambda_r, \lambda_s, w, \ell_1, \ell_2, c, \mathsf{Seed}_{\mathsf{pk}})$
4: Set h s.t. $2^{h+1}\ell(w-1)\exp(-2^c) \leq 2^{-\lambda_s}$
5: $\mathsf{ch} \leftarrow (\mathsf{ch}', c_s, h)$
6: **return** ch

$\mathsf{GenKey}(\mathsf{ch})$:
1: Parse $\mathsf{ch} = (\lambda_r, \lambda_s, w, \ell_1, \ell_2, c, \mathsf{Seed}_{\mathsf{pk}}, c_s, h)$
2: $N \leftarrow 2^h$
3: $\mathsf{Seed}_{\mathsf{sk}} \xleftarrow{\$} \{0,1\}^{c_s\lambda_r}$
4: **for** $i = 1, \ldots, N$ **do**
5: Set adrs_i according to specs.
6: $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{GenKey}'(\mathsf{ch}', \mathsf{Seed}_{\mathsf{sk}}, \mathsf{Seed}_{\mathsf{pk}}, \mathsf{adrs}_i)$
7: $\mathsf{lf}_i \leftarrow F(\mathsf{Seed}_{\mathsf{pk}}, \mathsf{adrs}_i, \mathsf{pk}_i)$
8: $\mathsf{root} \leftarrow \mathsf{Tree}_{\mathsf{H}}(\mathsf{lf}_1, \ldots, \mathsf{lf}_{\mathsf{N}}, \mathsf{Seed}_{\mathsf{pk}})$
9: $\mathsf{sk} \leftarrow \mathsf{Seed}_{\mathsf{sk}}$
10: $\mathsf{pk} \leftarrow (\mathsf{root}, \mathsf{Seed}_{\mathsf{pk}})$
11: $\mathsf{st} \leftarrow 0$
12: **return** $(\mathsf{pk}, \mathsf{sk}, \mathsf{st})$.

$\mathsf{Sign}_{\mathsf{sk}}(m; \mathsf{st})$:
1: Parse $\mathsf{sk} = \mathsf{Seed}_{\mathsf{sk}}$
2: $N \leftarrow 2^h$
3: **if** $\mathsf{st} \geq N$ **then return** $\perp$
4: **else**
5: **for** $i = 1, \ldots, N$ **do**
6: Set adrs_i according to specs.
7: $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{GenKey}_H(\mathsf{ch}', \mathsf{Seed}_{\mathsf{sk}}, \mathsf{Seed}_{\mathsf{pk}}, \mathsf{adrs}_i)$
8: $\mathsf{lf}_i \leftarrow F(\mathsf{Seed}_{\mathsf{pk}}, \mathsf{adrs}_i, \mathsf{pk}_i)$
9: $\sigma' \leftarrow \mathsf{Sign}'_{\mathsf{sk}_{\mathsf{st}}}(m)$
10: $\mathsf{auth} \leftarrow \mathsf{TreeGenAuth}_{\mathsf{H}}(\mathsf{lf}_1, \ldots, \mathsf{lf}_{\mathsf{N}}, \mathsf{adrs}_{\mathsf{st}}, \mathsf{Seed}_{\mathsf{pk}})$
11: $\sigma \leftarrow (\sigma', \mathsf{auth})$
12: $\mathsf{st} \leftarrow \mathsf{st} + 1$
13: **return** $(\sigma; \mathsf{st})$.

$\mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$:
1: Parse $\sigma = (\sigma', \mathsf{auth})$
2: $b \leftarrow \mathsf{VrfyAuth}_{\mathsf{H}}(\mathsf{pk}, \mathsf{auth})$
3: Extract pk^i from auth
4: $b' \leftarrow \mathsf{VrfySig}'_{\mathsf{pk}^i}(m, \sigma')$
5: **return** $b \wedge b'$

$\mathsf{PoF}_{\mathsf{sk}}((m, \sigma), \mathsf{ch})$:
1: Parse $\sigma = (\sigma', \mathsf{auth})$
2: Parse $\mathsf{ch} = (\mathsf{ch}', c_s, h)$
3: **if** $0 \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$ **then**
4: **return** $\perp$
5: Extract i from σ.
6: $(\bar{\sigma}', \bar{\mathsf{auth}}) \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m; i)$
7: $P \leftarrow \mathsf{CTree}_{\mathsf{H}}(\mathsf{root}, \mathsf{Seed}_{\mathsf{pk}}, \mathsf{auth}, \bar{\mathsf{auth}})$
8: **if** $P \neq \perp$ **then** $\pi \leftarrow (0, P)$
9: **else**
10: $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{GenKey}'(\mathsf{ch}', \mathsf{Seed}_{\mathsf{sk}}, \mathsf{Seed}_{\mathsf{pk}}, \mathsf{adrs}_i)$
11: Parse $\mathsf{auth} = (\mathsf{lf}, \mathsf{auth}')$
12: $\pi' \leftarrow \mathsf{PoF}'_{\mathsf{sk}_i}((m, \sigma'), \mathsf{ch}')$
13: $\pi \leftarrow (1, \pi')$
14: **return** π

$\mathsf{VrfyPoF}(\mathsf{ch}, \pi)$:
1: Parse $\pi = (b, \pi')$
2: **if** $b = 0$ **then**
3: Parse $P = (T^0, n_0^0, n_1^0, T^1, n_0^1, n_1^1)$
4: $x_0 \leftarrow H(\mathsf{Seed}_{\mathsf{pk}}, T^0, n_0^0, n_1^0)$
5: $x_1 \leftarrow H(\mathsf{Seed}_{\mathsf{pk}}, T^1, n_0^1, n_1^1)$
6: $b' \leftarrow (x_0 = x_1)$
7: **else**
8: Parse $\mathsf{ch} = (\mathsf{ch}', c_s, h)$
9: $b' \leftarrow \mathsf{VrfyPoF}(\mathsf{ch}', \pi')$
10: **return** b'

$N = 2^h$ messages of l bits by combining N parallel instantiations of FSS.WOTS in a binary tree. Such tree is constructed using the THF $H := \mathbf{H}_2$. An internal state allows the signer to keep track of which keys have been already used.

FSS.WOTS with Public Key Compression. As the public key pk of FSS.WOTS has a size linear in the length of the messages, in XMSS and FSS.XMSS it is compressed using a THF $F := \mathbf{H}_\ell$ into a n-bits long string lf. The $\{\mathsf{lf}_i\}_i$ constitutes the leaves of the binary tree. The secret keys of the various FSS.WOTS

instances are generated using the PRF $\mathsf{PRF}_1 : \{0,1\}^{c_s\lambda_r} \times \mathcal{T} \to \{0,1\}^{\lambda_r + c(w-1)}$ from a secret seed $\mathsf{Seed}_{\mathsf{sk}}$ and from the address adrs, and the chains are generated from the same seed $\mathsf{Seed}_{\mathsf{pk}}$. We abuse the notation and give $(\mathsf{Seed}_{\mathsf{sk}}, \mathsf{Seed}_{,\mathsf{pk}}, \mathsf{adrs})$ as input to the key generation of FSS.WOTS. Analogously, the FSS.WOTS verification now also checks that the tops of the chains obtained from the signature hash to the correct value lf_i. Different public keys hashing to the same leaf constitute a valid proof of forgery for this modified FSS.WOTS.

As XMSS is a stateful signature, the syntax of the FSS is adapted accordingly. The proof of forgery can be derived either from a collision on the tree, or as in FSS.WOTS, thus π contains a bit b that specifies which case it is.

7 Security of FSS.XMSS

Lemma 8 (Non-adaptive Unforgeability). *If* FSS.WOTS *is an unforgeable FSS,* PRF_1 *is a PRF, and* H *and* F *are SM-TCR secure THFs, then* FSS.XMSS *is unforgeable under non-adaptive CPA.*

Proof (sketch). Consider the following sequence of hybrid games:

- $\mathcal{H}_1$: this is Experiment 4 for FSS.XMSS.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, but the sk's are random strings instead of output by PRF_1.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, but the winning condition now excludes cases when the forgery contains a collision on the tree.
- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, but the winning condition now excludes cases when the forgery contains a collision on a leaf.

Clearly, distinguishing $\mathcal{H}_1$ from $\mathcal{H}_2$ requires distinguishing outputs of the PRF from random. To distinguish $\mathcal{H}_2$ from $\mathcal{H}_3$ (resp., $\mathcal{H}_3$ from $\mathcal{H}_4$), the adversary has to return a collision on the tree (resp., on a leaf). This happens only if $\mathcal{A}$ returns a forgery σ^* where the tree path is computed from a different leaf than what was used to generate the root (resp., where the leaf is obtained hashing different values than the FSS.WOTS key pk_i). This means that the FSS.WOTS signature in σ^* has to verify w.r.t. a key pk that was not among the ones generated by the challenger. Hence a successful distinguisher can be used to break the SM-TCR security of H (resp., of F). Finally, to win $\mathcal{H}_4$ Adv has only one option: forging a signature using one of the FSS.WOTS keys generated by the challenger. Thus winning $\mathcal{H}_4$ is essentially equivalent to breaking the unforgeability of FSS.WOTS. A tighter proof can be obtained not relying on the unforgeability of FSS.WOTS as a black-box, but by reducing to the security of **Th** following the same steps as in the proof of Lemma 6. The only difference is that now the security of **Th** has to hold for a larger number of queries (polynomial in λ_s).

Lemma 9 (Security for the Signer). *If* FSS.WOTS *is secure for the signer, then* FSS.XMSS *is* $1/2$*-secure for the signer against an adversary* $\mathcal{A}$ *that satisfies Definition 5.*

Proof. Consider the following hybrid games sequence:

- $\mathcal{H}_1$: this is Experiment 1 for FSS.XMSS.
- $\mathcal{H}_2$: Same as $\mathcal{H}_1$, but the sk's are random strings instead of output by PRF_1.
- $\mathcal{H}_3$: Same as $\mathcal{H}_2$, but the winning condition now excludes cases when the forgery contains a collision on the tree.
- $\mathcal{H}_4$: Same as $\mathcal{H}_3$, but the winning condition now excludes cases when the forgery contains a collision on a leaf.

Distinguishing $\mathcal{H}_1$ from $\mathcal{H}_2$ requires to distinguish the output of the PRF from uniformly sampled strings, which is not possible in polynomial-time under our fine-grained assumption. Now, observe that the Merkle tree is generated with a public seed and a THF that does not guarantee the existence of more than one preimage per point. Thus, as the adversary runs in exponential time, $\mathcal{A}$ is able to reconstruct the whole

tree. Hence, $\mathcal{A}$ has the same success probability in both $\mathcal{H}_2$ and $\mathcal{H}_3$. An analogous reasoning holds for F, thus $\mathcal{A}$ has the same success probability also in $\mathcal{H}_4$. Finally, the proof in case of $\mathcal{H}_4$ is the same as for FSS.WOTS, but the probability decreases by a factor N as the more instances of FSS.WOTS, the higher the probability of one of them having only one preimage. In details, given $N \cdot \ell(w-1)$ points chosen at random, the probability of all having more than one preimage is greater than $1 - 2N \cdot \ell(w-1)\exp(-2^c)$. Thus, the probability that a honest signer can generate a proof of forgery is greater than $(1 - 2N\ell(w-1)\exp(-2^c)) \cdot \frac{1}{2}$ that is approximately $\frac{1}{2} - \mathsf{negl}(\lambda_s)$ analogously to the FSS.WOTS case, as in practice h is set to be smaller than 9, in which case the probability is greater than $1/2 - 2^{-73}$ for $c = 6$.

Lemma 10 (Security for the Recipient). *If* FSS.WOTS *is secure for the recipient, and H and F are* SM-TCR *secure THFs, then* FSS.XMSS *is secure for the recipient.*

Proof (sketch). The proof follows the same intuition of the proof of Lemma 8, in that a successful adversary $\mathcal{A}$ has to return either a collision for H or F, or a valid proof of forgery for FSS.WOTS for a honestly generated signature. As such, as long as both the building blocks are secure, FSS.XMSS is secure for the recipient.

7.1 Multiple instances of FSS.XMSS: the Hypertree

As we have seen, FSS.XMSS can be used to sign a limited amount of messages. The technique used in SPHINCS$^+$ to extend it to larger message spaces is hypertrees: small-depth FSS.XMSS trees connected to one another by FSS.WOTS signatures. In details, during key generation the signer has to generate a single FSS.XMSS tree. To expand its signing capabilities, whenever signing a message it can generate a new FSS.XMSS tree, and connect it to the previous one by signing the new root with one of the pk's of FSS.WOTS contained in the leaves of the original tree. This can be iterated for many layers, depending on the length of the message; the tree in the last layer is used to sign the message as in standard FSS.XMSS. Analogously to what happens in SPHINCS$^+$, security follows from the security of the building blocks: a forgery on the hypertree yieldsa forgery on one of the FSS.WOTS signatures. Observe that this is still a stateful signature! To get rid of the state SPHINCS$^+$ relies on the Hash-and-Sign paradigm, which turns out to be quite problematic in the FSS framework (cf. Section 8).

8 Augmenting Hash-and-Sign Schemes to an FSS

To support arbitrary size messages, practical digital signature schemes usually follow the Hash-and-Sign paradigm . In Hash-and-Sign, the message $m \in \{0,1\}^*$ is first hashed into a fixed size digest $d = HASH(m)$, which is signed as $\sigma = SIGN_{\mathsf{sk}}(d) = SIGN_{\mathsf{sk}}(HASH(m))$. It is advisable for the signer to add a random prefix to the message before computing $HASH(R||m)$, limiting even an adversary that can choose signed messages to finding a second preimage to break the scheme instead of a collision.

When trying to augment any Hash-and-Sign based signature to an FSS, we need to address the following generic forgery attack on the initial HASH phase. Our adversary can try to find a new message m^* and randomness R^* such that $HASH(R^*||m^*) = HASH(R||m)$, where m is any of the messages previously signed by the honest signer, and R is the randomness used. As the digest of both $R||m$ and $R^*||m^*$ is the same, σ will also be a valid signature for m^*. Assuming that the size of the digest is approximately the security parameter λ used by the digital signature, we can assume that an adversary that is able to forge the digital signature can also find such values R^* and m^*. We note that in some signature schemes such as FORS and SPHINCS$^+$, the adversary can also target some "interleaved" combination of the previously signed hash values. The problem is called Interleaved Target Subset Resilience (ITSR) (cf. Appendix D.2).

Augmenting Hash-and-Sign signatures to FSS is a very challenging task. We now present two possible solutions, and leave improving them as future work.

Saving a log file: One possible solution is for the honest signer to keep a log of all previously signed messages. In this case, when a forged signature (R^*, m^*, σ) is presented to the signer, it can search the corresponding honest signature (R, m, σ) in the log file and show that $HASH(R^*||m^*) = HASH(R||m)$ as a proof of forgery. However, the use of a log file raises the following question:

Doesn't using a log file means that the FSS augmentation results in a **state-full signature scheme***?*

For that, the answer is **no**. In state-full signature scheme such as XMSS, the state must be kept online, and if is lost, it can lead to a complete compromise of the scheme. Moreover, if multiple servers share the same secret key, they must also share exact up-to-date replicas of the state. However, in our case, the log can be stored in an offline storage, and the multiple servers sharing the same secret key can store their logs separately without online synchronization. Most importantly, if any part of the log is lost, the only result is that the signer will not be able to prove forgeries targeting the lost messages. As the main goal of our FSS augmentation is to stop mass exploitation of a forgery attack, as long as enough log files are stored, we will still be able to detect some forgery attacks and show that the scheme is now insecure.

We note that already today, we have relevant use cases where a log of all signatures is kept. For example, Certificate Authority (CA) servers are trusted parties that issue digital certificates to authenticate the identities of individuals, organizations, or devices for secure communication over a network or the internet. For audit purposes, such CAs usually log all of the certificates they sign. Widely supported standards such as Certificate Transparency [35] already provide an open framework for publicly logging and monitoring the issuance of digital certificates, offering a comprehensive log file of all signed certificates for improved transparency and security. As mentioned above, for some signature schemes such as FORS and SPHINCS$^+$, the adversary can target some "interleaved" combination of the previously signed digest values. The log file can also be used to prove forgery in this case, providing the set of honest messages that were "interleaved" to match the target forged digest. In case we do not want to log the signed message (as it might contain sensitive information or due to size constraints), we can slightly modify the hash phase of the signing process such that we will only need to store an intermediate randomized digest value of the message. Based on our fine-grained assumption, we will use an intermediate *strong* hash function $HASH' : \{0,1\}^* \to \{0,1\}^{c_s \cdot \lambda_r}$, this hash function as a larger digest and we assume that even our strong adversary can't break it. Our modified signing process will be $\sigma = SIGN_{\mathsf{sk}}(HASH(HASH'(R||M)))$. In our log file we will only need to store $d' = HASH'(R||M)$. Note that we assume that the adversary cannot find a collision on d', but it can find a collision on $d = HASH(HASH'(R||M))$.

c_s Parallel Signatures: To avoid the log file requirement, we propose another solution that is based on the fine-grained assumption of the adversary capabilities presented in Section 6.1. Recall that we assume that our (exponential-time) adversary can break the security assumptions of our scheme (including ITSR) for some security parameter λ. However, we assume this adversary is not powerful enough to break our security assumption of a larger security parameter $c_s \cdot \lambda$. To use this assumption, our FSS version will now include c_s signatures (or $\lceil c_s \rceil$ if $c_s \notin \mathcal{Z}$). We use a variant of the method used in [7] to calculate c_s separate digests and then sign them:

$$d_i = HASH(i||m) \qquad \sigma_i = SIGN(d_1) \qquad \sigma_{FSS} = \sigma_1||\sigma_2||\dots||\sigma_{c_s}$$

Similar to the analysis in Section 6.1, finding a message m^* such that all its c_s digests has been signed before becomes exponentially harder as c_s increases.

The proposed solution increases the running time and size of the signature by a factor of c_s. This raises the following question:

Why can't we just use the original signatures scheme with larger parameters?

As we mentioned above, increasing the security parameter of original scheme by the same factor of c_s can lead to a much larger increase in size. Recall that increasing the security parameter for SPHINCS$^+$ by a factor of 2 from 128 to 256 bits results in a size increase by a factor of 3.8 for the small size variant. This means that our solution can still result in a much smaller signature size.

9 Augmenting FORS to an FSS

FORS is the final building block needed for SPHINCS$^+$. In itself, it is very simple: the public key is the hash $G := \mathbf{H}_k$ (defined in Section 6.3) of the roots of k trees of depth d constructed using $H := \mathbf{H}_2$, whose leaves are obtained computing a THF $T := \mathbf{Th}_{w-1}$ (from the family $\mathbf{Th}$ of compressing THF described in Section 5) on the secret keys. The secret keys are output by a PRF PRF_2 evaluated on a secret seed and on its address in the tree. Signing a message requires splitting it into k blocks of d bits, and to interpret the i^{th} block as the address of a leaf in the i^{th} tree: a signature on the i^{th} block is the preimage of such a leaf, and its authentication path. The signature on the full message is the collection of all the signatures on the blocks. Finally, to avoid forgeries through recombination one needs to bound all the paths together, hence the digest to be signed is obtained as the hash of the messages (with the public key, the seed of T, and some message dependent randomness computed with a PRF $\mathsf{PRF}_{\mathsf{msg}}$) through a function $H_{\mathsf{msg}} : \{0,1\}^{\lambda_r} \times \{0,1\}^{\lambda_r} \times \mathcal{P} \times \mathcal{M} \to \{0,1\}^{dk}$. As the structure of FORS is extremely similar to XMSS, the fail-stop mechanism is integrated analogously: leaves are generated using a (strongly) compressing THF so that the adversary cannot recover the same preimages used by the signer (with constant probability). Trees are generated like in FSS.XMSS, thus we assume that a powerful adversary can reconstruct the from the public information. The formal description of the protocol can be found in Protocol 5.

On Hash-and-Sign, Fine-grained Assumptions, and Adaptivity. As FSS.FORS is a Hash-and-Sign style signature, its security requires either assuming ITSR security of H_{msg}, or using one of the workarounds presented in Section 8. For the sake of clarity we assume here that the adversary is not powerful enough to break ITSR, and deal with the other cases separately (cf. Section 8). Observe that assuming ITSR and *adaptive* SM-TCR allows to prove *adaptive* unforgeability without complexity leveraging (analogously to SPHINCS$^+$).

Lemma 11 (Security of FSS.FORS).

- *If T, G, and H are* adaptive *SM-TCR secure compressing THFs, PRF_2 and $\mathsf{PRF}_{\mathsf{msg}}$ are PRFs, and H_{msg} is a ITSR secure compressing THF, then* FSS.FORS *is unforgeable under adaptive CPA.*
- *Assume that $\mathcal{A}$ cannot break the ITSR security of H_{msg} nor invert PRF_2 and $\mathsf{PRF}_{\mathsf{msg}}$. If T is a compressing THF, then* FSS.FORS *is secure for signer against an adversary $\mathcal{A}$ with running time at most $2^{c_s \lambda_s/2}$ (in the QROM).*
- *If T, G, H are SM-TCR secure THFs,* FSS.FORS *is secure for the recipient.*

Proof. The unforgeability proof follows closely the reduction in [31, Theorem 3]. Hence we only include a sketch, and defer the proof to the full version.

$\mathcal{H}_1$**:** this is the *adaptive* $\mathsf{uf} - \mathsf{cma}$ experiment.
$\mathcal{H}_2$**:** Same as $\mathcal{H}_1$, but the sk's are random strings instead of output by PRF_2.
$\mathcal{H}_3$**:** Same as $\mathcal{H}_2$, but the output of $\mathsf{PRF}_{\mathsf{msg}}$ is replaced with random strings.
$\mathcal{H}_4$**:** Same as $\mathcal{H}_3$, but now the adversary looses if the signature is obtained combining k authentication paths that were already output by the signer during the querying phase.
$\mathcal{H}_5$**:** Same as $\mathcal{H}_4$, but now $\mathcal{A}$ looses if a FORS leaf in the forgery is different from the leaf that the signer would generate for that place.

Distinguishing $\mathcal{H}_1$ and $\mathcal{H}_2$ (resp., $\mathcal{H}_2$ and $\mathcal{H}_3$) requires breaking the pseudorandomness of PRF_2 (resp., $\mathsf{PRF}_{\mathsf{msg}}$). Distinguishing $\mathcal{H}_3$ from $\mathcal{H}_4$ requires breaking the ITSR property of H_{msg}, and the proof is analogous to the proof of [10, Claim 21]). Distinguishing $\mathcal{H}_4$ from $\mathcal{H}_5$ requires breaking the SM-TCR security of H. Finally, winning $\mathcal{H}_5$ requires breaking the SM-TCR security of either T or G.

Security for the signer and the receiver can be proved analogously to FSS.XMSS.

10 Augmenting SPHINCS$^+$ to an FSS

Augmenting SPHINCS$^+$ to an FSS just requires keeping its structure as is, and replacing XMSS by FSS.XMSS, FORS by FSS.FORS. We include a high level description of signing for those who are not familiar with SPHINCS$^+$ (cf. Appendix E for details). Essentially, (the hash of) a message is interpreted as the address idx of a leaf in the hypertree, and a message digest MD. The FSS.WOTS pk contained in such leaf is used to sign an FSS.FORS public key, which in turn is used to sign MD. A signature on the message includes then the randomness used in H_{msg}, the authentication path and FSS.WOTS signatures needed to go from the public root to the FSS.FORS key, and the FSS.FORS signature on MD. Differences arise if one does not want to use the fine-grained assumption that an adversary cannot break ITSR security. In such a case one has to use one of our generic augmentation of Hash-and-Sign to an FSS (cf. Section 8). Let PRF be the PRF used to generate the secret keys of FSS.WOTS and FSS.FORS.

Lemma 12 (Security of FSS.SPHINCS).

- *If* $\mathbf{Th} = \{\mathbf{Th}_i\}_i$ *is a family of THFs that is SM-UD, SM-TCR, and SM-PRE secure,* PRF, $\mathsf{PRF}_{\mathsf{msg}}$ *are PRFs,* $\mathbf{H} = \{\mathbf{H}_i\}_i$ *is a family of SM-TCR secure THFs, and* H_{msg} *is a ITSR secure compressing THF, then* FSS.SPHINCS *is unforgeable under adaptive CPA.*
- *Assume that* $\mathcal{A}$ *cannot break the ITSR security of* H_{msg} *nor invert* PRF *and* $\mathsf{PRF}_{\mathsf{msg}}$*. If* **Th** *and* **H** *are families of compressing THF, then* FSS.SPHINCS *is secure for signer against an adversary* $\mathcal{A}$ *with running time at most* $2^{c_s\lambda_s/2}$ *(in the QROM).*
- *If* **Th** *and* **H** *are families of SM-TCR secure THFs, then* FSS.SPHINCS *is secure for the recipient.*

Proving unforgeability can be done with the same reduction as in the NIST specifications (see Appendix G). Security for the signer and for the receiver can be proved analogously to what done for FSS.XMSS.

10.1 Parameters Choice for FSS.SPHINCS

We now discuss the cost of the fail-stop mechanism, taking the small 128-bits security variant of SPHINCS$^+$ (SPHINCS$^+$-128s) as our use case and computing the cost of augmenting it into an FSS.SPHINCS with 128-bit security, and with 256-bit security for the signer. Table 1 summarizes the results. Our augmentation does not affect the size of the public key, that remains 256 bits (128-bits seed and the 128-bits Merkle tree root). The secret key size increases multiplicatively by the expansion factor c_s. To get 256 bits of security for the PRF, it is enough to set $c_s = 256/\lambda_r = 2$. The size of the secret key is then $|PK| + 2 \cdot c_s \cdot \lambda_r = 768$ bits instead of the 512 bits of SPHINCS$^+$-128s (but smaller than the secret key of SPHINCS$^+$-256s). Hence we focus on computing the expansion in the signature size, which depends on the value of the compression factor.

Value of Compression Factor *c*. Recall that in SPHINCS$^+$, the entire hypertree (including all WOTS$^+$ and XMSS signatures, and also all of FORS signatures' Merkels trees) is deterministically derived from the private keys and seed, and can be considered as public information as it might be revealed as part of the benign signing process. The messages (and optional randomness) determine which private leaves of the FORS trees will be opened as part of the signatures. To simplify our calculation, we use the worst-case assumption that the adversary learns the entire hypertree structure of the SPHINCS$^+$ signature, and all FORS signatures' Merkle trees. [8]As another worst-case assumption, we assume that the adversary can forge a signature without being detected, as long as the is at least a single published value in any chain in any WOTS$^+$ signature,[9] or a single leaf in any FORS Merkle tree, that has only one preimage. We will now calculate the probability of such an event to occur in FSS.SPHINCS-128s (the augmentation of SPHINCS$^+$-128s to an FSS) instance as a function of the compression factor c. Overall, the number of values N_{val} we want to have more then

[8] In practice, the number of the hypertee leaves is close to the number of allowed signatures, thus a large fraction of the leaves will not be revealed.

[9] Unlike the case of WOTS$^+$ and XMSS, here $\mathcal{A}$ cannot chose the messages signed with WOTS$^+$, so it can only target the single value in the chain that was opened.

one preimage is the the sum of the number of leaves for FORS and the total number of chains in all of the WOTS^+ signatures,

$$N_{\mathsf{FORS}} = 2^h; N_{\mathsf{WOTS}^+} = 2^h \cdot (1 + 2^{-h/d} + 2^{-2h/d} + 2^{-3h/d} ...) < 2 \cdot 2^h$$

$$N_{val} = N_{\mathsf{FORS}} \cdot t \cdot k + N_{\mathsf{WOTS}^+} \cdot l \approx 2^h \cdot t \cdot k$$

Where h is the height of the hyper-tree, d is the height of the XMSS tree, t is the number of leaves in a FORS tree, k is the number of trees in FORS, and l is the number of chains in a WOTS^+ signatures. For $\mathsf{SPHINCS}^+$-128s $N_{val} \approx 2^{63} \cdot 2^{12} \cdot 14 < 2^{79}$. The probability that at least one of these points has one preimage can be bounded using Lemma 7 as $\varepsilon := N_{val} \cdot 2 \exp(-2^c)$. Hence, $c = 8$ is enough to have $\varepsilon \approx 2^{-368} \ll 2^{-128}$. We can now bound the number of additional bytes for FSS.SPHINCS by $c \cdot (k + d \cdot l \cdot w)$. For FSS.SPHINCS-128s with $c = 8$ this results in $8 \cdot (14 + 7 \cdot 35 \cdot 16)\texttt{bits} = 3934\texttt{bytes}$. In the case of the variant that returns parallel signatures, this number is doubled.

Acknowledgements

We thank Oded Golderich, Yehuda Lindell, and Guy Rothblum for valuable discussions and insights. We also thank Noga Amit, Gal Arnon, and Matan Hamilis for reading a preliminary version of this document. Cecilia Boschini has been supported by the Università della Svizzera Italiana under the SNSF project No. 182452, and by the Postdoc.Mobility grant No. P500PT_203075. Hila Dahari is a fellow of the Ariane de Rothschild Women Doctoral Program and supported in part by grants from the Israel Science Foundation (No. 950/15 and 2686/20) and by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness. Work done in part while a visitor at the FACT Research Center at Reichman University, supported in part by AFOSR Award FA9550-21-1-0046. Moni Naor is supported in part by grants from the Israel Science Foundation (no.2686/20), by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness and by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center. Eyal Ronen is partly supported by ISF grant no. 1807/23 and the Len Blavatnik and the Blavatnik Family Foundation.

Protocol 5: FSS.FORS

Let k, d, c_s be public parameters.
Let $\mathsf{PRF}_2 : \{0,1\}^{c_s\lambda_r} \times \mathcal{T} \to \{0,1\}^{\lambda_r+c}$, $\mathsf{PRF}_{\mathsf{msg}} : \{0,1\}^{c_s\cdot\lambda_r} \times \mathcal{T} \to \{0,1\}^{\lambda_r}$ be two PRFs.
Let $T : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{\lambda_r+c} \to \{0,1\}^{\lambda_r}$, $H : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{2\lambda_r} \to \{0,1\}^{\lambda_r}$, $G : \mathcal{P} \times \mathcal{T} \times \{0,1\}^{k\lambda_r} \to \{0,1\}^{\lambda_r}$ be THFs.
Let $T_{i,j,k}$ be the tweak of ith message in jth level in the kth tree. The height of the root is 1, and of the leaves is d.
Let $\mathcal{M}$ be the message space and $H_{\mathsf{msg}} : \{0,1\}^{\lambda_r} \times \{0,1\}^{\lambda_r} \times \mathcal{P} \times \mathcal{M} \to \{0,1\}^{dk}$ be a hash.

$\mathsf{GenCh}(1^{\lambda_r}, 1^{\lambda_s}, 1^{c_s}, 1^k, 1^d)$:
1: $t \leftarrow 2^d$
2: Set c so that $(k \cdot t)\frac{2^{(c_s\lambda_r-1)}}{2^{c_s\lambda_r-1}} \exp(-2^c) \le 2^{-\lambda_s}$.
3: $\mathsf{Seed}_H \xleftarrow{\$} \mathcal{P}$
4: $\mathsf{Seed}_T \xleftarrow{\$} \mathcal{P}$
5: $\mathsf{ch} \leftarrow (k, c, d, \mathsf{Seed}_T, \mathsf{Seed}_H)$
6: **return** ch

$\mathsf{GenKey}(\mathsf{ch})$:
1: Parse $\mathsf{ch} = (k, c, d, \mathsf{Seed}_T, \mathsf{Seed}_H)$
2: $t \leftarrow 2^d$
3: $\mathsf{Seed}_{\mathsf{sk}} \xleftarrow{\$} \{0,1\}^{c_s\lambda_r}$ // key to PRF_2
4: $\mathsf{Seed}_{\mathsf{msg}} \xleftarrow{\$} \{0,1\}^{c_s\lambda_r}$ // key to $\mathsf{PRF}_{\mathsf{msg}}$
5: **for** $i = 1, \ldots, k$ **do**
6: **for** $j = 1, \ldots, t$ **do**
7: $\mathsf{sk}_{i,j} \leftarrow \mathsf{PRF}_2(\mathsf{Seed}_{\mathsf{sk}}, \mathsf{adrs}_{i,j})$
8: $\mathsf{pk}_{i,j} \leftarrow T(\mathsf{Seed}_T, \mathsf{adrs}_{i,j}, \mathsf{sk}_{i,j})$
9: $\mathsf{root}_i \leftarrow \mathsf{Tree}_{\mathsf{H}}(\mathsf{pk}_{\mathsf{i},1}, \ldots, \mathsf{pk}_{\mathsf{i,t}}, \mathsf{Seed}_{\mathsf{H}})$
10: $\mathsf{sk} \leftarrow (\mathsf{Seed}_{\mathsf{sk}}, \mathsf{Seed}_{\mathsf{msg}})$
11: Set $T_{0,0,0}$ according to specs
12: $\mathsf{pk} \leftarrow G(\mathsf{Seed}_H, T_{0,0,0}, (\mathsf{root}_1, \ldots, \mathsf{root}_k))$
13: **return** $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}_{\mathsf{sk}}(m)$:
1: Parse $\mathsf{sk} = (\mathsf{Seed}_{\mathsf{sk}}, \mathsf{Seed}_{\mathsf{msg}})$
2: $R \leftarrow \mathsf{PRF}_{\mathsf{msg}}(\mathsf{Seed}_{\mathsf{msg}}, m)$
3: $(m_1, \ldots, m_{k\cdot d}) \leftarrow H_{\mathsf{msg}}(R, \mathsf{pk}, \mathsf{Seed}_T, m)$
4: **for** $i = 1, \ldots, k$ **do**
5: $\mathsf{idx}_i \leftarrow [m_{1+(i-1)\cdot d}, \ldots, m_{i\cdot d}]$
6: **for** $j = 1, \ldots, t$ **do**
7: $\mathsf{sk}_{i,j} \leftarrow \mathsf{PRF}_2(\mathsf{Seed}_{\mathsf{sk}}, \mathsf{adrs}_{i,j})$
8: $\mathsf{pk}_{i,j} \leftarrow T(\mathsf{Seed}_T, \mathsf{adrs}_{i,j}, \mathsf{sk}_{i,j})$
9: $\mathsf{auth}_i \leftarrow \mathsf{TreeGenAuth}_{\mathsf{H}}(\mathsf{pk}_{\mathsf{i},1}, \ldots, \mathsf{pk}_{\mathsf{i,t}}, \mathsf{idx}_{\mathsf{i}}, \mathsf{Seed}_{\mathsf{H}})$
10: $\mathsf{auth} \leftarrow (\mathsf{auth}_1, \mathsf{sk}_{1,\mathsf{idx}_1}, \ldots, \mathsf{auth}_k, \mathsf{sk}_{k,\mathsf{idx}_k})$
11: **return** (R, auth).

$\mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$:
1: Parse $\sigma = (R, \mathsf{auth}_1, \mathsf{sk}_1, \ldots, \mathsf{auth}_k, \mathsf{sk}_k)$
2: $(m_1, \ldots, m_{k\cdot d}) \leftarrow H_{\mathsf{msg}}(R, \mathsf{pk}, \mathsf{Seed}_T, m)$
3: **for** $i = 1, \ldots, k$ **do**
4: Parse $\mathsf{auth}_i = (\mathsf{pk}_i, \bar{\mathsf{auth}}_i, \mathsf{root}_i)$
5: $\mathsf{idx}_i \leftarrow [m_{1+(i-1)\cdot d}, \ldots, m_{i\cdot d}]$
6: Generate adrs from (i, idx_i)
7: $a_i \leftarrow (\mathsf{pk}_i = T(\mathsf{Seed}_T, \mathsf{adrs}, \mathsf{sk}_i))$
8: $b_i \leftarrow \mathsf{VrfyAuth}_{\mathsf{H}}(\mathsf{auth}_{\mathsf{i}}, \mathsf{Seed}_{\mathsf{H}})$
9: $c_i \leftarrow a_i \wedge b_i$
10: $\mathsf{pk}' \leftarrow G(\mathsf{Seed}_H, T_{0,0,0}, (\mathsf{root}_1, \ldots, \mathsf{root}_k))$
11: **if** $\mathsf{pk} = \mathsf{pk}'$ **then return** $c_1 \wedge \ldots \wedge c_k$
12: **else return** 0.

$\mathsf{PoF}_{\mathsf{sk}}((m, \sigma), \mathsf{ch})$:
1: **if** $0 \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$ **then return** $\perp$
2: $\sigma' \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$
3: **if** $\sigma' = \sigma$ **then return** $\perp$
4: Parse $\sigma = (R, \mathsf{auth}_1, \mathsf{sk}_1, \ldots, \mathsf{auth}_k, \mathsf{sk}_k)$
5: Parse $\sigma' = (R', \mathsf{auth}'_1, \mathsf{sk}'_1, \ldots, \mathsf{auth}'_k, \mathsf{sk}'_k)$
6: $(m_1, \ldots, m_{k\cdot d}) \leftarrow H_{\mathsf{msg}}(R, \mathsf{pk}, \mathsf{Seed}_T, m)$
7: **for** $i = 1, \ldots, k$ **do**
8: Parse $\mathsf{auth}_i = (\mathsf{pk}_i, \bar{\mathsf{auth}}_i, \mathsf{root}_i)$
9: Parse $\mathsf{auth}'_i = (\mathsf{pk}'_i, \bar{\mathsf{auth}}'_i, \mathsf{root}'_i)$
10: $\mathsf{idx}_i \leftarrow [m_{1+(i-1)\cdot d}, \ldots, m_{i\cdot d}]$
11: **if** $(\mathsf{pk}_i = \mathsf{pk}'_i) \wedge (\mathsf{sk}_i \neq \mathsf{sk}'_i)$ **then**
12: **return** $(1, \mathsf{sk}_i, \mathsf{sk}'_i, i, \mathsf{idx}_i)$
13: **if** $\mathsf{root}_i \neq \mathsf{root}'_i$ **then**
14: Set $T_{0,0,0}$ according to specs.
15: **return** $(2, T_{0,0,0}, \{\mathsf{root}_j, \mathsf{root}'_j\}_{j=1,\ldots,k})$
16: $\mathsf{forgery} \leftarrow \mathsf{CTree}_{\mathsf{H}}(\mathsf{root}_{\mathsf{i}}, (\mathsf{pk}_{\mathsf{i}}, \bar{\mathsf{auth}}_{\mathsf{i}}), (\mathsf{pk}'_{\mathsf{i}}, \bar{\mathsf{auth}}'_{\mathsf{i}}), \mathsf{idx}_{\mathsf{i}}, \mathsf{Seed}_{\mathsf{H}})$
17: **if** $\mathsf{forgery} \neq \perp$ **then return** $(3, \mathsf{forgery})$
return $\perp$

$\mathsf{VrfyPoF}(\mathsf{ch}, \pi)$:
1: Parse $\mathsf{ch} = (k, c, d, \mathsf{Seed}_T, \mathsf{Seed}_H)$
2: **if** $\pi = \perp$ **then return** 0
3: **else**
4: Parse $\pi = (b, \pi')$
5: **if** $b \notin \{1, 2, 3\}$ **then return** 0
6: **if** $b = 1$ **then**
7: Parse $\pi' = (\mathsf{sk}_i, \mathsf{sk}'_i, i, \mathsf{idx}_i)$
8: Generate adrs from (i, idx_i)
9: $x_0 \leftarrow T(\mathsf{Seed}_T, \mathsf{adrs}, \mathsf{sk}_i)$
10: $x_1 \leftarrow T(\mathsf{Seed}_T, \mathsf{adrs}, \mathsf{sk}'_i)$
11: $b' \leftarrow (x_0 = x_1)$
12: **if** $b = 2$ **then**
13: Parse $\pi' = (T, \{\mathsf{root}_j, \mathsf{root}'_j\}_{j=1,\ldots,k})$
14: $x_0 \leftarrow G(\mathsf{Seed}_H, T, (\mathsf{root}_1, \ldots, \mathsf{root}_k))$
15: $x_1 \leftarrow G(\mathsf{Seed}_H, T, (\mathsf{root}'_1, \ldots, \mathsf{root}'_k))$
16: $b' \leftarrow (x_0 = x_1)$
17: **if** $b = 3$ **then**
18: Parse $\pi' = (T^0, (n_0^0, n_1^0), T^1, (n_0^1, n_1^1))$
19: $x_0 \leftarrow T(\mathsf{Seed}_T, T^0, (n_0^0, n_1^0))$
20: $x_1 \leftarrow T(\mathsf{Seed}_T, T^1, (n_0^1, n_1^1))$
21: $b' \leftarrow (x_0 = x_1)$
return b'

References

[1] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 5–17. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813707

[2] Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_28

[3] Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D., Liu, Y.K.: Status report on the third round of the NIST post-quantum cryptography standardization process. Available at https://csrc.nist.gov/publications/detail/nistir/8413/final (2022), accessed: 2022-10-07

[4] Alperin-Sheriff, J.: Short signatures with short public keys from homomorphic trapdoor functions. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 236–255. Springer, Heidelberg (Mar / Apr 2015). https://doi.org/10.1007/978-3-662-46447-2_11

[5] Alwen, J., Peikert, C.: Generating shorter bases for hard random lattices. Theory Comput. Syst. **48**(3), 535–553 (2011). https://doi.org/10.1007/s00224-010-9278-3, https://doi.org/10.1007/s00224-010-9278-3

[6] Aumasson, J.P., Bernstein, D.J., Beullens, W., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hüulsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Westerbaan, B.: SPHINCS$^+$. submission to NIST's post-quantum crypto standardization project, v.3. Available at http://sphincs.org/data/sphincs+-round3-specification.pdf (2022), accessed: 2022-10-05

[7] Aviram, N., Dowling, B., Komargodski, I., Paterson, K.G., Ronen, E., Yogev, E.: Practical (post-quantum) key combiners from one-wayness and applications to TLS. IACR Cryptol. ePrint Arch. p. 65 (2022), https://eprint.iacr.org/2022/065

[8] Bari, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_33

[9] Ben-Sasson, E., Goldberg, L., Levit, D.: STARK friendly hash – survey and recommendation. Cryptology ePrint Archive, Report 2020/948 (2020), https://eprint.iacr.org/2020/948

[10] Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS$^+$ signature framework. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2129–2146. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3363229

[11] Beullens, W.: Breaking rainbow takes a weekend on a laptop. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 464–479. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_16

[12] Böhl, F., Hofheinz, D., Jager, T., Koch, J., Striecks, C.: Confined guessing: New signatures from standard assumptions. Journal of Cryptology **28**(1), 176–208 (Jan 2015). https://doi.org/10.1007/s00145-014-9183-z

[13] Boschini, C., Camenisch, J., Ovsiankin, M., Spooner, N.: Efficient post-quantum SNARKs for RSIS and RLWE and their applications to privacy. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 247–267. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-44223-1_14

[14] Boyen, X.: Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 499–517. Springer, Heidelberg (May 2010). https://doi.org/10.1007/978-3-642-13013-7_29

[15] Boyen, X., Li, Q.: Towards tightly secure lattice short signature and id-based encryption. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 404–434. Springer, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53890-6_14

[16] Buchmann, J.A., Dahmen, E., Hülsing, A.: XMSS - A practical forward secure signature scheme based on minimal security assumptions. In: Yang, B.Y. (ed.) Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011. pp. 117–129. Springer, Heidelberg (Nov / Dec 2011). https://doi.org/10.1007/978-3-642-25405-5_8

[17] Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. CoRR **cs.CR/0010019** (2000), https://arxiv.org/abs/cs/0010019

[18] Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_27

[19] Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). IACR Cryptol. ePrint Arch. p. 975 (2022), https://eprint.iacr.org/2022/975

[20] Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 319–327. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34799-2_25

[21] Damgård, I., Pedersen, T.P., Pfitzmann, B.: On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 250–265. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_22

[22] Ducas, L., Micciancio, D.: Improved short lattice signatures in the standard model. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 335–352. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44371-2_19

[23] F. Boudot, P. Gaudry, A.G.N.H.E.T., Zimmermann, P.: Factorization of rsa-250 (2020), https://sympa.inria.fr/sympa/arc/cado-nfs/2020-02/msg00001.html

[24] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12

[25] Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: STOC. pp. 212–219. ACM (1996)

[26] Haitner, I., Hoch, J.J., Reingold, O., Segev, G.: Finding collisions in interactive protocols - a tight lower bound on the round complexity of statistically-hiding commitments. In: 48th FOCS. pp. 669–679. IEEE Computer Society Press (Oct 2007). https://doi.org/10.1109/FOCS.2007.27

[27] Haitner, I., Reingold, O.: Statistically-hiding commitment from any one-way function. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 1–10. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250792

[28] Halevi, S., Micali, S.: Practical and provably-secure commitment schemes from collision-free hashing. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 201–215. Springer, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_16

[29] Hülsing, A.: W-OTS+ - shorter signatures for hash-based signature schemes. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 13. LNCS, vol. 7918, pp. 173–188. Springer, Heidelberg (Jun 2013). https://doi.org/10.1007/978-3-642-38553-7_10

[30] Hulsing, A., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Kampanakis, P., Kolbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Aumasson, J.P., Westerbaan, B., Beullens, W.: SPHINCS+. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

[31] Hülsing, A., Kudinov, M.A.: Recovering the tight security proof of sphincs$^+$. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13794, pp. 3–33. Springer (2022). https://doi.org/10.1007/978-3-031-22972-5_1, https://doi.org/10.1007/978-3-031-22972-5_1

[32] Hülsing, A., Rijneveld, J., Song, F.: Mitigating multi-target attacks in hash-based signatures. In: Cheng, C.M., Chung, K.M., Persiano, G., Yang, B.Y. (eds.) PKC 2016, Part I. LNCS, vol. 9614, pp. 387–416. Springer, Heidelberg (Mar 2016). https://doi.org/10.1007/978-3-662-49384-7_15
[33] Kiktenko, E.O., Kudinov, M.A., Bulychev, A.A., Fedorov, A.K.: Proof-of-forgery for hash-based signatures. In: di Vimercati, S.D.C., Samarati, P. (eds.) Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021. pp. 333–342. SCITEPRESS (2021)
[34] Lamport, L.: Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (Oct 1979)
[35] Laurie, B., Messeri, E., Stradling, R.: Certificate transparency version 2.0. RFC 9162, RFC Editor (December 2021)
[36] Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 1–31. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_1
[37] Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_43
[38] Lyubashevsky, V., Micciancio, D.: Asymptotically efficient lattice-based digital signatures. Journal of Cryptology **31**(3), 774–797 (Jul 2018). https://doi.org/10.1007/s00145-017-9270-z
[39] Maino, L., Martindale, C.: An attack on SIDH with arbitrary starting curve. IACR Cryptol. ePrint Arch. p. 1026 (2022), https://eprint.iacr.org/2022/1026
[40] Mashatan, A., Ouafi, K.: Efficient fail-stop signatures from the factoring assumption. In: Lai, X., Zhou, J., Li, H. (eds.) Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7001, pp. 372–385. Springer (2011). https://doi.org/10.1007/978-3-642-24861-0_25, https://doi.org/10.1007/978-3-642-24861-0_25
[41] MATZOV: Report on the security of lwe: Improved dual lattice attack. Available at https://zenodo.org/record/6493704#.Yz_qSUpBxkg, accessed: 2022-10-07
[42] Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_21
[43] Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_41
[44] Micciancio, D., Peikert, C.: Hardness of SIS and LWE with small parameters. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 21–39. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_2
[45] Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. In: 45th FOCS. pp. 372–381. IEEE Computer Society Press (Oct 2004). https://doi.org/10.1109/FOCS.2004.72
[46] NIST: Call for additional digital signature schemes for the post-quantum cryptography standardization process. Available at https://csrc.nist.gov/projects/pqc-dig-sig/standardization/call-for-proposals (2022), accessed: 2022-10-07
[47] Pedersen, T.P., Pfitzmann, B.: Fail-stop signatures. SIAM J. Comput. **26**(2), 291–330 (1997). https://doi.org/10.1137/S009753979324557X, https://doi.org/10.1137/S009753979324557X
[48] Perlner, R., Kelsey, J., Cooper, D.: Breaking category five sphincs+ with sha-256. Cryptology ePrint Archive, Paper 2022/1061 (2022), https://eprint.iacr.org/2022/1061, https://eprint.iacr.org/2022/1061
[49] Pfitzmann, B.: Digital Signature Schemes, General Framework and Fail-Stop Signatures, Lecture Notes in Computer Science, vol. 1100. Springer (1996). https://doi.org/10.1007/BFb0024619, https://doi.org/10.1007/BFb0024619
[50] Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and

Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
[51] Rogaway, P.: The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162 (2015), https://eprint.iacr.org/2015/1162
[52] Safavi-Naini, R., Susilo, W., Wang, H.: Fail-stop signature for long messages. In: Roy, B.K., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 165–177. Springer, Heidelberg (Dec 2000)
[53] Safavi-Naini, R., Susilo, W., Wang, H.: An efficient construction for fail-stop signature for long messages. J. Inf. Sci. Eng. **17**(6), 879–898 (2001), http://www.iis.sinica.edu.tw/page/jise/2001/200111_02.html
[54] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172, https://doi.org/10.1137/S0097539795293172
[55] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 570–596. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7_19
[56] Susilo, W.: Short fail-stop signature scheme based on factorization and discrete logarithm assumptions. Theor. Comput. Sci. **410**(8-10), 736–744 (2009). https://doi.org/10.1016/j.tcs.2008.10.025, https://doi.org/10.1016/j.tcs.2008.10.025
[57] Susilo, W., Mu, Y.: Provably secure fail-stop signature schemes based on RSA. Int. J. Wirel. Mob. Comput. **1**(1), 53–60 (2005). https://doi.org/10.1504/IJWMC.2005.008055, https://doi.org/10.1504/IJWMC.2005.008055
[58] Susilo, W., Safavi-Naini, R.: An efficient fail-stop signature scheme based on factorization. In: Lee, P.J., Lim, C.H. (eds.) Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2587, pp. 62–74. Springer (2002). https://doi.org/10.1007/3-540-36552-4_5, https://doi.org/10.1007/3-540-36552-4_5
[59] Susilo, W., Safavi-Naini, R., Gysin, M., Seberry, J.: A new and efficient fail-stop signature scheme. Comput. J. **43**(5), 430–437 (2000). https://doi.org/10.1093/comjnl/43.5.430, https://doi.org/10.1093/comjnl/43.5.430
[60] Susilo, W., Safavi-Naini, R., Pieprzyk, J.: Rsa-based fail-stop signature schemes. In: Proceedings of the 1999 International Conference on Parallel Processing Workshops, ICPPW 1999, Wakamatsu, Japan, September 21-24, 1999. pp. 161–166. IEEE Computer Society (1999). https://doi.org/10.1109/ICPPW.1999.800056, https://doi.org/10.1109/ICPPW.1999.800056
[61] van Heijst, E., Pedersen, T.P., Pfitzmann, B.: New constructions of fail-stop signatures and lower bounds (extended abstract). In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 15–30. Springer, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4_2
[62] van Heyst, E., Pedersen, T.P.: How to make efficient fail-stop signatures. In: Rueppel, R.A. (ed.) EUROCRYPT'92. LNCS, vol. 658, pp. 366–377. Springer, Heidelberg (May 1993). https://doi.org/10.1007/3-540-47555-9_30
[63] Waidner, M., Pfitzmann, B.: The dining cryptographers in the disco - underconditional sender and recipient untraceability with computationally secure serviceability (abstract) (rump session). In: Quisquater, J.J., Vandewalle, J. (eds.) EUROCRYPT'89. LNCS, vol. 434, p. 690. Springer, Heidelberg (Apr 1990). https://doi.org/10.1007/3-540-46885-4_69
[64] Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_2
[65] Wikipedia contributors: Flame (malware) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Flame_(malware)&oldid=1121984346 (2022), [Online; accessed 24-January-2023]
[66] Yamakawa, T., Kitajima, N., Nishide, T., Hanaoka, G., Okamoto, E.: A short fail-stop signature scheme from factoring. In: Chow, S.S.M., Liu, J.K., Hui, L.C.K., Yiu, S. (eds.) Provable Security - 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8782, pp. 309–316. Springer (2014). https://doi.org/10.1007/978-3-319-12475-9_22, https://doi.org/10.1007/978-3-319-12475-9_22

Supplementary Material

A Preliminaries

A standard digital signature is a triplet of PPT algorithms $\Sigma = (\mathsf{Sign.KeyGen}, \mathsf{Sign.Sign}, \mathsf{Sign.Ver})$. Its security is usually defined through a security experiment for an adversary $\mathcal{A}$ and security parameter λ:

Experiment 3: $\mathsf{Exp}^{\mathsf{eu-acma}}_{\mathcal{A},\Sigma}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sign.KeyGen}(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})$
if $(\mathsf{Sign.Ver}_{\mathsf{pk}}(m^*, \sigma^*) = 1) \ \wedge \ (m^* \notin \mathcal{Q})$ **then**
    return 1
else return 0

$\mathcal{O}_{\mathsf{sk}}(m)$:
$\sigma \leftarrow \mathsf{Sign.Sign}_{\mathsf{sk}}(m)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup m$
return σ

Experiment 4: $\mathsf{Exp}^{\mathsf{eu-nacma}}_{\mathcal{A},\Sigma,q}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sign.KeyGen}(1^\lambda)$
$(\mathcal{Q} = \{m_1, ..., m_q\}, S) \leftarrow \mathcal{A}_1(1^\lambda)$
for $i = 1, \ldots, q$ **do**
    $\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m_i)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}_2(\mathsf{pk}, S, \{(m_i, \sigma_i)\}_{i=1}^q)$
if $(\mathsf{Sign.Ver}_{\mathsf{pk}}(m^*, \sigma^*) = 1) \ \wedge \ (m^* \notin \mathcal{Q})$ **then**
    return 1
else return 0

Definition 6. *A signature scheme* $\Pi = (\mathsf{Sign.KeyGen}, \mathsf{Sign.Sign}, \mathsf{Sign.Ver})$ *is* existentially unforgeable under an adaptive chosen-message attack*, or just* secure*, if for all PPT adversaries* $\mathcal{A}$*, there is a negligible function* negl *such that*

$$\Pr[\mathsf{Exp}^{\mathsf{eu-acma}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where the security experiment is defined in Experiment 3. To model non-adaptive *unforgeability,* $\mathcal{A}$ *has to declare* $\mathcal{Q}$ *before seeing* pk *as in Experiment 4. For* one-time *signatures (OTS), it is enough to set* $|\mathcal{Q}| = 1$*.*

Definition 7 (universal family of hash functions). *Let* S *and* T *be two sets. A family of functions* $\mathcal{H} = \{h_i : S \to T\}$ *is called a* universal family of hash functions *if for any two different elements* $s_1 \neq s_2$ *in* S*, and for any two elements* $t_1, t_2 \in T$ *we have:*

$$\Pr[h(s_1) = t_1 \wedge h(s_2) = t_2] = \frac{1}{T^2}$$

A function $\mu(\cdot)$ is negligible in n, or just negligible, if for every positive polynomial $p(\cdot)$ and all sufficiently large n's it holds that $\mu(n) < 1/p(n)$. We use PPT as shorthand for probabilistic polynomial time.

Definition 8 (one-way functions). *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *is called* one-way *(OWF) if the following two conditions hold:*

- *Easy to compute: There exists a polynomial-time algorithm* A *such that on input* x*,* A *outputs* $f(x)$*.*
- *Hard to invert: For every PPT algorithm* A*, there is a negligible function* $\mu(\cdot)$ *such that for all sufficiently large* n*'s it holds:*

$$\Pr[A(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \mu(n)$$

Definition 9 (collision-resistant hash functions). *Let* $\ell : \mathbb{N} \to \mathbb{N}$*. A collection of functions* $H = \{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)}$ *is called a* family of collision-resistant hash functions *(CRH) if there exists a PPT sampling algorithm* I *such that the following holds:*

- *There exists a polynomial-time algorithm that, given* r *and* x*, returns* $h_r(x)$
- *For every PPT algorithm* A*, there is a negligible function* $\mu(\cdot)$ *such that for all sufficiently large* n*'s it holds:*

$$Pr[A(I(1^n), 1^n) = (x, x') \wedge x \neq x' \wedge h_{I(1^n)}(x) = h_{I(1^n)}(x')] < \mu(n)$$

where the probability is taken over the coin tosses of I *and* A*.*

We recall the definitions of correlation intractability, initially proposed in [17].

Definition 10 (Sparse relations). *A binary relation $\mathcal{R}$ is* sparse *with respect to length parameters $\ell(n), m(n)$,if there is a negligible function $\mu(\cdot)$ such that for every $x \in \{0,1\}^{\ell(n)}$:*

$$\Pr_{y \in \{0,1\}^{m(n)}}[\mathcal{R}(x,y) = 1] \leq \mu(n)$$

Definition 11. *A family of functions $H = \{h_k : \{0,1\}^{\ell(n)} \to \{0,1\}^{m(n)}\}_{n \in \mathbb{N}}$ is* correlation intractable *(CI) if for all PPT adversary $\mathcal{A}$, for all sparse relations $\mathcal{R}$, there is a negligible function $\mu(\cdot)$ such that for all sufficiently large n's it holds:*

$$\Pr_{k \xleftarrow{\$} \mathcal{H}_n}[x \leftarrow A(k) : \mathcal{R}(x, h_k(x)) = 1] \leq \mu(n)$$

In the definition above, the sparse relations may not be efficiently recognizable.

Definition 12. *Let λ be a security parameter. Let $F : \{0,1\}^{k(\lambda)} \times \{0,1\}^m \to \{0,1\}^\ell$. We call a function to be a λ-*PRF secure *if for every $\mathcal{A}$ with running time $t_\mathcal{A}$ at most 2^λ, the PRF is secure.*

$$|\Pr[\mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{F_k(\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

where $\mathcal{O} : \{0,1\}^m \to \{0,1\}^\ell$ is a random function, and k is uniformly chosen.

B FSS are equivalent to Digital Signatures

In this section we prove that FSS are equivalent to (a subset of) digital signatures. The proof that FSS implies a digital signature is quite intuitive, and not new (cf. [47, Thm 3.1]). As the original proof is quite informal, we include a formal proof of the claim in Theorem 4 to be thorough.

The other side of the implication (cf. Theorem 5) is less intuitive, and is (to the best of our knowledge) a new result. Our result only shows the equivalence of FSS and a specific subset of digital signatures, thus it is not enough to conclude that FSS can be constructed from OWF. However, Theorem 5 is a powerful tool: for example, it is enough to show the existence of lattice-based FSS (cf. Appendix C).

Definition 13. *Let R_L be a relation, and let $L = \{x \mid \exists w \ s.t \ (x,w) \in R_L\}$ be an NP language. We say that L is a λ-*hard language *for $\lambda \in \mathbb{N}$, $\lambda > 0$ if the following conditions hold:*

- Easy to sample statements: *There exists a probabilistic polynomial-time sampler S_L that on input 1^λ outputs a statement x where $x \in \{0,1\}^\lambda$, and $x \in L$.*
- Witness intractability: *For every PPT algorithm $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\left|\Pr[(S_L(1^\lambda), \mathcal{A}(S_L(1^n), 1^\lambda)) \in R_L]\right| \leq \mathsf{negl}(\lambda) \ .$$

The construction of a signature from a FSS is quite straightforward, and it was already shown by Pedersen and Pfizmann [47]. We include the formal proof of this fact in the following as a warm-up.

Theorem 4 ([47, Thm 3.1]). *Let L be a λ_r-hard language for $\lambda_r \in \mathbb{N}$, $\lambda_r > 0$. Let FSS be a fail-stop signature that is unforgeable assuming L, ε-secure for the signer for $0 < \varepsilon \leq 1$, and secure for the recipient with parameters (λ_r, ε). Then, there exists a signature scheme $\Pi = (\mathsf{Sign.KeyGen}, \mathsf{Sign.Sign}, \mathsf{Sign.Ver})$ and a hard problem L as in Definition 13 such that Π is $\mathsf{uf-acma}$ secure under L, that is, there exists a strictly polynomial-time security reduction from forging a signature to solving L in the standard model.*

Proof. Let $FSS = (\mathsf{GenCh}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{VrfySig}, \mathsf{PoF}, \mathsf{VrfyPoF})$ be a secure FSS signature (i.e., it satisfies Definitions 4,3,2) for parameters (λ_r, λ_s), and let $\Pi = (\mathsf{Sign.KeyGen}, \mathsf{Sign.Sign}, \mathsf{Sign.Ver})$ be the digital signature trivially defined by the FSS (cf. Protocol 6).

We first prove that Π is a $\mathsf{uf-acma}$ secure signature.

Protocol 6: Signature from FSS

$\mathsf{Sign.KeyGen}(1^\lambda)$
1: Sets $\lambda_r \leftarrow \lambda$, $\varepsilon \leftarrow 2^{-\lambda}$
2: $\mathsf{ch} \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^\varepsilon)$
3: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenKey}(\mathsf{ch})$
4: **return** $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Sign.Sign}_{\mathsf{sk}}(m)$
1: $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$.
2: **return** σ.

$\mathsf{Sign.Ver}_{\mathsf{pk}}(m, \sigma)$
1: $b \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$
2: **return** b

Lemma 13. *If* $(\mathsf{GenCh}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{VrfySig}, \mathsf{PoF}, \mathsf{VrfyPoF})$ *is a secure FSS assuming* L*, then* $\Pi = (\mathsf{Sign.KeyGen}, \mathsf{Sign.Sign}, \mathsf{Si}$ *is a* $\mathsf{uf} - \mathsf{acma}$ *secure signature assuming* L*.*

Proof. Assume by contradiction that there exists a PPT adversary $\mathcal{A}$ that breaks the unforgeability of Π with probability $\varepsilon_{\mathcal{A}}$. We construct an adversary $\mathcal{B}$ that breaks the unforgeability of the FSS with the same probability. $\mathcal{B}$ simulates the unforgeability game for a standard signature, and plays the unforgeability game for FSS. Thus, it is given pk, ch and oracle access to $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$ and aims to output a valid forgery for FSS. $\mathcal{B}$ runs $\mathcal{A}$ on pk, ch and simulates the signing oracle by forwarding $\mathcal{A}$'s queries to the signing oracle for FSS. At the end of the game, $\mathcal{B}$ returns the pair (m^*, σ^*) returned by $\mathcal{A}$. Note that,

$$\varepsilon_{\mathcal{A}} = \Pr[\mathsf{Exp}^{\mathsf{eu-acma}}_{\mathcal{A},\Pi}(\lambda) = 1] = \Pr[\mathsf{Exp}^{\mathsf{unf}}_{\mathcal{B},FSS}(\lambda_s, \lambda_r) = 1] \leq \mathsf{negl}(\lambda_r) \ .$$

□

It remains to prove that Π has a strictly polynomial-time security reduction from a hard problem L as defined in Definition 13 in the standard model.

Lemma 14. *If FSS is* ε*-secure for the signer and secure for the recipient with parameters* (λ_r, ε)*, then forging a signature for* Π *can be reduced to solving some hard problem* L *as defined in Definition 13 with a strictly polynomial-time reduction in the standard model.*

Proof. Let $L = \{x \mid \exists \tau \text{ such that } \mathsf{VrfyPoF}(x, \tau) = 1)\}$, and let $\mathcal{A}$ be a (possibly unbounded) adversary that breaks the unforgeability of Π in polynomial-time with probability $\varepsilon_{\mathcal{A}}$. We construct the security reduction $\mathcal{R}$ that solves L as follows:

Algorithm 1 [Security Reduction]

1. *On input an instance* $x \in R_L$*,* $\mathcal{R}$ *sets* ch *to be* x *and generates the key pair* $(pk, sk) \leftarrow \mathsf{GenKey}(1^n, \mathsf{ch})$.
2. *It gives* pk *to* $\mathcal{A}$*, and simulates the signing oracle using* sk*.* $\mathcal{R}$ *stores the queries that* $\mathcal{A}$ *asked in* $\mathcal{Q}$*.*
3. *When* $\mathcal{A}$ *returns* (m^*, σ^*)*,* $\mathcal{R}$ *checks if* $m^* \in \mathcal{Q}$*. In that case it aborts. Otherwise, it runs* $\tau \leftarrow \mathsf{PoF}_{\mathsf{sk}}(m^*, \sigma^*)$*, and it returns* τ *as a witness for* x*.*

The reduction $\mathcal{R}$ runs in polynomial-time, as $\mathcal{A}$ runs in polynomial-time. As FSS is ε-secure for the signer, $\mathcal{R}$ is successful with probability

$$\Pr[(x, \tau) \in R_L] = 1 - \Pr[\mathsf{Exp}^{\mathsf{ss}}_{\mathcal{A},FSS}(\lambda_s, \lambda_r) = 1] \geq 1 - \varepsilon = 1 - \mathsf{negl}(\lambda).$$

The witness indistinguishability property of L follows from Lemma 1. □

We now proceed to prove the other side of the implication. The intuition behind the construction of a FSS from a digital signature is that the security reduction of the signature can be made into a proof-of-forgery algorithm in some specific cases. In particular, extracting a witness for the (computationally hard) language should not require to perform actions that are not possible in the real world, such as rewinding the adversary or programming random oracles[10].

[10] This does not exclude signatures proved secure in the Random Oracle Model (ROM) entirely, but only the ones whose security reduction requires to *program* the RO.

Theorem 5. *Let λ be the security parameter. Let $\Pi = (\mathsf{Sign.KeyGen}, \mathsf{Sign.Sign}, \mathsf{Sign.Ver})$ be a signature such that:*

- *there is a strictly polynomial-time, black-box security reduction $\mathcal{R}$ from forging a signature to a λ-hard problem L as defined in Definition 13 with non-negligible success probability,*
- *such reduction does not rewind the adversary, nor does it program random oracles, and*
- *for any $(\mathsf{sk}, \mathsf{pk})$ generated by the simulator in the reduction $\mathcal{R}$ it holds that $H(sk \mid pk, Hist) \geq (N+1)(\lambda - 1)$, where $Hist$ is the list of the N signatures generated by the honest signer.*

Then there exists a secure Fail-Stop signature scheme (FSS) for parameters $(\varepsilon, \lambda_r) = (\mathsf{negl}(\lambda), \lambda)$ where negl is a negligible function.

Proof. Saying that Π has a strictly polynomial-time security reduction (that does not rewind the adversary or programs random oracles) of forging a signature to a λ-hard problem L implies that there exists a reduction $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3)$ that converts with probability $1 - \varepsilon_{\mathcal{R}}$ a forgery generated by a PPT adversary $\mathcal{A}$ to a witness of a given $x \in L$ using a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ as follows:

- Upon receiving a statement $x \leftarrow S_L(1^\lambda)$, $\mathcal{R}_1$ gets $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{S}_1(x)$. It outputs pk and a private state $st = \mathsf{sk}$.
- $\mathcal{R}_2$ is given the state st, and interacts with the adversary $\mathcal{A}(\mathsf{pk})$ against unforgeability. $\mathcal{R}_2$ answers its signing queries by running $\mathcal{S}_2$ on input sk, and stores the queries in $\mathcal{Q}$. Upon receiving the pair (m^*, σ^*) output as a potentially forgery by $\mathcal{A}$, $\mathcal{R}_2$ runs $b \leftarrow \mathsf{Sign.Ver}_{\mathsf{pk}}(m^*, \sigma^*)$ and checks if $m^* \in \mathcal{Q}$. If both checks pass, it returns (m^*, σ^*) and the private state st, otherwise it aborts, that is, it outputs $(\perp, st)$.
- $\mathcal{R}_3$ is given the private state st, and a forgery (m^*, σ^*). It runs the simulator $w \leftarrow \mathcal{S}_3(x, \mathsf{pk}, (m^*, \sigma^*), \mathsf{sk})$ outputs w.

The success probability of such reduction can be computed as $(1 - \varepsilon_{\mathcal{R}})\varepsilon_{\mathcal{A}}$, where $\varepsilon_{\mathcal{A}}$ is the success probability of $\mathcal{A}$ in the unforgeability game.

Observe that from the simulator Sone can construct an FSS as in Protocol 7.

Protocol 7: FSS from Signature

$\mathsf{GenCh}(1^\lambda)$
1: Runs $\mathsf{ch} \leftarrow \mathcal{S}_L(1^\lambda)$
2: **return** ch

$\mathsf{GenKey}(\mathsf{ch})$
1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{S}_1(\mathsf{ch})$.
2: **return** $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}_{\mathsf{sk}}(m)$
1: $\sigma \leftarrow \mathcal{S}_2(\mathsf{sk}, \mathsf{ch}, m)$.
2: **return** σ.

$\mathsf{VrfySig}_{\mathsf{pk}}(m, \sigma)$
1: $b \leftarrow \mathsf{Sign.Ver}_{\mathsf{pk}}(m)$
2: **return** b

$\mathsf{PoF}_{\mathsf{sk}}(m^*, \sigma^*, \mathsf{ch})$
1: $\tau \leftarrow \mathcal{S}_3(\mathsf{ch}, \mathsf{pk}, (m^*, \sigma^*), \mathsf{sk})$
2: **return** τ

$\mathsf{VrfyPoF}(\mathsf{ch}, \tau)$
1: **return** 1 iff $(\mathsf{ch}, \tau) \in R_L$
2: **return** 0

In the following we prove that $(\mathsf{GenCh}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{VrfySig}, \mathsf{PoF}, \mathsf{VrfyPoF})$ satisfies ε-security for signer, security for recipient, and unforgeability.

Lemma 15. *If L is a λ-hard problem, the scheme $(\mathsf{GenCh}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{VrfySig}, \mathsf{PoF}, \mathsf{VrfyPoF})$ is unforgeable with security parameter $\lambda_r = \lambda$.*

Proof. Trivially follows from the unforgeability of Π: A solver $\mathcal{B}$ that gets an instance $x \in L$ can exploit a successful adversary $\mathcal{A}$ against unforgeability to fine the witness by behaving exactly like a honest signer.

At the end of the unforgeability experiment, $\mathcal{A}$ returns a forged signature σ^* on a message msg^*, which $\mathcal{B}$ can directly give as input to $\mathcal{S}_3$ to obtain a witness for x. The success probability of $\mathcal{B}$ is the same as the original reduction $\mathcal{R}$, i.e., $(1-\varepsilon_{\mathcal{R}})\varepsilon_{\mathcal{A}}$, where $\varepsilon_{\mathcal{A}}$ is the probability that $\mathcal{A}$ returns a valid forgery. As L is a hard language it holds that $(1-\varepsilon_{\mathcal{R}})\varepsilon_{\mathcal{A}} \leq \mathsf{negl}(\lambda)$, i.e., $\varepsilon_{\mathcal{A}} \leq \mathsf{negl}(\lambda)$. Hence the FSS is unforgeable for security parameter $\lambda_r = \lambda$.

Lemma 16. *If L is a λ-hard language, the scheme* $(\mathsf{GenCh}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{VrfySig}, \mathsf{PoF}, \mathsf{VrfyPoF})$ *is secure for the recipient with parameter $\lambda_r = \lambda$.*

Proof. Let R_L be the corresponding NP-relation to L. Assume by contradiction that the scheme is not secure for the recipient, that is, there exists a probabilistic polynomial-time adversary $\mathcal{A}$ that breaks security for the recipient with probability $\varepsilon_{\mathcal{A}}$. Then it holds that

$$\begin{aligned}\Pr[(\mathsf{ch},\tau) \in R_L] &= \Pr[\mathsf{VrfyPoF}(\mathsf{ch},\tau) = 1] \\ &\geq \Pr\left[\mathsf{VrfyPoF}(\mathsf{ch},\tau) = 1 \mid \begin{array}{c}\mathsf{ch} \leftarrow \mathcal{S}_L(1^\lambda), \\ \tau \leftarrow \mathcal{A}(\mathsf{ch})\end{array}\right] \\ &= \varepsilon_{\mathcal{A}}\ .\end{aligned}$$

L being a λ-hard language implies $\varepsilon_{\mathcal{A}} = \mathsf{negl}(\lambda)$. □

Lemma 17. *If L is a λ-hard language, the scheme* $(\mathsf{GenCh}, \mathsf{GenKey}, \mathsf{Sign}, \mathsf{VrfySig}, \mathsf{PoF}, \mathsf{VrfyPoF})$ *is ε-secure for the signer, where $\varepsilon = \mathsf{negl}(\lambda)$ for a negligible function* negl.

Proof. Let $\mathcal{A}$ be an unbounded adversary against security for the signer, with winning probability $0 < \varepsilon_{\mathcal{A}} \leq 1$, that is,

$$\Pr[\mathsf{Exp}^{\mathsf{ss}}_{\mathcal{A},FSS}(\mathsf{negl}(\lambda),\lambda) = 1] = \varepsilon_{\mathcal{A}}$$

This means that with probability $\varepsilon_{\mathcal{A}}$ $\mathcal{A}$ returns a forgery for which it is not possible to generate a proof of forgery. Observe that the probability that $\mathcal{A}$ recovers the secret key from pk and some signatures is negligible in λ even for an unbounded $\mathcal{A}$ by the third assumption on the signature scheme. Thus, the forgery produced by the adversary has to be independent as a random variable from sk, and in particular, can be used by $\mathcal{R}$ to recover the witness of the instance $x \in L$. As the success probability of $\mathcal{R}$ is $(1-\varepsilon)\varepsilon_{\mathcal{A}}$ it holds that

$$\Pr[\mathcal{R} \text{ succeeds}] = (1-\varepsilon_{\mathcal{R}})\varepsilon_{\mathcal{A}} = \mathsf{negl}(\lambda)$$

which implies $\varepsilon_{\mathcal{A}} < \mathsf{negl}(\lambda)$ (as $\varepsilon = \mathsf{negl}(\lambda)$). Thus, $\varepsilon = \varepsilon_{\mathcal{A}} = \mathsf{negl}(\lambda)$. □

C Lattice-Based FSS

The existence of lattice-based FSS follows naturally from Theorem 5, as lattice-based signatures satisfying the hypotheses already exist. Appendix C.2 applies the result to a variant of the Lyubashevsky-Micciancio one-time signature [38] as a warm-up example. We intend this to be a proof-of-concept, so we do not rely on standard optimizations (such as relying on ideal/module lattices, or fancy Gaussian sampler) to distract the reader from the mechanics of the construction itself. Once the mechanics are clear, in Appendix C.3 we present a lattice-based generic FSS that can be "trivially" obtained applying Theorem 5 to an existing lattice-based signatures.

C.1 Preliminaries

Let $m, n \in \mathbb{N}^*$, let $q \in \mathbb{N}$ be a prime. For a vector $\mathbf{s} \in \mathbb{Z}^n$, let $\|\mathbf{s}\|$ be the Euclidean norm, and $\|\mathbf{s}\|_\infty$ the largest module of the coefficients of $\mathbf{s}$. A lattice $\Lambda \subset \mathcal{R}^n$ is a $\mathbb{Z}$-module over $\mathcal{R}^n$. A *basis* of a lattice is a full-rank matrix $\mathbf{B} \in \mathbb{Z}^{m\times m}$ such that $\Lambda = \Lambda(\mathbf{B}) = \{\mathbf{e} \in \mathbb{Z}^m \ : \ \exists \mathbf{c} \in \mathbb{Z}^m \ : \ \mathbf{e} = \mathbf{B}\mathbf{c}\}$; we denote by $\tilde{\mathbf{B}}$ the Gram-Schmidt orthogonalization of $\mathbf{B}$. In particular, given $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ we will be mainly interested in

the lattices $\Lambda_q^{\perp}(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m \ : \ \mathbf{Ae} \equiv_q \mathbf{0}\}$ and $\Lambda_q^{\mathbf{u}}(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m \ : \ \mathbf{Ae} \equiv_q \mathbf{u}\}$ for some $\mathbf{u} \in \mathbb{Z}_q^n$. A *trapdoor* matrix $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ for $\mathbf{A}$ is a full-rank matrix such that $\mathbf{AT} \equiv_q \mathbf{0}$, that is, a basis of $\Lambda_q^{\perp}(\mathbf{A})$. Let the Gaussian function be $\rho_{\mathbf{c},s}(\mathbf{x}) = \exp(-\pi\|\mathbf{x}-\mathbf{c}\|^2/s^2)$ where $\mathbf{c}, xv \in \mathcal{R}^n$, and $s \in \mathcal{R}$, $s > 0$. The *discrete Gaussian distribution* over a lattice Λ with center $\mathbf{c} \in \mathcal{R}^n$ and parameter $s \in \mathcal{R}$, $s > 0$ is defined as $\mathcal{D}_{\Lambda,\mathbf{c},s}(\mathbf{y}) = \rho_{\mathbf{c},s}(\mathbf{y})/\rho_{\mathbf{c},s}(\Lambda)$ for and $\mathbf{y} \in \Lambda$, where $\rho_{s,\mathbf{c}}(\Lambda) = \sum_{\mathbf{x}\in\Lambda} \rho_{\mathbf{c},s}(\mathbf{x})$. For convenience, $\rho_{\mathbf{0},s}$ and $\mathcal{D}_{\Lambda,\mathbf{0},s}$ are abbreviated as ρ_s and $\mathcal{D}_{\Lambda,s}$.

Lemma 18 ([45]). *For any lattice Λ of integer dimension m with basis $\mathbf{B}$, any $\mathbf{c} \in \mathcal{R}^m$ and Gaussian parameter $s \geq \left\|\tilde{\mathbf{B}}\right\| \cdot \omega(\sqrt{\log m})$, we have*

$$\Pr[\|\mathbf{x}-\mathbf{c}\| > s\sqrt{m} : \mathbf{x} \xleftarrow{\$} \mathcal{D}_{\Lambda,\mathbf{c},s}] \leq \mathsf{negl}(m) \ .$$

We now recall some preliminary algorithms regarding trapdoor sampling and preimage sampling over lattices.

Lemma 19 ([43, Thm 5.1]). *There is a PPT algorithm* $\mathsf{TrapGen}$ *that takes as input integers $n \geq 1$, $q \geq 2$ and a sufficiently large $m = \mathcal{O}(n \log q)$, outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a trapdoor matrix $\mathbf{T}_A \in \mathbb{Z}^{m\times m}$, such that $\mathbf{AT}_A = 0 \mod q$, the distribution of $\mathbf{A}$ is statistically close to the uniform distribution over $\mathbb{Z}_q^{n\times m}$ and $\left\|\tilde{\mathbf{T}}_A\right\| = \mathcal{O}\big(\sqrt{n \log q}\big)$.*

Lemma 20 ([2, Thm 3 and 4 + Lem 5]). *Let $q > 2$, $m > n$. Then there exists:*

- *a PPT algorithm* $\mathsf{SampleLeft}$ *that on input a full-rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a short basis $\mathbf{T}_A$ of $\Lambda_q^{\perp}(\mathbf{A})$, a matrix $\mathbf{B} \in \mathbb{Z}^{n\times m_1}$, a vector $u \in \mathbb{Z}_q^n$, and $s > \left\|\tilde{\mathbf{T}}_A\right\|\omega(\sqrt{\log(m+m_1)})$, outputs a vector $\mathbf{d} \in \mathbb{Z}^{m+m_1}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\mathbf{u}}([\mathbf{A} \mid \mathbf{B}]),s}$.*
- *a PPT algorithm* $\mathsf{SampleRight}$ *that on input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a random matrix $\mathbf{R} \xleftarrow{\$} \{-1,1\}^{m\times m}$, a full-rank matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$, a short basis $\mathbf{T}_B$ of $\Lambda_q^{\perp}(\mathbf{B})$, a vector $u \in \mathbb{Z}_q^n$, and $s > 12\sqrt{2m}\left\|\tilde{\mathbf{T}}_B\right\|\omega(\sqrt{\log(m)})$, outputs a vector $\mathbf{d} \in \mathbb{Z}^{2m}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\mathbf{u}}([\mathbf{A} \mid \mathbf{AR}+\mathbf{B}]),s}$.*

Lemma 21 (Gadget Matrix [43]). *Let q be a prime, and n, m be integers with $m = n \log q$. There exists a fixed full-rank matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ such that the lattice $\Lambda_q^{\perp}(\mathbf{G})$ has a publicly known trapdoor matrix $\mathbf{T_G} \in \mathbb{Z}^{n\times m}$ with $\left\|\tilde{\mathbf{T}_{\mathbf{G}}}\right\| \leq \sqrt{5}$.*

Lemma 22 ([15, Lem. 2.11]). *Let $C : \{0,1\}^\ell \to \{0,1\}$ be a NAND Boolean circuit of depth d. Let $\{\mathbf{A}_i = \mathbf{AR}_i + x_i\mathbf{G} \in \mathbb{Z}_q^{n\times m}\}_{i=1,\ldots,\ell}$ be ℓ different matrices correspond to each input wire of C where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\mathbf{R}_i \xleftarrow{\$} \{1,-1\}^{m\times m}$, $x_i \in \{0,1\}$, and $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ is the gadget matrix. There is an efficient deterministic algorithm* Eval *that takes as input C and $\{\mathbf{A}_i\}_{i=1,\ldots,\ell}$ and outputs a matrix $\mathbf{A}_C = \mathbf{AR}_C + C(x_1,\ldots,x_\ell)\mathbf{G} = \mathsf{Eval}(\mathsf{C}, \mathbf{A}_1, \ldots, \mathsf{A}_\ell)$ where $\mathbf{R}_C \in \mathbb{Z}^{m\times m}$ and $C(x_1,\ldots,x_\ell)$ is the output of C on the arguments $x_1,\ldots,x_\ell$.* Eval *runs in time $t_E = \mathsf{poly}(\lambda)(4^d, \ell, n, \log q)$. The norm of $\mathbf{R}_C$ in $\mathbf{A}_C$ output by* Eval *can be bounded, with overwhelming probability, by $\|\mathbf{R}\| \leq \mathcal{O}\big(4^d m^{3/2}\big)$.*

Unforgeability and security for the receiver will require a computational assumption: here we use the Short Integer Solution (SIS) problem.

Definition 14 (SIS problem). *Given $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\beta > 0$, solving the (homogeneous)* $\mathsf{SIS}_{n,m,q,\beta}$ *problem requires to find $\mathbf{s} \in \mathbb{Z}^m$ s.t. $\mathbf{As} \equiv_q 0$ and $\|\mathbf{s}\| < \beta$.*

C.2 Warm-up: One-time FSS from Lattices

Public Parameters. Let $m > n\log(q)$, q be a prime. Let $\mathcal{S}_\eta := \{\mathbf{s} \in \mathbb{Z}^m \ : \ \|\mathbf{s}\|_\infty \leq \eta\}$, the message space be $\mathcal{M} = \{0,1\}^\ell$, and $q > 2(\ell+1)\eta\sqrt{2m}$ (to avoid wrap-around in the norm).

Given the public parameters defined before, the FSS works as follows.

$\mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon})$**:** The authority generates the SIS instance, that is, the parameters (q, n, m, η) (satisfying the previously listed bounds), a random matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, and $\beta = \beta(\lambda_r) = (\ell+1)\eta\sqrt{2m} > 0$. Parameters are chosen so that the probability that any PPT adversary breaks $\mathsf{SIS}_{n,m,q,\beta}$ is $\mathsf{negl}(\lambda_r)$. Set $\mathsf{ch} = (n, m, q, \beta, \eta, \mathbf{A})$.

$\mathsf{GenKey}(\mathsf{ch})$**:** The signer samples a matrix $\mathbf{S}_1 \xleftarrow{\$} \mathcal{S}_\eta^{1\times\ell}$, a vector $\mathbf{s}_2 \xleftarrow{\$} \mathcal{S}_\eta$, and computes $B = \beta/2$, $\mathbf{u}_1 = \mathbf{A}\mathbf{S}_1 \mod q$, $\mathbf{u}_2 = \mathbf{A}\mathbf{s}_2 \mod q$. The keys are $\mathsf{sk} = (\mathbf{S}_1, \mathbf{s}_2)$ and $\mathsf{pk} = (\mathbf{u}_1, \mathbf{u}_2, B)$.

$\mathsf{Sign}_{\mathsf{sk}}(\mathbf{m})$**:** The signature on the binary vector $\mathbf{m}$ is $\sigma = \mathbf{S}_1\mathbf{m} + \mathbf{s}_2 \mod q$.

$\mathsf{VrfySig}_{\mathsf{pk}}(\mathbf{m}, \sigma)$**:** The verifier checks that $\|\sigma\| \leq B$ and $\mathbf{A}\sigma = \mathbf{u}_1\mathbf{m} + \mathbf{u}_2 \mod q$.

$\mathsf{PoF}_{\mathsf{sk}}((\mathbf{m}, \sigma^*), \mathsf{ch})$**:** On input a forged signature σ^* on message $\mathbf{m}$, the signer generates the honest signature on $\mathbf{m}$: $\sigma = \mathbf{S}_1\mathbf{m} + \mathbf{s}_2 \mod q$, and returns $\bar{\mathbf{s}} = \sigma^* - \sigma$ as the proof of forgery.

$\mathsf{VrfyPoF}(\mathsf{ch}, \bar{\mathbf{s}})$**:** The verifier checks that $\bar{\mathbf{s}} \neq \mathbf{0}^m$, $\mathbf{A}\bar{\mathbf{s}} = u_1 m + u_2 \mod q$, and $\|\bar{\mathbf{s}}\| < \beta$. Returns 1 if all the checks pass, and 0 otherwise.

The following lemma yields that the public key perfectly hides the secret key.

Lemma 23 (Leftover Hash Lemma, see [5, Section 2.2.1]). *Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ for $m > n\log(q)$. Then, for $\mathbf{s} \leftarrow \mathcal{S}_\eta$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ it holds $(\mathbf{A}, \mathbf{A}\mathbf{s}) \approx^s (\mathbf{A}, \mathbf{u})$.*

Observe that applying the LHL requires high-density SIS instances [37, Section 3], which are secure for some specific choices of parameters [44, Theorem 3.8] and make the signature rather long. One can improve this using ideal lattices.

Security for the Recipient. Trivially follows from $\mathsf{SIS}_{n,m,q,\beta}$: a signer that on input $\mathbf{A}$ returns a nonzero vector $\bar{\mathbf{s}}$ such that $\mathbf{A}\bar{\mathbf{s}} \equiv_q 0$ and $\|\bar{\mathbf{s}}\| < \beta$ can be trivially converted into a solver for SIS.

Unforgeability. Analogously to security for the recipient, unforgeability follows from the hardness of $\mathsf{SIS}_{n,m,q,\beta}$. The only difference is that one has to make sure that the adversary cannot recover the secret keys (e.g., if $\mathcal{A}$ can recover $\mathbf{s}_2$, it can sign the message $\mathbf{m} = \mathbf{0}^\ell$). This trivially follows from the LHL.

Security for the Signer. We need to bound the probability that the signer cannot generate a valid proof of forgery given a valid forgery. This happens in two cases: (1) $\mathcal{A}$ correctly guessed the signature on $\mathbf{m}$, i.e., $\sigma^* = \mathbf{S}_1\mathbf{m} + \mathbf{s}_2$, (2) $\bar{\mathbf{s}}$ does not satisfy verification (i.e., it has too large norm, or it does not satisfy the linear relation). The first case can be excluded applying the LHL (extended to matrices): as $(\mathbf{A}, \mathbf{A}[\mathbf{S}_1 \mid \mathbf{s}_2]) \approx^s (\mathbf{A}, \mathbf{U})$, where $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n\times(\ell+1)}$, $\mathcal{A}$ cannot extract information about the secret keys from the public keys, and it holds $\Pr[\sigma^* = \mathbf{S}_1\mathbf{m} + \mathbf{s}_2] = \mathsf{negl}(\lambda_s)$. By correctness $\sigma^* - (\mathbf{S}_1\mathbf{m} + \mathbf{s}_2)$ satisfies the linear relation, and the condition on β, hence the second case can also be excluded.

Few-time FSS. One can build a reusable signing key by sampling many key pairs $(\mathsf{sk}_i, \mathsf{pk}_i)$ of the one-time FSS where $i = 1, \ldots, ctr$, and ctr is the number of messages that one wants to sign. These can be compressed using (lattice-based) accumulators (see [8,36]), which are based on Merkle trees combined with a collision-resistant hash function from lattices $f_{[\mathbf{A}_0 \mid \mathbf{A}_1]}(\mathbf{X}_0, \mathbf{X}_1) = \mathbf{u} \Leftrightarrow \mathbf{A}_0\mathbf{X}_0 + \mathbf{A}_1\mathbf{X}_1 = \mathbf{G}\mathbf{u} \mod q$, where $\mathbf{G}$ is the gadget matrix [43]. This yields signatures that are $\log(ctr)(\ell+1)m\log(q)$ bits long.

C.3 Lattice-Based FSS

There is a number of lattice-based signatures in the standard model that satisfy the hypotheses of Theorem 5, see for example [14,18,43,22,4,12,15]. In this section we focus on the construction of Boyen and Li [15], as it has the tightest security reduction among the lattice-based signatures in the standard model, thus it yields the most efficient FSS construction when applying Theorem 5. Moreover, this example is useful as a starting point in case one would want to add a fail-stop mechanism to more advanced signature constructions, such as group signatures, as many existing constructions rely on similar trapdoor techniques (e.g., [13]).

Table 3. Parameters for FSS.BL based on the analysis in [15, Sec. 3.2]. Observe that parameters might depend on λ_r or λ_s due to the fine-grained security model.

Parameter	Description
λ_r	Hardness of SIS
λ_s	Security of PRF
$\varepsilon = 1/2$	Probability of failure of PoF
$k = k(\lambda_s)$	Length of the secret key of the PRF
$\ell = \ell(\lambda_r)$	Message length
$d = d(\lambda_r)$	Circuit depth of the PRF
$n = n(\lambda_r)$	Number of columns
$m = n^{1+\eta}$	for $\eta > 0$ such that $n^\eta > \mathcal{O}(\log q)$
$q = \mathcal{O}\big(16^d m^4\big)(\omega(\sqrt{\log m}))^2$	Modulus
$s = \mathcal{O}\big(4^d m^{3/2}\big)\omega(\sqrt{\log m})$	Gaussian parameter
$\beta = \mathcal{O}\big(16^d m^{7/2}\big)\omega(\sqrt{\log m})$	Bound on the norm of the SIS solution

Intuition. We define the algorithms of the FSS according to the *security reduction* of the signature by Boyen and Li. The unforgeability of this construction relies on the hardness of the SIS problem, and on the security of a PRF. As such, we will have to use the fine-grained definitions of security (in particular, Definition 5), where we set $c_2 = 2$ analogously to what observed for FSS.XMSS (cf. Section 6.1). Analogously to the one-time FSS, the authority plays the part of the SIS oracle in the reduction, that is, it creates an SIS instance. A proof of forgery will be a valid solution for such instance. Hence, the security parameter λ_r is the computational hardness of the SIS instance. The key generation and signing algorithms correspond to the algorithms in the reduction that simulate the signer, and exploit the trapdoor duality result in Lemma 20.

Public Parameters. Parameters are described in Table 3, and are chosen according to [15, Sec. 3.2]. The security of the FSS is defined by three parameters $(\varepsilon, \lambda_r, \lambda_s)$, where the computational power of the signer is bounded in λ_r, and a the computational power of the potential forgerer is bounded in $\lambda_s > \lambda_r$. Let the message space be $\mathcal{M} = \{0,1\}^\ell$ and $\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}$ be a λ_s-secure PRF whose depth when represented as a NAND circuit C_{PRF} is d.

Given the public parameters defined before, we define the lattice-based FSS scheme FSS.BL as follows.

$\mathsf{GenCh}(1^\varepsilon, 1^{\lambda_r}, 1^{\lambda_s})$**:** The authority generates the SIS instance, that is, the parameters (q, n, m, β) (satisfying the bounds in Table 3), and a random matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$. Parameters are chosen so that the probability that any PPT adversary breaks $\mathsf{SIS}_{n,m,q,\beta}$ is $\mathsf{negl}(\lambda_r)$. Then it chooses a λ_s-secure PRF $\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}$ and expresses it as a NAND circuit C_{PRF} with depth d. Set $\mathsf{ch} = (n, m, q, \beta, \mathbf{A}, C_{\mathsf{PRF}})$.

$\mathsf{GenKey}(\mathsf{ch})$**:** The signer does the following:

1. Select $k+4$ matrices $\mathbf{R}_{\mathbf{A}_0}$, $\mathbf{R}_{\mathbf{A}_1}$, $\{\mathbf{R}_{\mathbf{B}_i}\}_{i=1,\dots,k}$, $\mathbf{R}_{\mathbf{C}_0}$, $\mathbf{R}_{\mathbf{C}_1}$ uniformly at random in $\{1,-1\}^{m\times m}$.
2. Select a PRF key $K = (c_1, \dots, c_k) \xleftarrow{\$} \{0,1\}^k$
3. Set $\mathbf{A}_b := \mathbf{A}\mathbf{R}_{\mathbf{A}_b} + b\mathbf{G}$ and $\mathbf{C}_b := \mathbf{A}\mathbf{R}_{\mathbf{C}_b} + b\mathbf{G}$ for $b = 0, 1$, where $\mathbf{G}$ is the gadget matrix from Lemma 21.
4. Set $\mathbf{B}_i := \mathbf{A}\mathbf{R}_{\mathbf{B}_i} + c_i\mathbf{G}$ for $i = 1, \dots, k$.
5. Select a Gaussian parameter $s > 0$ satisfying the bounds in Table 3.

The keys are

$$\mathsf{sk} = (\mathbf{R}_{\mathbf{A}_0}, \mathbf{R}_{\mathbf{A}_1}, \{\mathbf{R}_{\mathbf{B}_i}\}_{i=1,\dots,k}, \mathbf{R}_{\mathbf{C}_0}, \mathbf{R}_{\mathbf{C}_1}, K)$$
$$\mathsf{pk} = (\mathbf{A}, \{\mathbf{A}_0, \mathbf{A}_1\}, \{\mathbf{B}_i\}_{i=1,\dots,k}, \{\mathbf{C}_0, \mathbf{C}_1\})$$

$\mathsf{Sign}_{\mathsf{sk}}(\mathsf{msg})$**:** Upon receiving a message $\mathsf{msg} = (m_1, \dots, m_\ell) \in \{0,1\}^\ell$ the signer does the following:

1. Run $\mathsf{Eval}(\mathsf{C}_{\mathsf{PRF}}, \{\mathbf{B}_i\}_{i=1,\dots,k}, \mathbf{C}_{m_1}, \dots, \mathbf{C}_{m_\ell})$ to compute $\mathbf{A}_{C_{\mathsf{PRF}}} := \mathbf{A}\mathbf{R}_{C_{\mathsf{PRF}},\mathsf{msg}} + \mathsf{PRF}(K, \mathsf{msg})\mathbf{G} \in \mathbb{Z}_q^{n\times m}$.

2. Let $b = \mathsf{PRF}(K, \mathsf{msg})$, it sets $\mathbf{F}_{\mathsf{msg},1-b} := [\mathbf{A}|\mathbf{A}_{1-b} - \mathbf{A}_{C_{\mathsf{PRF}},\mathsf{msg}}] = [\mathbf{A}|\mathbf{A}(\mathbf{R}\mathbf{A}_{1-b} - \mathbf{R}_{C_{\mathsf{PRF}},\mathsf{msg}}) + (1-2b)\mathbf{G}]$.
3. Run $\mathsf{SampleRight}(\mathbf{A}, \mathbf{R}\mathbf{A}_{1-b} - \mathbf{R}_{C_{\mathsf{PRF}},\mathsf{msg}}, (1-2b)\mathbf{G}, \mathbf{T}_{\mathbf{G}}, \mathbf{0}, s)$ to generate the signature $\mathbf{s} \approx \mathcal{D}_{\Lambda_q^{\perp}(\mathbf{F}\mathsf{msg},1-b),s}$.

Return $\sigma = \mathbf{s}$.

$\mathsf{VrfySig}_{\mathsf{pk}}(\mathsf{msg}, \sigma)$**:** To verify a signature requires to do the following:

1. It checks if $\mathbf{s} \in \mathbb{Z}^{2m}$, $\mathbf{s} \neq \mathbf{0}$, and $\|\mathbf{s}\| \leq s\sqrt{2m}$.
2. Compute $\mathbf{A}_{C_{\mathsf{PRF}},\mathsf{msg}} = \mathsf{Eval}(\mathsf{C}-\mathsf{PRF}, \{\mathbf{B}_{\mathsf{i}}\}_{\mathsf{i}=1,\ldots,\mathsf{k}}, \mathbf{C}_{\mathsf{m}_1}, \ldots, \mathbf{C}_{\mathsf{m}_\ell}) \in \mathbb{Z}_{\mathsf{q}}^{\mathsf{n}\times\mathsf{m}}$.
3. Check if $F_{\mathsf{msg},b}\mathbf{s} = [\mathbf{A}|\mathbf{A}_b - \mathbf{A}_{C_{\mathsf{PRF}},\mathsf{msg}}]\mathbf{s} = \mathbf{0} \mod q$ for $b = 0$ or 1.

If all the check pass, return 1; otherwise 0.

$\mathsf{PoF}_{\mathsf{sk}}((\mathsf{msg}, \sigma^*), \mathsf{ch})$**:** On input a possible forgery $\sigma^* = \mathbf{s}^*$ on a message msg^* the signer does the following:

1. Compute $b = \mathsf{PRF}(K, \mathsf{msg}^*)$.
2. If $\|\mathbf{s}^*\| > s\sqrt{2m}$ or $[\mathbf{A}|\mathbf{A}_{1-b} - \mathbf{A}_{C_{\mathsf{PRF}},\mathsf{msg}^*}]\mathbf{s}^* = \mathbf{0} \mod q$, return $\perp$.
   Otherwise, let $\mathbf{s}^* = \begin{pmatrix}\mathbf{s}_1^* \\ \mathbf{s}_2^*\end{pmatrix} \in \mathbb{Z}^{2m}$.
3. Compute $\mathbf{e} = \mathbf{s}_1^* + (\mathbf{R}\mathbf{A}_b - \mathbf{R}_{C_{\mathsf{PRF}},\mathsf{msg}^*})\mathbf{s}_2^*$ as a solution for the $\mathsf{SIS}_{n,q,\beta,m}$ problem instance.

The proof of forgery is $\pi = \mathbf{e}$.

$\mathsf{VrfyPoF}(\mathsf{ch}, \mathbf{e})$**:** The authority checks that $\|\mathbf{e}\| \leq \beta$ and that $\mathbf{A}\mathbf{e} = \mathbf{0} \mod q$. Returns 1 if all the checks pass, and 0 otherwise.

Lemma 24 (Correctness). *The FSS scheme* FSS.BL *instantiated with the parameters in Table 3 is correct, that is, for all positive* $(\lambda_r, \lambda_s) \in \mathbb{N}$, $0 < \varepsilon \leq 1$ *there exists a negligible function* negl *such that:*

$$\Pr\left[1 \leftarrow \mathsf{VrfySig}_{\mathsf{pk}}(\mathsf{msg}, \sigma) \;:\; \begin{array}{c}\mathsf{ch} \leftarrow \mathsf{GenCh}(1^{\varepsilon}, 1^{\lambda_r}, 1^{\lambda_s}), \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{GenKey}(\mathsf{ch}), \\ \sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathsf{msg})\end{array}\right] > 1 - \mathsf{negl}(\lambda_r)\ .$$

Proof. The parameter analysis is the same as for the original signature scheme, so we refer the reader to [15, Sec 3.2]. We just remark that by Lemma 18, a signature $\sigma = \mathbf{s}$ satisfies the norm bound with probability $p_1 = 1 - \mathsf{negl}(n)$. The second check of the verification is always correct by definition of Eval. □

Runtime of Sign. Observe that the runtime of the signer depends on the runtime of the simulator in the security reduction of the original signature scheme(cf. [15, Thm 3.1]). In particular, the runtime of the signing algorithm is $t_S + t_E = \mathsf{poly}(\lambda)(4^d, \ell, n, \log q)$, where t_S is the runtime of SampleRight, t_E is the runtime of Eval, and the asymptotic estimate comes from Lemma 20 and Lemma 22.

Lemma 25 (Security for the Recipient). *If* $\mathsf{SIS}_{n,q,\beta,m}$ *is* λ_r*-hard, then* FSS.BL *is secure for the recipient.*

Proof. The security for the recipient follows trivially from the hardness of $\mathsf{SIS}_{n,q,\beta,m}$ with an analogous reduction to the security for the recipient of the one-time construction (cf. Appendix C.2).

Lemma 26 (Unforgeability). *If* PRF *is a* λ_s*-secure PRF, and* $\mathsf{SIS}_{n,q,\beta,m}$ *is* λ_r*-hard, then* FSS.BL *is unforgeable under adaptive CPA.*

Proof. The reduction follows exactly the same steps as the unforgeability reduction of the original signature scheme, thus we refer the reader to the proof of [15, Thm 3.1] for the details. We only compute the final success probability of a cheating signer. Denote the probability of solving the SIS instance by ε_{SIS} and the probability of breaking the PRF security by $\varepsilon_{\mathsf{PRF}}$. From Boyen's and Li's reduction it follows that

$$\Pr\left[\mathsf{VrfyPoF}(\mathsf{ch}, \pi^*) = 1 \mid \begin{array}{c}\mathsf{ch} \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^{\varepsilon}), \\ \pi^* \leftarrow \mathcal{A}(\mathsf{ch})\end{array}\right] < 2\varepsilon_{SIS} + \varepsilon_{\mathsf{PRF}} + \mathsf{negl}(\lambda_r) = \mathsf{negl}(\lambda_r)$$

where the latter equality follows from the λ_r-hardness of SIS, the λ_s-security of the PRF, and because $\lambda_r < \lambda_s$ by definition of the fine-grained security framework.

Lemma 27 (Security for the Signer). *If* PRF *is a λ_s-secure PRF, and* $\mathsf{SIS}_{n,q,\beta,m}$ *is λ_r-hard, then* FSS.BL *is ε-secure for the signer in the fine-grained framework for parameters $(\varepsilon, \lambda_r, \lambda_s) = (1/2 + \mathsf{negl}(\lambda_s), \lambda_r, \lambda_s)$.*

Proof. Again, the reduction follows the same steps as the unforgeability proof of the original signature. Observe that the factor $1/2$ comes from the fact that the adversary could guess correctly the output bit b of the PRF even without recovering the key K. This would mean that the forged signature would verify w.r.t. the same public key that the signer would have used to sign msg^*, thus PoF would have to abort. Hence PoF aborts with probability $\varepsilon = 1/2 + \mathsf{negl}(\lambda_s)$.

Failure Probability of PoF . One could argue that the fail-stop mechanism is not very useful if it only works 50% of the times. However, analogously to the case of soundness in zero-knowledge proofs, the probability of failure can be increased by signing the same message multiple times, that is, by defining a valid signature as the concatenation of N signatures. Indeed, for the signer to be able to prove that the scheme has been compromise it is enough to prove that one of the N signatures is a forgery. Hence the probability of failure for the concatenated scheme is now $1/2^N$.

D The Security Assumptions of SPHINCS$^+$

SPHINCS$^+$ relies on tweakable hash functions (THFs) [10]. A THF is a function $\mathbf{Th} : P \times \mathcal{T} \times \{0,1\}^m \to \{0,1\}^n$, where $\mathcal{P}$ is the public parameters of the digital signature, $\mathcal{T}$ is the tweak (different in every evaluation of $\mathbf{Th}$ for security reasons), m is the message's length, and n is the signature's length. The security of SPHINCS$^+$ requires the underlying THF to have four security properties:

1. Single-function, Multi-target UnDetectability for distinct tweaks (SM-UD security): informally it means that $\mathbf{Th}(p,t,x)$ is computationally indistinguishable from x, where x is uniformly sampled.
2. Single-function, Multi-target Collision Resistance for distinct tweaks (SM-TCR security): the adversary cannot find a *collision* for a target from the set of its queries Q given an oracle access to $\mathbf{Th}(p,\cdot,\cdot)$.
3. Single-function, Multi-target (second) Preimage Resistance for distinct tweaks (SM-PRE security): the adversary cannot find a *second preimage* for a target from the set of its queries Q given an oracle access to $\mathbf{Th}(p,\cdot,x)$, where x is uniformly sampled in every call to the oracle.
4. Interleaved Target Subset Resilience (ITSR): the adversary cannot find a *second preimage* for a target among the answers to its queries, when given an oracle access to a keyed hash function that samples a fresh random key for each query.

Informally, our theorem is:

Theorem 6 (Informal).

- *If $\mathbf{Th} = \{\mathbf{Th}_i\}_i$ is a family of THFs that is SM-UD, SM-TCR, and SM-PRE secure,* PRF, $\mathsf{PRF}_{\mathsf{msg}}$ *are PRFs, $\mathbf{H} = \{\mathbf{H}_i\}_i$ is a family of SM-TCR secure THFs, and H_{msg} is a ITSR secure compressing THF, then* FSS.SPHINCS *is unforgeable under adaptive CPA.*
- *Assume that $\mathcal{A}$ cannot break the ITSR security of H_{msg} nor invert* PRF *and* $\mathsf{PRF}_{\mathsf{msg}}$*. If $\mathbf{Th}$ and $\mathbf{H}$ are families of compressing THF, then* FSS.SPHINCS *is secure for signer against an adversary $\mathcal{A}$ with running time at most $2^{c_s \lambda_s/2}$ (in the QROM).*
- *If $\mathbf{Th}$ and $\mathbf{H}$ are families of SM-TCR secure THFs, then* FSS.SPHINCS *is secure for the recipient.*

D.1 Tweakable Hash Functions

Definition 15 (Tweakable hash function - THF). *Let $n, m \in \mathbb{N}$, $\mathcal{P}$ the public parameters space and $\mathcal{T}$ the tweak space. A* tweakable hash function (THF) *is an efficient function $\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^m \to \{0,1\}^n$, $MD \leftarrow \mathbf{Th}(P,T,M)$ mapping a m-bit message M to an n-bit hash value MD using a function key composed by a public parameter $P \in \mathcal{P}$ and a tweak $T \in \mathcal{T}$.*

Let us define the predicate $\mathsf{DIST}(\{T_i\}_{i=1}^p) = (T_i \neq T_k \ \forall i,k \in [1,p], i \neq k)$, i.e., $\mathsf{DIST}(\{T_i\}_{i=1}^p)$ outputs 1 iff all tweaks are distinct. Then, security for THF is defined by three properties (cf. [33]).

Definition 16 (Single-function, Multi-target UnDetectability for distinct tweaks - SM-UD). *Let* **Th** *be a THF. Consider an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *playing the security game in Experiment 5, where* $\mathcal{A}_1$ *is allowed to make* p *classical queries to an oracle* $\mathcal{O}_P(T,b)$*, for* $p \leq |\mathcal{T}|$ *and* $b \in \{0,1\}$*.*

Experiment 5: $\mathsf{SM\text{-}UD}^b_{\mathcal{A},\mathbf{Th},p}(\lambda)$

1: $P \leftarrow_R \mathcal{P}$ // *The public parameters of the function*
2: *Define* $\mathcal{O}_P(T,b)$*:*
- *If* $b = 0$*, then* $\mathcal{O}_P(T,0)$*: return* $\mathbf{Th}(P,T,x)$*, where* x *is chosen uniformly at random for every the query.*
- *If* $b = 1$*, then* $\mathcal{O}_P(T,1)$*: return* x*, where* x *is chosen uniformly at random for every the query.*

3: $(\mathcal{Q} = \{T_i\}_{i=1}^p, S) \leftarrow \mathcal{A}_1^{\mathcal{O}_P(\cdot,b)}(P)$ // S *is the shared state between* $\mathcal{A}_1$ *and* $\mathcal{A}_2$*, and* $\mathcal{Q}$ *are the oracle queries*
4: $b' \leftarrow \mathcal{A}_2(Q,S,P)$
5: ***if*** DIST $(\{T_i\}_{i=1}^p)$ ***then***
6: ***return*** b'
7: ***else*** ***return*** $1-b$ // $\mathcal{A}$ *wins only if all its queries are distinct.*

The advantage of an adversary $\mathcal{A}$ *in this experiment is:*

$$\mathsf{Adv}^{SM-UD}_{\mathbf{Th},p}(\mathcal{A}) = |\Pr[1 \leftarrow \mathit{SM\text{-}UD}^1_{\mathcal{A},\mathbf{Th},p}(\lambda)] - \Pr[1 \leftarrow \mathit{SM\text{-}UD}^0_{\mathcal{A},\mathbf{Th},p}(\lambda)]| \ .$$

The THF **Th** *is SM-UD secure if the advantage is bounded by* $\mathsf{negl}(\lambda)$*.*

Definition 17 (Single-function, Multi-Target Collision Resistance for distinct tweaks - SM-TCR). *Let* **Th** *be a THF. Consider an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *playing the* non-adaptive *security game in Experiment 6, where* $\mathcal{A}_1$ *is allowed to make* p *classical queries to an oracle* $\mathcal{O}_P(\cdot,\cdot)$*, for* $p \leq |\mathcal{T}|$*.*

Experiment 6: $\mathsf{SM\text{-}TCR}_{\mathcal{A},\mathbf{Th},p}(\lambda)$

1: $P \leftarrow_R \mathcal{P}$ // *The public parameters of the function*
2: *Define* $\mathcal{O}_P(\cdot,\cdot)$*: on input* (T,M)*, it returns* $\mathbf{Th}(P,T,M)$*.*
3: $(\mathcal{Q} = \{(T_i,M_i)\}_{i=1}^p, S) \leftarrow \mathcal{A}_1^{\mathcal{O}_P(\cdot,\cdot)}()$
4: $(j,M) \leftarrow \mathcal{A}_2(\mathcal{Q},S,P)$
5: ***if*** $\mathbf{Th}(P,T_j,M_j) = \mathbf{Th}(P,T_j,M) \ \wedge \ M \neq M_j \ \wedge \ \mathsf{DIST}(\{T_i\}_{i=1}^p)$ ***then***
6: ***return*** *1*
7: ***else*** ***return*** *0*

We denote by $\mathsf{Succ}^{\mathit{SM\text{-}TCR}}_{\mathbf{Th},p}(\mathcal{A})$ *the probability that* $\mathcal{A}$ *wins the experiment. The THF* **Th** *is SM-TCR secure if* $\mathsf{Succ}^{\mathit{SM\text{-}TCR}}_{\mathbf{Th},p}(\mathcal{A}) \leq \mathsf{negl}(\lambda)$*.* Adaptive SM-TCR security *requires the adversary to win with overwhelming probability a variant of Experiment 6 in which* $\mathcal{A}_1$ *gets as input the parameter* P *instead of oracle access to the THF.*

Definition 18 (Single-function, Multi-target (second) Preimage REsistance for distinct tweaks - SM-PRE). *Let* **Th** *be a THF. Consider an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *playing the security game in Experiment 7, where* $\mathcal{A}_1$ *is allowed to make* p *classical queries to an oracle* $\mathcal{O}_P(\cdot, \{x_i\}_i)$*, for* $p \leq |\mathcal{T}|$*.*

Experiment 7: $\mathsf{SM\text{-}PRE}_{\mathcal{A},\mathbf{Th},p}(\lambda)$

$P \leftarrow_R \mathcal{P}$
for $i=1,\ldots,p$ ***do***
  $x_i \xleftarrow{\$} \{0,1\}^m$
Define $\mathcal{O}_P(\cdot,\{x_i\}_i)$*: on input the* i*-th query* T_i*, it returns* $\mathbf{Th}(P,T_i,x_i)$.
$(\mathcal{Q}=\{(T_i)\}_{i=1}^p, S) \leftarrow \mathcal{A}_1^{\mathcal{O}_P(\cdot,\{x_i\}_i)}()$
$(j,M) \leftarrow \mathcal{A}_2(\mathcal{Q},S,P)$
if $\mathbf{Th}(P,T_j,x_j)=\mathbf{Th}(P,T_j,M) \ \wedge \ M \neq x_j \ \wedge \ \mathsf{DIST}(\{T_i\}_{i=1}^p)$ ***then***
  return *1*
else ***return*** *0*

We denote by $\mathsf{Succ}^{\mathsf{SM\text{-}PRE}}_{\mathbf{Th},p}(\mathcal{A})$ *the probability that* $\mathcal{A}$ *wins the experiment. The THF* $\mathbf{Th}$ *is SM-PRE secure if* $\mathsf{Succ}^{\mathsf{SM\text{-}PRE}}_{\mathbf{Th},p}(\mathcal{A}) \leq \mathsf{negl}(\lambda)$.

Finally, to define Decisional Second Preimage Resistance (SM-DSPR) we need a second-preimage exists predicate for THFs.

Definition 19 ($SP_{P,T}$). *A second preimage exists predicate of a THF* $\mathbf{Th}: \mathcal{P}\times\mathcal{T}\times\{0,1\}^m \to \{0,1\}^n$ *with a fixed* $P\in\mathcal{P}$, $T\in\mathcal{T}$ *is the function* $SP_{P,T}:\{0,1\}^m\to\{0,1\}$ *defined as follows:*

$$SP_{P,T}(x) := \begin{cases} 1 & \text{if } |\mathbf{Th}^{-1}_{P,T}(\mathbf{Th}(P,T,x))| \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{Th}^{-1}_{P,T}$ *refers to the inverse of the tweakable hash function with fixed public parameter and tweak.*

Definition 20 (Decisional Second-Preimage Resistance - SM-DSPR). *Let* $\mathbf{Th}$ *be a THF. Consider an adversary* $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)$ *playing the security game in Experiment 8, where* $\mathcal{A}_1$ *is allowed to make* p *classical queries to an oracle* $\mathcal{O}_P(\cdot,\cdot)$*, for* $p\leq|\mathcal{T}|$.

Experiment 8: $\mathsf{SM\text{-}DSPR}_{\mathcal{A},\mathbf{Th},p}(\lambda)$

$P \leftarrow_R \mathcal{P}$
Define $\mathcal{O}_P(\cdot,\cdot)$*: on input* (T,M)*, it returns* $\mathbf{Th}(P,T,M)$.
$(\mathcal{Q}=\{(T_i,M_i)\}_{i=1}^p, S) \leftarrow \mathcal{A}_1^{\mathcal{O}_P(\cdot,\cdot)}()$
$(j,b) \leftarrow \mathcal{A}_2(\mathcal{Q},S,P)$
if $SP_{P,T_j}(x_j)=b \ \wedge \ \mathsf{DIST}(\{T_i\}_{i=1}^p)$ ***then***
  return *1*
else ***return*** *0*

We denote by $\mathsf{Succ}^{\mathsf{SM\text{-}DSPR}}_{\mathbf{Th},p}(\mathcal{A})$ *the probability that* $\mathcal{A}$ *wins the experiment. The THF* $\mathbf{Th}$ *is SM-DSPR secure if*

$$\max\left\{0, \mathsf{Succ}^{\mathsf{SM\text{-}DSPR}}_{\mathbf{Th},p}(\mathcal{A}) - \Pr\left[\begin{array}{c|c} SP_{P,T_j}(x_j)=b \ \wedge \ \mathsf{DIST}(\{T_i\}_{i=1}^p) & \begin{array}{c} P\leftarrow_R\mathcal{P}, \\ (\mathcal{Q}=\{(T_i,M_i)\}_{i=1}^p,S)\leftarrow\mathcal{A}_1^{\mathcal{O}_P(\cdot,\cdot)}(), \\ (j,b)\leftarrow\mathcal{A}_2(\mathcal{Q},S,P) \end{array} \end{array}\right]\right\} \leq \mathsf{negl}(\lambda)\ .$$

D.2 Interleaved Target Subset Resilience

Unforgeability for Hash-and-Sign signatures requires that $\mathcal{A}$ cannot find a message which is mapped to a key set (and a set of indexes in case of $\mathsf{SPHINCS}^+$ and FSS.SPHINCS) such that the adversary has already seen all secret values indicated by the indexes for that key set. This property is called Interleaved Target Subset Resilience (ITSR), and is defined as follows.

Definition 21 (Interleaved Target Subset Resilience - ITSR). *Let $H : \mathcal{K} \times \{0,1\}^{\ell} \to \{0,1\}^{m}$ be a keyed hash function. Consider a mapping function $MAP_{h,k,t} : \{0,1\}^{m} \to \{0,1\}^{h} \times [0, t-1]^{k}$ which maps an m-bit string to a set of k indexes. We denote those indexes as $((I, 1, J_1), \ldots, (I, k, J_k))$, where I is chosen from $[0, 2^h - 1]$ and each J_i is chosen from $[0, t-1]$. The success probability of an adversary $\mathcal{A}$ against ITSR of H is defined as follows. Let $G = MAP_{h,k,t} \circ H$. Let $\mathcal{O}(\cdot)$ be an oracle which on input of an ℓ-bit message m_i samples a key $k_i \xleftarrow{\$} \mathcal{K}$ and returns $G(k_i, m_i)$. The adversary $\mathcal{A}$ is allowed to query the oracle with messages of its choice. Denote the number of queries with q. Then,*

$$\mathsf{Succ}^{ITSR}_{H,q} = \Pr\left[\begin{matrix} G(k,m) \subseteq \bigcup_{j=1}^{q} G(k_j, m_j) \\ \wedge (k,m) \notin \{(k_j, m_j)\}_{j=1}^{q} \end{matrix} \mid (k,m) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(1^{\lambda})\right] ,$$

where $\{(k_j, m_j)\}_{j=1}^{q}$ represent the responses of the oracle $\mathcal{O}(\cdot)$. The hash H is ITSR if $\mathsf{Succ}^{ITSR}_{H,q} = \mathsf{negl}(\lambda)$.

E Introducing SPHINCS$^+$

SPHINCS$^+$ is composed of various building blocks: a one-time signature called WOTS$^+$ [29], which is transformed in a multiple-use signature called XMSS using Merkle trees [16], hypertrees, and the few-times signature scheme FORS [10].

E.1 Tweakable Hash Function in SPHINCS$^+$

To achieve security, tweakable hash function requires that a *unique tweak* is used in every call to the function. To ensure that each tweak is indeed unique SPHINCS$^+$ it includes a "context information" called ADRS that encodes the specific usage of the hash function and its location in the tree. In our paper, we denote this tweak as either T_{location} or T_{msg} for the hash of the message. For uniqueness between multiple instances of SPHINCS$^+$, the tweak also includes a unique public seed that is sampled during key generation that we denote by PubSeed.

E.2 WOTS$^+$

WOTS$^+$ [29] is hash-based one-time signature based on the Winternitz signature (first mentioned in [42]). The latter is preferable to Lamport signature [34], as it reduces the length of signature and keys by signing the representation of a message $m \in \{0,1\}^{h}$ base w, for some $w \in \mathbb{N}$ (WOTS$^+$ with $w = 2$ is essentially Lamport signature). The construction relies on a *chaining function* c^i_k, that is, a function that applies a somewhat collision-resistant, hard to invert function f_k $w - 1$ times to each secret key:

$$c^i_k(x, r) = \begin{cases} x & \text{if } i = 0 \\ f_k(c^{i-1}_k(x, r) \oplus r_i) & \text{otherwise} \end{cases}$$

where $r = (r_1, \ldots, r_{w-1})$ is a public random vector used in all the chains to increase entropy, and f_k is chosen at random from a family $F_n : \{f_k : \{0,1\}^{n} \to \{0,1\}^{n} \mid k \in K_n\}$ with key space K_n. This yields ℓ_1 chains, where $\ell_1 = \lceil h / \log w \rceil$, i.e., one chain per component of the representation of m in base w (denoted by $[m]_w$ from now on). Let m_i be the i-th component of $[m]_w$: the i-th component of the signature would be $c^{m_i}_k(\mathsf{sk})$.

This is not enough to guarantee unforgeability though. For example, by querying a signature on a message m such that $[m]_w = (0, \ldots, 0)$ the adversary gets all the secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_{\ell_1})$, thus in this case $\mathcal{A}$ can perfectly impersonate the signer. To avoid this, the message digest that is signed includes both the message m and a *checksum* $C = \sum_{i=1}^{\ell_2}(w - 1 - m_i)$. This increases the length of keys and signature by $\ell_2 = \lfloor \log_w(\ell_1(w-1)) \rfloor + 1$, but now guarantees unforgeability under some special assumptions on f_k.

Definition 22. *Let $n \in \mathbb{N}$, $F_n : \{f_k : \{0,1\}^n \to \{0,1\}^n \mid k \in K_n\}$ is a family of functions with a key space K_n. We say that F_n is* t-undetectable*, if for every distinguisher $\mathcal{D}$ that runs in time less or equal to t, there is a negligible function* negl *such that:*

$$\begin{aligned}|\Pr\left[\mathcal{D}(k,r) = 1 \mid k \leftarrow_R K_n, r \leftarrow_R \{0,1\}^n\right] - \\ - \Pr\left[\mathcal{D}(k, f_k(r)) = 1 \mid k \leftarrow_R K_n, r \leftarrow_R \{0,1\}^n\right]| \leq \mathsf{negl}(n)\ .\end{aligned}$$

The algorithms are formally described in Protocol 8.

Protocol 8: WOTS$^+$

Let $F_n : \{f_k^i : \{0,1\}^n \to \{0,1\}^n \mid k \in K_n\}$, and fix $w \in \mathbb{N} \setminus \{0,1\}$.
Let $[m]_w$ be the representation of m base w, and $\mathcal{M} = \{0,1\}^h$ be the message space.

$\mathsf{Sign.KeyGen}(1^\lambda, w, h)$:

$n \leftarrow \lambda$
$\ell_1 \leftarrow \lceil h / \log w \rceil$
$\ell_2 \leftarrow \lfloor \log_w(\ell_1(w-1)) \rfloor + 1$
$\ell \leftarrow \ell_1 + \ell_2$
$\mathsf{p} \leftarrow (w, h, \ell, \ell_1, n)$
for $i = 1, \ldots, w-1$ **do**
    $r_i \xleftarrow{\$} \{0,1\}^n$
$r \leftarrow (r_1, \ldots, r_{w-1})$
for $i = 1, \ldots, \ell$ **do**
    $\mathsf{sk}_i \xleftarrow{\$} \{0,1\}^n$
    $\mathsf{pk}_i \leftarrow c_k^{w-1}(\mathsf{sk}_i, r)$
$\mathsf{sk} \leftarrow (\mathsf{p}, \mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$
$\mathsf{pk} \leftarrow (\mathsf{p}, (r,k), \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$
return $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign.Sign}_{\mathsf{sk}}(m)$:

$(m_1, \ldots, m_{\ell_1}) \leftarrow [m]_w$, $m_i \in [0, w-1]$.
$C \leftarrow \sum_{i=1}^{\ell_1}(w - 1 - m_i)$ // Checksum
$(c_1, \ldots, c_{\ell_2}) \leftarrow [C]_w$, $c_{i'} \in [0, w-1]$
$(b_1, \ldots, b_\ell) \leftarrow (m_1, \ldots m_{\ell_1}, c_1, \ldots, c_{\ell_2})$
for $i = 1, \ldots, \ell$ **do**
    Compute $\sigma_i = c_k^{b_i}(\mathsf{sk}_i, r)$
$\sigma \leftarrow (\sigma_1, \ldots, \sigma_\ell)$
return σ.

$\mathsf{Sign.Ver}_{\mathsf{pk}}(m, \sigma)$:

$(\sigma_1, \ldots, \sigma_\ell) \leftarrow \sigma$
$(m_1, \ldots, m_{\ell_1}) \leftarrow [m]_w$, $m_i \in [0, w-1]$.
$C \leftarrow \sum_{i=1}^{\ell_1}(w - 1 - m_i)$ // Checksum
$(c_1, \ldots, c_{\ell_2}) \leftarrow [C]_w$, $c_{i'} \in [0, w-1]$
$(b_1, \ldots, b_\ell) \leftarrow (m_1, \ldots m_{\ell_1}, c_1, \ldots, c_{\ell_2})$
for $i = 1, \ldots, \ell$ **do**
    $\sigma'_i = c_k^{w-b_i}(\sigma_i, \vec{r})$
if $\mathsf{pk}_i = \sigma'_i,\ \forall\ i \in [1, w-1]$ **then**
    return 1.
else return 0.

Theorem 7 ([29]). Informal. *If the chaining function is a second-preimage resistant family of undetectable one-way functions, then* WOTS$^+$ *is unforgeable under chosen message attacks.*

E.3 Proof of Lemma 7

Proof. Let $\mathcal{N}_y := |\{x \in \{0,1\}^{n+\delta} \ :\ f(x) = y\}|$, i.e., the number of preimages of a point $y \in \{0,1\}^n$ under f. For any y in the image of f, trivially there exists at least one preimage x. Counting the rest of the preimages is equivalent to evaluating f on the whole $\{0,1\}^{n+\delta} \setminus \{x\}$, and counting how many times one gets y. As f is chosen at random, the probability that $f(x') = y$ is 2^{-n}. Thus, for y in the domain, $y = f(x)$, the number of preimages can be written as $\mathcal{N}_y = 1 + \mathcal{N}$, where $\mathcal{N}$ is a random variable (independent of y) that follows a Bernoulli distribution with success probability 2^{-n} over $2^{n+\delta} - 1$ attempts. Therefore, for a fixed y in the domain,

$$\Pr\left(\exists! x \ :\ f(x) = y\right) = \Pr(\mathcal{N}_y = 1) = \Pr(\mathcal{N} = 0) = \left(1 - \frac{1}{2^n}\right)^{2^{n+\delta}-1} \leq 2e^{-2^\delta}\ .$$

The second to last inequality follows observing that $g(n) := (1 - 2^{-n})^{2^n}$ is a strictly increasing sequence of positive numbers such that $\lim_{n \to +\infty} g(n) = 1/e$. □

E.4 XMSS

XMSS [16] essentially combines Merkle trees with WOTS^+ to obtain a (stateful) multiple-use signature. To sign N messages, the signer generates in advanced N pairs of WOTS^+ keys $(\mathsf{sk}^i, \mathsf{pk}^i)$, hashes the pk^i to a n-bit string lf^i, and then constructs a Merkle tree using $\mathsf{lf}^1, \ldots, \mathsf{lf}^N$ as leaves. The root is the published pk. Signing the i-th message m requires revealing the authentication path to the i-th leaf, its preimage, and signing m with the WOTS^+ key sk^i.

This yields a multiple-use scheme with a short pk. However, XMSS has two problems: it is stateful, and the value of N cannot be too large (otherwise generating the full tree, and its reconstruction to verify a signature, would require too much time).

Practical Stateful XMSS*: Hypertrees.* In $\mathsf{SPHINCS}^+$, XMSS is used to build hypertrees, that is, a bunch of Merkle trees nested into each other as follows. The signer starts by generating a standard XMSS tree of depth d with root pk. Then, this tree can be connected to another XMSS tree of depth d by simply using the WOTS^+ secret key corresponding to one of the leaves of the first tree to sign the root of the second tree. This way one can start by generating just the first XMSS tree, and add the following layers of trees only when necessary. Remark that this still requires to keep track of which WOTS^+ key has been used to sign.

Making XMSS *Stateless.* To avoid having to keep track of which key has been used, the idea is applying a public hash function to the message to randomly path, and the index of the WOTS^+ key to be used. Security is up to collisions: as long as the number of possible keys is large enough, the probability of using twice the same WOTS^+ key pair is negligible.

E.5 FORS

FORS (Forest of Random Subset) [10] is a few time signature, that is, it can sign messages in $\mathcal{M} = \{0,1\}^{a \cdot k}$. The high-level idea of the signature is using k Merkle trees of height a, every tree having $t = 2^a$ leaves. The public key is an output of a tweakable hash function on the k roots of the trees. The secret key is a seed $\mathsf{Seed}_{\mathsf{sk}}$ of a PRF. Every leaf is an output of a tweakable hash of a secret value that was generated from the PRF (with respect to the location of the leaf). A message is hashed to a string of length $a \cdot k$, then split into k a-bits strings. The i-th string is interpreted as an index j, which represents the j-th leaf in the i-th tree. The signature is the authentication path from the leaf to the root. The authentication path includes all the siblings of the nodes on the path from the leaf to the root. Unforgeability follows in a standard way from the properties of Merkle trees.

E.6 Putting everything together: SPHINCS+

$\mathsf{SPHINCS}^+$ is obtained combining all the previous building blocks in a hypertree through the hash-and-sign paradigm. The hypertree is build using XMSS in all layer but the last one, where FORS is used. The hash of the message determines both the index of the FORS instance to be used to sign, and the message digest that gets signed using the FORS trees. Fig. 3 contains an overview of the structure of $\mathsf{SPHINCS}^+$.

F Proof of Lemma 6 (unforgeability of FSS.WOTS)

Proof. Recall that in the non-adaptive game, the adversary chooses the oracle queries *before* getting pk. To prove unforgeability, define the following hybrid games (see Algorithm 1 for the formal definition):

– $\mathcal{H}_1^j$ for $j = 0, \ldots, w-1$: Each of these hybrids is equal to Experiment 4, except for the signing oracle and the generation of the pk. To answer the message signing query, the oracle computes the signature applying the THF **Th** at most $b_i - j$ times (if $b_i > 0$):
  - It computes
$$a_i^j = \begin{cases} j & \text{if } b_i > j \\ b_i - 1 & \text{if } 0 < b_i \leq j \\ 0 & \text{if } b_i = 0 \end{cases} \tag{4}$$
  - It samples a secret key of appropriate length: $\mathsf{sk}_i \xleftarrow{\$} \{0,1\}^{n+c(w-a_i^j-1)}$.
  - It computes the signature as $\sigma_i \leftarrow c^{a_i^j, b_i a_i^j}(\mathsf{sk}_i, i, \mathsf{Seed})$.

  Observe that if $b_i = 0$, the oracle returns just a random string that is $n + c(w-1)$ bits long. The public key pk is generated from that signature by finishing each chain, that is: $\mathsf{pk}_i \leftarrow c^{b_i, w-1-b_i}(\sigma_i, i, \mathsf{Seed})$. Observe that $\mathcal{H}_1^0$ is exactly Experiment 4, and in $\mathcal{H}_1^{w-1}$ the signature is computed applying the THF at most once.
– $\mathcal{H}_2$: The experiment is the same as in $\mathcal{H}_1^{w-1}$ but now $\mathcal{A}$ looses also if it outputs a valid forgery (m^*, σ^*) where there exists an i such that $b_i^* < b_i$ and $c^{b_i^*, b_i - b_i^*}(\sigma^*, i, \mathsf{Seed}) \neq \sigma_i$. Observe that now by the properties of the checksum a valid forgery contains at least one i such that $b_i^* < b_i$, and in such a case it also holds that
$$c^{0,1}(\mathsf{sk}_i, i, \mathsf{Seed}) = \sigma_i = c^{b_i^*, b_i - b_i^*}(\sigma^*, i, \mathsf{Seed}) \ .$$
  For any such i the values that get computed from the forgery during verification fully agree with those values that are computed during the verification of the signature by the last game hop. This means that we can use an $\mathcal{A}$ that wins in $\mathcal{H}_2$ to break the SM-PRE security of the THF.

Algorithm 1: Hybrids for FSS.WOTS unforgeability

Differences are highlighted in red.

Hybrid $\mathcal{H}_1^j$ **for** $j = 0, \ldots, w-1$

$(\lambda_r, \lambda_s w, \ell_1, \ell_2, c, \mathsf{Seed}) \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^{\lambda_s})$
$(\mathcal{Q} = \{m\}, S) \leftarrow \mathcal{A}_1(1^\lambda)$
$(m_1, \ldots, m_{\ell_1}) \leftarrow [m]_w$
$C \leftarrow \sum_{i=1}^{\ell_1}(w - 1 - m_i)$ // checksum
$(c_1, \ldots, c_{\ell_2}) \leftarrow [C]_w$, $c_{i'} \in [0, w-1]$
$(b_1, \ldots, b_\ell) \leftarrow (m_1, \ldots m_{\ell_1}, c_1, \ldots, c_{\ell_2})$
for $i = 1, \ldots, \ell$ **do**
  if $b_i > j$ **then** $a_i^j \leftarrow j$
  else if $0 < b_i \leq j$ **then** $a_i^j \leftarrow b_i - 1$
  else $a_i^j \leftarrow 0$
  $\mathsf{sk}_i \xleftarrow{\$} \{0,1\}^{n+c(w-a_i^j-1)}$
  $\sigma_i \leftarrow c^{a_i^j, b_i - a_i^j}(\mathsf{sk}_i, i, \mathsf{Seed})$
  $\mathsf{pk}_i \leftarrow c^{b_i, w-1-b_i}(\sigma_i, i, \mathsf{Seed})$
$\mathsf{sk} \leftarrow (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$
$\sigma \leftarrow (\sigma_1, \ldots, \sigma_\ell)$
$\mathsf{pk} \leftarrow (\mathsf{Seed}, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}_2(\mathsf{pk}, S, (m, \sigma))$
if $(\mathsf{Sign.Ver}_{pk}(m^*, \sigma^*) = 1) \ \wedge \ (m^* \neq m)$ **then**
  return 1
else return 0

Hybrid $\mathcal{H}_2$

$(\lambda_r, \lambda_s w, \ell_1, \ell_2, c, \mathsf{Seed}) \leftarrow \mathsf{GenCh}(1^{\lambda_r}, 1^{\lambda_s})$
$(\mathcal{Q} = \{m\}, S) \leftarrow \mathcal{A}_1(1^\lambda)$
$(m_1, \ldots, m_{\ell_1}) \leftarrow [m]_w$
$C \leftarrow \sum_{i=1}^{\ell_1}(w - 1 - m_i)$ // checksum
$(c_1, \ldots, c_{\ell_2}) \leftarrow [C]_w$, $c_{i'} \in [0, w-1]$
$(b_1, \ldots, b_\ell) \leftarrow (m_1, \ldots m_{\ell_1}, c_1, \ldots, c_{\ell_2})$
for $i = 1, \ldots, \ell$ **do**
  if $b_i > 0$ **then** $a_i \leftarrow b_i - 1$
  else $a_i \leftarrow 0$
  $\mathsf{sk}_i \xleftarrow{\$} \{0,1\}^{n+c(w-a_i-1)}$
  $\sigma_i \leftarrow c^{a_i, b_i - a_i}(\mathsf{sk}_i, i, \mathsf{Seed})$
  $\mathsf{pk}_i \leftarrow c^{b_i, w-1-b_i}(\sigma_i, i, \mathsf{Seed})$
$\mathsf{sk} \leftarrow (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$
$\sigma \leftarrow (\sigma_1, \ldots, \sigma_\ell)$
$\mathsf{pk} \leftarrow (\mathsf{Seed}, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}_2(\mathsf{pk}, S, (m, \sigma))$
if $(\mathsf{Sign.Ver}_{pk}(m^*, \sigma^*) = 1) \ \wedge \ (m^* \neq m) \ \wedge$
$\forall \ i \ : \ b_i^* < b_i, \ c^{b_i^*, b_i - b_i^*}(\sigma^*, i, \mathsf{Seed}) = \sigma_i$ **then**
  return 1
else return 0

Let $\mathsf{Succ}^{\mathcal{H}}(\mathcal{A})$ be the probability that $\mathcal{A}$ wins the hybrid $\mathcal{H}$, that is, that $\mathcal{H}$ returns $b = 1$. We split the proof in the following three lemmas.

Lemma 28. *If* $\mathbf{Th}_j$ *is* SM-UD *secure (cf. Definition 16), then* $\mathcal{H}_1^j$ *and* $\mathcal{H}_1^{j-1}$ *are computationally indistinguishable for every* $j \in [1, w-1]$.

Proof. The proof is essentially equal to the proof of [31, Claim 2]. We include it to show the changes that having a compressing THF yields.

First, observe that H_1^{j-1} and $\mathcal{H}_1^j$ behave exactly the same if $\mathcal{A}$ queries a message m such that $a_i^{j-1} = a_i^j$, that is, if m is such that $b_i \leq j-1$ for all $i = 1, \ldots, \ell$. Thus, to have non-negligible distinguishing advantage $\mathcal{A}$ has to query a message m such that $\exists\ i \in \{1, \ldots, \ell\}$ such that $b_i \geq j$. Assume now that there exists a PPT adversary $\mathcal{A}$ such that

$$|\mathsf{Succ}^{\mathcal{H}_1^j}(\mathcal{A}) - \mathsf{Succ}^{\mathcal{H}_1^{j-1}}(\mathcal{A})| = \varepsilon\ .$$

We show a PPT algorithm $\mathcal{B}$ that wins the SM-UD experiment of $\mathbf{Th}_j$ (cf. Experiment 5) exploiting $\mathcal{A}$. Upon receiving Seed from the experiment and (m, S) from $\mathcal{A}$, $\mathcal{B}$ computes $(b_1, \ldots, b_\ell)$, computes the tweaks $(T_{1,j}, \ldots, T_{\ell,j})$, and simulates the secret key as follows:

$$\phi_i \ \text{ such that } \begin{cases} \phi_i \xleftarrow{\$} \{0,1\}^{n+c(w-1)} & \text{if } b_i = 0 \\ \phi_i \xleftarrow{\$} \{0,1\}^{n+c(w-b_i)} & \text{if } 0 < b_i \leq j-1 \\ \mathcal{O}_{\mathsf{Seed}}(T_{i,j}, b) & \text{if } b_i \geq j \end{cases}$$

where $\mathcal{O}_{\mathsf{Seed}}(\cdot, b)$ is the oracle in the SM-UD experiment. Then it generates σ and pk as in $\mathcal{H}_1^j$. Observe that if $b = 0$, that is, if $\mathcal{O}_{\mathsf{Seed}}(\cdot, b)$ returns random strings, $\mathcal{B}$ is perfectly simulating $\mathcal{H}_1^j$. In the other case, $\mathcal{B}$ is perfectly simulating $\mathcal{H}_1^{j-1}$. At the end of the game, $\mathcal{B}$ returns 1 if $\mathcal{A}$ returns a valid forgery, and 0 otherwise. Therefore, the advantage of $\mathcal{B}$ is

$$\begin{aligned} \mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathbf{Th}_j,\ell}(\mathcal{B}^{\mathcal{A}}) &= |\Pr[1 \leftarrow \mathsf{SM\text{-}UD}^1_{\mathcal{B},\mathbf{Th}_j,\ell}(\lambda_s)] - \Pr[1 \leftarrow \mathsf{SM\text{-}UD}^0_{\mathcal{B},\mathbf{Th}_j,\ell}(\lambda_s)]| \\ &= |\mathsf{Succ}^{\mathcal{H}_1^j}(\mathcal{A}) - \mathsf{Succ}^{\mathcal{H}_1^{j-1}}(\mathcal{A})| = \varepsilon\ . \end{aligned}$$

□

Lemma 29. *If the THF family is* SM-TCR *secure,* $\mathcal{H}_1^{w-1}$ *and* $\mathcal{H}_2$ *are indistinguishable.*

Proof. The proof of this claim is essentially equal to the proof of [31, Claim 3]. The only difference is that here we require a series of hybrid games to deal with the compression along the chain (as in the proof of Lemma 28).

Lemma 30. $\Pr[1 \leftarrow \mathcal{H}_2] \leq \mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathbf{Th},\ell}$,

Proof. The proof is essentially the same as the proof of [31, Claim 3]. The only difference is that here we require a series of hybrid games to deal with the compression along the chain (as in the proof of Lemma 28).

G Proof of Lemma 12 (Unforgeability of FSS.SPHINCS)

Lemma 31 (Security of FSS.SPHINCS).

- *If* $\mathbf{Th} = \{\mathbf{Th}_i\}_i$ *is a family of THFs that is* SM-UD, SM-TCR, *and* SM-PRE *secure,* PRF, $\mathsf{PRF}_{\mathsf{msg}}$ *are PRFs,* $\mathbf{H} = \{\mathbf{H}_i\}_i$ *is a family of* SM-TCR *secure THFs, and* H_{msg} *is a* ITSR *secure compressing THF, then* FSS.SPHINCS *is unforgeable under adaptive CPA.*
- *Assume that* $\mathcal{A}$ *cannot break the ITSR security of* H_{msg} *nor invert* PRF *and* $\mathsf{PRF}_{\mathsf{msg}}$. *If* **Th** *and* **H** *are families of compressing THF, then* FSS.SPHINCS *is secure for signer against an adversary* $\mathcal{A}$ *with running time at most* $2^{c_s\lambda_s/2}$ *(in the QROM).*
- *If* **Th** *and* **H** *are families of* SM-TCR *secure THFs, then* FSS.SPHINCS *is secure for the recipient.*

Proof (Sketch.). The proof combines all the techniques presented in the proofs of FSS.WOTS, FSS.XMSS, and FSS.FORS. In this sketch we only address the question of how to prove *adaptive* unforgeability. The proof follows the reduction $\mathsf{SPHINCS}^+$ from NIST's specification:

- $\mathcal{H}_0$: this is the adaptive $\mathsf{uf}-\mathsf{cma}$ experiment.
- $\mathcal{H}_1$: same as $\mathcal{H}_0$, except that the outputs of PRF are replaced with random strings.
- $\mathcal{H}_2$: same as $\mathcal{H}_1$, except that the outputs of $\mathsf{PRF}_{\mathsf{msg}}$ are replaced with random strings.
- $\mathcal{H}_3$: differs from $\mathcal{H}_3$ in that the game is lost if the FSS.FORS part of the signature (the authentication path and/or the preimage) is different than what the signer would generate.
- $\mathcal{H}_4$: differs from $\mathcal{H}_2$ in that $\mathcal{A}$ looses if it outputs a valid forgery (msg, σ) where the FSS.FORS signature part of σ contains a secret value which is the same as that of a honestly generated signature on msg, but was not contained in any of the signatures obtained by $\mathcal{A}$ querying the signing oracle.

We want to bound the success probability of an adversary $\mathcal{A}$ against the $\mathsf{uf}-\mathsf{cma}$ security of FSS.SPHINCS. Clearly the difference in success probability of $\mathcal{A}$ in $\mathcal{H}_0$ and $\mathcal{H}_1$ is bounded by the security of PRF, and the difference in success probability of $\mathcal{A}$ to wins in $\mathcal{H}_1$ and $\mathcal{H}_2$ is bounded by the security of $\mathsf{PRF}_{\mathsf{msg}}$.

The difference in success probability of $\mathcal{A}$ in $\mathcal{H}_3$ and $\mathcal{H}_2$ is bounded by the SM-UD, SM-TCR, and SM-PRE security of $\mathbf{Th}$, $\mathbf{H}$ or H_{msg}. Indeed, to distinguish the games $\mathcal{A}$ should return a forgery such that the FSS.FORS section is not the same as the signer would generate. Doing that requires breaking the SM-PRE, SM-UD, and SM-TCR security of $\mathbf{Th}$, $\mathbf{H}$ or H_{msg} as it would imply either finding a collision in one of the trees, or forging a FSS.WOTS signature. More details on the reduction can be found in [32, Appendix B, Proof of theorem 2].

The difference in success probability of $\mathcal{A}$ in $\mathcal{H}_3$ and $\mathcal{H}_4$ is bounded by $1/2$ times the success probability of $\mathcal{A}$ in $\mathcal{H}_3$. The reason is that the secret values which were not disclosed to $\mathcal{A}$ before still contain 1 bit of entropy, even for an unbounded $\mathcal{A}$.

Finally, we need to bound the success probability of $\mathcal{A}$ in $\mathcal{H}_4$. This experiment is exactly ITSR's experiment, since if $\mathcal{A}$ wins in $\mathcal{H}_4$, the FSS.FORS signature must be valid and consist only of values that have been observed by $\mathcal{A}$ in previous signatures. Hence, the success probability of $\mathcal{A}$ in $\mathcal{H}_4$ is bounded by ITSR's security. This concludes the proof.

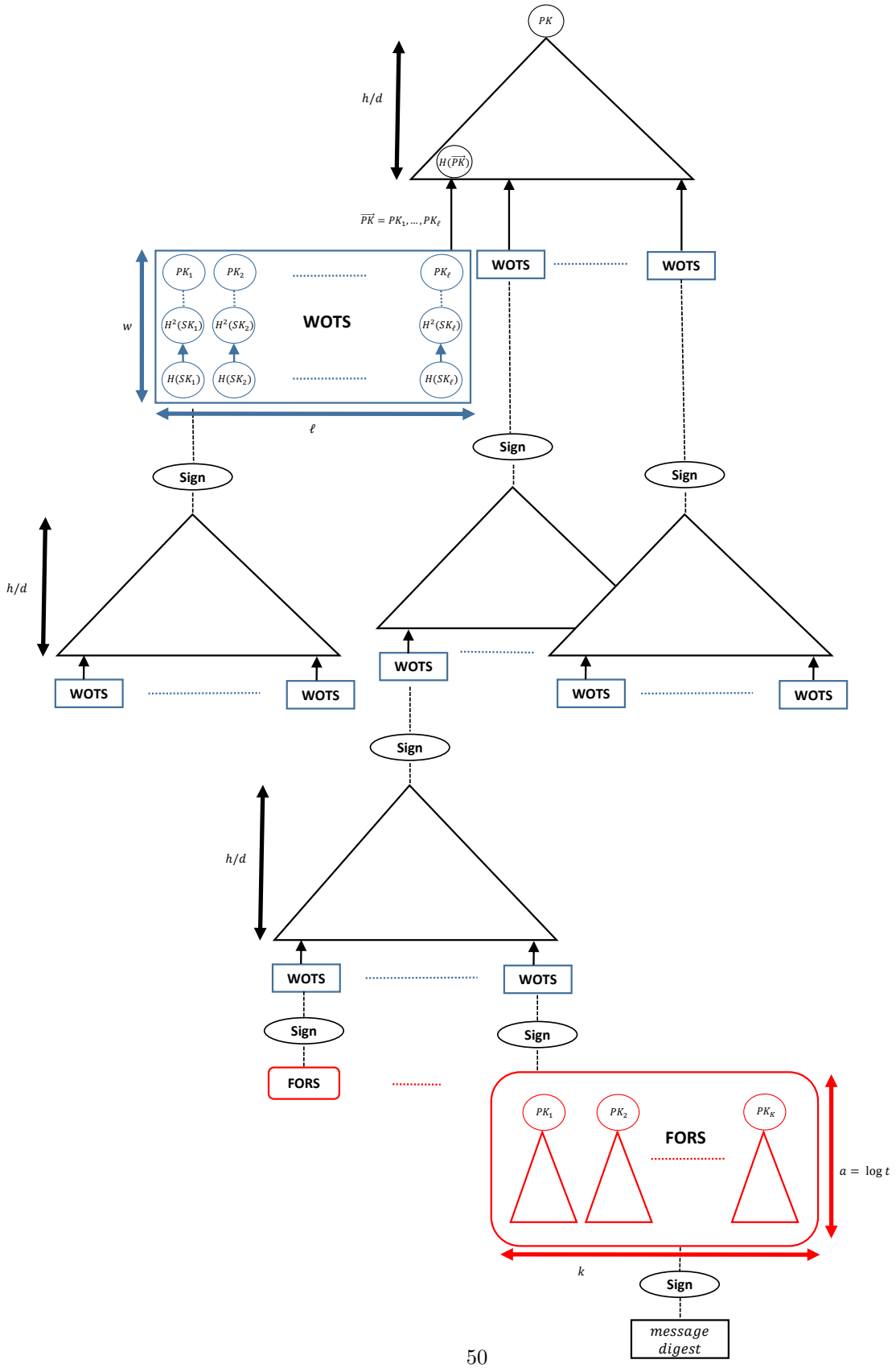

Fig. 3. SPHINCS$^+$ structure: all the leaves in the internal trees are the one-time signature WOTS$^+$, the trees are Merkle trees. The top tree is a Merkle tree of all the potential pks, the root of the top tree is the pk of the scheme. The leaves of the lowest level of SPHINCS$^+$ are the few-time FORS signatures.