Grafting Trees: a Fault Attack against the SPHINCS framework

Laurent Castelnovi[1⋆], Ange Martinelli[2], and Thomas Prest[2]

[1] Alten Sud-Ouest
laurent.castelnovi@live.fr
[2] Thales Communications & Security
{ange.martinelli,thomas.prest}@thalesgoup.com

Abstract Because they require no assumption besides the preimage or collision resistance of hash functions, hash-based signatures are a unique and very attractive class of post-quantum primitives. Among them, the schemes of the SPHINCS family are arguably the most practical stateless schemes, and can be implemented on embedded devices such as FPGAs or smart cards. This naturally raises the question of their resistance to implementation attacks.
In this paper, we propose the first fault attack against the framework underlying SPHINCS, GRAVITY-SPHINCS and SPHINCS$^+$. Our attack allows to forge any message signature at the cost of a single faulted message. Furthermore, the fault model is very reasonable and the faulted signatures remain valid, which renders our attack both stealthy and practical. As the attack involves a non-negligible computational cost, we propose a fine-grained trade-off allowing to lower this cost by slightly increasing the number of faulted messages. Our attack is generic in the sense that it does not depend on the underlying hash function(s) used.

1 Introduction

Hash-based signatures base their security solely on the hardness of finding collisions or (second) preimages for hash functions, and do not require any additional assumption. This striking property makes them stand out even among other post-quantum schemes. From a strict viewpoint of security assumptions, one can hardly expect better as Rompel [Rom90] has shown that secure signatures exist if and only if one-way functions exist, and Song [Son14] has extended this result to quantum adversaries. In addition, hash-based signatures are easy to analyze and their security does not depend on the choice of the underlying primitive.

Since Lamport [Lam79] proposed the first hash-based signature scheme – which could sign only one message –, several constructions have been proposed to improve its efficiency. They can be separated in two classes: stateful and stateless constructions. Stateful signatures, introduced by Merkle in 1990 [Mer90],

⋆ Large parts of this work were done when Laurent Castelnovi was an intern at Thales.

constrain the signer to maintain a record of its used keys. Such a requirement may generate operational problems, in particular when the key is used by multiples servers. Stateless signatures, as introduced by Goldreich [Gol86], lift this requirement but at the cost of a huge blow-up in the signature time and size.

It is only recently that practical stateless constructions have been proposed. In 2015, Bernstein, Hopwood, Hülsing, Lange, Niederhagen, Papachristodoulou, Schneider, Schwabe and Wilcox-O'Hearn [BHH+15] proposed SPHINCS, a hash-based signature scheme which achieved both statelessness and a reasonable efficiency (with respect to the running time and signature size) by combining the constructions of Goldreich and Merkle, and using a few-time signature scheme. In 2017, two variations of SPHINCS were proposed to NIST's call for post-quantum cryptographic schemes [NIS16]: GRAVITY-SPHINCS [AE17b] by Aumasson and Endignoux, and SPHINCS+ [BDE+17] by Bernstein, Dobraunig, Eichlseder, Fluhrer, Gazdag, Hülsing, Kampanakis, Kölbl, Lange, Lauridsen, Mendel, Niederhagen, Rechberger, Rijneveld and Schwabe.

Hash functions can be implemented very efficiently on constrained devices and it is not surprising that several implementations of hash-based signatures on micro-controllers have been proposed [RED+08,HBB12], including an ARM implementation [HRS16] of SPHINCS. However, embedded devices are known to be sensitive to physical attacks such as side-channel analysis or fault attacks.

Since the seminal article of Boneh, DeMillo and Lipton [BDL97], fault attacks have proved to be the strongest kind of cryptanalysis on embedded devices. In a fault attack, we suppose that the attacker is strong enough to corrupt the internal state of an algorithm during its execution. While this supposes a rather powerful attacker, these conditions can often be fulfilled in real life and generally result in devastating attacks. However, to the best of our knowledge, no fault attack against hash-based signatures has been publicly proposed.

1.1 Our contribution

At a very high level, the SPHINCS framework (in this document, this notion encompasses the original SPHINCS scheme, as well as GRAVITY-SPHINCS and SPHINCS+) combines hash trees and several one-time signature schemes (OTS) inside a tree data structure in order to obtain a stateless signature scheme. We propose the first fault injection attack against the SPHINCS framework. The attack is done in two steps, a faulting part and a grafting part:

1. *The faulting step.* Two signatures for the same message are queried. During the second signature computation, a fault is provoked so that an OTS inside the SPHINCS framework ends up signing a different value than the first time. Usually, an OTS key is only used to sign a single value, but our fault attack compels it to do otherwise.
2. *The grafting step.* We show that the knowledge of the two signatures – the correct one and the faulted one – can be exploited to recover parts of the secret key of the OTS which was subjected to the fault, and therefore to

partially compromise it. In turn, an attacker will use this compromised OTS as a mean to authenticate a tree different from the one it is supposed to authenticate.
The attacker then generates a tree which is entirely under its control, and will use the compromised OTS to *graft* it to the SPHINCS tree, which is why we call this step the grafting step.

The grafted tree is chosen by the attacker and independent from the secret key, while allowing to generate valid signatures for some messages. We show that it is more than enough to provide universal forgery ability to an attacker while explaining how she can achieve it. The attack requires little power from the attacker – which makes it practical – and produces valid signatures, which renders it particularly stealthy.

Whereas this attack comes with a non-negligible computational cost for each forgery, we propose trade-offs to lower this cost by slightly increasing the number of faulted signatures available to the attacker. Our attack is generic in the sense that it targets the SPHINCS framework: it is successful regardless of the underlying hash function used, and is indifferent to the specificities of the original SPHINCS, GRAVITY-SPHINCS or SPHINCS$^+$.

1.2 Roadmap

First we will introduce the notions related to trees. In section 2, we will give a quick overview of hash-based signatures constructions. Then we will describe our attack in section 3. The grafting step will be presented before the faulting step, as it only requires two signatures by the same OTS and is indifferent to whether they were obtained through a fault attack. We will then discuss countermeasures in section 3.4. Section 4 will conclude this paper and expose open questions.

1.3 Related works

Our grafting technique relies on and extend a result by Groot Bruinderink and Hülsing [GBH16] about the security of common OTS's under two-message attacks.

Due to their relative novelty, the resistance of post-quantum cryptographic schemes against fault attacks has only recently been studied. A wide array of attacks against lattice-based schemes has been covered in [BBK16], and a loop-abort attack has been demonstrated in [EFGT16]. For schemes based on supersingular isogenies, loop-abort and point decompression attacks have been investigated in [BG15,Ti17,GW17]. While we know of no fault attack against hash-based signatures, countermeasures have been studied in [MKAA16].

Hash functions have been targeted by fault attacks on their keyed operation modes. Notably Hemme and Hoffmann [HH11] propose a differential fault analysis allowing the attacker to recover the internal state of a SHA-1 instance

using about 1000 faulted hashes with the fault targeting a specific variable in the computation. A similar attack targeting SHA-3 was presented in 2015 [BGS15]. This attack involves random single bit faults on 80 messages to recover most of the SHA-3 internal state. In comparison, our attack requires only one fault, and the precision needed by the attacker in order to succeed is very low.

2 Preliminaries

We first set up the notations, then introduce the security models used for signature schemes and present Merkle's and Goldreich's constructions.

2.1 Notations and conventions

We denote by λ the security parameter of a signature scheme. $H : \{0,1\}^* \to \{0,1\}^\lambda$ denotes a cryptographic hash function. We will note vectors in bold lowercase. Whenever we consider faulting the value of a vector $\mathbf{v}$, we denote by $\mathbf{v}^\star$ the faulted value of $\mathbf{v}$.

2.2 Dendrologic notations

We recall notions related to trees. We suppose that the definitions of (balanced) binary tree, parent, child, sibling, root, leaf and internal node are known.

We denote by $\&\mathfrak{f}$ the address of a leaf $\mathfrak{f}$. The height of a tree is the length of the longest path between the root and any node. Two nodes are at the same height (resp. in the same layer, resp. at the same level) if they lie at the same distance from the root.

In this article, we also deal with hypertrees, which are trees whose nodes are trees. The height of a hypertree is the sum of the largest heights of each layer. To avoid confusion between the hypertree and the node trees, we will refer to the *layers* of the hypertree and the *levels* of the node trees.

As an example, Figure 3 depicts a toy version of a SPHINCS hypertree. This example has 2 layers of height 2, hence it has a total height of 4. SPHINCS-256 has 12 layers of height 5, so its total height is 60.

2.3 Security models for signature schemes

We briefly recall some classical security notions for signature schemes.

Definition 21 *Existential forgery* – An adversary is able of existential forgery if there exists a message m such that she can exhibit a valid (message, signature) pair (m, σ^*) where σ^* was not produced by the legitimate signer.

Definition 22 *Universal forgery* – An adversary is able of universal forgery if for any message m, she can exhibit a valid signature σ^*.

Any attacker able of universal forgery is able of existential forgery. For more formal notations, we refer the reader to e.g. [GBH16].

2.4 Hash-based signatures

Hash-based signatures stem from a very simple idea: the public key is a commitment of the secret key, whereas the signature of a message consists of revealing some information from which the verifier can recompute the commitment.

A simple way to build a hash-based one-time signature (OTS) can be defined as follows. Given a hash function H, let a secret key $S = (S_1, S_2)$ and the message space be $[\![0, M-1]\!]$ for an integer M. The public key is

$$P = (P_1, P_2) = (H^M(S_1), H^M(S_2)).$$

The signature of a message $\mathsf{m} \in [\![0, M-1]\!]$ is

$$\sigma = (\sigma_1, \sigma_2) = (H^{\mathsf{m}}(S_1), H^{M-\mathsf{m}}(S_2)).$$

The verifier only needs to check that

$$(\sigma_1^{M-\mathsf{m}}, \sigma_2^{\mathsf{m}}) = (P_1, P_2).$$

This scheme is one-time if H is preimage-resistant. However, it is not two-time, since given signatures for messages $\mathsf{m}_1 < \mathsf{m}_2$ one can compute signatures for any $\mathsf{m}_1 < \mathsf{m}_3 < \mathsf{m}_2$, thus breaking existential unforgeability.

We draw the readers' attention to the fact that in the scheme we presented, the public key can be computed from any valid signature. This is a common feature among hash-based signatures and will effectively be the case for all the schemes considered in this paper. From this feature, one does not need the public key if it is able to authenticate it. It allows to derive many-times signatures schemes from OTS by computing signature trees.

In the rest of this section, we present an OTS, two few-times signatures (FTS) and two many-times signatures, a stateful one and a stateless one.

2.4.1 An OTS: WOTS

WOTS is a one-time signature whose principle was enunciated by Merkle [Mer90] following an idea from Winternitz. It is parameterized by three values:

- w: the size of the words used by WOTS
- ℓ_1: the fixed number of words of size w of the messages to be signed
- ℓ_2: the fixed number of words of size w of the parity-check value used in the signature algorithm.

Let $\ell = \ell_1 + \ell_2$ be the fixed number of words of size w of the signature. We can now detail WOTS.

- Keygen()
  1. Let $\mathsf{sk} = (s_i)_{i=1,\ldots,\ell}$ where the s_i are uniformly random w-bits words;
  2. For $1 \leqslant i \leqslant \ell$, $p_i \leftarrow H^{w-1}(s_i)$;

3. *public key:* $\mathsf{pk} \leftarrow (p_1, \ldots, p_\ell)$, *private key:* sk.

- $\mathsf{Sign}(\mathsf{m}, \mathsf{sk})$
  1. Express m in base w: $\mathsf{m} = (\mathsf{m}_1 \mathsf{m}_2 \ldots \mathsf{m}_{\ell_1})_w$;
  2. Compute the parity-check value $C \leftarrow \sum_{i=1}^{\ell_1} (w - 1 - \mathsf{m}_i)$;
  3. Express C in base w: $C = (C_1 C_2 \ldots C_{\ell_2})_w$;
  4. $\mathbf{b} = (b_1, b_2, \ldots, b_\ell) \leftarrow (\mathsf{m}_1, \ldots, \mathsf{m}_{\ell_1}, C_1, \ldots, C_{\ell_2})$ – we will later call it the *b-vector* of m;
  5. For $1 \leqslant i \leqslant \ell$, $\sigma_i \leftarrow H^{b_i}(s_i)$;
  6. *signature:* $\boldsymbol{\sigma} \leftarrow (\sigma_1, \ldots, \sigma_\ell)$.
- $\mathsf{Verify}(\mathsf{m}, \boldsymbol{\sigma}, \mathsf{pk})$
  1. Compute the b-vector of m as in the signature algorithm (steps 1-4);
  2. Accept if and only if $\forall i \in [\![1, \ell]\!]$, $p_i = H^{w-1-b_i}(\sigma_i)$.

Remark 1. GRAVITY-SPHINCS implements the unmasked version of WOTS described above, but SPHINCS$^{(+)}$ replaces WOTS by a variant, WOTS+, which uses random masks in order to replace collision resistance by second preimage resistance. Since our attack is indifferent to the presence of masks, we present it only in the case of the mask-less scheme (WOTS) as it makes our exposition simpler.

Parameters: In practice, SPHINCS-256 (it is the practical instantiation of SPHINCS proposed in [BHH+15]), GRAVITY-SPHINCS and SPHINCS$^+$ set:

$$\begin{cases} \ell_1 = 64, \\ \ell_2 = 3, \\ w = 16. \end{cases}$$

These parameters offer a good trade-off between size and speed and are usually chosen in the most recent constructions.

In [GBH16, Section 5], the authors study (among other scenarii) WOTS resistance against existential forgery under two-random-message attacks. They argue that the probability of being able to forge the signature of a random message m_3 knowing the signature of two known random messages m_1 and m_2 is roughly equal to the probability that for all $0 \leqslant i < \ell$, the i-th coordinate of the b-vector of m_3 is lower that the i-th coordinate of the b-vector of m_1 or m_2. We will see that this existential forgery on WOTS can be extended to a universal forgery on the SPHINCS framework in section 3.1.

2.4.2 FTS

In order to expand an OTS construction to a FTS signature scheme, one can generate many OTS, link the public keys using an authentication tree and use the root of this tree as public key. To sign a message, the signer only needs to choose a subset of the OTS generated and sign the message with each of them. The verifier only has to recover the various public keys from the signatures and to check if the public key is equal to the authentication value associated with the public keys and the corresponding authentication path.

All three algorithms based on the SPHINCS framework make use of different FTS. In the context of our attack, one only needs to keep two facts in mind:

- just like WOTS, the FTSs are entirely deterministic;
- in each of these FTSs, the public key can be directly computed from a valid signature.

2.4.3 A stateful construction: Merkle's scheme

Merkle's scheme [Mer90] is based on hash trees, which are (generally balanced) binary trees in which each internal node is defined as the hash of its two concatenated child nodes. In Merkle's construction, each leaf of a hash tree is an OTS public key: such a hash tree is called a *Merkle tree*. The public key for Merkle's scheme is the root of the Merkle tree and the private key is the set of all the OTS private keys paired with the OTS public keys.

For a leaf $\mathfrak{f}$ of a Merkle tree, we denote by $A(\mathfrak{f})$ and call authentication path of $\mathfrak{f}$, the unique set of nodes (with one node per level, excluding the root) such that the root of the Merkle tree can be recomputed from $\mathfrak{f}$ and $A(\mathfrak{f})$.

To sign a message, the signer chooses an unused OTS key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ in a leaf of the Merkle tree: he signs m with sk_i and sends the signature together with pk_i and its authentication path $A(\mathsf{pk}_i)$. The receiver verifies that: 1) the message's signature using the OTS is valid, and 2) the general public key (which is the root of the Merkle tree) can be recomputed from pk_i and $A(\mathsf{pk}_i)$.

This scheme has two major drawbacks. First, the signature time – or memory requirement – is exponential in the tree height, since the whole tree must either be stored or recomputed each time a signature is performed.[1] Second, the signer must keep track of the used OTS key pairs, which makes the scheme stateful.

2.4.4 A stateless construction: Goldreich's signature

Goldreich's proposal [Gol86] solves the two aforementioned issues: it is still based on a binary tree whose leaves are OTS public keys, but internal nodes are now OTS key pairs. Each node of the tree is uniquely indexed by a bitstring which is used, together with a seed which is part of the overall private key, to pseudo-randomly generate the node's key pair.

The scheme's public key is the root's public key and its private key is composed of the root's private key and the seed referred to above. To sign a message, one randomly selects a leaf and then signs the message with this key pair. Each node (specifically the public key inside) between this leaf and the root is then signed, together with its sibling node, by its father node. The verifier accepts if and only if all the signatures are valid.

The drawback of this approach is the signature size. For 128 bits of pre-quantum security, one needs a 256-layer tree; using for example a Winternitz OTS with parameter $w = 16$ (see section 2.4.1) and a hash function with a 256-bit output, the signature size reaches 1.65 MB.

---

[1] There exist techniques which get rid of exponential running time at the expense of somewhat increasing state size, such as the tree traversal algorithm of [BDS08].

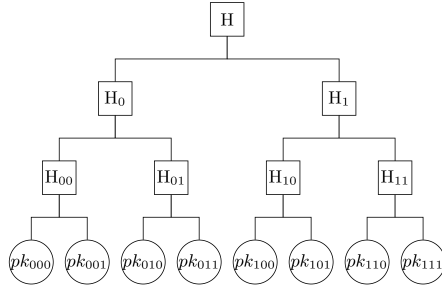

Figure 1: A Merkle tree. Two merging arrows mean "the parent is the hash of the two children". The authentication path of $\mathfrak{f} = pk_{000}$ would be $A(\mathfrak{f}) = \{pk_{001}, \mathrm{H}_{01}, \mathrm{H}_1\}$.

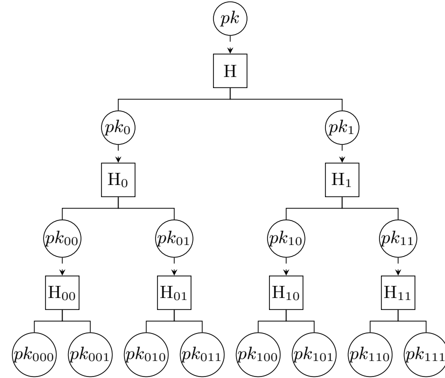

Figure 2: A Goldreich tree. In addition to the notations of Figure 1, a dashed arrow means "the leaf signs the root below".

2.5 SPHINCS

The aim of SPHINCS is twofold: to achieve moderate signature time and size, and to get rid of any kind of state. To reach this goal, the SPHINCS tree is designed as a Goldreich tree whose nodes are Merkle trees.

In this new configuration, each leaf in a Merkle tree is used to sign the root of a Merkle tree located in the layer below. Such a construction can also be found in GMSS [BDK+07] and XMSS [BDH11,HRB13]. Moreover, leaves of a SPHINCS tree sign a public key of a few-time signature (FTS) scheme, which security is not compromised if the same key pair is used on few different messages. This is summarized in Figure 3.

In order to have a quick overview of SPHINCS, one can see it as a combination of 3 types of trees. Namely:

1. The SPHINCS hypertree: a Goldreich tree of height h (60 in SPHINCS-256) organised in d layers (12 in SPHINCS-256). Each layer's leaf signs the root of a Merkle tree.
2. Merkle trees of size h/d (= 5 in SPHINCS-256) whose leaves are public keys for the OTS used in the Goldreich construction: WOTS.
3. The FTS used to sign the message is signed by the last layer of the hypertree.

We now delve a bit deeper into SPHINCS's machinery.

A SPHINCS tree of height h can be seen as a Goldreich tree of d layers with Merkle trees of height h/d instead of nodes. In [BHH+15], a few modifications have been made to the Merkle tree construction described in section 2.4.3. One of them is important for our work: in the leaves of SPHINCS's Merkle subtrees, all WOTS public keys are compressed as follows: their ℓ parts are considered as

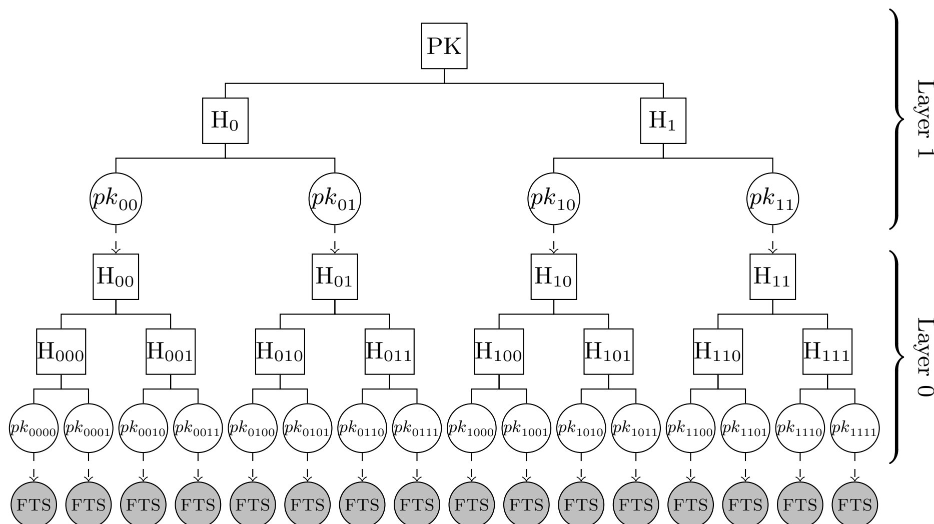

Figure 3: A SPHINCS hypertree of height 4, which can be seen as a Goldreich tree of two layers with Merkle trees of height 2 in node places. In addition to the notations of Figures 1 and 2, gray discs denote FTS instances (HORST for the original SPHINCS, PORST for GRAVITY-SPHINCS, FORS for SPHINCS$^+$).

leaves of a binary hash tree; this tree's root is then computed applying this rule: if a node has no sibling, then it is lifted to a higher level in the tree until it has one. The tree's root stands as the compressed WOTS public key.

Like in Goldreich's construction, where each node is indexed, each *leaf* has an address in SPHINCS, which contains its layer in the SPHINCS hypertree, the number of its Merkle tree in the layer and its position in the Merkle tree.

We now describe the original SPHINCS signature scheme (which we will call O-SPHINCS to disambiguate it from the SPHINCS framework).

- $\mathsf{Keygen}()$
  Pick a pair of seeds $(S_1, S_2) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ at random and generate the top Merkle tree (the one in layer $d-1$), whose root is the overall public key pk. The private key is $\mathsf{sk} \leftarrow (S_1, S_2)$.
- $\mathsf{Sign}(\mathsf{m}, \mathsf{sk})$
  1. Generate 2 pseudo-random values $(R_1, R_2) \in (\{0,1\}^\lambda)^2$ from m and S_2;
  2. Compute $D = H(R_1 || \mathsf{m})$;
  3. $\mathsf{idx} \leftarrow$ the h leftmost bits of R_2;
  4. Generate the HORST key pair of index idx;
  5. $\boldsymbol{\sigma}_H$ = signature of D using this HORST key pair;
  6. $\boldsymbol{\sigma}_0$ = signature of the HORST public key using the WOTS key pair at layer 0, which is (in compressed form) in the leaf $\mathfrak{f}_0$ with $\&\mathfrak{f}_0 = d || \mathsf{idx}$;

7. For $1 \leqslant i < d$, $\boldsymbol{\sigma}_i$ = signature of the root of the Merkle tree containing $\mathfrak{f}_{i-1}$, using the WOTS key pair in the appropriate leaf $\mathfrak{f}_i$ of the layer i;
8. *signature:* $\boldsymbol{\sigma} = (\mathsf{idx}, R_1, \boldsymbol{\sigma}_H, \boldsymbol{\sigma}_0, A(\mathfrak{f}_0), \boldsymbol{\sigma}_1, A(\mathfrak{f}_1), \ldots, \boldsymbol{\sigma}_{d-1}, A(\mathfrak{f}_{d-1}))$.

- $\mathsf{Verify}(\mathsf{m}, \boldsymbol{\sigma}, \mathsf{pk})$
  1. Compute $D = H(R_1||\mathsf{m})$;
  2. Compute the HORST public key assuming $\boldsymbol{\sigma}_H$ is valid;
  3. For $0 \leqslant i < d$:
     (a) assume that the WOTS signature $\boldsymbol{\sigma}_i$ is valid and deduce a WOTS public key from it and from the root computed the step before[2];
     (b) assume that $A(\mathfrak{f}_i)$ is correct and compute the root of its Merkle tree;
  4. compare this last root to the SPHINCS public key: accept if and only if they are equal.

2.6 Gravity-SPHINCS and SPHINCS$^+$ modifications

GRAVITY-SPHINCS was proposed in [AE17b], with several changes to O-SPHINCS aiming at improving its performance and signature size. The changes relevant to our attack are the following:

1. The top layer of the hypertree is now cached as it is always used in the signature algorithm, and its height is increased from 5 to 20 (thus lowering the number of layers in the hypertree). As a result, the number of leaves in the topmost Merkle tree is increased from 32 to 2^{20}.
2. The index of the FTS instance is now derived directly from the message and a public salt computed by the signer from its secret key (this is well summarized in [AE17a, Figure 3]). As a consequence, the verifier can verify the index and the attacker cannot choose it anymore – but we will see that it is easy to get around this protection.

Independently, SPHINCS$^+$ was proposed in [BDE$^+$17]. The modification relevant to our attack is that the message digest md and FTS index idx are computed as

$$(\mathsf{md}\|\mathsf{idx}) \leftarrow H(\mathsf{r}, \mathsf{pk}, \mathsf{m}), \tag{1}$$

where r is a public salt generated by the signer from the message and a private seed. This change in index generation is similar to the one of GRAVITY-SPHINCS. For simplicity, this document will only focus on the parameter sets targeting NIST's security level 1.

We will see that these modifications, while theoretically increasing the cost of our attack, actually have a very limited impact on its efficiency.

Parameters. O-SPHINCS, GRAVITY-SPHINCS and SPHINCS$^+$ propose parameters to provide 128 bits of *quantum* security against existential forgery (assuming 256-bits messages). Table 1 summarizes these parameters.[3] We note that [AE17b] proposed several trade-offs between efficiency and signature size, as well as variations on the number of signatures allowed by the context.

[2] Which is the value whose signature is $\boldsymbol{\sigma}_i$.
[3] We choose the NIST-oriented version of GRAVITY-SPHINCS according to [AE17b].

Scheme	Security	w	ℓ	h	d	Sig. size (kB)
O-SPHINCS	128	16	67	60	12	41
GRAVITY-SPHINCS	128	16	67	60	6	27
SPHINCS$^+$-128f	128	16	67	60	20	17
SPHINCS$^+$-128s	128	16	67	64	8	8

Table1: Parameters for O-SPHINCS, GRAVITY-SPHINCS and SPHINCS$^+$

3 A grafting attack against the SPHINCS framework

In this section we propose a new kind of attack against the hyper-tree structure of the SPHINCS framework. The goal of this attack is to insert a branch under our control below a leaf $\mathfrak{f}_{d-1}$ (which is an OTS public key) of the top layer. In order to do this, one must be able to provide a signature for the root of the branch which is valid for the key $\mathfrak{f}_{d-1}$. Once the root is authenticated – and the branch grafted, one has total control over the branch and can easily modify any of the nodes inside, both by modifying the seed used for its generation and by randomizing unverifiable values.

In the rest of this section we will detail the principles of the grafting attack and its implications in terms of security. Finally, we will provide a practical fault attack that leads to a universal forgery on the SPHINCS framework, and we will discuss different complexity trade-offs. At last we will provide a short overview of the possible countermeasures to our attack.

3.1 Grafting a branch in the SPHINCS hyper-tree

Let us target a leaf $\mathfrak{f}_{d-1}$ of the top layer of the hyper-tree. We suppose that the corresponding WOTS key has signed two different values, which the attacker knows along with their signatures. According to [GBH16], she is able to forge a WOTS signature with a probability $p_{\mathrm{W}} \approx 2^{-34}$. We convert this existential forgery capability against WOTS into a universal forgery capability against any SPHINCS-style scheme. In order forge a message m, we proceed as follows:

1. Randomly generate a seed such that the index of the FTS to be used falls under the targeted WOTS instance. It happens with probability p_{F}, where p_{F} is 1 for O-SPHINCS,[4] 2^{-20} for GRAVITY-SPHINCS,[5] and 2^{-3} (resp. 2^{-8}) for SPHINCS$^+$-128f (resp. -128s).
2. From this seed and the message m, compute the signature up to the penultimate layer of the hyper-tree. With probability p_{W}, the root of the layer can be signed by the attacker capacity on the corresponding WOTS signature.

[4] O-SPHINCS provides the verifier no mechanism to check that the FTS index is valid. An attacker can therefore directly pick a suitable index, hence the probability 1.

[5] The probability to find such a seed is equal to the inverse of number of leaves in the top-most layer of the hyper-tree, which is 2^{-20} for GRAVITY-SPHINCS.

3. Complete the signature using the legitimate authentication path of the signer, known from the legitimate signature of any message whose authentication path in the hyper-tree goes through $\mathfrak{f}_{d-1}$.

The naive way to achieve the forgery described above is to randomly choose the seed (from which are generated the OTS and all the FTS used during the signature) in order to fulfill two requirements:

1. the FTS used is under the targeted WOTS instance: this happens with the probabilities p_{F} stated in section 3.1.
2. the attacker can sign the root of the Merkle's tree used in the penultimate layer with the targeted WOTS secret key: this happens with probability p_{W}.

To find a seed which simultaneously fulfills both requirements, an attacker needs to try about $1/p_{\text{F}}p_{\text{W}}$ seeds for each message. These trials can be done entirely offline. We note that the number of hash computations is even higher as every trial costs around 2^{15} hashes. However, it is possible to do better.

Indeed, even though an honest signer generates (the OTS secret keys corresponding to the leaves of) Merkle trees with a private seed, there is no way a verifier can check that this was effectively the case. Therefore, the search of a suitable Merkle tree for the penultimate layer (by suitable, we mean that its root can be signed with the targeted WOTS key) can be *decorrelated* from the search for a suitable FTS index. This makes the number of trials drop from $1/p_{\text{F}}p_{\text{W}}$ to $1/p_{\text{F}} + 1/p_{\text{W}}$.

In addition, a signature does not contain whole Merkle trees but only, for each of them, a leaf $\mathfrak{f}_i$[6] and its authentication path $A(\mathfrak{f}_i)$; this reduces signature size as well as verification time. However, it also allows to speed up forgery as the attacker does not have to generate a suitable Merkle tree but only a leaf $\mathfrak{f}_{d-2}$ and an authentication path $A(\mathfrak{f}_{d-2})$ which looks like an authentication path in a suitable Merkle tree. To do this, the attacker can simply choose all the values of $A(\mathfrak{f}_{d-2})$ at random, except the last one. She then tries several values for this last value, until the root computed from $\mathfrak{f}_{d-2}$ and $A(\mathfrak{f}_{d-2})$ can be signed with $\mathfrak{f}_{d-1}$. With this improvement, each new trial now costs *one* hash instead of 2^{15} hashes.

With these improvements, the cost of a forgery on any SPHINCS scheme drops from up to $2^{15}/p_{\text{F}}p_{\text{W}}$ hashes down to $1/p_{\text{F}} + 1/p_{\text{W}}$ hashes. As an illustration, this represents a drop from 2^{69} to 2^{34} for GRAVITY-SPHINCS.

3.2 Fault Injection against the SPHINCS Framework

As we have been seen before, the entire attack depends on the capability of the attacker to obtain two distinct WOTS signatures for the same secret key. In the context of the SPHINCS framework, the whole construction of the hyper-tree is

[6] Precisely, the signature contains a WOTS signature from which one can recover $\mathfrak{f}_i$.

deterministic and a signature is entirely dependent of both the message and the secret key. This characteristic leads to the fact that no OTS can sign distinct messages, thus ensuring the security of the scheme.

In the following we present a fault injection attack allowing an attacker to recover the signature of two different messages with the same WOTS key.

Let us denote $\mathbf{sk} = (s_1, s_2, \ldots, s_\ell)$ the targeted WOTS secret key corresponding to $\mathfrak{f}_{d-1}$ and δ the Merkle tree root authenticated by it. We ask for the signature of a message m about which we suppose, without loss of generality, that at the last step it requires signing δ. We note δ's WOTS signature $\boldsymbol{\sigma}_{d-1} = (\sigma_{d-1,1}, \ldots \sigma_{d-1,\ell})$. We receive the overall signature

$$\boldsymbol{\sigma} = (\mathsf{idx}, R_1, \boldsymbol{\sigma}_H, \boldsymbol{\sigma}_0, A(\mathfrak{f}_0), \ldots, \boldsymbol{\sigma}_{d-1}, A(\mathfrak{f}_{d-1})).$$

In the next step, we will ask again the signature of the same message m. As the algorithm is entirely deterministic, the resulting signature should be the same as $\boldsymbol{\sigma}$. However we will perturb the operations done in the computation of the authentication path $A(\mathfrak{f}_{d-2})$. This perturbation will result in the computation of a Merkle tree root $\delta^\star$ distinct from δ. The resulting WOTS signature $\boldsymbol{\sigma}^\star_{d-1}$ of $\delta^\star$ gives the attacker the possibility to mount the grafting attack as shown previously. An overview of the fault can be seen in Figure 4.

A nice feature of the fault model is that it is a very weak one. Indeed it verifies the following properties:

- only a single fault is needed per signature as a single fault in the computation of the Merkle tree of the penultimate layer fulfills the required modification;
- the fault is very permissive as we do not use the actual value of the faulted variable: we need the variable to change but do not need to know the actual value of the change;
- the fault can be done in a wide time period. Indeed, since the verification algorithm uses $A(\mathfrak{f}_{d-2})$ to compute $\delta^\star$, this authentication path must be faulted: otherwise, the attacker would deduce from it the legitimate root δ instead of the faulted one $\delta^\star$. This implies that one cannot directly fault the nodes which computations are redone by the signature verifier, but faulting all the other nodes will lead to a successful attack. In other words, one can fault any node "below" the authentication path, whereas it is not of interest for our purposes to fault any node "above". In practice, it means that, in O-SPHINCS 33 227 hash computations may eventually be the target of the fault while 273 352 hash computations are available as targets in GRAVITY-SPHINCS with parameters given in Table 1.
  These numbers stands for roughly 6% of the whole O-SPHINCS computation and 18% of the whole GRAVITY-SPHINCS[7].

[7] If the top layer of GRAVITY-SPHINCS is not cached, this percentage falls drastically but GRAVITY-SPHINCS also becomes very slow for these parameters, requiring about 2^{30} hashes per signature.

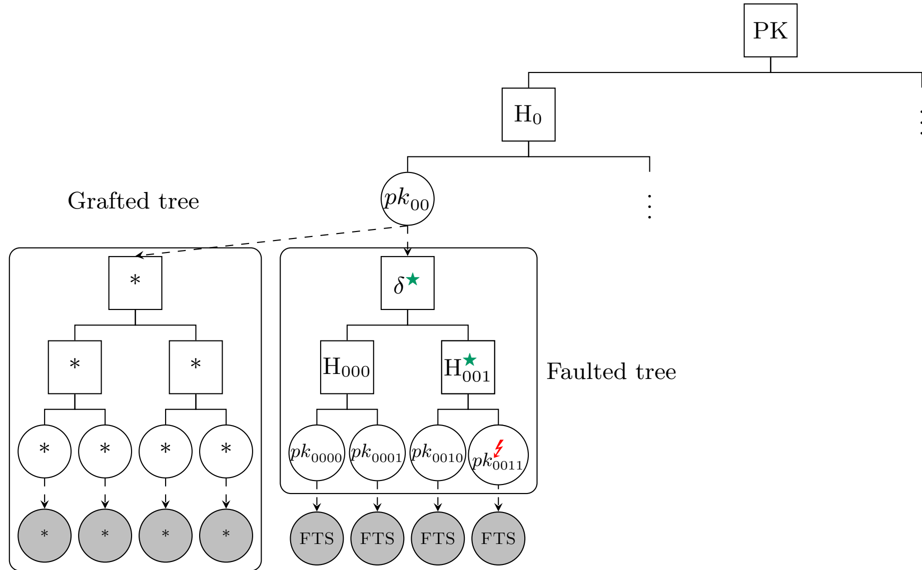

Figure 4: Principle of our attack with a SPHINCS hyper-tree of height 4. Fault injection is denoted by a lightning and the affected elements of the faulted Merkle tree by a star. We note that the public key PK is not affected. The part of the hyper-tree which is below vertical dots is irrelevant to our attack.

Remark 2. The faulted overall SPHINCS signatures produced by this attack are valid. Indeed, $\boldsymbol{\sigma}^{\star}_{d-1}$ is the valid signature of $\delta^{\star}$, computed from $A(\mathfrak{f}_{d-2})^{\star}$, which is given in the overall signature $\boldsymbol{\sigma}^{\star}$. Moreover, all other elements of $\boldsymbol{\sigma}^{\star}$ are correct. Thus $\boldsymbol{\sigma}^{\star}$ is accepted by the verification algorithm.

3.3 Compromise between faulted signatures and computational power

We have seen that one can achieve universal forgery on the existing schemes of the SPHINCS family at the price of one faulted signature. However, each of the signatures forged comes at a non-negligible computational cost as one needs to try about $1/p_{\mathrm{W}} \approx 2^{34}$ values for the penultimate Merkle tree root (see section 3.1) to be able to use the capacity on WOTS. In this section we provide trade-offs between the number of faulted signatures allowed to the attacker and the computational cost needed to forge a signature.

3.3.1 Total break on WOTS

First, we estimate how many faulted signatures are necessary to recover an entire WOTS private key.

It is safe to model H as a random oracle. As a consequence of this hypothesis, when the computation of δ is faulted, $\delta^\star$ takes a uniformly random value between 0 and $2^\lambda - 1$. Each b_i for $1 \leqslant i \leqslant \ell_1$ is therefore 0 with probability $1/w$.

Then the checksum C follows the law of a sum of $\ell_1 = 64$ random variables (r.v.) following the uniform law over $[\![0, w-1]\!]$ which, thanks to the central limit theorem, we shall approximate by a normal law with parameters $\mu = \ell_1(w-1)/2$ and $\sigma^2 = \ell_1(w^2-1)/12$.

Let us write C in base w in the big-endian convention. Then $b_i = 0$ for $\ell_1 < i \leqslant \ell$ means $C \bmod w^j < w^{j-1}$ with $j = i - \ell_1$. This event has probability:

$$\mathbb{P}(C \bmod w^j < w^{j-1}) = \sum_{q=0}^{\lfloor \ell_1(w-1)/w^j \rfloor} \sum_{z=qw^j}^{qw^j + w^{j-1}-1} \mathbb{P}(C = z),$$

with $\mathbb{P}(C = z) = \rho_{\mu,\sigma}(z)/\sum_{n\in\mathbb{Z}} \rho_{\mu,\sigma}(n)$, where $\rho_{\mu,\sigma}(x) = \exp(-(x-\mu)^2/(2\sigma^2))$.

We obtain $\mathbb{P}(b_{65} = 0) \approx 1/w, \mathbb{P}(b_{66} = 0) \approx 0.098, \mathbb{P}(b_{67} = 0) \approx 2^{-30.7}$. This last value calls for a remark. It means that we have to ask for $2^{30.7}$ signatures in average to get s_{67}, which is a lot, but, in the same time, we need s_{67} to sign a root with probability $2^{-30.7}$. So, with respect to s_{67}, we can only look for $H(s_{67})$. Given the very high probability of finding this value ($\mathbb{P}(b_{67} = 1) \approx 0.80$), we shall suppose that the average number of signatures required to recover $s_1, \ldots, s_{66}$ is high enough to find $H(s_{67})$ with overwhelming probability, and therefore we do not care about s_{67} anymore. We will later see that this is verified in practice.

Finally, we rely on the values of $\mathbb{P}(b_{65} = 0)$ and $\mathbb{P}(b_{66} = 0)$ to justify this approximation which will be done hereafter: $b_1, b_2, \ldots, b_{66}$ are viewed as 66 uniform deviate in $[\![0, w-1]\!]$.

Let us now consider the number of signatures required on average to carry out the attack. Let X be the random variable which models the number of requested signatures to find $\mathbf{sk}$ (except s_{67}). Our problem then boils down to computing $\mathbb{E}(X)$.

Let $\{\boldsymbol{\sigma}^{(1)\star}, \boldsymbol{\sigma}^{(2)\star}, \ldots, \boldsymbol{\sigma}^{(n)\star}\}$ be the set of the n requested faulted signatures at a certain point in the attack. We define: $V_j^{(n)} := \left\{\sigma_{d-1,j}^{(i)\star} \mid 1 \leqslant i \leqslant n\right\}$, that is, the set of values taken by the jth coordinate of all received $\boldsymbol{\sigma}_{d-1}^{(i)\star}$'s, when the attacker has gathered n faulted signatures. We also define the event $B_n := \{\exists 1 \leqslant j < \ell \text{ s.t. } s_j \notin V_j^{(n)}\}$. It can be shown that $\mathbb{P}(X = n) = \mathbb{P}(B_{n-1} \cap \overline{B_n})$ and that it leads to:

$$\mathbb{P}(X = n) = \mathbb{P}(\overline{B_n}) - \mathbb{P}(\overline{B_{n-1}}). \tag{2}$$

Since the coordinates of $\boldsymbol{\sigma}_{d-1}^\star$ are pairwise independent and follow the same uniform law over $[\![0, w-1]\!]$ by assumption, we have:

$$\mathbb{P}(\overline{B_n}) = \prod_{j=1}^{\ell-1} \mathbb{P}\left(s_j \in V_j^{(n)}\right) = \left(1 - \left(\frac{w-1}{w}\right)^n\right)^{\ell-1}. \tag{3}$$

Composing equations 2 and 3 with the parameters set in the SPHINCS schemes, we estimate that $\mathbb{E}(X) \approx 74.5$. Note that this leads to a probability to find $H(s_{67})$ greater than $1 - 2^{-170}$, which is indeed more than enough to do so.

One whole WOTS private key can thus be recovered querying 74.5 faulted signatures on average for the parameters used in the SPHINCS schemes.

3.3.2 Trade-offs

We have seen that the attack can be mounted with only one faulted signature, with a non-negligible computational cost needed to forge every signatures, or that an attacker can make about 75 faulted signatures to ensure a free selection of the Merkle tree root. We now investigate the various trade-offs we can obtain by increasing step by step the number of faulted messages available to the attacker.

For this purpose, we extend Hülsing and Groot Bruinderink's [GBH16] reasoning. Let us denote by δ the root which WOTS signature we want to forge, and by $\delta^{(1)}, \delta^{(2)}, \ldots, \delta^{(n)}$ n uniformly random roots for which we have valid signatures by the same WOTS private key. Let $\mathbf{b} = (b_1, \ldots, b_\ell)$ be the b-vector of δ and $\mathbf{b}^{(i)} = (b_1^{(i)}, \ldots, b_\ell^{(i)})$ be the b-vector of $\delta^{(i)}$ for all $1 \leqslant i \leqslant n$. Then we can forge a valid WOTS signature for δ if and only if the following expression is true:

$$\bigwedge_{j=1}^{\ell} \bigvee_{i=1}^{n} \{b_j \geqslant b_j^{(i)}\}.$$

In order to estimate the probability of this event, we make the assumption that coordinates of the b-vector of a uniformly random word are pairwise independent and uniformly distributed in $[\![0, w-1]\!]$. If this assumption is clearly true for the first ℓ_1 coordinates, it is clearly not for the last ℓ_2 ones. However, we shall see later that the resulting theoretical probabilities are very close to probabilities obtained by simulations. Thus we work with this assumption for the sake of simplicity.

Moreover, we make the assumption that random roots $\delta^{(i)}$ are pairwise independent, *i.e.* that corresponding b-vectors are pairwise independent. By coordinates independence assumption:

$$\mathbb{P}\Big(\bigwedge_{j=1}^{\ell} \bigvee_{i=1}^{n} \{b_j \geqslant b_j^{(i)}\}\Big) = \mathbb{P}\Big(\bigvee_{i=1}^{n} \{b_j \geqslant b_j^{(i)}\}\Big)^{\ell} = \Big(1 - \mathbb{P}(b_j < b_j^{(1)})^n\Big)^{\ell} = \big(1 - \tfrac{1}{w^{n+1}} \textstyle\sum_{a=0}^{w-1} a^n\big)^{\ell}, \quad (4)$$

the second equality coming from b-vectors independence assumption and the last one by identical distribution assumption. Table 2 presents the average number of roots to try before finding one whose signature can be forged, based on the number of faulted signatures the attacker has – note that the attacker is supposed to have the legitimate signature in addition to the faulted ones. The numbers are obtained from equation 4, and are matched by experiments.

We can observe that the computational complexity of the forgery of a message m is essentially the sum of the complexities of three operations:

Faulted signatures	1	2	3	4	5	10	20
Number of trees to try	$2^{34.9}$	$2^{24.0}$	$2^{18.0}$	$2^{14.2}$	$2^{11.7}$	$2^{5.5}$	$2^{2.0}$

Table2: Number of grafted trees to generate randomly before finding one that can be signed by the faulted OTS, in function of the number of faulted signatures. We note that for 1 faulted signature, this number is $1/p_{\mathrm{W}}$.

- the number of seeds to try to assign a satisfying index to the message – from 1 for O-SPHINCS to 2^{20} for GRAVITY-SPHINCS;
- the number of Merkle tree root values to try before being able to forge the WOTS signature – depending on the number of faulted signatures, see Table 2;
- the complexity of the signature[8].

With this observation, we can state that only 3 faulted messages are needed to provide universal forgery against GRAVITY-SPHINCS at the cost of about 2^{20} hash computations.

3.4 Countermeasures

Generic countermeasures such as making the signature computation redundant can complicate our attack, but they may incur a significant overhead (for redundancy, a factor 2 in time and space). Indeed, a simple verification of the signature would not be efficient in our case as the attack provides valid signatures. Moreover, only a small part of the execution will be faulted and thus the redundancy must be checked for each of the roots of a Merkle sub-tree. However redundant computation is an efficient way to significantly constrain the attacker to a more powerful model as the fault should be exactly replicable on both executions. As generic countermeasures are well documented, our discussion will focus on the countermeasures that are specific to the SPHINCS framework.

In [MKAA16] the authors propose a specific recomputation designed to avoid faults in Merkle trees, called Recomputating with Swapped Nodes (RESN). Whereas the countermeasure provides efficient security at an acceptable overhead by lightly pipelining the circuit, it does not cover the Goldreich construction. The main impact to our attack additionally from classical recomputation methods is that it limits the fault to targeting only the computation of the root of the Merkle tree as any other faulted hash computation would be detected by the RESN. We note that, in this case, the faulted signature will not be a valid one anymore and the result could be verified with an additional overcost.

A naïve way to protect SPHINCS against our attack would be to compute the index of the FTS from public values instead of secret ones. Indeed, if one

[8] The complexity of the forged signature can be slightly lowered because the attacker does not need to compute valid values for the authentication path and can simply generate random values.

computes the index from the message and the public key, the attacker is not able anymore to choose the index of the message to sign.

However, the cure would be worse than the disease. Indeed, while our attack would be thwarted by this modification, a malicious user would now be able to provoke multi-collisions on the index idx of the FTS by trying several messages.

In the schemes studied, the number of FTS leaves is upper bounded by 2^{64}. This implies that given a fixed value for the index idx, an attacker can find k messages leading to this index with a computational effort about $k \times 2^{64}$. Therefore, this modification would lead to universal forgeability of the targeted schemes *without any fault*.

An efficient countermeasure would be to somehow link the different layers of the hyper-tree so that a fault in the computation of the tree would result in a non valid signature, *i.e.* a root value distinct from the public key. A simple check of the validity before returning the signature would prevent any fault attack. However, in order to link these layers, one cannot compute the OTS keys only from its index and the secret key, so the whole hyper-tree would have to be recomputed for each signature, ensuing a huge overhead in signing time.

4 Conclusion and Open Questions

In this paper we propose the first fault attack against signature schemes of the SPHINCS family. After an initial cost of a single faulted message, it allows to forge signatures for any message at an (offline) cost of 2^{34} hashes per message.

We proposed several trade-offs to lower this computational cost while slightly increasing the number of faulted messages. For any of the targeted schemes, we can forge any message at a cost of about 2^{20} hashes functions knowing only 3 faulted messages. Moreover, the fault model is very permissive.

While our attack can be thwarted by generic (but possibly costly) countermeasures against fault attacks, we did not find any specific countermeasure.

As demonstrated by this work, the deterministic nature of several hash-based signatures and their internal use of OTS can be a weakness against fault attacks. On the defensive side, an interesting line of work would be to propose hash-based constructions which offer some innate resilience against fault attacks.

On the offensive side, a natural extension of this work would be to implement the proposed fault attack in practice. Our attack target the SPHINCS framework, but it would be interesting to extend it to other multi-tree constructions such as multi-tree XMSS or GMSS. One could also devise an alternative way (other than fault injection) to recover two distinct WOTS signatures for the same key, which would allow to apply our grafting attack. We leave this for future work.

Acknowledgements

We would like to thank the anonymous PQCrypto reviewers for their helpful comments. We also thank Andreas Hülsing, whose insightful advices helped us

make our attack simpler, more generic and more powerful. Finally, we acknowledge the support of the French *Programme d'Investissement d'Avenir* under national project RISQ.

References

AE17a. Jean-Philippe Aumasson and Guillaume Endignoux. Clarifying the subset-resilience problem. Cryptology ePrint Archive, Report 2017/909, 2017. https://eprint.iacr.org/2017/909. 10

AE17b. Jean-Philippe Aumasson and Guillaume Endignoux. Improving stateless hash-based signatures. Cryptology ePrint Archive, Report 2017/933, 2017. https://eprint.iacr.org/2017/933. 2, 10

BBK16. Nina Bindel, Johannes A. Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. 2016. 3

BDE+17. Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe. SPHINCS+, 2017. https://sphincs.org/. 2, 10

BDH11. Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - a practical forward secure signature scheme based on minimal security assumptions. In *PQ Crypto*, volume 7071 of *LNCS*, pages 117–129. Springer, 2011. 8

BDK+07. Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. Merkle signatures with virtually unlimited signature capacity. In *Applied Cryptography and Network Security*, volume 4521 of *LNCS*, pages 31–45. Springer, 2007. 8

BDL97. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997. 2

BDS08. Johannes A. Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In *PQCrypto*, volume 5299 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2008. 7

BG15. Johannes Blömer and Peter Günther. Singular curve point decompression attack. In *FDTC*, pages 71–84. IEEE Computer Society, 2015. 3

BGS15. Nasour Bagheri, Navid Ghaedi, and Somitra Kumar Sanadhya. Differential fault analysis of SHA-3. In *INDOCRYPT*, volume 9462 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2015. 4

BHH+15. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 368–397. Springer, 2015. 2, 6, 8

EFGT16. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop abort faults on lattice-based fiat-shamir & hash'n sign signatures. *SAC*, 2016. 3

GBH16. Leon Groot Bruinderink and Andreas Hülsing. "Oops, I did it again" – Security of one-time signatures under two-message attacks. IACR Cryptology ePrint Archive, 2016. http://eprint.iacr.org/2016/1042. 3, 4, 6, 11, 16

Gol86. Oded Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In *CRYPTO '86*, volume 263 of *LNCS*, pages 104–110. Springer, 1986. 2, 7

GW17. Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In *PQCrypto*, volume 10346 of *Lecture Notes in Computer Science*, pages 93–106. Springer, 2017. 3

HBB12. Andreas Hülsing, Christoph Busold, and Johannes A. Buchmann. Forward secure signatures on smart cards. In *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2012. 2

HH11. Ludger Hemme and Lars Hoffmann. Differential fault analysis on the SHA1 compression function. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, pages 54–62, 2011. 3

HRB13. Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal Parameters for $XMSS^{MT}$. In *Security Engineering and Intelligence Informatics*, volume 8128 of *LNCS*, pages 194–208. Springer, 2013. 8

HRS16. Andreas Hülsing, Joost Rijneveld, and Peter Schwabe. ARMed SPHINCS computing a 41KB signature in 16KB of RAM. In *PKC 2016*, volume 9614 of *LNCS*, pages 446–470. Springer, 2016. 2

Lam79. Leslie Lamport. Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979. 1

Mer90. Ralph C. Merkle. A certified digital signature. In *CRYPTO' 89*, volume 435 of *LNCS*, pages 218–238. Springer, 1990. 1, 5, 7

MKAA16. Mehran Mozaffari-Kermani, Reza Azarderakhsh, and Anita Aghaie. Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarcked on ASIC. *ACM Transactions on Embedded Computing Systems*, 16(2), 2016. article 59. 3, 17

NIS16. NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. 2

RED+08. Sebastian Rohde, Thomas Eisenbarth, Erik Dahmen, Johannes A. Buchmann, and Christof Paar. Fast hash-based signatures on constrained devices. In *CARDIS*, volume 5189 of *Lecture Notes in Computer Science*, pages 104–117. Springer, 2008. 2

Rom90. John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394. ACM, 1990. 1

Son14. Fang Song. A note on quantum security for post-quantum cryptography. In *PQCrypto*, volume 8772 of *Lecture Notes in Computer Science*, pages 246–265. Springer, 2014. 1

Ti17. Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. In *PQCrypto*, volume 10346 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2017. 3