From: pqc-forum@list.nist.gov on behalf of Mikhail Kudinov <mkudinov@qapp.tech>
Sent: Thursday, July 23, 2020 11:10 AM
To: pqc-forum
Subject: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

Dear all,

In this comment, we would like to point out a flaw of existing security proofs of the SPHINCS+ hash-based scheme. Particularly, we would like to pay attention to security proofs of the underlying WOTS+ scheme with preimage resistance (PRE) requirement replaced by second preimage resistance (SPR) + "at least two preimages for every image" requirements [see eq. (14) in Round 2 submission] or decisional second preimage resistance (DSPR) + SPR requirements [see Bernstein et al. "The SPHINCS+ signature framework" 2019].

Both of these approaches are based on the claim that in the case where the given image has several preimages under some cryptographic hash function, the original preimage is information-theoretically hidden among all preimages (see "Case 2" in the Proof of Theorem 2 in [Hülsing et al. "Mitigating Multi-Target Attacks in Hash-based Signatures" 2016] and "SM-DSPR success probability" in the proof of Claim 23 in [Bernstein et al. "The SPHINCS+ signature framework" 2019]). Though this claim is quite reasonable in the case of a single hash function query, the situation becomes much more complicated when one deals with a chain of hash functions like in the WOTS+ scheme.

Let h_i with i=1,...,w-1 be a hash function used to obtain a value at i'th level of the WOTS+ scheme from the one at (i-1)'th level. That is pk_j = h_{w-1}(h_{w-2}(… h_1(sk_j) …)), where sk_j and pk_j are elements of secret and public key respectively and w is a Winternitz parameter (commonly w = 16). Here we assume that all bitmasks are included in h_i. Let IMG_i be an image set of h_i, and let PREIMG_i(y) be a set of all preimages for given y taken from IMG_i. The proposed security proofs are based either on assumption that for each y one has |PREIMG_i(y)| > 1, or that it is computationally hard to recognize whether |PREIMG_i(y)| = 1 or not. The latter is called a DSPR property [D.J. Bernstein, A. Hülsing "Decisional second-preimage resistance: When does SPR imply PRE?" 2019].

Consider the set WOTS_IMG_i = h_i(h_{i-1}(... h_1({0,1}^n) ...)) that is an image set of the whole WOTS+ chain up to level i from a set of all possible secret keys {0,1}^n (n is a security parameter, typically equals to 256). One can reasonably expect that for a secure hash function built in the chain functions and i > 1, |WOTS_IMG_i| < |IMG_i| because of collisions at levels 1, … , i-1. Let WOTS_PREIMG_i(y) be a set of preimages of y under h_i belonging to WOTS_IMG_{i-1}. Having a Challenger's signature, a WOTS+-breaking adversary is able to choose a position in the chain where |WOTS_PREIMG_i(y)| = 1, even though |PREIMG_i(y)| > 1 for some known element y in the WOTS+ structure. In the result, the adversary manages to forge a signature avoiding breaking SPR property (because the forgery consists of the same element used by the Challenger), and by choosing elements having |PREIMG_i(y)| > 1 or |PREIMG_i(y)| = 1 with a proper probability, avoiding breaking DSPR property. Thus the reduction proof fails.

We note that the security proof of the original SPHINCS scheme [Bernstein et al. "SPHINCS: practical stateless hash-based signature" 2015] which is based on PRE+SPR+undetectability (UD) assumptions does not have this flaw, though shows lower security level for the same scheme parameters. We also note that the updated detailed security proof of the WOTS+ scheme based on PRE+SPR+UD assumptions can be found in https://arxiv.org/abs/2002.07419.

With kind regards,
Mikhail Kudinov, Evgeniy Kiktenko, Aleksey Fedorov
Russian Quantum Center (www.rqc.ru) and QApp (www.qapp.tech)
--

From: pqc-forum@list.nist.gov on behalf of Andreas Hülsing <ietf@huelsing.net>
Sent: Friday, July 31, 2020 7:27 AM
To: Mikhail Kudinov; pqc-forum
Cc: contact@sphincs.org
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

Dear Mikhail, Evgeniy, and Aleksey, dear all,

Due to the summer holidays, the following is not agreed on with the whole SPHINCS+ team and hence should be considered my personal opinion.

Thank you for evaluating our proposal and pointing us to this mistake in the proof. You are right that our reasoning that the input used to compute the image is information theoretically hidden is incorrect in the context of hash chains. It should be noted that

a) this a mere mistake in the proof and does not imply an attack, and

b) in any case, the non-tight proof still applies (as you also state).

The state of affairs is as follows. Looking into the existing proof, there seems not to be any quick fix for this issue that corrects the existing proof. However, this motivated another look at the whole thing and there seems to be an easier proof (following the non-tight proof) that works specifically for SPHINCS+. The very rough outline is as follows: The critical part of the tight proof was written for stateful schemes where WOTS is used to sign adversarialy chosen messages. However, in SPHINCS and SPHINCS+ the messages signed using WOTS are fully controlled by the honest user (or the reduction). That means that we only require security under known-message attacks for which case the non-tight proof becomes tight as the reduction does not have to guess.

We will work on an update after the vacation time is over. We will also continue to look into actual fixes for the existing proof as the above idea only works for stateful schemes when modelling the message digest function as a (quantum-accessible) random oracle, or making additional, stronger hardness assumptions like the existence of chameleon hash functions.

Best wishes,

Andreas

On 23-07-2020 17:09, Mikhail Kudinov wrote:

> Dear all,
>
> In this comment, we would like to point out a flaw of existing security proofs of the SPHINCS+ hash-based scheme. Particularly, we would like to pay attention to security proofs of the underlying WOTS+ scheme with preimage resistance (PRE) requirement replaced by second preimage resistance (SPR) + "at least two preimages for every image" requirements [see eq. (14) in Round 2 submission] or decisional second preimage resistance (DSPR) + SPR requirements [see Bernstein et al. "The SPHINCS+ signature framework" 2019].
>
> Both of these approaches are based on the claim that in the case where the given image has several preimages under some cryptographic hash function, the original preimage is information-theoretically hidden among all preimages (see "Case 2" in the Proof of Theorem 2 in [Hülsing et al. "Mitigating Multi-

Kerman, Sara J. (Fed)

From:	Михаил Кудинов <outlook_B4764F62462D62E4@outlook.com>
Sent:	Sunday, October 31, 2021 5:23 PM
To:	pqc-comments
Cc:	pqc-forum; andreas@huelsing.net
Subject:	ROUND 3 OFFICIAL COMMENT: SPHINCS+
Attachments:	SPHINCS+-updated.pdf

Dear all,

Attached you find a paper that describes a new, tight security proof for SPHINCS+. Our paper fixes the flaw that was pointed out by Kudinov, Kiktenko, and Fedorov last year, following the outline we gave in the response to that message back then. More details can be found in the attached file.

Best regards,
Mikhail & Andreas

Security of WOTS-TW scheme with a weak adversary

Andreas Hülsing[1] and Mikhail Kudinov[1,2]

[1] Eindhoven University of Technology, Eindhoven, Netherlands
[2] Russian Quantum Center, QApp, Skolkovo, Moscow 143025, Russia

Abstract. In 2020, Kudinov, Kiktenko, and Fedorov pointed out a flaw in the tight security proof of the SPHINCS$^+$ construction. This work gives a new tight security proof for SPHINCS$^+$. The flaw can be traced back to the security proof for the used Winternitz one-time signature scheme (WOTS).
We give the first standalone description of the WOTS variant used in SPHINCS$^+$ that we call WOTS-TW. We provide a security proof for WOTS-TW and and multi-instance WOTS-TW in the EU-naCMA model, a non-adaptive chosen message attack setting where the adversary only learns the full public key after it made its signature queries. Afterwards, we show that this is sufficient to give a tight security proof for SPHINCS$^+$. We almost recover the same bound for the security of SPHINCS$^+$, with only a factor w loss, where w is the Winternitz parameter that is commonly set to 16.

Keywords: Post-quantum cryptography, hash-based signatures, SPHINCS$^+$ W-OTS, WOTS-TW.

1 Introduction

Hash-based signatures recently received a lot of attention. Hash-based signatures are widely considered the most conservative choice for post-quantum signature schemes. At the time of writing, SPHINCS$^+$, a stateless hash-based signature scheme, is a third round alternate candidate in the NIST PQC competition. However, NIST highlighted over and over that

> "NIST sees SPHINCS+ as an extremely conservative choice for standardization. If NIST's confidence in better performing signature algorithms is shaken by new analysis , SPHINCS+ could provide an immediately available algorithm for standardization at the end of the third round."
> (Dustin Moody on the pqc-forum mailing list by after new attacks on Rainbow and GeMSS were published, January 21, 2021)

This work was funded by an NWO VDI grant (Project No. VI.Vidi.193.066). Part of this work was done while M.K. was still affiliated with the Russian Quantum Center, QApp. Date: October 31, 2021

One more supporting argument for the security of SPHINCS$^+$ would be a tight security reduction that allows to derive attack complexities for a given set of parameters. However, the tight proof for SPHINCS$^+$ that was given in [BHK$^+$19] turned out to be flawed [MK]. The flaw, pointed out by Kudinov, Kiktenko, and Fedorov is related to the proof of security of the used WOTS scheme. Although the flaw could not be translated into an attack, this resulted in an unsatisfactory situation. While there still exists a non-tight reduction for the security of SPHINCS$^+$, this reduction can not support the claimed security of the used SPHINCS$^+$ parameters.

In this work we give a new tight security proof for SPHINCS$^+$ that closes the gap again without modifying the scheme.

To make the proof easier accessible, we first extract the variant of WOTS scheme that is used in SPHINCS$^+$ and formally define it, naming it WOTS-TW. WOTS-TW is different from other WOTS variants in that it uses tweakable hash functions, introduce in [BHK$^+$19], to construct the function chains. We then prove the security of WOTS-TW in the EU-naCMA model. This model differs from the common EU-CMA model in that the adversary only receives the public key after it made its signature query. We choose this model because it allows for a tight security proof while it suffices for a proof of applications like SPHINCS$^+$. The important feature here is that a reduction can generate the WOTS-TW public key based on the signature query and does not have to guess that query. This is possible as WOTS-TW is used to sign roots of hash trees in applications like SPHINCS$^+$. Our new proof combines the work of Dods, Smart, and Stam [DSS05] that uses undetectability to plant preimage challenges, with the second-preimage resistance version of Hülsing [Hül13], and the approach of multi-target mitigation by Hülsing, Rijneveld, and Song [HRS16] and lifts it to the setting of tweakable hash functions.

While the single instance proof allows us to state the new proof in a more accessible setting, SPHINCS$^+$ uses multiple instances at once. Hence, we afterwards extend the result to multiple instances. This setting turns out to be slightly more involved as we have to allow for messages to depend on public keys of previously used instances. We end up with a proof in a slightly more involved model that, however, allows us to use the result in the security proof for SPHINCS$^+$. We conclude the work with the tight security proof for SPHINCS$^+$.

While we give a full proof for SPHINCS$^+$, we leave one aspect for future work. In section 2.5 we collect bounds on the success probability of generic attacks against the used properties of tweakable hash functions. So far only one of these bounds is fully proven. All others make some form of conjecture. We hope to be able to close this gap in future work.

The paper is organized as follows. We introduce necessary definitions and notations as well as describe the EU-naCMA security model in Section 2. At the end of Section 2 we also summarize the state of the art for generic security bounds. The description of the WOTS-TW scheme is given in Section 3. In Section 4 we provide a security reduction for WOTS-TW in the single instance setting and in Section 5 we lift the result to the multi-instance setting with possibly dependent messages. The security proof for SPHINCS$^+$ that uses WOTS-TW as a building block is then given in Section 6.

2 Preliminaries

In this section we introduce the definitions of building blocks, and security notions for hash functions and signatures that we use. We begin with the notion of a tweakable hash function, introduced in the construction of SPHINCS$^+$ [BHK$^+$19], and its security. Afterwards we move on to digital signatures.

2.1 Tweakable hash functions.

In this section we recall the definition of tweakable hash functions and related security notions from [BHK$^+$19]. These properties will later be used to prove the security of our WOTS-TW scheme.

Function definition. A *tweakable* hash function takes public parameters P and context information in form of a *tweak* T in addition to the message input. The public parameters might be thought of as a function key or index. The tweak might be interpreted as a nonce.

Definition 1 (Tweakable hash function). *Let $n, m \in \mathbb{N}$, $\mathcal{P}$ the public parameters space and $\mathcal{T}$ the tweak space. A tweakable hash function is an efficient function*

$$\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n,\ \mathrm{MD} \leftarrow \mathbf{Th}(P, T, M)$$

mapping an m-bit message M to an n-bit hash value MD *using a function key called public parameter $P \in \mathcal{P}$ and a tweak $T \in \mathcal{T}$.*

We will sometimes denote $\mathbf{Th}(P, T, M)$ as $\mathbf{Th}_{P,T}(M)$. In SPHINCS$^+$, a public seed *Seed* is used as public parameter which is part of the SPHINCS$^+$ public key. As tweak a so-called hash function address (**ADRS**) is used that identifies the position of the hash function call within the virtual structure defined by a SPHINCS$^+$ key pair. We use the same approach for WOTS-TW, i.e., the public parameter is a seed value that becomes part of the public key if WOTS-TW is used standalone. If it is used in a bigger structure like SPHINCS$^+$, the public parameters will typically be those used in the bigger structure and are therefore only part of that bigger structure public key. In this case, the hash addresses have to be unique within the whole bigger structure. Therefore, the address may contain a prefix determined by the calling structure.

Security notions. To provide a security proof for WOTS-TW we require that the used tweakable hash functions have certain security properties. We require the following properties or some variations of them which will be discussed below:

– *post-quantum single function, multi-target-collision resistance for distinct tweaks* (PQ-SM-TCR);

- *post-quantum single function, multi-target-preimage resistance for distinct tweaks* (PQ-SM-DT-PRE);
- *post-quantum single function, multi-target-undetectability for distinct tweaks* (PQ-SM-DT-UD).

These properties were already considered in previous work. We only slightly adapt them, introducing a *weak* variant that adds some limitation on when the adversary is allowed to get access to the tweakable hash function and in what way. Moreover, in the context of multi-instance constructions like SPHINCS$^+$, we need another generic extension to collections of tweakable hash functions, discussed at the end of the subsection.

We generally consider post-quantum security in this work. Therefore, we will omit the PQ prefix from now on and consider it understood that we always consider quantum adversaries. Since we are working in the post-quantum setting, we assume that adversaries have access to a quantum computer but honest parties do not. Hence, any oracles that implement secretly keyed functions only allow for classical queries.

In all of the properties an adversary can influence the challenges by specifying the tweaks used in challenges. We generally restrict this control in so far as we do not allow more than one challenge for the same tweak. As we have this restriction for all of our properties we do not add any label to the security notions for this.

Now we will discuss above properties and their variations.

SM-TCR. One can view SM-TCR as a variant of target-collision resistance [RSM09]. Consider an adversary $\mathcal{A}$ which consists of two parts $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}$ will play a two-stage game. $\mathcal{A}_1$ is allowed to adaptively specify p targets (multi-target). The target specification is implemented via access to an oracle implementing the function with a fixed public parameter (single-function as the same public parameter is used for all targets). Every query to this oracle defines a target. It is important that $\mathcal{A}$ is not allowed to query the oracle with the same tweak more than once. The adversary wins if it finds a collision for one of the targets.

Lets consider a variant of SM-TCR in which the adversary gets the description of the tweakable function only after he has made the queries. This variation allows to make quantum queries to the hash function only after the targets are specified. We call it Weak-SM-TCR. In the reduction we will use this variant of the property. We will denote such a variant as W-SM-TCR. Notice that our reduction proof will show that if the scheme is broken at least one of the required properties of the tweakable hash function is broken. One can see that breaking the W-SM-TCR is harder than SM-TCR (at least not easier), hence it is less likely that the property will be broken and this leads to a higher security of the scheme.

We formalize the above in the following definition.

Definition 2 (W-SM-TCR). *In the following let* **Th** *be a tweakable hash function as defined above. We define the success probability of any adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against the* W-SM-TCR *security of* **Th**. *The definition is parameterized by the number of targets p for which it must hold that $p \leq |\mathcal{T}|$. In the* *definition,* $\mathcal{A}_1$ *is allowed to make* p *queries to an oracle* $\mathbf{Th}(P, \cdot, \cdot)$*. We denote the set of* $\mathcal{A}_1$*'s queries by* $Q = \{(T_i, M_i)\}_{i=1}^{p}$ *and define the predicate* $\mathbf{DIST}(\{T_i\}_{i=1}^{p}) = (\forall i, k \in [1, p], i \neq k) : T_i \neq T_k$*, i.e.,* $\mathbf{DIST}(\{T_i\}_{i=1}^{p})$ *outputs 1 iff all tweaks are distinct.*

$$\begin{aligned}&\mathrm{Succ}_{\mathbf{Th},p}^{\mathrm{W-SM-TCR}}(\mathcal{A}) = \Pr[P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,\cdot)}(\ );\\&(j, M) \leftarrow \mathcal{A}_2(Q, S, P, \mathbf{Th}) : \mathbf{Th}(P, T_j, M_j) = \mathbf{Th}(P, T_j, M)\\&\wedge M \neq M_j \wedge \mathbf{DIST}(\{T_i\}_{i=1}^{p})]\end{aligned}$$

We define the insecurity of a tweakable hash function against p *target, time* ξ*,* W-SM-TCR *adversaries as the maximum success probability of any possibly quantum adversary* $\mathcal{A}$ *with* p *targets and running time* $\leq \xi$ *:*

$$\mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Succ}_{\mathbf{Th},p}^{\mathrm{W-SM-TCR}}(\mathcal{A})\}$$

SM-PRE. As for W-SM-TCR, SM-PRE is a two-stage game and can be seen as a variant of preimage resistance. Adversary $\mathcal{A}_1$ is allowed to specify p targets during the first stage. The speciation is again done by using an oracle that implements a tweakable hash function with a fixed public parameter. The adversary wins if it finds a preimage for one of the targets.

We again consider a weaker version of SM-PRE in which the adversary gets the description of the tweakable function only after he has made the queries. We will denote such a variant as W-SM-PRE. The intuition behind this variant is the same as in W-SM-TCR.

We formalize the above in the following definition.

Definition 3 (W-SM-PRE)**.** *In the following let* $\mathbf{Th}$ *be a tweakable hash function as defined above. We define the success probability of any adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against the* W-SM-PRE *security of* $\mathbf{Th}$*. The definition is parameterized by the number of targets* p *for which it must hold that* $p \leq |\mathcal{T}|$*. In the definition,* $\mathcal{A}_1$ *is allowed to make* p *queries to an oracle* $\mathbf{Th}(P, \cdot, x_i)$*, where* x_i *is chosen uniformly at random for the query* i *(the value of* x_i *stays hidden from* $\mathcal{A}$*). We denote the set of* $\mathcal{A}_1$*'s queries by* $Q = \{T_i\}_{i=1}^{p}$ *and define the predicate* $\mathbf{DIST}(\{T_i\}_{i=1}^{p})$ *as we did in the definition above.*

$$\begin{aligned}&\mathrm{Succ}_{\mathbf{Th},p}^{\mathrm{W-SM-PRE}}(\mathcal{A}) = \Pr[P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,x_i)}(\ );\\&(j, M) \leftarrow \mathcal{A}_2(Q, S, P, \mathbf{Th}) : \mathbf{Th}(P, T_j, M) = \mathbf{Th}(P, T_j, x_j)\\&\wedge \mathbf{DIST}(\{T_i\}_{i=1}^{p})]\end{aligned}$$

We define the insecurity of a tweakable hash function against p *target, time* ξ*,* W-SM-PRE *adversaries as the maximum success probability of any possibly quantum adversary* $\mathcal{A}$ *with* p *targets and running time* $\leq \xi$ *:*

$$\mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Succ}_{\mathbf{Th},p}^{\mathrm{W-PRE}}(\mathcal{A})\}$$

SM-UD. Also SM-UD is a variant of an established notion, in this case undetectability [DSS05], that makes use of a two stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. $\mathcal{A}_1$ specifies p targets during the first stage through oracle interactions. The oracle is initialized either with the tweakable hash function with a fixed public parameter or a random function. The adversary wins if it answers correctly whether it interacted with a random function or with a function **Th**. We formalize the above in the following definition. As for the previous notions, we consider the weak variant where $\mathcal{A}$ gets the description of the tweakable hash function only after the challenge queries are made.

Definition 4 (W-SM-UD)**.** *In the following let* **Th** *be a tweakable hash function as defined above. We define the success probability of any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the* W-SM-UD *security of* **Th**. *The definition is parameterized by the number of targets p for which it must hold that $p \leq |\mathcal{T}|$. First the challenger flips a fair coin b and and chooses a public parameter $P \xleftarrow{\$} \mathcal{P}$. Next consider an oracle $\mathcal{O}_P(\mathcal{T}, \{0,1\})$, which works the following: $\mathcal{O}_P(T, 0)$ returns $\mathbf{Th}(P, T, x_i)$, where x_i is chosen uniformly at random for the query i; $\mathcal{O}_P(T, 1)$ returns y_i, where y_i is chosen uniformly at random for the query i. In the definition, $\mathcal{A}_1$ is allowed to make p queries to an oracle $\mathcal{O}_P(\cdot, b)$. We denote the set of $\mathcal{A}_1$'s queries by $Q = \{T_i\}_{i=1}^{p}$ and define the predicate $\mathbf{DIST}(\{T_i\}_{i=1}^{p})$ as we did above.*

$$
\begin{aligned}
&\mathrm{Succ}^{\mathrm{W-SM-UD}}_{\mathbf{Th},p}(\mathcal{A}) = \Pr[P \xleftarrow{\$} \mathcal{P}; b \xleftarrow{\$} \{0,1\}; S \leftarrow \mathcal{A}_1^{\mathcal{O}_P(\cdot,b)}(\ );\\
&b' \leftarrow \mathcal{A}_2(Q, S, P.\mathbf{Th}) : b' = b\\
&\wedge \mathbf{DIST}(\{T_i\}_{i=1}^{p})]
\end{aligned}
$$

We define the insecurity of a tweakable hash function against p target, time ξ, W-SM-UD *adversaries as the maximum success probability of any possibly quantum adversary $\mathcal{A}$ with p targets and running time $\leq \xi$:*

$$
\mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{W-SM-UD}}_{\mathbf{Th},p}(\mathcal{A})\}
$$

Here we have finished describing the properties that will be needed to construct a reduction proof for WOTS-TW. But for the further analysis of those properties and analysis of bigger schemes such as SPHINCS$^+$ one would need several more properties, which will be listed below.

First we start with the notion of Decisional Second Preimage Resistance (DSPR) and it variants that was introduced in [BH19a]. This notion helps to reduce the property of preimage resistance to properties of second preimage resistance and DSPR. It is used in SPHINCS$^+$ to analyze the security of FORS scheme (this scheme will be described later). This property will be also utilized to analyze quantum generic security of W-SM-PRE. The initial description of the property and its justification can be found in [BH19a], here we will use the definitions from [BHK$^+$19] that fit the notion of a tweakable hash function.

We will go straight to a multi-target version of DSPR which is denoted as SM-DSPR. To do so first we need to introduce a predicate SPexists.

Definition 5 ($\mathsf{SPexists_{P,T}}$)**.** *A second preimage exists* $\mathsf{SPexists}$ *predicate of tweakable hash function* $\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^m \to \{0,1\}^n$ *with a fixed* $P \in \mathcal{P}$*,* $T \in \mathcal{T}$ *is the function* $\mathsf{SP}_{P,T} : \{0,1\}^m \to \{0,1\}$ *defined as follows:*

$$\mathsf{SP}_{P,T}(x) \overset{def}{=} \begin{cases} 1 \ \textit{if}\ |\mathbf{Th}^{-1}_{P,T}(\mathbf{Th}_{P,T}(x))| \geq 2 \\ 0 \ \textit{otherwise,} \end{cases}$$

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

In other words $\mathsf{SP}_{P,T}(x) = 0$ *is there is no second preimage for* x *under function* $\mathbf{Th}(P, T, \cdot)$*. Theoretically you can find a second preimage of* x *under* $\mathbf{Th}(P, T, \cdot)$ *only of* $\mathsf{SP}_{P,T}(x) = 1$*.*

Now we present the definition of SM-DSPR from [BHK+19].

Definition 6 (SM-DSPR)**.** *Let* $\mathbf{Th}$ *be a tweakable hash function. Let* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be a two stage adversary. The number of targets is denoted with* p*, where the following inequality must hold:* $p \leq |\mathcal{T}|$*.* $\mathcal{A}_1$ *is allowed to make* p *queries to an oracle* $\mathbf{Th}(P, \cdot, \cdot)$*. We denote the query set* Q *and predicate* $\mathsf{DIST}(\{T_i\}_1^p)$ *as in previous definitions.*

$$\mathrm{Adv}^{\mathrm{SM-DSPR}}_{\mathbf{Th},p}(\mathcal{A}) = \max\{0, \mathsf{succ} - \mathsf{triv}\},$$

where

$$\begin{aligned} \mathrm{Succ} = \Pr[&P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,\cdot)}(); (j,b) \leftarrow \mathcal{A}_2(Q,S,P) : \\ &\mathsf{SP}_{P,T_j}(M_j) = b \wedge \mathsf{DIST}(\{T_i\}_1^p)]. \\ \mathsf{triv} = \Pr[&P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,\cdot)}(); (j,b) \leftarrow \mathcal{A}_2(Q,S,P) : \\ &\mathsf{SP}_{P,T_j}(M_j) = 1 \wedge \mathsf{DIST}(\{T_i\}_1^p)]. \end{aligned}$$

We define the SM-DSPR *insecurity of a tweakable hash function against* p *target, time* ξ *adversaries as the maximum advantage of any (possibly quantum) adversary* $\mathcal{A}$ *with* p *targets and running time* $\leq \xi$*:*

$$\mathrm{InSec}^{\mathrm{SM-DSPR}}(\mathbf{Th}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Adv}^{\mathrm{SM-DSPR}}_{\mathbf{Th},p}(\mathcal{A})\}$$

W-SM-DSPR is a variant in which the description of the underlying hash function is given only after all the challenge queries are done.

Security for a collection of tweakable hash functions. In more complex constructions like SPHINCS$^+$, we make use a collection of tweakable hash functions which we call $\mathbf{Th}_\lambda$. In this case $\mathbf{Th}$ consists of a set of tweakable hash functions $\mathbf{Th}_m$ that differ in terms of m, the length of messages they process. This notion of a collection of tweakable hash functions is necessary as we use the same public parameters for all functions in the collection. Especially, it is necessary to make the security notions above usable in the context of SPHINCS$^+$. The problem is that when used in constructions like SPHINCS$^+$ or XMSS queries to the challenge oracle queries may depend on the outputs of other functions in the collection, or even the same function but with different tweaks. This is incompatible with above definitions as the public parameters are only given to the adversary after all challenge queries are made.

We solve this issue extending all the above *standalone* security properties to the case of collections. The definitions for functions that are part of a collection only differ from the above in a single spot. We give the first adversary $\mathcal{A}_1$, that makes the challenge queries, access to another oracle $\mathbf{Th}_\lambda(P, \cdot, \cdot)$, initialized with P. The oracle takes an input M and a tweak T and, depending on the length $m = |M|$ of M returns $\mathbf{Th}_m(P, M, T)$. The only limitation is that $\mathcal{A}$ is not allowed to use a tweak in queries to both oracles, the challenge oracle and the collection oracle. In general, $\mathcal{A}$ is allowed to query the challenge oracle as well as $\mathbf{Th}_\lambda$ with a message of length x, as long as the used tweak is never used in a query to the challenge oracle.

Definition 7 (W-SM-TCR**,** W-SM-PRE**,** W-SM-UD**,** SM-DSPR **for members of a collection).** *Let* $\mathbf{Th}$ *be a tweakable hash function as defined above with message length* x*. Moreover, let* $\mathbf{Th}$ *be an element of a collection of tweakable hash functions* $\mathbf{Th}_\lambda$ *as described above. Consider an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against the* W-SM-TCR *(,* W-SM-PRE*,* W-SM-UD*,* SM-DSPR*) security of* $\mathbf{Th}$ *in presence of collection* $\mathbf{Th}_\lambda$*. Let* $\mathbf{Th}_\lambda(P, \cdot, \cdot)$ *denote an oracle for* $\mathbf{Th}_\lambda$ *as described above and denote by* $\{T_i^\lambda\}_1^{p_\lambda}$ *the tweaks used in the queries made by* $\mathcal{A}$*. We define the success probability of* $\mathcal{A}$ *against* W-SM-TCR *(,* W-SM-PRE*,* W-SM-UD*,* SM-DSPR*) security of* $\mathbf{Th}$ *in presence of collection* $\mathbf{Th}_\lambda$ *as the success probability of* $\mathcal{A}$ *against standalone* W-SM-TCR *(,* W-SM-PRE*,* W-SM-UD*,* SM-DSPR*) security of* $\mathbf{Th}$ *defined above, when* $\mathcal{A}_1$ *is additionally given oracle access to* $\mathbf{Th}_\lambda(P, \cdot, \cdot)$ *with the condition that* $\{T_i\}_1^p \cap \{T_i^\lambda\}_1^{p_\lambda} = \emptyset$*.*

In the case of W-SM-TCR, we will abuse notation when it comes to the security of SPHINCS$^+$ and consider the joined security of several members of a collection of tweakable hash functions.

2.2 Message digest computation

In SPHINCS$^+$ a special function to compute message digest will be introduced. An expected property of that function is interleaved target subset resilience. Let give a formal definition of this property.

Definition 8 (ITSR **[BHK$^+$19]).** *Let* $\mathrm{H} : \{0,1\}^{\mathcal{K}} \times \{0,1\}^\alpha \to \{0,1\}^m$ *be a keyed hash function. Also consider a mapping function* $\mathsf{MAP}_{h,k,t} : \{0,1\}^m \to \{0,1\}^h \times [0, t-1]^k$ *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* $G = \mathsf{MAP}_{h,k,t} \circ \mathrm{H}$*. Let* $O(\cdot)$ *be an oracle which on input of an* α*-bit* *message M_i samples a key $K_i \xleftarrow{\$} \{0,1\}^{\mathcal{K}}$ and returns $G(K_i, M_i)$. The adversary $\mathcal{A}$ is allowed to query the oracle with massages of its choice. Denote the amount of queries with q.*

$$\begin{aligned}\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},q}(\mathcal{A}) = \Pr[&(R, M) \leftarrow \mathcal{A}^{O(\cdot)}(1^n) \\ &s.t.\ G(K,M) \subseteq \bigcup_{j=1}^{q} G(K_j, M_j) \wedge (K,M) \notin \{(K_j, M_j)\}_{j=1}^{q}],\end{aligned}$$

where $\{(K_j, M_j)\}_{j=1}^{q}$ represent the responses of the oracle $O(\cdot)$. We define the ITSR *insecurity of a keyed hash function against q-query, time-ξ adversaries as the maximum advantage of any quantum adversary $\mathcal{A}$ with running time $\leq \xi$, that makes no more than q queries:*

$$\mathrm{InSec}^{\mathrm{ITSR}}(\mathrm{H}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},q}(\mathcal{A})\}$$

2.3 Pseudorandom functions

To construct a scheme we will need pseudorandom functions. In this subsection we give a definition of pseudorandom functions, provide security notions.

Function definition. A *pseudorandom* function takes a secret parameter S and context information in form of a *tweak* T. The secret parameter might be thought of as a function key or index. The tweak might be interpreted as a nonce.

Definition 9 (Pseudorandom function). *Let $n \in \mathbb{N}$, $\mathcal{S}$ the secret parameters space and $\mathcal{T}$ the tweak space. A pseudorandom function is an efficient function*

$$F : \mathcal{S} \times \mathcal{T} \rightarrow \{0,1\}^n$$

generating an n-bit value out of secret parameter and a tweak.

Security notion In the following we give the definition for PRF security of a function $F : \mathcal{S} \times \mathcal{T} \rightarrow \{0,1\}^n$. In the definition of the PRF distinguishing advantage the adversary $\mathcal{A}$ gets (classical) oracle access to either $F(S, \cdot)$ for a uniformly random secret parameter $S \in \mathcal{S}$ or to a function G drawn from the uniform distribution over the set $\mathcal{G}(\mathcal{T}, n)$ of all functions with domain $\mathcal{T}$ and range $\{0,1\}^n$. The goal of $\mathcal{A}$ is to distinguish both cases.

Definition 10 (PRF**).** *Let F be defined as above. We define the* PRF *distinguishing advantage of an adversary $\mathcal{A}$ as*

$$\mathrm{Adv}^{\mathrm{PRF}}_{F}(\mathcal{A}) = |\Pr_{S \xleftarrow{\$} \mathcal{S}}[\mathcal{A}^{F(S,\cdot)} = 1] - \Pr_{G \xleftarrow{\$} \mathcal{G}(\mathcal{T},n)}[\mathcal{A}^{G(\cdot)} = 1]|.$$

We define the PRF *insecurity of a pseudorandom function F against q-query, time-ξ adversaries as the maximum advantage of any possibly quantum adversary that runs in time ξ and makes no more then q queries to its oracle:*

$$\mathrm{InSec}^{\mathrm{PRF}}(F; \xi, q) = \max_{\mathcal{A}}\{\mathrm{Adv}^{\mathrm{PRF}}_{F}(\mathcal{A})\}.$$

2.4 Security model

In this part we describe the security model in which we will prove the security of the one-time digital signature scheme.

Definition 11 (Digital signature schemes). *Let $\mathcal{M}$ be a message space. A digital signature scheme* $\mathsf{Dss} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ *is a triple of probabilistic polynomial time algorithms:*

- $\mathsf{Kg}(1^n)$ *on input of a security parameter* 1^n *outputs a private key* sk *and a public key* pk*;*
- $\mathsf{Sign}(\mathsf{sk}, M)$ *outputs a signature* σ *under secret key* sk *for message* $M \in \mathcal{M}$*;*
- $\mathsf{Vf}(\mathsf{pk}, \sigma, M)$ *outputs 1 iff* σ *is a valid signature on* M *under* pk*;*

such that $\forall(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^n), \forall(M \in \mathcal{M}) : \mathsf{Vf}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, M), M) = 1.$

Consider a signature scheme $\mathsf{Dss}(1^n)$, where n is the security parameter. We introduce a definition for the security of $\mathsf{Dss}(1^n)$, which we call the existential unforgeability under non-adaptive chosen message attack (EU-naCMA). It is defined using the following experiment.

```
Experiment $\mathsf{Exp}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A})$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kg}(1^n)$.
    $\{M_1, \ldots, M_q\} \leftarrow \mathcal{A}()$.
    Compute $\{(M_i, \sigma_i)\}_{i=1}^q$ using $\mathsf{Sign}(\mathsf{sk}, \cdot)$.
    $(M^\star, \sigma^\star) \leftarrow \mathcal{A}(\{(M_i, \sigma_i)\}_{i=1}^q, pk)$
    Return 1 iff $\mathrm{Vf}(\mathsf{pk}, \sigma^\star, M^\star) = 1$ and $M^\star \notin \{M_i\}_{i=1}^q$.
```

The adversary is forced to output a list of messages $M_1, ..., M_q$ it wants to see signed before obtaining the public key pk. In our work we consider one-time signatures, so the number of allowed messages q is set to 1.

Let $\mathrm{Succ}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A}) = 1\right]$ be the success probability of an adversary $\mathcal{A}$ in the above experiment.

Definition 12 (EU-naCMA). *Let* $t, n \in \mathbb{N}$, $t = \mathrm{poly}(n)$, $\mathrm{Dss}(1^n)$ *is a digital signature scheme. We call* Dss $\mathrm{EU-naCMA}$*-secure if the maximum success probability* $\mathrm{InSec}^{\mathrm{EU-naCMA}}(\mathrm{Dss}(1^n), t)$ *of all possibly probabilistic adversaries* $\mathcal{A}$ *running in time* $\leq t$ *is negligible in* n*:*

$$\mathrm{InSec}^{\mathrm{EU-naCMA}}(\mathrm{Dss}(1^n); t) \overset{\mathrm{def}}{=} \max_{\mathcal{A}} \left\{\mathrm{Succ}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A})\right\} = \mathrm{negl}(n).$$

2.5 Estimated security

In this section we collect bounds on the complexity of generic attacks against the described properties. A (tweakable) hash function **Th** is commonly considered a good function if there are no attacks known for any security property that perform better against **Th** than against a random function. Table 1 summarizes the current situation.

Table 1: Success probability of generic attacks – In the "Success probability" column we give the bound for a quantum adversary $\mathcal{A}$ that makes q quantum queries to the function and p classical queries to the challenge oracle. The security parameter n is the output length of $\mathbf{Th}$. We use $X = \sum_{\gamma} \left(1 - \left(1 - \frac{1}{t}\right)^{\gamma}\right)^{k} \binom{p}{\gamma} \left(1 - \frac{1}{2^h}\right)^{p-\gamma} \frac{1}{2^{h\gamma}}$.

Property	Success probability	Status
W-SM-TCR	$\Theta((q+1)^2/2^n)$	proven ([BHK+19, HRS16])
SM-DSPR	$\Theta((q+1)^2/2^n)$	conjecture ([BHK+19])
W-SM-PRE	$\Theta((q+1)^2/2^n)$	conjecture ([BH19a, BHK+19])
PRF	$\Theta((q+1)^2/2^n)$	conjecture ([HRS16])
W-SM-UD	$\mathcal{O}((q+1)^2/2^n)$	conjecture (this work / [DSS05])
ITSR	$\Theta((q+1)^2 \cdot X)$	conjecture ([BHK+19])

The success probability of generic attacks against W-SM-TCR was analyzed in [BHK+19]. Assuming that $\mathbf{Th}$ behaves like a random function a reduction to an average-case search problem was given. A generic attack using Grover search against plain TCR is given in [HRS16]. That attack is also applicable against W-SM-TCR – as it simply runs a second preimage search when all information is available – and has a success probability matching the proven bound.

A conjecture for the success probability against SM-DSPR was also given in [BHK+19]. There is a proof for single target DSPR property in [BH19a] that gives the following bound: $O((q+1)^2/2^n)$. A non-tight proof for a multi-target version is also analyzed in [BH19a]. The best attack against DSPR for now is a second-preimage search that gives the same bound for the multi-target case.

For PRE and also multi-function, multi-target PRE of a hash function h, a bound of $\mathrm{Succ}^{\mathrm{PRE}}_{h,p}(\mathcal{A}) = \Theta((q+1)^2/2^n)$ is given in [HRS16]. The bound is proven for h that are compressing by at least a factor 2 in the message length and it is conjectured that it also applies for length preserving hash functions, i.e., functions that map n-bit messages to n-bit outputs, ignoring additional inputs like function keys or tweaks. This is exactly the case that we are interested in. Another way to support the conjecture is a tight bound using W-SM-TCR and SM-DSPR ($\mathrm{Succ}^{\mathrm{W-SM-PRE}}_{\mathbf{Th},p}(\mathcal{A}) \leq 3 \cdot \mathrm{Succ}^{\mathrm{W-SM-TCR}}_{\mathbf{Th},p}(\mathcal{A}) + \mathrm{Adv}^{\mathrm{SM-DSPR}}_{\mathbf{Th},p}(\mathcal{A})$) given in [BHK+19, BH19a]. With this we can derive the same bound of $\mathrm{Succ}^{\mathrm{W-SM-PRE}}_{\mathbf{Th},p}(\mathcal{A}) = \Theta((q+1)^2/2^n)$. Also in this case it is a conjecture as the bound for SM-DSPR is only conjectured for now.

Exhaustive search on an unstructured space is traditionally considered to be the best known attack for the PRF property. Considering this as a preimage search for a random function and using the results from [HRS16] we obtain the stated bound. We note it as conjectured as we are not aware of a formal proof for this result.

The notion of undetectability was introduced in [DSS05]. In that work, the authors give a bound for one target undetectability considering classical adversaries as $\mathcal{O}(q/2^n)$. The notion was not used in more recent works and we are not aware of a proven bound against quantum adversaries. We conjecture a bound of $\mathcal{O}(q^2/2^n)$ for quantum adversaries. The reasoning behind this is that as long as the adversary does not know if a target (likely) has a preimage under **Th**, it cannot do better than guessing. This suggests the same bound as for W-SM-PRE.

For all notions we conjecture that the bounds are exactly the same for the case of collections. The reason is that for a random tweakable function, every tweak is related to an independent random function. Hence, giving access to those does not give any information about the targets to the adversary. This is also reflected in the reductions that we know so far. In these, the function for a tweak that is not used for a challenge is simulate by an independent random function and we can give access to this function in parallel to the challenge oracle as we do not touch it in the reduction.

3 WOTS-TW

SPHINCS$^+$ [BHK$^+$19] developed its own variant of the Winternitz OTS. However, the authors never explicitly defined that variant. As the flaw in the SPHINCS$^+$ security proof was in the proof for their WOTS scheme, we give a separate description of the scheme in this section. As the distinguishing feature of this variant is the use of tweakable hash functions, we call it WOTS-TW.

3.1 Parameters

WOTS-TW uses several parameters. The main security parameter is $n \in \mathbb{N}$. m is the message length, which we sign. In case of SPHINCS$^+$ $m = n$. The tweak space $\mathcal{T}$ must be at least of size lw. The size of tweak space should be bigger if we use several instances of WOTS-TW in a bigger construction such as SPHINCS$^+$ so we can use a different tweak for each hash function call. Here $w \in \mathbb{N}$ is a so-called Winternitz parameter, which determines a base of the representation that is used in the scheme, and l is defined through the following constants:

$$l_1 = \left\lceil \frac{m}{\log(w)} \right\rceil, \quad l_2 = \left\lfloor \frac{\log(l_1(w-1))}{\log(w)} \right\rfloor + 1, \quad l = l_1 + l_2.$$

We also need a pseudorandom function $\mathbf{PRF} : \{0,1\}^n \times \mathcal{T} \rightarrow \{0,1\}^n$, and a tweakable hash function $\mathbf{Th} : \{0,1\}^n \times \mathcal{T} \times \{0,1\}^n \rightarrow \{0,1\}^n$.

3.2 Addressing scheme

For the tweakable hash functions to guarantee security, they have to be called with different tweaks. This is achieved using what was called an addressing scheme in SPHINCS$^+$. Such addressing scheme assigns a unique address to every tweakable hash function call in the scheme and the address space is part of the tweak space such that addresses can be used as tweaks. We do not specify a concrete addressing scheme in this work (see the SPHINCS$^+$ specification [ABB$^+$20] for an example). Abstractly, we achieve unique addresses the following way. A Winternitz key pair defines a structure of l hash chains, each of which makes $w-1$ calls to the tweakable hash function. For a unique addressing scheme, one may use any injective function that takes as input $i \in [0, l-1]$, $j \in [0, w-2]$, and possibly a prefix, and maps into the address space. The prefix is necessary to ensure uniqueness if many instances of WOTS-TW are used in a single construction. We will use **ADRS** to denote that prefix. The tweak associated with the j-th function call in the i-th chain is then defined as the output of this function on input i, j (and a possible prefix) and denoted as $T_{i,j}$.

3.3 WOTS-TW scheme

The main difference between WOTS variants is in the way they do hashing. Previously, the distinction was made in the definition of the so called chaining function that describes how the hash chains are computed. For WOTS-TW this distinction is further shifted into the construction of the tweakable hash function **Th**. The chaining function then looks as follows:

Chaining function $\boldsymbol{c^{j,k}(x, i, \mathrm{Seed})}$: The chaining function takes as inputs a message $x \in \{0,1\}^n$, iteration counter $k \in \mathbb{N}$, start index $j \in \mathbb{N}$, chain index i, and public parameters Seed (the name comes from a specific construction of a tweakable hash function that uses the public parameters as seed for a PRG). The chaining function then works the following way. In case $k \leq 0$, c returns x, i.e., $c^{j,0}(x, i, \mathrm{Seed}) = x$. For $k > 0$ we define c recursively as

$$c^{j,k}(x, i, \mathrm{Seed}) = \mathbf{Th}(\mathrm{Seed}, T_{i,j+k-1}, c^{j,k-1}(x, i, \mathrm{Seed}))\,.$$

If we consider several instances of WOTS-TW than we will use $c^{j,k}_{\mathbf{ADRS}}(x, i, \mathrm{Seed})$ to denote that tweaks that are used to construct the chain have **ADRS** as a prefix. With this chaining function, we can describe the algorithms of WOTS-TW.

Key Generation Algorithm $(\mathsf{SK}, \mathsf{PK}) \leftarrow$ WOTS.kg$(\mathcal{C}; \mathcal{S})$: The key generation algorithm optionally takes as input context information $\mathcal{C} = (\mathrm{Seed}, \mathbf{ADRS})$, consisting of a public seed $\mathrm{Seed} \in \{0,1\}^n$ and a global address **ADRS**, as well as randomness $\mathcal{S} \in \{0,1\}^n$ which we call the secret seed. These inputs are meant for the use in more complex protocols. If they are not provided, key generation randomly samples the seeds and sets **ADRS** to 0. The key generation algorithm then computes the internal secret key $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)$ as $sk_i \leftarrow \mathbf{PRF}(\mathcal{S}, T_{i,0})$), i.e., the $l \cdot n$ bit secret key elements are derived form the secret key seed using addresses. The element of the public key pk is computed as

$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_l) = (c^{0,w-1}(\mathsf{sk}_1, 1, \mathrm{Seed}), \ldots, c^{0,w-1}(\mathsf{sk}_l, l, \mathrm{Seed}))$$

The key generation algorithm returns $\mathsf{SK} = (\mathcal{S}, \mathcal{C})$ and $\mathsf{PK} = (\mathsf{pk}, \mathcal{C})$ Note that we can compute sk and pk from SK.

Signature Algorithm $\boldsymbol{\sigma} \leftarrow$ WOTS.sign$(\boldsymbol{M}, \mathsf{SK})$: On input of an m-bit message M, and the secret key $\mathsf{SK} = (\mathcal{S}, \mathcal{C})$, the signature algorithm first computes a base w representation of $M : M = (M_1, \ldots, M_{l_1})$, $M_i \in \{0, \ldots, w-1\}$. That is, M is treated as the binary representation of a natural number x and then the w-ary representation of x is computed. Next it computes the checksum $C = \sum_{i=1}^{l}(w - 1 - M_i)$ and its base w representation $C = (C_1, \ldots, C_{l_2})$. We set $B = (b_1, \ldots, b_l) = M||C$, the concatenation of the base w representations of M and C. Then the internal secret key is regenerated using $\mathsf{sk}_i \leftarrow \mathbf{PRF}(\mathcal{S}, T_{i,0})$ the same way as during key generation. The signature is computed as

$$\sigma = (\sigma_1, \ldots, \sigma_l) = (c^{0,b_1}(\mathsf{sk}_1, 1, \mathrm{Seed}), \ldots, c^{0,b_l}(\mathsf{sk}_l, l, \mathrm{Seed}))$$

Verification Algorithm $(\{0,1\} \leftarrow$ WOTS.vf$(\boldsymbol{M}, \boldsymbol{\sigma}, \mathsf{PK}))$: On input of m-bit message M, a signature σ, and public key $\mathsf{PK} = (\mathsf{pk}, \mathcal{C})$, the verification algorithm first computes the $b_i, 1 \leq i \leq l$ as described above. Then it checks if

$$\mathsf{pk} \stackrel{?}{=} \mathsf{pk}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_l) = (c^{b_1, w-1-b_1}(\sigma_1, 1, \mathrm{Seed}), \ldots, c^{b_l, w-1-b_l}(\sigma_l, l, \mathrm{Seed}))\,.$$

On the equality the algorithm outputs true and false otherwise.

4 Security of WOTS-TW

Now we will reduce the security of WOTS-TW to the security properties of the tweakable hash function **Th** and the pseudorandom function family **PRF**. To do so we will give a standard game hopping proof. Intuitively the proof goes through the following steps.

– First, we replace the inner secret key elements that are usually generated using **PRF** by uniformly random values. The two cases must be computationally indistinguishable if **PRF** is indeed pseudorandom.
– Next we replace the blocks in the chains that become part of the signature by the hash of random values. We need this so that we can later place preimage challenges at these positions of the chain. Here it is important to note that preimage challenges are exactly such hashes of random domain elements and not random co-domain elements. To argue that these two cases are indistinguishable, we need a hybrid argument since for most chains we replace the outcome of several iterations of hashing with a random value.
– Next we show that breaking the EU-naCMA property of our scheme in this final case will either allow us to extract a target-collision or a preimage for a given challenge.

Theorem 1. *Let n, $w \in \mathbb{N}$ and $w = poly(n)$. Let* $\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *be a* W-SM-TCR, W-SM-PRE, *and* W-SM-UD *function. Let* $\mathbf{PRF} : \mathcal{S} \times \mathcal{T} \rightarrow \{0,1\}^n$ *be a pseudorandom function. Then the insecurity of the WOTS-TW scheme against EU-nCMA attack is bounded by*

$$\begin{aligned}\mathrm{InSec}^{EU\text{-}nCMA}(WOTS-TW(1^n, w); t, 1) < \\ \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l) + \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw) + \\ \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l) + w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)\end{aligned}$$

with $\widetilde{t} = t + lw$, where time is given in number of $\mathbf{Th}$ *evaluations.*

Proof. First consider the following two games: GAME.1 is the original EU-nCMA game and GAME.2 is the same as GAME.1 but instead of using pseudorandom elements from $\mathbf{PRF}$ a truly random function $\mathbf{RF} : \mathcal{S} \times \mathcal{T} \rightarrow \{0,1\}^n$ is used. We claim that the difference in the success probability of $\mathcal{A}$ playing these games must be bound by $\mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l)$.

Next we consider GAME.3 which is the same as GAME.2 but to answer the message signing request we build the signature from nodes that are computed applying $\mathbf{Th}$ only once instead of b_i times (except if $b_i = 0$, then we return a random value as in the previous game). The public key is constructed from that signature by finishing the chain according to the usual algorithm. We will detail the process in the proof below. We claim that the difference in the success probability of $\mathcal{A}$ playing these games must be bound by $w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)$.

Afterwards, we consider GAME.4, which differs from GAME.3 in that we are considering the game lost if an adversary outputs a valid forgery (M', σ') where there exists an i such that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$. We claim that the difference in the success probability of $\mathcal{A}$ playing these games must be bound by $\mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw)$.

If we now consider how $\mathcal{A}$ can win in GAME.4 there is just we viable case left. Namely, this is the case where the values that get computed from the forgery during verification fully agree with those values that are computed during the verification of the signature. As the checksum ensures that there is at least we index for the forgery that is smaller than the respective index of the signature, this means that we can use an $\mathcal{A}$ that wins in GAME.4 to find preimages. We claim that the success probability of the adversary $\mathcal{A}$ in GAME.4 must be bounded by $\mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l)$.

So we get the following claims:

Claim 1. $|\mathrm{Succ}^{\mathrm{GAME.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l)$

Claim 2. $|\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)$

Claim 3. $|\mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.4}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw)$

Claim 4. $\mathrm{Succ}^{\mathrm{GAME.4}}(\mathcal{A}) \leq \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l)$

The remainder of the proof consists of proving these claims. We then combine the bounds from the claims to obtain the bound of the theorem.

Proof of Claim 1

Claim 1. $|\mathrm{Succ}^{\mathrm{GAME.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l)$

Proof. Our argument proceeds in two steps. First, we replace **PRF** in GAME.1 by the oracle provided by the PRF game and output 1 whenever $\mathcal{A}$ succeeds. If the oracle is the real **PRF** function keyed with a random secret key, the view of $\mathcal{A}$ is identical to that in GAME.1. If the oracle is the truly random function the argument is a bit more involved. In this case, it is important to note that $\mathcal{A}$ never gets direct access to the oracle but only receives outputs of the oracle. The inputs on which the oracle is queried to obtain these outputs are all unique. Hence, the outputs are uniformly random values. Therefore, the view of $\mathcal{A}$ in this case is exactly that of GAME.2. Consequently, the difference of the probabilities that the reduction outputs we in either of the two cases (which is the PRF distinguishing advantage) is exactly the difference of the success probabilities of $\mathcal{A}$ in the two games.

Proof of Claim 2 We first give a more detailed description of GAME.3. In the EU-nCMA game the adversary $\mathcal{A}$ asks to sign a message M without knowing the public key. This message M gets encoded as $B = b_1, \ldots, b_l$. In GAME.3 to answer the query we will perform the following operations. First we generate l values uniformly at random: $u_i \xleftarrow{\$} \{0,1\}^n,\ i \in \{1, \ldots, l\}$. Next we answer the signing query with a signature $\sigma = (\sigma_1, \ldots, \sigma_l)$, where $\sigma_i = \mathbf{Th}(\mathrm{Seed}, T_{i,b_i-1}, u_i)$ if $b_i > 0$ and $\sigma_i = u_i$ if $b_i = 0$. Then the public key is constructed as

$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_l) = (c^{b_1, w-1-b_1}(\sigma_1, 1, \mathrm{Seed}), \ldots, c^{b_l, w-1-b_l}(\sigma_l, l, \mathrm{Seed}))\,, \quad (1)$$

and public key and signature are returned to the adversary. The reason we consider this game is that to bound the final success probability in GAME.4 we will have a reduction replace the u_i with W-SM-PRE challenges. The resulting signatures have exactly the same distribution as the ones we get here. To show that this cannot change the adversary's success probability in a significant way, we now prove the following claim.

Claim 2. $|\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)$

Proof. Consider the following scenario. Let the adversary's query be M. During the signing algorithm M is encoded as $B = \{b_1, \ldots, b_l\}$. Consider two distributions $D_0 = \{\xi_1, \ldots, \xi_l\}$, where $\xi_i \xleftarrow{\$} \{0,1\}^n,\ i \in [1, l]$ and $D_{Kg} = \{y_1, \ldots, y_l\}$, where $y_i = c^{0, b_i-1}(\xi_i, i, \mathrm{Seed}),\ \xi_i \xleftarrow{\$} \{0,1\}^n,\ i \in [1, l]$. Samples from the first distribution are just random values, and the samples from D_{Kg} are distributed the same way as the $(b_i - 1)$-th values of valid WOTS-TW chains. Assume we play a game where we get access to an oracle $\mathcal{O}_\phi$ that on input B returns $\phi = \{\phi_1, \ldots, \phi_l\}$, either initialized with a sample from D_0 or with a sample from

Algorithm 1: $\mathcal{M}^{\mathcal{A}}_{2-3}$

```
   Input  : Access to a distribution oracle $\mathcal{O}_\phi$ and forger $\mathcal{A}$
   Output: 0 or 1.
 1 Start $\mathcal{A}$ to obtain query with a message $M$.
 2 Encode $M$ as $B = b_1, \ldots, b_l$ as in signature algorithm.
 3 Call $\mathcal{O}_\phi(B)$ to obtain sample $\phi$
 4 Construct the signature $\sigma$ doing we chain step on each sample and compute
     the public key from the signature:
 5 for $1 \leq i \leq l$ do
 6     $\sigma_i = c^{b_i-1,1}(\phi_i, i, \text{Seed})$
 7     $\mathsf{pk}_i = c^{b_i, w-1-b_i}(\sigma_i, i, \text{Seed})$
 8 Send $\mathsf{PK} = (\mathsf{pk}, \text{Seed})$ and $\sigma$ to $\mathcal{A}$.
 9 if $\mathcal{A}$ returns a valid forgery $(M', \sigma')$ then
10     return 1
11 else
12     return 0
```

D_{Kg}. Each case occurs with probability 1/2. Then we can construct an algorithm $\mathcal{M}^{\mathcal{A}}_{2-3}$ as in Algorithm 1 that can distinguish these two cases using a forger $\mathcal{A}$.

Let us consider the behavior of $\mathcal{M}^{\mathcal{A}}_{2-3}$ when $\mathcal{O}_\phi$ samples from $\mathcal{D}_{Kg}$. In this case all the elements in the chains are distributed the same as in GAME.2. The probability that $\mathcal{M}^{\mathcal{A}}_{2-3}$ outputs 1 is the same as the success probability of the adversary in GAME.2.

If ϕ instead is from $\mathcal{D}_{\mathcal{U}}$, then the distribution of the elements in the chains is the same as in GAME.3. Hence the probability that $\mathcal{M}^{\mathcal{A}}_{\mathcal{UD}}$ outputs 1 is the same as the success probability of the adversary in GAME.3.

By definition, the advantage of $\mathcal{M}^{\mathcal{A}}_{2-3}$ in distinguishing $\mathcal{D}_{\mathcal{U}}$ from $\mathcal{D}_{Kg}$ is hence given by

$$\text{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}^{\mathcal{A}}_{2-3}) = |\text{Succ}^{\text{GAME.2}}(\mathcal{A}) - \text{Succ}^{\text{GAME.3}}(\mathcal{A})| \tag{2}$$

The remaining step is to derive an upper bound for $\text{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}^{\mathcal{A}}_{\text{UD}})$ using the insecurity of the W-SM-UD property. For this purpose we use a hybrid argument.

Let $b_{\max} = \max\{b_1, \ldots, b_l\}$ be the maximum of the values in the message encoding of M. Let H_k be the distribution obtained by computing the values in ϕ as $\phi_i = c^{k, b_i-1-k}(\xi_i, i, Seed),\ \xi_i \xleftarrow{\$} \{0,1\}^n$. Then $H_0 = D_{Kg}$ and $H_{b_{\max}-1} = D_0$ (Note that the chaining function returns the identity when asked to do a negative amount of steps). As $\mathcal{M}^{\mathcal{A}}_{2-3}$ distinguishes the extreme cases, by a hybrid argument there are two consecutive hybrids H_j and H_{j+1} that can be distinguished with probability $\geq \text{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}^{\mathcal{A}}_{2-3})/(b_{\max} - 1)$.

To bound the success probability of an adversary in distinguishing two such consecutive hybrids, we build a second reduction $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ that uses $\mathcal{B} = \mathcal{M}^{\mathcal{A}}_{2-3}$ to break W-SM-UD. For this purpose, $\mathcal{M}^{\mathcal{B}}_{\text{UD}}$ simulates $\mathcal{O}_\phi$. To answer a query for

$B = b_1, \dots, b_l$, $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ plays in the W-SM-UD game, interacting with the W-SM-UD oracle $\mathcal{O}_{\mathrm{UD}}(\cdot, b)$ to construct hybrid H_{j+b}, depending on the secret bit b of the oracle. To do so $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ makes queries to $\mathcal{O}_{\mathrm{UD}}$ with tweaks $\{T_{1,j}, \dots, T_{l,j}\}$. Then, depending on b, the responses ψ of $\mathcal{O}_{\mathrm{UD}}$ are either l random values or $\psi = (c^{j,1}(\xi_1, 1, Seed), \dots, c^{j,1}(\xi_l, l, Seed),\ \xi_i \xleftarrow{\$} \{0,1\}^n,\ i \in [1, l])$. After that $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ requests Seed from the W-SM-UD challenger. Next, $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ applies the hash chain to the oracle responses ψ to compute samples

$$\phi_i = \begin{cases} c^{j+1, b_i - 1 - (j+1)}(\psi_i, i, Seed), & \text{if } j < b_i - 1 \\ \xi_i \xleftarrow{\$} \{0,1\}^n, & \text{otherwise}, \end{cases}$$

and returns it to $\mathcal{M}_{2-3}^{\mathcal{A}}$. $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ returns whatever $\mathcal{M}_{2-3}^{\mathcal{A}}$ returns.

If ψ consisted of random values the distribution was H_{j+1}, otherwise H_j. Consequently, the advantage of distinguishing any two hybrids must be bound by $\mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, l)$.

Putting things together, we see that $b_{\max} \leq w$ for any message M. Hence we get

$$\begin{aligned} |\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| &= \mathrm{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}_{2-3}^{\mathcal{A}}) \\ &\leq w \cdot \mathrm{Succ}_{\mathbf{Th},l}^{\mathrm{W-SM-UD}}(\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}) \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, l) \end{aligned}$$

which concludes the proof.

Proof of Claim 3 Recall that GAME.4 differs from GAME.3 in that we are considering the game lost if an adversary outputs a valid forgery (M', σ') where there exist such i that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$. So the difference in success probability is exactly the probability that $\mathcal{A}$ outputs a valid forgery and there exists an i such that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$. We will now prove Claim 3 which claims the following bound on this probability:

Claim 3. $|\mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.4}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw)$

Proof. To prove the claim we construct an algorithm $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ that reduces W-SM-TCR of **Th** to the task of forging a signature that fulfills the above condition. The algorithm is based on the following idea. $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ simulates GAME.4. In Game.4 the adversary sends a query to sign a message M. To answer this query and compute the public key, $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ interacts with the W-SM-TCR oracle. This way, $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ obtains target-collision challenges corresponding to the nodes in the signature and all intermediate values in the chain computations made to compute the public key. Then $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ requests the public parameters P from the W-SM-TCR challenger. We set the public seed Seed of WOTS-TW equal to P. We answer on the query with the constructed signature and public key. Per assumption there now exists i such that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$, hence by a pigeon hole argument there must be a collision on the way to public key element. $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ extracts this collision and returns it. Algorithm 2 gives a

Algorithm 2: $\mathcal{M}^{\mathcal{A}}_{\text{TCR}}$

```
Input  : Security parameter $n$, oracle access to W-SM-TCR challenger $C$ and
         EU-nCMA forger $\mathcal{A}$.
Output: A pair $(j, M)$ or fail.
1  begin Challenge placement
2      Start $\mathcal{A}$ to obtain query with a message $M$.
3      Encode $M$ as $B = b_1, \ldots, b_l$ as in signature algorithm.
4      for $i \in \{1, \ldots, l\}$ do
5          if $b_i = 0$ then
6              Set $\sigma_i \overset{\$}{\leftarrow} \{0,1\}^n$.
7          else
8              Sample $\xi_i \overset{\$}{\leftarrow} \{0,1\}^n$,
9              Query $C$ for W-SM-TCR challenge with inputs $\xi_i, T_{1,b_i-1}$.
10             Store answer as $\sigma_i$. // i.e., $\sigma_i = \mathbf{Th}(P, T_{i,b_i-1}, \xi_i)$
11         Compute public key element $\mathsf{pk}_i = c^{b_i, w-1-b_i}(\sigma_i, i, \cdot)$ as in the
               verification algorithm but using the W-SM-TCR challenge oracle
               provided by $C$ in place of $\mathbf{Th}$. // That is why no Seed is
               needed
12     Get public parameters $P$ from the challenger and set Seed $= P$.
13     Set signature $\sigma = (\sigma_1, \ldots, \sigma_l)$ and $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_l)$.
14 begin Obtaining the result
15     Return $\sigma$ and $\mathsf{PK} = (\mathsf{pk}, \text{Seed})$ to the adversary $\mathcal{A}$.
16     if *The adversary returns a valid forgery* $(M', \sigma')$ then
17         Encode $M'$ as $B' = (b'_1, \ldots, b'_l)$ according to sign.
18         if $\exists$ *i such that* $b'_i < b_i$ *and* $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \text{Seed}) \neq \sigma_i$ then
19             Let $j$ be the smallest integer such that the chains collide:
                 $c^{b_i, j}(y_i, i, \text{Seed}) = c^{b'_i, j}(\sigma'_i, i, \text{Seed}))$.
20             return W-SM-TCR *solotion* $(i, c^{b'_i, (j-1)}(\sigma'_i, i, \text{Seed}))$
21         else
22             return *fail*
23     else
24         return *fail*
```

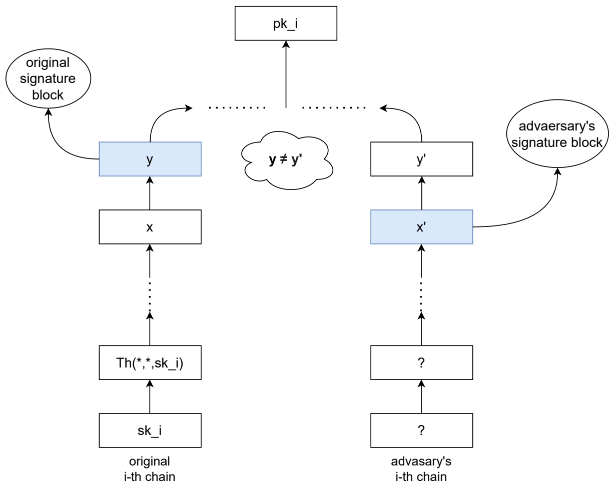

Fig. 1: Example of a case in claim 3

detailed description of $\mathcal{M}^{\mathcal{A}}_{\text{TCR}}$ in pseudocode. The algorithm is broken into two logically separated parts: Challenge placement and obtaining the result.

Here we detail which W-SM-TCR challenges we create per chain in line 11 of Algorithm 2. Assume we have σ_i at position b_i. Then the first query will be (T_{i,b_i}, σ_i). Lets denote the answer for that query as c_1. The next query will be (T_{i,b_i+1}, c_1). We denote the answer for that query as c_2. In general we will make queries of the form (T_{i,b_i+k}, c_k). And the answers for that queries we denote as c_{k+1}. We make queries until we get c_{w-1-b_i}. We set pk_i to be c_{w-1-b_i}.

As we are set to bound the probability of those cases where the adversary outputs a valid forgery and there exists such i that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \text{Seed}) \neq \sigma_i$, $\mathcal{M}^{\mathcal{A}}_{\text{TCR}}$ never runs into the fail cases in lines 22 and 24. Moreover, the distribution of inputs to $\mathcal{A}$ when run by $\mathcal{M}^{\mathcal{A}}_{\text{TCR}}$ is identical to that in GAME.4. Therefore, $\mathcal{M}^{\mathcal{A}}_{\text{TCR}}$ returns a target-collision with probability $|\text{Succ}^{\text{GAME.3}}(\mathcal{A}) - \text{Succ}^{\text{GAME.4}}(\mathcal{A})|$ which concludes the proof.

Proof of Claim 4 It remains to prove the last claim. Consider a forgery σ' and the positions b'_i of the σ' elements. There must exist a j such that $b'_j < b_j$ by the properties of the checksum. Remember that in GAME.4, the case where $c^{(b'_j, b_j - b'_j)}(\sigma'_j, j, \text{Seed}) \neq \sigma_j$ is excluded for all such j. Hence, it must hold for these j that $c^{(b'_j, b_j - b'_j)}(\sigma'_j, j, \text{Seed}) = \sigma_j$. Therefore, we can use $\mathcal{A}$ to compute a preimage of σ_j. We use this to prove Claim 4.

Claim 4. $\text{Succ}^{\text{GAME.4}}(\mathcal{A}) \leq \text{InSec}^{\text{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l)$

Proof. As for the previous claim, we construct an algorithm $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ that uses a forger in GAME.4 to solve a W-SM-PRE challenge. In the beginning, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ receives a query to sign a message M from the adversary $\mathcal{A}$ and encodes it

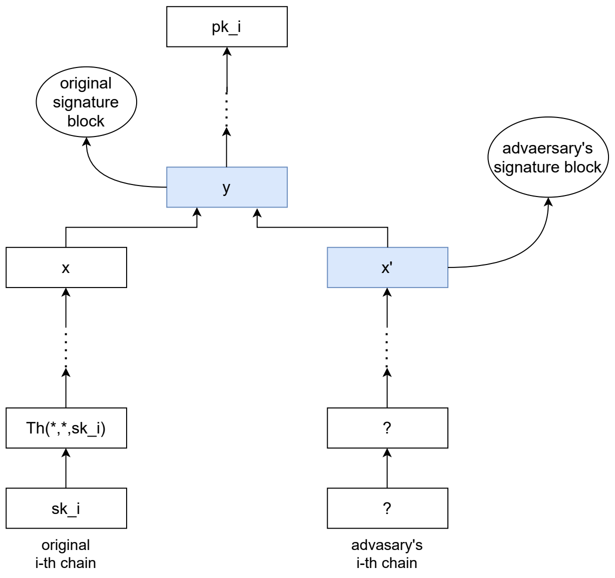

Fig. 2: Example of a case in Claim 4.

into b_i's. To answer the query $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ interacts with the W-SM-PRE challenger to receive preimage challenges y_i for tweaks that make the challenges fit into positions b_i. That way, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ can use the challenges as signature values $\sigma_i = y_i$. Then $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ asks the W-SM-PRE challenger to return public parameters P. Given P, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ can construct the public key using the recomputation method used in the signature verification algorithm. $\mathcal{M}^{\mathcal{A}}$ sets the public seed Seed of WOTS-TW to be P and returns the constructed signature and public key to $\mathcal{A}$. When $\mathcal{A}$ returns a valid forgery, this forgery must contain a signature value σ_j with index j such that $b'_j < b_j$ and $c^{b'_j,(b_j-b'_j)}(\sigma'_j, j, \text{Seed}) = \sigma_j$ per definition of the game. $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ returns preimage $(j, c^{b'_j,(b_j-b'_j-1)}(\sigma'_j, j, \text{Seed}))$. A pseudocode version of $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ is given as Algorithm 3. The algorithm is broken into two logically separated parts: Challenge placement and obtaining the result.

Due to the properties of GAME.4, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ succeeds whenever $\mathcal{A}$ succeeds, as the failure case in line 19 never occurs when $\mathcal{A}$ succeeds. Moreover, the distribution of the inputs to $\mathcal{A}$ when run by $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ is identical to that in GAME.4 (this was ensured in the game hop to GAME.3). Therefore, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ returns preimages with probability $\text{Succ}^{\text{GAME.4}}(\mathcal{A})$ which proves the claim.

5 Extension to multiple instances with same public seed

One-time signatures are often used in more complex constructions. Indeed, WOTS-TW was developed as part of SPHINCS$^+$. The distinguishing feature of this setting is that many WOTS-TW instances are used within one instance of the

Algorithm 3: $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$

```
Input  : Security parameter $n$, access to W-SM-PRE challenger $C$ and forger $\mathcal{A}$.
Output: A pair $(j, M)$ or fail.
1  begin Challenge placement
2      Run $\mathcal{A}$ to receive initial query for a signature on message $M$.
3      Encode $M$ as $B = b_1, \ldots, b_l$ following the steps in the signature algorithm.
4      for $1 \le i \le l$ do
5          if $b_i > 0$ then
6              Query $C$ for preimage challenge $y_i$ with tweak $T_{1,b_i-1}$.
                   // $y_i = \mathbf{Th}(P, T_{i,b_i-1}, \xi_i)$
7          else
8              $y_i \xleftarrow{\$} \{0,1\}^n$.
9          Set $\sigma_i = y_i$.
10     Get the seed $P$ from $C$ and set Seed $= P$.
11     Compute public key $\mathsf{pk} = (\mathsf{pk}_1, \ldots \mathsf{pk}_l)$, as $\mathsf{pk}_i = c^{w-1-b_i}(y_i, i, \text{Seed})$.
12 begin Obtaining the result
13     Return $\sigma$ and $\mathsf{PK} = (\mathsf{pk}, P)$ to $\mathcal{A}$.
14     if $\mathcal{A}$ returns a valid forgery $(M', \sigma')$ then
15         Compute $B' = (b'_1, \ldots, b'_l)$ encoding $M'$
16         if $\exists 1 \le j \le l$ such that $b'_j < b_j$ and $c^{(b'_j, b_j - b'_j)}(\sigma'_j, j, \text{Seed}) = \sigma_j$ then
17             return W-SM-PRE solution $(j, c^{b'_j, (b_j - b'_j - 1)}(\sigma'_j, j, \text{Seed}))$
18         else
19             return fail
20     else
21         return fail
```

construction. In this section we will show that we can reduce the security of multiple WOTS-TW instances to the same multi-target security properties used for a single instance. While, obviously, the number of targets increases, we argued in Section 2.5 that the complexity of generic attacks is not influenced by the number of targets for these notions. Hence, there is no decrease in security to be expected when using multiple instances. We will show that this even works when the same public seed is used for all instances, as long as different prefixes are used for the tweaks.

In SPHINCS-like constructions WOTS-TW is used to sign the roots of trees which are not controlled by an adversary against the construction but by the signer. More generally, this is the case in many such constructions. Hence we use an extension of the EU-nCMA model from last section to d instances as security model.

We formally define EU-nCMA security for d instances. We introduce a definition for the security of $\mathsf{Dss}(1^n)$, which we call the d-existential unforgeability under non-adaptive chosen message attack (d-EU-naCMA). It is defined using the following experiment. In this experiment a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, A_2)$ is allowed to make signing queries to an oracle $sign(\cdot, \cdot, \mathcal{S}, Seed)$ and query oracle $\mathbf{Th}_\lambda$. The signing oracle takes **ADRS** as a first input an message M as a second input. First it runs $(\mathsf{SK}, \mathsf{PK}) \leftarrow \text{WOTS.kg}(\mathcal{C} = (Seed, \mathbf{ADRS}); \mathcal{S})$. Then it computes $\sigma \leftarrow \text{WOTS.sign}(M; \mathsf{SK})$. Let us denote PK' which is equal to PK but there is no $Seed$ in PK', i.e. $\mathsf{PK}' = (\mathsf{pk}, \mathbf{ADRS})$. The signing oracle return $(\sigma, M, \mathsf{PK}')$ to the adversary. We also restrict $\mathcal{A}$ from querying $\mathbf{Th}_\lambda$ with tweaks for **ADRS**-s that are used in signature queries. We define a function $\mathsf{adrs}(\cdot)$ that takes a tweak as an input and returns **ADRS** of that tweak. We denote the set of queries to signing oracle as $Q = \{(\mathbf{ADRS}_1, M_1), \dots, (\mathbf{ADRS}_d, M_d)\}$ and the set of tweaks that are used to query $\mathbf{Th}_\lambda$ is $T = \{T_1, \dots T_p\}$. We are concerned with one-time signatures, so the number of allowed signing queries for each **ADRS** is restricted to 1.

Experiment $\mathsf{Exp}^{\mathsf{d-EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A})$

- $Seed \xleftarrow{\$} \{0,1\}^n$
- $\mathcal{S} \xleftarrow{\$} \{0,1\}^n$
- $state \leftarrow \mathcal{A}_1^{sign(\cdot,\cdot,Seed,\mathcal{S}),\mathbf{Th}_\lambda(Seed,\cdot,\cdot)}()$
- $(M^\star, \sigma^\star, j) \leftarrow \mathcal{A}_2(state, Seed),\ j \in [1, d]$
- Return 1 iff $[\text{Vf}(\mathsf{PK}_j, \sigma^\star, M^\star) = 1] \wedge [M^\star \neq M_j] \wedge [\forall \mathbf{ADRS}_i \in Q, \mathbf{ADRS}_i \notin T' = \{\mathsf{adrs}(\mathsf{T_i})\}_{i=1}^p]$.

The following theorem can be proved by generalization of the proof of theorem 1. The main idea behind the proof is the following. First of all we use different tweaks in different instances of WOTS-TW as we use different **ADRS** for each instance. Next point is that we obtain d times more challenges and we separate them in d sets. Each set will be used for appropriate instance of WOTS. Then the proof follows the same path as in theorem 1.

Theorem 2. *Let n, $w \in \mathbb{N}$ and $w = poly(n)$. Let* $\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *be a* W-SM-TCR*,* W-SM-PRE*,* W-SM-UD *function. Let* $\mathbf{PRF} : \mathcal{S} \times \mathcal{T} \rightarrow \{0,1\}^n$ *be a pseudorandom function. Then the insecurity of the WOTS-TW scheme against d-EU-naCMA attack is bounded by*

$$\begin{aligned}\mathrm{InSec}^{\mathrm{d-EU-naCMA}}(\text{WOTS-TW}(1^n, w); t, 1) < \\ \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, d \cdot l) + \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, d \cdot lw) + \\ \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, d \cdot l) + w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, d \cdot l) \quad (3)\end{aligned}$$

with $\widetilde{t} = t + d \cdot lw$, where time is given in number of $\mathbf{Th}$ *evaluations.*

Proof sketch. Let us give a brief description how the generalization will be obtain. We have the same game hopping as in Theorem 1.

GAME.1 is the original EU-nCMA game and GAME.2 is the same as GAME.1 but instead of using pseudorandom elements from $\mathbf{PRF}$ a truly random function $\mathbf{RF} : \mathcal{S} \times \mathcal{T} \rightarrow \{0,1\}^n$ is used. $|\mathrm{Succ}^{\mathrm{GAME.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, d \cdot l)$ The reasoning here is the same as in Claim 1 in Theorem 1. Note that here inputs on which the oracle in PRF game is queried are all unique due to the unique **ADRS**-s for each instance. Hence, the outputs are uniformly random values as desired.

GAME.3 is different from GAME.2 in that for each signing query we answer with a hash of random value rather than building it with a chaining function. In Claim 2 of Theorem 1 we reduced it to W-SM-UD property by using hybrid argument. Here we need to apply the same reasoning. To obtain needed hybrids in case of d instances we will do the following. Let us denote the transformation of M_i into $b_1, \ldots, b_l$ with extra index. So M_i is transferred into $b_{i,1}, \ldots, b_{i,l}$. We denote two distributions where $D_{d-Kg} = \{y_{1,1}, \ldots, y_{1,l}, \ldots, y_{d,1}, \ldots, y_{d,l}\}$, where $y_{i,j} = c^{0,b_j-1}_{\mathbf{ADRS}_i}(\xi_{i,j}, j, \mathrm{Seed})$, $\xi_{i,j} \xleftarrow{\$} \{0,1\}^n$, $i \in [1,d]$, $j \in [1,l]$, and $D_{d-0} = \{\xi_{1,1}, \ldots, \xi_{1,l}, \ldots, \xi_{d,1}, \ldots, \xi_{d,l}\}$, where $\xi_{i,j} \xleftarrow{\$} \{0,1\}^n$, $i \in [1,d]$, $j \in [1,l]$. The distinguishing advantage of adversary of those two distributions is exactly the difference of success probability this game hop. To limit this distinguishing advantage we need to build hybrids. We do this is the same manner as in Theorem 1. Let $b_{\max} = \max\{b_{1,1}, \ldots, b_{1,l} \ldots, b_{d,1}, \ldots, b_{d,l}\}$ be the maximum of the values in the message encoding of all M_i. Let H_k be the distribution obtained by computing the values as $c^{k,b_{i,j}-1-k}_{\mathbf{ADRS}_i}(\xi_{i,j}, j, Seed)$, $\xi_{i,j} \xleftarrow{\$} \{0,1\}^n$. One can notice that $H_0 = D_{Kg}$ and $H_{b_{\max}-1} = D_0$. There must be two consecutive hybrids H_γ and $H_{\gamma+1}$ that we can distinguish with probability close to distinguishing advantage. By playing W-SM-UD and interacting with the oracle $\mathcal{O}(\cdot, b)$ we can construct hybrid $H_{\gamma+b}$ this is done in just the same way as in Claim 2 of Theorem 1. Hence we obtain the following bound:

$$|\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, d \cdot l)$$

Notice that in case of one instance we obtained Seed from W-SM-UD challenger that we used to construct the hybrids and obtain WOTS-TW public key. Here instead of using Seed we need to interact with $\mathbf{Th}_\lambda(\text{Seed}.\cdot,\cdot)$ oracle. Only after all of the signing queries are made we will obtain the Seed.

GAME.4 is different from GAME.3 in that we are considering the game lost if an adversary outputs a valid forgery $(M^\star, \sigma^\star, j)$ where there exist such i that $b^\star_{i,j} < b_{i,j}$ and $c^{(b^\star_{i,j}, b_{i,j}-b^\star_{i,j})}_{\mathbf{ADRS}_i}(\sigma^\star_j, j, \text{Seed}) \neq \sigma_j$. To show that $|\text{Succ}^{\text{GAME.3}}(\mathcal{A}) - \text{Succ}^{\text{GAME.4}}(\mathcal{A})| \leq \text{InSec}^{\text{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, d \cdot lw)$ we can build the reduction that works as follows.To answer signature queries query and compute the public key, the reduction interacts with the W-SM-TCR oracle. The difference in case of d instances from one instance is that we will need d times more interactions with W-SM-TCR oracle. The chains that we build and chains obtained from the forged signature are different but lead to the same public key. Hence by a pigeon hole argument there must be a collision on the way to public key element. This collision is a solution for W-SM-TCR challenge. So we proved that

$$|\text{Succ}^{\text{GAME.3}}(\mathcal{A}) - \text{Succ}^{\text{GAME.4}}(\mathcal{A})| \leq \text{InSec}^{\text{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, d \cdot lw)$$

To give a bound on success probability for GAME.4 we use W-SM-PRE property. To answer signing queries we will interact with W-SM-PRE oracle and place challenges obtained from that oracle in place of signatures. To construct public keys of WOTS-TW instances we will behave in the same way as in the undetectability case. By interacting with $\mathbf{Th}_\lambda(\text{Seed},\cdot,\cdot)$ we can build the chains of WOTS-TW structures. Again there must exist a j such that $b^\star_{i,j} < b_{i,j}$ by the properties of the checksum. And since we excluded the case where $c^{(b^\star_{i,j}, b_{i,j}-b^\star_{i,j})}_{\mathbf{ADRS}_i}(\sigma^\star_j, j, \text{Seed}) \neq \sigma_j$ we can obtain a preimage by computing $c^{(b^\star_{i,j}-1, b_{i,j}-b^\star_{i,j})}_{\mathbf{ADRS}_i}(\sigma^\star_j, j, \text{Seed})$ So we obtained

$$|\text{Succ}^{\text{GAME.4}}(\mathcal{A})| \leq \text{InSec}^{\text{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, d \cdot l)$$

This concludes the sketch of the proof.

6 SPHINCS$^+$

In this section we will recap the SPHINCS$^+$ structure and afterwards give fixes to the original SPHINCS$^+$ proof. To obtain a fixed proof we will utilize the results from Theorem 2.

6.1 Brief description

First we give a brief description of the SPHINCS$^+$ signature scheme. An example of SPHINCS$^+$ structure can be seen on the following picture:

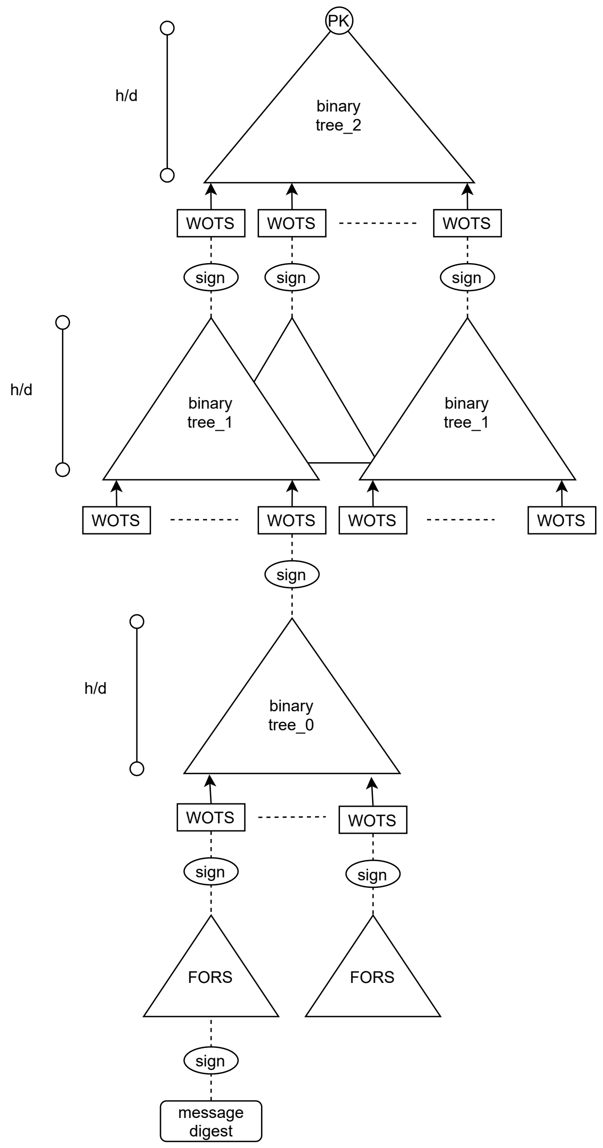

Fig. 3: Example of a SPHINCS$^+$ structure

The public key consists of two n-bit values: a random public seed **PK**.seed and the root of the top tree in the hypertree structure. **PK**.seed is used as a first argument for all of the tweakable hash functions calls. The private key contains two more n-bit values **SK**.seed and **SK**.prf.

Lets describe the main parts of it. First we have to describe addressing scheme. In this scheme, the addresses of certain structures and calls of hash functions will be used. This is for pseudo-random data generation. The address is a 32 byte value. Address coding can be done in any convenient way. An address will be represented with a tweak. Each tweak has a prefix that denotes to which part of SPHINCS$^+$ structure it belongs. We denoted this prefix as **ADRS** in previous sections. So for each hashing in the structure a different tweak will be used.

Then we need to discuss binary trees. In the SPHINCS$^+$ algorithm, binary trees of height γ always have 2^γ leaves. Each leaf $L_i,\ i\ in[2^\gamma - 1]$ is a bit string of length n. Each node of the tree $N_{i,j},\ 0 < j \leq \gamma, 0 \leq i < 2^{\gamma - j}$ is also a bit string of length n. The values of the internal nodes of the tree are calculated as a hash function from the children of that node. A leaf of a binary tree is a hash of elements of public key of a WOTS-TW signature scheme instance.

Binary trees and WOTS-TW signature schemes are used to construct a hypertree structure. WOTS-TW instances are used to sign root of binary trees on lower levels. WOTS-TW instances on the lowest level used to signed the root of a FORS structure. FORS is defined with the following parameters: $k \in \mathbb{N}$, $t = 2^a$. This algorithm can sign messages of length ka-bits.

FORS key pair. The private key of FORS consists of kt pseudorandomly generated n-bit values grouped into k sets of t elements each.

To get the public key k binary hash trees are constructed. The leaves in these trees are k sets (one for each tree) which consist of t values. Thus we get k trees of height a. As roots of k binary trees are calculated they are compressed using a hash function. The resulting value will be the FORS public key.

FORS Signature. A message from ka bits is divided into k lines of a bits. Each of these lines is interpreted as a leaf index corresponding to one of the k trees. The signature consists of these sheets and their authentication paths. The verifier reconstructs the tree roots, compresses them, and verifies them against the public key. If there is a match, it is said that the signature was verified. Otherwise, it is declared invalid.

The last thing to discuss is the way the message digest is calculated. First, we will prepare the pseudo-random value $\mathbf{R}$. It is calculated from $\mathbf{SK}.\mathsf{prf}$ and the message. Also, this value can be non-deterministic, this can be achieved by adding a random value $\mathsf{OptRand}$. Thus $\mathbf{R} = \mathbf{PRF}_{msg}(\mathbf{SK}_{\mathsf{prf}}, \mathsf{OptRand}, M)$. The $\mathbf{R}$ value is part of the signature. Now, using $\mathbf{R}$, we calculate the index of the sheet with which the message will be signed and the hash of the message itself. $(\mathsf{MD}||\mathsf{idx}) = \mathbf{H_{msg}}(\mathbf{R}, \mathbf{PK}.\mathsf{seed}, \mathbf{PK}.\mathsf{root}, M)$.

The signature consists of the randomness $\mathbf{R}$, FORS signature (which corresponds to index idx from $\mathbf{H_{msg}}$) of the message digest, the WOTS-TW signature of the corresponding FORS public key, and a set of authentication paths and WOTS-TW signatures of tree roots. To test this chain, the verifier iteratively reconstructs the public keys and tree roots until it gets the root of the top tree.

Given the brief description of SPHINCS$^+$ lets analyze the modifications of its reduction proof.

6.2 SPHINCS$^+$ proof

In this part we want to update the proof of security of SPHINCS$^+$ framework. The security had several issues which are described in [MK, ABB$^+$20]. One can look up for the description of the SPHINCS$^+$ scheme in [BHK$^+$19].

For the SPHINCS$^+$ construction we will need the following functions:

- $\mathbf{F}$: $\mathcal{P} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$;
- $\mathbf{H}$: $\mathcal{P} \times \mathcal{T} \times \{0,1\}^{2n} \to \{0,1\}^n$;
- $\mathbf{Th}_l$: $\mathcal{P} \times \mathcal{T} \times \{0,1\}^{ln} \to \{0,1\}^n$;
- PRF: $\{0,1\}^n \times \{0,1\}^{256} \to \{0,1\}^n$;
- $\mathbf{PRF_{msg}}$: $\{0,1\}^n \times \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$;
- $\mathbf{H_{msg}}$: $\{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^m$;

In this section, we prove the following Theorem. Note that $\mathbf{F}$, $\mathbf{H}$, $\mathbf{Th}_l$ are members of a collection of tweakable hash functions with different message length.

Theorem 3. *For parameters n, w, h, d, m, t, k as described in [BHK$^+$19] the following bound can be obtained:*

$$\begin{aligned}
&\mathrm{InSec}^{\mathrm{EU-CMA}}(SPINCS+; \xi, q_s) \leq \\
&\mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}, \xi, q_1) + \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF_{msg}}, \xi, q_s) + \\
&\mathrm{InSec}^{\mathrm{ITSR}}(\mathbf{H_{msg}}, \xi, q_s) + w^2 \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, q_2) + \\
&\mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_3) + \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \xi, q_4) + \\
&3 \cdot \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_5) + \mathrm{InSec}^{\mathrm{SM-DSPR}}(\mathbf{Th}; \xi, q_5)
\end{aligned}$$

where $q_1 < 2^{h+1}(kt + l)$, $q_2 < 2^{h+1} \cdot l$, $q_3 < 2^{h+2}(w \cdot l + 2kt)$, $q_4 < 2^{h+1} \cdot l \cdot w$, $q_5 < 2^h \cdot kt$ and q_s denotes the number of signing queries made by $\mathcal{A}$.

Proof. We want to bound the success probability of a (quantum) adversary $\mathcal{A}$ against the EU-CMA security of SPHINCS$^+$. Towards this end we use the following series of games. We start with GAME.0 which is the EU-CMA experiment for SPHINCS$^+$. Now consider a GAME.1 which is essentially GAME.0 but the experiment makes use of a SPHINCS$^+$ version where all the outputs of PRF, i.e., the WOTS-TW + and FORS secret-key elements, get replaced by truly random values.

Next, consider a game GAME.2, which is the same as GAME.1 but in the signing oracle $\mathbf{PRF_{msg}}(\mathrm{SK.prf}, \cdot)$ is replaced by a truly random function.

Afterwards, we consider GAME.3, which differs from GAME.2 in that we are considering the game lost if an adversary outputs a valid forgery $(\mathrm{M}, \mathrm{SIG})$ where the FORS signature part of SIG contains only secret values which were contained in previous signatures with that FORS key pair obtained by $\mathcal{A}$ via the signing oracle.

Now consider what are the possibilities of the adversary to win the game. The FORS signature in a forgery must include the preimage of a FORS leaf node that was not previously revealed to it. There are two separate cases for that leaf:

1. The FORS leaf is different to the leaf that we would generate for that place.
2. The FORS leaf is the same to the leaf that we would generate for that place;

Lets consider GAME.4 which differs from GAME.3 in that we are considering that the WOTS-TW signatures are made as described in the proof of Claim 2 in Section 4, the game is lost in the first "leaf case" scenario or an adversary outputs a valid forgery (M, SIG) which (implicitly or explicitly) contains a second preimage for an input to **Th** that was part of a signature returned as a signing-query response.

Now lets analyze those games.

GAME.0 - GAME.3 The hops between GAME.0 and GAME.3 are fully presented in the SHINCS+ paper [BHK+19]. The bound for these games are

$$|\mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.0}} - \mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.1}}| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}, \xi, q_1), \tag{4}$$

$$|\mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.1}} - \mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.2}}| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF_{msg}}, \xi, q_s), \tag{5}$$

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.2} - \mathrm{Succ}_{\mathcal{A}}^{GAME.3}| \leq \mathrm{InSec}^{\mathrm{ITSR}}(\mathbf{H_{msg}}, \xi, q_s), \tag{6}$$

where $q_1 < 2^{h+1}(kt + l)$ and q_s denotes the number of signing queries made by $\mathcal{A}$.

GAME.3 - GAME.4 Lets break the hop between GAME.3 and GAME.4 into several steps. First assume GAME.3.1 in which random values are used to create WOTS-TW signatures. This is the case that we discussed in the proof of Claim 2 in Section 4. But in this case we have several message queries as described in previous section. Using $\mathbf{Th}_\lambda$ oracle we first obtain FORS instances. Then we can query W-SM-UD to obtain challenges that will be placed in WOTS-TW instances. Again after placing those challenges we use $\mathbf{Th}_\lambda$ to obtain WOTS-TW public keys and build Merkle trees so we go to the next level of the hypertree. Having the roots of Merkle trees we again interact with W-SM-UD challenger to obtain challenges for this set of WOTS-TW instances. We continue this process until we reach the root of the highest Merkle tree in our hypertree structure. Then we obtain public parameter P and set $\mathbf{PK}.\mathsf{seed} = P$. By the reasoning from previous section we obtain

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.3} - \mathrm{Succ}_{\mathcal{A}}^{GAME.3.1}| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, q_2) \tag{7}$$

where $q_2 < 2^{h+1} \cdot len$.

Next consider game GAME.3.2, which differs from GAME.3.1 in that we are considering the game lost if an adversary outputs a valid forgery (M, SIG) which (implicitly or explicitly) contains a second preimage for an input to **Th** that was part of a signature returned as a signing-query response. By implicitly we here refer to a second preimage which is observed during the verification of the signature.

We can build a reduction $\mathcal{M}^{\mathcal{A}}$ that breaks H-SM-TCR. Here we slightly abuse the notation and we assume that we we obtain W-SM-TCR challenges for different members of collection, i.e. playing against several instances of the tweakable collection at one. We use $\mathbf{Th}_m$ (that differs in m) to compute the whole SPHINCS$^+$ structure. The reduction builds the whole SPHINCS$^+$ structure of a key pair (the key pair plus the whole hypertree including all FORS key pairs and WOTS-TW) during setup using the H-SM-TCR and $\mathbf{Th}_\lambda$ oracles and stores all computed values. Assume the collision occurs in the WOTS-TW instances. Than we deal with this case as was show in the Theorem 2. If the collision occurs not in WOTS-TW instances then we can obtain challenges for those places through interaction with W-SM-TCR oracle for several members. Here the difference from [BHK$^+$19] is only in the way we deal with WOTS-TW.

Thereby it defines all inputs to $\mathbf{Th}_m$ as targets. In total,the reduction $\mathcal{M}^{\mathcal{A}}$ makes $q_3 < 2^{h+2}(w \cdot len + 2kt)$ queries to its oracles. When $\mathcal{M}^{\mathcal{A}}$ is done, it obtains the public parameters from the challenger and puts these into the public key together with the generated root. Then it runs $\mathcal{A}$ with this public key as input. $\mathcal{M}^{\mathcal{A}}$ can answer all signature queries and perfectly simulates the EU-CMA game for SPHINCS$^+$.

When $\mathcal{A}$ returns a forgery, $\mathcal{M}^{\mathcal{A}}$ runs verification and compares all computed values to the values it computed during set-up. If $\mathcal{M}^{\mathcal{A}}$ finds a second preimage it outputs it together with its query index (indicating when it was sent to the W-SM-TCR oracle).

Hence we obtain

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.3.1} - \mathrm{Succ}_{\mathcal{A}}^{GAME.3.2}| \leq \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_3) \tag{8}$$

So the only case left to hop to GAME.4 is a WOTS-TW forgery that gives us the preimage. Since iff there is no WOTS-TW forgery then there must be a collision. And if there is the WOTS-TW forgery we can obtain either a collision (this case we have already excluded) or a preimage. To do so we first obtain preimage challenges for appropriate tweaks that were used in every WOTS-TW instance. To obtain the positions to place challenges for WOTS-TW instances we will again use $\mathbf{Th}_\lambda$ oracle (in the same way as for W-SM-UD). We use it to build WOTS-TW public keys and the whole SPHINCS$^+$ structure. This is done the same way as it was described in previous section. We obtain the following bound

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.3.2} - \mathrm{Succ}_{\mathcal{A}}^{GAME.4}| \leq \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \xi, q_4) \tag{9}$$

where $q_4 < 2^{h+1} \cdot l \cdot w$.

GAME.4 The analysis of the GAME.4 can be found in the SPHINCS$^+$ paper [BHK$^+$19](Claim 23). Here we note that we can not use W-SM-PRE bound as the reduction uses T-openPRE game that was introduced in [BH19b]. The only difference is that we have already excluded the WOTS-TW preimage case. Hence we obtain the following bound:

$$\mathrm{Succ}_{\mathcal{A}}^{GAME.4} \leq 3 \cdot \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_5) + \mathrm{InSec}^{\mathrm{SM-DSPR}}(\mathbf{Th}; \xi, q_5) \quad (10)$$

where $q_5 < 2^h \cdot kt$

Combining the inequalities we obtain the bound from the theorem.

References

ABB$^+$20. Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS$^+$. Submission to NIST's post-quantum crypto standardization project, v.3, 2020. http://sphincs.org/data/sphincs+-round3-specification.pdf.

BH19a. Daniel J. Bernstein and Andreas Hülsing. Decisional second-preimage resistance: When does SPR imply PRE? In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 33–62. Springer, Heidelberg, December 2019.

BH19b. Daniel J. Bernstein and Andreas Hülsing. Decisional second-preimage resistance: When does SPR imply PRE? Cryptology ePrint Archive, Report 2019/492, 2019. https://eprint.iacr.org/2019/492.

BHK$^+$19. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2129–2146. ACM Press, November 2019.

DSS05. C. Dods, Nigel P. Smart, and Martijn Stam. Hash based digital signature schemes. In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 96–115. Springer, Heidelberg, December 2005.

HRS16. Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 387–416. Springer, Heidelberg, March 2016.

Hül13. Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13*, volume 7918 of *LNCS*, pages 173–188. Springer, Heidelberg, June 2013.

MK. Aleksey Fedorov Mikhail Kudinov, Evgeniy Kiktenko. [pqc-forum] round 3 official comment: Sphincs+. https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/official-comments/Sphincs-Plus-round3-official-comment.pdf. Accessed: 2010-10-30.

RSM09. Mohammad Reza Reyhanitabar, Willy Susilo, and Yi Mu. Enhanced target collision resistant hash functions revisited. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 327–344. Springer, Heidelberg, February 2009.

Kerman, Sara J. (Fed)

From:	Andreas Hülsing <andreas@huelsing.net>
Sent:	Tuesday, November 2, 2021 5:00 AM
To:	pqc-comments
Cc:	pqc-forum
Subject:	ROUND 3 OFFICIAL COMMENT: SPHINCS+
Attachments:	SPHINCS+-updated.pdf

Hi all,

my student Mike sent below message on Sunday. As it does not seem to get through I am resending it from my mail.
TL,DR: We fixed the tight SPHINCS+ security proof.

Cheers,

Andreas

Dear all,

Attached you find a paper that describes a new, tight security proof for SPHINCS+. Our paper fixes the flaw that was pointed out by Kudinov, Kiktenko, and Fedorov last year, following the outline we gave in the response to that message back then. More details can be found in the attached file.

Best regards,
Mikhail & Andreas

Security of WOTS-TW scheme with a weak adversary

Andreas Hülsing[1] and Mikhail Kudinov[1,2]

[1] Eindhoven University of Technology, Eindhoven, Netherlands
[2] Russian Quantum Center, QApp, Skolkovo, Moscow 143025, Russia

Abstract. In 2020, Kudinov, Kiktenko, and Fedorov pointed out a flaw in the tight security proof of the SPHINCS$^+$ construction. This work gives a new tight security proof for SPHINCS$^+$. The flaw can be traced back to the security proof for the used Winternitz one-time signature scheme (WOTS).
We give the first standalone description of the WOTS variant used in SPHINCS$^+$ that we call WOTS-TW. We provide a security proof for WOTS-TW and and multi-instance WOTS-TW in the EU-naCMA model, a non-adaptive chosen message attack setting where the adversary only learns the full public key after it made its signature queries. Afterwards, we show that this is sufficient to give a tight security proof for SPHINCS$^+$. We almost recover the same bound for the security of SPHINCS$^+$, with only a factor w loss, where w is the Winternitz parameter that is commonly set to 16.

Keywords: Post-quantum cryptography, hash-based signatures, SPHINCS$^+$ W-OTS, WOTS-TW.

1 Introduction

Hash-based signatures recently received a lot of attention. Hash-based signatures are widely considered the most conservative choice for post-quantum signature schemes. At the time of writing, SPHINCS$^+$, a stateless hash-based signature scheme, is a third round alternate candidate in the NIST PQC competition. However, NIST highlighted over and over that

> "NIST sees SPHINCS+ as an extremely conservative choice for standardization. If NIST's confidence in better performing signature algorithms is shaken by new analysis , SPHINCS+ could provide an immediately available algorithm for standardization at the end of the third round."
> (Dustin Moody on the pqc-forum mailing list by after new attacks on Rainbow and GeMSS were published, January 21, 2021)

This work was funded by an NWO VDI grant (Project No. VI.Vidi.193.066). Part of this work was done while M.K. was still affiliated with the Russian Quantum Center, QApp. Date: October 31, 2021

One more supporting argument for the security of SPHINCS$^+$ would be a tight security reduction that allows to derive attack complexities for a given set of parameters. However, the tight proof for SPHINCS$^+$ that was given in [BHK$^+$19] turned out to be flawed [MK]. The flaw, pointed out by Kudinov, Kiktenko, and Fedorov is related to the proof of security of the used WOTS scheme. Although the flaw could not be translated into an attack, this resulted in an unsatisfactory situation. While there still exists a non-tight reduction for the security of SPHINCS$^+$, this reduction can not support the claimed security of the used SPHINCS$^+$ parameters.

In this work we give a new tight security proof for SPHINCS$^+$ that closes the gap again without modifying the scheme.

To make the proof easier accessible, we first extract the variant of WOTS scheme that is used in SPHINCS$^+$ and formally define it, naming it WOTS-TW. WOTS-TW is different from other WOTS variants in that it uses tweakable hash functions, introduce in [BHK$^+$19], to construct the function chains. We then prove the security of WOTS-TW in the EU-naCMA model. This model differs from the common EU-CMA model in that the adversary only receives the public key after it made its signature query. We choose this model because it allows for a tight security proof while it suffices for a proof of applications like SPHINCS$^+$. The important feature here is that a reduction can generate the WOTS-TW public key based on the signature query and does not have to guess that query. This is possible as WOTS-TW is used to sign roots of hash trees in applications like SPHINCS$^+$. Our new proof combines the work of Dods, Smart, and Stam [DSS05] that uses undetectability to plant preimage challenges, with the second-preimage resistance version of Hülsing [Hül13], and the approach of multi-target mitigation by Hülsing, Rijneveld, and Song [HRS16] and lifts it to the setting of tweakable hash functions.

While the single instance proof allows us to state the new proof in a more accessible setting, SPHINCS$^+$ uses multiple instances at once. Hence, we afterwards extend the result to multiple instances. This setting turns out to be slightly more involved as we have to allow for messages to depend on public keys of previously used instances. We end up with a proof in a slightly more involved model that, however, allows us to use the result in the security proof for SPHINCS$^+$. We conclude the work with the tight security proof for SPHINCS$^+$.

While we give a full proof for SPHINCS$^+$, we leave one aspect for future work. In section 2.5 we collect bounds on the success probability of generic attacks against the used properties of tweakable hash functions. So far only one of these bounds is fully proven. All others make some form of conjecture. We hope to be able to close this gap in future work.

The paper is organized as follows. We introduce necessary definitions and notations as well as describe the EU-naCMA security model in Section 2. At the end of Section 2 we also summarize the state of the art for generic security bounds. The description of the WOTS-TW scheme is given in Section 3. In Section 4 we provide a security reduction for WOTS-TW in the single instance setting and in Section 5 we lift the result to the multi-instance setting with possibly dependent messages. The security proof for SPHINCS$^+$ that uses WOTS-TW as a building block is then given in Section 6.

2 Preliminaries

In this section we introduce the definitions of building blocks, and security notions for hash functions and signatures that we use. We begin with the notion of a tweakable hash function, introduced in the construction of SPHINCS$^+$ [BHK$^+$19], and its security. Afterwards we move on to digital signatures.

2.1 Tweakable hash functions.

In this section we recall the definition of tweakable hash functions and related security notions from [BHK$^+$19]. These properties will later be used to prove the security of our WOTS-TW scheme.

Function definition. A *tweakable* hash function takes public parameters P and context information in form of a *tweak* T in addition to the message input. The public parameters might be thought of as a function key or index. The tweak might be interpreted as a nonce.

Definition 1 (Tweakable hash function). *Let $n, m \in \mathbb{N}$, $\mathcal{P}$ the public parameters space and $\mathcal{T}$ the tweak space. A tweakable hash function is an efficient function*

$$\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^m \to \{0,1\}^n, \ \mathrm{MD} \leftarrow \mathbf{Th}(P, T, M)$$

mapping an m-bit message M to an n-bit hash value MD *using a function key called public parameter $P \in \mathcal{P}$ and a tweak $T \in \mathcal{T}$.*

We will sometimes denote $\mathbf{Th}(P, T, M)$ as $\mathbf{Th}_{P,T}(M)$. In SPHINCS$^+$, a public seed *Seed* is used as public parameter which is part of the SPHINCS$^+$ public key. As tweak a so-called hash function address (**ADRS**) is used that identifies the position of the hash function call within the virtual structure defined by a SPHINCS$^+$ key pair. We use the same approach for WOTS-TW, i.e., the public parameter is a seed value that becomes part of the public key if WOTS-TW is used standalone. If it is used in a bigger structure like SPHINCS$^+$, the public parameters will typically be those used in the bigger structure and are therefore only part of that bigger structure public key. In this case, the hash addresses have to be unique within the whole bigger structure. Therefore, the address may contain a prefix determined by the calling structure.

Security notions. To provide a security proof for WOTS-TW we require that the used tweakable hash functions have certain security properties. We require the following properties or some variations of them which will be discussed below:

- *post-quantum single function, multi-target-collision resistance for distinct tweaks* (PQ-SM-TCR);

– *post-quantum single function, multi-target-preimage resistance for distinct tweaks* (PQ-SM-DT-PRE);
– *post-quantum single function, multi-target-undetectability for distinct tweaks* (PQ-SM-DT-UD).

These properties were already considered in previous work. We only slightly adapt them, introducing a *weak* variant that adds some limitation on when the adversary is allowed to get access to the tweakable hash function and in what way. Moreover, in the context of multi-instance constructions like SPHINCS$^+$, we need another generic extension to collections of tweakable hash functions, discussed at the end of the subsection.

We generally consider post-quantum security in this work. Therefore, we will omit the PQ prefix from now on and consider it understood that we always consider quantum adversaries. Since we are working in the post-quantum setting, we assume that adversaries have access to a quantum computer but honest parties do not. Hence, any oracles that implement secretly keyed functions only allow for classical queries.

In all of the properties an adversary can influence the challenges by specifying the tweaks used in challenges. We generally restrict this control in so far as we do not allow more than one challenge for the same tweak. As we have this restriction for all of our properties we do not add any label to the security notions for this.

Now we will discuss above properties and their variations.

SM-TCR. One can view SM-TCR as a variant of target-collision resistance [RSM09]. Consider an adversary $\mathcal{A}$ which consists of two parts $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}$ will play a two-stage game. $\mathcal{A}_1$ is allowed to adaptively specify p targets (multi-target). The target specification is implemented via access to an oracle implementing the function with a fixed public parameter (single-function as the same public parameter is used for all targets). Every query to this oracle defines a target. It is important that $\mathcal{A}$ is not allowed to query the oracle with the same tweak more than once. The adversary wins if it finds a collision for one of the targets.

Lets consider a variant of SM-TCR in which the adversary gets the description of the tweakable function only after he has made the queries. This variation allows to make quantum queries to the hash function only after the targets are specified. We call it Weak-SM-TCR. In the reduction we will use this variant of the property. We will denote such a variant as W-SM-TCR. Notice that our reduction proof will show that if the scheme is broken at least one of the required properties of the tweakable hash function is broken. One can see that breaking the W-SM-TCR is harder than SM-TCR (at least not easier), hence it is less likely that the property will be broken and this leads to a higher security of the scheme.

We formalize the above in the following definition.

Definition 2 (W-SM-TCR). *In the following let* **Th** *be a tweakable hash function as defined above. We define the success probability of any adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against the* W-SM-TCR *security of* **Th**. *The definition is parameterized by the number of targets p for which it must hold that $p \leq |\mathcal{T}|$. In the* *definition, $\mathcal{A}_1$ is allowed to make p queries to an oracle $\mathbf{Th}(P,\cdot,\cdot)$. We denote the set of $\mathcal{A}_1$'s queries by $Q = \{(T_i, M_i)\}_{i=1}^{p}$ and define the predicate $\mathbf{DIST}(\{T_i\}_{i=1}^{p}) = (\forall i,k \in [1,p], i \neq k) : T_i \neq T_k$, i.e., $\mathbf{DIST}(\{T_i\}_{i=1}^{p})$ outputs 1 iff all tweaks are distinct.*

$$\begin{aligned}&\mathrm{Succ}^{\mathrm{W-SM-TCR}}_{\mathbf{Th},p}(\mathcal{A}) = \Pr[P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,\cdot)}(\ );\\ &(j,M) \leftarrow \mathcal{A}_2(Q,S,P,\mathbf{Th}) : \mathbf{Th}(P,T_j,M_j) = \mathbf{Th}(P,T_j,M)\\ &\wedge M \neq M_j \wedge \mathbf{DIST}(\{T_i\}_{i=1}^{p})]\end{aligned}$$

We define the insecurity of a tweakable hash function against p target, time ξ, W-SM-TCR *adversaries as the maximum success probability of any possibly quantum adversary $\mathcal{A}$ with p targets and running time $\leq \xi$:*

$$\mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th};\xi,p) = \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{W-SM-TCR}}_{\mathbf{Th},p}(\mathcal{A})\}$$

SM-PRE. As for W-SM-TCR, SM-PRE is a two-stage game and can be seen as a variant of preimage resistance. Adversary $\mathcal{A}_1$ is allowed to specify p targets during the first stage. The speciation is again done by using an oracle that implements a tweakable hash function with a fixed public parameter. The adversary wins if it finds a preimage for one of the targets.

We again consider a weaker version of SM-PRE in which the adversary gets the description of the tweakable function only after he has made the queries. We will denote such a variant as W-SM-PRE. The intuition behind this variant is the same as in W-SM-TCR.

We formalize the above in the following definition.

Definition 3 (W-SM-PRE)**.** *In the following let* $\mathbf{Th}$ *be a tweakable hash function as defined above. We define the success probability of any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the* W-SM-PRE *security of* $\mathbf{Th}$*. The definition is parameterized by the number of targets p for which it must hold that $p \leq |\mathcal{T}|$. In the definition, $\mathcal{A}_1$ is allowed to make p queries to an oracle $\mathbf{Th}(P,\cdot,x_i)$, where x_i is chosen uniformly at random for the query i (the value of x_i stays hidden from $\mathcal{A}$). We denote the set of $\mathcal{A}_1$'s queries by $Q = \{T_i\}_{i=1}^{p}$ and define the predicate $\mathbf{DIST}(\{T_i\}_{i=1}^{p})$ as we did in the definition above.*

$$\begin{aligned}&\mathrm{Succ}^{\mathrm{W-SM-PRE}}_{\mathbf{Th},p}(\mathcal{A}) = \Pr[P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,x_i)}(\ );\\ &(j,M) \leftarrow \mathcal{A}_2(Q,S,P,\mathbf{Th}) : \mathbf{Th}(P,T_j,M) = \mathbf{Th}(P,T_j,x_j)\\ &\wedge \mathbf{DIST}(\{T_i\}_{i=1}^{p})]\end{aligned}$$

We define the insecurity of a tweakable hash function against p target, time ξ, W-SM-PRE *adversaries as the maximum success probability of any possibly quantum adversary $\mathcal{A}$ with p targets and running time $\leq \xi$:*

$$\mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th};\xi,p) = \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{W-PRE}}_{\mathbf{Th},p}(\mathcal{A})\}$$

SM-UD. Also SM-UD is a variant of an established notion, in this case undetectability [DSS05], that makes use of a two stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. $\mathcal{A}_1$ specifies p targets during the first stage through oracle interactions. The oracle is initialized either with the tweakable hash function with a fixed public parameter or a random function. The adversary wins if it answers correctly whether it interacted with a random function or with a function $\mathbf{Th}$. We formalize the above in the following definition. As for the previous notions, we consider the weak variant where $\mathcal{A}$ gets the description of the tweakable hash function only after the challenge queries are made.

Definition 4 (W-SM-UD). *In the following let* $\mathbf{Th}$ *be a tweakable hash function as defined above. We define the success probability of any adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against the* W-SM-UD *security of* $\mathbf{Th}$. *The definition is parameterized by the number of targets* p *for which it must hold that* $p \leq |\mathcal{T}|$. *First the challenger flips a fair coin* b *and and chooses a public parameter* $P \xleftarrow{\$} \mathcal{P}$. *Next consider an oracle* $\mathcal{O}_P(\mathcal{T}, \{0,1\})$, *which works the following:* $\mathcal{O}_P(T, 0)$ *returns* $\mathbf{Th}(P, T, x_i)$, *where* x_i *is chosen uniformly at random for the query* i; $\mathcal{O}_P(T, 1)$ *returns* y_i, *where* y_i *is chosen uniformly at random for the query* i. *In the definition,* $\mathcal{A}_1$ *is allowed to make* p *queries to an oracle* $\mathcal{O}_P(\cdot, b)$. *We denote the set of* $\mathcal{A}_1$*'s queries by* $Q = \{T_i\}_{i=1}^p$ *and define the predicate* $\mathbf{DIST}(\{T_i\}_{i=1}^p)$ *as we did above.*

$$\begin{aligned}&\mathrm{Succ}_{\mathbf{Th},p}^{\mathrm{W-SM-UD}}(\mathcal{A}) = \Pr[P \xleftarrow{\$} \mathcal{P}; b \xleftarrow{\$} \{0,1\}; S \leftarrow \mathcal{A}_1^{\mathcal{O}_P(\cdot,b)}(\ );\\ &b' \leftarrow \mathcal{A}_2(Q, S, P.\mathbf{Th}) : b' = b\\ &\wedge \mathbf{DIST}(\{T_i\}_{i=1}^p)]\end{aligned}$$

We define the insecurity of a tweakable hash function against p *target, time* ξ, W-SM-UD *adversaries as the maximum success probability of any possibly quantum adversary* $\mathcal{A}$ *with* p *targets and running time* $\leq \xi$:

$$\mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Succ}_{\mathbf{Th},p}^{\mathrm{W-SM-UD}}(\mathcal{A})\}$$

Here we have finished describing the properties that will be needed to construct a reduction proof for WOTS-TW. But for the further analysis of those properties and analysis of bigger schemes such as SPHINCS$^+$ one would need several more properties, which will be listed below.

First we start with the notion of Decisional Second Preimage Resistance (DSPR) and it variants that was introduced in [BH19a]. This notion helps to reduce the property of preimage resistance to properties of second preimage resistance and DSPR. It is used in SPHINCS$^+$ to analyze the security of FORS scheme (this scheme will be described later). This property will be also utilized to analyze quantum generic security of W-SM-PRE. The initial description of the property and its justification can be found in [BH19a], here we will use the definitions from [BHK$^+$19] that fit the notion of a tweakable hash function.

We will go straight to a multi-target version of DSPR which is denoted as SM-DSPR. To do so first we need to introduce a predicate $\mathsf{SPexists}$.

Definition 5 ($\mathsf{SPexists}_{\mathsf{P,T}}$)**.** *A second preimage exists* $\mathsf{SPexists}$ *predicate of tweakable hash function* $\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^m \rightarrow \{0,1\}^n$ *with a fixed* $P \in \mathcal{P}$*,* $T \in \mathcal{T}$ *is the function* $\mathsf{SP}_{P,T} : \{0,1\}^m \rightarrow \{0,1\}$ *defined as follows:*

$$\mathsf{SP}_{P,T}(x) \overset{def}{=} \begin{cases} 1 \text{ if } |\mathbf{Th}^{-1}_{P,T}(\mathbf{Th}_{P,T}(x))| \geq 2 \\ 0 \text{ otherwise,} \end{cases}$$

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

In other words $\mathsf{SP}_{P,T}(x) = 0$ *is there is no second preimage for* x *under function* $\mathbf{Th}(P, T, \cdot)$*. Theoretically you can find a second preimage of* x *under* $\mathbf{Th}(P, T, \cdot)$ *only of* $\mathsf{SP}_{P,T}(x) = 1$*.*

Now we present the definition of SM-DSPR from [BHK+19].

Definition 6 (SM-DSPR)**.** *Let* $\mathbf{Th}$ *be a tweakable hash function. Let* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be a two stage adversary. The number of targets is denoted with* p*, where the following inequality must hold:* $p \leq |\mathcal{T}|$*.* $\mathcal{A}_1$ *is allowed to make* p *queries to an oracle* $\mathbf{Th}(P, \cdot, \cdot)$*. We denote the query set* Q *and predicate* $\mathsf{DIST}(\{T_i\}_1^p)$ *as in previous definitions.*

$$\mathrm{Adv}^{\mathrm{SM-DSPR}}_{\mathbf{Th},p}(\mathcal{A}) = \max\{0, \mathsf{succ} - \mathsf{triv}\},$$

where

$$\begin{aligned} \mathrm{Succ} = \Pr[&P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,\cdot)}(); (j, b) \leftarrow \mathcal{A}_2(Q, S, P) : \\ &\mathsf{SP}_{P,T_j}(M_j) = b \wedge \mathsf{DIST}(\{T_i\}_1^p)]. \\ \mathsf{triv} = \Pr[&P \xleftarrow{\$} \mathcal{P}; S \leftarrow \mathcal{A}_1^{\mathbf{Th}(P,\cdot,\cdot)}(); (j, b) \leftarrow \mathcal{A}_2(Q, S, P) : \\ &\mathsf{SP}_{P,T_j}(M_j) = 1 \wedge \mathsf{DIST}(\{T_i\}_1^p)]. \end{aligned}$$

We define the SM-DSPR *insecurity of a tweakable hash function against* p *target, time* ξ *adversaries as the maximum advantage of any (possibly quantum) adversary* $\mathcal{A}$ *with* p *targets and running time* $\leq \xi$*:*

$$\mathrm{InSec}^{\mathrm{SM-DSPR}}(\mathbf{Th}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Adv}^{\mathrm{SM-DSPR}}_{\mathbf{Th},p}(\mathcal{A})\}$$

W-SM-DSPR is a variant in which the description of the underlying hash function is given only after all the challenge queries are done.

Security for a collection of tweakable hash functions. In more complex constructions like SPHINCS$^+$, we make use a collection of tweakable hash functions which we call $\mathbf{Th}_\lambda$. In this case $\mathbf{Th}$ consists of a set of tweakable hash functions $\mathbf{Th}_m$ that differ in terms of m, the length of messages they process. This notion of a collection of tweakable hash functions is necessary as we use the same public parameters for all functions in the collection. Especially, it is necessary to make the security notions above usable in the context of SPHINCS$^+$. The problem is that when used in constructions like SPHINCS$^+$ or XMSS queries to the challenge oracle queries may depend on the outputs of other functions in the collection, or even the same function but with different tweaks. This is incompatible with above definitions as the public parameters are only given to the adversary after all challenge queries are made.

We solve this issue extending all the above *standalone* security properties to the case of collections. The definitions for functions that are part of a collection only differ from the above in a single spot. We give the first adversary $\mathcal{A}_1$, that makes the challenge queries, access to another oracle $\mathbf{Th}_\lambda(P, \cdot, \cdot)$, initialized with P. The oracle takes an input M and a tweak T and, depending on the length $m = |M|$ of M returns $\mathbf{Th}_m(P, M, T)$. The only limitation is that $\mathcal{A}$ is not allowed to use a tweak in queries to both oracles, the challenge oracle and the collection oracle. In general, $\mathcal{A}$ is allowed to query the challenge oracle as well as $\mathbf{Th}_\lambda$ with a message of length x, as long as the used tweak is never used in a query to the challenge oracle.

Definition 7 (W-SM-TCR, W-SM-PRE, W-SM-UD, SM-DSPR **for members of a collection).** *Let* $\mathbf{Th}$ *be a tweakable hash function as defined above with message length* x*. Moreover, let* $\mathbf{Th}$ *be an element of a collection of tweakable hash functions* $\mathbf{Th}_\lambda$ *as described above. Consider an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against the* W-SM-TCR *(,* W-SM-PRE*,* W-SM-UD*,* SM-DSPR*) security of* $\mathbf{Th}$ *in presence of collection* $\mathbf{Th}_\lambda$*. Let* $\mathbf{Th}_\lambda(P, \cdot, \cdot)$ *denote an oracle for* $\mathbf{Th}_\lambda$ *as described above and denote by* $\{T_i^\lambda\}_1^{p_\lambda}$ *the tweaks used in the queries made by* $\mathcal{A}$*. We define the success probability of* $\mathcal{A}$ *against* W-SM-TCR *(,* W-SM-PRE*,* W-SM-UD*,* SM-DSPR*) security of* $\mathbf{Th}$ *in presence of collection* $\mathbf{Th}_\lambda$ *as the success probability of* $\mathcal{A}$ *against standalone* W-SM-TCR *(,* W-SM-PRE*,* W-SM-UD*,* SM-DSPR*) security of* $\mathbf{Th}$ *defined above, when* $\mathcal{A}_1$ *is additionally given oracle access to* $\mathbf{Th}_\lambda(P, \cdot, \cdot)$ *with the condition that* $\{T_i\}_1^p \cap \{T_i^\lambda\}_1^{p_\lambda} = \emptyset$*.*

In the case of W-SM-TCR, we will abuse notation when it comes to the security of SPHINCS$^+$ and consider the joined security of several members of a collection of tweakable hash functions.

2.2 Message digest computation

In SPHINCS$^+$ a special function to compute message digest will be introduced. An expected property of that function is interleaved target subset resilience. Let give a formal definition of this property.

Definition 8 (ITSR **[BHK$^+$19]).** *Let* $\mathrm{H} : \{0,1\}^{\mathcal{K}} \times \{0,1\}^\alpha \to \{0,1\}^m$ *be a keyed hash function. Also consider a mapping function* $\mathsf{MAP}_{h,k,t} : \{0,1\}^m \to \{0,1\}^h \times [0, t-1]^k$ *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* $G = \mathsf{MAP}_{h,k,t} \circ \mathrm{H}$*. Let* $O(\cdot)$ *be an oracle which on input of an* α*-bit* *message M_i samples a key $K_i \xleftarrow{\$} \{0,1\}^{\mathcal{K}}$ and returns $G(K_i, M_i)$. The adversary $\mathcal{A}$ is allowed to query the oracle with massages of its choice. Denote the amount of queries with q.*

$$\begin{aligned}\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},q}(\mathcal{A}) = \Pr[&(R, M) \leftarrow \mathcal{A}^{O(\cdot)}(1^n)\\ &s.t.\ G(K, M) \subseteq \bigcup_{j=1}^{q} G(K_j, M_j) \wedge (K, M) \notin \{(K_j, M_j)\}_{j=1}^{q}],\end{aligned}$$

where $\{(K_j, M_j)\}_{j=1}^{q}$ represent the responses of the oracle $O(\cdot)$. We define the ITSR *insecurity of a keyed hash function against q-query, time-ξ adversaries as the maximum advantage of any quantum adversary $\mathcal{A}$ with running time $\leq \xi$, that makes no more than q queries:*

$$\mathrm{InSec}^{\mathrm{ITSR}}(\mathrm{H}; \xi, p) = \max_{\mathcal{A}}\{\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},q}(\mathcal{A})\}$$

2.3 Pseudorandom functions

To construct a scheme we will need pseudorandom functions. In this subsection we give a definition of pseudorandom functions, provide security notions.

Function definition. A *pseudorandom* function takes a secret parameter S and context information in form of a *tweak* T. The secret parameter might be thought of as a function key or index. The tweak might be interpreted as a nonce.

Definition 9 (Pseudorandom function). *Let $n \in \mathbb{N}$, $\mathcal{S}$ the secret parameters space and $\mathcal{T}$ the tweak space. A pseudorandom function is an efficient function*

$$F : \mathcal{S} \times \mathcal{T} \to \{0,1\}^n$$

generating an n-bit value out of secret parameter and a tweak.

Security notion In the following we give the definition for PRF security of a function $F : \mathcal{S} \times \mathcal{T} \to \{0,1\}^n$. In the definition of the PRF distinguishing advantage the adversary $\mathcal{A}$ gets (classical) oracle access to either $F(S, \cdot)$ for a uniformly random secret parameter $S \in \mathcal{S}$ or to a function G drawn from the uniform distribution over the set $\mathcal{G}(\mathcal{T}, n)$ of all functions with domain $\mathcal{T}$ and range $\{0,1\}^n$. The goal of $\mathcal{A}$ is to distinguish both cases.

Definition 10 (PRF**).** *Let F be defined as above. We define the* PRF *distinguishing advantage of an adversary $\mathcal{A}$ as*

$$\mathrm{Adv}^{\mathrm{PRF}}_{F}(\mathcal{A}) = |\Pr_{S \xleftarrow{\$} \mathcal{S}}[\mathcal{A}^{F(S,\cdot)} = 1] - \Pr_{G \xleftarrow{\$} \mathcal{G}(\mathcal{T},n)}[\mathcal{A}^{G(\cdot)} = 1]|.$$

We define the PRF *insecurity of a pseudorandom function F against q-query, time-ξ adversaries as the maximum advantage of any possibly quantum adversary that runs in time ξ and makes no more then q queries to its oracle:*

$$\mathrm{InSec}^{\mathrm{PRF}}(F; \xi, q) = \max_{\mathcal{A}}\{\mathrm{Adv}^{\mathrm{PRF}}_{F}(\mathcal{A})\}.$$

2.4 Security model

In this part we describe the security model in which we will prove the security of the one-time digital signature scheme.

Definition 11 (Digital signature schemes). *Let $\mathcal{M}$ be a message space. A digital signature scheme* $\mathsf{Dss} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ *is a triple of probabilistic polynomial time algorithms:*

– $\mathsf{Kg}(1^n)$ *on input of a security parameter* 1^n *outputs a private key* sk *and a public key* pk*;*
– $\mathsf{Sign}(\mathsf{sk}, M)$ *outputs a signature* σ *under secret key* sk *for message* $M \in \mathcal{M}$*;*
– $\mathsf{Vf}(\mathsf{pk}, \sigma, M)$ *outputs 1 iff* σ *is a valid signature on* M *under* pk*;*

such that $\forall(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^n), \forall(M \in \mathcal{M}) : \mathsf{Vf}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, M), M) = 1.$

Consider a signature scheme $\mathsf{Dss}(1^n)$, where n is the security parameter. We introduce a definition for the security of $\mathsf{Dss}(1^n)$, which we call the existential unforgeability under non-adaptive chosen message attack (EU-naCMA). It is defined using the following experiment.

```
Experiment $\mathsf{Exp}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A})$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kg}(1^n)$.
    $\{M_1, \ldots, M_q\} \leftarrow \mathcal{A}()$.
    Compute $\{(M_i, \sigma_i)\}_{i=1}^q$ using $\mathsf{Sign}(\mathsf{sk}, \cdot)$.
    $(M^\star, \sigma^\star) \leftarrow \mathcal{A}(\{(M_i, \sigma_i)\}_{i=1}^q, pk)$
    Return 1 iff $\mathrm{Vf}(\mathsf{pk}, \sigma^\star, M^\star) = 1$ and $M^\star \notin \{M_i\}_{i=1}^q$.
```

The adversary is forced to output a list of messages $M_1, ..., M_q$ it wants to see signed before obtaining the public key pk. In our work we consider one-time signatures, so the number of allowed messages q is set to 1.

Let $\mathrm{Succ}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A}) = 1\right]$ be the success probability of an adversary $\mathcal{A}$ in the above experiment.

Definition 12 (EU-naCMA). *Let* $t, n \in \mathbb{N}$*,* $t = \mathrm{poly}(n)$*,* $\mathrm{Dss}(1^n)$ *is a digital signature scheme. We call* $\mathrm{Dss}\ \mathrm{EU-naCMA}$*-secure if the maximum success probability* $\mathrm{InSec}^{\mathrm{EU-naCMA}}(\mathrm{Dss}(1^n), t)$ *of all possibly probabilistic adversaries* $\mathcal{A}$ *running in time* $\leq t$ *is negligible in* n*:*

$$\mathrm{InSec}^{\mathrm{EU-naCMA}}(\mathrm{Dss}(1^n); t) \overset{\mathrm{def}}{=} \max_{\mathcal{A}} \left\{ \mathrm{Succ}^{\mathsf{EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A}) \right\} = \mathrm{negl}(n).$$

2.5 Estimated security

In this section we collect bounds on the complexity of generic attacks against the described properties. A (tweakable) hash function **Th** is commonly considered a good function if there are no attacks known for any security property that perform better against **Th** than against a random function. Table 1 summarizes the current situation.

Table 1: Success probability of generic attacks – In the "Success probability" column we give the bound for a quantum adversary $\mathcal{A}$ that makes q quantum queries to the function and p classical queries to the challenge oracle. The security parameter n is the output length of $\mathbf{Th}$. We use $X = \sum_{\gamma} \left(1 - \left(1 - \frac{1}{t}\right)^{\gamma}\right)^{k} \binom{p}{\gamma} \left(1 - \frac{1}{2^h}\right)^{p-\gamma} \frac{1}{2^{h\gamma}}$.

Property	Success probability	Status
W-SM-TCR	$\Theta((q+1)^2/2^n)$	proven ([BHK+19, HRS16])
SM-DSPR	$\Theta((q+1)^2/2^n)$	conjecture ([BHK+19])
W-SM-PRE	$\Theta((q+1)^2/2^n)$	conjecture ([BH19a, BHK+19])
PRF	$\Theta((q+1)^2/2^n)$	conjecture ([HRS16])
W-SM-UD	$\mathcal{O}((q+1)^2/2^n)$	conjecture (this work / [DSS05])
ITSR	$\Theta((q+1)^2 \cdot X)$	conjecture ([BHK+19])

The success probability of generic attacks against W-SM-TCR was analyzed in [BHK+19]. Assuming that $\mathbf{Th}$ behaves like a random function a reduction to an average-case search problem was given. A generic attack using Grover search against plain TCR is given in [HRS16]. That attack is also applicable against W-SM-TCR – as it simply runs a second preimage search when all information is available – and has a success probability matching the proven bound.

A conjecture for the success probability against SM-DSPR was also given in [BHK+19]. There is a proof for single target DSPR property in [BH19a] that gives the following bound: $O((q+1)^2/2^n)$. A non-tight proof for a multi-target version is also analyzed in [BH19a]. The best attack against DSPR for now is a second-preimage search that gives the same bound for the multi-target case.

For PRE and also multi-function, multi-target PRE of a hash function h, a bound of $\mathrm{Succ}^{\mathrm{PRE}}_{h,p}(\mathcal{A}) = \Theta((q+1)^2/2^n)$ is given in [HRS16]. The bound is proven for h that are compressing by at least a factor 2 in the message length and it is conjectured that it also applies for length preserving hash functions, i.e., functions that map n-bit messages to n-bit outputs, ignoring additional inputs like function keys or tweaks. This is exactly the case that we are interested in. Another way to support the conjecture is a tight bound using W-SM-TCR and SM-DSPR ($\mathrm{Succ}^{\mathrm{W-SM-PRE}}_{\mathbf{Th},p}(\mathcal{A}) \leq 3 \cdot \mathrm{Succ}^{\mathrm{W-SM-TCR}}_{\mathbf{Th},p}(\mathcal{A}) + \mathrm{Adv}^{\mathrm{SM-DSPR}}_{\mathbf{Th},p}(\mathcal{A})$) given in [BHK+19, BH19a]. With this we can derive the same bound of $\mathrm{Succ}^{\mathrm{W-SM-PRE}}_{\mathbf{Th},p}(\mathcal{A}) = \Theta((q+1)^2/2^n)$. Also in this case it is a conjecture as the bound for SM-DSPR is only conjectured for now.

Exhaustive search on an unstructured space is traditionally considered to be the best known attack for the PRF property. Considering this as a preimage search for a random function and using the results from [HRS16] we obtain the stated bound. We note it as conjectured as we are not aware of a formal proof for this result.

The notion of undetectability was introduced in [DSS05]. In that work, the authors give a bound for one target undetectability considering classical adversaries as $\mathcal{O}(q/2^n)$. The notion was not used in more recent works and we are not aware of a proven bound against quantum adversaries. We conjecture a bound of $\mathcal{O}(q^2/2^n)$ for quantum adversaries. The reasoning behind this is that as long as the adversary does not know if a target (likely) has a preimage under **Th**, it cannot do better than guessing. This suggests the same bound as for W-SM-PRE.

For all notions we conjecture that the bounds are exactly the same for the case of collections. The reason is that for a random tweakable function, every tweak is related to an independent random function. Hence, giving access to those does not give any information about the targets to the adversary. This is also reflected in the reductions that we know so far. In these, the function for a tweak that is not used for a challenge is simulate by an independent random function and we can give access to this function in parallel to the challenge oracle as we do not touch it in the reduction.

3 WOTS-TW

SPHINCS$^+$ [BHK$^+$19] developed its own variant of the Winternitz OTS. However, the authors never explicitly defined that variant. As the flaw in the SPHINCS$^+$ security proof was in the proof for their WOTS scheme, we give a separate description of the scheme in this section. As the distinguishing feature of this variant is the use of tweakable hash functions, we call it WOTS-TW.

3.1 Parameters

WOTS-TW uses several parameters. The main security parameter is $n \in \mathbb{N}$. m is the message length, which we sign. In case of SPHINCS$^+$ $m = n$. The tweak space $\mathcal{T}$ must be at least of size lw. The size of tweak space should be bigger if we use several instances of WOTS-TW in a bigger construction such as SPHINCS$^+$ so we can use a different tweak for each hash function call. Here $w \in \mathbb{N}$ is a so-called Winternitz parameter, which determines a base of the representation that is used in the scheme, and l is defined through the following constants:

$$l_1 = \left\lceil \frac{m}{\log(w)} \right\rceil, \quad l_2 = \left\lfloor \frac{\log(l_1(w-1))}{\log(w)} \right\rfloor + 1, \quad l = l_1 + l_2.$$

We also need a pseudorandom function $\mathbf{PRF} : \{0,1\}^n \times \mathcal{T} \to \{0,1\}^n$, and a tweakable hash function $\mathbf{Th} : \{0,1\}^n \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$.

3.2 Addressing scheme

For the tweakable hash functions to guarantee security, they have to be called with different tweaks. This is achieved using what was called an addressing scheme in SPHINCS$^+$. Such addressing scheme assigns a unique address to every tweakable hash function call in the scheme and the address space is part of the tweak space such that addresses can be used as tweaks. We do not specify a concrete addressing scheme in this work (see the SPHINCS$^+$ specification [ABB+20] for an example). Abstractly, we achieve unique addresses the following way. A Winternitz key pair defines a structure of l hash chains, each of which makes $w-1$ calls to the tweakable hash function. For a unique addressing scheme, one may use any injective function that takes as input $i \in [0, l-1]$, $j \in [0, w-2]$, and possibly a prefix, and maps into the address space. The prefix is necessary to ensure uniqueness if many instances of WOTS-TW are used in a single construction. We will use **ADRS** to denote that prefix. The tweak associated with the j-th function call in the i-th chain is then defined as the output of this function on input i, j (and a possible prefix) and denoted as $T_{i,j}$.

3.3 WOTS-TW scheme

The main difference between WOTS variants is in the way they do hashing. Previously, the distinction was made in the definition of the so called chaining function that describes how the hash chains are computed. For WOTS-TW this distinction is further shifted into the construction of the tweakable hash function **Th**. The chaining function then looks as follows:

Chaining function $c^{j,k}(x, i, \mathrm{Seed})$: The chaining function takes as inputs a message $x \in \{0,1\}^n$, iteration counter $k \in \mathbb{N}$, start index $j \in \mathbb{N}$, chain index i, and public parameters Seed (the name comes from a specific construction of a tweakable hash function that uses the public parameters as seed for a PRG). The chaining function then works the following way. In case $k \leq 0$, c returns x, i.e., $c^{j,0}(x, i, \mathrm{Seed}) = x$. For $k > 0$ we define c recursively as

$$c^{j,k}(x, i, \mathrm{Seed}) = \mathbf{Th}(\mathrm{Seed}, T_{i,j+k-1}, c^{j,k-1}(x, i, \mathrm{Seed}))\,.$$

If we consider several instances of WOTS-TW than we will use $c^{j,k}_{\mathbf{ADRS}}(x, i, \mathrm{Seed})$ to denote that tweaks that are used to construct the chain have **ADRS** as a prefix. With this chaining function, we can describe the algorithms of WOTS-TW.

Key Generation Algorithm $(\mathsf{SK}, \mathsf{PK}) \leftarrow$ WOTS.kg$(\mathcal{C}; \mathcal{S})$: The key generation algorithm optionally takes as input context information $\mathcal{C} = (\mathrm{Seed}, \mathbf{ADRS})$, consisting of a public seed $\mathrm{Seed} \in \{0,1\}^n$ and a global address **ADRS**, as well as randomness $\mathcal{S} \in \{0,1\}^n$ which we call the secret seed. These inputs are meant for the use in more complex protocols. If they are not provided, key generation randomly samples the seeds and sets **ADRS** to 0. The key generation algorithm then computes the internal secret key $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_l)$ as $sk_i \leftarrow \mathbf{PRF}(\mathcal{S}, T_{i,0}))$, i.e., the $l \cdot n$ bit secret key elements are derived form the secret key seed using addresses. The element of the public key pk is computed as

$$\mathsf{pk} = (\mathsf{pk}_1, \dots, \mathsf{pk}_l) = (c^{0,w-1}(\mathsf{sk}_1, 1, \mathrm{Seed}), \dots, c^{0,w-1}(\mathsf{sk}_l, l, \mathrm{Seed}))$$

The key generation algorithm returns $\mathsf{SK} = (\mathcal{S}, \mathcal{C})$ and $\mathsf{PK} = (\mathsf{pk}, \mathcal{C})$ Note that we can compute sk and pk from SK.

Signature Algorithm $\boldsymbol{\sigma} \leftarrow$ WOTS.sign($\boldsymbol{M}, \mathsf{SK}$): On input of an m-bit message M, and the secret key $\mathsf{SK} = (\mathcal{S}, \mathcal{C})$, the signature algorithm first computes a base w representation of $M : M = (M_1, \dots, M_{l_1})$, $M_i \in \{0, \dots, w-1\}$. That is, M is treated as the binary representation of a natural number x and then the w-ary representation of x is computed. Next it computes the checksum $C = \sum_{i=1}^{l}(w - 1 - M_i)$ and its base w representation $C = (C_1, \dots, C_{l_2})$. We set $B = (b_1, \dots, b_l) = M||C$, the concatenation of the base w representations of M and C. Then the internal secret key is regenerated using $\mathsf{sk}_i \leftarrow \mathbf{PRF}(\mathcal{S}, T_{i,0})$ the same way as during key generation. The signature is computed as

$$\sigma = (\sigma_1, \dots, \sigma_l) = (c^{0,b_1}(\mathsf{sk}_1, 1, \mathrm{Seed}), \dots, c^{0,b_l}(\mathsf{sk}_l, l, \mathrm{Seed}))$$

Verification Algorithm ($\{0,1\} \leftarrow$ WOTS.vf($\boldsymbol{M}, \boldsymbol{\sigma}, \mathsf{PK}$)): On input of m-bit message M, a signature σ, and public key $\mathsf{PK} = (\mathsf{pk}, \mathcal{C})$, the verification algorithm first computes the $b_i, 1 \leq i \leq l$ as described above. Then it checks if

$$\mathsf{pk} \stackrel{?}{=} \mathsf{pk}' = (\mathsf{pk}'_1, \dots, \mathsf{pk}'_l) = (c^{b_1, w-1-b_1}(\sigma_1, 1, \mathrm{Seed}), \dots, c^{b_l, w-1-b_l}(\sigma_l, l, \mathrm{Seed}))\,.$$

On the equality the algorithm outputs true and false otherwise.

4 Security of WOTS-TW

Now we will reduce the security of WOTS-TW to the security properties of the tweakable hash function **Th** and the pseudorandom function family **PRF**. To do so we will give a standard game hopping proof. Intuitively the proof goes through the following steps.

– First, we replace the inner secret key elements that are usually generated using **PRF** by uniformly random values. The two cases must be computationally indistinguishable if **PRF** is indeed pseudorandom.
– Next we replace the blocks in the chains that become part of the signature by the hash of random values. We need this so that we can later place preimage challenges at these positions of the chain. Here it is important to note that preimage challenges are exactly such hashes of random domain elements and not random co-domain elements. To argue that these two cases are indistinguishable, we need a hybrid argument since for most chains we replace the outcome of several iterations of hashing with a random value.
– Next we show that breaking the EU-naCMA property of our scheme in this final case will either allow us to extract a target-collision or a preimage for a given challenge.

Theorem 1. *Let n, $w \in \mathbb{N}$ and $w = poly(n)$. Let* $\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ *be a* W-SM-TCR*,* W-SM-PRE*, and* W-SM-UD *function. Let* $\mathbf{PRF} : \mathcal{S} \times \mathcal{T} \to \{0,1\}^n$ *be a pseudorandom function. Then the insecurity of the WOTS-TW scheme against EU-nCMA attack is bounded by*

$$\begin{aligned}\mathrm{InSec}^{EU\text{-}nCMA}(WOTS - TW(1^n, w); t, 1) <& \\ &\mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l) + \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw) + \\ &\quad \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l) + w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)\end{aligned}$$

with $\widetilde{t} = t + lw$*, where time is given in number of* $\mathbf{Th}$ *evaluations.*

Proof. First consider the following two games: GAME.1 is the original EU-nCMA game and GAME.2 is the same as GAME.1 but instead of using pseudorandom elements from $\mathbf{PRF}$ a truly random function $\mathbf{RF} : \mathcal{S} \times \mathcal{T} \to \{0,1\}^n$ is used. We claim that the difference in the success probability of $\mathcal{A}$ playing these games must be bound by $\mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l)$.

Next we consider GAME.3 which is the same as GAME.2 but to answer the message signing request we build the signature from nodes that are computed applying $\mathbf{Th}$ only once instead of b_i times (except if $b_i = 0$, then we return a random value as in the previous game). The public key is constructed from that signature by finishing the chain according to the usual algorithm. We will detail the process in the proof below. We claim that the difference in the success probability of $\mathcal{A}$ playing these games must be bound by $w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)$.

Afterwards, we consider GAME.4, which differs from GAME.3 in that we are considering the game lost if an adversary outputs a valid forgery (M', σ') where there exists an i such that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$. We claim that the difference in the success probability of $\mathcal{A}$ playing these games must be bound by $\mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw)$.

If we now consider how $\mathcal{A}$ can win in GAME.4 there is just we viable case left. Namely, this is the case where the values that get computed from the forgery during verification fully agree with those values that are computed during the verification of the signature. As the checksum ensures that there is at least we index for the forgery that is smaller than the respective index of the signature, this means that we can use an $\mathcal{A}$ that wins in GAME.4 to find preimages. We claim that the success probability of the adversary $\mathcal{A}$ in GAME.4 must be bounded by $\mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l)$.

So we get the following claims:

Claim 1. $|\mathrm{Succ}^{\mathrm{GAME.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l)$

Claim 2. $|\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)$

Claim 3. $|\mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.4}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw)$

Claim 4. $\mathrm{Succ}^{\mathrm{GAME.4}}(\mathcal{A}) \leq \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l)$

The remainder of the proof consists of proving these claims. We then combine the bounds from the claims to obtain the bound of the theorem.

Proof of Claim 1

Claim 1. $|\mathrm{Succ}^{\mathrm{GAME.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, l)$

Proof. Our argument proceeds in two steps. First, we replace **PRF** in GAME.1 by the oracle provided by the PRF game and output 1 whenever $\mathcal{A}$ succeeds. If the oracle is the real **PRF** function keyed with a random secret key, the view of $\mathcal{A}$ is identical to that in GAME.1. If the oracle is the truly random function the argument is a bit more involved. In this case, it is important to note that $\mathcal{A}$ never gets direct access to the oracle but only receives outputs of the oracle. The inputs on which the oracle is queried to obtain these outputs are all unique. Hence, the outputs are uniformly random values. Therefore, the view of $\mathcal{A}$ in this case is exactly that of GAME.2. Consequently, the difference of the probabilities that the reduction outputs we in either of the two cases (which is the PRF distinguishing advantage) is exactly the difference of the success probabilities of $\mathcal{A}$ in the two games.

Proof of Claim 2 We first give a more detailed description of GAME.3. In the EU-nCMA game the adversary $\mathcal{A}$ asks to sign a message M without knowing the public key. This message M gets encoded as $B = b_1, \ldots, b_l$. In GAME.3 to answer the query we will perform the following operations. First we generate l values uniformly at random: $u_i \xleftarrow{\$} \{0,1\}^n,\ i \in \{1, \ldots, l\}$. Next we answer the signing query with a signature $\sigma = (\sigma_1, \ldots, \sigma_l)$, where $\sigma_i = \mathbf{Th}(\mathrm{Seed}, T_{i,b_i-1}, u_i)$ if $b_i > 0$ and $\sigma_i = u_i$ if $b_i = 0$. Then the public key is constructed as

$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_l) = (c^{b_1, w-1-b_1}(\sigma_1, 1, \mathrm{Seed}), \ldots, c^{b_l, w-1-b_l}(\sigma_l, l, \mathrm{Seed}))\,, \quad (1)$$

and public key and signature are returned to the adversary. The reason we consider this game is that to bound the final success probability in GAME.4 we will have a reduction replace the u_i with W-SM-PRE challenges. The resulting signatures have exactly the same distribution as the ones we get here. To show that this cannot change the adversary's success probability in a significant way, we now prove the following claim.

Claim 2. $|\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, l)$

Proof. Consider the following scenario. Let the adversary's query be M. During the signing algorithm M is encoded as $B = \{b_1, \ldots, b_l\}$. Consider two distributions $D_0 = \{\xi_1, \ldots, \xi_l\}$, where $\xi_i \xleftarrow{\$} \{0,1\}^n,\ i \in [1, l]$ and $D_{Kg} = \{y_1, \ldots, y_l\}$, where $y_i = c^{0, b_i-1}(\xi_i, i, \mathrm{Seed}),\ \xi_i \xleftarrow{\$} \{0,1\}^n,\ i \in [1, l]$. Samples from the first distribution are just random values, and the samples from D_{Kg} are distributed the same way as the $(b_i - 1)$-th values of valid WOTS-TW chains. Assume we play a game where we get access to an oracle $\mathcal{O}_\phi$ that on input B returns $\phi = \{\phi_1, \ldots, \phi_l\}$, either initialized with a sample from D_0 or with a sample from

Algorithm 1: $\mathcal{M}^{\mathcal{A}}_{2-3}$

```
   Input  : Access to a distribution oracle $\mathcal{O}_\phi$ and forger $\mathcal{A}$
   Output: 0 or 1.
1  Start $\mathcal{A}$ to obtain query with a message $M$.
2  Encode $M$ as $B = b_1, \ldots, b_l$ as in signature algorithm.
3  Call $\mathcal{O}_\phi(B)$ to obtain sample $\phi$
4  Construct the signature $\sigma$ doing we chain step on each sample and compute
     the public key from the signature:
5  for $1 \leq i \leq l$ do
6      $\sigma_i = c^{b_i-1,1}(\phi_i, i, \mathrm{Seed})$
7      $\mathsf{pk}_i = c^{b_i, w-1-b_i}(\sigma_i, i, \mathrm{Seed})$
8  Send $\mathsf{PK} = (\mathsf{pk}, \mathrm{Seed})$ and $\sigma$ to $\mathcal{A}$.
9  if $\mathcal{A}$ returns a valid forgery $(M', \sigma')$ then
10     return 1
11 else
12     return 0
```

D_{Kg}. Each case occurs with probability 1/2. Then we can construct an algorithm $\mathcal{M}^{\mathcal{A}}_{2-3}$ as in Algorithm 1 that can distinguish these two cases using a forger $\mathcal{A}$.

Let us consider the behavior of $\mathcal{M}^{\mathcal{A}}_{2-3}$ when $\mathcal{O}_\phi$ samples from $\mathcal{D}_{Kg}$. In this case all the elements in the chains are distributed the same as in GAME.2. The probability that $\mathcal{M}^{\mathcal{A}}_{2-3}$ outputs 1 is the same as the success probability of the adversary in GAME.2.

If ϕ instead is from $\mathcal{D}_{\mathcal{U}}$, then the distribution of the elements in the chains is the same as in GAME.3. Hence the probability that $\mathcal{M}^{\mathcal{A}}_{\mathcal{UD}}$ outputs 1 is the same as the success probability of the adversary in GAME.3.

By definition, the advantage of $\mathcal{M}^{\mathcal{A}}_{2-3}$ in distinguishing $\mathcal{D}_{\mathcal{U}}$ from $\mathcal{D}_{Kg}$ is hence given by

$$\mathrm{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}^{\mathcal{A}}_{2-3}) = |\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| \qquad (2)$$

The remaining step is to derive an upper bound for $\mathrm{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}^{\mathcal{A}}_{\mathrm{UD}})$ using the insecurity of the W-SM-UD property. For this purpose we use a hybrid argument.

Let $b_{\max} = \max\{b_1, \ldots, b_l\}$ be the maximum of the values in the message encoding of M. Let H_k be the distribution obtained by computing the values in ϕ as $\phi_i = c^{k, b_i-1-k}(\xi_i, i, Seed)$, $\xi_i \xleftarrow{\$} \{0,1\}^n$. Then $H_0 = D_{Kg}$ and $H_{b_{\max}-1} = D_0$ (Note that the chaining function returns the identity when asked to do a negative amount of steps). As $\mathcal{M}^{\mathcal{A}}_{2-3}$ distinguishes the extreme cases, by a hybrid argument there are two consecutive hybrids H_j and H_{j+1} that can be distinguished with probability $\geq \mathrm{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}^{\mathcal{A}}_{2-3})/(b_{\max} - 1)$.

To bound the success probability of an adversary in distinguishing two such consecutive hybrids, we build a second reduction $\mathcal{M}^{\mathcal{B}}_{\mathrm{UD}}$ that uses $\mathcal{B} = \mathcal{M}^{\mathcal{A}}_{2-3}$ to break W-SM-UD. For this purpose, $\mathcal{M}^{\mathcal{B}}_{\mathrm{UD}}$ simulates $\mathcal{O}_\phi$. To answer a query for

$B = b_1, \dots, b_l$, $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ plays in the W-SM-UD game, interacting with the W-SM-UD oracle $\mathcal{O}_{\mathrm{UD}}(\cdot, b)$ to construct hybrid H_{j+b}, depending on the secret bit b of the oracle. To do so $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ makes queries to $\mathcal{O}_{\mathrm{UD}}$ with tweaks $\{T_{1,j}, \dots, T_{l,j}\}$. Then, depending on b, the responses ψ of $\mathcal{O}_{\mathrm{UD}}$ are either l random values or $\psi = (c^{j,1}(\xi_1, 1, Seed), \dots, c^{j,1}(\xi_l, l, Seed),\ \xi_i \xleftarrow{\$} \{0,1\}^n,\ i \in [1,l])$. After that $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ requests Seed from the W-SM-UD challenger. Next, $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ applies the hash chain to the oracle responses ψ to compute samples

$$\phi_i = \begin{cases} c^{j+1, b_i - 1 - (j+1)}(\psi_i, i, Seed), & \text{if } j < b_i - 1 \\ \xi_i \xleftarrow{\$} \{0,1\}^n, & \text{otherwise}, \end{cases}$$

and returns it to $\mathcal{M}_{2-3}^{\mathcal{A}}$. $\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}$ returns whatever $\mathcal{M}_{2-3}^{\mathcal{A}}$ returns.

If ψ consisted of random values the distribution was H_{j+1}, otherwise H_j. Consequently, the advantage of distinguishing any two hybrids must be bound by $\mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, l)$.

Putting things together, we see that $b_{\max} \leq w$ for any message M. Hence we get

$$\begin{aligned} |\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| &= \mathrm{Adv}_{\mathcal{D}_{\mathcal{U}}, \mathcal{D}_{Kg}}(\mathcal{M}_{2-3}^{\mathcal{A}}) \\ &\leq w \cdot \mathrm{Succ}_{\mathbf{Th},l}^{\mathrm{W-SM-UD}}(\mathcal{M}_{\mathrm{UD}}^{\mathcal{B}}) \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, l) \end{aligned}$$

which concludes the proof.

Proof of Claim 3 Recall that GAME.4 differs from GAME.3 in that we are considering the game lost if an adversary outputs a valid forgery (M', σ') where there exist such i that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$. So the difference in success probability is exactly the probability that $\mathcal{A}$ outputs a valid forgery and there exists an i such that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$. We will now prove Claim 3 which claims the following bound on this probability:

Claim 3. $|\mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.4}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, lw)$

Proof. To prove the claim we construct an algorithm $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ that reduces W-SM-TCR of **Th** to the task of forging a signature that fulfills the above condition. The algorithm is based on the following idea. $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ simulates GAME.4. In Game.4 the adversary sends a query to sign a message M. To answer this query and compute the public key, $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ interacts with the W-SM-TCR oracle. This way, $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ obtains target-collision challenges corresponding to the nodes in the signature and all intermediate values in the chain computations made to compute the public key. Then $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ requests the public parameters P from the W-SM-TCR challenger. We set the public seed Seed of WOTS-TW equal to P. We answer on the query with the constructed signature and public key. Per assumption there now exists i such that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \mathrm{Seed}) \neq \sigma_i$, hence by a pigeon hole argument there must be a collision on the way to public key element. $\mathcal{M}_{\mathrm{TCR}}^{\mathcal{A}}$ extracts this collision and returns it. Algorithm 2 gives a

Algorithm 2: $\mathcal{M}^{\mathcal{A}}_{\text{TCR}}$

```
Input  : Security parameter $n$, oracle access to W-SM-TCR challenger $C$ and
         EU-nCMA forger $\mathcal{A}$.
Output: A pair $(j, M)$ or fail.
1  begin Challenge placement
2      Start $\mathcal{A}$ to obtain query with a message $M$.
3      Encode $M$ as $B = b_1, \dots, b_l$ as in signature algorithm.
4      for $i \in \{1, \dots, l\}$ do
5          if $b_i = 0$ then
6              Set $\sigma_i \xleftarrow{\$} \{0,1\}^n$.
7          else
8              Sample $\xi_i \xleftarrow{\$} \{0,1\}^n$,
9              Query $C$ for W-SM-TCR challenge with inputs $\xi_i, T_{1,b_i-1}$.
10             Store answer as $\sigma_i$. // i.e., $\sigma_i = \mathbf{Th}(P, T_{i,b_i-1}, \xi_i)$
11         Compute public key element $\mathsf{pk}_i = c^{b_i, w-1-b_i}(\sigma_i, i, \cdot)$ as in the
               verification algorithm but using the W-SM-TCR challenge oracle
               provided by $C$ in place of $\mathbf{Th}$. // That is why no Seed is
               needed
12     Get public parameters $P$ from the challenger and set Seed $= P$.
13     Set signature $\sigma = (\sigma_1, \dots, \sigma_l)$ and $\mathsf{pk} = (\mathsf{pk}_1, \dots, \mathsf{pk}_l)$.
14 begin Obtaining the result
15     Return $\sigma$ and $\mathsf{PK} = (\mathsf{pk}, \text{Seed})$ to the adversary $\mathcal{A}$.
16     if The adversary returns a valid forgery $(M', \sigma')$ then
17         Encode $M'$ as $B' = (b'_1, \dots, b'_l)$ according to sign.
18         if $\exists\ i$ such that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \text{Seed}) \neq \sigma_i$ then
19             Let $j$ be the smallest integer such that the chains collide:
                 $c^{b_i, j}(y_i, i, \text{Seed}) = c^{b'_i, j}(\sigma'_i, i, \text{Seed}))$.
20             return W-SM-TCR solotion $(i, c^{b'_i, (j-1)}(\sigma'_i, i, \text{Seed}))$
21         else
22             return fail
23     else
24         return fail
```

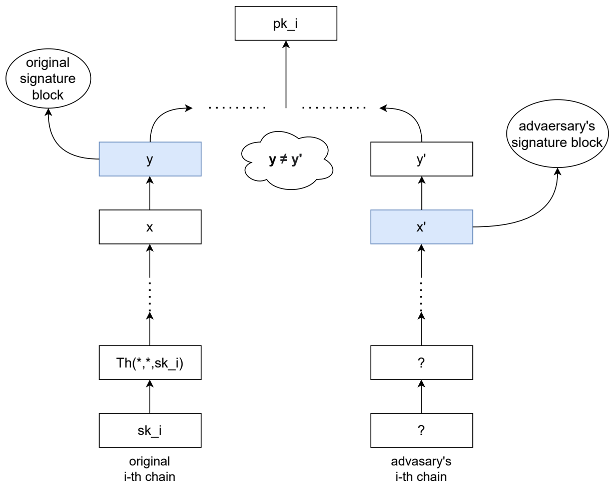

Fig. 1: Example of a case in claim 3

detailed description of $\mathcal{M}_{\text{TCR}}^{\mathcal{A}}$ in pseudocode. The algorithm is broken into two logically separated parts: Challenge placement and obtaining the result.

Here we detail which W-SM-TCR challenges we create per chain in line 11 of Algorithm 2. Assume we have σ_i at position b_i. Then the first query will be (T_{i,b_i}, σ_i). Lets denote the answer for that query as c_1. The next query will be (T_{i,b_i+1}, c_1). We denote the answer for that query as c_2. In general we will make queries of the form (T_{i,b_i+k}, c_k). And the answers for that queries we denote as c_{k+1}. We make queries until we get c_{w-1-b_i}. We set pk_i to be c_{w-1-b_i}.

As we are set to bound the probability of those cases where the adversary outputs a valid forgery and there exists such i that $b'_i < b_i$ and $c^{(b'_i, b_i - b'_i)}(\sigma'_i, i, \text{Seed}) \neq \sigma_i$, $\mathcal{M}_{\text{TCR}}^{\mathcal{A}}$ never runs into the fail cases in lines 22 and 24. Moreover, the distribution of inputs to $\mathcal{A}$ when run by $\mathcal{M}_{\text{TCR}}^{\mathcal{A}}$ is identical to that in GAME.4. Therefore, $\mathcal{M}_{\text{TCR}}^{\mathcal{A}}$ returns a target-collision with probability $|\text{Succ}^{\text{GAME.3}}(\mathcal{A}) - \text{Succ}^{\text{GAME.4}}(\mathcal{A})|$ which concludes the proof.

Proof of Claim 4 It remains to prove the last claim. Consider a forgery σ' and the positions b'_i of the σ' elements. There must exist a j such that $b'_j < b_j$ by the properties of the checksum. Remember that in GAME.4, the case where $c^{(b'_j, b_j - b'_j)}(\sigma'_j, j, \text{Seed}) \neq \sigma_j$ is excluded for all such j. Hence, it must hold for these j that $c^{(b'_j, b_j - b'_j)}(\sigma'_j, j, \text{Seed}) = \sigma_j$. Therefore, we can use $\mathcal{A}$ to compute a preimage of σ_j. We use this to prove Claim 4.

Claim 4. $\text{Succ}^{\text{GAME.4}}(\mathcal{A}) \leq \text{InSec}^{\text{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, l)$

Proof. As for the previous claim, we construct an algorithm $\mathcal{M}_{\text{PRE}}^{\mathcal{A}}$ that uses a forger in GAME.4 to solve a W-SM-PRE challenge. In the beginning, $\mathcal{M}_{\text{PRE}}^{\mathcal{A}}$ receives a query to sign a message M from the adversary $\mathcal{A}$ and encodes it

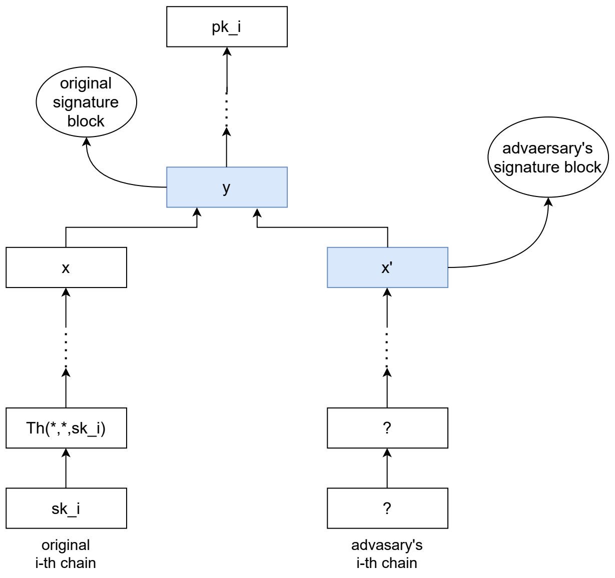

Fig. 2: Example of a case in Claim 4.

into b_i's. To answer the query $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ interacts with the W-SM-PRE challenger to receive preimage challenges y_i for tweaks that make the challenges fit into positions b_i. That way, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ can use the challenges as signature values $\sigma_i = y_i$. Then $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ asks the W-SM-PRE challenger to return public parameters P. Given P, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ can construct the public key using the recomputation method used in the signature verification algorithm. $\mathcal{M}^{\mathcal{A}}$ sets the public seed Seed of WOTS-TW to be P and returns the constructed signature and public key to $\mathcal{A}$. When $\mathcal{A}$ returns a valid forgery, this forgery must contain a signature value σ_j with index j such that $b'_j < b_j$ and $c^{b'_j,(b_j-b'_j)}(\sigma'_j, j, \text{Seed}) = \sigma_j$ per definition of the game. $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ returns preimage $(j, c^{b'_j,(b_j-b'_j-1)}(\sigma'_j, j, \text{Seed}))$. A pseudocode version of $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ is given as Algorithm 3. The algorithm is broken into two logically separated parts: Challenge placement and obtaining the result.

Due to the properties of GAME.4, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ succeeds whenever $\mathcal{A}$ succeeds, as the failure case in line 19 never occurs when $\mathcal{A}$ succeeds. Moreover, the distribution of the inputs to $\mathcal{A}$ when run by $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ is identical to that in GAME.4 (this was ensured in the game hop to GAME.3). Therefore, $\mathcal{M}^{\mathcal{A}}_{\text{PRE}}$ returns preimages with probability $\text{Succ}^{\text{GAME.4}}(\mathcal{A})$ which proves the claim.

5 Extension to multiple instances with same public seed

One-time signatures are often used in more complex constructions. Indeed, WOTS-TW was developed as part of SPHINCS$^+$. The distinguishing feature of this setting is that many WOTS-TW instances are used within one instance of the

Algorithm 3: $\mathcal{M}_{\text{PRE}}^{\mathcal{A}}$

```
Input  : Security parameter $n$, access to W-SM-PRE challenger $C$ and forger $\mathcal{A}$.
Output: A pair $(j, M)$ or fail.
1  begin Challenge placement
2      Run $\mathcal{A}$ to receive initial query for a signature on message $M$.
3      Encode $M$ as $B = b_1, \dots, b_l$ following the steps in the signature algorithm.
4      for $1 \leq i \leq l$ do
5          if $b_i > 0$ then
6              Query $C$ for preimage challenge $y_i$ with tweak $T_{1,b_i-1}$.
                   // $y_i = \mathbf{Th}(P, T_{i,b_i-1}, \xi_i)$
7          else
8              $y_i \xleftarrow{\$} \{0,1\}^n$.
9          Set $\sigma_i = y_i$.
10     Get the seed $P$ from $C$ and set Seed $= P$.
11     Compute public key $\mathsf{pk} = (\mathsf{pk}_1, \dots \mathsf{pk}_l)$, as $\mathsf{pk}_i = c^{w-1-b_i}(y_i, i, \text{Seed})$.
12 begin Obtaining the result
13     Return $\sigma$ and $\mathsf{PK} = (\mathsf{pk}, P)$ to $\mathcal{A}$.
14     if $\mathcal{A}$ returns a valid forgery $(M', \sigma')$ then
15         Compute $B' = (b'_1, \dots, b'_l)$ encoding $M'$
16         if $\exists 1 \leq j \leq l$ such that $b'_j < b_j$ and $c^{(b'_j, b_j - b'_j)}(\sigma'_j, j, \text{Seed}) = \sigma_j$ then
17             return W-SM-PRE solution $(j, c^{b'_j, (b_j - b'_j - 1)}(\sigma'_j, j, \text{Seed}))$
18         else
19             return fail
20     else
21         return fail
```

construction. In this section we will show that we can reduce the security of multiple WOTS-TW instances to the same multi-target security properties used for a single instance. While, obviously, the number of targets increases, we argued in Section 2.5 that the complexity of generic attacks is not influenced by the number of targets for these notions. Hence, there is no decrease in security to be expected when using multiple instances. We will show that this even works when the same public seed is used for all instances, as long as different prefixes are used for the tweaks.

In SPHINCS-like constructions WOTS-TW is used to sign the roots of trees which are not controlled by an adversary against the construction but by the signer. More generally, this is the case in many such constructions. Hence we use an extension of the EU-nCMA model from last section to d instances as security model.

We formally define EU-nCMA security for d instances. We introduce a definition for the security of $\mathsf{Dss}(1^n)$, which we call the d-existential unforgeability under non-adaptive chosen message attack (d-EU-naCMA). It is defined using the following experiment. In this experiment a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, A_2)$ is allowed to make signing queries to an oracle $sign(\cdot, \cdot, \mathcal{S}, Seed)$ and query oracle $\mathbf{Th}_\lambda$. The signing oracle takes **ADRS** as a first input an message M as a second input. First it runs $(\mathsf{SK}, \mathsf{PK}) \leftarrow \text{WOTS.kg}(\mathcal{C} = (Seed, \mathbf{ADRS}); \mathcal{S})$. Then it computes $\sigma \leftarrow \text{WOTS.sign}(M; \mathsf{SK})$. Let us denote PK' which is equal to PK but there is no $Seed$ in PK', i.e. $\mathsf{PK}' = (\mathsf{pk}, \mathbf{ADRS})$. The signing oracle return $(\sigma, M, \mathsf{PK}')$ to the adversary. We also restrict $\mathcal{A}$ from querying $\mathbf{Th}_\lambda$ with tweaks for **ADRS**-s that are used in signature queries. We define a function $\mathsf{adrs}(\cdot)$ that takes a tweak as an input and returns **ADRS** of that tweak. We denote the set of queries to signing oracle as $Q = \{(\mathbf{ADRS}_1, M_1), \dots, (\mathbf{ADRS}_d, M_d)\}$ and the set of tweaks that are used to query $\mathbf{Th}_\lambda$ is $T = \{T_1, \dots T_p\}$. We are concerned with one-time signatures, so the number of allowed signing queries for each **ADRS** is restricted to 1.

Experiment $\mathsf{Exp}^{\mathsf{d-EU-naCMA}}_{\mathsf{Dss}(1^n)}(\mathcal{A})$

- $Seed \xleftarrow{\$} \{0,1\}^n$
- $\mathcal{S} \xleftarrow{\$} \{0,1\}^n$
- $state \leftarrow \mathcal{A}_1^{sign(\cdot,\cdot,Seed,\mathcal{S}), \mathbf{Th}_\lambda(Seed,\cdot,\cdot)}()$
- $(M^\star, \sigma^\star, j) \leftarrow \mathcal{A}_2(state, Seed),\ j \in [1, d]$
- Return 1 iff $[\text{Vf}(\mathsf{PK}_j, \sigma^\star, M^\star) = 1] \wedge [M^\star \neq M_j] \wedge [\forall \mathbf{ADRS}_i \in Q, \mathbf{ADRS}_i \notin T' = \{\mathsf{adrs}(\mathsf{T_i})\}_{i=1}^p]$.

The following theorem can be proved by generalization of the proof of theorem 1. The main idea behind the proof is the following. First of all we use different tweaks in different instances of WOTS-TW as we use different **ADRS** for each instance. Next point is that we obtain d times more challenges and we separate them in d sets. Each set will be used for appropriate instance of WOTS. Then the proof follows the same path as in theorem 1.

Theorem 2. *Let n, $w \in \mathbb{N}$ and $w = poly(n)$. Let* $\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ *be a* W-SM-TCR*,* W-SM-PRE*,* W-SM-UD *function. Let* $\mathbf{PRF} : \mathcal{S} \times \mathcal{T} \to \{0,1\}^n$ *be a pseudorandom function. Then the insecurity of the WOTS-TW scheme against d-EU-naCMA attack is bounded by*

$$\begin{aligned}\mathrm{InSec}^{\mathrm{d-EU-naCMA}}(\text{WOTS-TW}(1^n, w); t, 1) <& \\ &\mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, d \cdot l) + \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \widetilde{t}, d \cdot lw) + \\ &\quad \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \widetilde{t}, d \cdot l) + w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, d \cdot l) \quad (3)\end{aligned}$$

with $\widetilde{t} = t + d \cdot lw$*, where time is given in number of* $\mathbf{Th}$ *evaluations.*

Proof sketch. Let us give a brief description how the generalization will be obtain. We have the same game hopping as in Theorem 1.

GAME.1 is the original EU-nCMA game and GAME.2 is the same as GAME.1 but instead of using pseudorandom elements from **PRF** a truly random function $\mathbf{RF} : \mathcal{S} \times \mathcal{T} \to \{0,1\}^n$ is used. $|\mathrm{Succ}^{\mathrm{GAME.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A})| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \widetilde{t}, d \cdot l)$ The reasoning here is the same as in Claim 1 in Theorem 1. Note that here inputs on which the oracle in PRF game is queried are all unique due to the unique **ADRS**-s for each instance. Hence, the outputs are uniformly random values as desired.

GAME.3 is different from GAME.2 in that for each signing query we answer with a hash of random value rather than building it with a chaining function. In Claim 2 of Theorem 1 we reduced it to W-SM-UD property by using hybrid argument. Here we need to apply the same reasoning. To obtain needed hybrids in case of d instances we will do the following. Let us denote the transformation of M_i into $b_1, \dots, b_l$ with extra index. So M_i is transferred into $b_{i,1}, \dots, b_{i,l}$. We denote two distributions where $D_{d-Kg} = \{y_{1,1}, \dots, y_{1,l}, \dots, y_{d,1}, \dots, y_{d,l}\}$, where $y_{i,j} = c^{0,b_j-1}_{\mathbf{ADRS}_i}(\xi_{i,j}, j, \mathrm{Seed})$, $\xi_{i,j} \xleftarrow{\$} \{0,1\}^n$, $i \in [1,d]$, $j \in [1,l]$, and $D_{d-0} = \{\xi_{1,1}, \dots, \xi_{1,l}, \dots, \xi_{d,1}, \dots, \xi_{d,l}\}$, where $\xi_{i,j} \xleftarrow{\$} \{0,1\}^n$, $i \in [1,d]$, $j \in [1,l]$. The distinguishing advantage of adversary of those two distributions is exactly the difference of success probability this game hop. To limit this distinguishing advantage we need to build hybrids. We do this is the same manner as in Theorem 1. Let $b_{\max} = \max\{b_{1,1}, \dots, b_{1,l} \dots, b_{d,1}, \dots, b_{d,l}\}$ be the maximum of the values in the message encoding of all M_i. Let H_k be the distribution obtained by computing the values as $c^{k,b_{i,j}-1-k}_{\mathbf{ADRS}_i}(\xi_{i,j}, j, Seed)$, $\xi_{i,j} \xleftarrow{\$} \{0,1\}^n$. One can notice that $H_0 = D_{Kg}$ and $H_{b_{\max}-1} = D_0$. There must be two consecutive hybrids H_γ and $H_{\gamma+1}$ that we can distinguish with probability close to distinguishing advantage. By playing W-SM-UD and interacting with the oracle $\mathcal{O}(\cdot, b)$ we can construct hybrid $H_{\gamma+b}$ this is done in just the same way as in Claim 2 of Theorem 1. Hence we obtain the following bound:

$$|\mathrm{Succ}^{\mathrm{GAME.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathrm{GAME.3}}(\mathcal{A})| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \widetilde{t}, d \cdot l)$$

Notice that in case of one instance we obtained Seed from W-SM-UD challenger that we used to construct the hybrids and obtain WOTS-TW public key. Here instead of using Seed we need to interact with $\mathbf{Th}_\lambda(\text{Seed}.\cdot,\cdot)$ oracle. Only after all of the signing queries are made we will obtain the Seed.

GAME.4 is different from GAME.3 in that we are considering the game lost if an adversary outputs a valid forgery $(M^\star, \sigma^\star, j)$ where there exist such i that $b^\star_{i,j} < b_{i,j}$ and $c^{(b^\star_{i,j}, b_{i,j}-b^\star_{i,j})}_{\mathbf{ADRS}_i}(\sigma^\star_j, j, \text{Seed}) \neq \sigma_j$. To show that $|\text{Succ}^{\text{GAME.3}}(\mathcal{A}) - \text{Succ}^{\text{GAME.4}}(\mathcal{A})| \leq \text{InSec}^{\text{W-SM-TCR}}(\mathbf{Th}; \tilde{t}, d \cdot lw)$ we can build the reduction that works as follows.To answer signature queries query and compute the public key, the reduction interacts with the W-SM-TCR oracle. The difference in case of d instances from one instance is that we will need d times more interactions with W-SM-TCR oracle. The chains that we build and chains obtained from the forged signature are different but lead to the same public key. Hence by a pigeon hole argument there must be a collision on the way to public key element. This collision is a solution for W-SM-TCR challenge. So we proved that

$$|\text{Succ}^{\text{GAME.3}}(\mathcal{A}) - \text{Succ}^{\text{GAME.4}}(\mathcal{A})| \leq \text{InSec}^{\text{W-SM-TCR}}(\mathbf{Th}; \tilde{t}, d \cdot lw)$$

To give a bound on success probability for GAME.4 we use W-SM-PRE property. To answer signing queries we will interact with W-SM-PRE oracle and place challenges obtained from that oracle in place of signatures. To construct public keys of WOTS-TW instances we will behave in the same way as in the undetectability case. By interacting with $\mathbf{Th}_\lambda(\text{Seed}, \cdot, \cdot)$ we can build the chains of WOTS-TW structures. Again there must exist a j such that $b^\star_{i,j} < b_{i,j}$ by the properties of the checksum. And since we excluded the case where $c^{(b^\star_{i,j}, b_{i,j}-b^\star_{i,j})}_{\mathbf{ADRS}_i}(\sigma^\star_j, j, \text{Seed}) \neq \sigma_j$ we can obtain a preimage by computing $c^{(b^\star_{i,j}-1, b_{i,j}-b^\star_{i,j})}_{\mathbf{ADRS}_i}(\sigma^\star_j, j, \text{Seed})$ So we obtained

$$|\text{Succ}^{\text{GAME.4}}(\mathcal{A})| \leq \text{InSec}^{\text{W-SM-PRE}}(\mathbf{Th}; \tilde{t}, d \cdot l)$$

This concludes the sketch of the proof.

6 SPHINCS$^+$

In this section we will recap the SPHINCS$^+$ structure and afterwards give fixes to the original SPHINCS$^+$ proof. To obtain a fixed proof we will utilize the results from Theorem 2.

6.1 Brief description

First we give a brief description of the SPHINCS$^+$ signature scheme. An example of SPHINCS$^+$ structure can be seen on the following picture:

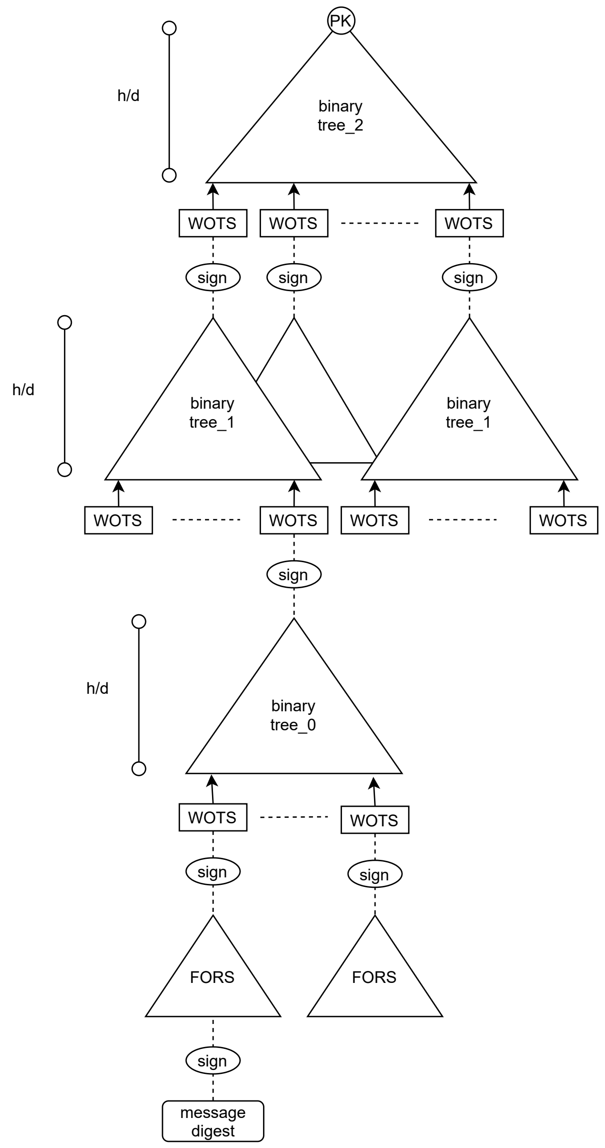

Fig. 3: Example of a SPHINCS$^+$ structure

The public key consists of two n-bit values: a random public seed **PK**.seed and the root of the top tree in the hypertree structure. **PK**.seed is used as a first argument for all of the tweakable hash functions calls. The private key contains two more n-bit values **SK**.seed and **SK**.prf.

Lets describe the main parts of it. First we have to describe addressing scheme. In this scheme, the addresses of certain structures and calls of hash functions will be used. This is for pseudo-random data generation. The address is a 32 byte value. Address coding can be done in any convenient way. An address will be represented with a tweak. Each tweak has a prefix that denotes to which part of SPHINCS$^+$ structure it belongs. We denoted this prefix as **ADRS** in previous sections. So for each hashing in the structure a different tweak will be used.

Then we need to discuss binary trees. In the SPHINCS$^+$ algorithm, binary trees of height γ always have 2^γ leaves. Each leaf $L_i,\ i\ in[2^\gamma - 1]$ is a bit string of length n. Each node of the tree $N_{i,j},\ 0 < j \leq \gamma, 0 \leq i < 2^{\gamma - j}$ is also a bit string of length n. The values of the internal nodes of the tree are calculated as a hash function from the children of that node. A leaf of a binary tree is a hash of elements of public key of a WOTS-TW signature scheme instance.

Binary trees and WOTS-TW signature schemes are used to construct a hypertree structure. WOTS-TW instances are used to sign root of binary trees on lower levels. WOTS-TW instances on the lowest level used to signed the root of a FORS structure. FORS is defined with the following parameters: $k \in \mathbb{N}$, $t = 2^a$. This algorithm can sign messages of length ka-bits.

FORS key pair. The private key of FORS consists of kt pseudorandomly generated n-bit values grouped into k sets of t elements each.

To get the public key k binary hash trees are constructed. The leaves in these trees are k sets (one for each tree) which consist of t values. Thus we get k trees of height a. As roots of k binary trees are calculated they are compressed using a hash function. The resulting value will be the FORS public key.

FORS Signature. A message from ka bits is divided into k lines of a bits. Each of these lines is interpreted as a leaf index corresponding to one of the k trees. The signature consists of these sheets and their authentication paths. The verifier reconstructs the tree roots, compresses them, and verifies them against the public key. If there is a match, it is said that the signature was verified. Otherwise, it is declared invalid.

The last thing to discuss is the way the message digest is calculated. First, we will prepare the pseudo-random value $\mathbf{R}$. It is calculated from $\mathbf{SK}.\mathsf{prf}$ and the message. Also, this value can be non-deterministic, this can be achieved by adding a random value $\mathsf{OptRand}$. Thus $\mathbf{R} = \mathbf{PRF}_{msg}(\mathbf{SK}_{\mathsf{prf}}, \mathsf{OptRand}, M)$. The $\mathbf{R}$ value is part of the signature. Now, using $\mathbf{R}$, we calculate the index of the sheet with which the message will be signed and the hash of the message itself. $(\mathsf{MD}||\mathsf{idx}) = \mathbf{H_{msg}}(\mathbf{R}, \mathbf{PK}.\mathsf{seed}, \mathbf{PK}.\mathsf{root}, M)$.

The signature consists of the randomness $\mathbf{R}$, FORS signature (which corresponds to index idx from $\mathbf{H_{msg}}$) of the message digest, the WOTS-TW signature of the corresponding FORS public key, and a set of authentication paths and WOTS-TW signatures of tree roots. To test this chain, the verifier iteratively reconstructs the public keys and tree roots until it gets the root of the top tree.

Given the brief description of SPHINCS$^+$ lets analyze the modifications of its reduction proof.

6.2 SPHINCS$^+$ proof

In this part we want to update the proof of security of SPHINCS$^+$ framework. The security had several issues which are described in [MK, ABB$^+$20]. One can look up for the description of the SPHINCS$^+$ scheme in [BHK$^+$19].

For the SPHINCS$^+$ construction we will need the following functions:

- $\mathbf{F}$: $\mathcal{P} \times \mathcal{T} \times \{0,1\}^n \rightarrow \{0,1\}^n$;
- $\mathbf{H}$: $\mathcal{P} \times \mathcal{T} \times \{0,1\}^{2n} \rightarrow \{0,1\}^n$;
- $\mathbf{Th}_l$: $\mathcal{P} \times \mathcal{T} \times \{0,1\}^{ln} \rightarrow \{0,1\}^n$;
- PRF: $\{0,1\}^n \times \{0,1\}^{256} \rightarrow \{0,1\}^n$;
- $\mathbf{PRF_{msg}}$: $\{0,1\}^n \times \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^n$;
- $\mathbf{H_{msg}}$: $\{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^m$;

In this section, we prove the following Theorem. Note that $\mathbf{F}$, $\mathbf{H}$, $\mathbf{Th}_l$ are members of a collection of tweakable hash functions with different message length.

Theorem 3. *For parameters n, w, h, d, m, t, k as described in [BHK$^+$19] the following bound can be obtained:*

$$\begin{aligned}
&\mathrm{InSec}^{\mathrm{EU-CMA}}(SPINCS+; \xi, q_s) \leq \\
&\mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}, \xi, q_1) + \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF_{msg}}, \xi, q_s) + \\
&\mathrm{InSec}^{\mathrm{ITSR}}(\mathbf{H_{msg}}, \xi, q_s) + w^2 \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, q_2) + \\
&\mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_3) + \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \xi, q_4) + \\
&3 \cdot \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_5) + \mathrm{InSec}^{\mathrm{SM-DSPR}}(\mathbf{Th}; \xi, q_5)
\end{aligned}$$

where $q_1 < 2^{h+1}(kt + l)$, $q_2 < 2^{h+1} \cdot l$, $q_3 < 2^{h+2}(w \cdot l + 2kt)$, $q_4 < 2^{h+1} \cdot l \cdot w$, $q_5 < 2^h \cdot kt$ and q_s denotes the number of signing queries made by $\mathcal{A}$.

Proof. We want to bound the success probability of a (quantum) adversary $\mathcal{A}$ against the EU-CMA security of SPHINCS$^+$. Towards this end we use the following series of games. We start with GAME.0 which is the EU-CMA experiment for SPHINCS$^+$. Now consider a GAME.1 which is essentially GAME.0 but the experiment makes use of a SPHINCS$^+$ version where all the outputs of PRF, i.e., the WOTS-TW + and FORS secret-key elements, get replaced by truly random values.

Next, consider a game GAME.2, which is the same as GAME.1 but in the signing oracle $\mathbf{PRF_{msg}}(\mathrm{SK.prf}, \cdot)$ is replaced by a truly random function.

Afterwards, we consider GAME.3, which differs from GAME.2 in that we are considering the game lost if an adversary outputs a valid forgery $(\mathrm{M}, \mathrm{SIG})$ where the FORS signature part of SIG contains only secret values which were contained in previous signatures with that FORS key pair obtained by $\mathcal{A}$ via the signing oracle.

Now consider what are the possibilities of the adversary to win the game. The FORS signature in a forgery must include the preimage of a FORS leaf node that was not previously revealed to it. There are two separate cases for that leaf:

1. The FORS leaf is different to the leaf that we would generate for that place.
2. The FORS leaf is the same to the leaf that we would generate for that place;

Lets consider GAME.4 which differs from GAME.3 in that we are considering that the WOTS-TW signatures are made as described in the proof of Claim 2 in Section 4, the game is lost in the first “leaf case” scenario or an adversary outputs a valid forgery (M, SIG) which (implicitly or explicitly) contains a second preimage for an input to **Th** that was part of a signature returned as a signing-query response.

Now lets analyze those games.

GAME.0 - GAME.3 The hops between GAME.0 and GAME.3 are fully presented in the SHINCS+ paper [BHK+19]. The bound for these games are

$$|\mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.0}} - \mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.1}}| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}, \xi, q_1), \tag{4}$$

$$|\mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.1}} - \mathrm{Succ}_{\mathcal{A}}^{\mathrm{GAME.2}}| \leq \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF_{msg}}, \xi, q_s), \tag{5}$$

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.2} - \mathrm{Succ}_{\mathcal{A}}^{GAME.3}| \leq \mathrm{InSec}^{\mathrm{ITSR}}(\mathbf{H_{msg}}, \xi, q_s), \tag{6}$$

where $q_1 < 2^{h+1}(kt + l)$ and q_s denotes the number of signing queries made by $\mathcal{A}$.

GAME.3 - GAME.4 Lets break the hop between GAME.3 and GAME.4 into several steps. First assume GAME.3.1 in which random values are used to create WOTS-TW signatures. This is the case that we discussed in the proof of Claim 2 in Section 4. But in this case we have several message queries as described in previous section. Using $\mathbf{Th}_\lambda$ oracle we first obtain FORS instances. Then we can query W-SM-UD to obtain challenges that will be placed in WOTS-TW instances. Again after placing those challenges we use $\mathbf{Th}_\lambda$ to obtain WOTS-TW public keys and build Merkle trees so we go to the next level of the hypertree. Having the roots of Merkle trees we again interact with W-SM-UD challenger to obtain challenges for this set of WOTS-TW instances. We continue this process until we reach the root of the highest Merkle tree in our hypertree structure. Then we obtain public parameter P and set $\mathbf{PK}.\mathsf{seed} = P$. By the reasoning from previous section we obtain

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.3} - \mathrm{Succ}_{\mathcal{A}}^{GAME.3.1}| \leq w \cdot \mathrm{InSec}^{\mathrm{W-SM-UD}}(\mathbf{Th}; \xi, q_2) \tag{7}$$

where $q_2 < 2^{h+1} \cdot len$.

Next consider game GAME.3.2, which differs from GAME.3.1 in that we are considering the game lost if an adversary outputs a valid forgery (M, SIG) which (implicitly or explicitly) contains a second preimage for an input to **Th** that was part of a signature returned as a signing-query response. By implicitly we here refer to a second preimage which is observed during the verification of the signature.

We can build a reduction $\mathcal{M}^{\mathcal{A}}$ that breaks H-SM-TCR. Here we slightly abuse the notation and we assume that we we obtain W-SM-TCR challenges for different members of collection, i.e. playing against several instances of the tweakable collection at one. We use $\mathbf{Th}_m$ (that differs in m) to compute the whole SPHINCS$^+$ structure. The reduction builds the whole SPHINCS$^+$ structure of a key pair (the key pair plus the whole hypertree including all FORS key pairs and WOTS-TW) during setup using the H-SM-TCR and $\mathbf{Th}_\lambda$ oracles and stores all computed values. Assume the collision occurs in the WOTS-TW instances. Than we deal with this case as was show in the Theorem 2. If the collision occurs not in WOTS-TW instances then we can obtain challenges for those places through interaction with W-SM-TCR oracle for several members. Here the difference from [BHK$^+$19] is only in the way we deal with WOTS-TW.

Thereby it defines all inputs to $\mathbf{Th}_m$ as targets. In total,the reduction $\mathcal{M}^{\mathcal{A}}$ makes $q_3 < 2^{h+2}(w \cdot len + 2kt)$ queries to its oracles. When $\mathcal{M}^{\mathcal{A}}$ is done, it obtains the public parameters from the challenger and puts these into the public key together with the generated root. Then it runs $\mathcal{A}$ with this public key as input. $\mathcal{M}^{\mathcal{A}}$ can answer all signature queries and perfectly simulates the EU-CMA game for SPHINCS$^+$.

When $\mathcal{A}$ returns a forgery, $\mathcal{M}^{\mathcal{A}}$ runs verification and compares all computed values to the values it computed during set-up. If $\mathcal{M}^{\mathcal{A}}$ finds a second preimage it outputs it together with its query index (indicating when it was sent to the W-SM-TCR oracle).

Hence we obtain

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.3.1} - \mathrm{Succ}_{\mathcal{A}}^{GAME.3.2}| \leq \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_3) \tag{8}$$

So the only case left to hop to GAME.4 is a WOTS-TW forgery that gives us the preimage. Since iff there is no WOTS-TW forgery then there must be a collision. And if there is the WOTS-TW forgery we can obtain either a collision (this case we have already excluded) or a preimage. To do so we first obtain preimage challenges for appropriate tweaks that were used in every WOTS-TW instance. To obtain the positions to place challenges for WOTS-TW instances we will again use $\mathbf{Th}_\lambda$ oracle (in the same way as for W-SM-UD). We use it to build WOTS-TW public keys and the whole SPHINCS$^+$ structure. This is done the same way as it was described in previous section. We obtain the following bound

$$|\mathrm{Succ}_{\mathcal{A}}^{GAME.3.2} - \mathrm{Succ}_{\mathcal{A}}^{GAME.4}| \leq \mathrm{InSec}^{\mathrm{W-SM-PRE}}(\mathbf{Th}; \xi, q_4) \tag{9}$$

where $q_4 < 2^{h+1} \cdot l \cdot w$.

GAME.4 The analysis of the GAME.4 can be found in the SPHINCS$^+$ paper [BHK$^+$19](Claim 23). Here we note that we can not use W-SM-PRE bound as the reduction uses T-openPRE game that was introduced in [BH19b]. The only difference is that we have already excluded the WOTS-TW preimage case. Hence we obtain the following bound:

$$\mathrm{Succ}_{\mathcal{A}}^{GAME.4} \leq 3 \cdot \mathrm{InSec}^{\mathrm{W-SM-TCR}}(\mathbf{Th}; \xi, q_5) + \mathrm{InSec}^{\mathrm{SM-DSPR}}(\mathbf{Th}; \xi, q_5) \quad (10)$$

where $q_5 < 2^h \cdot kt$

Combining the inequalities we obtain the bound from the theorem.

References

ABB$^+$20. Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS$^+$. Submission to NIST's post-quantum crypto standardization project, v.3, 2020. http://sphincs.org/data/sphincs+-round3-specification.pdf.

BH19a. Daniel J. Bernstein and Andreas Hülsing. Decisional second-preimage resistance: When does SPR imply PRE? In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 33–62. Springer, Heidelberg, December 2019.

BH19b. Daniel J. Bernstein and Andreas Hülsing. Decisional second-preimage resistance: When does SPR imply PRE? Cryptology ePrint Archive, Report 2019/492, 2019. https://eprint.iacr.org/2019/492.

BHK$^+$19. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2129–2146. ACM Press, November 2019.

DSS05. C. Dods, Nigel P. Smart, and Martijn Stam. Hash based digital signature schemes. In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 96–115. Springer, Heidelberg, December 2005.

HRS16. Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 387–416. Springer, Heidelberg, March 2016.

Hül13. Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13*, volume 7918 of *LNCS*, pages 173–188. Springer, Heidelberg, June 2013.

MK. Aleksey Fedorov Mikhail Kudinov, Evgeniy Kiktenko. [pqc-forum] round 3 official comment: Sphincs+. https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/official-comments/Sphincs-Plus-round3-official-comment.pdf. Accessed: 2010-10-30.

RSM09. Mohammad Reza Reyhanitabar, Willy Susilo, and Yi Mu. Enhanced target collision resistant hash functions revisited. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 327–344. Springer, Heidelberg, February 2009.

From: Sydney Antonov <ska84@protonmail.com>
Sent: Wednesday, April 20, 2022 7:21 PM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: SPHINCS+

Dear all,

SPHINCS+ relies on the distinct-function, multi-target second-preimage
resistance (DM-SPR) of the underlying keyed hash function. This property can be broken for SHA-256(key||message)
(which is used by
SPHINCS+-SHA-256) using around (t-1)2**128 + 2^256/t compression
SPHINCS+function
calls when attacking t targets using the following attack:

In this attack keys are initial hash values instead of message prefixes, without loss of generality.

Let C: {0,1}^256 x {0,1}^512 -> {0,1}^256 be SHA-256's compression function.

1. If there are multiple keys:
1.1. For each pair of keys (k, l):
1.1.1. Find an (x_k, x_l) such that y = C(k, x_k) = C(l, x_l).
 This can be done using around 2^128 compression function calls.
1.1.2. Replace the pair with the single key y.
1.2. If there's a remaining key k because there was an odd number of keys replace it with C(k, 0).
1.3. Repeat step 1.
2. Find a SHA-256 preimage of one of the targets using the final key as
 an IV. This can be done using around 2^256/t compression function
 calls.
3. Concatenate the sequence of compression function blocks used in step
 1 to derive the final key from the key corresponding to the target
 for which a second-preimage was found by step 2.
4. Concatenate the results of step 3 and step 2.

The result of step 4 is a preimage for one of the targets and an attacker can ensure it's almost certainly not the original one by randomizing step 2.

This attack means the use of unique prefixes/IVs with SHA-256 provides very little protection against multi-target second-preimage attacks when t < 2^64, but it's less effective on truncated SHA-256.

If "internal claws" could be found with cost c, this attack would cost (t-1)c + 2^256/t, so if they could be found more efficiently it could also threaten SHAKE256 and truncated SHA-256. Also claw-freeness seems almost as strong an assumption as collision-resistance so I don't think it's a desirable assumption for a collision-resilient signature scheme.

Sydney

From: Andreas Hülsing <ietf@huelsing.net>
Sent: Thursday, April 21, 2022 3:51 AM
To: Sydney Antonov; pqc-comments
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

Dear Sydney,

Thanks for looking at SPHINCS+. This is an interesting attack that does demonstrate that our real hash functions do not perfectly behave like random oracles (and that the chaining value of SHA2 may be a bit too short for some use cases). Luckily its impact against SPHINCS+ is limited (that is of course if I did not miss anything). The reason is that SPHINCS+ only needs DM-SPR for SHA2 with fixed length messages. The longest message length used is 67*n (+n for the key) bytes which occurs in the compression of the WOTS public keys for w=16 and uses 34 compression function calls. This means you can at most gain a 2^33 speed-up / 33 bits of security for the n=32 variant, classically.

As you mentioned, the impact is less for the truncated variants. To be precise, for n = 16 there will be no advantage as the internal state is twice the length of the hash values, so finding collisions on that state takes as much time as finding preimages. For n=24 the maximum message length processed by SHA2 in the relevant setting is bounded by 51, meaning 26 compression function calls, and hence at most a 2^26 speed-up.

For quantum attacks, the difference between collision search and preimage search shrinks down even if we do not take access times for quantum accessible memory into account. E.g., for n = 24 the attack cost would be (t-1)2**64 + 2^96/t in the ideal case where memory access is free. In this case one can at most gain 16 bits anyway and using reasonable estimates for memory access, even less.

Best wishes,

Andreas

On 21-04-2022 01:20, 'Sydney Antonov' via pqc-forum wrote:

> Dear all,
>
> SPHINCS+ relies on the distinct-function, multi-target second-preimage
> resistance (DM-SPR) of the underlying keyed hash function. This
> property can be broken for SHA-256(key||message) (which is used by
> SPHINCS+-SHA-256) using around (t-1)2**128 + 2^256/t compression
> SPHINCS+function
> calls when attacking t targets using the following attack:
>
> In this attack keys are initial hash values instead of message
> prefixes, without loss of generality.
>
> Let C: {0,1}^256 x {0,1}^512 -> {0,1}^256 be SHA-256's compression
> function.
>
> 1. If there are multiple keys:

From: msg <msg260@gmail.com>
Sent: Thursday, April 21, 2022 4:24 AM
To: pqc-forum
Cc: Andreas Hülsing; pqc-forum; Sydney Antonov; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

Thanks for the discussion.

What do you think using KMAC or HMAC instead of Hash(Key||Message). I think these structures resist Sydney's attack.

Best.

21 Nisan 2022 Perşembe tarihinde saat 10:51:44 UTC+3 itibarıyla Andreas Hülsing şunları yazdı:

> Dear Sydney,
>
> Thanks for looking at SPHINCS+. This is an interesting attack that does demonstrate that our real hash functions do not perfectly behave like random oracles (and that the chaining value of SHA2 may be a bit too short for some use cases). Luckily its impact against SPHINCS+ is limited (that is of course if I did not miss anything). The reason is that SPHINCS+ only needs DM-SPR for SHA2 with fixed length messages. The longest message length used is 67*n (+n for the key) bytes which occurs in the compression of the WOTS public keys for w=16 and uses 34 compression function calls. This means you can at most gain a 2^33 speed-up / 33 bits of security for the n=32 variant, classically.
>
> As you mentioned, the impact is less for the truncated variants. To be precise, for n = 16 there will be no advantage as the internal state is twice the length of the hash values, so finding collisions on that state takes as much time as finding preimages. For n=24 the maximum message length processed by SHA2 in the relevant setting is bounded by 51, meaning 26 compression function calls, and hence at most a 2^26 speed-up.
>
> For quantum attacks, the difference between collision search and preimage search shrinks down even if we do not take access times for quantum accessible memory into account. E.g., for n = 24 the attack cost would be (t-1)2**64 + 2^96/t in the ideal case where memory access is free. In this case one can at most gain 16 bits anyway and using reasonable estimates for memory access, even less.
>
> Best wishes,
>
> Andreas
>
>
> On 21-04-2022 01:20, 'Sydney Antonov' via pqc-forum wrote:

From: Sydney Antonov <ska84@protonmail.com>
Sent: Thursday, April 21, 2022 6:11 AM
To: msg
Cc: pqc-forum; Andreas Hülsing; pqc-comments
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

> For quantum attacks, the difference between collision search and
> preimage search shrinks down even if we do not take access times for
> quantum accessible memory into account. E.g., for n = 24 the attack
> cost would be (t-1)2**64 + 2^96/t in the ideal case where memory
> access is free. In this case one can at most gain 16 bits anyway and
> using reasonable estimates for memory access, even less.

I think the cost of 192-bit Grover search is at least 2^128 because attackers don't have time for 2^96 iterations of Grover's algorithm so instead would have to do something like build 2^64 quantum computers which each do 2^64 iterations in parallel.

> What do you think using KMAC or HMAC instead of Hash(Key||Message).
> I think these structures resist Sydney's attack.

KMAC is a totally different construction to HMAC. It adsorbs the key once at the start so it's as vulnerable as to my attack SHAKE. That is, it could become vulnerable if a breakthrough is made in Keccak cryptanalysis. Internal collisions in SHAKE256 require around 2^256 classical queries to the Keccak permutation if it's modeled as a random oracle. The best known attacks are no better than generic attacks.

Sydney

From:	Sydney Antonov <ska84@protonmail.com>
Sent:	Thursday, April 21, 2022 3:19 PM
To:	Andreas Hülsing
Cc:	pqc-comments; pqc-forum
Subject:	Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

Dear Andreas,

> Thanks for looking at SPHINCS+. This is an interesting attack that
> does demonstrate that our real hash functions do not perfectly behave
> like random oracles (and that the chaining value of SHA2 may be a bit
> too short for some use cases). Luckily its impact against SPHINCS+ is
> limited (that is of course if I did not miss anything). The reason is
> that SPHINCS+ only needs DM-SPR for SHA2 with fixed length messages.
> The longest message length used is 67*n (+n for the key) bytes which
> occurs in the compression of the WOTS public keys for w=16 and uses 34
> compression function calls. This means you can at most gain a 2^33
> speed-up / 33 bits of security for the n=32 variant, classically.

This can be improved upon using a ternary tree of 3-claws with that message block limitation.

The number of 3-claws required is just under half the number of targets
(1/3 + 1/9 + ... = 1/2) and finding a 3-claw uses around 3*2**((2/3)256) compression function calls so the total cost is around 1.5t2^((2/3)256) + (2^256)/t compression function calls.

The optimal number of targets for that formula is sqrt((2^(256/3))/1.5) = 2^42.374... = 3^26.735..., resulting in a 2^214.63 attack.

P.S. Maybe this attack can be optimized further by mixing 2-claws and 3-claws.

Sydney

From: Sydney Antonov <ska84@protonmail.com>
Sent: Friday, April 22, 2022 5:49 AM
To: Sydney Antonov
Cc: Andreas Hülsing; pqc-comments; pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

> P.S. Maybe this attack can be optimized further by mixing 2-claws and 3-claws.

It can. The following is a 2^209.91 attack against 2^10*3^23 = 2^46.454... targets:

In this attack keys are initial hash values instead of message prefixes, without loss of generality.

Let C: {0,1}^256 x {0,1}^512 -> {0,1}^256 be SHA-256's compression function.

```
1. Repeat 10 times:
1.1. For each pair of keys (k, l):
1.1.1. Find an (x_k, x_l) such that y = C(k, x_k) = C(l, x_l).
    This can be done using around 2*2^(256/2) compression function calls.
1.1.2. Replace the pair with the single key y.
2. Repeat 23 times:
2.1. For each trio of keys (k, l, m):
2.1.1. Find an (x_k, x_l, x_m) such that y = C(k, x_k) = C(l, x_l) = C(m, x_m).
    This can be done using around (3/2)2^((2/3)256) compression function calls.
2.1.2. Replace the trio with the single key y.
3. Find a SHA-256 preimage of one of the targets using the final key as an IV. This can be done using around
2^256/2^10/3^23 compression function calls.
4. Concatenate the sequence of compression function blocks used in steps
1 and 2 to derive the final key from the key corresponding to the target for which a preimage was found by step 3.
5. Concatenate the results of step 4 and step 3.
```

The result of step 5 is a preimage for one of the targets and an attacker can ensure it's almost certainly not the original one by randomizing step 3.

In total this attack makes around 2^10*3^23*2*2^(256/2) +
3^23*(3/2)*2^((2/3)256) + 2^256/2^10/3^23 or 2^209.91 compression function calls.

Sydney

From: Andreas Hülsing <andreas@huelsing.net>
Sent: Friday, June 10, 2022 3:49 AM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: SPHINCS+
Attachments: sphincs+-r3.1-specification.pdf; r3.1_modifications.pdf

Dear all,

In response to the recently posted issue with the SHA2 instantiation of
SPHINCS+, we decided to change the function we use in the concerned
implementations from SHA2-256 to SHA2-512. The problem was caused by the small internal state of SHA2. With this change, this problem is solved.

We updated our specification, also including other changes that we announced on this list since the 3rd round submission. In the attachment you also find a list of all the modifications we made. The code in our public github repository (https://github.com/sphincs/sphincsplus) is also updated.

Best wishes,

Andreas

From: pqc-forum@list.nist.gov on behalf of Doge Protocol <dogeprotocol1@gmail.com>
Sent: Friday, June 10, 2022 1:09 PM
To: pqc-forum
Cc: Andreas Hülsing; pqc-forum; pqc-comments
Subject: [pqc-forum] Re: ROUND 3 OFFICIAL COMMENT: SPHINCS+

From table in the paper, SHA2-256 corresponds to security level 1 (n = 16) and SHA2-512 corresponds to security levels 3,5 (n=24 , n= 32). Does this mean only security levels 3,5 should be considered for SPHINCS+ and level 1 is not applicable since it has below problem? Is understanding correct?

https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/FVItvyRea28/m/-oKR3ZKpDAAJ

On Friday, June 10, 2022 at 12:54:41 AM UTC-7 Andreas Hülsing wrote:

> Dear all,
>
> In response to the recently posted issue with the SHA2 instantiation of SPHINCS+, we decided to change the function we use in the concerned implementations from SHA2-256 to SHA2-512. The problem was caused by the small internal state of SHA2. With this change, this problem is solved.
>
> We updated our specification, also including other changes that we announced on this list since the 3rd round submission. In the attachment you also find a list of all the modifications we made. The code in our public github repository (https://github.com/sphincs/sphincsplus) is also updated.
>
> Best wishes,
>
> Andreas

From: Bas Westerbaan <bas@cloudflare.com>
Sent: Friday, June 10, 2022 1:15 PM
To: Doge Protocol
Cc: Andreas Hülsing; pqc-forum; pqc-comments
Subject: Re: [pqc-forum] Re: ROUND 3 OFFICIAL COMMENT: SPHINCS+

On Fri, Jun 10, 2022 at 7:08 PM Doge Protocol <dogeprotocol1@gmail.com> wrote:

> Does this mean only security levels 3,5 should be considered for SPHINCS+ and level 1 is not applicable since it has below problem? Is understanding correct?

No. Antonov's attack only applied to levels 3 and 5. Read the end of step 1.1.1.

Best,

Bas

From: Andreas Hülsing <ietf@huelsing.net>
Sent: Friday, June 10, 2022 5:11 PM
To: Doge Protocol; pqc-forum
Cc: pqc-comments
Subject: Re: ROUND 3 OFFICIAL COMMENT: SPHINCS+

The document only contains parameters that we consider to achieve the claimed security level. So, all the proposed parameter sets can be used. The security of SHA2-256 is sufficient for the instantiations of all functions at security level 1 (and therefore the level 1 parameters using SHA2 can be used safely). Only at security levels 3 and 5, SHA2-256 is not sufficient for the instantiation of tweakable hash functions that take input messages of a length that is more than one message block length. Therefore, we switched the instantiation of the H and T functions to SHA2-512 at security levels 3 and 5.

Best wishes,

Andreas

On 10-06-2022 19:08, Doge Protocol wrote:

> From table in the paper, SHA2-256 corresponds to security level 1 (n = 16) and SHA2-512 corresponds to security levels 3,5 (n=24 , n= 32). Does this mean only security levels 3,5 should be considered for SPHINCS+ and level 1 is not applicable since it has below problem? Is understanding correct?
>
> https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/FVItvyRea28/m/-oKR3ZKpDAAJ
>
> On Friday, June 10, 2022 at 12:54:41 AM UTC-7 Andreas Hülsing wrote:
>
>> Dear all,
>>
>> In response to the recently posted issue with the SHA2 instantiation of SPHINCS+, we decided to change the function we use in the concerned implementations from SHA2-256 to SHA2-512. The problem was caused by the small internal state of SHA2. With this change, this problem is solved.
>>
>> We updated our specification, also including other changes that we announced on this list since the 3rd round submission. In the attachment you also find a list of all the modifications we made. The code in our public github repository (https://github.com/sphincs/sphincsplus) is also updated.
>>
>> Best wishes,
>>
>> Andreas

From: Sydney Antonov <ska84@protonmail.com>
Sent: Monday, June 13, 2022 9:35 AM
To: Andreas Hülsing
Cc: pqc-comments; pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: SPHINCS+

> In response to the recently posted issue with the SHA2 instantiation
> of
> SPHINCS+, we decided to change the function we use in the concerned
> implementations from SHA2-256 to SHA2-512. The problem was caused by
> the small internal state of SHA2. With this change, this problem is solved.

While this prevents my attack when collisions must be found using generic attacks, it would still be able to gain some advantage if better chosen-prefix collisions attacks on SHA-2 were found like with MD5 and SHA-1. I find this concerning because SPHINCS+ is designed to be collision-resilient and this means it isn't tightly chosen-prefix-collision-resilient.

Sydney

From: Doge Protocol <dogeprotocol1@gmail.com>
Sent: Monday, June 13, 2022 12:42 PM
To: pqc-forum
Cc: Andreas Hülsing; pqc-comments; Doge Protocol
Subject: Re: ROUND 3 OFFICIAL COMMENT: SPHINCS+

Is there any impact/trade-off, such as in performance, on account of this change to SHA2-512?

On Friday, June 10, 2022 at 2:11:06 PM UTC-7 Andreas Hülsing wrote:

> The document only contains parameters that we consider to achieve the claimed security level. So, all the proposed parameter sets can be used. The security of SHA2-256 is sufficient for the instantiations of all functions at security level 1 (and therefore the level 1 parameters using SHA2 can be used safely). Only at security levels 3 and 5, SHA2-256 is not sufficient for the instantiation of tweakable hash functions that take input messages of a length that is more than one message block length. Therefore, we switched the instantiation of the H and T functions to SHA2-512 at security levels 3 and 5.
>
> Best wishes,
>
> Andreas
>
> On 10-06-2022 19:08, Doge Protocol wrote:
>
> > From table in the paper, SHA2-256 corresponds to security level 1 (n = 16) and SHA2-512 corresponds to security levels 3,5 (n=24 , n= 32). Does this mean only security levels 3,5 should be considered for SPHINCS+ and level 1 is not applicable since it has below problem? Is understanding correct?
> >
> > https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/FVItvyRea28/m/-oKR3ZKpDAAJ
> >
> > On Friday, June 10, 2022 at 12:54:41 AM UTC-7 Andreas Hülsing wrote:
> >
> > > Dear all,
> > >
> > > In response to the recently posted issue with the SHA2 instantiation of SPHINCS+, we decided to change the function we use in the concerned implementations from SHA2-256 to SHA2-512. The problem was caused by the small internal state of SHA2. With this change, this problem is solved.
> > >
> > > We updated our specification, also including other changes that we announced on this list since the 3rd round submission. In the attachment you also find a list of all the modifications we made. The code in our public github repository (https://github.com/sphincs/sphincsplus) is also updated.
> > >
> > > Best wishes,
> > >
> > > Andreas

From: Bas Westerbaan <bas@cloudflare.com>
Sent: Monday, June 13, 2022 12:46 PM
To: Doge Protocol
Cc: pqc-forum; Andreas Hülsing; pqc-comments
Subject: Re: [pqc-forum] Re: ROUND 3 OFFICIAL COMMENT: SPHINCS+

It depends on the platform and parameter set; on some it's faster and on some it's slower.

Here you can see the differences for a Ryzen 5 1600: https://github.com/sphincs/sphincsplus/pull/31#issuecomment-1146220423:~:text=commented-,10%20days%20ago,-sphincs%2Dsha256%2D256s

On Mon, Jun 13, 2022 at 6:41 PM Doge Protocol <dogeprotocol1@gmail.com> wrote:

> Is there any impact/trade-off, such as in performance, on account of this change to SHA2-512?
>
> On Friday, June 10, 2022 at 2:11:06 PM UTC-7 Andreas Hülsing wrote:
>
> > The document only contains parameters that we consider to achieve the claimed security level. So, all the proposed parameter sets can be used. The security of SHA2-256 is sufficient for the instantiations of all functions at security level 1 (and therefore the level 1 parameters using SHA2 can be used safely). Only at security levels 3 and 5, SHA2-256 is not sufficient for the instantiation of tweakable hash functions that take input messages of a length that is more than one message block length. Therefore, we switched the instantiation of the H and T functions to SHA2-512 at security levels 3 and 5.
> >
> > Best wishes,
> >
> > Andreas
> >
> > On 10-06-2022 19:08, Doge Protocol wrote:
> >
> > > From table in the paper, SHA2-256 corresponds to security level 1 (n = 16) and SHA2-512 corresponds to security levels 3,5 (n=24 , n= 32). Does this mean only security levels 3,5 should be considered for SPHINCS+ and level 1 is not applicable since it has below problem? Is understanding correct?
> > >
> > > https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/FVItvyRea28/m/-oKR3ZKpDAAJ
> > >
> > > On Friday, June 10, 2022 at 12:54:41 AM UTC-7 Andreas Hülsing wrote:
> > >
> > > > Dear all,
> > > >
> > > > In response to the recently posted issue with the SHA2 instantiation of SPHINCS+, we decided to change the function we use in the concerned implementations from SHA2-256 to SHA2-512. The problem was caused by the small internal state of SHA2. With this change, this problem is solved.
> > > >
> > > > We updated our specification, also including other changes that we announced on this list since the 3rd round submission. In the attachment you also find a list of all the modifications we made. The code in our public github repository (https://github.com/sphincs/sphincsplus) is also updated.