Machine-Checked Security for XMSS as in RFC 8391 and SPHINCS +

Manuel Barbosa, François Dupressoir, Benjamin Grégoire, Andreas Hülsing, Matthias Meijers, Pierre-Yves Strub

Machine-Checked Security for XMSS as in RFC 8391 and SPHINCS$^+$

Manuel Barbosa[1][0000−0002−6848−5564], François Dupressoir[2][0000−0003−3497−3110], Benjamin Grégoire[3][0000−0001−6650−9924], Andreas Hülsing[4][0000−0003−2215−4134], Matthias Meijers[4][0000−0002−5351−991X], and Pierre-Yves Strub[5][0000−0002−8196−7875]

[1] University of Porto (FCUP) and INESC TEC, Portugal
[2] University of Bristol, United Kingdom
[3] Université Côte d'Azur, Inria, France
[4] Eindhoven University of Technology, The Netherlands
[5] Meta, France
fv-xmss@mmeijers.com

Abstract This work presents a novel machine-checked tight security proof for XMSS — a stateful hash-based signature scheme that is (1) standardized in RFC 8391 and NIST SP 800-208, and (2) employed as a primary building block of SPHINCS$^+$, one of the signature schemes recently selected for standardization as a result of NIST's post-quantum competition.

In 2020, Kudinov, Kiktenko, and Fedoro pointed out a flaw affecting the tight security proofs of SPHINCS$^+$ and XMSS. For the case of SPHINCS$^+$, this flaw was fixed in a subsequent tight security proof by Hülsing and Kudinov. Unfortunately, employing the fix from this proof to construct an analogous tight security proof for XMSS would merely demonstrate security with respect to an insufficient notion.

At the cost of modeling the message-hashing function as a random oracle, we complete the tight security proof for XMSS and formally verify it using the EasyCrypt proof assistant. (Note that this merely extends the use of the random oracle model, as this model is already required in other parts of the security analysis to justify the currently standardized parameter values). As part of this endeavor, we formally verify the crucial step common to the security proofs of SPHINCS$^+$ and XMSS that was found to be flawed before, thereby confirming that the core of the aforementioned security proof by Hülsing and Kudinov is correct.

As this is the first work to formally verify proofs for hash-based signature schemes in EasyCrypt, we develop several novel libraries for the fundamental cryptographic concepts underlying such schemes — e.g., hash functions and digital signature schemes — establishing a common starting point for future formal verification efforts. These libraries will be particularly helpful in formally verifying proofs of other hash-based signature schemes such as LMS or SPHINCS$^+$.

Keywords: XMSS · SPHINCS$^+$ · EasyCrypt · Formal Verification · Machine-Checked Proofs · Computer-Aided Cryptography

1 Introduction

Quantum computers threaten the security of virtually all public-key cryptography deployed today [Mos18]. Although it is still unclear if and when large-scale quantum computers will become operational, there is continuous progress [GH19] and the stakes are too high to risk not being prepared. For this reason, in late 2016, the National Institute of Standards and Technology (NIST) initiated a standardization process for post-quantum cryptography, i.e., classically computable cryptographic constructions that can withstand attacks by quantum-capable adversaries [NIS16]. Nearly six years later, NIST finally announced the first four constructions to be standardized: the key encapsulation mechanism CRYSTALS-Kyber, and the digital signature schemes CRYSTALS-Dilithium, Falcon, and SPHINCS$^+$ [NIS22]. However, for early adopters, NIST published an initial standard (in 2020) describing the stateful hash-based signature schemes XMSS and LMS [CAD+20], both previously specified in Request For Comments (RFC) publications [HBG+18,MCF19]. These schemes provide post-quantum secure signatures to users that can handle a secret state, i.e., a secret key that changes over time. Interestingly, these schemes share a lot of structure with each other and with SPHINCS$^+$.

In 2020, Kudinov, Kiktenko, and Fedoro pointed out a flaw in the tight security proof of the Winternitz One-Time Signature Scheme (WOTS) [KKF20], one of the main building blocks of XMSS and SPHINCS$^+$. This flaw invalidated the tight security proof of XMSS [HRS16], as well as that of SPHINCS$^+$ [BHK+19]. (The non-tight security proofs were not affected by this flaw; however, these proofs could not justify practical parameters.) Regarding SPHINCS$^+$, this flaw was fixed by explicitly specifying the particular variant of WOTS employed in SPHINCS$^+$ (and XMSS), called WOTS-TW, and providing a new tight security proof for this variant [HK22]. This proof, however, only shows security of WOTS-TW against non-adaptive chosen-message attacks. For SPHINCS$^+$, this turns out to be sufficient because it uses WOTS-TW exclusively to sign user-controlled data; nevertheless, for XMSS, this is not sufficient as it additionally uses WOTS-TW to sign data that might be controlled by an adversary. This leaves the security of XMSS an open question. Moreover, the fact that this flaw was only found after four years — during which the concerned schemes received quite some attention — questions the guarantees provided by the novel security proof for SPHINCS$^+$.

Computer-Aided Cryptography. Hash-based signature schemes are not the only cryptographic constructions that have had flawed security proofs. Indeed, there exist numerous examples of proofs for cryptographic constructions that were widely considered to be correct after heavy scrutiny, but still turned out to be faulty. Furthermore, in some of these cases, the corresponding constructions were additionally shown to be insecure [KM19]. Often, the culprit in these situations is, at least partially, the sheer complexity of the cryptographic constructions and their proofs, as well as a lack of rigor in the exposition of the arguments in the proofs. Since post-quantum constructions and the correspond-

ing proofs tend to be relatively complex — and in many cases based on relatively novel and lesser-studied concepts — additional care and rigor should be applied in their evaluation.

Naturally, in the six years leading up to the final announcement in NIST's standardization process, each one of the eventually selected constructions underwent extensive scrutiny by the cryptographic community and, potentially after some adaptations, gained sufficient trust to be chosen for standardization. (Certainly, the above-mentioned flaw invalidating the original security proof of SPHINCS$^+$ was discovered during this process.) Nevertheless, this does not exclude the possibility that the current beliefs regarding the security of these constructions may be wrong, even if the corresponding proofs are currently deemed correct.

In an attempt to address the complexity issues associated with devising and evaluating cryptographic constructions and their proofs, the field of computer-aided cryptography has produced a multitude of tools and frameworks aimed at reducing the manual effort required for the verification of cryptography and, ideally, reducing this effort to merely checking the security claims. Over the years, these tools and frameworks have been successfully applied in the construction and verification of increasingly intricate and important use cases. For instance, in no particular order, CertiCrypt has been used to formally verify the security of OAEP [BGLZ11]; EasyCrypt has been used to formally verify the security and correctness of Saber's public-key encryption scheme [HMS22]; Jasmin (in conjunction with EasyCrypt) has been used to construct and formally verify a functionally correct, constant-time, and efficient implementation of SHA-3 [ABB$^+$19]; and Tamarin has been used to formally verify TLS 1.3 [CHH$^+$17]. For a more comprehensive overview and discussion of computer-aided cryptography, refer to [BBB$^+$21]. However, to the best of our knowledge, the security properties of the standardized post-quantum hash-based signature schemes have not been analyzed using computer-aided cryptography prior to this work.

Our Contribution. In this work, we face the challenge of reestablishing or increasing the trust in the security of (the parameter sets considered for) XMSS and SPHINCS$^+$. To this end, we give a new tight security proof for XMSS building on the analysis of [HK22] and, moreover, formally verify the entire proof. As the proof in [HK22], our proof bases security on the properties of several keyed and tweakable hash functions in the standard model (see Section 2 for definitions) and achieves a minimal security loss of $\log w$ bits (where w is the Winternitz parameter, often set to 16 or 256 in practice). By performing our proof in a modular manner, we independently verify the part of the proof that is shared with the proof for SPHINCS$^+$ presented in [HK22]. To complete the proof for XMSS, we need to handle the signing of arbitrary-length messages; for this to work, we have to model the message-compression function as a random oracle. Although this segment of the proof could be carried out in the Quantum-accessible Random Oracle Model (QROM) using [GHHM21], we restrict ourselves to the Random Oracle Model (ROM) due to the current limitations of the utilized tool. Nevertheless, we closely follow the proof of [GHHM21] to facilitate

the lifting to the quantum setting as soon as the relevant shortcomings of the tool are overcome. It should be noted that in order to justify the currently standardized parameter values, the security analysis of XMSS already requires the (Q)ROM for matters other than the compression of messages [HRS16]; hence, our work does not introduce the (Q)ROM but merely extends the use of it. More precisely, as discussed in [HRS16], the (Q)ROM is already needed to argue that one can reduce the size of the public key by replacing a lengthy list of random values by a single public seed (that can be expanded to this list of random values using a public function). This use of the (Q)ROM is not explicit in this work as it is hidden behind the notion of tweakable hash functions.

We employ EasyCrypt, a tool predominantly designed for the formal verification of security properties through code-based, game-playing proofs in the computational model [BCG+12]. Since this is the first proper effort to verify hash-based signatures in EasyCrypt, our work includes several additions to the tool. In particular, we construct multiple comprehensive EasyCrypt libraries containing generic specifications of several fundamental cryptographic concepts underlying the schemes considered in this work. Specifically, we provide libraries for hash functions — both keyed and tweakable — and digital signature schemes — both stateless and stateful. As their content is specified generically, these libraries can be reused in any other context that considers these concepts. Although not presented here in detail, these libraries can be found in the code corresponding to this work or in the standard library of EasyCrypt.

Summarizing, the purpose of this work is to establish greater confidence in the security of XMSS and, by extension, SPHINCS$^+$. Additionally, this work aims to facilitate future formal verification efforts by providing generic libraries that are reusable in a plethora of contexts.

Future Work. In future work, we will expand this work in two complementary directions.

Extension to Quantum Setting. Our proof is formalized in the classical (i.e., non-quantum) setting. However, since most of contemporary interest in (standalone) XMSS is due to the scheme's claimed quantum-resistance, verifying whether this is actually veracious would be of significant value. In principle, the reasoning used throughout most of the formal verification still holds true in the quantum setting. However, handling of the message-compression step in the proof occurs in the ROM and, while this step can also be performed in the QROM, the necessary techniques [GHHM21] make use of advanced concepts such as compressed oracles [Zha19]. At the time of writing, the quantum extension of EasyCrypt [BBF+21] does not yet support some of these necessary techniques, but ongoing work attempts to surmount this deficiency and lift the proof to the quantum setting.

Formal Verification of SPHINCS$^+$ *and* LMS. A significant portion of this work focuses on the security of the substructure of XMSS that it shares with LMS and SPHINCS$^+$. Indeed, given that our proof sticks to the abstraction of tweakable hash functions, a proof for LMS seems realizable by harnessing the foundations

established in this work. Similarly, this work marks a significant milestone in formally verifying the security proof of SPHINCS$^+$. Building on the relevant results from this work, such a formal verification project seems feasible. Nevertheless, we note that this still requires significant additional effort due to the complexity of the SPHINCS$^+$ construction. For instance, among others, it requires the additional incorporation of FORS, the few-time signature scheme used in SPHINCS$^+$. In this context, because improper instantiations of the employed tweakable hash functions can lead to compromised security [PKC22], another interesting avenue for future work is the validation of the constructions of tweakable hash functions from [BHK+19]. However, again, this necessitates an expansion of the current quantum extension of EasyCrypt.

Overview. The remainder of this paper is structured as follows. First, Section 2 introduces the primary concepts underlying the considered cryptographic constructions and their formal verification. Second, Section 3 presents a high-level overview of (the employed approach to) the formal verification. Finally, the remaining sections — Section 4, Section 5, and Section 6 — discuss different parts of the formal verification in more detail.

Acknowledgments. Andreas Hülsing and Matthias Meijers are funded by an NWO VIDI grant (Project No. VI.Vidi.193.066). We thank the Formosa Crypto consortium for support and discussions.

2 Preliminaries

Below, we provide the background necessary for the remainder of the paper.

Keyed Hash Functions. A Keyed Hash Function (KHF) is a function $\mathsf{KHF} : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$ where *key space* $\mathcal{K}$ denotes a set of keys, and *message space* $\mathcal{M}$ and *digest space* $\mathcal{Y}$ are sets of bitstrings. Often, $\mathcal{Y}$ constitutes the set of bitstrings of a certain length — that is, $\mathcal{Y} = \{0,1\}^n$ for some $n > 0$ — and $\mathcal{M}$ constitutes the set of bitstrings of arbitrary length — that is, for potentially many different $n > 0$, $\{0,1\}^n \subseteq \mathcal{M}$. At times, instead of viewing a KHF as a single function, we interpret and refer to it as a family of hash functions that is indexed by keys from the key space. Indeed, each hash function in such a family has the message space as its domain and the digest space as its codomain.

Regarding KHFs, we are merely concerned with a variant of the *Collision Resistance* (CR) property called *Multi-target extended Target Collision Resistance* (M-eTCR), and the *Pseudo-Random Function family* (PRF) property. Intuitively, a KHF has M-eTCR if it is computationally infeasible to, after selecting a number of target messages and receiving a randomly sampled hash function (from the family defined by the KHF) for each of these, find a different message that maps to the same digest under some hash function (from the family defined by the KHF) as one of the previously selected messages does under its associated (randomly sampled) hash function. A KHF is a PRF if querying an unknown, randomly selected hash function from the family defined by the KHF is computationally indistinguishable from randomly sampling elements from the digest

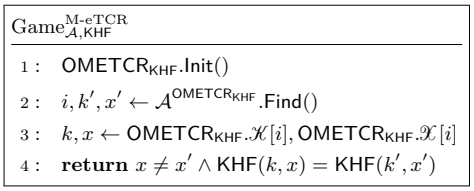

Figure 1. M-eTCR game for keyed hash functions.

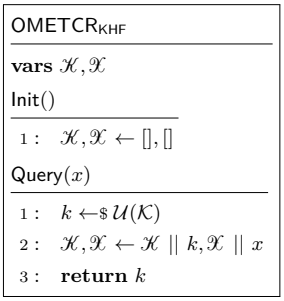

Figure 2. Oracle employed in the M-eTCR game for keyed hash functions.

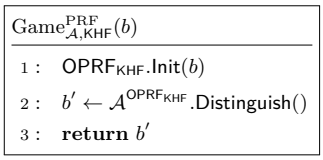

Figure 3. PRF game for keyed hash functions.

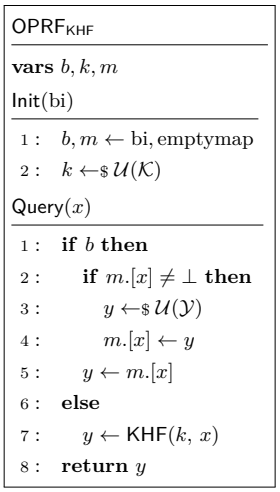

Figure 4. Oracle employed in the PRF game for keyed hash functions.

space. The M-eTCR and PRF properties for KHFs are formalized as the games in Figure 1 and Figure 3, respectively; the oracles employed in these games are given in Figure 2 and Figure 4. Certainly, in each of these games, the adversary merely gains access to the Query procedure of the provided oracle. Then, the advantage of any adversary $\mathcal{A}$ against M-eTCR is straightforwardly defined as follows.

$$\mathsf{Adv}^{\text{M-eTCR}}_{\mathsf{KHF}}(\mathcal{A}) = \Pr\left[\mathrm{Game}^{\text{M-eTCR}}_{\mathcal{A},\mathsf{KHF}} = 1\right]$$

Moreover, the advantage of any adversary $\mathcal{A}$ against PRF is defined as given below.

$$\mathsf{Adv}^{\text{PRF}}_{\mathsf{KHF}}(\mathcal{A}) = \left|\Pr\left[\mathrm{Game}^{\text{PRF}}_{\mathcal{A},\mathsf{KHF}}(0) = 1\right] - \Pr\left[\mathrm{Game}^{\text{PRF}}_{\mathcal{A},\mathsf{KHF}}(1) = 1\right]\right|$$

Tweakable Hash Functions. A Tweakable Hash Function (THF) is a function $\mathsf{THF} : \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{Y}$ where *(public) parameter space* $\mathcal{P}$ and *tweak space* $\mathcal{T}$ respectively denote a set of (public) parameters and a set of tweaks, and *message space* $\mathcal{M}$ and *digest space* $\mathcal{Y}$ are sets of bitstrings (corresponding to the same sets as the identically named sets in the definition of KHFs). THFs were first introduced in [BHK+19]. They form an extension of KHFs by allowing for the consideration of contextual data in the form of tweaks, i.e., elements from the tweak space.[6] Tweaks are predominantly used for the mitigation of multi-target attacks.

Alongside THFs, the authors of SPHINCS$^+$ introduced the concept of collections of such functions, containing a single THF for each possible length of the input messages [BHK+19]. Such a collection can be viewed as the set $\mathsf{THFC} = \{\mathsf{THF}_\lambda : \mathcal{P} \times \mathcal{T} \times \mathcal{M}_\lambda \to \mathcal{Y}\}_{\lambda \in \Lambda}$ where Λ is the index set that contains all the possible lengths of input messages and $\mathcal{M}_\lambda$ is the set of bitstrings of length λ, i.e., $\mathcal{M}_\lambda = \{0,1\}^\lambda$.

The properties we consider for (collections of) THFs in this work are the *Single-function, Multi-target, Distinct-Tweak* versions of *UnDetectability* (SM-DT-UD-C); *Target-Collision Resistance* (SM-DT-TCR-C); and *PREimage resistance* (SM-DT-PRE-C) *of a THF as a member of a Collection*. These properties were first introduced in [HK22] for the purpose of recovering the tight security proof of SPHINCS$^+$. For an extensive discussion and in-depth analysis of these properties, refer to [HK22]. As their names suggest, the THF properties we consider are rather similar in their formalizations. Namely, for each property, the formalization is approximately structured as follows. First, during initialization, a parameter used to index the THF collection is sampled uniformly at random. Then, both the *challenge oracle* and the *collection oracle* are initialized with the sampled parameter. The challenge oracle allows the adversary to adaptively define targets (through queries consisting of tweaks and, depending on the property, potentially messages or digests) and learn the (claimed) corresponding mappings under the considered THF. The collection oracle enables the adversary to adaptively query any THF from the considered collection without defining these queries as targets. After initialization, the target-selection stage commences. During this stage, the adversary has access to both oracles, specifying its targets by queries to the challenge oracle. Afterward, the attack stage begins. Here, the adversary is given the used parameter and is asked to provide a solution for (one of) the targets that it specified in the target-selection stage.

Certainly, the form that a solution takes depends on the considered game: For SM-DT-UD-C, a solution is a boolean b' indicating whether the adversary thinks the challenge oracle returned digest of uniformly random messages ($b' = \text{false}$) or uniformly distributed digests ($b' = \text{true}$); for SM-DT-TCR-C, a solution consists of an index i pointing to a target (tw, x), and a message x' that should map to the same digest as message x when using tweak tw; and for SM-DT-PRE-C, a solution consists of an index i pointing to a target (tw, y), and a message x that should map to y using tweak tw.

[6] The parameter space of THFs is analogous to the key space of KHFs.

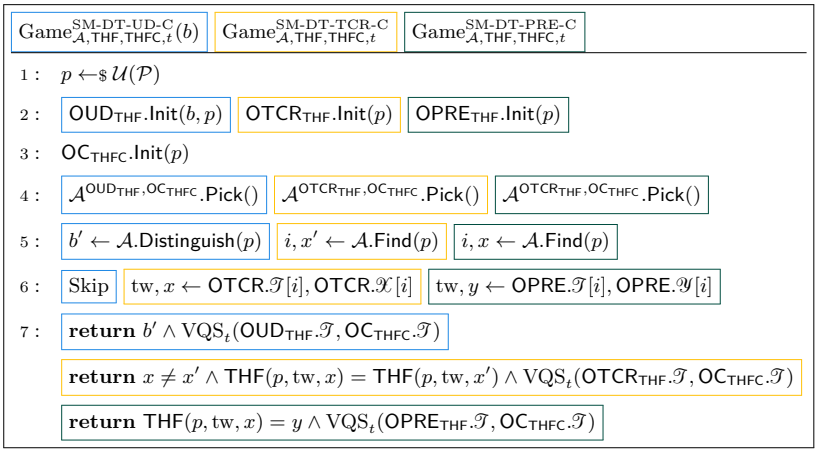

Figure 5. SM-DT-UD-C (blue boxes), SM-DT-TCR-C (yellow boxes), and SM-DT-PRE-C (green boxes) game for tweakable hash functions. Statements not within colored boxes are executed in every game.

Finally, following the attack stage, success of the adversary is checked by validating both the provided solution and the adversary's behavior. The latter is necessary as the adversary is not allowed to (1) specify more than a certain number of tweaks, (2) use the same tweak for different targets, and (3) query the collection oracle with a tweak occurring in any of the targets; this concurs with the fact that XMSS and SPHINCS$^+$ do not use the same tweak more than once. The games formalizing the considered THF properties are given in Figure 5; the oracles provided to the adversary in these games are specified in Figure 6 (SM-DT-UD-C challenge oracle), Figure 7 (SM-DT-TCR-C challenge oracle), Figure 8 (SM-DT-PRE-C challenge oracle), and Figure 9 (collection oracle). Naturally, the adversary exclusively gains access to the Query procedures of the given oracles. Furthermore, in these games, t denotes the number of targets the adversary may specify, and VQS_t is a predicate that validates the adversary's behavior based on the lists of tweaks from the challenge and collection oracles. Then, we respectively define the advantage of any adversary $\mathcal{A}$ against SM-DT-UD-C, SM-DT-TCR-C, and SM-DT-PRE-C as follows.

$$\begin{aligned}
\mathsf{Adv}^{\text{SM-DT-UD-C}}_{\mathsf{THF},\mathsf{THFC},t}(\mathcal{A}) &= \left|\Pr\left[\mathrm{Game}^{\text{SM-DT-UD-C}}_{\mathcal{A},\mathsf{THF},\mathsf{THFC},t}(0)=1\right]\right. \\
&\qquad \left. - \Pr\left[\mathrm{Game}^{\text{SM-DT-UD-C}}_{\mathcal{A},\mathsf{THF},\mathsf{THFC},t}(1)=1\right]\right|, \\
\mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{THF},\mathsf{THFC},t}(\mathcal{A}) &= \Pr\left[\mathrm{Game}^{\text{SM-DT-TCR-C}}_{\mathcal{A},\mathsf{THF},\mathsf{THFC},t}=1\right], \text{ and} \\
\mathsf{Adv}^{\text{SM-DT-PRE-C}}_{\mathsf{THF},\mathsf{THFC},t}(\mathcal{A}) &= \Pr\left[\mathrm{Game}^{\text{SM-DT-PRE-C}}_{\mathcal{A},\mathsf{THF},\mathsf{THFC},t}=1\right]
\end{aligned}$$

Addresses. XMSS — and, by extension, SPHINCS$^+$ — consists of multiple components; in each of these components, the same collection of THFs is employed.

$\mathsf{OUD}_{\mathsf{THF}}$

vars $b, p, \mathcal{T}$

$\mathsf{Init}(\mathrm{bi}, \mathrm{pi})$

1 : $b, p, \mathcal{T} \leftarrow \mathrm{bi}, \mathrm{pi}, [\,]$

$\mathsf{Query}(\mathrm{tw})$

1 : **if** b **then**
2 : $\quad y \leftarrow\!\!\$\ \mathcal{U}(\mathcal{Y})$
3 : **else**
4 : $\quad x \leftarrow\!\!\$\ \mathcal{U}(\mathcal{X})$
5 : $\quad y \leftarrow \mathsf{THF}(p, \mathrm{tw}, x)$
6 : $\mathcal{T} \leftarrow \mathcal{T} \,||\, \mathrm{tw}$
7 : **return** y

Figure 6. Challenge oracle employed in SM-DT-UD-C game.

$\mathsf{OTCR}_{\mathsf{THF}}$

vars $p, \mathcal{T}, \mathcal{X}$

$\mathsf{Init}(\mathrm{pi})$

1 : $p, \mathcal{T}, \mathcal{X} \leftarrow \mathrm{pi}, [\,], [\,]$

$\mathsf{Query}(\mathrm{tw}, x)$

1 : $y \leftarrow \mathsf{THF}(p, \mathrm{tw}, x)$
2 : $\mathcal{T}, \mathcal{X} \leftarrow \mathcal{T} \,||\, \mathrm{tw}, \mathcal{X} \,||\, x$
3 : **return** y

Figure 7. Challenge oracle employed in SM-DT-TCR-C game.

$\mathsf{OPRE}_{\mathsf{THF}}$

vars $p, \mathcal{T}, \mathcal{Y}$

$\mathsf{Init}(\mathrm{pi})$

1 : $p, \mathcal{T}, \mathcal{Y} \leftarrow \mathrm{pi}, [\,], [\,]$

$\mathsf{Query}(\mathrm{tw})$

1 : $x \leftarrow\!\!\$\ \mathcal{U}(\mathcal{X})$
2 : $y \leftarrow \mathsf{THF}(p, \mathrm{tw}, x)$
3 : $\mathcal{T}, \mathcal{Y} \leftarrow \mathcal{T} \,||\, \mathrm{tw}, \mathcal{Y} \,||\, y$
4 : **return** y

Figure 8. Challenge oracle employed in SM-DT-PRE-C game.

$\mathsf{OC}_{\mathsf{THFC}}$

vars $p, \mathcal{T}$

$\mathsf{Init}(\mathrm{pi})$

1 : $p, \mathcal{T} \leftarrow \mathrm{pi}, [\,]$

$\mathsf{Query}(\mathrm{tw}, x)$

1 : $y \leftarrow \mathsf{THFC}_{|x|}(p, \mathrm{tw}, x)$
2 : $\mathcal{T} \leftarrow \mathcal{T} \,||\, \mathrm{tw}$
3 : **return** y

Figure 9. Collection oracle employed in games of properties for tweakable hash functions.

As such, to mitigate multi-target attacks, schemes such as XMSS and SPHINCS$^+$ use a unique tweak for each THF call throughout the entire construction. For the generation of these tweaks, XMSS implements a particular addressing scheme. Although one could almost completely abstract this addressing scheme away in the analysis of XMSS and its components, we remain somewhat concrete as to keep the connection to the actual specification clear. In particular, XMSS employs addresses consisting of a fixed-length sequence of nonnegative integers indicating the location and purpose of the THF call in the virtual structure. Naturally, not all (fixed-length) sequences of nonnegative integers constitute valid addresses. Furthermore, when analyzing a specific component individually, some of the integers in the sequence may be irrelevant as they exclusively serve the purpose of achieving uniqueness when multiple instances of the same component

are considered simultaneously.[7] For these reasons, in this paper, we use "address" to refer to a fixed-length sequence of nonnegative integers that constitutes (the relevant part of a) valid XMSS or SPHINCS$^+$ address in the considered context. Further details regarding address validity will be provided, when relevant, throughout the paper.

(Tweakable Hash) Function Chains. Informally, a function chain is a sequence of values obtained by repeatedly applying a function on (a part of) its own output, starting with some given value as input. In the context of WOTS-TW, the chained function is a THF. In this chaining, the initially provided address is updated in each application of the THF as to ensure the address's uniqueness throughout. More precisely, given a function THF, parameter p, start index $s \in \mathbb{N}$, iteration counter $i \in \mathbb{N}$, message x, and address ad, the chaining function $\mathsf{Ch}_{\mathsf{THF}}$ is recursively defined as follows.

$$\mathsf{Ch}_{\mathsf{THF}}(p, \mathrm{ad}, s, i, x) = \begin{cases} x, & \text{if } i \leq 0 \\ \mathsf{THF}(p, \mathrm{ad}_{s+i-1}, \mathsf{Ch}_{\mathsf{THF}}(p, \mathrm{ad}, s, i-1, x)), & \text{otherwise} \end{cases}$$

Here, ad_{s+i-1} denotes the address resulting from adjusting ad to be the unique address corresponding to the $s+i-1$-th call to THF in the considered chain. Furthermore, this definition requires that the digest space of THF is contained in its message space; this is invariably the case for the THFs considered throughout this work. From this definition, we can derive the compositional property of chain functions that is represented by the following equality, where $i, j \in \mathbb{N}$, $0 \leq i$, $0 \leq j$, and the remaining values are as previously specified.

$$\mathsf{Ch}_{\mathsf{THF}}(p, \mathrm{ad}, s+i, j, \mathsf{Ch}_{\mathsf{THF}}(p, \mathrm{ad}, s, i, x)) = \mathsf{Ch}_{\mathsf{THF}}(p, \mathrm{ad}, s, i+j, x)$$

Intuitively, this equality states that chaining i times starting from position s — thus ending in position $s+i$ — and, subsequently, chaining j times from position $s+i$ is equal to chaining $i+j$ times starting from position s.

3 Approach

Our objectives in this work are basically twofold: First, we seek to formally verify the security property of XMSS as a standalone construction to increase the confidence in the security of this standardized scheme; second, we aim to formally verify the fix of the security proof of SPHINCS$^+$ —and, implicitly, XMSS— presented in [HK22] as to validate the remediation of the flaw in the original proof and pave the way for a complete formal verification of SPHINCS$^+$. Fortunately, when approached appropriately, the first objective can be achieved by extending

[7] For example, XMSS employs multiple instances of WOTS-TW, each of which is provided an address to perform its operations with. Since each instance manipulates and uses the same part of the provided address in an identical manner, XMSS ensures the part that is not considered by the WOTS-TW instances is different for each instance in order to still guarantee the uniqueness of the utilized addresses between instances.

(the results of) the second objective. In essence, this is because SPHINCS$^+$ employs (a variant of) XMSS as building block.

On a high level, XMSS is a Merkle signature scheme; that is, it comprises a binary hash tree of height h that authenticates the public keys of 2^h key pairs from a One-Time Signature (OTS) scheme. As alluded to before, the OTS employed in XMSS is (a variant of) WOTS. To reduce the size of the secret key, the sequence of 2^h WOTS secret keys — that originally constitutes the secret key — is replaced by a single seed used to (re)generate a WOTS secret key from the sequence whenever required via a PRF. A message is signed by first signing it with a WOTS secret key and then generating the authentication information for the corresponding public key. All of the above is the same for XMSS as standalone and XMSS as building block of SPHINCS$^+$.

There are two principal differences between XMSS as standalone and XMSS as used in SPHINCS$^+$. Foremost, standalone XMSS compresses messages using randomized hashing before they are signed. In SPHINCS$^+$, this is not necessary as messages signed by XMSS are public keys of other instances of signature schemes; these public keys already have the desired format. Second, the addresses in SPHINCS$^+$ have a slightly different format than those of standalone XMSS; nevertheless, their properties are identical.

As in the novel tight security proof for SPHINCS$^+$ [HK22], we describe XMSS using THFs whenever the input to the hash function includes an address (immediately giving rise to WOTS-TW [HK22] as the employed variant of WOTS), and base security on the properties of the utilized KHFs and THFs. That is, we formally verify that XMSS — both as standalone and as in SPHINCS$^+$ — is secure assuming that the employed KHFs and THFs have certain properties. This leads to a slightly more abstract description and result than for the "actual" standardized XMSS, where the utilized THFs are explicitly instantiated with KHFs (some of which are PRFs) [HBG+18]. Considering this slightly more abstract version of the standardized XMSS allows for a more natural way of achieving the above-mentioned objectives and makes the result more general, but precludes the security analysis of any concrete instantiations of the THFs (including the ones specified for XMSS in [HBG+18]). Henceforth, we refer to this slightly more abstract version of standalone XMSS as XMSS-TW; accordingly, we refer to XMSS as used in SPHINCS$^+$ as fixed-length XMSS-TW.

The fixed-length XMSS-TW employed by SPHINCS$^+$ does not achieve the standard Existential UnForgeability under adaptive Chosen-Message Attacks (EUF-CMA) security property that we typically expect standalone signature schemes, such as XMSS-TW, to possess. However, it does achieve the weaker Existential UnForgeability under Random Message Attacks (EUF-RMA) security property. Fortunately, because the difference between fixed-length XMSS-TW and (standalone) XMSS-TW can be seen as an instance of a particular transformation using the hash-then-sign paradigm, this weaker property of fixed-length

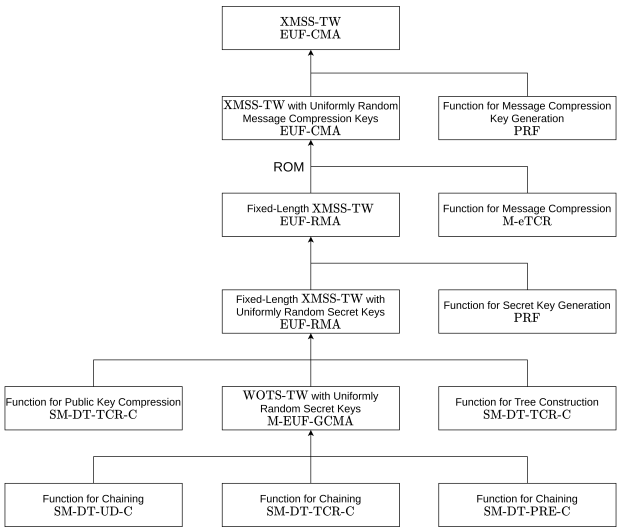

Figure 10. Overview of the (dependencies between) properties formally verified in this work. Within each node, the last line states the considered property of the cryptographic construction or function specified on the preceding lines. Furthermore, the property of the cryptographic construction or function associated with a node is implied by — or, equivalently, can be reduced from — the conjunction of the properties of the cryptographic constructions and functions associated with the origin nodes of the incoming arrows.

XMSS is sufficient to show — in the ROM — that XMSS-TW satisfies the desired EUF-CMA security property.[8]

A high-level overview of (the proofs underlying) our formal verification, i.e., the relations between the different properties we formally verify, is given in Figure 10. In this figure, each node signifies a property of a cryptographic construction or function: The initial lines state the considered construction or function; the last line states the considered property. Arrows denote the dependencies between properties; more precisely, the property of a cryptographic construction associated with a certain node is implied by — or, equivalently, can be reduced from — the conjunction of the properties of the cryptographic constructions and functions associated with the origin nodes of the incoming arrows.

The topmost node in Figure 10 states the first objective of this work: The formal verification of the EUF-CMA security of XMSS-TW. Now, this construction uses randomized hash-then-sign; that is, it compresses arbitrary-length messages to fixed-length messages using a KHF indexed by a (pseudo)random value. This value is freshly generated for each message compression using another KHF indexed by a (secret) key that is randomly sampled during key genera-

[8] Although one could consider the SEUF-CMA security notion for XMSS-TW (which is slightly stronger than EUF-CMA), we refrain from doing so because this would clutter the overall proof (and formal verification) without providing novel insights or being particularly relevant for most applications.

tion. This latter KHF is required to be a PRF in order to successfully perform adaptive reprogramming in the ROM based on the entropy of the values produced by this KHF. Moreover, the former KHF is required to have M-eTCR. As shown in [BHRv21], one can construct a reduction from weaker properties, i.e., some form of target-collision resistance. However, this requires an unconventional (index-bound) model for stateful signature schemes and their security; we consider this out-of-scope for this work.

In Figure 10, the two uppermost implications are related to the randomized hash-then-sign paradigm used in XMSS-TW. The latter of these implications is (partially) from the EUF-RMA property of fixed-length XMSS-TW which we, as the figure suggests, further deconstruct as follows. First, we demonstrate that this property is implied by the PRF property of the KHF employed in fixed-length XMSS-TW to generate secret keys and the EUF-RMA property of fixed-length XMSS-TW with uniformly random secret keys. Then, we show the EUF-RMA property of this variant of fixed-length XMSS-TW is implied by the SM-DT-TCR-C property of the two THFs used to construct the binary hash tree, and a variant of the EUF-CMA property specifically devised for WOTS-TW in [HK22]. However, instead of the original WOTS-TW — in which the secret key is generated via a PRF — we consider a version with uniformly random secret keys as a consequence of the PRF-related reduction for XMSS-TW mentioned above. Finally, we complete the second objective of this work — i.e, the formal verification of the fix of the security proof of SPHINCS$^+$ presented in [HK22] — by demonstrating that the SM-DT-UD-C, SM-DT-TCR-C, and SM-DT-PRE-C properties of the THF employed in WOTS-TW imply this dedicated variant of the EUF-CMA property.

After formally verifying each of the aforementioned implications, we combine the results to formally verify that the EUF-CMA security of XMSS-TW can exclusively be based on the properties of the employed THFs and KHFs (when the message-compression function is modeled as a random oracle), as desired.

In the subsequent sections, we discuss the formal verification process more thoroughly in a bottom-up manner; that is, we commence with WOTS-TW, proceed to fixed-length XMSS-TW, and finish with standalone XMSS-TW. Throughout this discussion, due to space considerations, we do not present any material directly from the produced formal verification artifacts. Instead, we go over the proofs that immediately underlie the formal verification in a manner that admits a near-direct translation to EasyCrypt and, hence, closely and accurately represents the formally verified material. Nonetheless, the produced formal verification artifacts can be found at `https://github.com/MM45/FV-XMSS-EC`.

4 WOTS-TW

The first explicit specification of (fixed-length) WOTS-TW was provided by Hülsing and Kudinov in their endeavor to recover the tight security proof of SPHINCS$^+$ [HK22]. As mentioned in the previous section, this original specification compresses the secret key via a PRF which, in our case, already happens

Algorithm 1 WOTS-TW$^{\$}$'s Public Key From Secret Key Algorithm

```
1: procedure WOTS-TW$.PkWotsFromSkWots(skWots, ps, ad)
2:     pkWots ← [ ]
3:     for i = 0, ..., len − 1 do
4:         ad.chainIndex ← i
5:         pkWots ← pkWots || CF(ps, ad, 0, w − 1, skWots[i])
6:     return pkWots
```

on the level of XMSS-TW, leading to a variant of WOTS-TW that straightforwardly considers uniformly random secret keys. In the ensuing, we denote this variant by WOTS-TW$^{\$}$ (and use WOTS-TW to refer to the original version).

Before presenting the actual specification, we go over several preliminaries. Foremost, the construction is defined with respect to parameters n — the byte-length of (1) secret key, public key, and signature elements, and (2) messages — and w — the Winternitz parameter, i.e., the radix in which messages are encoded. From these parameters, the following constants are computed: $\mathrm{len}_1 = \lceil \frac{8 \cdot n}{\log_2(w)} \rceil$ (number of w-ary digits necessary to represent any value of n bytes), $\mathrm{len}_2 = \lfloor \log_w(\mathrm{len}_1 \cdot (w-1)) \rfloor + 1$ (number of w-ary digits required to represent any value in the range $[0, \mathrm{len}_1 \cdot (w-1)]$), and $\mathrm{len} = \mathrm{len}_1 + \mathrm{len}_2$. In addition to these parameters and constants, WOTS-TW$^{\$}$ employs a THF with which it constructs function chains. Throughout the remainder, this THF and the corresponding chaining function are denoted by F and $\mathsf{CF}(:= \mathsf{Ch}_{\mathsf{F}})$, respectively. The message and digest space of F both equal $\{0,1\}^{8 \cdot n}$. Moreover, the parameter and tweak space of F are respectively referred to as the *public seed space* $\mathcal{PS}$ and *address space* $\mathcal{AD}$. Certainly, since we primarily consider WOTS-TW$^{\$}$ as a component of some greater structure such as (fixed-length) XMSS-TW or SPHINCS^{+}, these spaces may coincide with the corresponding spaces of the encompassing structure. In any case, we require the addresses to at least have a *chain index* — a nonnegative integer indicating the function chain in question — and a *hash index* — a nonnegative integer indicating the considered hash "iteration" within the function chain. As per the definition of a chaining function, the hash index is assumed to be updated internally by CF such that, even within a single chain, F is exclusively called with unique addresses. Besides these indices, the addresses may contain additional nonnegative integers that, for example, guarantee the uniqueness of the addresses between multiple instances of WOTS-TW$^{\$}$. As the concrete manifestation of such additional integers is irrelevant to the current analysis, we leave this unspecified here.

In WOTS-TW$^{\$}$, secret keys, public keys, and signatures consist of a sequence of len bitstrings, each of length $8 \cdot n$. Intuitively, the construction of these artifacts goes as follows. First, a secret key $\mathrm{sk} = \mathrm{sk}_0 \ldots \mathrm{sk}_{\mathrm{len}-1}$ is sampled uniformly at random from its domain. Then, the corresponding public key is computed by applying the chaining function to each sk_i, $0 \leq i < \mathrm{len}$, for $w-1$ iterations. Given a message $m \in \{0,1\}^{8 \cdot n}$, a signature is constructed by first encoding the message into a sequence of len w-ary digits. This encoding must have the prop-

Algorithm 2 WOTS-TW$^\$$'s Public Key From Signature Algorithm

procedure WOTS-TW$^\$$.PkWotsFromSig(m, sig, ps, ad)
  em $\leftarrow$ EncodeMessageWots(m)
  pkWots $\leftarrow$ [ ]
  for $i = 0, \ldots, \text{len} - 1$ **do**
    ad.chainIndex $\leftarrow i$
    pkWots $\leftarrow$ pkWots $\|$ CF(ps, ad, em$[i]$, $w - 1 -$ em$[i]$, sig$[i]$)
  return pkWots

Algorithm 3 WOTS-TW$^\$$'s Key Generation Algorithm

procedure WOTS-TW$^\$$.KeyGen(ps, ad)
  skWots $\leftarrow\$\ \mathcal{U}((\{0,1\}^{8 \cdot n})^{\text{len}})$
  pkWots $\leftarrow$ WOTS-TW$^\$$.PkWotsFromSkWots(skWots, ps, ad)
  return pk := (pkWots, ps, ad), sk := (skWots, ps, ad)

Algorithm 4 WOTS-TW$^\$$'s Signing Algorithm

procedure WOTS-TW$^\$$.Sign(sk := (skWots, ps, ad), m)
  em $\leftarrow$ EncodeMessageWots(m)
  sig $\leftarrow$ [ ]
  for $i = 0, \ldots, \text{len} - 1$ **do**
    ad.chainIndex $\leftarrow i$
    sig $\leftarrow$ sig $\|$ CF(ps, ad, 0, em$[i]$, skWots$[i]$)
  return sig

Algorithm 5 WOTS-TW$^\$$'s Verification Algorithm

procedure WOTS-TW$^\$$.Verify(pk := (pkWots, ps, ad), m, sig)
  pkWots$'$ $\leftarrow$ WOTS-TW$^\$$.PkWotsFromSig(m, sig, ps, ad)
  return pkWots$'$ = pkWots

erty that, for any other message, it contains at least one digit that is strictly less than the digit at the same index of the encoding of this other message. Albeit the majority of WOTS-based constructions — among which WOTS-TW and, hence, WOTS-TW$^\$$ — employ the same approach to encoding, we abstract away from the concrete approach and show that the results hold for any encoding with the foregoing property. Nevertheless, for completeness, we additionally demonstrate that the concrete encoding used by WOTS-TW possesses this property. Hereafter, we denote the operator performing the encoding by EncodeMessageWots. After encoding m into EncodeMessageWots$(m) = d_0 \ldots d_{\text{len}-1}$, the signature is obtained by applying the chaining function to sk_i for d_i iterations, $0 \leq i < \text{len}$. Notice that, from a message and its signature, the public key can be computed by completing the function chains. In fact, computing a public key in this manner and comparing it to the known public key is precisely how a signature is verified in WOTS-TW$^\$$.

$\mathrm{Game}^{\text{M-EUF-GCMA}}_{\mathcal{A},\text{WOTS-TW}^\$,\mathsf{THFC},d}$

1 : $\mathrm{ps} \leftarrow\$\ \mathcal{U}(\mathcal{PS})$
2 : $\mathsf{O}_{\text{WOTS-TW}^\$}.\mathsf{Init}(\mathrm{ps})$
3 : $\mathsf{OC}_{\mathsf{THFC}}.\mathsf{Init}(\mathrm{ps})$
4 : $\mathcal{A}^{\mathsf{O}_{\text{WOTS-TW}^\$},\mathsf{OC}_{\mathsf{THFC}}}.\mathsf{Choose}()$
5 : $i, m', \mathrm{sig}' \leftarrow \mathcal{A}.\mathsf{Forge}(\mathrm{ps})$
6 : $\mathrm{ad}, m, \mathrm{pkWots} \leftarrow \mathsf{O}_{\text{WOTS-TW}^\$}.\mathscr{A}[i], \mathsf{O}_{\text{WOTS-TW}^\$}.\mathscr{M}[i], \mathsf{O}_{\text{WOTS-TW}^\$}.\mathscr{P}[i]$
7 : $\mathrm{isValid} \leftarrow \text{WOTS-TW}^\$.\mathsf{Verify}((\mathrm{pkWots}, \mathrm{ps}, \mathrm{ad}), m', \mathrm{sig}')$
8 : $\mathrm{isFresh} \leftarrow m \neq m'$
9 : **return** $\mathrm{isValid} \wedge \mathrm{isFresh} \wedge \mathrm{VAD}_d(\mathsf{O}_{\text{WOTS-TW}^\$}.\mathscr{A}, \mathsf{OC}_{\mathsf{THFC}}.\mathscr{A})$

Figure 11. M-EUF-GCMA game for WOTS-TW$^\$$.

The specification of WOTS-TW$^\$$ is provided in Algorithm 1 through Algorithm 5. Here, the former two are auxiliary algorithms performing tasks necessary in both WOTS-TW$^\$$ and, as defined in the subsequent sections, (fixed-length) XMSS-TW; the latter three constitute the key generation, signature, and verification algorithms, respectively.

Security Property. For the security property of WOTS-TW$^\$$, we consider *Multi-instance Existential UnForgeability under Generic Chosen-Message Attack* (M-EUF-GCMA),[9] a variant of the EUF-CMA property that was specifically devised to recover the tight security proof of SPHINCS$^+$ [HK22]. Intuitively, this property captures the feasibility of forging a signature for any of several WOTS-TW$^\$$ instances after obtaining a signature (and corresponding public key) on an adaptively chosen address-message pair for each considered instance. Crucially, the public key of a WOTS-TW$^\$$ instance is only given to the adversary after it issued the signature/challenge query for that instance. The forged signature should be valid with respect to the same address as the observed signature for that instance of WOTS-TW$^\$$. The game that formalizes this property, parameterized on the considered THF collection THFC and the number of WOTS-TW$^\$$ instances d, is given in Figure 11; the oracles provided to the adversary in this game are specified in Figure 12 (signature/challenge oracle) and Figure 9 (collection oracle). As per usual, the adversary is merely given access to the Query procedures of these oracles.

Akin to the games formalizing the THF properties, $\mathrm{Game}^{\text{M-EUF-GCMA}}_{\mathcal{A},\text{WOTS-TW}^\$,\mathsf{THFC},d}$ is defined with respect to a two-stage adversary: In the first stage, this adversary is asked to select (up to) d target addresses, receiving corresponding signatures on chosen messages, while being able to query the considered (indexed) THF collection; in the second stage, given the public seed used to index F (in WOTS-TW$^\$$) and the considered THF collection, this adversary is asked to provide a signature on a fresh message that is valid with respect to one of the targets specified in the first stage. In the end, in addition to the legitimacy of the forgery, the adver-

[9] In [HK22], the authors introduce this property as D-EF-naCMA.

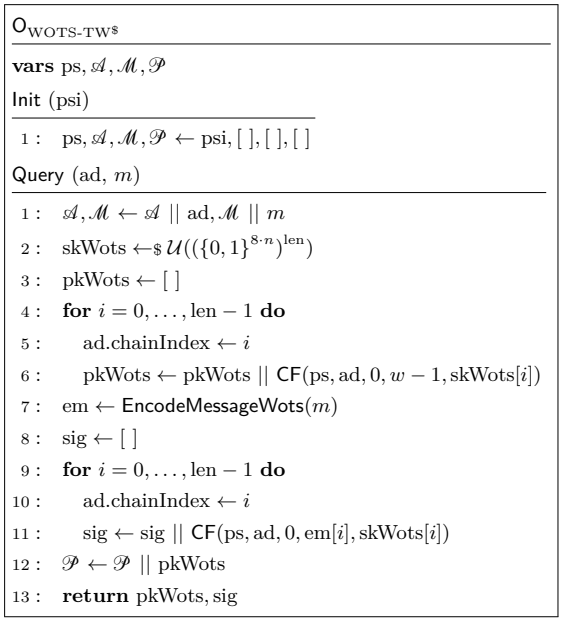

Figure 12. Signature/Challenge oracle employed in M-EUF-GCMA game for WOTS-TW$^\$$.

sary's behavior throughout the game is validated; this validation is performed by the $\text{VAD}_d(\mathsf{O}_{\text{WOTS-TW}^\$}.\mathscr{A}, \mathsf{OC}_{\mathsf{THFC}}.\mathscr{A})$ predicate, checking whether the number of specified target addresses was at most d, whether the target addresses were unique with respect to the part that can be used to differentiate between instances of WOTS-TW$^\$$ — i.e., the part excluding the aforementioned chain index and hash index — and whether the target addresses were never used in queries to the collection oracle. Then, the advantage of any adversary $\mathcal{A}$ against M-EUF-GCMA (of WOTS-TW$^\$$) is defined as follows.

$$\mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^\$,\mathsf{THFC},d}(\mathcal{A}) = \Pr\left[\text{Game}^{\text{M-EUF-GCMA}}_{\mathcal{A},\text{WOTS-TW}^\$,\mathsf{THFC},d} = 1\right]$$

Formal Verification. We presently discuss the (proof of) the security statement for WOTS-TW$^\$$ that we formally verify in this work. As depicted in Figure 10, we aim to demonstrate an implication from (the conjunction of) the SM-DT-UD-C, SM-DT-TCR-C, and SM-DT-PRE-C properties of F to the M-EUF-GCMA property of WOTS-TW$^\$$. More formally, the security theorem we prove is the following.

Security Theorem 1 (M-EUF-GCMA for WOTS-TW$^\$$)**.** *For any adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}_0$, $\mathcal{B}_1$, and $\mathcal{B}_2$ — each with approximately the same running time as $\mathcal{A}$ — such that the following inequality holds.*

$$\mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^\$,\mathsf{FC},d}(\mathcal{A}) \leq (w-2) \cdot \mathsf{Adv}^{\text{SM-DT-UD-C}}_{\mathsf{F},\mathsf{FC},t_{\text{udf}}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{F},\mathsf{FC},t_{\text{tcrf}}}(\mathcal{B}_1)$$

$$+ \mathsf{Adv}^{\text{SM-DT-PRE-C}}_{\mathsf{F},\mathsf{FC},t_{\text{pref}}}(\mathcal{B}_2)$$

Here, FC *denotes an arbitrary THF collection containing* F*,* $d \geq 1$*,* $t_{\text{udf}} = d \cdot \text{len}$*,* $t_{\text{tcrf}} = d \cdot \text{len} \cdot w$*, and* $t_{\text{pref}} = d \cdot \text{len}$*.*

Conceptually, the formal verification of the above security theorem closely follows the original proof presented in [HK22]. Specifically, the formal verification considers a sequence of two games; in order, we denote these games by $\text{Game}^0_{\mathcal{A}}$ and $\text{Game}^1_{\mathcal{A}}$. Both of these games only differ from $\text{Game}^{\text{M-EUF-GCMA}}_{\mathcal{A},\text{WOTS-TW}^\$,\mathsf{FC},d}$ — and, hence, each other — with respect to the Query procedure of the challenge oracle provided to the adversary. As such, the ensuing exposition of the proof predominantly focuses on the challenge oracles instead of the games. The advantage of any adversary $\mathcal{A}$ playing in $\text{Game}^i_{\mathcal{A}}$, $i \in \{0,1\}$, is defined similarly to $\mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^\$,\mathsf{FC},d}(\mathcal{A})$; we refer to such an advantage as $\mathsf{Adv}^i(\mathcal{A})$. Imminently, we relate or bound (the differences between) advantages obtained in the games of the game sequence. Afterward, we combine the obtained results to acquire the aforementioned implication from the desired properties of F to the M-EUF-GCMA property of WOTS-TW$^\$$.

Relation Between $\mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^\$,\mathsf{FC},d}(\mathcal{A})$ *and* $\mathsf{Adv}^0(\mathcal{A})$*.* As hinted at above, the first game in the game sequence, $\text{Game}^0_{\mathcal{A}}$, only differs from $\text{Game}^{\text{M-EUF-GCMA}}_{\mathcal{A},\text{WOTS-TW}^\$,\mathsf{FC},d}$ in the Query procedure of the challenge oracle. Namely, first, the order of the construction of the public key and signature is reversed; second, the public key is constructed by finishing the function chains based on the signature instead of computing the complete function chains based on the secret key. Figure 13 provides the specification of the resulting oracle procedure. Indeed, comparing $\mathsf{O}_{\text{WOTS-TW}^\$}.\mathsf{Query}$ and $\mathsf{O}_0.\mathsf{Query}$, we see that the for-loops (and the preceding initialization of the variables used in these loops) concerning the computation of the public key pkWots and signature sig are swapped. Furthermore, rather than computing the i-th element of pkWots immediately as $\mathsf{CF}(\text{ps}, \text{ad}, 0, w - 1, \text{skWots}[i])$, as $\mathsf{O}_0.\mathsf{Query}$ does, $\mathsf{O}_1.\mathsf{Query}$ computes it as $\mathsf{CF}(\text{ps}, \text{ad}, \text{em}[i], w - 1 - \text{em}[i], \text{sig}[i])$, where $\text{sig}[i]$ equals $\mathsf{CF}(\text{ps}, \text{ad}, 0, \text{em}[i], \text{skWots}[i])$. However, from the compositional property of chaining functions (see Section 2), it follows that $\mathsf{CF}(\text{ps}, \text{ad}, \text{em}[i], w - 1 - \text{em}[i], \mathsf{CF}(\text{ps}, \text{ad}, 0, \text{em}[i], \text{skWots}[i])) = \mathsf{CF}(\text{ps}, \text{ad}, 0, w - 1, \text{skWots}[i])$; as such, the different computations of the public key are equivalent. Then, the two Query procedures and, in turn, $\text{Game}^{\text{M-EUF-GCMA}}_{\mathcal{A},\text{WOTS-TW}^\$,\mathsf{FC},d}$ and $\text{Game}^0_{\mathcal{A}}$ are semantically equivalent. In consequence, we can derive the following result.

$$\forall_{\mathcal{A}} : \ \mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^\$,\mathsf{FC},d}(\mathcal{A}) = \mathsf{Adv}^0(\mathcal{A})$$

Bound on Difference Between $\mathsf{Adv}^0(\mathcal{A})$ *and* $\mathsf{Adv}^1(\mathcal{A})$*.* As $\text{Game}^{\text{M-EUF-GCMA}}_{\mathcal{A},\text{WOTS-TW}^\$,\mathsf{FC},d}$ and $\text{Game}^0_{\mathcal{A}}$, $\text{Game}^0_{\mathcal{A}}$ and $\text{Game}^1_{\mathcal{A}}$ exclusively differ in the Query procedure of their challenge oracles, the specifications of which are provided in Figure 13. Collating these procedures, we see that their disparity solely concerns the generation of the signature: $\mathsf{O}_0.\mathsf{Query}$ properly constructs the signature by applying the chaining function on each secret key element for the number of iterations indicated by the corresponding element of the encoded message; $\mathsf{O}_1.\mathsf{Query}$ merely

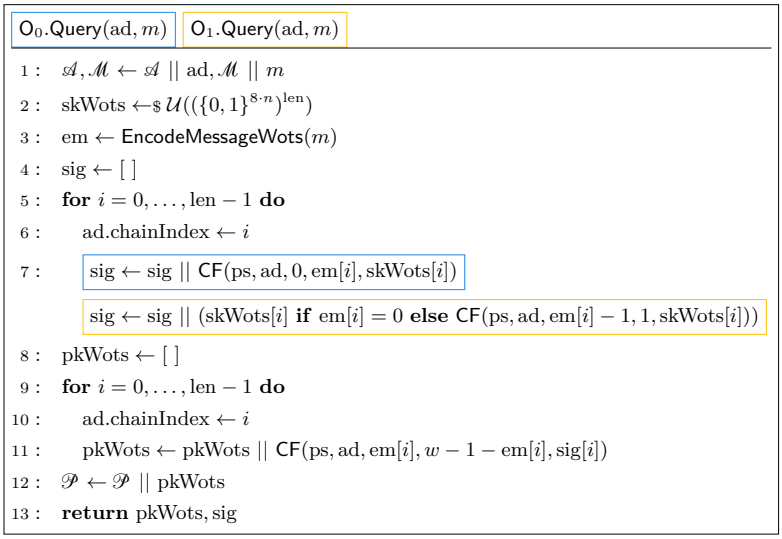

Figure 13. Query procedures of the challenge oracles employed in $\mathrm{Game}^0_{\mathcal{A}}$ (blue boxes) and $\mathrm{Game}^1_{\mathcal{A}}$ (yellow boxes). Statements not within colored boxes are executed in both procedures.

performs the final iteration of each of these applications of the chaining function. Here, remember that the chaining function essentially reduces to the identity function whenever the iteration counter is less than or equal to zero. As such, in both procedures, $\mathrm{sig}[i]$ equals $\mathrm{skWots}[i]$ if $\mathrm{em}[i] = 0$, where $0 \leq i < \mathrm{len}$.

Considering their difference, distinguishing between $\mathrm{Game}^0_{\mathcal{A}}$ and $\mathrm{Game}^1_{\mathcal{A}}$ intuitively boils down to distinguishing between, for any message and address (and uniformly random public seed), the signature distribution resulting from applying the chaining function the appropriate number of times on the elements of a uniformly random secret key, and the distribution resulting from only applying the final iteration of the chaining function on the elements of a uniformly random secret key. Surely, these distributions should be computationally indistinguishable if, for any address (and uniformly random public seed), the output of the chaining function — when applied on a uniformly random value — remains computationally indistinguishable from a uniformly random value for up to $w - 2$ iterations. Namely, this would imply that, irrespective of the value of $\mathrm{em}[i]$, $\mathsf{CF}(\mathrm{ps}, \mathrm{ad}, 0, \mathrm{em}[i] - 1, \mathrm{skWots}[i])$ is computationally indistinguishable from $\mathrm{skWots}[i]$. Indeed, this is closely related to the SM-DT-UD-C property we assume F to possess; in fact, by means of a hybrid argument based on the number of omitted initial applications of F in a call to CF, we can reduce this property to distinguishing between $\mathrm{Game}^0_{\mathcal{A}}$ and $\mathrm{Game}^1_{\mathcal{A}}$. More precisely, given an adversary $\mathcal{A}$ playing in $\mathrm{Game}^0_{\mathcal{A}}$ and $\mathrm{Game}^1_{\mathcal{A}}$, we can construct a reduction adversary $\mathcal{R}^{\mathcal{A}}$ playing in $\mathrm{Game}^{\text{SM-DT-UD-C}}_{\mathcal{R}^{\mathcal{A}},\mathsf{F},\mathsf{FC},t_{\mathrm{udf}}}(b)$ that samples $i \in [0, w-3]$ uniformly at random, constructs either the i-th or $i+1$-th hybrid — depending on whether its challenge oracle returns mappings of uniformly random values or returns uniformly

random values, respectively — and employs $\mathcal{A}$ to determine which hybrid it is, thereby achieving a related advantage in its own game. As a result, abstracting away the particular reduction adversary, we obtain the following bound.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_0} : \left|\mathsf{Adv}^0(\mathcal{A}) - \mathsf{Adv}^1(\mathcal{A})\right| \leq (w-2) \cdot \mathsf{Adv}^{\text{SM-DT-UD-C}}_{\mathsf{F},\mathsf{FC},t_{\text{udf}}}(\mathcal{B}_0)$$

Bound on $\mathsf{Adv}^1(\mathcal{A})$. In the situation where an adversary playing in $\text{Game}_1^{\mathcal{A}}$ returns a valid forgery, it *must* be the case that this forgery allows for the extraction of a collision or a preimage for F. Namely, a forgery in $\text{Game}_1^{\mathcal{A}}$ is a valid signature sig' on a fresh message m' under a previously established public key pkWots, address ad and public seed ps. That is, there already exists a valid signature sig on another message m (different from m') under this public key, address, and public seed. As m' and m are different, the message encoding guarantees that there exists an i, $0 \leq i < \text{len}$, such that $\text{em}'[i] < \text{em}[i]$ (where em' and em respectively denote the encodings of m' and m); consequently, for such an i, the verification algorithm performs more iterations of the chaining function on $\text{sig}'[i]$ than it performs on $\text{sig}[i]$. Nevertheless, since sig and sig' are both valid under pkWots, both function chains must result in $\text{pkWots}[i]$. As such, we can distinguish two cases: $\mathsf{CF}(\text{ps}, \text{ad}, \text{em}'[i], \text{em}[i] - \text{em}'[i], \text{sig}'[i]) \neq \text{sig}[i]$ and $\mathsf{CF}(\text{ps}, \text{ad}, \text{em}'[i], \text{em}[i] - \text{em}'[i], \text{sig}'[i]) = \text{sig}[i]$. In the former case, the value of the function chain of $\text{sig}'[i]$ at the same iteration as $\text{sig}[i]$ does not equal $\text{sig}[i]$. However, from both of these values, $\text{pkWots}[i]$ can be obtained by completing the function chains in an identical manner as, at this point, the same (number of) iterations remain for both function chains. Following, since at some point along the remainder of the function chains the output of F must become equal, the values in the function chains preceding these equal outputs constitute a collision for F. In the other case, the value of the function chain of $\text{sig}'[i]$ at the same iteration as $\text{sig}[i]$ does equal $\text{sig}[i]$. Then, the value in the function chain of $\text{sig}'[i]$ directly preceding the value at the same iteration as $\text{sig}[i]$ constitutes a preimage of $\text{sig}[i]$ under F.

In line with the above reasoning, we can construct reduction adversaries that are witnesses for the following bound.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_1,\mathcal{B}_2} : \mathsf{Adv}^1(\mathcal{A}) \leq \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{F},\mathsf{FC},t_{\text{tcrf}}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{SM-DT-PRE-C}}_{\mathsf{F},\mathsf{FC},t_{\text{pref}}}(\mathcal{B}_2)$$

Final Result. Combining the foregoing results, we can derive Security Theorem 1 as follows.

$$\begin{aligned}
&\forall_{\mathcal{A}} \exists_{\mathcal{B}_0,\mathcal{B}_1,\mathcal{B}_2} : \mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^{\$},\mathsf{FC},d}(\mathcal{A}) = \mathsf{Adv}^0(\mathcal{A}) \leq \\
&\left|\mathsf{Adv}^0(\mathcal{A}) - \mathsf{Adv}^1(\mathcal{A})\right| + \mathsf{Adv}^1(\mathcal{A}) \leq \\
&(w-2) \cdot \mathsf{Adv}^{\text{SM-DT-UD-C}}_{\mathsf{F},\mathsf{FC},t_{\text{udf}}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{F},\mathsf{FC},t_{\text{tcrf}}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{SM-DT-PRE-C}}_{\mathsf{F},\mathsf{FC},t_{\text{pref}}}(\mathcal{B}_2)
\end{aligned}$$

Although no formal runtime analysis is provided, it is evident from the prior discussion (and from the EasyCrypt artifacts) that there exists $\mathcal{B}_0$, $\mathcal{B}_1$, and $\mathcal{B}_2$ that not only satisfy the above inequality, but also execute in approximately the same time as $\mathcal{A}$. This completes the formal verification of the fix of the security proof of SPHINCS$^+$ presented in [HK22].

5 Fixed-Length XMSS-TW

Fixed-length XMSS-TW builds on WOTS-TW and is a component of SPHINCS$^+$. Conceptually, fixed-length XMSS-TW constitutes a binary hash tree, or Merkle tree, that employs WOTS-TW as its one-time signature scheme and exclusively processes messages of some fixed length; in fact, this fixed length matches the fixed length of the messages processed by WOTS-TW. As briefly elaborated on in Section 4, WOTS-TW only differs from WOTS-TW$^\$$ concerning the generation and handling of the secret keys. More precisely, rather than sampling the secret key uniformly at random and maintaining it in its entirety, WOTS-TW merely maintains a secret seed — an element from the *secret seed space* $\mathcal{SS}$ — (re)generating the secret key via a PRF each time it is required. For the purpose of (re)generating the secret key, WOTS-TW specifies an additional algorithm, provided in Algorithm 6. In this algorithm, SKWG is a KHF of which the key space and message space are instantiated with $\mathcal{SS}$ and $\mathcal{PS} \times \mathcal{AD}$, respectively. Then, the remainder of the algorithms of WOTS-TW are analogous to the algorithms of WOTS-TW$^\$$; as such, we refer to them using the same identifiers, yet preceded with WOTS-TW instead of WOTS-TW$^\$$.

Foremost, we go over several additional preliminaries. Besides the parameters required for WOTS-TW, fixed-length XMSS-TW is defined with respect to a parameter h that signifies the height of the tree. From h, since fixed-length XMSS-TW constitutes a Merkle tree, we can compute the number of leaves as $l = 2^h$. Furthermore, in addition to the THF employed in WOTS-TW, XMSS-TW utilizes two THFs: one for the compression of WOTS-TW public keys to leaves — denoted by PKCO — and one for the construction of the tree from the leaves — denoted by TRC. For both of these THFs, the parameter space and tweak space are, respectively, $\mathcal{PS}$ and $\mathcal{AD}$, identical to those of F. Nevertheless, as fixed-length XMSS-TW constitutes a larger structure than WOTS-TW, we require the addresses from $\mathcal{AD}$ to — on top of the previously introduced chain index and hash index required by WOTS-TW — contain a *type index*, *key pair index*, *tree height index*, and *tree breadth index*. These indices are nonnegative integers that, in order, indicate the considered type of operation (either function chaining, public key compression, or tree construction), the considered WOTS-TW key pair (or leaf), the height of the considered tree node, and the breadth of the considered tree node (at the height indicated by the tree height index). Here, the key pair index is only used for the function chaining and public key compression operations, and the tree height and tree breadth indices are only used for the tree construction operation; moreover, the chain and hash indices are only used for the function chaining operation.[10] Finally, besides these indices, the addresses may comprise other nonnegative integers that, e.g., guarantee the uniqueness of the addresses when (fixed-length) XMSS-TW is considered in an encompassing structure such as SPHINCS$^+$. As before, we leave these unspecified.

[10] Consequently, in practice, it may be the case that, e.g., the chain index and the tree height index refer to the same location of an address.

In fixed-length XMSS-TW, key pairs are, intuitively, constructed as follows. Foremost, a secret key is a four-tuple $\mathrm{sk} = (i, \mathrm{ss}, \mathrm{ps}, \mathrm{ad})$, where $i \in [0, l-1]$, $\mathrm{ss} \in \mathcal{SS}$, $\mathrm{ps} \in \mathcal{PS}$, and $\mathrm{ad} \in \mathcal{AD}$. Here, i indicates which WOTS-TW key pair is supposed to be used for the construction of the next signature. Then, the public key associated with sk is produced by, first, generating a sequence of l WOTS-TW secret keys via SKWG. Subsequently, the corresponding sequence of WOTS-TW public keys is computed and, in turn, transformed into a sequence of leaves by means of PKCO. Now, this sequence of leaves uniquely defines a Merkle tree of which we can obtain the root by iteratively computing each of the tree's layers. Specifically, in the construction of the layer at height thi, the node at breadth tbi is computed from its children cl and cr as $\mathsf{TRC}(\mathrm{ps}, \mathrm{ad}_{\mathrm{thi,tbi}}, \mathrm{cl} \,||\, \mathrm{cr})$, where $\mathrm{ad}_{\mathrm{thi,tbi}}$ signifies the address resulting from modifying ad to be the unique address for the node at height thi and breadth tbi. After obtaining the root rt, the public key is defined as the three-tuple $\mathrm{pk} = (\mathrm{rt}, \mathrm{ps}, \mathrm{ad})$. Henceforth, we denote the operator that performs this root computation by $\mathsf{RootFromLeaves}$.

Given a key pair $(\mathrm{pk}, \mathrm{sk})$ constructed as described above and a message $m \in \{0,1\}^{8 \cdot n}$, signatures are, on a high level, created and verified as follows. First, using the i-th WOTS-TW secret key from the aforementioned sequence (recall that i is part of sk), a WOTS-TW signature sigWots on m is produced. Next, a so-called *authentication path* is computed for the i-th leaf of the Merkle tree. This path is a sequence of nodes that, in order, comprises the siblings of the nodes on the path from the root to the i-th leaf. Hereafter, we denote the operator that computes this authentication path by $\mathsf{AuthPath}$. Given the authentication path ap, the signature on m is the three-tuple $\mathrm{sig} = (i, \mathrm{sigWots}, \mathrm{ap})$. Verification of sig is performed by, first, computing the WOTS-TW public key pkWots corresponding to sigWots and compressing it to a leaf lf using PKCO. Afterward, a candidate root value rt' for the Merkle tree is computed from lf and ap. Indeed, this is achieved by reconstructing the path from the i-th leaf to the root by using lf and the sibling nodes in ap. For example, if the i-th leaf is a left child, the second node on the path is reconstructed as $p_1 = \mathsf{TRH}(\mathrm{ps}, \mathrm{ad}_{1,j}, \mathrm{lf} \,||\, \mathrm{ap}[h-1])$, where $j = \lfloor i/2 \rfloor$; then, if p_1 is a right child, the third node on the path is reconstructed as $p_2 = \mathsf{TRH}(\mathrm{ps}, \mathrm{ad}_{2,k}, \mathrm{ap}[h-2] \,||\, p_1)$, where $k = \lfloor j/2 \rfloor$; et cetera.[11] Throughout the remainder, we denote the operator that performs this computation of a candidate root by $\mathsf{RootFromAuthPath}$. Finally, if rt' equals rt, verification succeeds; otherwise, verification fails.

Following the above description, the specification of fixed-length XMSS-TW is provided in Algorithm 7 through Algorithm 10. Here, the former is an auxiliary algorithm for the construction of the leaves from the secret seed, public seed, and address of the secret key; the latter three constitute the actual key generation, signing, and verification algorithms, respectively. In these algorithms, and from this point onward, we explicitly refer to fixed-length XMSS-TW as FL-XMSS-TW to prevent potential ambiguity with (standalone) XMSS-TW.

[11] Whether the nodes along the reconstructed path are left or right children can be determined from the value of i.

Algorithm 6 WOTS-TW's Secret Key Generation Algorithm

procedure WOTS-TW.$\mathsf{SkWotsGen}$(ss, ps, ad)
  skWots $\leftarrow$ [ ]
  for $i = 0, \ldots, \text{len} - 1$ **do**
    ad.chainIndex, ad.hashIndex $\leftarrow i, 0$
    skWots $\leftarrow$ skWots || SKWG(ss, (ps, ad))
  return skWots

Algorithm 7 FL-XMSS-TW's Leaves From Secret Key Algorithm

procedure FL-XMSS-TW.$\mathsf{LeavesFromSk}$(ss, ps, ad)
  leaves $\leftarrow$ [ ]
  for $i = 0, \ldots, l - 1$ **do**
    ad.typeIndex, ad.keypairIndex $\leftarrow$ chainType, i
    skWots $\leftarrow$ WOTS-TW.$\mathsf{SkWotsGen}$(ss, ps, ad)
    pkWots $\leftarrow$ WOTS-TW.$\mathsf{PkWotsFromSkWots}$(skWots, ps, ad)
    ad.typeIndex $\leftarrow$ compressionType
    leaves $\leftarrow$ leaves || PKCO(ps, ad, pkWots)
  return leaves

Algorithm 8 FL-XMSS-TW's Key Generation Algorithm

procedure FL-XMSS-TW.KeyGen(ss, ps, ad)
  leaves $\leftarrow$ FL-XMSS-TW.$\mathsf{LeavesFromSk}$(ss, ps, ad)
  ad.typeIndex $\leftarrow$ treeType
  rt $\leftarrow$ $\mathsf{RootFromLeaves}$(leaves, ps, ad)
  return pk := (rt, ps, ad), sk := (0, ss, ps, ad)

Algorithm 9 FL-XMSS-TW's Signing Algorithm

procedure FL-XMSS-TW.Sign(sk := $(i$, ss, ps, ad), $m)$
  ad.typeIndex, ad.keypairIndex $\leftarrow$ chainType, i
  sigWots $\leftarrow$ WOTS-TW.Sign((ss, ps, ad), m)
  leaves $\leftarrow$ FL-XMSS-TW.$\mathsf{LeavesFromSk}$(ss, ps, ad)
  ad.typeIndex $\leftarrow$ treeType
  ap $\leftarrow$ $\mathsf{AuthPath}$(i, leaves, ps, ad)
  return sig := $(i$, sigWots, ap), sk := $(i + 1$, ss, ps, ad)

Algorithm 10 FL-XMSS-TW's Verification Algorithm

procedure FL-XMSS-TW.Verify(pk := (rt, ps, ad), m, sig := $(i$, sigWots, ap))
  ad.typeIndex, ad.keypairIndex $\leftarrow$ chainType, i
  pkWots $\leftarrow$ WOTS-TW.$\mathsf{PkWotsFromSig}$(m, sigWots, ps, ad)
  ad.typeIndex $\leftarrow$ compressionType
  lf $\leftarrow$ PKCO(ps, ad, pkWots)
  rt$'$ $\leftarrow$ $\mathsf{RootFromAuthPath}$($i$, lf, ap)
  return rt$'$ = rt

Security Property. As hinted at in Section 3, the security property we consider for FL-XMSS-TW is the EUF-RMA property. However, to be more precise,

$\mathrm{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}}$

1 : $\mathrm{ss} \leftarrow\!\$\, \mathcal{U}(\mathcal{SS})$
2 : $\mathrm{ps} \leftarrow\!\$\, \mathcal{U}(\mathcal{PS})$
3 : $\mathrm{ad} \leftarrow \mathcal{A}.\mathsf{Choose}()$
4 : $\mathrm{pk}, \mathrm{sk} \leftarrow \text{FL-XMSS-TW}.\mathsf{KeyGen}(\mathrm{ss}, \mathrm{ps}, \mathrm{ad})$
5 : $\mathrm{ms}, \mathrm{sigs} \leftarrow [\,], [\,]$
6 : **for** $i = 0, \ldots, l-1$ **do**
7 : $\quad m \leftarrow\!\$\, \mathcal{U}(\{0,1\}^{8 \cdot n})$
8 : $\quad \mathrm{sig}, \mathrm{sk} \leftarrow \text{FL-XMSS-TW}.\mathsf{Sign}(\mathrm{sk}, m)$
9 : $\quad \mathrm{ms}, \mathrm{sigs} \leftarrow \mathrm{ms} \,||\, m, \mathrm{sigs} \,||\, \mathrm{sig}$
10 : $m', \mathrm{sig}' \leftarrow \mathcal{A}.\mathsf{Forge}(\mathrm{pk}, \mathrm{ms}, \mathrm{sigs})$
11 : $\mathrm{isValid} \leftarrow \text{FL-XMSS-TW}.\mathsf{Verify}(\mathrm{pk}, m', \mathrm{sig})$
12 : $\mathrm{isFresh} \leftarrow m' \notin ms$
13 : **return** $\mathrm{isValid} \wedge \mathrm{isFresh}$

Figure 14. EUF-RMA game for FL-XMSS-TW.

we actually consider a minor variant of this property that accounts for the fact that FL-XMSS-TW operates on an address that is provided by the environment and, besides having (valid values for) the previously described indices, may be arbitrarily structured.[12] Ensuring FL-XMSS-TW possesses the desired security property regardless of the additional components of the provided address, the variant of EUF-RMA we consider is defined with respect to a two-stage adversary that selects the address to be used in its first stage, and only attempts to provide a forgery in its second stage. Figure 14 provides the game formalizing this property. Then, the advantage of any adversary $\mathcal{A}$ against EUF-RMA (of FL-XMSS-TW) is defined as follows.

$$\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A}) = \Pr\left[\mathrm{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}} = 1\right]$$

Formal Verification. We now go over the (proof of) the security statement concerning FL-XMSS-TW that we formally verify in this work. As illustrated in Figure 10, we aim to show that the EUF-RMA property of FL-XMSS-TW is implied by the PRF property of SKWG, the M-EUF-GCMA property of WOTS-TW$^{\$}$, and the SM-DT-TCR-C property of PKCO and TRC. Specifically, we formally verify the following security theorem.

Security Theorem 2 (EUF-RMA for FL-XMSS-TW)**.** *For any adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}_0$, $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ — each with approximately the same running time as $\mathcal{A}$ — such that the following inequality holds.*

$$\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{PRF}}_{\mathsf{SKWG}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^{\$},\mathsf{THFC},l}(\mathcal{B}_1)$$

[12] For example, as previously mentioned, an address may contain additional indices that differentiate the context in an encompassing structure.

$$+ \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{PKCO},\mathsf{THFC},l}(\mathcal{B}_2)$$
$$+ \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRC},\mathsf{THFC},l-1}(\mathcal{B}_3)$$

Here, THFC *denotes an arbitrary THF collection containing* F, PKCO, *and* TRC.

The formal verification of Theorem 2 proceeds as follows. Foremost, we consider FL-XMSS-TW$^\$$ instead of FL-XMSS-TW to obtain $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$. Here, FL-XMSS-TW$^\$$ is analogous to WOTS-TW$^\$$ in that, rather than (re)generating the WOTS-TW secret keys via SKWG whenever necessary, it samples these keys uniformly at random and directly takes them as input whenever they are required. Alternatively stated, FL-XMSS-TW$^\$$ is obtained from FL-XMSS-TW by replacing the call to WOTS-TW.SkWotsGen in FL-XMSS-TW.LeavesFromSk with the appropriate sampling operation, and replacing the calls to the remaining WOTS-TW procedures with their WOTS-TW$^\$$ analogs. Given these differences, we reduce from the PRF property of SKWG to distinguishing between $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}}$ and $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$. Afterward, considering the situation in which $\mathcal{A}$ returns a valid forgery in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$, we perform a case analysis, allowing us to rephrase the corresponding probability as a sum of several terms. Subsequently, we bound each of these terms by providing a reduction from either the M-EUF-GCMA property of WOTS-TW$^\$$, or the SM-DT-TCR-C property of PKCO or TRC. Altogether, this suffices to derive the desired result.

Bound on Difference Between $\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A})$ *and* $\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}^\$}(\mathcal{A})$. As alluded to above, the sole semantic difference between $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}}$ and $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$ regards the manner in which the secret key is obtained: In FL-XMSS-TW, the secret key is (re)generated via SKWG each time it is required; in FL-XMSS-TW$^\$$, the secret key is sampled uniformly at random, maintained as is, and reused whenever it is required. Ergo, given an adversary $\mathcal{A}$ playing in these games, we can straightforwardly construct a reduction adversary $\mathcal{R}^{\mathcal{A}}$ achieving an advantage in $\text{Game}^{\text{PRF}}_{\mathcal{R}^{\mathcal{A}},\mathsf{SKWG}}(b)$ that equals the (absolute) difference between $\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A})$ and $\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}^\$}(\mathcal{A})$. Generalizing this result, we acquire the following bound.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_0} : \left| \mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A}) - \mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}^\$}(\mathcal{A}) \right| \leq \mathsf{Adv}^{\mathsf{SKWG}}_{\text{PRF}}(\mathcal{B}_0)$$

Case Distinction for $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$} = 1$. Considering the situation where an adversary playing in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$ provides a valid forgery, we can distinguish three (exhaustive) cases. Namely, a valid forgery in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$ consists of a message m' and a signature $\text{sig}' = (i', \text{sigWots}', \text{ap}')$ such that m' is fresh and sig' is valid for m' under the considered public key $\text{pk} = (\text{rt}, \text{ps}, \text{ad})$. Now, recall that sig' being valid for m' under pk means that the candidate root rt' equals the actual root rt. As such, at a certain point along the computation of rt', the considered values should coincide with the corresponding values used in the computation of rt. Building on this observation, the first case we distinguish concerns pkWots', the WOTS-TW$^\$$ public key corresponding to $\text{sigWots}'$,

coinciding with $\text{pkWots}_{i'}$, the i'-th WOTS-TW$^{\$}$ public key in the sequence used during key generation and, hence, the computation of rt. As per the verification procedure of WOTS-TW$^{\$}$, this means that $\text{sigWots}'$ is a valid signature for m' under $\text{pkWots}_{i'}$; since m' is fresh, it follows that m' and $\text{sigWots}'$ form a valid forgery for the WOTS-TW$^{\$}$ instance corresponding to the i'-th leaf of the Merkle tree. Henceforth, we denote the event that this first case occurs by E_F. Then, if pkWots' does not equal $\text{pkWots}_{i'}$, the second case we distinguish regards the leaves resulting from the compression of these WOTS-TW$^{\$}$ public keys coinciding. In this case, the inputs of PKCO are unequal yet map, under the same public seed and address, to the same output. Consequently, pkWots' and $\text{pkWots}_{i'}$ constitute a collision for PKCO. Hereafter, E_P signifies the event that this second case occurs. Lastly, if both of the preceding cases do not happen, it must be the case that, from a certain point onward, the reconstructed path coincides with the corresponding path through the original Merkle tree. Following, since the initial (few) nodes along the paths *do not* coincide, the first node for which the paths *do* coincide must be obtained by applying TRC, with the same public seed and address, on different inputs. As such, these inputs form a collision for TRC.

Formally, the foregoing can essentially be summarized by the following, where $G_{\mathcal{A}}$ serves as a shorthand for $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^{\$}}$.

$$\begin{aligned}\forall_{\mathcal{A}}: \Pr[G_{\mathcal{A}} = 1] &= \\ &\Pr[G_{\mathcal{A}} = 1 \wedge E_F] + \Pr[G_{\mathcal{A}} = 1 \wedge \neg E_F] = \\ &\Pr[G_{\mathcal{A}} = 1 \wedge E_F] + \Pr[G_{\mathcal{A}} = 1 \wedge \neg E_F \wedge E_P] + \Pr[G_{\mathcal{A}} = 1 \wedge \neg E_F \wedge \neg E_P]\end{aligned}$$

Bound on $\Pr\left[\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^{\$}} = 1 \wedge E_F\right]$. For the case where the (valid) forgery returned by $\mathcal{A}$ in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^{\$}}$ comprises a valid forgery for the indicated WOTS-TW$^{\$}$ instance, we can devise a reduction adversary $\mathcal{R}^{\mathcal{A}}$ playing in $\text{Game}^{\text{M-EUF-GCMA}}_{\mathcal{R}^{\mathcal{A}},\mathsf{F},\mathsf{THFC},l}$ as follows. Foremost, $\mathcal{R}^{\mathcal{A}}$ produces a sequence of l WOTS-TW$^{\$}$ signatures and public keys by repeatedly querying its challenge oracle on an appropriately updated address (based on the address obtained from $\mathcal{A}$) and a uniformly random message.[13] Then, utilizing the public seed provided in its second stage, the reduction adversary finishes the generation of, and provides $\mathcal{A}$ with, the FL-XMSS-TW$^{\$}$ signatures and public key corresponding to the previously produced sequence of WOTS-TW$^{\$}$ signatures and public keys. Finally, as soon as $\mathcal{A}$ returns a (valid) forgery — comprised of, say, message m' and signature $\text{sig}' = (i', \text{sigWots}', \text{ap}')$ — $\mathcal{R}^{\mathcal{A}}$ straightforwardly extracts and returns i', m', and $\text{sigWots}'$.

Based on the above, we can derive the ensuing result.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_1}: \Pr\left[\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^{\$}} = 1 \wedge E_F\right] \leq \mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^{\$},\mathsf{THFC},l}(\mathcal{B}_1)$$

[13] As such, it suffices to consider l simultaneous WOTS-TW$^{\$}$ instances and, accordingly, only allow l queries to the challenge oracle.

Bound on $\Pr\left[\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$} = 1 \wedge \neg E_F \wedge E_\text{P}\right]$. In the case that the (valid) forgery provided by $\mathcal{A}$ in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$ does not contain a WOTS-TW$^\$$ forgery but does allow for the extraction of a collision for PKCO, we consider the ensuing reduction adversary $\mathcal{R}^{\mathcal{A}}$ playing in $\text{Game}^{\text{SM-DT-TCR-C}}_{\mathcal{R}^{\mathcal{A}},\mathsf{PKCO},\mathsf{THFC},l}$. First, $\mathcal{R}^{\mathcal{A}}$ samples a sequence of l WOTS-TW$^\$$ secret keys uniformly at random and, subsequently, computes the corresponding public keys by continually querying its collection oracle on a properly updated address (based on the address provided by $\mathcal{A}$) and a function chain element. Afterward, the reduction adversary produces the corresponding sequence of leaves through its challenge oracle, thereby specifying every WOTS-TW$^\$$ public key as a collision target.[14] Then, employing the public seed provided in its second stage, $\mathcal{R}^{\mathcal{A}}$ constructs, and provides $\mathcal{A}$ with, the FL-XMSS-TW$^\$$ signatures (on uniformly random messages) and public key corresponding to the formerly obtained sequences. Lastly, when $\mathcal{A}$ returns a (valid) forgery — consisting of, say, message m' and signature $\text{sig}' = (i', \text{sigWots}', \text{ap}')$ — the reduction adversary computes the WOTS-TW$^\$$ public key corresponding to sigWots$'$ and returns it together with i'.

From the preceding, we can deduce the following bound.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_2} : \Pr\left[\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$} = 1 \wedge \neg E_F \wedge E_P\right] \leq \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{PKCO},\mathsf{THFC},l}(\mathcal{B}_2)$$

Bound on $\Pr\left[\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$} = 1 \wedge \neg E_F \wedge \neg E_P\right]$. In the final case, the (valid) forgery provided by $\mathcal{A}$ in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^\$}$ allows for the extraction of a collision for TRC; so, we consider a reduction adversary $\mathcal{R}^{\mathcal{A}}$ playing in $\text{Game}^{\text{SM-DT-TCR-C}}_{\mathcal{R}^{\mathcal{A}},\mathsf{PKCO},\mathsf{THFC},l}$. In essence, this reduction adversary is fairly similar to the reduction adversary considered in the previous case. Namely, $\mathcal{R}^{\mathcal{A}}$ commences identically but produces the leaves by querying its collection oracle instead of its challenge oracle. Subsequently, the reduction adversary actually computes the corresponding FL-XMSS-TW$^\$$ public key via its challenge oracle; as such, it specifies all (concatenations of sibling) nodes in the entire Merkle tree as collision targets.[15] Then, in its second stage, $\mathcal{R}^{\mathcal{A}}$ produces the FL-XMSS-TW$^\$$ signatures (on uniformly random messages) corresponding to the previously obtained values and provides these signatures, together with the FL-XMSS-TW$^\$$ public key, to $\mathcal{A}$. Ultimately, whenever $\mathcal{A}$ returns a (valid) forgery, the reduction adversary computes the corresponding path and searches for the first node on this path that coincides with the corresponding node in the original Merkle tree. After finding this node, the reduction adversary returns (the concatenation of) its children and an integer j such that the j-th query to the challenge oracle contained the colliding value from the original Merkle tree.

[14] Thus, allowing for at most l targets is sufficient, as this is precisely the number of considered WOTS-TW$^\$$ public keys.

[15] Hence, allowing for at most $l - 1$ targets is sufficient, since this is exactly the number of nodes in the Merkle tree (excluding the leaves).

Given the foregoing, we can derive the following result.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_3} : \Pr\left[\mathrm{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^{\$}} = 1 \wedge \neg E_F \wedge \neg E_P\right] \leq \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRC},\mathsf{THFC},l-1}(\mathcal{B}_3)$$

Final Result. Aggregating the results established above, we can derive Security Theorem 2 as shown below. In this derivation, $G_{\mathcal{A}}$ denotes $\mathrm{Game}^{\text{EUF-RMA}}_{\mathcal{A},\text{FL-XMSS-TW}^{\$}}$.

$$\begin{aligned}
&\forall_{\mathcal{A}} \exists_{\mathcal{B}_0,\mathcal{B}_1,\mathcal{B}_2,\mathcal{B}_3} : \mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A}) \leq \\
&\left|\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A}) - \mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}^{\$}}(\mathcal{A})\right| + \mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}^{\$}}(\mathcal{A}) \leq \\
&\mathsf{Adv}^{\mathsf{SKWG}}_{\mathrm{PRF}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}^{\$}}(\mathcal{A}) = \\
&\mathsf{Adv}^{\mathsf{SKWG}}_{\mathrm{PRF}}(\mathcal{B}_0) + \Pr[G_{\mathcal{A}} = 1 \wedge E_F] + \Pr[G_{\mathcal{A}} = 1 \wedge \neg E_F \wedge E_P] \\
&+ \Pr[G_{\mathcal{A}} = 1 \wedge \neg E_F \wedge \neg E_P] \leq \\
&\mathsf{Adv}^{\mathsf{SKWG}}_{\mathrm{PRF}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^{\$},\mathsf{THFC},l}(\mathcal{B}_1) + \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{PKCO},\mathsf{THFC},l}(\mathcal{B}_2) \\
&+ \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRC},\mathsf{THFC},l-1}(\mathcal{B}_3)
\end{aligned}$$

Once again, even though no formal runtime analysis is provided, it is clear from the preceding discussion (and from the EasyCrypt artifacts) that there exist $\mathcal{B}_0$, $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ that not only satisfy the above inequality, but also terminate in approximately the same time as $\mathcal{A}$.

Here, we could trivially combine Security Theorem 1 and Security Theorem 2 to obtain a bound on $\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{A})$ solely based on the properties of the employed KHFs and THFs.

6 XMSS-TW

XMSS-TW extends FL-XMSS-TW in a way that allows for the processing of arbitrary-length messages. In essence, the transformation from FL-XMSS-TW to XMSS-TW is an instance of the hash-then-sign paradigm. To this end, XMSS-TW employs two additional KHFs — MKG and MCO — to compress arbitrary-length messages before executing the relevant procedures of FL-XMSS-TW. More precisely, MKG is used to generate an indexing key for MCO; in turn, indexed on this key, MCO is used to compress the message. The specification of XMSS-TW is provided in Algorithm 11 (key generation), Algorithm 12 (signing), and Algorithm 13 (verification). Here, $\mathcal{MS}$ denotes the set of indexing keys for MKG, and ad_c signifies an arbitrary address that satisfies the requirements for addresses used in FL-XMSS-TW (see Section 5).

Security Property. As for the majority of standalone signature schemes, we require XMSS-TW to possess the EUF-CMA security property. However, since XMSS-TW only allows for the signing of at most l signatures, we consider a bounded version of EUF-CMA. The game formalizing this property is provided in Figure 15; the (signature) oracle given to the adversary in this game is specified in Figure 16. As before, the adversary exclusively gains access to the oracle's

Algorithm 11 XMSS-TW's Key Generation Algorithm

procedure XMSS-TW.KeyGen()
  $\text{ms} \leftarrow\$ \mathcal{U}(\mathcal{MS})$
  $\text{ss} \leftarrow\$ \mathcal{U}(\mathcal{SS})$
  $\text{ps} \leftarrow\$ \mathcal{U}(\mathcal{PS})$
  $\text{ad} \leftarrow \text{ad}_c$
  $\text{pk}, \_ \leftarrow$ FL-XMSS-TW.KeyGen$(\text{ss}, \text{ps}, \text{ad})$
  return $\text{pk} := (\text{rt}, \text{ps}, \text{ad}), \text{sk} := (\text{ms}, 0, \text{ss}, \text{ps}, \text{ad})$

Algorithm 12 XMSS-TW's Signing Algorithm

procedure XMSS-TW.Sign$(\text{sk} := (\text{ms}, i, \text{ss}, \text{ps}, \text{ad}),\ m)$
  $\text{mk} \leftarrow \mathsf{MKG}(\text{ms}, i)$
  $\text{cm} \leftarrow \mathsf{MCO}(\text{mk}, m)$
  $(i, \text{sigWots}, \text{ap}), \_ \leftarrow$ FL-XMSS-TW.Sign$((i, \text{ss}, \text{ps}, \text{ad}), \text{cm})$
  return $\text{sig} := (\text{mk}, i, \text{sigWots}, \text{ap}), \text{sk} := (\text{ms}, i+1, \text{ss}, \text{ps}, \text{ad})$

Algorithm 13 XMSS-TW's Verification Algorithm

procedure XMSS-TW.Verify$(\text{pk} := (\text{rt}, \text{ps}, \text{ad}),\ m,\ \text{sig} := (\text{mk}, i, \text{sigWots}, \text{ap}))$
  $\text{cm} \leftarrow \mathsf{MCO}(\text{mk}, m)$
  $\text{ver} \leftarrow$ FL-XMSS-TW.Verify$(\text{pk}, \text{cm}, (i, \text{sigWots}, \text{ap}))$
  return ver

$\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}}$

1 : $\text{pk}, \text{sk} \leftarrow$ XMSS-TW.KeyGen()
2 : $\mathsf{O}_{\text{XMSS-TW}}$.Init(sk)
3 : $m', \text{sig}' \leftarrow \mathcal{A}^{\mathsf{O}_{\text{XMSS-TW}}}$.Forge(pk)
4 : $\text{isValid} \leftarrow$ XMSS-TW.Verify$(\text{pk}, m', \text{sig}')$
5 : $\text{isFresh} \leftarrow m' \notin \mathsf{O}_{\text{XMSS-TW}}.\mathcal{M}$
6 : **return** $\text{isValid} \wedge \text{isFresh} \wedge |\mathsf{O}_{\text{XMSS-TW}}.\mathcal{M}| \leq l$

Figure 15. EUF-CMA game for XMSS-TW.

$\mathsf{O}_{\text{XMSS-TW}}$

vars $\text{sk}, \mathcal{M}$

Init(ski)

1 : $\text{sk}, \mathcal{M} \leftarrow \text{ski}, [\,]$

Query(m)

1 : $\mathcal{M} \leftarrow \mathcal{M} \mid\mid m$
2 : $\text{sig}, \text{sk} \leftarrow$ XMSS-TW.Sign(sk, m)
3 : **return** sig

Figure 16. Signature oracle employed in EUF-CMA game for XMSS-TW.

Query procedure. Then, the advantage of any adversary $\mathcal{A}$ against EUF-CMA (of XMSS-TW) is defined as follows.

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A}) = \Pr\left[\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}} = 1\right]$$

Formal Verification. Next, we cover the (proof of) the security statement concerning XMSS-TW that we formally verify in this work. As can be extracted from Figure 10, we aim to demonstrate that the EUF-CMA property of XMSS-TW is implied by the PRF property of MKG, the M-eTCR property of MCO, and the EUF-RMA property of FL-XMSS-TW. More precisely, we formally verify the following security statement.

Security Theorem 3 (EUF-CMA for XMSS-TW)**.** *Let* MCO *be a random oracle. Then, for any adversary* $\mathcal{A}$*, there exist adversaries* $\mathcal{B}_0$*,* $\mathcal{B}_1$*, and* $\mathcal{B}_2$ *— each with approximately the same running time as* $\mathcal{A}$ *— such that the following inequality holds.*

$$\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{PRF}}_{\mathsf{MKG}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{M-eTCR}}_{\mathsf{MCO}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{B}_2) + \frac{(q_M + q_S + 1) \cdot q_S}{|\mathcal{MS}|} + \frac{l}{2^{8 \cdot n}}$$

Here, q_M *and* q_S *denote the number of queries that* $\mathcal{A}$ *issues to* MCO *and* $\mathsf{O}_{\text{XMSS-TW}}$*, respectively.*

Evidently, as MCO is assumed to be a random oracle, this security theorem — and, consequently, its formal verification — manifests itself in the ROM. Intuitively, this is required because, at some point, MCO needs to be adaptively reprogrammed in order to properly simulate the signature oracle. This reprogramming induces the additional $((q_M + q_S + 1) \cdot q_S)/|\mathcal{MS}|$ term in the bound. Furthermore, since MCO is considered to be a random oracle, the corresponding M-eTCR property essentially becomes a statistical bad event that occurs when two key-message pairs queried to the random oracle turn out to have the same output.

Loosely speaking, the formal verification of Security Theorem 3 proceeds as follows. Foremost, we change $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}}$ to $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$; that is, we consider XMSS-TW$^\$$ instead of XMSS-TW. Here, XMSS-TW$^\$$ is nearly identical to XMSS-TW, merely replacing the call to MKG by a sampling from the appropriate uniform distribution; accordingly, XMSS-TW$^\$$ does not sample and maintain ms, i.e., the value from XMSS-TW that is exclusively used as input to MKG. As these constitute the sole differences, we can reduce the PRF property of MKG to distinguishing between $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}}$ to $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$. Then, we separate the situation in which an adversary playing in $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$ returns a valid forgery into two cases. For both of these cases, we bound the probability via a reduction from either the M-eTCR property of MCO or the EUF-RMA property of FL-XMSS-TW. Here, the reduction from the EUF-RMA property of FL-XMSS-TW requires the consideration of a bad event that gives rise to the additional $l/2^{8 \cdot n}$ term in the security theorem. Collectively, this allows us to acquire the desired result.

Bound on Difference Between $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A})$ *and* $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}^\$}(\mathcal{A})$*.* Considering the differences between XMSS-TW and XMSS-TW$^\$$ described above, we can — given an adversary $\mathcal{A}$ playing in $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}}$ and $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$ — straightforwardly construct a reduction adversary $\mathcal{R}^{\mathcal{A}}$ attaining an advantage in $\text{Game}^{\text{PRF}}_{\mathcal{R}^{\mathcal{A}},\mathsf{MKG}}(b)$ that equals the (absolute) difference between $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A})$ and $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}^\$}(\mathcal{A})$. In consequence, we obtain the following bound.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_0} : \left| \mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A}) - \mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}^\$}(\mathcal{A}) \right| \leq \mathsf{Adv}^{\text{PRF}}_{\mathsf{MKG}}(\mathcal{B}_0)$$

Case Distinction for $\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$} = 1$. In the situation where an adversary playing in $\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$ provides a valid forgery — consisting of, say, message m' and signature $\mathrm{sig}' = (\mathrm{mk}', i', \mathrm{sigWots}', \mathrm{ap}')$ — we distinguish two (exhaustive) cases: In the first case, the provided forgery allows for the extraction of a (M-eTCR) collision for MCO; in the second case, it does not. More precisely, in the first case, mk' and m' map to the same value under MCO as (at least) one of the pairs of values used during the signature queries of $\mathcal{A}$. Furthermore, in the second case, it is possible — by appropriately reprogramming MCO — to guarantee (up to some bad events) that the forgery for XMSS-TW$^\$$ contains a valid forgery for FL-XMSS-TW with respect to the EUF-RMA property. Hereafter, E_{COLL} represents the event that the forgery allows for the extraction of a collision for MCO in the above way.

Formally, the preceding can be summarized by the following equality, where $G_{\mathcal{A}}$ denotes $\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$.

$$\forall_{\mathcal{A}} : \Pr[G_{\mathcal{A}} = 1] = \Pr[G_{\mathcal{A}} = 1 \wedge E_{\text{COLL}}] + \Pr[G_{\mathcal{A}} = 1 \wedge \neg E_{\text{COLL}}]$$

Bound on $\Pr\left[\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$} = 1 \wedge E_{\text{COLL}}\right]$. In case the (valid) forgery provided by $\mathcal{A}$ in $\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$ allows for the extraction of a (M-eTCR) collision for MCO in the previously described manner, we can trivially construct a reduction adversary $\mathcal{R}^{\mathcal{A}}$ playing in $\mathrm{Game}^{\text{M-eTCR}}_{\mathcal{R}^{\mathcal{A}},\mathsf{MCO}}$ that searches for this collision in the message compressions corresponding to $\mathcal{A}$'s signature queries and returns it. As a result, we can deduce the following bound.

$$\forall_{\mathcal{A}} \exists_{\mathcal{B}_1} : \Pr\left[\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$} = 1 \wedge E_{\text{COLL}}\right] \leq \mathsf{Adv}^{\text{M-eTCR}}_{\mathsf{MCO}}(\mathcal{B}_1)$$

Bound on $\Pr\left[\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$} = 1 \wedge \neg E_{\text{COLL}}\right]$. In case the (valid) forgery provided by $\mathcal{A}$ in $\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^\$}$ does not allow for the extraction of a collision for MCO in the above manner, we can construct a reduction adversary $\mathcal{R}^{\mathcal{A}}$ playing in $\mathrm{Game}^{\text{EUF-RMA}}_{\mathcal{R}^{\mathcal{A}},\text{FL-XMSS-TW}}$ that directly forwards each random oracle query, but reprograms MCO in every signature query. Specifically, when $\mathcal{A}$ issues the i-th signature query, the reduction adversary samples an indexing key for MCO uniformly at random and, subsequently, reprograms MCO to map this indexing key and the message from the signature query to the message of the i-th message-signature pair it received from its own game. Afterward, it returns the signature of the i-th message-signature pair, prepending the previously sampled indexing key. Indeed, $\mathcal{A}$ can only detect this reprogramming if it queried MCO on an indexing key and message *before* it issued a signature query with this same message in which, by pure chance, $\mathcal{R}^{\mathcal{A}}$ happened to sample the same indexing key. Thus, since $\mathcal{A}$ issues q_S signature queries — each of which has a probability of at most $\frac{1}{|\mathcal{MS}|}$ of coinciding with *any* of the up to q_M random oracle queries $\mathcal{A}$ issued before and *any* of the up to q_S previous reprogrammings as part of signature queries — the probability that $\mathcal{A}$ detects *any* reprogramming is at most

$((q_M+q_S)\cdot q_S)/|\mathcal{MS}|$.[16] In the ensuing, we denote the event that detection occurs by E_{DR}. Then, if $\mathcal{A}$ does not detect any reprogramming, at some point it returns a (valid) forgery — comprised of, say, m' and $\text{sig}' = (\text{mk}', i', \text{sigWots}', \text{ap}')$ — for XMSS-TW$^{\$}$ with respect to the EUF-CMA property. Certainly, provided that $\mathsf{MCO}(\text{mk}', m')$ is still fresh in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{R}^{\mathcal{A}},\text{FL-XMSS-TW}}$, $\mathsf{MCO}(\text{mk}', m')$ and $(i', \text{sigWots}', \text{ap}')$ then constitute a valid forgery for FL-XMSS-TW with respect to the EUF-RMA property. Given that m' is fresh in $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^{\$}}$ and the forgery provided by $\mathcal{A}$ does not allow for the extraction of a collision for MCO in the previously discussed way, $\mathsf{MCO}(\text{mk}', m')$ is fresh in $\text{Game}^{\text{EUF-RMA}}_{\mathcal{R}^{\mathcal{A}},\text{FL-XMSS-TW}}$ if it does not equal any of the messages from the message-signature pairs (provided to $\mathcal{R}^{\mathcal{A}}$) that *were not* used in answering the signature queries of $\mathcal{A}$. As there are l message-signature pairs of which each message is independently sampled uniformly at random, the (bad) event that m' equals any of the messages from the unused pairs is at most $l/2^{8\cdot n}$.

Based on the above, we can derive the following result. Here, $G_{\mathcal{A}}$ serves as a shorthand for $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^{\$}}$.

$$\begin{aligned}\forall_{\mathcal{A}}\exists_{\mathcal{B}_2}:\ &\Pr[G_{\mathcal{A}}=1\wedge\neg E_{\text{COLL}}]=\\ &\Pr[G_{\mathcal{A}}=1\wedge\neg E_{\text{COLL}}\wedge E_{\text{DR}}]+\Pr[G_{\mathcal{A}}=1\wedge\neg E_{\text{COLL}}\wedge\neg E_{\text{DR}}]\leq\\ &\frac{(q_M+q_S+1)\cdot q_S}{|\mathcal{MS}|}+\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{B}_2)+\frac{l}{2^{8\cdot n}}\end{aligned}$$

Final Result. Amalgamating the results obtained above, we can derive Security Theorem 3 as follows. In this derivation, $G_{\mathcal{A}}$ signifies $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{XMSS-TW}^{\$}}$.

$$\begin{aligned}&\forall_{\mathcal{A}}\exists_{\mathcal{B}_0,\mathcal{B}_1,\mathcal{B}_2}:\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A})\leq\\ &\left|\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A})-\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}^{\$}}(\mathcal{A})\right|+\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}^{\$}}(\mathcal{A})\leq\\ &\mathsf{Adv}^{\text{PRF}}_{\mathsf{MKG}}(\mathcal{B}_0)+\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}^{\$}}(\mathcal{A})=\\ &\mathsf{Adv}^{\text{PRF}}_{\mathsf{MKG}}(\mathcal{B}_0)+\Pr[G_{\mathcal{A}}=1\wedge E_{\text{COLL}}]+\Pr[G_{\mathcal{A}}=1\wedge\neg E_{\text{COLL}}]\leq\\ &\mathsf{Adv}^{\text{PRF}}_{\mathsf{MKG}}(\mathcal{B}_0)+\mathsf{Adv}^{\text{M-eTCR}}_{\mathsf{MCO}}(\mathcal{B}_1)+\mathsf{Adv}^{\text{EUF-RMA}}_{\text{FL-XMSS-TW}}(\mathcal{B}_2)+\frac{(q_M+q_S+1)\cdot q_S}{|\mathcal{MS}|}+\frac{l}{2^{8\cdot n}}\end{aligned}$$

As for the preceding security theorems, even in the absence of a formal runtime analysis, it is evident from the foregoing discussion (and from the EasyCrypt artifacts) that there exist $\mathcal{B}_0$, $\mathcal{B}_1$, and $\mathcal{B}_2$ that not only satisfy the above inequality, but also execute in approximately the same time as $\mathcal{A}$.

At this point, we can straightforwardly combine Security Theorem 1, Security Theorem 2, and Security Theorem 3 to obtain a bound on $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{XMSS-TW}}(\mathcal{A})$ that is exclusively based on the properties of the employed KHFs and THFs. This completes the formal verification of the security of XMSS-TW as a standalone construction.

[16] In the final bound, we get an extra one in the numerator. This is merely a proof artifact caused by the reduction adversary having to make a final query to verify the forgery.

References

ABB+19. José Bacelar Almeida, Cecile Baritel-Ruet, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Alley Stoughton, and Pierre-Yves Strub. Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of SHA-3. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 1607–1622, London, UK, November 11–15, 2019. ACM Press.

BBB+21. Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-aided cryptography. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 777–795. IEEE Computer Society, May 2021.

BBF+21. Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. EasyPQC: Verifying post-quantum cryptography. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2564–2586, New York, NY, USA, 2021. Association for Computing Machinery.

BCG+12. Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, and Santiago Zanella Béguelin. Computer-aided cryptographic proofs. In Lennart Beringer and Amy Felty, editors, *Interactive Theorem Proving*, pages 11–27, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

BGLZ11. Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago Zanella Béguelin. Beyond provable security verifiable IND-CCA security of OAEP. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 180–196, San Francisco, CA, USA, February 14–18, 2011. Springer, Heidelberg, Germany.

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: 26th Conference on Computer and Communications Security*, pages 2129–2146, London, UK, November 11–15, 2019. ACM Press.

BHRv21. Joppe W. Bos, Andreas Hülsing, Joost Renes, and Christine van Vredendaal. Rapidly verifiable XMSS signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):137–168, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8730.

CAD+20. David Cooper, Daniel Apon, Quynh Dang, Michael Davidson, Morris Dworkin, and Carl Miller. Recommendation for stateful hash-based signature schemes, 2020-10-29 00:10:00 2020.

CHH+17. Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1773–1788, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

GH19. Emily Grumbling and Mark Horowitz. *Quantum Computing: Progress and Prospects*. National Academies of Sciences, Engineering, and Medicine. The National Academies Press, 1st edition, April 2019.

GHHM21. Alex B. Grilo, Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Tight adaptive reprogramming in the QROM. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 637–667, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

HBG+18. Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018.

HK22. Andreas Hülsing and Mikhail Kudinov. Recovering the tight security proof of SPHINCS$^+$. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 3–33, Cham, 2022. Springer Nature Switzerland.

HMS22. Andreas Hülsing, Matthias Meijers, and Pierre-Yves Strub. Formal verification of Saber's public-key encryption scheme in EasyCrypt. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 622–653, Cham, 2022. Springer Nature Switzerland.

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: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 387–416, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany.

KKF20. Mikhail Kudinov, Evgeniy Kiktenko, and Aleksey Fedorov. [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, 2020. Accessed: 2022-2-1.

KM19. Neal Koblitz and Alfred J. Menezes. Critical perspectives on provable security: Fifteen years of "another look" papers. *Advances in Mathematics of Communications*, 13(4):517–558, 2019.

MCF19. David McGrew, Michael Curcio, and Scott Fluhrer. Leighton-Micali Hash-Based Signatures. RFC 8554, April 2019.

Mos18. Michele Mosca. Cybersecurity in an era with quantum computers: Will we be ready? *IEEE Security & Privacy*, 16:38–41, September 2018.

NIS16. NIST. National Institute for Standards and Technology. announcing request for nominations for public-key post-quantum cryptographic algorithms., December 2016. https://csrc.nist.gov/News/2016/Public-Key-Post-Quantum-Cryptographic-Algorithms.

NIS22. NIST. National Institute for Standards and Technology. PQC standardization process: Announcing four candidates to be standardized, plus fourth round candidates., March 2022. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4.

PKC22. Ray Perlner, John Kelsey, and David Cooper. Breaking category five SPHINCS$^+$ with SHA-256. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography*, pages 501–522, Cham, 2022. Springer International Publishing.

Zha19. Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 239–268, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.