A Tight Security Proof for SPHINCS$^+$, Formally Verified

Manuel Barbosa[1], François Dupressoir[2], Andreas Hülsing[3,4], Matthias Meijers[3], and Pierre-Yves Strub[5]

[1] University of Porto (FCUP) and INESC TEC, Portugal
[2] University of Bristol, United Kingdom
[3] Eindhoven University of Technology, The Netherlands
[4] SandboxAQ, USA
[5] PQShield, France
`fv-sphincsplus@mmeijers.com`

Abstract SPHINCS$^+$ is a post-quantum signature scheme that, at the time of writing, is being standardized as SLH-DSA. It is the most conservative option for post-quantum signatures, but the original tight proofs of security were flawed — as reported by Kudinov, Kiktenko and Fedorov in 2020. In this work, we formally prove a tight security bound for SPHINCS$^+$ using the EasyCrypt proof assistant, establishing greater confidence in the general security of the scheme and that of the parameter sets considered for standardization. To this end, we reconstruct the tight security proof presented by Hülsing and Kudinov (in 2022) in a modular way. A small but important part of this effort involves a complex argument relating four different games at once, of a form not yet formalized in EasyCrypt (to the best of our knowledge). We describe our approach to overcoming this major challenge, and develop a general formal verification technique aimed at this type of reasoning.
Enhancing the set of reusable EasyCrypt artifacts previously produced in the formal verification of stateful hash-based cryptographic constructions, we (1) improve and extend the existing libraries for hash functions and (2) develop new libraries for fundamental concepts related to hash-based cryptographic constructions, including Merkle trees. These enhancements, along with the formal verification technique we develop, further ease future formal verification endeavors in EasyCrypt, especially those concerning hash-based cryptographic constructions.

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

1 Introduction

The advent of sufficiently powerful quantum computers would jeopardize essentially all of the currently deployed public-key cryptography [BL17]. Albeit it is still uncertain if and when such computers will be practically realized, ongoing advancements and current prospects in the field lead many experts to believe that the likelihood of this happening in the near future is quite substantial [GH19,MP23]. Together with the potentially disastrous ramifications, this suggests that adequate preparation is paramount and urgent. Therefore, in 2016, the National Institute of Standards and Technology (NIST) started a process aimed at the standardization of post-quantum cryptography — cryptography that is executable on classical computers but provides security against attacks from both classical and quantum computers [BL17,NIS16]. In 2022, NIST announced the initial four cryptographic constructions to be standardized as a result of this process: CRYSTALS-Kyber for key encapsulation, together with CRYSTALS-Dilithium, Falcon, and SPHINCS$^+$ for digital signatures [NIS22]. Interestingly, two years prior, NIST already standardized two post-quantum digital signature schemes — XMSS and LMS (as well as their multi-tree variants) — independently from the ongoing standardization process [CAD$^+$20]. Although their maturity justified the standardization, these schemes are challenging to deploy in many contexts due to the required state management [CAD$^+$20,MKF$^+$16]. Hence, they do not suffice to fully replace contemporary digital signature schemes, which is the rationale for additionally standardizing the schemes from the standardization process.

During the above-mentioned standardization process, Kudinov, Kiktenko and Fedorov discovered an error in the tight security proof for a variant of the Winternitz One-Time Signature (WOTS) scheme, WOTS$^+$ [FKK20,KKF20]. As this scheme is (implicitly) a fundamental component of XMSS and SPHINCS$^+$, the tight security proofs for the latter two schemes used similar erroneous reasoning and, hence, were invalid as well [BHK$^+$19,HRS16]. Following this discovery, Hülsing and Kudinov remediated the error for the case of SPHINCS$^+$ by explicitly specifying the employed variant of WOTS —called WOTS-TW — defining (and proving) a specific security notion for this variant, and proving the tight security of SPHINCS$^+$ using this security notion [HK22]. Sadly, this approach did not directly translate to the case of XMSS due to the data processed by WOTS-TW being adversarially controlled (while it is user controlled in SPHINCS$^+$) [BDG$^+$23]. Nevertheless, building on the work by Hülsing and Kudinov [HK22], Barbosa, Dupressoir, Grégoire, Hülsing, Meijers, and Strub later constructed a novel tight security proof for XMSS; moreover, they formally verified this security proof using the EasyCrypt proof assistant [BDG$^+$23]. Unfortunately, in that work, the formal verification of the security proof for SPHINCS$^+$ in [HK22] was considered out of scope and left as future work. Given that the error in the original SPHINCS$^+$ security proof was only detected after several years of intense scrutiny, an increase in confidence regarding the novel security proof and its guarantees — as could be accomplished by, e.g., the formal verification of the proof — is no frivolous luxury.

As it is referred to above, formal verification (of cryptography) is an endeavor belonging to the field of computer-aided cryptography. This field aims to address the ever-increasing complexity of constructing and evaluating cryptography by employing computers to make these processes more rigorous and streamlined [BBB$^+$21]. Certainly, this is especially valuable in the context of complex cryptography that is still relatively novel, such as most of the post-quantum cryptography considered for standardization today. Over time, many tools and frameworks have been developed and proven effective in the construction and evaluation of progressively involved and significant cryptographic applications. For instance, as discussed before, EasyCrypt has been used to formally verify the novel security proof for XMSS [BDG$^+$23], but also to formally verify the correctness and security of Saber's Public-Key Encryption (PKE) scheme [HMS22]. Moreover, in combination with Jasmin, EasyCrypt has been used to construct and verify functionally correct, constant-time and efficient implementations of ChaCha20-Poly1305 [ABB$^+$20], SHA-3 [ABB$^+$19], and the aforementioned CRYSTALS-Kyber [ABB$^+$23]. Further examples using different tools include the formal verification of Hybrid Public-Key Encryption (HPKE) using CryptoVerif [ABH$^+$21], as well as the formal verification of Transport Layer Security (TLS) 1.3 [CHH$^+$17] and (the key establishment of) Signal using Tamarin [CCD$^+$20]. A more thorough and systematic overview of computer-aided cryptography with additional examples and success stories is provided in [BBB$^+$21].

Our Contribution. In this work, we aim to renew or boost the confidence in the security of (the parameter sets considered for) SPHINCS$^+$. Crudely put, we achieve this goal by formally verifying the novel tight security proof for SPHINCS$^+$ from [HK22]. However, we commence this endeavor by reconstructing the entire proof, essentially obtaining a modular version that is significantly more detailed. This reconstruction allows us to somewhat manage the complexity of the formal verification, and reuse some of the artifacts produced in the formal verification of the new tight security proof for XMSS [BDG$^+$23]. Nevertheless, the formal verification poses significant, novel challenges that we overcome, including the formal analysis of the considered few-time signature scheme and hypertree structure. Furthermore, one of the modular components we formally verify constitutes a generic relation between variants of the multi-target PREimage resistance (PRE), Target Collision Resistance (TCR), and Decisional Second-Preimage Resistance (DSPR) properties. This statement is comparable to Theorem 38 in [BH19], the proof of which employs non-standard reasoning. Correspondingly, the proof for the statement we consider is similarly non-standard. Loosely speaking, instead of utilizing a standard approach such as (a sequence of) reductions between pairs of games, this proof simultaneously compares four games through an extremely granular case analysis on the associated success probabilities. In the process of understanding and developing a proof technique aimed at this kind of reasoning, we formally verify the simpler of the fundamental theorems in [BH19], Theorem 25, that relates the (standard) PRE, SPR, and DSPR properties — allowing us to try out the arguments in a simpler context.

Due to the nature of the considered proof and the artifacts we build on, we opt to employ EasyCrypt — a powerful and expressive tool primarily aimed at the formal verification of code-based, game-playing security proofs in the computational model [BGHZ11] — for this work. As part of this work's contribution, we facilitate future formal verification endeavors in two ways. First, we extend EasyCrypt by creating and enhancing libraries based on the features required in this work. Specifically, we construct libraries containing (generic) definitions and properties for binary trees and Merkle trees; furthermore, we enhance the libraries for hash functions — originally produced in [BDG$^+$23] — by adding new properties and adjusting some of the definitions to be easier to use in different scenarios. Second, we develop a general formal verification technique targeting the type of non-standard reasoning required for the proof of the aforementioned relation between (variants of) the PRE, TCR, and DSPR properties. To the best of our knowledge, this is a novelty in the context of EasyCrypt. Although this paper only covers some of these artifacts in more detail, all of them can be found repository associated with this work, located at `https://github.com/MM45/FV-SPHINCSPLUS-EC`, or in the standard library of EasyCrypt.

Overview. The remainder of this paper is organized as follows. First, Section 2 introduces the fundamental concepts underlying SPHINCS$^+$ and its formal verification. Section 3 provides an overview of the formal verification. Lastly, Section 4, 5, and 6 discuss several aspects of the formal verification in detail.

2 Preliminaries

In the ensuing, we introduce the concepts used throughout the paper. Most of the fundamentals directly coincide with those of previous works [BDG$^+$23,BHK$^+$19]; however, we still provide them here for completeness.

Keyed Hash Functions. A Keyed Hash Function (KHF) is a function $\mathsf{KHF} : \mathcal{K} \times \mathcal{M} \to \mathcal{Y}$ where *key space* $\mathcal{K}$, *message space* $\mathcal{M}$, and *digest space* $\mathcal{Y}$ respectively denote sets of keys, messages, and digests. In practice, these spaces are essentially all sets of bitstrings. However, in specifications, each of these spaces may also be left abstract or be instantiated with any set relevant in the considered context — e.g., the set of integers within a certain range. Occasionally, we interpret and refer to a KHF as a family of hash functions indexed by keys from the key space.

For KHFs, we consider the *Interleaved Target Subset Resilience* (ITSR) and *Pseudo-Random Function family* (PRF) properties. Intuitively, a KHF is a PRF if querying an unknown, randomly selected hash function from the family defined by the KHF is computationally indistinguishable from querying an actual random function.[6] Formally, the ITSR and PRF properties for KHFs are respectively defined as the games in Figures 1 and 3; the oracles employed in these games are specified in Figures 2 and 4. In the ITSR game, $\mathrm{IB}^{\mathsf{MAP}}_{\mathsf{KHF}}$ is a predicate validating whether its arguments constitute an ITSR break; more precisely, this

[6]Unlike the PRF property, the ITSR property is specifically designed for SPHINCS$^+$ and does not admit as much of an intuitive interpretation out of context.

```
Game^{ITSR}_{A,KHF,MAP}
1 : $\mathsf{OITSR}_{\mathsf{KHF}}.\mathsf{Init}()$
2 : $(k, x) \leftarrow \mathcal{A}^{\mathsf{OITSR.Query}}.\mathsf{Find}()$
3 : return $\mathrm{IB}^{\mathsf{MAP}}_{\mathsf{KHF}}(k, x, \mathsf{OITSR}_{\mathsf{KHF}}.\mathscr{T})$
```

Figure 1. ITSR game.

```
OITSR
vars $\mathscr{T}$
Init()
1 : $\mathscr{T} \leftarrow [\,]$
Query($x$)
1 : $k \leftarrow\$ \mathcal{U}(\mathcal{K})$
2 : $\mathscr{T} \leftarrow \mathscr{T} \,||\, (k, x)$
3 : return $k$
```

Figure 2. Oracle employed in ITSR game.

```
Game^{PRF}_{A,KHF}(b)
1 : $\mathsf{OPRF}_{\mathsf{KHF}}.\mathsf{Init}(b)$
2 : $b' \leftarrow \mathcal{A}^{\mathsf{OPRF}_{\mathsf{KHF}}.\mathsf{Query}}.\mathsf{Distinguish}()$
3 : return $b'$
```

Figure 3. PRF game.

```
OPRF_KHF
vars $b, k, m$
Init(bi)
1 : $b, m \leftarrow \mathrm{bi}, \mathrm{emptymap}$
2 : $k \leftarrow\$ \mathcal{U}(\mathcal{K})$
Query($x$)
1 : if $b$ then
2 :   if $m.[x] = \perp$ then
3 :     $y \leftarrow\$ \mathcal{U}(\mathcal{Y})$
4 :     $m.[x] \leftarrow y$
5 :   $y \leftarrow m.[x]$
6 : else
7 :   $y \leftarrow \mathsf{KHF}(k, x)$
8 : return $y$
```

Figure 4. Oracle employed in PRF game.

predicate is defined as follows.

$$\mathrm{IB}^{\mathsf{MAP}}_{\mathsf{KHF}}(k, x, \mathscr{T}) = (k, x) \notin \mathscr{T} \;\wedge\; \mathsf{MAP}(\mathsf{KHF}(k, x)) \in \bigcup_{i=0}^{|\mathscr{T}|-1} \mathsf{MAP}(\mathsf{KHF}(\mathscr{T}[i][0], \mathscr{T}[i][1]))$$

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

$$\mathsf{Adv}^{\mathrm{ITSR}}_{\mathsf{KHF},\mathsf{MAP}}(\mathcal{A}) = \Pr\left[\mathrm{Game}^{\mathrm{ITSR}}_{\mathcal{A},\mathsf{KHF},\mathsf{MAP}} = 1\right]$$

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

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

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* $\mathcal{P}$, *tweak space* $\mathcal{T}$, *message space* $\mathcal{M}$, and *digest space* $\mathcal{Y}$ denote sets of (public) parameters, tweaks, messages, and digests, respectively. As for KHFs, in practice, these spaces are essentially all sets of bitstrings. In specifications, they may also be left abstract or be instantiated with any set relevant in the considered context. Nevertheless, throughout this work, the message and digest space of any THF are, respectively, the set of arbitrary-length bitstrings (i.e., $\{0,1\}^*$) and a set of fixed-length bitstrings (i.e., $\{0,1\}^k$ for some $k > 0$). Conceptually, THFs extend KHFs by explicitly considering contextual data in the form of tweaks, primarily serving the purpose of mitigating multi-target attacks. At times, we view and refer to a THF as a family of hash functions (mapping tweaks and messages to digests) indexed by (public) parameters from the (public) parameter space.

Alongside individual THFs, we consider collections of such functions — a concept introduced by the authors of SPHINCS$^+$ [BHK$^+$19] — containing a single THF for each possible length of the input messages. Alternatively stated, a collection of THFs constitutes a set $\mathsf{THFC} = \{\mathsf{THF}_\lambda : \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{Y}\}_{\lambda \in \Lambda}$ where Λ is the index set comprising the possible input lengths.[7]

For THFs, the properties we are concerned with in this work are the *Single-function, Multi-target, Distinct-Tweak* variants of *Target-Collision Resistance* (SM-DT-TCR), *Decisional Second-Preimage Resistance* (SM-DT-DSPR), and *Opening-Preimage Resistance* (SM-DT-OpenPRE). Additionally, we consider an extension of SM-DT-TCR, denoted by SM-DT-TCR-C, that takes the relevant THF collection into account. As their names suggest, all of these properties model a similar scenario where (1) a single, uniformly random (public) parameter is considered throughout (*single-function*); (2) the attack's targets, of which there may be multiple (*multi-target*), must be specified before the parameter is revealed; and (3) the tweaks used in the attack's targets must be distinct (*distinct-tweak*). Unsurprisingly, this scenario shares quite some similarities with the manner in which SPHINCS$^+$ operates; in particular, SPHINCS$^+$ uses the same (public) parameter — which is sampled uniformly at random during setup — and a unique tweak for each THF evaluation. For a more in-depth discussion and analysis of these properties, see [BHK$^+$19,HK22].

The considered THF properties are formalized through the games and oracles in Figures 5, 6, and 7 (SM-DT-TCR(-C) game, challenge oracle, and collection oracle); Figures 8 and 9 (SM-DT-DSPR games and challenge oracle); and Figures 10 and 11 (SM-DT-OpenPRE game and challenge oracle). In these games, t denotes the upper bound on the number of targets, $\mathrm{SPE}_{\mathsf{THF}}$ is a predicate that indicates whether there exists a second-preimage of the given message under THF (when the first two arguments to THF are the given parameter and tweak), and VQS_t is a predicate that validates the adversary's behavior by checking whether (1) the number of targets is less than or equal to t, (2) the provided index i is a valid index into the target list(s), and (3) the target tweaks are distinct from each other and, in case the relevant collection is considered, from the tweaks issued to the collection oracle. Moreover, for the (non-standard) advantage definition of SM-DT-DSPR, we need to define SM-DT-SPprob, a game that essentially repre-

[7] Technically, we could restrict the message space of each THF in a collection to only contain messages of the relevant length, but this does not yield significant advantages.

```
$\mathrm{Game}^{\text{SM-DT-TCR}\boxed{\text{-C}}}_{\mathcal{A},\mathsf{THF},\boxed{\mathsf{THFC}},t}$
$p \leftarrow\$ \mathcal{U}(\mathcal{P})$
$\mathsf{OTCR}_{\mathsf{THF}}.\mathsf{Init}(p)$
$\boxed{\mathsf{OC}_{\mathsf{THFC}}.\mathsf{Init}(p)}$
$\mathcal{A}^{\mathsf{OTCR}_{\mathsf{THF}}.\mathsf{Query},\boxed{\mathsf{OC}_{\mathsf{THFC}}.\mathsf{Query}}}.\mathsf{Pick}()$
$i, x' \leftarrow \mathcal{A}.\mathsf{Find}(p)$
$\mathrm{tw}, x \leftarrow \mathsf{OTCR}.\mathcal{T}[i], \mathsf{OTCR}.\mathcal{X}[i]$
$\textbf{return } x \neq x' \wedge \mathsf{THF}(p, \mathrm{tw}, x) = \mathsf{THF}(p, \mathrm{tw}, x')$
    $\wedge\, \mathrm{VQS}_t(i, \mathsf{OTCR}_{\mathsf{THF}}.\mathcal{T}, \boxed{\mathsf{OC}_{\mathsf{THFC}}.\mathcal{T}})$
```

Figure 5. SM-DT-TCR(-C) game. Outlined code is only considered in SM-DT-TCR-C.

```
$\mathsf{OTCR}_{\mathsf{THF}}$
$\textbf{vars } p, \mathcal{T}, \mathcal{X}$
$\mathsf{Init}(\mathrm{pi})$
    $p, \mathcal{T}, \mathcal{X} \leftarrow \mathrm{pi}, [\,], [\,]$
$\mathsf{Query}(\mathrm{tw}, x)$
    $y \leftarrow \mathsf{THF}(p, \mathrm{tw}, x)$
    $\mathcal{T}, \mathcal{X} \leftarrow \mathcal{T} \mid\mid \mathrm{tw}, \mathcal{X} \mid\mid x$
    $\textbf{return } y$
```

Figure 6. Challenge oracle employed in SM-DT-TCR(-C) game.

```
$\mathsf{OC}_{\mathsf{THFC}}$
$\textbf{vars } p, \mathcal{T}$
$\mathsf{Init}(\mathrm{pi})$
    $p, \mathcal{T} \leftarrow \mathrm{pi}, [\,]$
$\mathsf{Query}(\mathrm{tw}, x)$
    $y \leftarrow \mathsf{THFC}_{|x|}(p, \mathrm{tw}, x)$
    $\mathcal{T} \leftarrow \mathcal{T} \mid\mid \mathrm{tw}$
    $\textbf{return } y$
```

Figure 7. Collection oracle employed in games for tweakable hash functions.

sents the trivial attack against SM-DT-DSPR. Then, we define the advantage of any adversary $\mathcal{A}$ against Prop $\in$ {SM-DT-TCR, SM-DT-OpenPRE} as follows.

$$\mathsf{Adv}^{\mathrm{Prop}}_{\mathsf{THF},t}(\mathcal{A}) = \Pr\left[\mathrm{Game}^{\mathrm{Prop}}_{\mathcal{A},\mathsf{THF},t} = 1\right]$$

For the remaining THF properties, the corresponding advantages are given below, where $p = \Pr\left[\mathrm{Game}^{\text{SM-DT-DSPR}}_{\mathcal{A},\mathsf{THF},t} = 1\right]$ and $q = \Pr\left[\mathrm{Game}^{\text{SM-DT-SPprob}}_{\mathcal{A},\mathsf{THF},t} = 1\right]$.

$$\mathsf{Adv}^{\text{SM-DT-DSPR}}_{\mathsf{THF},t}(\mathcal{A}) = \max(0, p - q)$$
$$\mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{THF},\mathsf{THFC},t}(\mathcal{A}) = \Pr\left[\mathrm{Game}^{\text{SM-DT-TCR-C}}_{\mathcal{A},\mathsf{THF},\mathsf{THFC},t} = 1\right]$$

Hash Addresses. An instance of SPHINCS$^+$ employs the same collection of THFs throughout its entire execution; furthermore, it invariably uses the same (public) parameter to index the THFs. Thus, to mitigate multi-target attacks, SPHINCS$^+$ uses a unique, fixed tweak in each THF evaluation. For the construction of these tweaks, SPHINCS$^+$ utilizes a specific addressing scheme. In this scheme, an address essentially encodes (uniquely) identifying information for the THF evaluation in which the address is used. More precisely, each address

```
$\mathrm{Game}^{\text{SM-DT}\boxed{\text{-DSPR}}\boxed{\text{-SPprob}}}_{\mathcal{A},\mathsf{THF},t}$
    $p \leftarrow\$ \mathcal{U}(\mathcal{P})$
    $\mathsf{ODSPR}_{\mathsf{THF}}.\mathsf{Init}(p)$
    $\mathcal{A}^{\mathsf{ODSPR}_{\mathsf{THF}}.\mathsf{Query}}.\mathsf{Pick}()$
    $i, b \leftarrow \mathcal{A}.\mathsf{Find}(p)$
    $\mathrm{tw}, x \leftarrow \mathsf{ODSPR}.\mathcal{T}[i], \mathsf{ODSPR}.\mathcal{X}[i]$
    return $\mathrm{SPE}_{\mathsf{THF}}(p, \mathrm{tw}, x)\boxed{= b}$
        $\wedge\ \mathrm{VQS}_t(i, \mathsf{ODSPR}_{\mathsf{THF}}.\mathcal{T})$
```

Figure 8. SM-DT-DSPR (blue) and SM-DT-SPprob (yellow) game. Non-outlined code is considered in both games.

```
$\mathsf{ODSPR}_{\mathsf{THF}}$
vars $p, \mathcal{T}, \mathcal{X}$
$\mathsf{Init}(\mathrm{pi})$
    $p, \mathcal{T}, \mathcal{X} \leftarrow \mathrm{pi}, [\,], [\,]$
$\mathsf{Query}(\mathrm{tw}, x)$
    $y \leftarrow \mathsf{THF}(p, \mathrm{tw}, x)$
    $\mathcal{T}, \mathcal{X} \leftarrow \mathcal{T} \mid\mid \mathrm{tw}, \mathcal{X} \mid\mid x$
    return $y$
```

Figure 9. Challenge oracle employed in SM-DT-DSPR and SM-DT-SPprob game.

```
$\mathrm{Game}^{\text{SM-DT-OpenPRE}}_{\mathcal{A},\mathsf{THF},t}$
    $p \leftarrow\$ \mathcal{U}(\mathcal{P})$
    $\mathrm{tws} \leftarrow \mathcal{A}.\mathsf{Pick}()$
    $\mathrm{ys} \leftarrow \mathsf{OOPRE}_{\mathsf{THF}}.\mathsf{Init}(p, \mathrm{tws})$
    $i, x \leftarrow \mathcal{A}^{\mathsf{OOPRE}_{\mathsf{THF}}.\mathsf{Open}}.\mathsf{Find}(p, \mathrm{ys})$
    $\mathrm{tw}, y \leftarrow \mathrm{tws}[i], \mathrm{ys}[i]$
    return $\mathsf{THF}(p, \mathrm{tw}, x) = y$
        $\wedge\ i \notin \mathsf{OOPRE}_{\mathsf{THF}}.\mathcal{O}$
        $\wedge\ \mathrm{VQS}_t(i, \mathrm{tws})$
```

Figure 10. SM-DT-OpenPRE game for tweakable hash functions.

```
$\mathsf{OOPRE}_{\mathsf{THF}}$
vars $p, \mathcal{X}, \mathcal{O}$
$\mathsf{Init}(\mathrm{pi}, \mathrm{twsi})$
    $p, \mathcal{X}, \mathcal{O}, \mathrm{ys} \leftarrow \mathrm{pi}, [\,], [\,], [\,]$
    for tw in twsi do
        $x \leftarrow\$ \mathcal{U}(\mathcal{M})$
        $\mathcal{X} \leftarrow \mathcal{X} \mid\mid x$
        $\mathrm{ys} \leftarrow \mathrm{ys} \mid\mid \mathsf{THF}(p, \mathrm{tw}, x)$
    return ys
$\mathsf{Open}(i)$
    $\mathcal{O} \leftarrow \mathcal{O} \mid\mid i$
    return $\mathcal{X}[i]$
```

Figure 11. Challenge oracle employed in SM-DT-OpenPRE game.

constitutes a fixed-length sequence of nonnegative integers encoding the location and purpose of a THF evaluation within the virtual structure of a SPHINCS$^+$ instance. Naturally, not every (fixed-length) sequence of nonnegative integers constitutes a valid address in this scheme. Furthermore, because we approach the analysis of SPHINCS$^+$ in a modular manner, parts of the addresses may be irrelevant at certain points;[8] in such cases, we disregard the irrelevant part of the addresses. Throughout this paper, we use "address" to refer to a fixed-length sequence of nonnegative integers that constitutes (the relevant part of) a valid SPHINCS$^+$ address in the considered context. Additional clarification on address validity will be provided as necessary.

[8]For example, in a modular part that exclusively operates on a single layer of the virtual structure, the part of the addresses that indicates this layer is irrelevant.

3 Approach

The primary objective of this work is to renew or increase confidence in the (parameter sets considered for) SPHINCS$^+$, which we achieve via the formal verification of a tight security proof. To this end, we initially reconstruct the (handwritten) tight security proof for SPHINCS$^+$ from [HK22] in a modular manner, adding a significant amount of detail in the process. Utilizing this reconstructed proof as a guideline for the subsequent formal verification facilitates the overall process in several ways. First, the modularity reduces the complexity (of the formal verification) of individual statements by limiting their scope. Second, the modularity allows for the reuse of certain artifacts previously produced in the formal verification of the novel tight security proof for XMSS (which shares components with SPHINCS$^+$) [BDG$^+$23]. Third, the additional granularity serves as a foundation for the exceptional rigor and detail inevitably required by the formal verification.

Before discussing our approach to the proof and its formal verifications, we introduce the structure and workings of SPHINCS$^+$. On a high level, a SPHINCS$^+$ instance consists of (1) an instance of a hypertree-based signature scheme akin to XMSS$^{\text{MT}}$ [HBG$^+$18,HRB13], and (2) an instance of a forest-based — i.e., considering a sequence of individual trees — signature scheme for each leaf of this hypertree. This latter scheme is a few-time signature scheme called Forest of Random Subsets (FORS) which was introduced together with SPHINCS$^+$ [BHK$^+$19]. In the hypertree construction, each "node" constitutes a Merkle signature scheme similar to XMSS, using WOTS-TW, a variant of WOTS introduced in [HK22] as One-Time Signature (OTS) scheme. To sign an arbitrary-length message m with SPHINCS$^+$, the message is initially processed in a way that results in a fixed-length message m_c and an index i pointing to a leaf of the hypertree. Subsequently, the FORS instance associated with the i-th leaf is used to sign m_c; in turn, the hypertree construction is used to sign the public key of this FORS instance. Then, the SPHINCS$^+$ signature on m consists of the information used to obtain m_c and i from m, the FORS signature on m_c, and the hypertree signature on the public key of the employed FORS instance. Intuitively, the FORS signature can be seen as the actual signature on the message, while the signature of the hypertree construction can be seen as a proof that the FORS instance used to sign the message is actually part of the considered SPHINCS$^+$ instance.

Figure 12 presents a high-level overview of (the proofs underlying) our formal verification. In this figure, each node represents a property of a cryptographic construction, KHF, or THF; each edge indicates an implication between properties, i.e., from origin nodes to destination nodes.

The leftmost node in Figure 12 denotes the primary objective of this work: The formal verification of the *Existential UnForgeability under Chosen-Message Attacks* (EUF-CMA) security of SPHINCS$^+$. As depicted in the diagram, we show that this property is implied by (1) the PRF property of SKG and MKG, KHFs used for the generation of secret keys and message compression keys, respectively; (2) the EUF-CMA property of M-FORS$^\$$, a multi-instance variant of FORS that employs actual randomness instead of pseudorandomness;

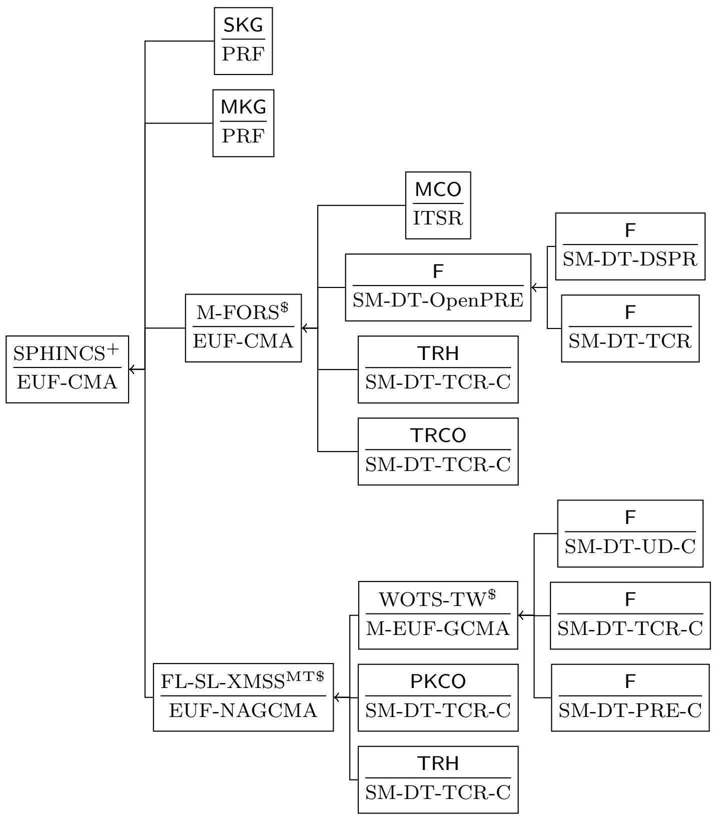

Figure 12. High-level overview of (the proofs underlying) our formal verification. Nodes represent properties of cryptographic constructions or functions: the text above the line indicates the construction or function; the text below the line indicates the property. Edges represent implications between properties: a property denoted by a destination node is implied by the conjunction of properties denoted by origin nodes.

and (3) the EUF-NAGCMA property — a non-adaptive, generic version of the EUF-CMA property — of FL-SL-XMSS$^{\text{MT}\$}$, a fixed-length, stateless variant of XMSS$^{\text{MT}}$ that uses actual randomness instead of pseudorandomness. The reason for considering variants of the cryptographic constructions that use actual randomness is exactly because of the initial PRF-related reductions mentioned above; indeed, these reductions replace pseudorandomness by actual randomness throughout the entire construction (so also throughout all "sub-constructions").

Proceeding in a modular fashion, we demonstrate that the EUF-CMA security of M-FORS$^{\$}$ can be based on (1) the ITSR property of MCO, a KHF used for the compression of (arbitrary-length) messages; (2) the SM-DT-OpenPRE property of F, a THF employed to generate Merkle tree leaves from secret key elements; and (3) the SM-DT-TCR-C property of TRH and TRCO, THFs used for the construction of Merkle trees from their leaves and, respectively, the compression of Merkle tree roots. In turn, we establish that the SM-DT-OpenPRE property of F is implied by its own SM-DT-DSPR and SM-DT-TCR properties. Interestingly, this implication can be considered a THF analog of Theorem 38 in [BH19], which states a comparable implication for KHFs. Correspondingly, the proofs require similar non-standard reasoning which, to the best of our knowledge, is unprecedented in EasyCrypt. Employing Theorem 25 in [BH19] — the proof of which only requires a relatively basic form of this reasoning — as an initial case study, we develop a formal verification technique aimed at this kind of reasoning. Building on this, we formally verify the implication required for SPHINCS$^+$.

Then, for FL-SL-XMSS$^{\mathrm{MT}\$}$, we show that its EUF-NAGCMA security is implied by (1) the M-EUF-GCMA property — a multi-instance, generic version of EUF-CMA specifically devised for the purpose of recovering the SPHINCS$^+$ proof [HK22] — of WOTS-TW$^\$$, a variant of WOTS-TW that employs actual randomness instead of pseudorandomness; (2) the SM-DT-TCR-C property of PKCO and TRH, THFs respectively employed for the compression of WOTS-TW$^\$$ public keys and the construction of Merkle trees from their leaves.[9] At this point, a single implication remains: The implication from several properties of THF F to the M-EUF-GCMA security of WOTS-TW$^\$$. Fortunately, in previous work, this implication has already been formally verified in a way that facilitates reuse [BDG$^+$23]. We capitalize on this and do not formally verify this implication anew.

Finally, combining all modular parts, we formally verify that the EUF-CMA security of SPHINCS$^+$ can solely be based on the properties of the employed KHFs and THFs, as desired. For convenience, we provide an overview of the considered cryptographic constructions and functions in Appendix A.

In the ensuing sections, we discuss the formal verification process more extensively, going by the cryptographic (sub-)constructions. Specifically, in order, we go over M-FORS$^\$$, FL-SL-XMSS$^{\mathrm{MT}\$}$, and SPHINCS$^+$. Throughout this discussion, we do not include any material directly from the produced formal verification artifacts in the interest of space. Instead, we cover the proofs underlying the formal verification in a way that allows for a near-verbatim translation to EasyCrypt, thus accurately representing the formally verified material.

4 M-FORS$^\$$

FORS was first introduced in [BHK$^+$19] as the few-time signature scheme used in SPHINCS$^+$. In practice, FORS is used with pseudorandom keys. However, our proof performs a PRF-related step on the level of SPHINCS$^+$ that replaces all the pseudorandom values with random values. Thus, when analyzing the security of FORS, we actually analyze FORS$^\$$, a version of FORS that operates using actual randomness. In fact, it turns out that considering a multi-instance variant of FORS$^\$$, M-FORS$^\$$, is convenient for the proof as SPHINCS$^+$ and the ITSR property (inherently) consider multiple instances of FORS$^\$$.

[9] Indeed, TRH is the same function in both M-FORS$^\$$ and FL-SL-XMSS$^{\mathrm{MT}\$}$.

Intuitively, the (virtual) structure of a $\mathrm{FORS}^{\$}$ instance is a sequence of Merkle trees, the leaves of which are digests of secret key values. The public key of such an instance is a single digest obtained by compressing the roots of the Merkle trees. A $\mathrm{FORS}^{\$}$ signature comprises, for each Merkle tree, a single secret key value and the corresponding authentication path (defined below). The selection of secret key values in the signature is derived from the message.

A $\mathrm{FORS}^{\$}$ instance is defined with respect to parameters k, a, and n, respectively denoting the number of Merkle trees, the height of each Merkle tree, and the byte-length of the secret key elements, the public key, and the (values associated with the) nodes of the Merkle trees. From a, we compute the number of leaves for each Merkle tree as $t = 2^a$. Furthermore, $\mathrm{FORS}^{\$}$ employs the THFs F, TRH, and TRCO. These functions have the same (public) parameter space and tweak space — referred to as the *public seed space* $\mathcal{PS}$ and *address space* $\mathcal{AD}$ — as well as the same message space $\{0,1\}^*$ and digest space $\{0,1\}^{8 \cdot n}$.

An instance of $\mathrm{M\text{-}FORS}^{\$}$ essentially manages multiple $\mathrm{FORS}^{\$}$ instances divided into sequences, where the number of sequences and the size of each sequence are respectively determined by parameters s and l'. $\mathrm{M\text{-}FORS}^{\$}$ utilizes the KHF MCO to process arbitrary-length messages, obtaining (1) a fixed-length message — processable by a $\mathrm{FORS}^{\$}$ instance — and (2) an index uniquely identifying a specific $\mathrm{FORS}^{\$}$ instance. Moreover, it uses a random function $\mathsf{MKG}^{\$}$ to generate a fresh indexing key for each message compression. Lastly, to guarantee a unique address for each THF evaluation in $\mathrm{M\text{-}FORS}^{\$}$'s operations, we require that addresses have a corresponding *xtree index* (xtri), *keypair index* (kpi), *type index* (typei), *ftree height index* (ftrhi), and *ftree breadth index* (ftrbi). These indices are nonnegative integers that indicate, in the given order, the sequence of $\mathrm{FORS}^{\$}$ instances, the $\mathrm{FORS}^{\$}$ instance within the sequence, the type of operation (tree hashing or tree root compression), the height of the node (in the $\mathrm{FORS}^{\$}$ instance), and the breadth of the node (in the $\mathrm{FORS}^{\$}$ instance).[10] Here, the breadth and height indices are only relevant for tree hashing operations.

In essence, provided with a public seed ps and a address ad, the key pair of a $\mathrm{FORS}^{\$}$ instance is constructed as follows. Initially, a $\mathrm{FORS}^{\$}$ secret key $\mathrm{sk} = \mathrm{sk}_0 \ldots \mathrm{sk}_{k \cdot t - 1}$ — $\mathrm{sk}_i \in \{0,1\}^{8 \cdot n}$ for $0 \leq i < k \cdot t$ — is sampled uniformly at random. To obtain the corresponding public key, first, a sequence of $k \cdot t$ Merkle tree leaves is computed from the secret key by processing each element with F. The resulting sequence contains k non-overlapping subsequences of t leaves, each uniquely defining a Merkle tree of height a. The roots of these trees can be obtained by iteratively computing the layers of each tree. Specifically, in the construction of the layer at height h in the j-th Merkle tree, the node at breadth b can be computed from its children c_l and c_r as $\mathsf{TRH}(\mathrm{ps}, \mathrm{ad}_{j,h,b}, c_l \mid\mid c_r)$, where $\mathrm{ad}_{j,h,b}$ denotes the unique address for this evaluation of TRH (obtained from appropriately adjusting ad based on j, h and b). Hereafter, we denote the operator that performs this computation for Merkle trees of height h (i.e., for

[10] For the ftree breadth index, we do not consider a single tree, but rather the full sequence of trees in a $\mathrm{FORS}^{\$}$ instance. This way, addresses are unique even for nodes indifferent trees but at the same height and breadth of their respective tree.

Listing 1 FORS$^\$$ Primary

```
proc FORS$^\$$.KeyGen(ps, ad)
    skF $\leftarrow\$ \mathcal{U}((\{0,1\}^{8 \cdot n})^{k \cdot t})$
    lvs $\leftarrow$ FORS$^\$$.SkFToLvs(skF, ps, ad)
    rts $\leftarrow$ [ ]
    ad.typei $\leftarrow$ ftrhType
    for $i = 0, \ldots, k-1$ do
        lvsst $\leftarrow$ lvs[$i \cdot t : (i+1) \cdot t$]
        rt $\leftarrow$ LvsToRt$_a$(ps, ad, lvsst, $i$)
        rts $\leftarrow$ rts || rt
    ad.typei $\leftarrow$ ftrcType
    pkF $\leftarrow$ TRCO(ps, ad, flatten(rts))
    return (pkF, ps, ad), (skF, ps, ad)
proc FORS$^\$$.Sign(sk := (skF, ps, ad), $m$)
    lvs $\leftarrow$ FORS$^\$$.SkFToLvs(skF, ps, ad)
    sig $\leftarrow$ [ ]
    ad.typei $\leftarrow$ ftrhType
    for $i = 0, \ldots, k-1$ do
        $j \leftarrow$ toint($m[i \cdot a : (i+1) \cdot a]$)
        skele $\leftarrow$ skF[$i \cdot t + j$]
        lvsst $\leftarrow$ lvs[$i \cdot t : (i+1) \cdot t$]
        ap $\leftarrow$ ConsAP$_a$(ps, ad, lvsst, $j$, $i$)
        sig $\leftarrow$ sig || (skele, ap)
    return sig
proc FORS$^\$$.Verify(pk := (pkF, ps, ad), $m$, sig)
    pkF$'$ $\leftarrow$ FORS$^\$$.SigToPkF($m$, sig, ps, ad)
    return pkF$'$ = pkF
```

Listing 2 FORS$^\$$ Auxiliary

```
proc FORS$^\$$.SkFToLvs(skF, ps, ad)
    lvs $\leftarrow$ [ ]
    ad.typei, ad.ftrhi $\leftarrow$ ftrhType, 0
    for $i = 0, \ldots, k \cdot t - 1$ do
        ad.ftrhb $\leftarrow i$
        lf $\leftarrow$ F(ps, ad, skF[$i$])
        lvs $\leftarrow$ lvs || lf
    return lvs
proc FORS$^\$$.SigToPkF($m$, sig, ps, ad)
    rts $\leftarrow$ [ ]
    ad.typei $\leftarrow$ ftrhType
    for $i = 0, \ldots, k-1$ do
        skele, ap $\leftarrow$ sig[$i$]
        $j \leftarrow$ toint($m[i \cdot a : (i+1) \cdot a]$)
        ad.ftrhi, ad.ftrbi $\leftarrow 0, i \cdot t + j$
        lf $\leftarrow$ F(ps, ad, skele)
        rt $\leftarrow$ APToRt$_a$(ps, ad, ap, lf, $j$, $i$)
        rts $\leftarrow$ rts || rt
    pkF $\leftarrow$ TRCO(ps, ad, flatten(rts))
    return pkF
```

lists of leaves of length 2^h) by $\mathsf{LvsToRt}_h$. Lastly, after computing the Merkle tree roots, the FORS$^\$$ public key pk is obtained by compressing the concatenation of these roots using TRCO. Since, for proper functioning, signing and verifying requires the public seed and address that were used in key generation, we include them in both the public and secret key for convenience.

Given a FORS$^\$$ key pair, a message $m \in \{0,1\}^{k \cdot a}$ is signed and verified in the following manner. Initially, m is split into k bitstrings of length a, each of which is interpreted as the big-endian binary representation of an integer in $[0, 2^a)$. This gives rise to a k-tuple of integers $(i_0, \ldots, i_{k-1})$. Next, for every i_j, $0 \leq j < k$, a so-called *authentication path* is constructed for the i_j-th leaf of the j-th Merkle tree in the FORS$^\$$ instance. This path is the sequence comprising, in order, the sibling nodes along the path from the root of the considered Merkle tree to the considered leaf. Indeed, this path can be computed from the list of leaves and the index of the leaf. Throughout the remainder, we denote the operator that constructs these paths for Merkle trees of height h by ConsAP_h.Then, the FORS$^\$$ signature on m is a k-tuple of pairs $(\mathrm{sk}_{i_j}, \mathrm{ap}_{i_j})$, $0 \leq j < k$, where sk_{i_j} and ap_{i_j} are the secret key element and authentication path corresponding to the i_j-th leaf of the j-th Merkle tree. Verification of a signature on m is performed by, initially, extracting the integers $(i_0, \ldots, i_{k-1})$ from m in the same way as before. Subsequently, the secret key elements in the signature are transformed into the

Listing 3 M-FORS$^\$$

```
proc M-FORS$^\$$.KeyGen(ps, ad)
    pkMF, skMF ← [ ], [ ]
    for $i = 0, \ldots, s \cdot l' - 1$ do
        ad.xtri, ad.kpi ← $\lfloor i/l' \rfloor, i \bmod l'$
        (pkF, _, _), (skF, _, _) ← FORS$^\$$.KeyGen(ps, ad)
        pkMF, skMF ← pkMF || pkF, skMF || skF
    return (pkMF, ps, ad), (skMF, ps, ad)
proc M-FORS$^\$$.Sign(sk := (skMF, ps, ad), $m$)
    mk ← MKG$^\$(m)$
    $m_c, i$ ← MCO(mk, $m$)
    ad.xtri, ad.kpi ← $\lfloor i/l' \rfloor, i \bmod l'$
    sigF ← FORS$^\$$.Sign((skMF[$i$], ps, ad), $m_c$)
    return mk, sigF
proc M-FORS$^\$$.Verify(pk := (pkMF, ps, ad), $m$, sig := (mk, sigF))
    $m_c, i$ ← MCO(mk, $m$)
    ad.xtri, ad.kpi ← $\lfloor i/l' \rfloor, i \bmod l'$
    isValid ← FORS$^\$$.Verify((pkMF[$i$], ps, ad), $m_c$, sigF)
    return isValid
```

corresponding leaves via F. Combining each of these leaves with the associated authentication path in the signature, the root of each Merkle tree in the FORS$^\$$ instance is computed. This is achieved by iteratively reconstructing the path from the leaf to the root using the sibling nodes in the authentication path. For instance, if the i_j-th leaf is a right child, the second node on the path is computed as $n_1 = \mathsf{TRH}(\mathrm{ps}, \mathrm{ad}_{j,1,x}, \mathrm{ap}_{i_j}[a-1] \mid\mid \mathrm{lf}_{i_j})$, where $x = \lfloor i_j/2 \rfloor$, $\mathrm{ad}_{j,1,x}$ is the unique address for this evaluation of TRH, and lf_{i_j} is the i_j-th leaf; then, if n_1 is a left child, the third node on the path is equals $\mathsf{TRH}(\mathrm{ps}, \mathrm{ad}_{j,2,y}, n_2 \mid\mid \mathrm{ap}_{i_j}[a-2])$, where $y = \lfloor x/2 \rfloor$; and so forth.[11] Henceforth, we denote the operator that performs this computation for Merkle trees of height h (i.e., for authentication paths of length $\log_2 h$) by ApToRt_h. Finally, the produced roots are compressed using TRCO to obtain a candidate public key. If and only if this candidate public key matches the original public key, verification succeeds.

Following the foregoing descriptions, Listing 1 provides the specification of FORS$^\$$'s key generation, signing, and verification algorithm. These algorithms employ auxiliary procedures for the computation of (1) a sequence of Merkle tree leaves corresponding to a FORS$^\$$ secret key, and (2) a FORS$^\$$ public key corresponding to a FORS$^\$$ signatures. For reuse purposes, we specify these auxiliary procedures separately in Listing 2. In the specifications, $l[i:j]$ denotes the slice of list l from index i (including) up to index j (excluding), flatten(l) denotes the sequential concatenation of all elements in list l, and toint(s) denotes the integer corresponding to bitstring s (assuming big-endian binary representation).

At this point, it is rather straightforward to specify M-FORS$^\$$, as it essentially constitutes a collection of FORS$^\$$ instances combined with a way to compress messages and select which instance to use for signing and verification. Listing 3 provides the specification of M-FORS$^\$$'s algorithms.

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

Security Property. For M-FORS$^\$$, we effectively consider a slight variant of the customary EUF-CMA security property that accounts for the fact that M-FORS$^\$$ expects to be provided with a public seed and an address. Furthermore, for the usage of the THFs to be secure (with respect to their assumed properties), this public seed should be sampled uniformly at random. The game and oracle formalizing this security property are respectively provided in Figures 13 and 14. Here, ad_z denotes an arbitrary address used for initialization.

```
Game^{EUF-CMA}_{$\mathcal{A}$,M-FORS$^\$$}
1 : $\mathrm{ad} \leftarrow \mathrm{ad}_z$
2 : $\mathrm{ps} \leftarrow\$ \mathcal{U}(\mathcal{PS})$
3 : $(\mathrm{pk}, \mathrm{sk}) \leftarrow \text{M-FORS}^\$.\mathsf{KeyGen}(\mathrm{ps}, \mathrm{ad})$
4 : $\mathsf{O}_{\text{M-FORS}^\$}.\mathsf{Init}(\mathrm{sk})$
5 : $m', \mathrm{sig}' \leftarrow \mathcal{A}^{\mathsf{O}_{\text{M-FORS}^\$}.\mathsf{Query}}.\mathsf{Forge}(\mathrm{pk})$
6 : $\mathrm{isValid} \leftarrow \text{M-FORS}^\$.\mathsf{Verify}(\mathrm{pk}, m, \mathrm{sig})$
7 : $\mathrm{isFresh} \leftarrow m' \notin \mathsf{O}_{\text{M-FORS}^\$}.\mathcal{M}$
8 : return $\mathrm{isValid} \wedge \mathrm{isFresh}$
```

Figure 13. EUF-CMA game for M-FORS$^\$$.

```
$\mathsf{O}_{\text{M-FORS}^\$}$
vars $\mathrm{sk}, \mathcal{M}$
$\mathsf{Init}(\mathrm{ski})$
    1 : $\mathrm{sk}, \mathcal{M} \leftarrow \mathrm{ski}, [\,]$
$\mathsf{Query}(m)$
    1 : $\mathrm{sig} \leftarrow \text{M-FORS}^\$.\mathsf{Sign}(\mathrm{sk}, m)$
    2 : $\mathcal{M} \leftarrow \mathcal{M} \mathbin{||} m$
    3 : return
```

Figure 14. Oracle employed in EUF-CMA game for M-FORS$^\$$.

Formal Verification. As illustrated in Figure 12, we demonstrate that the EUF-CMA security of M-FORS$^\$$ is implied by the ITSR property of MCO, the SM-DT-OpenPRE property of F, and the SM-DT-TCR-C property of TRH and TRCO. For the ITSR property of MCO, we instantiate MAP (see Figure 1) with CM, a function that maps $(m_c, i) \in \{0,1\}^{k \cdot a} \times [0, s \cdot l]$ — i.e., outputs from MCO — to the set $S = \{(i, j, \mathrm{toint}(m_c[j \cdot a : (j+1) \cdot a])) \mid 0 \leq j < k\}$. Intuitively, a tuple (x, y, z) from this set can be interpreted as an index pointing to the z-th leaf of the y-th Merkle tree in the x-th FORS$^\$$ instance. Formally, the security theorem we consider is the following.

Theorem 1 (**EUF-CMA for M-FORS$^\$$**). *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.*

$$\begin{aligned}\mathsf{Adv}^{\text{EUF-CMA}}_{\text{M-FORS}^\$}(\mathcal{A}) \leq \mathsf{Adv}^{\text{ITSR}}_{\mathsf{MCO},\mathsf{CM}}(\mathcal{B}_0) &+ \mathsf{Adv}^{\text{SM-DT-OpenPRE}}_{\mathsf{F},t_\mathrm{f}}(\mathcal{B}_1)\\ &+ \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRH},\mathsf{THFC},t_\mathrm{trh}}(\mathcal{B}_2)\\ &+ \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRCO},\mathsf{THFC},t_\mathrm{trco}}(\mathcal{B}_3)\end{aligned}$$

Here, THFC *denotes an arbitrary THF collection containing* F, TRH, *and* TRCO. *Furthermore, $t_\mathrm{f} = s \cdot l' \cdot k \cdot t$, $t_\mathrm{trh} = s \cdot l' \cdot k \cdot (t-1)$, and $t_\mathrm{trco} = s \cdot l'$.*

In essence, the formal verification of Theorem 1 proceeds by an exhaustive case analysis on the situation where $\mathcal{A}$ wins $\mathrm{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{M-FORS}^\$}$. This case analysis comprises four distinct cases; for each of these cases, the probability is bounded by exactly one of the advantage terms on the right-hand side of the theorem's inequality. In the following, $G_{\mathcal{A}}^{\top}$ signifies the event $\mathrm{Game}_{\mathcal{A},\text{M-FORS}^{\$}}^{\text{EUF-CMA}} = 1$.

Case Distinction for $G_{\mathcal{A}}^{\top}$. First, note that a valid EUF-CMA forgery for M-FORS$^{\$}$ consists of a message m' and a signature $\mathrm{sig}' = (\mathrm{mk}', \mathrm{sigF}')$ such that m' is fresh and sig' is a valid signature on m' under the considered public key $\mathrm{pk} = (\mathrm{pkMF}, \mathrm{ps}, \mathrm{ad})$. Here, recall that sig' is only valid if the FORS$^{\$}$ candidate public key pkF', computed from $(m'_c, i') = \mathsf{MCO}(\mathrm{mk}', m')$ and sigF', equals $\mathrm{pkF} = \mathrm{pkMF}[i']$. By the nature of the computations, validity of the forgery implies that, at some point during the construction of pkF', the considered values must coincide with the corresponding values in the original construction of pkF.

Harnessing the above observation, the first case we distinguish is one where the compression of m' (using MCO indexed on mk') results in the selection of a set of secret key elements from a FORS$^{\$}$ instance such that all of these values were already revealed as part of (the replies to) the issued signature queries.[12] Alternatively stated, the set $\mathsf{CM}(m'_c, i')$ is contained in the union of the analogous sets for the key/message pairs corresponding to the issued signature queries. As m' is fresh, it follows that the pair (mk', m') can be used to break ITSR. In the remaining, E_M denotes the event that captures this case.

If the first case does not occur $(\neg E_M)$, the forgery contains at least one secret key element skele' not revealed during the game. Then, the second case we distinguish concerns the leaf lf' produced from this secret key element equaling the corresponding leaf lf in the computation of pkF. In this case, skele' is a preimage of lf under F and, hence, can be used to break SM-DT-OpenPRE (for F). Hereafter, E_F signifies the event capturing this case (within $\neg E_M$).

If both the first and second case do not happen $(\neg E_M \wedge \neg E_F)$, the forgery contains a secret key element skele' that (1) was not revealed during the game and (2) does not produce the same leaf lf' as the one in the original construction of pkF. As such, the third case we distinguish regards the Merkle tree root computed from lf' and the associated authentication path ap' (from the same pair in the forgery) equaling the corresponding Merkle tree root in the original computation of pkF. Here, it must be the case that, at a certain point, the values on the reconstructed path (using lf' and ap') coincide with the corresponding values in the original Merkle tree. So, because the initial node(s) on these paths *are not* equal, the first node for which the paths converge must be obtained by applying TRH on different inputs. These inputs form a collision for TRH and, thus, can be used to break SM-DT-TCR-C (for TRH). Henceforth, we denote the event that captures this case (within $\neg E_M \wedge \neg E_F$) by E_T.

Finally, if all of the foregoing cases do not transpire $(\neg E_M \wedge \neg E_F \wedge \neg E_T)$, it must be the case that one of the Merkle tree roots provided as (part of the) input to TRCO to produce pkF' does not equal the corresponding root used in the original computation of pkF. Therefore, in this case, the (concatenated)

[12]The values need not all be revealed in (the reply to) a single signature query. They may have been revealed over (the replies to) any number of signature queries.

Merkle tree roots used to compute pkF$'$ and pkF form a collision for TRCO and can be used to break SM-DT-TCR-C (for TRCO).

Bound on $\Pr\left[G^{\top}_{\mathcal{A}} \wedge E_M\right]$. If the compression of m' (using mk$'$) indicates a set of secret key elements already revealed in (the responses to) the issued signature queries, we construct a reduction adversary $\mathcal{R}^{\mathcal{A}}$ playing in $\mathrm{Game}^{\mathrm{ITSR}}_{\mathcal{A},\mathsf{MCO},\mathsf{CM}}$ that straightforwardly simulates an execution of $\mathrm{Game}^{\mathrm{EUF\text{-}CMA}}_{\mathcal{A},\mathrm{M\text{-}FORS}^{\$}}$ but, instead of sampling, uses OITSR to obtain message keys for the compression of messages contained in queries by $\mathcal{A}$. Upon receiving the forgery from $\mathcal{A}$, $\mathcal{R}^{\mathcal{A}}$ directly extracts and returns (mk', m'), winning its own game. As a result, we can bound $\Pr\left[G^{\top}_{\mathcal{A}} \wedge E_M\right]$ by $\mathsf{Adv}^{\mathrm{ITSR}}_{\mathsf{MCO},\mathsf{CM}}(\mathcal{R}^{\mathcal{A}})$.

Bound on $\Pr\left[G^{\top}_{\mathcal{A}} \wedge \neg E_M \wedge E_F\right]$. In case the compression of m' indicates an unprecedented secret key element for which the image under F coincides with the corresponding original Merkle tree leaf, we construct the following reduction adversary playing in $\mathrm{Game}^{\mathrm{SM\text{-}DT\text{-}OpenPRE}}_{\mathcal{R}^{\mathcal{A}},\mathsf{F},t_{\mathrm{f}}}$. In its first stage, $\mathcal{R}^{\mathcal{A}}$ constructs and returns a list containing every address used to create Merkle tree leaves from secret key elements in M-FORS$^{\$}$. Then, in its second stage, $\mathcal{R}^{\mathcal{A}}$ utilizes the given public seed and Merkle tree leaves to compute the corresponding M-FORS$^{\$}$ public key and runs $\mathcal{A}$ with this public key, the public seed, and the initialization address. During the execution of $\mathcal{A}$, the reduction adversary answers signature queries in accordance with M-FORS$^{\$}$.$\mathsf{Sign}$, acquiring any necessary secret key elements via $\mathsf{OOPRE}_{\mathsf{F}}.\mathsf{Open}$. Upon receiving the forgery from $\mathcal{A}$, $\mathcal{R}^{\mathcal{A}}$ finds the secret key element not revealed in (responses to) the issued signature queries, and returns this element together with the associated index. By construction, the reduction adversary did not query any indices corresponding to secret key elements not included in (responses to) the issued signature queries. Consequently, $\mathcal{R}^{\mathcal{A}}$ wins its own game and we can bound $\Pr\left[G^{\top}_{\mathcal{A}} \wedge \neg E_M \wedge E_F\right]$ by $\mathsf{Adv}^{\mathrm{SM\text{-}DT\text{-}OpenPRE}}_{\mathsf{F},t_{\mathrm{f}}}(\mathcal{R}^{\mathcal{A}})$.

Bound on $\Pr\left[G^{\top}_{\mathcal{A}} \wedge \neg E_M \wedge \neg E_F \wedge E_T\right]$. If the leaf obtained from the unprecedented secret key element in the forgery does not equal the corresponding leaf in the original Merkle tree, but the root computed based on the associated authentication path does coincide with the root of the original Merkle tree, we construct the ensuing reduction adversary playing in $\mathrm{Game}^{\mathrm{SM\text{-}DT\text{-}TCR\text{-}C}}_{\mathcal{R}^{\mathcal{A}},\mathsf{TRH},\mathsf{THFC},t_{\mathrm{trh}}}$. In its first stage, $\mathcal{R}^{\mathcal{A}}$ constructs a key pair in line with M-FORS$^{\$}$.$\mathsf{KeyGen}$ by utilizing the provided oracles. Specifically, for each FORS$^{\$}$ instance, the reduction adversary samples the secret key, computes the Merkle tree leaves by querying the collection oracle on the secret key elements, computes the Merkle tree roots by querying the challenge oracle on the (concatenation of) sibling nodes — specifying these as targets — and computes the FORS$^{\$}$ public keys by querying the collection oracle on the (concatenation of) Merkle tree roots. Then, in its second stage, $\mathcal{R}^{\mathcal{A}}$ runs $\mathcal{A}$ with the previously generated public key, the received public seed, and the initialization address. Since $\mathcal{R}^{\mathcal{A}}$ constructed the considered key pair itself, it can trivially simulate the signing oracle for $\mathcal{A}$. Upon receiving the forgery from $\mathcal{A}$, $\mathcal{R}^{\mathcal{A}}$ computes the non-matching leaf from the unprecedented secret key element; extracts the collision based on this leaf, the associated authentication path, and the original Merkle tree; and returns the extracted collision and the associated index, winning its own game. As such, we can bound $\Pr\left[G_{\mathcal{A}}^{\top} \wedge \neg E_M \wedge \neg E_F \wedge E_T\right]$ by $\mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRH},\mathsf{THFC},t_{\text{trh}}}(\mathcal{R}^{\mathcal{A}})$.

Bound on $\Pr\left[G_{\mathcal{A}}^{\top} \wedge \neg E_M \wedge \neg E_F \wedge \neg E_T\right]$. Finally, if none of the previous cases occurs, we construct a reduction adversary playing in $\text{Game}^{\text{SM-DT-TCR-C}}_{\mathcal{R}^{\mathcal{A}},\mathsf{TRCO},\mathsf{THFC},t_{\text{trco}}}$ that, in essence, is extremely similar to the one considered in the preceding case. Namely, in its first stage, $\mathcal{R}^{\mathcal{A}}$ constructs a M-FORS$^{\$}$ key pair in the same way as the previous reduction adversary. However, in this case, $\mathcal{R}^{\mathcal{A}}$ employs the collection oracle for the construction of Merkle trees and the challenge oracle for the compression of Merkle tree roots. In its second stage, $\mathcal{R}^{\mathcal{A}}$ also proceeds in the same way as the previous reduction adversary, except that it now extracts and returns a collision (and the associated index) based on the Merkle tree root computed from the forgery. Following, we can bound $\Pr\left[G_{\mathcal{A}}^{\top} \wedge \neg E_M \wedge \neg E_F \wedge \neg E_T\right]$ by $\mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRCO},\mathsf{THFC},t_{\text{trco}}}(\mathcal{R}^{\mathcal{A}})$.

Final Result. At this point, Theorem 1 trivially follows from the established bounds and the fact that the sum of the probabilities for the considered cases is precisely equal to $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{M-FORS}^{\$}}(\mathcal{A})$.

4.1 SM-DT-OpenPRE From SM-DT-TCR and SM-DT-DSPR

At this stage, we go over the formal verification of the aforementioned generic relation between the SM-DT-OpenPRE, SM-DT-DSPR and SM-DT-TCR properties of a THF with a finite message space. By instantiating this relation with F[13] (and combining it with Theorem 1), we complete the modular part of the formal verification rooted at M-FORS$^{\$}$ (see Figure 12). Formally, the security statement we consider is the following.

Theorem 2 (SM-DT-OpenPRE **for a THF).** *For any adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}_0$ and $\mathcal{B}_1$ — each with approximately the same running time as $\mathcal{A}$ — such that the following inequality holds.*

$$\mathsf{Adv}^{\text{SM-DT-OpenPRE}}_{\mathsf{THF},t}(\mathcal{A}) \leq \mathsf{Adv}^{\text{SM-DT-DSPR}}_{\mathsf{THF},t}(\mathcal{B}_0) + 3 \cdot \mathsf{Adv}^{\text{SM-DT-TCR}}_{\mathsf{THF},t}(\mathcal{B}_1)$$

Here, THF *is an arbitrary THF with a finite message space $\mathcal{M}$, and $t \geq 0$.*

In [BH19], the authors demonstrate generic relations between similar properties for KHFs. The proofs in [BH19] make use of non-standard techniques that we also use for the proof of Theorem 2. As these techniques are unprecedented in EasyCrypt, we elaborate on the formal verification and its challenges here.

Typical proofs considered in EasyCrypt compare, at each step, (the simultaneous execution of) two games. This encompasses usual proofs via direct reduction or game hopping. Namely, in these cases, the tool's probabilistic relational logic enables formal reasoning about the desired equivalences (potentially up to

[13] Although F technically has an infinite message space (see Section 2), we can replace it in the context of M-FORS$^{\$}$/SPHINCS^{+} by an equivalent function with finite message space $\{0,1\}^{8 \cdot n}$ because F is only evaluated on messages from this space in this context.

some failure event) between each pair of games. Unfortunately, it is not possible to refer to any games beyond the two collated games, which would be needed to formally verify our proof directly. Particularly, the proof for Theorem 2, which closely resembles the proofs for Theorems 25 and 38 in [BH19], requires simultaneous reasoning about *four games*: $\mathrm{Game}^{\text{SM-DT-OpenPRE}}_{\mathcal{A},\mathsf{THF},t}$, $\mathrm{Game}^{\text{SM-DT-DSPR}}_{\mathcal{R}^{\mathcal{A}}_{D},\mathsf{THF},t}$, $\mathrm{Game}^{\text{SM-DT-SPprob}}_{\mathcal{R}^{\mathcal{A}}_{D},\mathsf{THF},t}$, and $\mathrm{Game}^{\text{SM-DT-TCR}}_{\mathcal{R}^{\mathcal{A}}_{T},\mathsf{THF},t}$. Here, the reduction adversaries $\mathcal{R}^{\mathcal{A}}_{D}$ and $\mathcal{R}^{\mathcal{A}}_{T}$ are relatively straightforward. Specifically, in their first stage, both reduction adversaries run $\mathcal{A}$'s first stage to obtain the list of tweaks; then, for each tweak in this list, they query their own oracle on this tweak and a uniformly random message (freshly sampled for each query), constructing a list of digests from the responses. In their second stage, the reduction adversaries run $\mathcal{A}$'s second stage, providing it with the received public parameter and the previously constructed digest list. Upon receiving (i', x') from $\mathcal{A}$, $\mathcal{R}^{\mathcal{A}}_{T}$ returns (i', x') and $\mathcal{R}^{\mathcal{A}}_{D}$ returns (i', b), where b guesses that the message contained in the $\mathcal{R}^{\mathcal{A}}_{D}$'s i'-th query only has a single preimage if and only if x' equals this message.

Using the above four games, the proof (and its formal verification) proceeds by performing an extremely granular case analysis across multiple dimensions, expressing the success probability associated with each game as a sum of probabilities of fine-grained events. More precisely, these dimensions of analysis are (1) the index j chosen by $\mathcal{A}$, (2) the number of preimages for the digest pointed to by j, and (3) the validity of $\mathcal{A}$'s provided preimage. On a more technical level, we perform this case analysis by defining F^j_i and S^j_i, two auxiliary games parameterized on the number of preimages i and the index j. Intuitively, these games are analogous to the similarly named auxiliary games in [BH19]: F^j_i and S^j_i respectively capture the failure and success cases for the considered SM-DT-OpenPRE game. Utilizing these auxiliary games, the proof advances by performing the following for each game (of the four primary games): First, decompose the success probability across the above-mentioned dimensions; second, show that, for some cases, the probability equals that of F^j_i and S^j_i; third, show that, for some (other) cases, the probability equals 0; fourth, show that, for the remaining cases — corresponding to the adversary finding a preimage different from the original one chosen by the reduction adversary (which is information-theoretically hidden in the preimage set of size i) — the probability can be expressed as $\frac{i-1}{i} \cdot S^j_i$; and, lastly, combining the results into a closed formula. Subsequently, the resulting closed formulas can be combined to derive Theorem 2. In the process performed for each game, the second and third step constitute customary proofs for EasyCrypt, while the first and fourth step are non-standard and technically involved. For the non-standard steps, we develop and apply design patterns on techniques aimed at the required reasoning. We elaborate on these below.

In the decomposition of the success probabilities, our objective is to express the success probability of a game G ($\Pr[G^{\top}]$) as follows. Here, $G^{\top}_{i,j}$ denotes an event capturing a specific case of winning the game, parameterized by i and j.

$$\Pr\big[G^{\top}\big] = \sum_{j=0}^{t-1}\left(\sum_{i=0}^{|\mathcal{M}|}\Pr\big[G^{\top}_{i,j}\big]\right)$$

We prove this equality by two applications of induction from the outside in — i.e., first on j, then on i. So, we start with proving by induction that, for all z, the following holds, where ti denotes the (adversarially chosen) target index.

$$\Pr\big[G^{\top} \wedge 0 \leq \mathrm{ti} < z\big] = \sum_{j=0}^{z-1}\Pr\big[G^{\top} \wedge \mathrm{ti} = j\big]$$

In this proof, the base case (0) is trivial, and the inductive step directly follows from the fact that the events are disjoint. We obtain the desired summation in the range $[0, t)$ by showing that the adversary loses if it exceeds the number of targets. Then, we continue the deconstruction by introducing the second summation. Specifically, we prove by induction that, for all z, the following holds, where ntp denotes the number of preimages of the (adversarially chosen) target.

$$\Pr\big[G^{\top} \wedge 0 \leq \mathrm{ti} < t \wedge 0 \leq \mathrm{ntp} \leq z\big] = \sum_{j=0}^{t-1}\left(\sum_{i=0}^{z}\Pr\big[G^{\top} \wedge \mathrm{ti} = j \wedge \mathrm{ntp} = i\big]\right)$$

This proof is similar to the previous one, except that the base case requires us to argue that both probability expressions collapse to the case where the selected target has no preimages. We acquire the intended summation by proving that the number of preimages cannot exceed the (finite) number of messages $|\mathcal{M}|$. The resulting decomposition allows us to continue along the above proof outline.

Lastly, the most technically involved part of the formal verification concerns the reasoning about information-theoretically hidden preimages, which is at the heart of expressing the probability that the adversary finds a second preimage in S^j_i as the probability of sampling an element uniformly at random from a set of cardinality i and it not equaling a fixed element from this set. Proving this in EasyCrypt is technically involved because there is no inherent mechanism for reasoning about complex conditional probabilities (related to the execution of games). In our case, it requires transforming S^j_i into a variant that initially samples a digest y from the distribution induced by THF and, only in case y has exactly i preimages, samples x from the set of y's preimages after the adversary returned x'. In actuality, this transformation requires several intermediate transformations, each of which needs to guarantee that either the adversary's view is unaltered or the relevant event is not triggered. Loosely speaking, we alter the original S^j_i in the following sequence of game hops. First, we use the sampled message x only when the corresponding digest y has exactly i preimages, and make the adversary's view independent of it otherwise. Second, we invert the order of the sampling by sampling y first and sampling x from the preimage set of y. Third, we move the sampling of x to the end of the game. At this point, since x is sampled after the adversary returns its guess, the desired probability claim is relatively straightforward to prove using EasyCrypt's logic. For the technically involved and novel (for EasyCrypt) part of this proof, we developed several reusable results that permit reasoning about distributions over sets of images and preimages in functions with finite domain.

5 FL-SL-XMSS$^{\mathbf{MT\$}}$

XMSS$^{\text{MT}}$ is a stateful post-quantum digital signature scheme that is standardized as a standalone construction, meaning that it is used to sign arbitrary-length messages [CAD$^+$20]. In effect, SPHINCS$^+$ employs a stateless variant of this scheme that is exclusively used to sign fixed-length messages, i.e., FORS public keys, and therefore omits any initial message compression. We denote this variant by FL-SL-XMSS$^{\text{MT}}$. Within SPHINCS$^+$, FL-SL-XMSS$^{\text{MT}}$ operates — akin to FORS/M-FORS — using pseudorandomness. As we perform an all-encompassing PRF-related step on the level of SPHINCS$^+$, we only consider FL-SL-XMSS$^{\text{MT\$}}$, a variant of FL-SL-XMSS$^{\text{MT}}$ which operates using actual randomness.

Intuitively, the (virtual) structure of a FL-SL-XMSS$^{\text{MT\$}}$ instance constitutes a hypertree. Each "node" in this hypertree is an instance of a Merkle signature scheme that uses WOTS-TW$^\$$ as its OTS scheme, essentially constituting a variant of XMSS; we refer to these instances as "inner trees". The inner trees on the bottom layer of the hypertree are used to sign messages; all other inner trees are used to sign the roots of the inner trees one layer below. An instance of FL-SL-XMSS$^{\text{MT\$}}$ is defined with respect to parameters n, analogous to the identically named parameter for FORS$^\$$; h', the height of each inner tree; and d, the number of layers in the hypertree. From h' and d, we compute the number of leaves of each inner tree as $l' = 2^{h'}$, the height of the hypertree as $h = 2^{h' \cdot d}$, and the number of leaves of the hypertree as $l = 2^h$. Furthermore, FL-SL-XMSS$^{\text{MT\$}}$ employs, in addition to the previously introduced F and TRH, the THF PKCO for the compression of WOTS-TW$^\$$ public keys to inner tree leaves. PKCO has the same domain and range as the other THFs,[14] which largely remain identical to those used in FORS$^\$$. Here, the only difference concerns the (minimal) indices associated with the addresses. Specifically, in this context, we require addresses to have an associated *layer index* (li), *xtree index* (xtri), *type index* (typei), *key pair index* (kpi), *xtree height index* (xtrhi), and *xtree breadth index* (xtrbi). Respectively, these indices indicate the layer, the inner tree within the layer, the type of operation (chaining, public key compression, or tree hashing), the leaf (within the inner tree), and the height and breadth of the node (within the inner tree). Lastly, as we reuse formal verification artifacts for WOTS-TW$^\$$, we mostly abstract this scheme away, only providing details when needed. For more information about this scheme and its formal verification, see [BDG$^+$23].

Loosely speaking, given a public seed and an address, a FL-SL-XMSS$^{\text{MT\$}}$ key pair is constructed as follows. First, a FL-SL-XMSS$^{\text{MT\$}}$ secret key is a uniformly

[14] Recall that we consider the same message space for each THF (i.e., $\{0,1\}^*$).

random sequence consisting of all WOTS-TW$^\$$ secret keys used throughout the construction. Each of these WOTS-TW$^\$$ secret keys comprises len bitstrings of length $8 \cdot n$ and is associated with exactly one leaf of a single inner tree. The corresponding FL-SL-XMSS$^{\mathrm{MT}\$}$ public key is the root of the hypertree, which is computed by (1) transforming the WOTS-TW$^\$$ secret keys associated with the topmost inner tree into the corresponding public keys via F, (2) compressing these public keys with PKCO to obtain the corresponding leaves, and (3) computing the root of the topmost inner tree by iteratively constructing its layers (from these leaves) using TRH. For the same reasons as in FORS$^\$$/M-FORS$^\$$, we include the public seed and address in both the public and secret key.

A FL-SL-XMSS$^{\mathrm{MT}\$}$ signature on a message $m \in \{0,1\}^{8\cdot}$ is a sequence of d pairs, where the i-th pair consists of a WOTS-TW$^\$$ signature and an authentication path corresponding to (a particular leaf of) an inner tree on the i-th layer. Here, the inner tree and leaf to be used on the bottom layer are provided as input, completely determining the utilized inner trees and leaves from the upper layers. Naturally, the WOTS-TW$^\$$ signatures are produced with appropriate calls to the signing procedure of WOTS-TW$^\$$; the associated authentication paths are constructed analogously to the construction of such paths in FORS$^\$$, where the considered leaf is obtained by compressing the corresponding WOTS-TW$^\$$ public key via PKCO. Then, verification of a FL-SL-XMSS$^{\mathrm{MT}\$}$ signature succeeds if and only if the candidate root of the hypertree — constructed from the signature — equals the actual root of the hypertree contained in the public key. More precisely, in a bottom-up manner, the roots of the inner trees corresponding to the pairs in the signature are computed, where the message m serves as the initial root: First, the WOTS-TW$^\$$ public key is computed from the WOTS-TW$^\$$ signature (in the pair) and the considered root; second, the leaf corresponding to the obtained public key is produced via PKCO; third, the root of the inner tree is computed using the obtained leaf and the authentication path (in the pair), again in a similar manner to the analogous computation in FORS$^\$$. Repeating this process for each pair in the signature results in the candidate hypertree root.

In line with the preceding, the specification of FL-SL-XMSS$^{\mathrm{MT}\$}$ is provided in Listings 4 and 5; respectively, these listings specify the primary and auxiliary algorithms, the latter of which are specified separately for reuse purposes. In these specifications, the ConsAP, $\mathsf{LvsToRt}$, and APToRt are the same operators as in FORS$^\$$, yet parameterized on h' instead of a.[15] Furthermore, nrtrees(i) denotes the number of inner trees in layer i (which equals $2^{h'\cdot(d-i-1)}$), and — preventing clutter due to indexing — skMX$_{i,j}$ denotes the subsequence (of length l') corresponding to the j-th inner tree on the i-th layer. Finally, although the WOTS-TW$^\$$ procedures are not explicitly specified here, their names are purposely indicative of their functioning; moreover, they are mostly analogous to the similarly named procedures previously specified in this paper.

[15]The final argument to these operators is omitted as it was only used to determine which Merkle tree in the FORS$^\$$ instance to consider.

Listing 4 FL-SL-XMSS$^{\mathrm{MT}\$}$ Primary

```
proc FL-SL-XMSS$^{\mathrm{MT}\$}$.KeyGen(ps, ad)
    skMX ← [ ]
    for $i = 0, \dots, d-1$ do
        for $j = 0, \dots, \mathrm{nrtrees}(i) - 1$ do
            skX ←$ $\mathcal{U}(((\{0,1\}^{8 \cdot n})^{\mathrm{len}})^{l'})$
            skMX ← skMX || skX
    pkMX ← FL-SL-XMSS$^{\mathrm{MT}\$}$.SkMXToPkMX(skMX, ps, ad)
    return (pkMX, ps, ad), (skMX, ps, ad)
proc FL-SL-XMSS$^{\mathrm{MT}\$}$.Sign(sk := (skMX, ps, ad), $m$, $i$)
    rt ← $m$
    ad.xtri ← $i$
    sig ← [ ]
    for $j = 0, \dots, d-1$ do
        ad.li, ad.xtri, ad.kti ← $j$, $\lfloor \mathrm{ad.xtri}/l' \rfloor$, ad.xtri mod $l'$
        skX ← $\mathrm{skMX}_{\mathrm{ad.li},\mathrm{ad.xtri}}$
        skW ← skX[ad.kpi]
        ad.typei ← chType
        sigW ← WOTS-TW$^{\$}$.Sign((skW, ps, ad), rt)
        lvsX ← FL-SL-XMSS$^{\mathrm{MT}\$}$.SkXToLvsX(skX, ps, ad)
        ad.typei ← xtrhType
        apX ← $\mathsf{ConsAP}_{h'}$(ps, ad, lvsX, ad.kpi)
        rt ← $\mathsf{LvsToRt}_{h'}$(ps, ad, lvsX)
        sig ← sig || (sigW, apX)
    return sig
proc FL-SL-XMSS$^{\mathrm{MT}\$}$.Verify(pk := (pkMX, ps, ad), $m$, sig, $i$)
    pkMX′ ← FL-SL-XMSS$^{\mathrm{MT}\$}$.SigToPkMX($m$, sig, $i$, ps, ad)
    return pkMX′ = pkMX
```

Security Property. Regarding FL-SL-XMSS$^{\mathrm{MT}\$}$, we consider a non-adaptive, generic variant of the EUF-CMA security property denoted EUF-NAGCMA. Here, "non-adaptive" refers to the fact that the selection of messages for which the adversary receives signatures must happen at once; "generic" refers to the fact that this selection happens without knowledge of the considered public key. Akin to the EUF-CMA property for M-FORS$^{\$}$, this property accounts for the fact that FL-XMSS-TW$^{\$}$ expects a public seed and an address, where the public seed should be sampled uniformly at random. Furthermore, this property uses an indexed version of the conventional freshness definition. Figure 15 provides the game formalizing this property, where ad_z represents an arbitrary address.

Formal Verification. As Figure 12 depicts, we show that the EUF-NAGCMA security of FL-SL-XMSS$^{\mathrm{MT}\$}$ can be based on the M-EUF-GCMA property of WOTS-TW$^{\$}$, as well as the SM-DT-TCR-C property of PKCO and TRH. Here, we impose some additional constraints on the adversary's behavior in terms of its collection oracle queries. Intuitively, this limited oracle access can be interpreted as modeling that the scheme remains secure in a greater context where the same collection of THFs is also evaluated on different addresses (e.g., SPHINCS$^+$). Formally, the security theorem we consider is stated below.

Theorem 3 (EUF-NAGCMA **for** FL-SL-XMSS$^{\mathrm{MT}\$}$)**.** *For any adversary $\mathcal{A}$ that does not query its collection oracle on addresses used in* FL-SL-XMSS$^{\mathrm{MT}\$}$*,*

Listing 5 FL-SL-XMSS$^{\mathrm{MT\$}}$ Auxiliary

```
proc FL-SL-XMSS^{MT$}.SkXToLvsX(skX, ps, ad)
    lvsX ← [ ]
    for $i = 0, \dots, l' - 1$ do
        ad.typei, ad.kpi ← chType, $i$
        pkW ← WOTS-TW^$.SkWToPkW(skX[$i$], ps, ad)
        ad.typei ← pkcoType
        lf ← PKCO(ps, ad, flatten(pkW))
        lvs ← lvs || lf
    return lvs
proc FL-SL-XMSS^{MT$}.SkMXToPkMX(skMX, ps, ad)
    ad.li, ad.xtri ← $d - 1, 0$
    skX ← $\mathrm{skMX}_{d-1,0}$
    lvsX ← FL-SL-XMSS^{MT$}.SkXToLvsX(skX, ps, ad)
    ad.typei ← xtrhType
    pkMX ← $\mathsf{LvsToRt}_{h'}$(ps, ad, lvsX)
    return pkMX
proc FL-SL-XMSS^{MT$}.SigToPkMX($m$, sig, $i$, ps, ad)
    rt ← $m$
    ad.xtri ← $i$
    for $j = 0, \dots, d - 1$ do
        ad.li, ad.xtri, ad.kti ← $j$, $\lfloor \mathrm{ad.xtri}/l' \rfloor$, ad.xtri mod $l'$
        sigW, apX ← sig[$j$]
        ad.typei ← chType
        pkW ← WOTS-TW^$.SigToPkW(rt, sigW, ps, ad)
        ad.typei ← pkcoType
        lf ← PKCO(ps, ad, flatten(pkW))
        ad.typei ← xtrhType
        rt ← $\mathsf{APToRt}_{h'}$(ps, ad, apX, lf, ad.kpi)
    return rt
```

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.

$$\mathsf{Adv}^{\text{EUF-NAGCMA}}_{\text{FL-SL-XMSS}^{\text{MT\$}},\mathsf{THFC}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^{\$},\mathsf{THFC},t_{\text{wtw}}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{PKCO},\mathsf{THFC},t_{\text{pkco}}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRH},\mathsf{THFC},t_{\text{trh}}}(\mathcal{B}_2)$$

Here, THFC *denotes an arbitrary THF collection containing* F*,* PKCO*, and* TRH*. Furthermore,* $t_{\text{wtw}} = \sum_{i=0}^{d-1} \mathrm{nrtrees}(i) \cdot l'$*,* $t_{\text{pkco}} = \sum_{i=0}^{d-1} \mathrm{nrtrees}(i) \cdot l'$*, and* $t_{\text{trh}} = \sum_{i=0}^{d-1} \mathrm{nrtrees}(i) \cdot (l' - 1)$*.*

Concerning the M-EUF-GCMA property for WOTS-TW$^{\$}$, it suffices to know the following for the upcoming discussion. This property considers a two-stage adversary that, only in its first stage, is given access to a WOTS-TW$^{\$}$ signing oracle and $\mathsf{OC}_{\mathsf{THFC}}$. In this first stage, the adversary can issue queries consisting of a message and an address to the signing oracle, receiving a WOTS-TW$^{\$}$ public key and signature (on the query's message) that were freshly constructed *using the query's address in any THF evaluations.* In its second stage, the adversary is asked to produce a fresh and valid forgery under one of the public keys received in the first stage. For further details about this property, see [BDG+23].

The formal verification of Theorem 3 proceeds by an exhaustive case analysis on the scenario where $\mathcal{A}$ wins $\mathrm{Game}^{\text{EUF-NAGCMA}}_{\mathcal{A},\text{FL-SL-XMSS}^{\text{MT\$}},\mathsf{THFC}}$, bounding the proba-

```
$\mathrm{Game}^{\mathrm{EUF\text{-}NAGCMA}}_{\mathcal{A},\mathrm{FL\text{-}SL\text{-}XMSS}^{\mathrm{MT\$}},\mathsf{THFC}}$
1 :  $\mathrm{ad} \leftarrow \mathrm{ad}_z$
2 :  $\mathrm{ps} \leftarrow\$ \mathcal{U}(\mathcal{PS})$
3 :  $\mathsf{OC}_{\mathsf{THFC}}.\mathsf{Init}(\mathrm{ps})$
4 :  $\mathrm{ml} \leftarrow \mathcal{A}^{\mathsf{OC}_{\mathsf{THFC}}.\mathsf{Query}}.\mathsf{Choose}()$
5 :  $\mathrm{pk}, \mathrm{sk} \leftarrow \mathrm{FL\text{-}SL\text{-}XMSS}^{\mathrm{MT\$}}.\mathsf{KeyGen}(\mathrm{ps}, \mathrm{ad})$
6 :  $\mathrm{sigl} \leftarrow [\,]$
7 :  $\textbf{for } i = 0 \ldots \min(|\mathrm{ml}|, l) - 1 \textbf{ do}$
8 :    $\mathrm{sig} \leftarrow \mathrm{FL\text{-}SL\text{-}XMSS}^{\mathrm{MT\$}}.\mathsf{Sign}(\mathrm{sk}, \mathrm{ml}[i], i)$
9 :    $\mathrm{sigl} \leftarrow \mathrm{sigl} \mathbin{||} \mathrm{sig}$
10 : $m', \mathrm{sig}', i' \leftarrow \mathcal{A}.\mathsf{Forge}(\mathrm{pk}, \mathrm{sigl})$
11 : $\mathrm{isValid} \leftarrow \mathrm{FL\text{-}SL\text{-}XMSS}^{\mathrm{MT\$}}.\mathsf{Verify}(\mathrm{pk}, m', \mathrm{sig}', i')$
12 : $\mathrm{isFresh} \leftarrow m' \neq \mathrm{ml}[i']$
13 : $\textbf{return } \mathrm{isValid} \wedge \mathrm{isFresh} \wedge 0 \leq i < |\mathrm{ml}|$
```

Figure 15. EUF-NAGCMA game for FL-SL-XMSS$^{\mathrm{MT\$}}$.

bility of each of these cases by the relevant advantage term. Here, much of the reasoning related to the nature of the computations and the behavior of the reduction adversaries is analogous to the reasoning presented in the discussion on the formal verification for M-FORS$^{\$}$. As such, we do not elaborate as much here. In the ensuing, $G^{\top}_{\mathcal{A}}$ refers to the event $\mathrm{Game}^{\mathrm{EUF\text{-}NAGCMA}}_{\mathcal{A},\mathrm{FL\text{-}SL\text{-}XMSS}^{\mathrm{MT\$}},\mathsf{THFC}} = 1$.

Case Distinction for $G^{\top}_{\mathcal{A}}$ and Corresponding Bounds. First, remark that a valid EUF-NAGCMA forgery for FL-SL-XMSS$^{\mathrm{MT\$}}$ comprises a message m', a signature sig$'$, and an index i' such that m' is fresh — which, in this context, means that m' is different from the message at index i' in the list of selected (and signed) messages — and sig$'$ is a valid signature on m' under the considered public key $\mathrm{pk} = (\mathrm{pkMX}, \mathrm{ps}, \mathrm{ad})$ and i'. Recall that sig$'$ is only valid if the candidate root pkMX$'$ computed from m', sig$'$, and i' equals the actual root pkMX. By the verification procedure, validity of the forgery implies that the values considered in the computation of pkMX$'$ must at some point coincide with the corresponding values in the original computation of pkMX. As such, we can distinguish the following three exhaustive cases in the verification of the FL-SL-XMSS$^{\mathrm{MT\$}}$ forgery. In the first case, either (1) the initial WOTS-TW$^{\$}$ signature in the forgery is valid on m' or (2) we encounter an inner tree root that is different from the corresponding original root, but the associated WOTS-TW$^{\$}$ signature is valid. For this case, we denote the corresponding event and reduction adversary by E_W and $\mathcal{R}^{\mathcal{A}}_{W}$. In the second case, we encounter a WOTS-TW$^{\$}$ public key (computed from a WOTS-TW$^{\$}$ signature in the forgery) that does not match the corresponding original public key, but the inner tree leaf resulting from the compression of the public key equals to the corresponding original leaf. In the last case, we encounter an inner tree leaf that does not equal the corresponding original leaf, but the root computed with the associated authentication path is equal to the corresponding original root. Hereafter, in order, $\mathcal{R}_W^{\mathcal{A}}$, $\mathcal{R}_P^{\mathcal{A}}$, and $\mathcal{R}_T^{\mathcal{A}}$ denote the reduction adversaries considered in each case. Furthermore, E_W and E_P refer to the events capturing the first two cases, respectively.

On a high level, the constructed reduction adversaries all follow a similar approach. Specifically, in their first stage, the reduction adversaries run $\mathcal{A}$'s first stage, answering collection oracle queries via their own collection oracle, and obtain a list of messages to sign (ml). Then, the reduction adversaries construct a hypertree structure in line with FL-SL-XMSS$^{\mathrm{MT}}$.KeyGen using the provided oracles. Here, the primary difference between the reduction adversaries concerns which oracle they employ to compute certain values: $\mathcal{R}_W^{\mathcal{A}}$ uses the signing oracle to obtain WOTS-TW$^{\$}$ public keys and signatures on the messages and roots of inner trees; $\mathcal{R}_P^{\mathcal{A}}$ uses the challenge oracle to compress the WOTS-TW$^{\$}$ public keys to inner tree leaves; and $\mathcal{R}_T^{\mathcal{A}}$ uses the challenge oracle to compute the inner tree nodes from their leaves. All reduction adversaries compute the (for them) remaining values using the collection oracle. In their second stage, the reduction adversaries sign the messages in ml, conforming to FL-SL-XMSS$^{\mathrm{MT}}$.Sign, using the (hypertree) values obtained in their first stage. Subsequently, they run $\mathcal{A}$'s second stage, giving it their public key (i.e., hypertree root), the received public seed, the initialization address, and the list of signatures. Upon receiving the forgery from $\mathcal{A}$, the reduction adversaries find their forgery or collision and return this together with the associated index, winning their game. Importantly, as the reduction adversaries simulate the collection oracle for $\mathcal{A}$ through their own collection oracle, the constraint on $\mathcal{A}$'s queries guarantees that no addresses between the reduction adversaries' oracles overlap, which is required to win their games. Concluding, we can bound $\Pr\left[G_{\mathcal{A}}^{\top} \wedge E_W\right]$ by $\mathsf{Adv}^{\text{M-EUF-GCMA}}_{\text{WOTS-TW}^{\$},\mathsf{THFC},t_{\text{wtw}}}(\mathcal{R}_W^{\mathcal{A}})$, $\Pr\left[G_{\mathcal{A}}^{\top} \wedge \neg E_W \wedge E_P\right]$ by $\mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{PKCO},\mathsf{THFC},t_{\text{pkco}}}(\mathcal{R}_P^{\mathcal{A}})$, and $\Pr\left[G_{\mathcal{A}}^{\top} \wedge \neg E_W \wedge \neg E_P\right]$ by $\mathsf{Adv}^{\text{SM-DT-TCR-C}}_{\mathsf{TRH},\mathsf{THFC},t_{\text{trh}}}(\mathcal{R}_T^{\mathcal{A}})$.

Final Result. Combining these bounds and the fact that the sum of the considered probabilities equals $\mathsf{Adv}^{\text{EUF-NAGCMA}}_{\text{FL-SL-XMSS}^{\text{MT}\$},\mathsf{THFC}}(\mathcal{A})$, Theorem 3 follows.

6 SPHINCS^{+}

SPHINCS^{+} is essentially a straightforward extension of the *pseudorandom* versions of the constructions considered hitherto. Namely, a SPHINCS^{+} instance uses the KHFs MKG and SKG to generate pseudorandom values when necessary, rather than sampling and maintaining all of these values throughout. To this end, in addition to a public seed and address, it initializes and maintains a *message seed* and a *secret seed* used to index MKG and SKG, respectively. Furthermore, as alluded to above, it employs the pseudorandom versions of FORS$^{\$}$ (FORS), M-FORS$^{\$}$ (M-FORS), and FL-SL-XMSS$^{\mathrm{MT}\$}$ (FL-SL-XMSS$^{\mathrm{MT}}$). Although not explicitly presented here, these pseudorandom versions are trivially obtained from their counterparts by replacing (1) the sampled (sequence of) values in the secret key by the message seed and secret seed, (2) evaluations of $\mathsf{MKG}^{\$}$ by MKG (indexed by the message seed), and (3) references to sampled secret key

Listing 6 SPHINCS$^+$

```
proc SPHINCS$^+$.Keygen()
    ad $\leftarrow$ ad$_z$
    ms $\leftarrow\$\ \mathcal{U}(\mathcal{MS})$
    ss $\leftarrow\$\ \mathcal{U}(\mathcal{SS})$
    ps $\leftarrow\$\ \mathcal{U}(\mathcal{PS})$
    pkS $\leftarrow$ FL-SL-XMSS$^{\mathrm{MT}}$.SkMXToPkMX(ss, ps, ad)
    return (pkS, ps), (ms, ss, ps)
proc SPHINCS$^+$.Sign(sk := (ms, ss, ps), $m$)
    ad $\leftarrow$ ad$_z$
    mk, sigF $\leftarrow$ M-FORS.Sign((ms, ss, ps, ad), $m$)
    $m_c, i \leftarrow$ MCO(mk, $m$)
    ad.xtri, ad.kpi, ad.typei $\leftarrow \lfloor i/l' \rfloor, i \bmod l'$, ftrhtype
    pkF $\leftarrow$ FORS.SigToPkF($m_c$, sigF, ps, ad)
    sigMX $\leftarrow$ FL-SL-XMSS$^{\mathrm{MT}}$.Sign((ss, ps, ad), pkF, $i$)
    return mk, sigF, sigMX
proc SPHINCS$^+$.Verify(pk := (pkS, ps), $m$, sig := (mk, sigF, sigMX))
    ad $\leftarrow$ ad$_z$
    $m_c, i \leftarrow$ MCO(mk, $m$)
    ad.xtri, ad.kpi, ad.typei $\leftarrow \lfloor i/l' \rfloor, i \bmod l'$, ftrhtype
    pkF$'$ $\leftarrow$ FORS.SigToPkF($m_c$, sigF, ps, ad)
    pkS$'$ $\leftarrow$ FL-SL-XMSS$^{\mathrm{MT}}$.SigToPkMX(pkF$'$, sigMX, $i$, ps, ad)
    return pkS$'$ = pkS
```

values by the generation of these values through SKG (indexed by the secret seed and an appropriately adjusted address). Lastly, the set of valid addresses for SPHINCS$^+$ is the union of those for M-FORS and FL-SL-XMSS$^{\mathrm{MT}}$.

Following the preceding, Listing 6 specifies SPHINCS$^+$'s algorithms. Here, ad$_z$ is an initialization address that, as per the official SPHINCS$^+$ specification, has every associated index set to 0 (and the type index set to chType).

Security Property. As is customary for standalone signature schemes, we consider the conventional EUF-CMA security property for SPHINCS$^+$. For completeness, this property and the corresponding oracle are given in Figures 16 and 17, respectively.

```
Game$^{\text{EUF-CMA}}_{\mathcal{A},\text{SPHINCS}^+}$
1 : (pk, sk) $\leftarrow$ SPHINCS$^+$.Keygen()
2 : O$_{\text{SPHINCS}^+}$.Init(sk)
3 : $m'$, sig$'$ $\leftarrow \mathcal{A}^{\mathsf{O}_{\text{SPHINCS}^+}.\mathsf{Query}}$.Forge(pk)
4 : isValid $\leftarrow$ SPHINCS$^+$.Verify(pk, $m$, sig)
5 : isFresh $\leftarrow m' \notin$ O$_{\text{SPHINCS}^+}$.$\mathcal{M}$
6 : return isValid $\wedge$ isFresh
```

Figure 16. EUF-CMA game for SPHINCS$^+$.

```
O$_{\text{SPHINCS}^+}$
vars sk, $\mathcal{M}$
Init(ski)
    1 : sk, $\mathcal{M}$ $\leftarrow$ ski, [ ]
Query($m$)
    1 : sig $\leftarrow$ SPHINCS$^+$.Sign(sk, $m$)
    2 : $\mathcal{M} \leftarrow \mathcal{M}$ || $m$
    3 : return
```

Figure 17. Oracle employed in EUF-CMA game for SPHINCS$^+$.

Formal Verification. As Figure 12 shows, we demonstrate that the EUF-CMA security of SPHINCS$^+$ can be based on the PRF property of MKG and SKG, the EUF-CMA property of M-FORS$^\$$, and the EUF-NAGCMA property of FL-SL-XMSS$^{\text{MT}\$}$. More formally, we consider the following security theorem.

Theorem 4 (EUF-CMA **for** SPHINCS$^+$**).** *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.*

$$\begin{aligned}\mathsf{Adv}^{\text{EUF-CMA}}_{\text{SPHINCS}^+}(\mathcal{A}) \leq \mathsf{Adv}^{\text{PRF}}_{\mathsf{SKG}}(\mathcal{B}_0) + \mathsf{Adv}^{\text{PRF}}_{\mathsf{MKG}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{EUF-CMA}}_{\text{M-FORS}^\$}(\mathcal{B}_2) \\ + \mathsf{Adv}^{\text{EUF-NAGCMA}}_{\text{FL-SL-XMSS}^{\text{MT}\$},\mathsf{THFC}}(\mathcal{B}_3)\end{aligned}$$

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

Conceptually, the formal verification of this security theorem performs two main sequential steps: The substitution of all pseudorandomness by actual randomness and, subsequently, the extraction of a forgery for one of the considered subconstructions. In essence, the former step replaces SPHINCS$^+$ by SPHINCS$^{+\$}$, a variant that uses M-FORS$^\$$ and FL-SL-XMSS$^{\text{MT}\$}$ instead of their deterministic counterparts. The latter step essentially shows that a valid EUF-CMA forgery for SPHINCS$^{+\$}$ contains a valid EUF-CMA forgery for M-FORS$^\$$ or a valid EUF-NAGCMA forgery for FL-SL-XMSS$^{\text{MT}\$}$, thereafter relating each case to the corresponding advantage. In the ensuing, $G^{\top}_{\mathcal{A}}$ denotes $\text{Game}^{\text{EUF-CMA}}_{\mathcal{A},\text{SPHINCS}^{+\$}} = 1$.

Bound on $\left|\mathsf{Adv}^{\text{EUF-CMA}}_{\text{SPHINCS}^+}(\mathcal{A}) - \mathsf{Adv}^{\text{EUF-CMA}}_{\text{SPHINCS}^{+\$}}(\mathcal{A})\right|$. Considering the differences between SPHINCS$^+$ and SPHINCS$^{+\$}$, the transition from the former to the latter basically comes down to replacing SKG and MKG by actual random functions with the appropriate domain and range, on top of some refactoring to maintain functional correctness (e.g., moving all evaluations of SKG to the key generation and storing the result in the secret key). In fact, for SKG, we can replace each evaluation by a pure random sampling because each provided input is unique. Thus, for both functions, we can straightforwardly construct a reduction adversary playing in the corresponding PRF game that perfectly simulates an execution of the EUF-CMA game that $\mathcal{A}$ is playing in by substituting each evaluation of the considered function by an appropriate query to the provided PRF oracle. As such, we can bound $\left|\mathsf{Adv}^{\text{EUF-CMA}}_{\text{SPHINCS}^+}(\mathcal{A}) - \mathsf{Adv}^{\text{EUF-CMA}}_{\text{SPHINCS}^{+\$}}(\mathcal{A})\right|$ by the sum of $\mathsf{Adv}^{\text{PRF}}_{\mathsf{SKG}}(\mathcal{R}^{\mathcal{A}}_{S})$ and $\mathsf{Adv}^{\text{PRF}}_{\mathsf{MKG}}(\mathcal{R}^{\mathcal{A}}_{M})$, where $\mathcal{R}^{\mathcal{A}}_{S}$ and $\mathcal{R}^{\mathcal{A}}_{M}$ are the relevant reduction adversaries.

Case Distinction for $G^{\top}_{\mathcal{A}}$ and Corresponding Bounds. First, observe that a valid EUF-CMA forgery for SPHINCS$^{+\$}$ consists of a message m' and a signature $\text{sig}' = (\text{mk}', \text{sigF}', \text{sigMX}')$ such that m' is fresh and sig' is a valid signature for m' under the considered public key $\text{pk} = (\text{pkS}, \text{ps})$. Moreover, by the verification procedure of SPHINCS$^{+\$}$, sig' requires that sigMX' is a valid signature on the FORS$^\$$ public key pkF' derived from sigF'. Thus, if pkF' *does not* equal the corresponding original public key, it constitutes a different "message" than the original one signed by FL-SL-XMSS$^{\mathrm{MT}\$}$ in the considered SPHINCS$^{+\$}$ instance. Otherwise, by definition, $(\mathrm{mk}', \mathrm{sigF}')$ is a valid M-FORS$^{\$}$ signature on m', where m' is fresh. Indeed, the former case allows for the extraction of an EUF-NAGCMA forgery for FL-SL-XMSS$^{\mathrm{MT}\$}$; the latter case allows for the extraction of an EUF-CMA forgery for M-FORS$^{\$}$. In the imminent, E_X and $\mathcal{R}_X^{\mathcal{A}}$ denote the event and reduction adversary for the former case; $\mathcal{R}_M^{\mathcal{A}}$ denotes the reduction adversary for the latter case ($\neg E_X$ suffices to capture the latter case).

Loosely speaking, both reduction adversaries we construct follow a similar approach. Namely, both reduction adversaries construct a key pair for the sub-construction they *are not* an adversary for, using the provided collection oracle (FL-SL-XMSS$^{\mathrm{MT}}$)[16] or public seed and address (M-FORS$^{\$}$). Then, to simulate the signing oracle for $\mathcal{A}$, the reduction adversaries use the constructed key pair to create signatures for the corresponding sub-construction, use either the provided list of signatures (FL-SL-XMSS$^{\mathrm{MT}}$) or their own signing oracle (M-FORS$^{\$}$) to obtain signatures for the other sub-construction, and combine the signatures to construct the corresponding SPHINCS$^+$ signature. Upon receiving the forgery from $\mathcal{A}$, they extract and return the relevant forgery, winning their own game. As a result, we can bound $\Pr\left[G_{\mathcal{A}}^{\top} \wedge E_X\right]$ by $\mathsf{Adv}^{\text{EUF-NAGCMA}}_{\text{FL-SL-XMSS}^{\mathrm{MT}\$},\mathsf{THFC}}(\mathcal{R}_X^{\mathcal{A}})$ and $\Pr\left[G_{\mathcal{A}}^{\top} \wedge \neg E_X\right]$ by $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{M-FORS}^{\$}}(\mathcal{R}_M^{\mathcal{A}})$.

Final Result. From the above results, Theorem 4 follows. At last, we can combine the security theorem regarding WOTS-TW$^{\$}$ in [BDG+23] with the security theorems considered in this work to acquire a bound on the EUF-CMA security of SPHINCS$^+$ that is entirely based on the properties of the employed KHFs and THFs. This completes the formal verification of the security of SPHINCS$^+$.

Acknowledgments. We gratefully acknowledge discussions and support from the Formosa Crypto consortium. This research is supported by Amazon Web Services, as an Amazon Research Award supporting the Formosa Crypto consortium; by the InnovateUK ATI programme (10065634). Andreas Hülsing and Matthias Meijers are funded by an NWO VIDI grant (Project No. VI.Vidi.193.066).

Disclosure of Interests. All authors are members of the Formosa Crypto consortium. François Dupressoir is a member of the technical evaluation committee for PEPR en Cybersécurité.

[16] Crucially, this means that $\mathcal{R}_X^{\mathcal{A}}$ *does not* query its collection oracle on addresses used in FL-SL-XMSS$^{\mathrm{MT}}$, as required for the application of Theorem 3.

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.

ABB$^+$20. José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, Vincent Laporte, Tiago Oliveira, and Pierre-Yves Strub. The last mile: High-assurance and high-speed cryptographic implementations. In *2020 IEEE Symposium on Security and Privacy*, pages 965–982, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

ABB$^+$23. José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, Peter Schwabe, Antoine Séré, and Pierre-Yves Strub. Formally verifying kyber episode IV: Implementation correctness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):164–193, 2023.

ABH$^+$21. Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the HPKE standard. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 87–116, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

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*, pages 777–795, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

BDG$^+$23. Manuel Barbosa, François Dupressoir, Benjamin Grégoire, Andreas Hülsing, Matthias Meijers, and Pierre-Yves Strub. Machine-checked security for XMSS as in RFC 8391 and SPHINCS$^+$. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, volume 14085, pages 421–454, Cham, August 2023. Springer Nature Switzerland.

BGHZ11. Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

BH19. Daniel J. Bernstein and Andreas Hülsing. Decisional second-preimage resistance: When does SPR imply PRE? In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 33–62, Kobe, Japan, December 8–12, 2019. 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.

BL17. Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography. *Nature*, 549(7671):188–194, Sep 2017.

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.

CCD$^+$20. Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the Signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, October 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.

FKK20. Aleksey Fedorov, Evgeniy Kiktenko, and Mikhail Kudinov. [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: 22-05-2024.

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.

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 A. Kudinov. Recovering the tight security proof of SPHINCS$^+$. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 3–33, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany.

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, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 622–653, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

HRB13. Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal parameters for XMSS$^{\mathrm{MT}}$. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, pages 194–208, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

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 A. Kudinov, Evgeniy O. Kiktenko, and Aleksey K. Fedorov. Security analysis of the w-ots$^+$ signature scheme: Updating security bounds. *CoRR*, abs/2002.07419, 2020.

MKF$^+$16. David McGrew, Panos Kampanakis, Scott Fluhrer, Stefan-Lukas Gazdag, Denis Butin, and Johannes Buchmann. State management for hash-based signatures. Cryptology ePrint Archive, Report 2016/357, 2016. https://eprint.iacr.org/2016/357.

MP23. Michele Mosca and Marco Piani. Quantum threat timeline report 2023. Technical report, Global Risk Insitute, dec 2023.

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.

A Overview of Cryptographic Constructions and Functions

Tables 1 and 2 respectively present an overview of all the cryptographic constructions and functions considered throughout this work, providing the relevant security properties and a short description for each of them.

Table 1. Overview of Cryptographic Constructions.

Construction	*Property*	*Description*
SPHINCS$^+$	EUF-CMA	Main construction considered in this work.
FORS	-	Sub-construction of SPHINCS$^+$. Multiple instances in a SPHINCS$^+$ instance (one per leaf of hypertree).
M-FORS	-	Sub-construction of SPHINCS$^+$. Single instance in a SPHINCS$^+$ instance. Represents collection of all FORS instances in a SPHINCS$^+$ instance.
FL-SL-XMSS$^{\mathrm{MT}}$	-	Sub-construction of SPHINCS$^+$. Single instance in a SPHINCS$^+$ instance. Represents hypertree part of SPHINCS$^+$.
WOTS-TW	-	Sub-construction of FL-SL-XMSS$^{\mathrm{MT}}$. Multiple instances in a FL-SL-XMSS$^{\mathrm{MT}}$ instance (one per inner tree leaf). Used to sign messages (SPHINCS$^+$: FORS public keys) and roots of inner trees.
SPHINCS$^{+\$}$	-	Variant of SPHINCS$^+$ using actual randomness instead of pseudorandomness.
FORS$^{\$}$	-	Variant of FORS using actual randomness instead of pseudorandomness.
M-FORS$^{\$}$	EUF-CMA	Variant of M-FORS using actual randomness instead of pseudorandomness.
FL-SL-XMSS$^{\mathrm{MT}\$}$	EUF-NAGCMA	Variant of FL-SL-XMSS$^{\mathrm{MT}}$ using actual randomness instead of pseudorandomness.
WOTS-TW$^{\$}$	M-EUF-GCMA	Variant of WOTS-TW using actual randomness instead of pseudorandomness.

Table 2. Overview of Functions.

Function	*Property*	*Description*
SKG	PRF	Keyed hash function used for generating secret keys.
MKG	PRF	Keyed hash function used for generating keys used for message compression (with MCO).
MCO	ITSR	Keyed hash function used for compressing messages.
F	Several	Tweakable hash function used for computing Merkle tree leaves (in FORS and $\text{FORS}^{\$}$) and hash chaining (in WOTS-TW and $\text{WOTS-TW}^{\$}$).
TRH	SM-DT-TCR-C	Tweakable hash function used for computing Merkle tree nodes (in FORS, $\text{FORS}^{\$}$, $\text{FL-SL-XMSS}^{\text{MT}}$, and $\text{FL-SL-XMSS}^{\text{MT\$}}$).
TRCO	SM-DT-TCR-C	Tweakable hash function used for compressing (sequences of) Merkle tree roots to public keys (in FORS and $\text{FORS}^{\$}$).
PKCO	SM-DT-TCR-C	Tweakable hash function used for compressing $\text{WOTS-TW}^{\$}$ public keys to Merkle tree leaves (in $\text{FL-SL-XMSS}^{\text{MT}}$ and $\text{FL-SL-XMSS}^{\text{MT\$}}$).