SHRINCS: Shrunken SPHINCS$^+$

Security analysis*

Mikhail Kudinov

Draft November 18, 2026

Abstract

SHRINCS (Shrunken SPHINCS) is a composite post-quantum signature scheme optimized for Bitcoin, combining efficient stateful hash-based signatures with a stateless fallback. SHRINCS is built from two components under a single public key: a stateful flexible XMSS scheme (FXMSS) with WOTS+C one-time signatures, and a stateless SLH-DSA instance with a non-standard parameter set. The signer fixes the shape of the FXMSS tree at key generation; two shapes are suggested. With the unbalanced shape (UXMSS), the q-th signature ($q \geq 1$) occupies $532 + 16q$ bytes for $q \leq 8$ (one more byte per further 8 levels, up to $q = 64$) and verifies with $242 + \min(q, 255)$ tweakable-hash calls at depth 255; with the balanced shape (BXMSS), all signatures have the constant size of 854 bytes and verify with 262 tweakable-hash calls at the reference tree depth 20. When state is lost or exhausted, SHRINCS falls back to a 5777-byte stateless signature (a one-byte indicator and an SLH-DSA signature), standard except for a lower limit on the number of signatures. SHRINCS supports static seed-phrase backup and a "stateless-only after recovery" mode. This document provides the full specification of SHRINCS and its EU-CMA security proof, reducing security to the security properties of the underlying hash functions.

Contents

1 Introduction **3**
- 1.1 SHRINCS . . . 3
- 1.2 Our Contributions . . . 3
- 1.3 Organization . . . 5

2 The Building Blocks of SHRINCS **5**
- 2.1 Parameters . . . 5
- 2.2 Cryptographic (Hash) Function Families . . . 6
- 2.3 WOTS-TW . . . 12
- 2.4 WOTS+C . . . 14
- 2.5 XMSS and FXMSS . . . 17
- 2.6 FORS . . . 21
- 2.7 The Stateless Component: Hypertree and SLH-DSA . . . 23

3 The Complete SHRINCS Scheme **24**
- 3.1 Correctness . . . 27

*Status: Proposal for discussion, not the final version.

4 Security Analysis **32**
4.1 Hash Functions . . . 32
4.2 Security notions of hash functions . . . 32
4.3 Collections of Tweakable Hash Functions . . . 39
4.4 Modeling the Instantiated Functions . . . 43
4.5 Signature Security . . . 44
4.6 Main Security Theorems . . . 46
4.7 Proof of Theorem 4.2 (FXMSS Security) . . . 50
4.8 Proof of Theorem 4.1 (SHRINCS Security) . . . 61

5 Discussion: Reduction Accounting and What the Bounds Certify **69**
5.1 Concrete Security and Its Resources . . . 69
5.2 What an Attack Costs . . . 70
5.3 The Runtime in the SPHINCS$^+$ Accounting . . . 73
5.4 When Is the Collection Oracle Harmless? . . . 74

6 Concrete Security of the Proposed Parameters **75**
6.1 Summary of Results . . . 75
6.2 Generic Security . . . 77
6.3 Conventions . . . 77
6.4 Additive Constants and the Verification of the Forgery . . . 78
6.5 The Reprogramming Terms . . . 80
6.6 The Two Paths . . . 81
6.7 The Full Scheme . . . 83
6.8 Reduction Work under Convention M . . . 85

7 Open Questions and Work in Progress **86**

A HMAC **92**

B MGF1 **92**

C (Quantum) Random Oracle Tools **93**

D The Collection and Independence Framework: Experiments and Proofs **104**
D.1 Formal Definition of Security for a Member of a Collection . . . 104
D.2 Proof of the Collection Lemma . . . 105
D.3 Proof of the Independence Lemma . . . 107
D.4 Proof of the Domain-Separation Lemma . . . 108

E A Generic Bound for Multi-Target Target Collision Resistance with Exclusive Tweaks **114**

F Generic Bounds for Interleaved Target-Subset Resilience **117**
F.1 A Classical Bound without Reprogramming . . . 117
F.2 Bounds via Reprogramming . . . 119

G Generic Bounds for Multi-Target Collision Resistance with Random Sampling **121**

1 Introduction

Current Bitcoin signature schemes (ECDSA and Schnorr over secp256k1) are vulnerable to quantum attacks via Shor's algorithm [Sho94]. Hash-based signature schemes offer post-quantum security relying only on the hardness of breaking cryptographic hash functions, which are believed to resist quantum attacks apart from Grover's algorithm. The NIST-standardized stateless hash-based scheme SLH-DSA (also known as SPHINCS$^+$ [BHK+19]) provides a conservative post-quantum choice, but its signature sizes range from 7.8 KB to 50 KB, substantially larger than the 64-byte Schnorr signatures currently used in Bitcoin.

Two fundamental challenges arise when deploying hash-based signatures in Bitcoin:

1. **Signature size.** Large signatures impose direct costs in fee-per-byte accounting systems such as Bitcoin and reduce the block capacity for transactions.

2. **Stateful vs. stateless tradeoff.** Stateful schemes such as XMSS [BDH11] offer compact signatures but require careful state management: signing state cannot be recovered from a seed phrase alone, breaking the standard Bitcoin wallet model.

1.1 SHRINCS

SHRINCS (*Shrunken SPHINCS*)[KN25, Nic25, The25] is a composite post-quantum signature scheme that addresses both challenges simultaneously. It combines the efficiency of stateful signatures with the robustness of stateless ones under a single public key: the stateful component is a flexible XMSS scheme (FXMSS) whose tree shape the signer fixes at key generation, and the stateless component is SLH-DSA with a non-standard parameter set. Two tree shapes are prescribed: BXMSS (balanced), which allows for a bigger number of stateful signatures, and UXMSS (unbalanced), which minimizes the first signature size.

The key advantages of SHRINCS are the following.

- **Efficient stateful path.** During normal operation with intact state, SHRINCS uses the FXMSS tree with WOTS+C one-time signatures [HKRY23]. With the UXMSS shape, the q-th signature occupies $532 + 16q$ bytes for $q \leq 8$ (one more byte per further 8 levels, up to $q = 64$) and verifies with $242 + \min(q, 255)$ tweakable-hash calls, where $q \geq 1$ is the signature counter and the UXMSS depth is 255. With the BXMSS shape of depth D, all signatures have the constant size $531 + \lceil \min(D, 64)/8 \rceil + 16D$ bytes and verify with $242 + D$ tweakable-hash calls (e.g., 854 bytes and 262 calls at $D = 20$).

- **Robust stateless fallback.** When state is lost, corrupted, or exhausted, SHRINCS falls back to a stateless SLH-DSA signature (5777 bytes: a one-byte indicator and the 5776-byte SLH-DSA signature, roughly 2 KB smaller than SLH-DSA-SHA2-128s).

- **Static seed backup.** Users back up their wallet using a standard seed phrase. Upon recovery, the wallet operates in stateless-only mode, preserving funds while accepting larger signatures.

- **Capacity.** The construction supports up to 256 stateful signatures with the UXMSS shape, up to 2^D stateful signatures with the BXMSS shape of tree depth D (typically $D \leq 20$), and up to 2^{40} stateless signatures.

1.2 Our Contributions

In this work, we make the following contributions.

- **Complete specification.** We give a self-contained description of the SHRINCS signature scheme, including all building blocks (WOTS-TW, WOTS+C, XMSS, FXMSS with its

UXMSS and BXMSS shapes, FORS, and the SLH-DSA composition), the composite construction, its addressing scheme, and a concrete parameter set. The presentation of the scheme mainly follows the reference specification [The25] and is included here to keep this document self-contained.

- **Security proof.** We give the EU-CMA security proof of SHRINCS (Theorem 4.1), reducing its security to the properties of the underlying tweakable hash functions, message-digest functions, and pseudorandom functions. The proof builds on the SPHINCS$^+$ proof of [HK22] for the stateless path and on the SPHINCS+C proof of [HKRY23] for the WOTS+C block of the stateful path. It does not combine the two component bounds as black boxes: it opens both component proofs, charges each PRF replacement once, and lets the reductions of each path simulate the other path. The adversary chooses the FXMSS structure before key generation (Definition 4.15), so the bound holds for every admissible shape of the stateful tree; the flexible tree requires a variant of SM-TCR with exclusive tweaks (Definition 4.7).

Compared with the approach of [HK22, HKRY23], the proof differs in the following points.

- **Accounting of the reductions' work.** The tight SPHINCS$^+$ theorem of [HK22] states its hash-function terms at the forger's time, while its reductions complete the entire virtual structure: more than 2^{72} hash evaluations at the $h = 63$ parameter sets of FIPS 205. Charged as attack queries, this work alone sets a floor of about 2^{-52} on a single summand for $n = 128$, already for classical adversaries (Section 5.3). We route this work through a collection oracle restricted to tweaks disjoint from the challenge tweaks (Section 4.3), prove that for ideal collections the oracle changes the insecurities only by a parameter-guessing term, for classical and for quantum adversaries (Lemma 4.1), state the transfer to the concrete collection as a conjecture (Conjecture 4.1), and restate the SPHINCS$^+$ bound under this accounting (Theorem 4.3). Section 5 discusses what the resulting bounds certify.

- **Reprogramming.** The generic bounds of the message-digest properties and the proof of FXMSS reprogram the random oracle at every signing query. We state this step as a reprogramming property with distinct inputs (Definition C.2), share a single reprogramming hop between the two paths (Claim 4.14), and bound it more sharply: classically by exact bounds (Lemma C.4 and Corollaries C.2 and G.1) in place of the generic lemma, and quantumly by the lemma of Liao, Ge, Xue, and Zhou [LGXZ26] in place of that of [GHHM21], in one of two forms chosen per place (Section 6.5). Along the way we re-derive the generic ITSR bound of [YTA23] without the challenge count in the prefactor (Section F) and the quantum SM-rTCR bound of [DKKW25] with the new lemma (Section G). Classically, a tighter ITSR bound (Theorem F.1) avoids the reprogramming step and charges a hash query at a later challenge point only by the probability that it helps to cover a forgery.

- **Concrete bounds.** We evaluate every bound for the proposed parameters (Section 6). **Classically, the bounds certify between** 121.4 **and** 122.7 **bits in the query-counting sense of Section 5.1; quantumly, they reach** 1 **at** $2^{58.0}$ **to** $2^{59.2}$ **hash queries, with the quadratic quantum bounds for sm-ud and sm-pre of [Kud26b].** Compared with the generic reprogramming bounds, the sharper bounds add 34 bits classically on the stateless path and 33.4 bits for the full scheme, and quantumly 11.7 and 11.2 bits of the query count at which the bound reaches 1, where they also remove a query-independent floor of $2^{-3.4}$ (Remark 6.1). The figures rest on three conjectures: the SM-DSPR bound of Table 6 (stateless path and full scheme, Section 7), Conjecture 4.1 on the collection oracle (every figure that does not charge the reductions' collection-oracle work, Section 6.3), and Conjecture 3.1, that the counter search of the stateful signer always succeeds within its 2^{16} counters (stateful path and full scheme).

1.3 Organization

Section 2 introduces all building blocks of SHRINCS: the cryptographic hash function families and addressing scheme (Section 2.2), the WOTS-TW and WOTS+C one-time signature schemes (Sections 2.3 and 2.4), the XMSS construction and the flexible FXMSS scheme with its UXMSS and BXMSS shapes (Section 2.5), the FORS few-time signature scheme (Section 2.6), and the SLH-DSA composition of the stateless path (Section 2.7). Section 3 assembles these building blocks into the complete SHRINCS scheme. Section 4 states the hash-function definitions, the security notions, and the signature security model that the analysis uses, and contains the EU-CMA security proof, structured as a game sequence that reduces the security of SHRINCS to the properties of the underlying tweakable hash functions and pseudorandom functions. Section 5 discusses what these bounds certify: how the resources of a reduction should be accounted for, where the accounting of the tight SPHINCS$^+$ proof fails, and what the response adopted here costs. Section 6 evaluates the bounds for the proposed parameters, for classical and quantum adversaries, and collects the generic bounds of the hash-function properties (Section 6.2). Section 7 lists the open questions and the work in progress: smaller constants for the reprogramming lemma, removing the SM-DSPR conjecture, a tighter quantum bound for ITSR, and quantum reprogramming at distinct inputs. Appendices A and B provide self-contained specifications of HMAC and MGF1 as used in the scheme. Section C collects the (quantum) random-oracle tools: adaptive reprogramming, average-case search, and a two-stage point-function experiment. Section D gives the formal collection experiment and the proofs of the collection, independence, and domain-separation lemmas. Sections E to G derive the generic bounds for SM-TCR-EXCL, ITSR, and SM-rTCR used in Section 6; for ITSR the appendix also quotes the classical bound of [Kud26a] (Section F.1).

Acknowledgements

We thank Jonas Nick for finding and helping to fix the accounting problem in the tight SPHINCS$^+$ proof of [HK22] (Section 5.3), and Andreas Hülsing for discussions of the problem, its consequences, and the repair directions.

2 The Building Blocks of SHRINCS

SHRINCS is a composite post-quantum signature scheme operating in two modes: a **stateful** mode using a flexible XMSS tree (FXMSS) with WOTS+C one-time signatures, and a **stateless** fallback using SLH-DSA [Nat24] with a non-standard parameter set. Both modes share a single public key

$$\mathsf{PK} = \mathsf{PK.seed} \,\|\, \mathsf{PK.sl} \,\|\, \mathsf{PK.sf},$$

where PK.sl is the root of the stateless hypertree and PK.sf is the root of the stateful FXMSS tree. A signature from either mode is sufficient to pass verification: the verifier recomputes the root of the corresponding path from the signature and compares it against the matching component of PK. The encoding of PK is chosen such that dropping the trailing $n/8$ bytes (PK.sf) yields a valid SLH-DSA public key.

2.1 Parameters

SHRINCS is parameterized as in Table 1. In contrast to SLH-DSA, SHRINCS defines a single parameter set; all values in the table are fixed constants of the scheme[1]. The only per-key choice is the *structure* of the stateful tree (called `sf_structure` in the reference specification): a

[1]For cross-reference with the reference specification: l_{sf} = `WOTS_C_CHAIN_COUNT`, S_{wn} = `WOTS_C_CONSTANT_SUM`, l_1 = `WOTS_TW_CHAIN_COUNT1`, l_2 = `WOTS_TW_CHAIN_COUNT2`, d = `SPHX_LAYER_COUNT`, h' = `SPHX_XMSS_HEIGHT`, a = `SPHX_FORS_HEIGHT`, k = `SPHX_FORS_COUNT`, h_{sf} = `FXMSS_HEIGHT`.

Table 1: Parameters of SHRINCS

Parameter	Value	Description
n	128	Security parameter: hash output length (bits)
w	16	Winternitz parameter; each chain digit encodes $\log_2 w = 4$ bits
l_{sf}	32	Number of chains per WOTS+C instance (stateful path)
S_{wn}	240	WOTS+C target chain sum, $S_{wn} = \lceil l_{\mathsf{sf}}(w-1)/2 \rceil$
c	16	WOTS+C grinding-counter length (bits); at most 2^c attempts
$l_1,\ l_2$	32, 3	WOTS-TW message and checksum chains
l_{sl}	35	Number of chains per WOTS-TW instance, $l_{\mathsf{sl}} = l_1 + l_2$ (stateless path)
d	5	Number of hypertree layers (stateless path)
h'	9	Height of each XMSS tree within the hypertree
h_{sl}	45	Total hypertree height, $h_{\mathsf{sl}} = d \cdot h'$
k	10	Number of FORS trees
a	13	Height of each FORS tree ($t = 2^a$ leaves per tree)
m	192	Digest bits consumed by the stateless path, $m = 8\big(\lceil \frac{ka}{8} \rceil + \lceil \frac{h_{\mathsf{sl}} - h'}{8} \rceil + \lceil \frac{h'}{8} \rceil\big)$
h_{sf}	255	Height of the imaginary complete binary tree hosting the stateful FXMSS tree

two-byte value $(\mathsf{shape}, \mathsf{depth})$ selected at key generation, which determines the placement of the WOTS+C keys within an imaginary complete binary tree of height h_{sf} (see Section 2.5). The structure affects neither the hash-function instantiations nor the verification algorithm.

Remark 2.1. The structure of the stateful path is part of the secret key (Section 3). If the structure description is lost or corrupted, the signer cannot use the stateful path and must rely on the stateless path alone.

The stateless parameters (d, h', a, k) are calibrated for a budget of 2^{40} signatures, compared to 2^{64} for the standardized SLH-DSA parameter sets[2].

Remark 2.2. The length parameters n, c, and m are measured in *bits* throughout this document: $n = 128$, $c = 16$, $m = 192$. This differs from FIPS-205 and the reference specification, which measure the corresponding parameters in bytes ($n = 16$, $c = 2$, $m = 24$). The bit convention is the natural one for the security analysis, where hash domains and ranges are sets of bit strings $\{0,1\}^n$ and bounds are stated in terms of 2^{-n}. Serialized objects, such as keys, signatures, addresses, are still measured in bytes.

Both paths use an n-bit randomizer R in the message digest. The message digest itself is $2n = 256$ bits (32 bytes) on both paths: the stateless path consumes its first $m = 192$ bits, i.e., $24 = 17 + 5 + 2$ bytes (split into a FORS digest, a hypertree-leaf index, and an XMSS-leaf index; see Section 3), while the stateful path feeds the full $2n$-bit digest into the grinding hash $\mathbf{H}_{\text{grind}}$.

2.2 Cryptographic (Hash) Function Families

SHRINCS instantiates every hash function with SHA-256 truncated to its first n bits, i.e., $n/8 = 16$ bytes:

$$\mathsf{SHA\text{-}256}(m)[0{:}n/8].$$

For a byte string X we write $X[i{:}j]$ for the substring consisting of its bytes at positions $i, \ldots, j-1$, and $\mathsf{toByte}(x, y)$ for the y-byte big-endian encoding of the integer x; for a positive integer x we write $[x] = \{1, \ldots, x\}$.

Since PK.seed is 16 bytes, it is zero-padded to 64 bytes inside every tweakable-hash and **PRF** call: the padded seed $\mathsf{PK.seed} \| 0^{48}$ (48 zero bytes) fills exactly one SHA-256 message block, so its

[2]One can explore other parameters tradeoff using this source: https://blockstreamresearch.github.io/SPHINCS-Parameters/site/stateless.html

compression $P_{\mathsf{PK.seed}} = C(\mathsf{IV}, \mathsf{PK.seed}\|0^{48})$ can be precomputed once and reused across all calls. The message-digest functions $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}, \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ and the randomizer functions $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}, \mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$ absorb the bare 16-byte $\mathsf{PK.seed}$ without padding.

SHRINCS relies on multiple function families with different cryptographic properties. It uses tweakable hash functions (see Definition 4.2) to mitigate multi-target attacks [HRS16]. In addition, SHRINCS utilizes one pseudorandom function for secret-value derivation, two keyed functions (one per path) for randomizer generation, and two keyed hash functions (one per path) for computing the message digest. With these definitions, every hash call on the stateless path is bit-identical to the SHA2 instantiation of SLH-DSA [Nat24]; only the parameter set is non-standard.

Tweakable hash functions. A tweakable hash function of SHRINCS takes a public seed ($\in \mathcal{P}$), a tweak ($\in \mathcal{T}$), and an input ($\in \mathcal{M}$) and produces a digest ($\in \mathcal{N}$) (Definition 4.2). The tweakable hash functions of SHRINCS form the collection Th_λ, keyed by a single shared public parameter $\mathsf{PK.seed} \in \{0,1\}^n$: one member Th_ℓ per message length ℓn, and the grinding hash $\mathbf{H}_{\text{grind}}$ below. We fix $\mathcal{P} = \{0,1\}^n$ and $\mathcal{T} = \{0,1\}^{176}$: tweaks are the 22-byte addresses of Section 2.2. Every member other than $\mathbf{H}_{\text{grind}}$ is instantiated as

$$\mathsf{Th}_\ell : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^{\ell n} \to \{0,1\}^n,$$
$$\mathsf{Th}_\ell(\mathsf{PK.seed}, \mathbf{ADRS}, M_\ell) = \mathsf{SHA\text{-}256}\big(\mathsf{PK.seed} \,\|\, 0^{48} \,\|\, \mathbf{ADRS} \,\|\, M_\ell\big)[0{:}n/8].$$

Five members of this form are used, distinguished by their message length; with $\mathbf{H}_{\text{grind}}$, the collection has six members. The five are:

- $F \stackrel{\text{def}}{=} \mathsf{Th}_1$ — used to construct the hash chains in WOTS-TW and WOTS+C, and for leaf hashing in FORS;
- $H \stackrel{\text{def}}{=} \mathsf{Th}_2$ — used to compute internal nodes of every Merkle tree in the scheme (the XMSS trees of the hypertree, the FORS trees, and the stateful FXMSS tree);
- Th_k (with $k = 10$) — used to compress the k FORS roots;
- $\mathsf{Th}_{l_{\mathsf{sl}}}$ (with $l_{\mathsf{sl}} = 35$) — used to compress WOTS-TW public keys (stateless path);
- $\mathsf{Th}_{l_{\mathsf{sf}}}$ (with $l_{\mathsf{sf}} = 32$) — used to compress WOTS+C public keys (stateful path).

The reference specification names these members F, H, T_k, T_{sl}, and T_{sf}, respectively. Note that the message length determines the member: the six members occupy pairwise disjoint sets of SHA-256 input strings, so a query to one member is never an evaluation of another.

Grinding hash. The stateful path maps the message digest to a constant-sum base-w encoding by grinding a counter through the hash, whose tweak set $\mathcal{T}_{\mathsf{g}}$ is defined after the display

$$\mathbf{H}_{\text{grind}} : \{0,1\}^n \times \mathcal{T}_{\mathsf{g}} \times \{0,1\}^{2n} \times \{0,1\}^c \to \{0,1\}^n,$$
$$\mathbf{H}_{\text{grind}}(\mathsf{PK.seed}, \mathbf{ADRS}[0{:}10], md, \mathsf{ctr}) =$$
$$\mathsf{SHA\text{-}256}\big(\mathsf{PK.seed} \,\|\, 0^{48} \,\|\, \mathbf{ADRS}[0{:}10] \,\|\, md \,\|\, 0^4 \,\|\, \mathsf{toByte}(\mathsf{ctr}, c/8)\big)[0{:}n/8],$$

where $\mathbf{ADRS}[0{:}10]$ is the 10-byte prefix of a stateful address of type `SF_WOTS_C_GRIND` (Section 2.2): the node height, the node index, and the type byte. The 12-byte payload (all zero for this type) is dropped.

The truncation is aimed to improve performance without degrading security. With $P_{\mathsf{PK.seed}}$ precomputed, the remaining input ($10 + 32 + 4 + 2 = 48$ bytes plus SHA-256 padding) fits into a single compression call, so each grinding attempt costs exactly one compression. The four zero

bytes in front of the counter align it with a message-schedule word boundary: the two counter bytes are the only non-zero bytes of schedule word W_{11}, so an optimized implementation updates a single schedule word per attempt. The security argument requires all tweaks to be distinct; this is enforced by the first 10 bytes, which uniquely identify the anchor. As a member of the collection (Section 4.3), $\mathbf{H}_{\text{grind}}$ takes the full address as its tweak: its tweak set $\mathcal{T}_{\mathsf{g}} \subseteq \{0,1\}^{176}$ is the set of addresses of type `SF_WOTS_C_GRIND` with zero payload, which is the payload that Table 3 prescribes for this type. We keep writing $\mathbf{H}_{\text{grind}}(\mathsf{PK.seed}, \mathbf{ADRS}[0{:}10], md, \mathsf{ctr})$ and name a tweak of $\mathbf{H}_{\text{grind}}$ by its prefix $\mathbf{ADRS}[0{:}10]$. Its message is the pair (md, ctr) of $2n + c$ bits; the four zero bytes are a fixed part of the encoding.

Pseudorandom function for secret values. Secret chain and leaf values are derived from $\mathsf{SK.seed}$ on demand with the pseudorandom function

$$\mathbf{PRF} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{176} \to \{0,1\}^n,$$
$$\mathbf{PRF}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}) = \mathsf{SHA\text{-}256}\big(\mathsf{PK.seed} \,\|\, 0^{48} \,\|\, \mathbf{ADRS} \,\|\, \mathsf{SK.seed}\big)[0{:}n/8].$$

Observe that **PRF** has the same input shape as $F = \mathsf{Th}_1$; the two families are kept apart by the address type byte, since **PRF** is only ever invoked with the types `SL_WOTS_TW_PRF`, `SL_FORS_PRF`, and `SF_WOTS_C_PRF`, none of which is used with F (see Table 2).

Message digest. Each path computes its message digest with a dedicated keyed hash function. Both functions absorb an n-bit randomizer R, the bare (unpadded) $\mathsf{PK.seed}$, and the root of their own path, and both output $2n$ bits (32 bytes).

The *stateless* digest is

$$\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sl}, M) = \mathsf{SHA\text{-}256}\big(R \,\|\, \mathsf{PK.seed} \,\|\, \mathsf{SHA\text{-}256}(R \,\|\, \mathsf{PK.seed} \,\|\, \mathsf{PK.sl} \,\|\, M) \,\|\, 0^4\big).$$

The stateless path consumes the first m output bits, i.e., $m/8 = 24$ bytes (see Section 3). The trailing 4-byte zero word is the block counter of MGF1: since only one output block is needed, $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ coincides with $\mathsf{MGF1\text{-}SHA\text{-}256}\big(R\|\mathsf{PK.seed}\|\mathsf{SHA\text{-}256}(R\|\mathsf{PK.seed}\|\mathsf{PK.sl}\|M),\, 32\big)$, i.e., with the definition of $\mathbf{H}_{msg}$ for the SHA2 parameter sets of FIPS-205 (the specification of MGF1 is given in Section B).

The *stateful* digest additionally absorbs the position of the signing leaf idx. This position consists of two values: height of the WOTS+C instance used, and the leaf index on that height:

$$\mathsf{idx} = \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(i^\star, 8) \in \{0,1\}^{72}$$

This is the first 9 bytes of its stateful address (Section 2.2). Then

$$\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}, M) = \\ \mathsf{SHA\text{-}256}\big(R \,\|\, \mathsf{PK.seed} \,\|\, \mathsf{SHA\text{-}256}(R \,\|\, \mathsf{PK.seed} \,\|\, \mathsf{PK.sf} \,\|\, \mathsf{idx} \,\|\, M) \,\|\, \mathsf{idx}\big).$$

$\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ is a SHRINCS-specific construction, which mimics but also intentionally differs from $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ in two ways: the leaf identifier is bound into both calls, domain-separating the digest by the leaf used to sign it, and in the outer call it takes the place of the trailing zero word (MGF1 compatibility is not required off the FIPS-205 path). In the outer call idx is appended after the inner digest: the prefix $R \,\|\, \mathsf{PK.seed} \,\|\, \mathsf{SHA\text{-}256}(\cdots)$ fills exactly one SHA-256 block, so the leaf identifier occupies the final block, in the same way as the counter word of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$. The full $2n$-bit output is consumed by $\mathbf{H}_{\text{grind}}$. The verifier reconstructs idx from the signature, which makes the FXMSS verification routine agnostic to the tree shape (Section 2.5).

Randomizer generation. The signer derives the randomizer R pseudorandomly from $\mathsf{SK.prf}$, using HMAC-SHA-256 (specified in Section A):

$$\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}(\mathsf{SK.prf}, \mathit{opt_rand}, M) = \mathsf{HMAC\text{-}SHA\text{-}256}\big(\mathsf{SK.prf},\ \mathit{opt_rand} \,\|\, M\big)[0{:}n/8],$$
$$\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}(\mathsf{SK.prf}, \mathsf{PK.seed}, \mathsf{idx}, M) = \mathsf{HMAC\text{-}SHA\text{-}256}\big(\mathsf{SK.prf} \,\|\, \mathtt{0xFF}^{48},\ \mathsf{PK.seed} \,\|\, \mathsf{idx} \,\|\, M\big)[0{:}n/8].$$

On the stateless path, *opt_rand* is set either to $\mathsf{PK.seed}$ (the *deterministic* variant of SLH-DSA) or to a fresh random n-bit value (the *hedged* variant, which adds protection against side-channel attacks). The stateful variant is deterministic: per-signature uniqueness of the pair (idx, M) is guaranteed because each leaf signs at most once. The two variants use distinct HMAC keys for every value of $\mathsf{SK.prf}$: HMAC zero-pads keys shorter than its 64-byte block, so the stateless call effectively uses the key $\mathsf{SK.prf}\|\mathtt{0x00}^{48}$, while the stateful call explicitly pads with $\mathtt{0xFF}^{48}$. For the analysis we view the two variants as a single keyed function: the *combined randomizer function* $\mathbf{PRF}_{\mathbf{msg}}$ takes the key $\mathsf{SK.prf}$, a selector $b \in \{\mathsf{sl}, \mathsf{sf}\}$, and the arguments of the selected variant, and evaluates $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ or $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$ accordingly. The selector is realized by the key padding, so the two variants are the restrictions of $\mathbf{PRF}_{\mathbf{msg}}$ to the two selector values under the single key $\mathsf{SK.prf}$. The security analysis substitutes $\mathbf{PRF}_{\mathbf{msg}}$ once, for both paths together (Claim 4.12), which reflects that one secret underlies both derivations.

Domain separation. Every hash invocation in SHRINCS is bound to the path (stateless or stateful) it belongs to and to the role it plays there. Since both paths share the padded-seed construction and the single 22-byte address format, the separation is enforced by the following mechanisms (the figures in the proof of Lemma 4.3 depict the byte layout and SHA-256 block alignment of every hash call in the scheme):

- *Address types.* The type byte partitions all Th_ℓ and $\mathbf{PRF}$ calls between the paths: types 0–6 appear only on the stateless path and types 16–22 only on the stateful path (Table 2). Within a path, the type byte further separates the roles (chain hashing, tree hashing, key compression, secret-value derivation), following FIPS-205.

- *Input lengths.* Members Th_ℓ with distinct ℓ, as well as $\mathbf{H}_{\mathrm{grind}}$, have pairwise distinct SHA-256 input lengths, so no two of them can collide on the same input string. $\mathbf{PRF}$ shares its input length with $F = \mathsf{Th}_1$ and is separated from it by the type byte, as noted above.

- *Message digests.* $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ are separated from each other, and from all padded-seed and HMAC calls, by the layout of their SHA-256 inputs. Their outer calls have distinct fixed lengths (68 versus 73 bytes: $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ ends with a 4-byte zero word, whereas $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ absorbs the 9-byte leaf identifier idx), and their inner calls bind their own path's root ($\mathsf{PK.sl}$, resp. $\mathsf{PK.sf}$) immediately after $R \,\|\, \mathsf{PK.seed}$. The one non-obvious coincidence is an outer call of one digest matching an *inner* call of the other or of itself, which would correlate two digest values. For the scheme's own calls, the byte layout rules out an outer call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ matching an inner call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$; the other direction remains as a residual event (Lemma 4.3 (iii.b)), and Remark 4.2 discusses these coincidences in detail.

- *Randomizers.* $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$ use distinct HMAC keys for every $\mathsf{SK.prf}$, as described above.

- *Keyed functions vs. padded-seed functions.* The remaining separation, between the keyed functions ($\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$, $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$), which carry no address type, and the padded-seed functions above, can be seen for example through the byte structure of the hash inputs. One such separator is the pair of bytes 16–31 of a SHA-256 input. They are the zero padding 0^{16} in every Th_ℓ, $\mathbf{PRF}$, and $\mathbf{H}_{\mathrm{grind}}$ call, while they equal $\mathsf{PK.seed}$ in

every $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ call (both start with $R\|\mathsf{PK.seed}$); and they are a fixed nonzero pad constant ($\mathtt{0x36}^{16}$, $\mathtt{0x5C}^{16}$, $\mathtt{0xC9}^{16}$, or $\mathtt{0xA3}^{16}$) in both SHA-256 calls performed by $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$ (the ipad/opad-masked key blocks, Section A). The separation can be seen through other elements as well, but this is one of the examples.

Addressing scheme. For the tweakable hash functions to provide security against multi-target attacks, they must be invoked with distinct tweaks. This is accomplished through an *addressing scheme*, which assigns a unique address to each tweakable-hash call within the scheme. The address space is embedded within the tweak space; we use these terms interchangeably throughout this document.

SHRINCS uses a single 22-byte address format for all Th_ℓ and **PRF** calls on both paths[3]. An address consists of a 9-byte position prefix, a type byte, and a 12-byte payload of three 4-byte words; the interpretation of the prefix depends on the signing path (stateless or stateful) the call belongs to, and the interpretation of the payload depends on the type byte. Whenever the type in an **ADRS** is changed, the 12 payload bytes are initialized to zero, following the convention of FIPS-205, with two exceptions: if the new type has a key-pair field in Table 3, the key-pair address (payload bytes 10–13) keeps its previous value, as FIPS-205 does by re-setting it after each type change; and an address of type `SF_WOTS_C_PRF` keeps the two structure bytes in payload bytes 10–11. The main domain separator is the type byte; within a given type, the remaining address fields provide further separation.

Stateless-path prefix. On the stateless path, the prefix consists of a 1-byte *layer address* followed by an 8-byte *tree address* (Fig. 1). The layer address is the index of the hypertree layer containing the current XMSS tree: trees on the bottom layer have layer address zero, and the single tree at the top has layer address $d-1$. The tree address is the index of the XMSS tree within its layer; the leftmost tree has tree address zero.

layer address	tree address	*type*	payload
1 byte	8 bytes	1 byte	12 bytes

Figure 1: Stateless-path address layout (22 bytes).

Stateful-path prefix. On the stateful path, the prefix consists of a 1-byte *node height* followed by an 8-byte *node index* (Fig. 2). Together they identify a position (z, i) in the imaginary complete binary tree of height h_{sf} that hosts the FXMSS tree (Section 2.5): the node height z is the distance from the leaf level, with the root at height h_{sf}, and the node index i is the horizontal position within the level, counted from the left. The same pair doubles as the identifier of the WOTS+C instance anchored at that node. Since the node index is an 8-byte field, only the leftmost 2^{64} positions of a level are addressable; this bounds the admissible tree structures (Section 2.5) while keeping a fixed-length encoding. The stateful path consists of a single tree, so no analog of the layer and tree addresses is needed.

node height	node index	*type*	payload
1 byte	8 bytes	1 byte	12 bytes

Figure 2: Stateful-path address layout (22 bytes).

Address types. SHRINCS uses twelve address types, drawn from two disjoint ranges: types 0–6 live on the stateless path and retain their FIPS-205 values and semantics (FIPS-205 names them

[3]The 22-byte format coincides with the *compressed* address ADRS^c used inside the SHA2 instantiations of FIPS-205. SHRINCS adopts it directly as the address format, rather than first defining 32-byte addresses and compressing them inside the hash instantiations; the resulting hash inputs on the stateless path are bit-identical to SLH-DSA.

Table 2: Address types in SHRINCS.

Type	Name	Used when	Path
0	SL_WOTS_TW_HASH	computing WOTS-TW hash chains	Stateless
1	SL_WOTS_TW_PK	compressing WOTS-TW public keys	Stateless
2	SL_XMSS_TREE	hashing internal XMSS tree nodes	Stateless
3	SL_FORS_TREE	hashing FORS leaves and tree nodes	Stateless
4	SL_FORS_ROOTS	compressing the k FORS roots	Stateless
5	SL_WOTS_TW_PRF	deriving WOTS-TW secret values	Stateless
6	SL_FORS_PRF	deriving FORS secret values	Stateless
16	SF_WOTS_C_HASH	computing WOTS+C hash chains	Stateful
17	SF_WOTS_C_PK	compressing WOTS+C public keys	Stateful
18	SF_FXMSS_TREE	hashing internal FXMSS tree nodes	Stateful
21	SF_WOTS_C_PRF	deriving WOTS+C secret values	Stateful
22	SF_WOTS_C_GRIND	grinding WOTS+C message digests	Stateful

Table 3: Payload layout per address type (three 4-byte words; bytes 10–21 of the address).

Type	bytes 10–13	bytes 14–17	bytes 18–21
SL_WOTS_TW_HASH	key-pair address	chain address	hash address
SL_WOTS_TW_PK	key-pair address	0	0
SL_XMSS_TREE	0	tree height	tree index
SL_FORS_TREE	key-pair address	tree height	tree index
SL_FORS_ROOTS	key-pair address	0	0
SL_WOTS_TW_PRF	key-pair address	chain address	0
SL_FORS_PRF	key-pair address	0	tree index
SF_WOTS_C_HASH	0	chain address	hash address
SF_WOTS_C_PK	0	0	0
SF_FXMSS_TREE	0	0	0
SF_WOTS_C_PRF	$\mathsf{shape} \,\|\| \, \mathsf{depth} \,\|\| \, 0^2$	chain address	0
SF_WOTS_C_GRIND	0	0	0

WOTS_HASH, WOTS_PK, TREE, FORS_TREE, FORS_ROOTS, WOTS_PRF, FORS_PRF; this document uses the reference specification's SL-/SF-prefixed identifiers), while types 16–22 live on the stateful path. Each stateful type value equals the value of its stateless counterpart plus 16; the slots 19 and 20 (the counterparts of the FORS types) are unused, since FORS appears only on the stateless path, and the slot 22 is assigned to the grinding address, which has no stateless counterpart. Table 2 summarizes the types, and Table 3 defines the payload layout for each of them.

The SF_WOTS_C_PRF address is the only stateful address that carries the structure bytes $\mathsf{shape} \,\|\, \mathsf{depth}$ of the key (Section 2.5). The reason is the reuse of one seed under two structures, for example when a secret key is imported with incorrect structure bytes. Different structures can place an anchor at the same position: the first UXMSS signature ($\mathsf{ctr} = 0$) and the second BXMSS signature of depth 1 ($\mathsf{ctr} = 1$) both use the anchor at height 254 and index 1. Without the structure bytes, the two anchors would hold the same WOTS+C key, and signing one message under each structure would use a one-time key twice. With the structure bytes, the two keys derive different secret values. The security analysis of Section 4 fixes a single structure per key before key generation and does not model this setting.

Cost in compression calls. With $P_{\mathsf{PK.seed}}$ precomputed, the number of online SHA-256 compression calls per invocation is determined by the address and message lengths:

- F and **PRF** ($22 + 16$ message bytes): **one** call;
- H ($22 + 32 = 54$ bytes, which still fits into a single block together with the SHA-256 padding): **one** call;

- Th_k ($22 + 160$ bytes): **three** calls;
- $\mathsf{Th}_{l_{\mathsf{sf}}}$ ($22 + 512$ bytes): **nine** calls;
- $\mathsf{Th}_{l_{\mathsf{sl}}}$ ($22 + 560$ bytes): **ten** calls;
- $\mathbf{H}_{\text{grind}}$ ($10 + 32 + 4 + 2 = 48$ bytes): **one** call per counter attempt.

2.3 WOTS-TW

WOTS-TW is the one-time signature scheme underlying SPHINCS$^+$ [BHK+19]. We recall a self-contained description of WOTS-TW here. WOTS-TW is used in the stateless path of SHRINCS, exactly as in SLH-DSA [Nat24]. It also serves as the basis for the WOTS+C variant used in the stateful path of SHRINCS.

Parameters. The main security parameter is $n \in \mathbb{N}$ (in bits). Messages to be signed are n-bit strings. The Winternitz parameter $w \in \mathbb{N}$ fixes the base of the digit representation and determines the chain-length parameters:

$$l_1 = \left\lceil \frac{n}{\log_2 w} \right\rceil, \quad l_2 = \left\lfloor \frac{\log_2\big(l_1(w-1)\big)}{\log_2 w} \right\rfloor + 1, \quad l_{\mathsf{sl}} = l_1 + l_2.$$

For the SHRINCS parameters ($n = 128$, $w = 16$) this gives $l_1 = 32$, $l_2 = 3$, and $l_{\mathsf{sl}} = 35$. In SHRINCS, WOTS-TW uses the following instantiations of the building blocks defined in Section 2.2:

- A pseudorandom function $\mathbf{PRF} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{176} \to \{0,1\}^n$ for secret key derivation, called with an address of type `SL_WOTS_TW_PRF`.
- A tweakable hash function $F : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^n \to \{0,1\}^n$ for chain evaluation, called with an address of type `SL_WOTS_TW_HASH`.
- A tweakable hash function $\mathsf{Th}_{l_{\mathsf{sl}}} : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^{l_{\mathsf{sl}} n} \to \{0,1\}^n$ for public key compression, called with an address of type `SL_WOTS_TW_PK`.

The public seed $\mathsf{PK.seed} \in \{0,1\}^n$ serves as the public parameter for all three functions. Uniqueness of each hash call within a larger construction is enforced by the addressing scheme described in Section 2.2: each invocation receives a distinct $\mathbf{ADRS} \in \{0,1\}^{176}$ encoding its position in the global structure.

WOTS-TW scheme. The central building block of WOTS-TW is the chaining function, which iterates the tweakable hash function F along a chain while updating the hash address in **ADRS** at each step. The message encoding maps an n-bit input to l_{sl} base-w digits, and each digit determines how far along the corresponding chain the signature element sits.
Chaining function $c^{j,k}(x, i, \mathsf{PK.seed}, \mathbf{ADRS})$: The chaining function takes a value $x \in \{0,1\}^n$, a start index $j \in \mathbb{N}$, an iteration count $k \in \mathbb{N}$, a chain index i, the public seed $\mathsf{PK.seed}$, and an address **ADRS** of type `SL_WOTS_TW_HASH` with chain address set to i. It requires $j + k \leq w - 1$: a chain position never exceeds the end of its Winternitz chain. It is defined recursively:

- For $k \leq 0$, the function returns x unchanged:

$$c^{j,0}(x, i, \mathsf{PK.seed}, \mathbf{ADRS}) = x.$$

- For $k > 0$, let $\mathbf{ADRS}'$ be **ADRS** with the hash address set to $j + k - 1$; then

$$c^{j,k}(x, i, \mathsf{PK.seed}, \mathbf{ADRS}) = F\Big(\mathsf{PK.seed},\ \mathbf{ADRS}',\ c^{j,k-1}(x, i, \mathsf{PK.seed}, \mathbf{ADRS})\Big).$$

Message encoding. An n-bit message m is interpreted as the sequence of its l_1 base-w digits (for $w = 16$, its nibbles, most significant first):

$$\mathsf{m} = (a_0, \ldots, a_{l_1-1}), \quad a_i \in \{0, \ldots, w-1\}.$$

The checksum

$$C = \sum_{i=0}^{l_1-1} (w - 1 - a_i) \ \in \ \{0, \ldots, l_1(w-1)\}$$

is appended as its big-endian base-w representation with exactly l_2 digits: $C = \sum_{i=0}^{l_2-1} C_i \cdot w^{l_2-1-i}$ with $C_i \in \{0, \ldots, w-1\}$. The encoded message is $B = (b_0, \ldots, b_{l_{\mathsf{sl}}-1}) = (a_0, \ldots, a_{l_1-1}, C_0, \ldots, C_{l_2-1})$.

Key Generation Algorithm $(\mathsf{SK}, \mathsf{PK}) \leftarrow$ **WOTS-TW**.$\mathsf{kg}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS})$**:**

The key generation algorithm receives the public seed $\mathsf{PK.seed} \in \{0,1\}^n$, the secret seed $\mathsf{SK.seed} \in \{0,1\}^n$, and a global address $\mathbf{ADRS} \in \{0,1\}^{176}$ identifying the WOTS-TW instance within the larger scheme. The l_{sl} secret key elements are derived deterministically: for each $i \in \{0, \ldots, l_{\mathsf{sl}} - 1\}$, let $\mathbf{ADRS}_i$ be $\mathbf{ADRS}$ with type set to `SL_WOTS_TW_PRF` and chain address set to i, and compute

$$\mathsf{sk}_i \leftarrow \mathbf{PRF}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}_i).$$

Each chain is then run to its end to obtain the public key elements. Let $\mathbf{ADRS}^{\mathsf{h}}$ be $\mathbf{ADRS}$ with type set to `SL_WOTS_TW_HASH`; then

$$\mathsf{pk}_i = c^{0,\,w-1}(\mathsf{sk}_i,\ i,\ \mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{h}}), \quad i \in \{0, \ldots, l_{\mathsf{sl}} - 1\}.$$

The l_{sl} public key elements are compressed into a single n-bit value using $\mathsf{Th}_{l_{\mathsf{sl}}}$. Let $\mathbf{ADRS}^{\mathsf{pk}}$ be $\mathbf{ADRS}$ with type set to `SL_WOTS_TW_PK`; then

$$\mathsf{pk} = \mathsf{Th}_{l_{\mathsf{sl}}}(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{pk}},\ \mathsf{pk}_0 \| \cdots \| \mathsf{pk}_{l_{\mathsf{sl}}-1}).$$

The algorithm outputs $\mathsf{sk} = (\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS})$ and $\mathsf{PK} = (\mathsf{pk}, \mathsf{PK.seed}, \mathbf{ADRS})$.

Signature Algorithm $\sigma \leftarrow$ **WOTS-TW**.$\mathsf{sign}(\mathsf{m}, \mathsf{SK})$**:**

Given an n-bit message m and secret key $\mathsf{SK} = (\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS})$, the algorithm computes the encoded message $B = (b_0, \ldots, b_{l_{\mathsf{sl}}-1})$ as described above. The secret key elements are re-derived as in key generation, and the signature is

$$\sigma = (\sigma_0, \ldots, \sigma_{l_{\mathsf{sl}}-1}) = \big(c^{0,\,b_0}(\mathsf{sk}_0, 0, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}}),\ \ldots,\ c^{0,\,b_{l_{\mathsf{sl}}-1}}(\mathsf{sk}_{l_{\mathsf{sl}}-1}, l_{\mathsf{sl}}-1, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}})\big),$$

where $\mathbf{ADRS}^{\mathsf{h}}$ is as defined in key generation. A WOTS-TW signature consists of $l_{\mathsf{sl}} \cdot n/8 = 560$ bytes.

Verification Algorithm $b \leftarrow$ **WOTS-TW**.$\mathsf{vf}(\mathsf{m}, \sigma, \mathsf{PK})$**:**

Given m, a signature $\sigma = (\sigma_0, \ldots, \sigma_{l_{\mathsf{sl}}-1})$, and public key $\mathsf{PK} = (\mathsf{pk}, \mathsf{PK.seed}, \mathbf{ADRS})$, the verifier recomputes $(b_0, \ldots, b_{l_{\mathsf{sl}}-1})$ as above and completes each chain from its current position:

$$\mathsf{pk}' = \mathsf{Th}_{l_{\mathsf{sl}}}\Big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{pk}},\ c^{b_0,\,w-1-b_0}(\sigma_0, 0, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}}) \ \| \ \cdots \\ \| \ c^{b_{l_{\mathsf{sl}}-1},\,w-1-b_{l_{\mathsf{sl}}-1}}(\sigma_{l_{\mathsf{sl}}-1}, l_{\mathsf{sl}}-1, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}})\Big).$$

The algorithm outputs 1 if $\mathsf{pk}' = \mathsf{pk}$ and 0 otherwise.

Public Key Recovery $\mathsf{pk} \leftarrow$ **WOTS-TW**.$\mathsf{pkFromSig}(\mathsf{m}, \sigma, \mathsf{PK.seed}, \mathbf{ADRS})$**:**

In SLH-DSA, the WOTS-TW public key is never verified directly. Instead, it is recomputed from the signature and fed as a leaf into an XMSS tree; the Merkle authentication path then establishes its authenticity as part of the global SLH-DSA verification algorithm. The following algorithm captures this recomputation.

Given an n-bit message m, a signature $\sigma = (\sigma_0, \ldots, \sigma_{l_{\mathsf{sl}}-1})$, the public seed $\mathsf{PK.seed}$, and an address $\mathbf{ADRS}$ identifying the WOTS-TW instance, the algorithm recomputes the digit vector $(b_0, \ldots, b_{l_{\mathsf{sl}}-1})$ as in signing and completes each chain:

$$\mathsf{pk}'_i = c^{b_i,\, w-1-b_i}(\sigma_i,\, i,\, \mathsf{PK.seed},\, \mathbf{ADRS}^{\mathsf{h}}), \quad i \in \{0, \ldots, l_{\mathsf{sl}} - 1\}.$$

It then compresses the result and returns

$$\mathsf{pk}' = \mathsf{Th}_{l_{\mathsf{sl}}}(\mathsf{PK.seed},\, \mathbf{ADRS}^{\mathsf{pk}},\, \mathsf{pk}'_0 \| \cdots \| \mathsf{pk}'_{l_{\mathsf{sl}}-1}).$$

The one-time security of WOTS-TW rests on the following observation. An adversary who observes a valid pair (m, σ) and attempts to forge (m', σ') with $\mathsf{m}' \neq \mathsf{m}$ must produce a chain value that precedes some σ_i in at least one chain. This is forced by the checksum: if every digit in the encoding of m' were at least as large as the corresponding digit in the encoding of m, the derived checksum C' would be strictly smaller than C, so the base-w representation of C' must contain at least one digit strictly smaller than the corresponding digit of C. Consequently, any successful forgery requires inverting F at some chain position, which contradicts the security of the underlying tweakable hash function.

2.4 WOTS+C

WOTS+C (Compressed WOTS+) [HKRY23] is the one-time signature scheme used in the stateful path of SHRINCS. It is a variant of WOTS-TW that replaces the WOTS+ checksum with a grinding step on the message digest: the signer iterates a counter until the tweakable hash of the digest and counter has a base-w digit sum equal to a fixed target. This eliminates the l_2 checksum chains and shrinks both signature size and verification cost, at the price of a signing-time search.

Parameters. The main security parameter is $n \in \mathbb{N}$ (in bits). Messages passed to WOTS+C are the $2n$-bit digests produced by $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$. The Winternitz parameter $w \in \mathbb{N}$ fixes the base of the digit representation; the chain-count parameter is

$$l_{\mathsf{sf}} = \left\lceil \frac{n}{\log_2 w} \right\rceil,$$

and the target chain-sum is

$$S_{wn} = \left\lceil \frac{l_{\mathsf{sf}}(w-1)}{2} \right\rceil,$$

the digit sum maximizing the number of valid encodings. The grinding counter is a c-bit value, bounding the search to at most 2^c attempts. For the SHRINCS parameters ($n = 128$, $w = 16$, $c = 16$) this gives $l_{\mathsf{sf}} = 32$ and $S_{wn} = 240$.

In SHRINCS, WOTS+C uses the following instantiations of the building blocks defined in Section 2.2:

- A pseudorandom function $\mathbf{PRF} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{176} \to \{0,1\}^n$ for secret key derivation, called with an address of type `SF_WOTS_C_PRF`.
- A tweakable hash function $F : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^n \to \{0,1\}^n$ for chain evaluation, called with an address of type `SF_WOTS_C_HASH`.
- A tweakable hash function $\mathbf{H}_{\text{grind}} : \{0,1\}^n \times \mathcal{T}_{\mathsf{g}} \times \{0,1\}^{2n} \times \{0,1\}^c \to \{0,1\}^n$ for digest grinding, called with an address of type `SF_WOTS_C_GRIND`, of which it absorbs the 10-byte prefix (see Section 2.2).

- A tweakable hash function $\mathsf{Th}_{l_{\mathsf{sf}}} : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^{l_{\mathsf{sf}} n} \to \{0,1\}^n$ for public key compression, called with an address of type `SF_WOTS_C_PK`. Note that the number of chains $l_{\mathsf{sf}} = 32$ differs from the $l_{\mathsf{sl}} = 35$ chains of WOTS-TW, so the two public-key compression functions are distinct members of the collection Th_λ.

The public seed $\mathsf{PK.seed} \in \{0,1\}^n$ serves as the public parameter for all four functions. Uniqueness of each hash call within a larger construction is enforced by the addressing scheme described in Section 2.2: each invocation receives a distinct $\mathbf{ADRS} \in \{0,1\}^{176}$ following the stateful layout of Fig. 2, with the node-height and node-index fields carrying the position $(z^\star, i^\star)$ of the WOTS+C instance in the imaginary tree of Section 2.5.

WOTS+C scheme. The chaining function and key derivation are inherited from WOTS-TW without modification, up to the stateful address types (the reference code accordingly defines a separate chain-iteration function per scheme, each pinning its own address type). The only change is in how chain depths are determined from the message: rather than a checksum-extended digit vector, WOTS+C uses the digits of a grinded digest whose digit sum is forced to equal S_{wn}.
Chaining function $c^{j,k}(x, i, \mathsf{PK.seed}, \mathbf{ADRS})$: The chaining function is identical to that of WOTS-TW (Section 2.3). It takes a value $x \in \{0,1\}^n$, a start index $j \in \mathbb{N}$, an iteration count $k \in \mathbb{N}$, a chain index i, the public seed $\mathsf{PK.seed}$, and an address $\mathbf{ADRS}$ of type `SF_WOTS_C_HASH` with chain address set to i.
Grinding step $(d, \mathsf{ctr}) \leftarrow \mathsf{grind}(\mathsf{m}, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{g}})$: Given a $2n$-bit digest m, the public seed $\mathsf{PK.seed}$, and an address $\mathbf{ADRS}^{\mathsf{g}}$ of type `SF_WOTS_C_GRIND`, the algorithm iterates $\mathsf{ctr} = 0, 1, 2, \ldots, 2^c - 1$ in increasing order and computes

$$d = \mathbf{H}_{\mathrm{grind}}(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{g}}[0{:}10],\ \mathsf{m},\ \mathsf{ctr}).$$

It returns the first pair (d, ctr) such that the base-w representation $(a_0, \ldots, a_{l_{\mathsf{sf}}-1})$ of d satisfies $\sum_{i=0}^{l_{\mathsf{sf}}-1} a_i = S_{wn}$. If no counter in range satisfies the condition, the algorithm fails and returns $\perp$[4]. The iteration order is normative: running grind on the same inputs must always return the same pair (d, ctr), i.e., the smallest valid counter. Otherwise, the security might be violated. The verifier, in contrast, accepts *any* counter value that maps the digest to a constant-sum digit vector; it does not check minimality.
Key Generation Algorithm $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathbf{WOTS+C}.\mathsf{kg}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS})$:

The key generation algorithm is identical to that of WOTS-TW, with l_{sf} in place of l_{sl} and the stateful address types in place of the stateless ones. It receives the public seed $\mathsf{PK.seed} \in \{0,1\}^n$, the secret seed $\mathsf{SK.seed} \in \{0,1\}^n$, and a global address $\mathbf{ADRS} \in \{0,1\}^{176}$ (with node-height and node-index fields set to the imaginary-tree position $(z^\star, i^\star)$ of this WOTS+C instance, and with payload bytes 10–11 holding the structure bytes of the key, Section 2.5). The l_{sf} secret key elements are derived deterministically: for each $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$, let $\mathbf{ADRS}_i$ be $\mathbf{ADRS}$ with type set to `SF_WOTS_C_PRF`, the structure bytes in payload bytes 10–11 kept (Table 3), and chain address set to i, and compute

$$\mathsf{sk}_i \leftarrow \mathbf{PRF}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}_i).$$

Each chain is then run to its end to obtain the public key elements. Let $\mathbf{ADRS}^{\mathsf{h}}$ be $\mathbf{ADRS}$ with type set to `SF_WOTS_C_HASH`; then

$$\mathsf{pk}_i = c^{0,\,w-1}(\mathsf{sk}_i,\ i,\ \mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{h}}), \quad i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}.$$

[4]For the SHRINCS parameters, a uniformly random digest maps to a constant-sum digit vector with probability $\approx 2^{-6}$, so the probability that all 2^{16} counter values fail is below 2^{-1450}. This bound is parameter-specific; other parameter sets can give substantially weaker bounds.

The l_{sf} public key elements are compressed into a single n-bit value using $\mathsf{Th}_{l_{\mathsf{sf}}}$. Let $\mathbf{ADRS}^{\mathsf{pk}}$ be $\mathbf{ADRS}$ with type set to `SF_WOTS_C_PK`; then

$$\mathsf{pk} = \mathsf{Th}_{l_{\mathsf{sf}}}(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{pk}},\ \mathsf{pk}_0 \| \cdots \| \mathsf{pk}_{l_{\mathsf{sf}}-1}).$$

The algorithm outputs $\mathsf{SK} = (\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS})$ and $\mathsf{PK} = (\mathsf{pk}, \mathsf{PK.seed}, \mathbf{ADRS})$.

Signature Algorithm $\sigma \leftarrow$ WOTS+C.sign$(\mathsf{m}, \mathsf{SK})$:

Given a $2n$-bit digest m and secret key $\mathsf{SK} = (\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS})$, the algorithm proceeds as follows. Let $\mathbf{ADRS}^{\mathsf{g}}$ be $\mathbf{ADRS}$ with type set to `SF_WOTS_C_GRIND`, and run the grinding step

$$(d, \mathsf{ctr}) \leftarrow \mathsf{grind}(\mathsf{m}, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{g}}),$$

If grind returns $\perp$, WOTS+C.sign returns $\perp$. Otherwise it obtains a digest d with base-w representation

$$d = (a_0, \ldots, a_{l_{\mathsf{sf}}-1}), \quad a_i \in \{0, \ldots, w-1\}, \quad \sum_{i=0}^{l_{\mathsf{sf}}-1} a_i = S_{wn}.$$

The secret key elements sk_i are re-derived as in key generation, and the signature is

$$\begin{aligned}\sigma &= \big(\mathsf{toByte}(\mathsf{ctr}, c/8),\ \sigma_0, \ldots, \sigma_{l_{\mathsf{sf}}-1}\big) \\ &= \big(\mathsf{toByte}(\mathsf{ctr}, c/8),\ c^{0,a_0}(\mathsf{sk}_0, 0, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}}),\ \ldots,\ c^{0,a_{l_{\mathsf{sf}}-1}}(\mathsf{sk}_{l_{\mathsf{sf}}-1}, l_{\mathsf{sf}}-1, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}})\big),\end{aligned}$$

where $\mathbf{ADRS}^{\mathsf{h}}$ is as defined in key generation. The counter is serialized first, so a WOTS+C signature consists of $c/8 + l_{\mathsf{sf}} \cdot n/8 = 514$ bytes.

Verification Algorithm $b \leftarrow$ WOTS+C.vf$(\mathsf{m}, \sigma, \mathsf{PK})$:

Given m, a signature $\sigma = (\mathsf{ctr}, \sigma_0, \ldots, \sigma_{l_{\mathsf{sf}}-1})$, and public key $\mathsf{PK} = (\mathsf{pk}, \mathsf{PK.seed}, \mathbf{ADRS})$, the verifier recomputes the digest

$$d = \mathbf{H}_{\mathrm{grind}}(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{g}}[0{:}10],\ \mathsf{m},\ \mathsf{ctr})$$

and parses it as $(a_0, \ldots, a_{l_{\mathsf{sf}}-1})$. It rejects unless $\sum_{i=0}^{l_{\mathsf{sf}}-1} a_i = S_{wn}$. Otherwise, it completes each chain from its current position and recompresses:

$$\begin{aligned}\mathsf{pk}' = \mathsf{Th}_{l_{\mathsf{sf}}}\Big(&\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{pk}},\ c^{a_0,\,w-1-a_0}(\sigma_0, 0, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}}) \ \|\ \cdots \\ &\|\ c^{a_{l_{\mathsf{sf}}-1},\,w-1-a_{l_{\mathsf{sf}}-1}}(\sigma_{l_{\mathsf{sf}}-1}, l_{\mathsf{sf}}-1, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{h}})\Big).\end{aligned}$$

The algorithm outputs 1 if $\mathsf{pk}' = \mathsf{pk}$ and 0 otherwise.

Public Key Recovery $\mathsf{pk} \leftarrow$ WOTS+C.pkFromSig$(\mathsf{m}, \sigma, \mathsf{PK.seed}, \mathbf{ADRS})$:

As in WOTS-TW, the WOTS+C public key is never verified directly within SHRINCS. Instead, it is recomputed from the signature and fed as a leaf into the surrounding FXMSS tree; the Merkle authentication path then establishes its authenticity as part of the global verification algorithm. The following algorithm captures this recomputation.

Given a $2n$-bit digest m, a signature $\sigma = (\mathsf{ctr}, \sigma_0, \ldots, \sigma_{l_{\mathsf{sf}}-1})$, the public seed $\mathsf{PK.seed}$, and an address $\mathbf{ADRS}$ identifying the WOTS+C instance, the algorithm recomputes the digit vector $(a_0, \ldots, a_{l_{\mathsf{sf}}-1})$ as in verification. If the digit sum is not S_{wn}, it returns $\perp$; the caller must treat $\perp$ as a verification failure. Otherwise, it completes each chain:

$$\mathsf{pk}'_i = c^{a_i,\,w-1-a_i}(\sigma_i,\ i,\ \mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{h}}), \quad i \in \{0, \ldots, l_{\mathsf{sf}}-1\}.$$

It then compresses the result and returns

$$\mathsf{pk}' = \mathsf{Th}_{l_{\mathsf{sf}}}(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{pk}},\ \mathsf{pk}'_0 \| \cdots \| \mathsf{pk}'_{l_{\mathsf{sf}}-1}).$$

The one-time security of WOTS+C rests on the same observation as WOTS-TW, with the fixed digit sum playing the role of the WOTS+ checksum. An adversary who observes a valid pair (m, σ) and attempts to forge (m', σ') with $\mathsf{m}' \neq \mathsf{m}$ must produce a chain value preceding some σ_i in at least one chain. This is forced by the digit-sum constraint: both digit vectors $(a_0, \ldots, a_{l_{\mathsf{sf}}-1})$ and $(a'_0, \ldots, a'_{l_{\mathsf{sf}}-1})$ derived from the grinded digests sum to S_{wn}, so if every a'_i were at least a_i, equality of sums would force $a'_i = a_i$ for all i, in which case the forger must find a target collision for $\mathbf{H}_{\text{grind}}$ (under fixed $\mathsf{PK.seed}$ and $\mathbf{ADRS}^{\mathsf{g}}$) to obtain a different message yielding the same constrained digest. Otherwise some $a'_i < a_i$ and the forger must invert F at some chain position.

2.5 XMSS and FXMSS

XMSS [BDH11] is a many-time hash-based signature scheme built by combining one-time signatures with a Merkle tree: the leaves are one-time public keys, and the root serves as a single short public key. A signature reveals one leaf together with the authentication path to the root. SHRINCS instantiates this template twice:

- *XMSS* of height $h' = 9$ with WOTS-TW leaves, a complete binary tree, is the building block of the SLH-DSA hypertree in the stateless path (Section 2.7).
- *Flexible XMSS* (FXMSS) with WOTS+C leaves is the stateful component. The signer chooses the shape of its tree at key-generation time; the verifier is agnostic to that choice.

In both instantiations the internal-node hashing follows the same template, using $H = \mathsf{Th}_2$; the differences are the tree shape and the addresses (`SL_XMSS_TREE` with the stateless prefix versus `SF_FXMSS_TREE` with the stateful prefix, Section 2.2). One distinction is crucial for security: within the hypertree, XMSS only ever signs values generated by the signer itself (roots of child trees and FORS public keys), whereas FXMSS signs digests of adversarially chosen messages. The interfaces of the two instantiations differ accordingly.

XMSS (stateless path). An XMSS tree of height h' is a complete binary tree with $2^{h'}$ leaves, each of which is the public key of a WOTS-TW instance. Internal nodes are the tweakable hash of their two children. Nodes are indexed by their height $z \in \{0, \ldots, h'\}$ (the distance from the leaf level, so $z = 0$ for leaves and $z = h'$ for the root) and their horizontal index $j \in \{0, \ldots, 2^{h'-z} - 1\}$, counted from the left. The position of the tree itself within the hypertree is fixed by the layer and tree addresses of **ADRS**, which are common to all hash calls of the instance.
Tree-hash function $\mathsf{xmss_root}(\mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS})$**:** For a node at height z and index j, let $\mathbf{ADRS}^{\mathsf{tr}}_{z,j}$ denote **ADRS** with type set to `SL_XMSS_TREE`, tree height set to z, and tree index set to j. The node $N_{z,j}$ is defined recursively:

- For $z = 0$, the leaf is the WOTS-TW public key at index j:

$$N_{0,j} = \text{WOTS-TW}.\mathsf{kg}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}_j).\mathsf{pk},$$

  where $\mathbf{ADRS}_j$ is **ADRS** with the key-pair address set to j.

- For $z > 0$,

$$N_{z,j} = H\big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{tr}}_{z,j},\ N_{z-1,\,2j} \,\|\, N_{z-1,\,2j+1}\big)\,.$$

The root is $N_{h',0}$.
Signature Algorithm $\sigma_{\mathbf{XMSS}} \leftarrow \mathbf{XMSS}.\mathsf{sign}(\mathsf{m}, \mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS}, q)$**:**
Given an n-bit message m, the seeds $\mathsf{SK.seed}, \mathsf{PK.seed}$, an address **ADRS** identifying the XMSS instance within the hypertree, and a leaf index $q \in \{0, \ldots, 2^{h'} - 1\}$, the algorithm computes

the one-time signature $\sigma_q \leftarrow \text{WOTS-TW.}\mathsf{sign}(\mathsf{m}, \mathsf{SK}_q)$, where SK_q is the leaf secret key derived from $\mathsf{SK.seed}$ for the address with key-pair address q. The authentication path is

$$\mathsf{auth} = (A_0, \ldots, A_{h'-1}), \quad A_z = N_{z,\lfloor q/2^z \rfloor \oplus 1},$$

i.e., A_z is the sibling of the node on the leaf-to-root path at height z. The algorithm returns $\sigma_{\text{XMSS}} = (\sigma_q, \mathsf{auth})$, a string of $(l_{\mathsf{sl}} + h') \cdot n/8$ bytes.

Public Key Recovery $\mathsf{pk}_{\mathbf{XMSS}} \leftarrow \mathbf{XMSS}.\mathsf{pkFromSig}(\mathsf{m}, \sigma_{\mathbf{XMSS}}, \mathsf{PK.seed}, \mathbf{ADRS}, q)$:

As with WOTS-TW, the XMSS root is not verified directly within SHRINCS. It is recomputed from the signature and either compared against a published root or fed as the message into the next layer of the hypertree (Section 2.7).

Given m, $\sigma_{\text{XMSS}} = (\sigma_q, \mathsf{auth})$, $\mathsf{PK.seed}$, $\mathbf{ADRS}$, and q, the algorithm first recovers the leaf one-time public key

$$N'_{0,q} = \text{WOTS-TW.}\mathsf{pkFromSig}(\mathsf{m}, \sigma_q, \mathsf{PK.seed}, \mathbf{ADRS}_q).$$

It then walks up the tree, combining each candidate node with the corresponding sibling from auth:

$$N'_{z+1,\lfloor q/2^{z+1} \rfloor} = \begin{cases} H(\mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{tr}}_{z+1,\lfloor q/2^{z+1} \rfloor}, N'_{z,\lfloor q/2^z \rfloor} \,\|\, A_z) & \text{if } \lfloor q/2^z \rfloor \text{ is even,} \\ H(\mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{tr}}_{z+1,\lfloor q/2^{z+1} \rfloor}, A_z \,\|\, N'_{z,\lfloor q/2^z \rfloor}) & \text{otherwise.} \end{cases}$$

The algorithm returns $\mathsf{pk}_{\text{XMSS}} = N'_{h',0}$.

FXMSS: the stateful component. Different wallets have different usage patterns: an on-chain wallet that avoids address reuse signs a handful of times per key and wants its first signatures to be as small as possible, while a high-frequency signer wants a large budget of constant-size signatures. FXMSS (Flexible XMSS) serves both patterns with a single verification routine by letting the signer choose, at key-generation time, which positions of a binary tree carry WOTS+C keys. The verifier neither knows nor checks the signer's tree shape: it recomputes the root from a single WOTS+C signature, the position of the signing key, and an authentication path.

Imaginary complete-tree embedding. Fix the declared height $h_{\mathsf{sf}} = 255$ (set by the 1-byte node-height field of the stateful address, Section 2.2). The FXMSS tree is embedded into an imaginary complete binary tree of height h_{sf}: nodes are identified by their height $z \in \{0, \ldots, h_{\mathsf{sf}}\}$ (distance from the leaf level, with the root at $z = h_{\mathsf{sf}}$) and their horizontal index $i \in \{0, \ldots, 2^{h_{\mathsf{sf}}-z} - 1\}$, counted from the left. We call $\delta = h_{\mathsf{sf}} - z$ the *depth* of a node. The path length from (z, i) to the root is δ. Since the node-index field of the stateful address is 8 bytes, only positions with $i < 2^{64}$ are addressable[5].

A WOTS+C key may be placed at *any* addressable node $(z^\star, i^\star)$, not only at the leaf level $z = 0$.

Tree structure. Which nodes carry WOTS+C keys is determined by the *structure*, a two-byte value

$$\mathsf{structure} = (\mathsf{shape}, \mathsf{depth}),$$

chosen at key generation and stored in the SHRINCS secret key (Section 3). The shape byte selects one of two prescribed shapes: $\texttt{SHAPE_UNBALANCED} = 0$ (UXMSS) or $\texttt{SHAPE_BALANCED} = 1$ (BXMSS), with values 2–255 reserved for future shapes. The depth byte is the distance from the root to the deepest WOTS+C placement. A structure defines a set $\mathcal{A}$ of *anchors* (positions carrying WOTS+C keys) and a set $\mathcal{I}$ of internal nodes, given in depth coordinates $\delta = h_{\mathsf{sf}} - z$ as follows.

[5]This bounds the admissible tree structures: while the imaginary tree is 255 levels deep, only the leftmost 2^{64} nodes of each level can host keys or tree nodes. This provides enough space while keeping a fixed-length index encoding.

- UXMSS (unbalanced, left-leaning): a spine of internal nodes descends along horizontal index 0, with a WOTS+C anchor hanging off at index 1 on every level, and a final anchor terminating the spine:

$$\mathcal{A} = \big\{(\delta, 1) : 1 \le \delta \le \mathsf{depth}\big\} \cup \big\{(\mathsf{depth}, 0)\big\}, \qquad \mathcal{I} = \big\{(\delta, 0) : 0 \le \delta < \mathsf{depth}\big\}.$$

  The tree has $\mathsf{depth} + 1$ anchors. Any $1 \le \mathsf{depth} \le 255$ is admissible; we recommend the maximum, $\mathsf{depth} = 255$, since even the deepest UXMSS anchor yields a stateful signature smaller than a stateless one.

- BXMSS (balanced): a complete binary tree of height depth whose 2^{depth} leaves are all anchors:

$$\mathcal{A} = \big\{(\mathsf{depth}, i) : 0 \le i < 2^{\mathsf{depth}}\big\}, \qquad \mathcal{I} = \big\{(\delta, i) : 0 \le \delta < \mathsf{depth},\ 0 \le i < 2^{\delta}\big\}.$$

  The node-index bound requires $1 \le \mathsf{depth} \le 64$; each additional level doubles the signature budget and the key-generation cost, and typical values range from $\mathsf{depth} = 8$ (256 signatures) to $\mathsf{depth} = 20$ (2^{20} signatures).

Depth-zero structures (a single WOTS+C key sitting at the root) are disallowed for both shapes, and the verifier rejects signatures with an empty authentication path. **Valid structures.** A structure is *valid* if its anchors are addressable and the set $\mathcal{A} \cup \mathcal{I}$, with the parent and child relation of the imaginary tree, is a *full binary tree* with root $(\delta, i) = (0, 0)$, internal nodes $\mathcal{I}$ and leaves $\mathcal{A}$: the sets $\mathcal{A}$ and $\mathcal{I}$ are disjoint, the root lies in $\mathcal{I}$, every other node of $\mathcal{A} \cup \mathcal{I}$ has its parent in $\mathcal{I}$, and every node of $\mathcal{I}$ has both of its children in $\mathcal{A} \cup \mathcal{I}$. Every node then has either zero or two children, and no anchor lies below another anchor or at the root. The tree need not be complete or perfect: UXMSS is valid with anchors at every depth from 1 to depth, and BXMSS is the perfect case. The security proof of FXMSS (Claim 4.10) uses this property.
Leaf selection $(z^\star, i^\star) \leftarrow \mathsf{leafSelect}_{\mathsf{structure}}(q)$**:** The signer maintains a state counter $q \in \{0, 1, \ldots\}$, the number of stateful signatures already issued, and derives from it the anchor for the next signature:

$$\mathsf{leafSelect}_{(\mathsf{shape},\mathsf{depth})}(q) = \begin{cases} (h_{\mathsf{sf}} - 1 - q,\ 1) & \mathsf{shape} = \texttt{UNBALANCED},\ 0 \le q < \mathsf{depth}, \\ (h_{\mathsf{sf}} - \mathsf{depth},\ 0) & \mathsf{shape} = \texttt{UNBALANCED},\ q = \mathsf{depth} \ge 1, \\ (h_{\mathsf{sf}} - \mathsf{depth},\ q) & \mathsf{shape} = \texttt{BALANCED},\ 0 \le q < 2^{\mathsf{depth}},\ \mathsf{depth} \ge 1, \\ \bot & \text{otherwise.} \end{cases}$$

For the two prescribed shapes, the $\bot$ branch covers depth-zero structures, an exhausted signature budget, and invalid counter values; a future shape defines its own map.[6]
Node computation $\mathsf{treeNode}_{\mathsf{structure}}(z, i, \mathsf{SK.seed}, \mathsf{PK.seed})$**:** The hash value of a node of the structure is computed recursively:

$$\mathsf{treeNode}_{\mathsf{structure}}(z, i, \mathsf{SK.seed}, \mathsf{PK.seed}) = \begin{cases} \mathrm{WOTS{+}C.kg}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}^{\mathsf{w}}_{(z,i)}).\mathsf{pk} & \text{if } (h_{\mathsf{sf}} - z,\ i) \in \mathcal{A}, \\ H\big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{tr}}_{(z,i)},\ N_0 \,\|\, N_1\big) & \text{if } (h_{\mathsf{sf}} - z,\ i) \in \mathcal{I}, \\ \bot & \text{otherwise,} \end{cases}$$

[6] A future shape, with one of the reserved shape values, must define its anchor and internal-node sets and its own leafSelect. Any such map must be a function of the structure and the counter alone that evaluates no hash function, maps $\{0, \ldots, |\mathcal{A}| - 1\}$ bijectively onto the anchor set $\mathcal{A}$, and returns $\bot$ for $q \ge |\mathcal{A}|$, as the map above does for the two prescribed shapes. These are the only properties of leafSelect that the security proof uses, so the proof covers such a shape unchanged.

where $N_b = \mathsf{treeNode}_{\mathsf{structure}}(z{-}1,\, 2i{+}b,\, \mathsf{SK.seed}, \mathsf{PK.seed})$ for $b \in \{0,1\}$ are the two children, $\mathbf{ADRS}^{\mathsf{w}}_{(z,i)}$ is a stateful address with node height z and node index i (a fresh WOTS+C address for the instance anchored at (z,i), whose payload bytes 10–11 carry the two structure bytes $\mathsf{shape} \,\|\, \mathsf{depth}$; only the WOTS+C key derivation keeps them, every other stateful type zeroes them (Table 3), so the verifier, which never derives secret values, sets them to zero), and $\mathbf{ADRS}^{\mathsf{tr}}_{(z,i)}$ is a stateful address with type `SF_FXMSS_TREE`, node height z, and node index i. The FXMSS public key is the root

$$\mathsf{PK.sf} = \mathsf{treeNode}_{\mathsf{structure}}(h_{\mathsf{sf}},\, 0,\, \mathsf{SK.seed}, \mathsf{PK.seed}).$$

Signing Algorithm $\sigma_{\mathsf{tree}} \leftarrow \mathbf{FXMSS}.\mathsf{sign}(md, \mathsf{SK.seed}, \mathsf{PK.seed}, (z^\star, i^\star), \mathsf{structure})$**:**

Given a $2n$-bit message digest md (produced by $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$), the seeds, an anchor position $(z^\star, i^\star) = \mathsf{leafSelect}_{\mathsf{structure}}(q)$, and the structure, the algorithm first checks that $(h_{\mathsf{sf}} - z^\star, i^\star) \in \mathcal{A}$ and returns $\perp$ otherwise. It then proceeds:

1. *WOTS+C signature.* Let $\mathsf{SK}_{(z^\star,i^\star)} = (\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}^{\mathsf{w}}_{(z^\star,i^\star)})$ and compute

   $$\sigma_{\text{WOTS+C}} \leftarrow \text{WOTS+C}.\mathsf{sign}(md,\ \mathsf{SK}_{(z^\star,i^\star)}).$$

   If $\sigma_{\text{WOTS+C}} = \perp$, return $\perp$.

2. *Authentication path.* Set $(z,i) \leftarrow (z^\star, i^\star)$. For $k = 1, 2, \ldots, h_{\mathsf{sf}} - z^\star$:

   (a) Compute the sibling of the current node:

   $$A_k \leftarrow \mathsf{treeNode}_{\mathsf{structure}}(z,\ i \oplus 1,\ \mathsf{SK.seed}, \mathsf{PK.seed})\,.$$

   (b) Advance to the parent: $(z,i) \leftarrow (z+1,\ \lfloor i/2 \rfloor)$.

3. Return $\sigma_{\mathsf{tree}} = (\sigma_{\text{WOTS+C}},\ A_1,\ \ldots,\ A_{h_{\mathsf{sf}}-z^\star})$.

Writing $\delta^\star = h_{\mathsf{sf}} - z^\star$ for the anchor depth, the signature is a string of $c/8 + (l_{\mathsf{sf}} + \delta^\star) \cdot n/8 = 514 + 16\,\delta^\star$ bytes. For both prescribed shapes, every sibling position visited by the loop lies in $\mathcal{A} \cup \mathcal{I}$, so all $\mathsf{treeNode}$ calls are well-defined.

Public Key Recovery $\mathsf{pk} \leftarrow \mathbf{FXMSS}.\mathsf{pkFromSig}(md, (z^\star, i^\star), \sigma_{\mathsf{tree}}, \mathsf{PK.seed})$**:**

Given a $2n$-bit digest md, the anchor position $(z^\star, i^\star)$, a signature σ_{tree}, and the public seed, the algorithm recovers a candidate root. The algorithm proceeds as follows.

1. *Validation.* Set $\delta^\star \leftarrow h_{\mathsf{sf}} - z^\star$, and return $\perp$ unless all of the following hold, where $|\sigma_{\mathsf{tree}}|$ denotes the length in bytes:

   $$1 \;\le\; \delta^\star \;\le\; h_{\mathsf{sf}}, \qquad |\sigma_{\mathsf{tree}}| = c/8 + (l_{\mathsf{sf}} + \delta^\star) \cdot n/8, \qquad i^\star < 2^{\min(64,\, \delta^\star)}.$$

   The condition $\delta^\star \ge 1$ (equivalently $z^\star < h_{\mathsf{sf}}$) ensures that the authentication path is never empty, matching the ban on depth-zero structures.

2. *Anchor recovery.* Parse $\sigma_{\mathsf{tree}} = (\sigma_{\text{WOTS+C}}, A_1, \ldots, A_{\delta^\star})$ and compute

   $$N \leftarrow \text{WOTS+C}.\mathsf{pkFromSig}(md, \sigma_{\text{WOTS+C}}, \mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{w}}_{(z^\star,i^\star)}).$$

   If $N = \perp$ (invalid grinding counter), return $\perp$.

3. *Root recovery.* Set $(z,i) \leftarrow (z^\star, i^\star)$. For $k = 1, 2, \ldots, \delta^\star$:

   (a) Let $(z_p, i_p) \leftarrow (z+1,\ \lfloor i/2 \rfloor)$ be the parent of (z,i).

(b) Update

$$N \leftarrow \begin{cases} H(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{tr}}_{(z_p,i_p)},\ N \,\|\, A_k) & \text{if } i \text{ is even (left child),} \\ H(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{tr}}_{(z_p,i_p)},\ A_k \,\|\, N) & \text{if } i \text{ is odd (right child).} \end{cases}$$

(c) Set $(z, i) \leftarrow (z_p, i_p)$.

4. Return N, the candidate for the root $(h_{\mathsf{sf}}, 0)$; the caller compares it against $\mathsf{PK.sf}$.

Without the exact length constraint, a single signature would admit multiple valid encodings (for instance, by appending trailing bytes that the parser ignores), and the number of authentication-path nodes would not be tied to the claimed anchor height.

Discussion. The verifier consumes only the anchor position $(z^\star, i^\star)$ and is therefore agnostic to the signing-time shape: UXMSS, BXMSS, or a future shape. The reserved shape values leave room for future shapes (footnote to the definition of leafSelect); for security and privacy we recommend the two prescribed shapes.

2.6 FORS

FORS (Forest of Random Subsets) [BHK+19] is a few-time signature scheme used in the stateless path of SHRINCS to sign the ka-bit message digests extracted from the output of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ (Section 2.7). It selects leaves from disjoint subsets, where each subset is used as leaves of a Merkle tree. In SHRINCS, FORS plays the same structural role as in SLH-DSA: its recovered public key is passed as the message to the WOTS-TW layer at the bottom of the hypertree, so FORS signatures are never verified in isolation.

Parameters. The main security parameter is $n \in \mathbb{N}$ (in bits). FORS uses two additional parameters: a tree count $k \in \mathbb{N}$ and a per-tree height $a \in \mathbb{N}$, giving $t = 2^a$ leaves per tree and a total digest length of ka bits. For the SHRINCS parameters ($k = 10$, $a = 13$), a FORS instance signs the leading $ka = 130$ bits of the $\lceil ka/8 \rceil = 17$-byte digest slice. In SHRINCS, a FORS instance is anchored to a position in the SLH-DSA hypertree via the layer and tree fields of **ADRS**, and to a specific leaf of the bottom-layer XMSS tree via the key-pair field.

The building blocks are:

- A pseudorandom function $\mathbf{PRF} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{176} \to \{0,1\}^n$ for deriving the kt private-key values from SK.seed, called with address type `SL_FORS_PRF`.
- A tweakable hash function $\mathbf{F} : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^n \to \{0,1\}^n$ for hashing each private-key value into a leaf node, called with address type `SL_FORS_TREE` at tree height 0.
- A tweakable hash function $H : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^{2n} \to \{0,1\}^n$ for computing internal tree nodes, called with address type `SL_FORS_TREE` at the appropriate tree height.
- A tweakable hash function $\mathsf{Th}_k : \{0,1\}^n \times \{0,1\}^{176} \times \{0,1\}^{kn} \to \{0,1\}^n$ for compressing the k roots into the FORS public key, called with address type `SL_FORS_ROOTS`.

As elsewhere, every hash call receives a distinct address encoding its position, enforced by the scheme in Section 2.2.

FORS scheme. The global tree index of a node at height $z \in \{0, \ldots, a\}$ and horizontal position $\ell \in \{0, \ldots, 2^{a-z} - 1\}$ within tree $i \in \{0, \ldots, k-1\}$ is $i \cdot 2^{a-z} + \ell$, counted continuously across all k trees. Let $\mathbf{ADRS}^{\mathsf{ft}}_{z,\, i \cdot 2^{a-z}+\ell}$ denote **ADRS** with type `SL_FORS_TREE`, tree height z, and tree index $i \cdot 2^{a-z} + \ell$. As in XMSS, the tree-height field encodes the distance from the leaf level ($z = 0$ at leaves, $z = a$ at the root of each tree).

Private-key derivation $\mathsf{sk}_{i,\ell} \leftarrow \mathbf{FORS}.\mathsf{skGen}(\mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS}, i, \ell)$:

The private key is implicit in $(\mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS})$. For tree i and leaf position ℓ, the private-key value is derived on demand:

$$\mathsf{sk}_{i,\ell} = \mathbf{PRF}\Big(\mathsf{PK.seed},\ \mathsf{SK.seed},\ \mathbf{ADRS}^{\mathsf{prf}}_{i,\ell}\Big),$$

where $\mathbf{ADRS}^{\mathsf{prf}}_{i,\ell}$ is $\mathbf{ADRS}$ with type `SL_FORS_PRF`, key-pair address set to the index of the WOTS-TW key that signs this FORS instance, and tree index $i \cdot t + \ell$.

Tree-hash function $\mathsf{fors_root}_i(\mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS})$:

The root of the i-th FORS tree is a complete binary tree of height a over the t private-key values of tree i. A node at height z and position ℓ within tree i is defined recursively:

- For $z = 0$, the leaf is the tweakable hash of the private-key value:

$$N^{(i)}_{0,\ell} = \mathbf{F}\Big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{ft}}_{0,\,i\cdot t+\ell},\ \mathsf{sk}_{i,\ell}\Big).$$

- For $z > 0$, the internal node is the tweakable hash of its two children:

$$N^{(i)}_{z,\ell} = H\Big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{ft}}_{z,\,i\cdot 2^{a-z}+\ell},\ N^{(i)}_{z-1,\,2\ell} \,\|\, N^{(i)}_{z-1,\,2\ell+1}\Big).$$

The root of tree i is $R_i = N^{(i)}_{a,0}$.

Key generation $\mathsf{pk}_{\mathbf{FORS}} \leftarrow \mathbf{FORS}.\mathsf{kg}(\mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS})$:

Compute the k roots $R_0, \ldots, R_{k-1}$ and compress them into the FORS public key:

$$\mathsf{pk}_{\mathrm{FORS}} = \mathsf{Th}_k\big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{roots}},\ R_0 \,\|\, \cdots \,\|\, R_{k-1}\big),$$

where $\mathbf{ADRS}^{\mathsf{roots}}$ is $\mathbf{ADRS}$ with type `SL_FORS_ROOTS`. The private key is implicit.

Signature $\sigma_{\mathbf{FORS}} \leftarrow \mathbf{FORS}.\mathsf{sign}(md, \mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS})$:

Given a ka-bit message digest md, split it into k indices $idx_0, \ldots, idx_{k-1}$ with each $idx_i \in \{0, \ldots, t-1\}$ selecting a leaf in tree i. For each i, the signature component consists of:

- The private-key value $\mathsf{sk}_{i,\,idx_i}$.
- The authentication path $\mathsf{auth}^{(i)} = (A^{(i)}_0, \ldots, A^{(i)}_{a-1})$, where $A^{(i)}_z = N^{(i)}_{z,\,\lfloor idx_i/2^z \rfloor \oplus 1}$ is the sibling of the on-path node at height z.

The full signature is $\sigma_{\mathrm{FORS}} = \big((\mathsf{sk}_{0,idx_0}, \mathsf{auth}^{(0)}), \ldots, (\mathsf{sk}_{k-1,idx_{k-1}}, \mathsf{auth}^{(k-1)})\big)$.

Public key recovery $\mathsf{pk}_{\mathbf{FORS}} \leftarrow \mathbf{FORS}.\mathsf{pkFromSig}(md, \sigma_{\mathbf{FORS}}, \mathsf{PK.seed}, \mathbf{ADRS})$:

As with XMSS, the FORS public key is not verified directly in SHRINCS; it is recomputed from the signature and passed as the message to the WOTS-TW layer of the hypertree.

For each $i \in \{0, \ldots, k-1\}$, recover the root R'_i as follows. First hash the private-key value to obtain the candidate leaf:

$$N' \leftarrow \mathbf{F}\Big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{ft}}_{0,\,i\cdot t+idx_i},\ \mathsf{sk}_{i,\,idx_i}\Big).$$

Then walk up the tree for $z = 0, 1, \ldots, a-1$: let $c = \lfloor idx_i/2^z \rfloor$, set $\mathbf{ADRS}^{\mathsf{tr}}$ to type `SL_FORS_TREE`, tree height $z+1$, tree index $i \cdot 2^{a-z-1} + \lfloor c/2 \rfloor$, and update

$$N' \leftarrow \begin{cases} H\Big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{tr}},\ N' \,\|\, A^{(i)}_z\Big) & \text{if } c \text{ is even (left child)},\\ H\Big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{tr}},\ A^{(i)}_z \,\|\, N'\Big) & \text{if } c \text{ is odd (right child)}. \end{cases}$$

After a steps set $R'_i \leftarrow N'$. Finally, compress the k recovered roots:

$$\mathsf{pk}'_{\mathrm{FORS}} = \mathsf{Th}_k\big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{roots}},\ R'_0 \,\|\, \cdots \,\|\, R'_{k-1}\big).$$

2.7 The Stateless Component: Hypertree and SLH-DSA

The stateless path of SHRINCS is SLH-DSA [Nat24] (the standardized version of the SPHINCS$^+$ construction [BHK+19]) instantiated with the non-standard parameter set of Table 1. It composes the building blocks defined above: a *hypertree* of XMSS trees certifies one of many FORS instances, and the selected FORS instance signs the message digest. All algorithms in this subsection coincide with those of FIPS-205; the SHRINCS-specific behavior (the choice of message and context string) enters only at the top level, in Section 3.

Hypertree. The hypertree consists of d layers of XMSS trees of height h' (Section 2.5). Layer $d-1$ contains a single tree, whose root is the stateless public component $\mathsf{PK.sl}$. Each WOTS-TW leaf of a tree at layer $j > 0$ certifies (i.e., signs) the root of one tree at layer $j-1$, so layer j contains $2^{(d-1-j)h'}$ trees; the WOTS-TW leaves of the bottom layer ($j = 0$) sign FORS public keys. In total, the hypertree provides $2^{h_{\mathsf{sl}}}$ FORS instances, where $h_{\mathsf{sl}} = d \cdot h'$. Since every tree is derived deterministically from $\mathsf{SK.seed}$ via the addressing scheme, the signer materializes only the trees needed for a given signature.

A hypertree signature is a chain of d XMSS signatures, starting from the layer-0 tree.
Hypertree signing $\sigma_{\mathsf{HT}} \leftarrow \mathsf{ht.sign}(\mathsf{m}, \mathsf{SK.seed}, \mathsf{PK.seed}, i_{\mathsf{tree}}, i_{\mathsf{leaf}})$**:** Given an n-bit message m (in SLH-DSA, a FORS public key), the seeds, the index i_{tree} of the layer-0 tree, and the index i_{leaf} of the signing leaf within that tree, the algorithm iterates over the layers. For $j = 0, \ldots, d-1$:

1. Set the layer address of **ADRS** to j and the tree address to i_{tree}.
2. $\sigma^{(j)} \leftarrow \mathrm{XMSS.}\mathsf{sign}(\mathsf{m}, \mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS}, i_{\mathsf{leaf}})$.
3. If $j < d-1$: replace the message by the root of the current tree, $\mathsf{m} \leftarrow \mathrm{XMSS.}\mathsf{pkFromSig}(\mathsf{m}, \sigma^{(j)}, \mathsf{PK.seed}, \mathbf{ADRS}, i_{\mathsf{leaf}})$, and advance to the parent tree: $i_{\mathsf{leaf}} \leftarrow i_{\mathsf{tree}} \bmod 2^{h'}$, $i_{\mathsf{tree}} \leftarrow \left\lfloor i_{\mathsf{tree}}/2^{h'} \right\rfloor$.

The result $\sigma_{\mathsf{HT}} = \sigma^{(0)} \| \cdots \| \sigma^{(d-1)}$ consists of $d \cdot (l_{\mathsf{sl}} + h') \cdot n/8 = 3520$ bytes.
Hypertree verification $b \leftarrow \mathsf{ht.verify}(\mathsf{m}, \sigma_{\mathsf{HT}}, \mathsf{PK.seed}, i_{\mathsf{tree}}, i_{\mathsf{leaf}}, \mathsf{PK.sl})$**:** The verifier recomputes the same chain: for $j = 0, \ldots, d-1$ it sets the layer and tree addresses as above, replaces $\mathsf{m} \leftarrow \mathrm{XMSS.}\mathsf{pkFromSig}(\mathsf{m}, \sigma^{(j)}, \mathsf{PK.seed}, \mathbf{ADRS}, i_{\mathsf{leaf}})$, and updates $(i_{\mathsf{leaf}}, i_{\mathsf{tree}})$ as in signing. It accepts if and only if the final recovered root equals $\mathsf{PK.sl}$.

Message digest parsing. Both signer and verifier derive the FORS digest and the position of the FORS instance from the randomizer R and the message M by parsing the output of the message hash:

$$\mathit{digest} \leftarrow \mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sl}, M) \in \{0,1\}^{2n},$$

$$\begin{aligned}
&\mathit{md} \leftarrow \mathit{digest}[0:17] && \text{(the leading } \lceil ka/8 \rceil \text{ bytes, the FORS message digest),}\\
&i_{\mathsf{tree}} \leftarrow \mathsf{toInt}\big(\mathit{digest}[17:22]\big) \bmod 2^{h_{\mathsf{sl}}-h'} && \text{(the next } \lceil (h_{\mathsf{sl}}-h')/8 \rceil \text{ bytes),}\\
&i_{\mathsf{leaf}} \leftarrow \mathsf{toInt}\big(\mathit{digest}[22:24]\big) \bmod 2^{h'} && \text{(the next } \lceil h'/8 \rceil \text{ bytes),}
\end{aligned}$$

where $\mathsf{toInt}(X)$ denotes the unsigned big-endian integer encoded by the byte string X. The parsing selects, pseudorandomly per pair (R, M), the FORS instance — at leaf i_{leaf} of layer-0 tree i_{tree} — used to sign md. Because the signer cannot be forced to reuse a FORS instance on adversarially chosen messages more often than random chance allows, security degrades gracefully with the number of issued signatures; this is captured by the ITSR notion (Definition 4.11), where the composition of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ with this parsing plays the role of the digest map.

SLH-DSA signing and verification. We first state the internal signing and verification algorithms, then the context wrapper that FIPS-205 places around them.
Internal signing $\sigma \leftarrow \mathsf{slh.sign_internal}(M, \mathsf{SK.seed}, \mathsf{SK.prf}, \mathsf{PK.seed}, \mathsf{PK.sl}, \mathit{opt_rand})$**:**

1. $R \leftarrow \mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}(\mathsf{SK.prf}, \mathit{opt_rand}, M)$, where $\mathit{opt_rand} = \mathsf{PK.seed}$ in the deterministic variant and a fresh random n-bit string in the hedged variant (Section 2.2).

2. Compute $\mathit{digest} \leftarrow \mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sl}, M)$ and split it, as described in the message digest parsing above, into the FORS message digest md and the indices i_{tree} and i_{leaf} of the FORS instance to be used.

3. Initialize **ADRS** with layer address 0, tree address i_{tree}, and key-pair address i_{leaf}; compute $\sigma_{\mathrm{FORS}} \leftarrow \mathrm{FORS.sign}(md, \mathsf{SK.seed}, \mathsf{PK.seed}, \mathbf{ADRS})$ and $\mathsf{pk}_{\mathrm{FORS}} \leftarrow \mathrm{FORS.pkFromSig}(md, \sigma_{\mathrm{FORS}}, \mathsf{PK.seed}, \mathbf{ADRS})$.

4. $\sigma_{\mathsf{HT}} \leftarrow \mathsf{ht.sign}(\mathsf{pk}_{\mathrm{FORS}}, \mathsf{SK.seed}, \mathsf{PK.seed}, i_{\mathsf{tree}}, i_{\mathsf{leaf}})$.

5. Return $\sigma = R \,\|\, \sigma_{\mathrm{FORS}} \,\|\, \sigma_{\mathsf{HT}}$.

The signature length is

$$(n/8) \cdot \big(1 + k(a+1) + d(h' + l_{\mathsf{sl}})\big) = 16 \cdot (1 + 140 + 220) = 5776 \text{ bytes.}$$

Internal verification $b \leftarrow \mathsf{slh.verify_internal}(M, \sigma, \mathsf{PK.seed}, \mathsf{PK.sl})$**:** The verifier rejects unless $|\sigma| = 5776$ bytes. It parses $\sigma = R \| \sigma_{\mathrm{FORS}} \| \sigma_{\mathsf{HT}}$, recomputes $(md, i_{\mathsf{tree}}, i_{\mathsf{leaf}})$ from (R, M) via the digest parsing, recovers $\mathsf{pk}_{\mathrm{FORS}} \leftarrow \mathrm{FORS.pkFromSig}(md, \sigma_{\mathrm{FORS}}, \mathsf{PK.seed}, \mathbf{ADRS})$ with **ADRS** initialized as in signing, and returns $\mathsf{ht.verify}(\mathsf{pk}_{\mathrm{FORS}}, \sigma_{\mathsf{HT}}, \mathsf{PK.seed}, i_{\mathsf{tree}}, i_{\mathsf{leaf}}, \mathsf{PK.sl})$.

Context wrapper. FIPS-205 defines the external algorithms slh.sign and slh.verify as thin wrappers around the internal ones that prepend a domain-separated context string to the message:

$$M' = \mathsf{toByte}(0,1) \,\|\, \mathsf{toByte}(|ctx|, 1) \,\|\, ctx \,\|\, M, \qquad |ctx| \le 255 \text{ bytes.}$$

SHRINCS invokes the wrapper with the caller's context string and the message $\mathsf{PK.sf} \| M$ (Section 3), which binds the stateless signature to the stateful component of the key and to the context. Consequently, the stateless component of SHRINCS remains directly usable with unmodified, NIST-compliant SLH-DSA implementations that expose the new parameter set.

3 The Complete SHRINCS Scheme

We now describe the complete SHRINCS scheme by composing the building blocks defined in Sections 2.4 to 2.7. A SHRINCS key pair combines the stateful FXMSS component and the stateless SLH-DSA component under a single public key; the signer uses the compact stateful path while signing state is available and falls back to the stateless path otherwise, and a signature from either path is sufficient to pass verification. The shape of the stateful tree is fixed per key by the $\mathsf{structure} = (\mathsf{shape}, \mathsf{depth})$ parameter of Section 2.5; the two prescribed shapes are UXMSS (unbalanced) and BXMSS (balanced).
Keys. The public key is the 48-byte string

$$\mathsf{PK} = \mathsf{PK.seed} \,\|\, \mathsf{PK.sl} \,\|\, \mathsf{PK.sf},$$

and the secret key is the 82-byte string

$$\mathsf{SK} = \mathsf{SK.seed} \,\|\, \mathsf{SK.prf} \,\|\, \mathsf{PK.seed} \,\|\, \mathsf{PK.sl} \,\|\, \mathsf{structure} \,\|\, \mathsf{PK.sf}.$$

Both encodings are chosen for SLH-DSA compatibility: dropping the trailing $n/8$ bytes of PK yields a valid SLH-DSA public key, and dropping the trailing $n/8 + 2$ bytes of SK yields a valid SLH-DSA secret key. In addition to SK, the signer maintains a *state counter* ctr, the number of stateful signatures already issued. The distinguished value $\mathsf{ctr} = \perp$ marks the *stateless-only* mode, in which the stateful path is permanently disabled (the reference code represents ctr as a 64-bit unsigned integer and encodes $\perp$ as `None`).

Cross-binding the two paths. Each SHRINCS signature commits to *both* components of the public key, not only to the root it is verified against: the stateless path signs $\mathsf{PK.sf} \| M$ in place of M, and the stateful digest absorbs a message carrying the $\mathsf{PK.sl}$ prefix (the bound message, defined next). This is aimed at ruling out key-substitution attacks in which a signature issued under one SHRINCS key also verifies under a different key sharing one of the two roots (for instance, an adversarially crafted key that reuses the victim's stateless root). Domain separation between all hash calls of the two paths is enforced at the level of the building blocks (Section 2.2).

Contexts. Signing and verification take, besides the message, a caller-chosen *context string* ctx of at most 255 bytes. Contexts separate signatures by use case, protocol, or application: typically ctx is a hard-coded constant of the calling protocol (for instance, a transaction-signing context versus an authentication context), and the default is the empty string $ctx = \varepsilon$. On the stateful path, the string being digested is the *bound message*

$$\widetilde{M} \;=\; \mathsf{toByte}(0,1) \,\|\, \mathsf{toByte}(|ctx|,1) \,\|\, ctx \,\|\, \mathsf{PK.sl} \,\|\, M,$$

mirroring the FIPS-205 context wrapper; on the stateless path, the pair $(\mathsf{PK.sf} \| M,\ ctx)$ is handed to SLH-DSA, whose wrapper produces the same shape with the roles of the two roots exchanged (Section 2.7). The encoding is injective in (ctx, M): the second byte pins the context length, and the root has fixed width. A signature produced with one context therefore does not verify under any other, except with the forgery probability of Section 4.

State and recovery rules. Every WOTS+C key in the FXMSS tree may sign at most once, so any reuse of a counter value breaks security. SHRINCS treats the state as a single source of truth on a single device and enforces the following rules.

- The signer must persist the incremented counter $\mathsf{ctr} + 1$ *before* creating a stateful signature; Algorithm 2 persists it before the message digest is even computed. In particular, a signature must never leave the device before the updated state is durably stored.
- The state counter must never decrement, must not be backed up and restored, must not live in storage that other software can modify, and must not be exported to other devices.
- If the state is lost, corrupted, or otherwise unrecoverable, the signer must set $\mathsf{ctr} \leftarrow \perp$ and use only the stateless path from then on.
- If the secret seed is imported on a second device (e.g., during wallet recovery from a seed phrase, or when provisioning a backup signer), the new device must initialize itself with $\mathsf{ctr} = \perp$: the counter cannot be synchronized between independently signing devices, so any device that is not the unique holder of the state must avoid the stateful path entirely.

The recovery procedure $\mathsf{SHRINCS.Restore}(seed, \mathsf{structure})$ re-derives SK deterministically from the master seed and outputs $\mathsf{ctr} = \perp$, which by the rules above keeps the recovered device in stateless-only mode.

Key generation. Key generation (Algorithm 1) consumes $3n$ random bits (48 bytes) and the structure of the stateful tree. It derives the two roots: the top-layer XMSS root of the hypertree for the stateless component, and the FXMSS root for the stateful component, and assembles the keys.

The $\mathsf{structure}$ argument must come from a trusted source or be validated before use: key-generation cost is exponential in the BXMSS depth, so an adversarially supplied structure can cause key generation to fail or to exhaust computational resources.

Algorithm 1: SHRINCS.KeyGen($seed$, structure)

Input : Seed $seed \in \{0,1\}^{3n}$ sampled from a secure RNG; structure (shape, depth)
Output : Secret key SK, public key PK, initial state ctr

SK.seed $\leftarrow$ $seed[0{:}n/8]$; SK.prf $\leftarrow$ $seed[n/8{:}2n/8]$; PK.seed $\leftarrow$ $seed[2n/8{:}3n/8]$
ADRS $\leftarrow$ zeroed stateless address with layer address $d-1$, tree address 0
PK.sl $\leftarrow$ xmss_root(SK.seed, PK.seed, **ADRS**) // root of the top hypertree layer, Section 2.5
PK.sf $\leftarrow$ $\mathsf{treeNode}_{\mathsf{structure}}(h_{\mathsf{sf}}, 0, \mathsf{SK.seed}, \mathsf{PK.seed})$ // FXMSS root, Section 2.5
SK $\leftarrow$ SK.seed‖SK.prf‖PK.seed‖PK.sl‖structure‖PK.sf; PK $\leftarrow$ PK.seed‖PK.sl‖PK.sf
return (SK, PK, ctr $\leftarrow 0$)

Algorithm 2: SHRINCS.Sign(M, ctx, SK, ctr, opt_rand)

Input : Message M with $|M| \le 2^{61} - 384$ bytes, context ctx with $|ctx| \le 255$ bytes, secret key SK, state counter ctr, optional randomness opt_rand
Output : Signature σ, updated state

Parse SK = SK.seed‖SK.prf‖PK.seed‖PK.sl‖structure‖PK.sf
$pos \leftarrow \perp$; **if** ctr $\neq \perp$ **then** $pos \leftarrow \mathsf{leafSelect}_{\mathsf{structure}}(\mathsf{ctr})$
if $pos = \perp$ **then** // stateless path
  return $\big(\mathsf{toByte}(h_{\mathsf{sf}}, 1) \,\|\, \mathsf{slh.sign}(\mathsf{PK.sf}\|M,\ ctx,\ \mathsf{SK.seed}, \mathsf{SK.prf}, \mathsf{PK.seed}, \mathsf{PK.sl},\ opt_rand),\ \mathsf{ctr}\big)$
else // stateful path
  $(z^\star, i^\star) \leftarrow pos$
  Persist $\mathsf{ctr}' \leftarrow \mathsf{ctr} + 1$ *before any output is produced*
  idx $\leftarrow \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(i^\star, 8)$
  $\widetilde{M} \leftarrow \mathsf{toByte}(0,1) \,\|\, \mathsf{toByte}(|ctx|, 1) \,\|\, ctx \,\|\, \mathsf{PK.sl} \,\|\, M$ // bound message, Section 3
  $R \leftarrow \mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}(\mathsf{SK.prf}, \mathsf{PK.seed}, \mathsf{idx}, \widetilde{M})$
  $md \leftarrow \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}, \widetilde{M})$
  $\sigma_{\mathsf{tree}} \leftarrow \mathrm{FXMSS.sign}(md, \mathsf{SK.seed}, \mathsf{PK.seed}, (z^\star, i^\star), \mathsf{structure})$
  if $\sigma_{\mathsf{tree}} = \perp$ **then return** $(\perp, \mathsf{ctr}')$ // failed counter search; the leaf stays consumed
  return $\big(\mathsf{toByte}(z^\star, 1) \,\|\, R \,\|\, \mathsf{toByte}(i^\star, \ell_{\mathsf{idx}}(h_{\mathsf{sf}} - z^\star)) \,\|\, \sigma_{\mathsf{tree}},\ \mathsf{ctr}'\big)$
end

Signing. The signing algorithm (Algorithm 2) dispatches on the state counter: if $\mathsf{leafSelect}_{\mathsf{structure}}(\mathsf{ctr})$ yields a valid anchor, the stateful path is taken; otherwise the algorithm falls back to the stateless path. Both signing and verification restrict the message length to at most $2^{61} - 384$ bytes: the longest fixed prefix a message sits behind is 107 bytes (on the stateful path: HMAC's key block, PK.seed, idx, the two wrapper bytes, and PK.sl), plus up to 255 context bytes, and the resulting strings must not exceed the SHA-256 input limit of $2^{61} - 1$ bytes; the cap is the largest multiple of the SHA-256 block size within that limit.

On the stateful path, the signer derives the randomizer and the message digest from the 9-byte leaf identifier $\mathsf{idx} = \mathsf{toByte}(z^\star, 1)\|\mathsf{toByte}(i^\star, 8)$ of the selected anchor, binding the digest to the signing position (Section 2.2); the digest additionally absorbs the bound message $\widetilde{M}$ (Section 3), binding the signature to the stateless component and to the caller's context. The resulting signature transmits the anchor *height* $z^\star$ as a leading *indicator byte* (the first byte of idx), the randomizer, the anchor *index* $i^\star$ in $\ell_{\mathsf{idx}}(\delta^\star) = \lceil \min(\delta^\star, 64)/8 \rceil$ bytes, where $\delta^\star = h_{\mathsf{sf}} - z^\star$ is the anchor depth, and the FXMSS signature. Since $i^\star < 2^{\min(64,\delta^\star)}$ (Section 2.5), these $\ell_{\mathsf{idx}}(\delta^\star)$ bytes encode $i^\star$ without loss.

On the stateless path, the signer invokes the external SLH-DSA signing algorithm of Section 2.7 on the message $\mathsf{PK.sf}\|M$ with the caller's context string. The optional input opt_rand selects between the deterministic and hedged variants and is not used by the stateful path.

On the stateful path, signing the same message twice with the same counter does not by itself admit a forgery: the algorithm is deterministic per counter, so the second run reproduces the identical signature. Using the same counter for *distinct* messages, however, is a critical security failure that enables forgeries. This is why the incremented counter is persisted before the signature is created: a crash between the two events then costs at most one unused leaf, never a reused one[7].

The stateful signature has the format

$$\sigma = \mathsf{toByte}(z^\star, 1) \,\|\, R \,\|\, \mathsf{toByte}(i^\star, \ell_{\mathsf{idx}}(\delta^\star)) \,\|\, \sigma_{\mathsf{tree}},$$
$$|\sigma| = \underbrace{1 + n/8 + \ell_{\mathsf{idx}}(\delta^\star)}_{\text{header}} + \underbrace{c/8 + (l_{\mathsf{sf}} + \delta^\star)\, n/8}_{|\sigma_{\mathsf{tree}}|} = 531 + \ell_{\mathsf{idx}}(\delta^\star) + 16\,\delta^\star \text{ bytes},$$

where $\delta^\star = h_{\mathsf{sf}} - z^\star$ is the depth of the signing anchor, and the stateless signature has the format $\sigma = \mathsf{toByte}(h_{\mathsf{sf}}, 1) \,\|\, \sigma_{\mathsf{slh}}$ of $1 + 5776 = 5777$ bytes, with σ_{slh} the SLH-DSA signature of Section 2.7.

Verification. A SHRINCS signature is dispatched to the correct path by its first byte, the indicator byte b. The value $b = h_{\mathsf{sf}}$ selects the stateless path, and every value $b < h_{\mathsf{sf}}$ selects the stateful path with anchor height $z^\star = b$. Stateless signatures have the fixed length 5777 bytes, and stateful signatures occupy the range [548, 4619] bytes ($\delta^\star \in \{1, \ldots, 255\}$, since depth-zero structures are disallowed), the exact length being determined by the indicator byte. Although the dispatch does not rely on it, the reference specification still requires the largest stateful signature to stay below the stateless size, so that the two paths also remain distinguishable by length.

On the stateful branch, the verifier reads the anchor height from the indicator byte and validates the signature format (an exact length check, and the checks of step 1 of FXMSS.pkFromSig (Section 2.5)). It then recomputes the message digest from the bound message, recovers the candidate FXMSS root, and compares it against PK.sf. On the stateless branch, it invokes the SLH-DSA verifier of Section 2.7 on PK.sf$\|M$ with the caller's context, which compares the recovered hypertree root against PK.sl.

Verification cost. Stateful verification performs exactly $l_{\mathsf{sf}}(w-1) - S_{wn} = 240$ chain steps (calls to F): the target-sum encoding fixes the total $\sum_i (w - 1 - a_i)$ to a constant of the scheme, independent of the message. Together with one $\mathsf{Th}_{l_{\mathsf{sf}}}$ compression, $\delta^\star$ tree hashes, and one $\mathbf{H}_{\text{grind}}$ call, this gives $242 + \delta^\star$ tweakable-hash calls plus one $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ evaluation: $242 + \min(q, 255)$ for the q-th UXMSS signature ($q \geq 1$, depth 255) and 262 at the BXMSS reference depth 20, or roughly $250 + \delta^\star$ online SHA-256 compressions (Section 2.2). Stateless verification costs $k(a+1) + 1 + d\big(\sum_i (w - 1 - b_i) + h' + 1\big)$ tweakable-hash calls plus one $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ evaluation; the per-layer chain steps are input-dependent with mean ≈ 267.1 for a uniformly random digest (slightly above $l_{\mathsf{sl}}(w-1)/2 = 262.5$, since the checksum digits are not uniform), so the online cost is at most roughly 2 900 compressions and about half that on average (cf. Table 4).

3.1 Correctness

Both signing paths are deterministic recomputations once the randomizer R is fixed, and the verifier repeats them from the transmitted data: it recovers the same digest, completes the same chains, and climbs the same tree. The single step that can fail on the signer's side is the counter search of WOTS+C: if none of the 2^c counter values maps the digest to a constant-sum digit vector, grind returns $\perp$, and Algorithm 2 returns $(\perp, \mathsf{ctr}')$. Correctness therefore splits into two claims. The first is unconditional: every signature the signer outputs verifies. The second is probabilistic: the signer outputs $\perp$ only with a probability that is negligible for the SHRINCS parameters. Since signing takes a state and returns an updated one, the statement ranges over whole sequences of signatures produced from an honestly maintained state.

[7] The grinding step inside WOTS+C can in principle exhaust its counter space and return $\perp$ (Section 2.4), in which case Algorithm 2 also returns $\perp$ together with the incremented counter ctr'. Since the state was already incremented, the affected leaf is treated as consumed.

Algorithm 3: SHRINCS.Verify(M, ctx, σ, PK)

Input : Message M with $|M| \le 2^{61} - 384$ bytes, context ctx with $|ctx| \le 255$ bytes, signature σ, public key PK
Output : 1 if σ is valid, 0 otherwise

if $|\mathsf{PK}| \neq 3n/8$ *bytes* **then return** 0
Parse $\mathsf{PK} = \mathsf{PK.seed} \,\|\, \mathsf{PK.sl} \,\|\, \mathsf{PK.sf}$
if $|\sigma| = 0$ **then return** 0
Parse $\sigma = \mathsf{toByte}(b, 1) \,\|\, \sigma'$ // indicator byte
if $b = h_{\mathsf{sf}}$ **then** // stateless path
  return slh.verify($\mathsf{PK.sf} \| M$, σ', ctx, PK.seed, PK.sl)
else // stateful path
  $z^\star \leftarrow b$; $\delta^\star \leftarrow h_{\mathsf{sf}} - z^\star$
  if $|\sigma| \neq 1 + n/8 + \ell_{\mathsf{idx}}(\delta^\star) + c/8 + (l_{\mathsf{sf}} + \delta^\star)\, n/8$ **then return** 0
  Parse $\sigma' = R \,\|\, \mathsf{toByte}(i^\star, \ell_{\mathsf{idx}}(\delta^\star)) \,\|\, \sigma_{\mathsf{tree}}$
  if $i^\star \ge 2^{\min(64,\, \delta^\star)}$ **then return** 0
  $\mathsf{idx} \leftarrow \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(i^\star, 8)$
  $\widetilde{M} \leftarrow \mathsf{toByte}(0, 1) \,\|\, \mathsf{toByte}(|ctx|, 1) \,\|\, ctx \,\|\, \mathsf{PK.sl} \,\|\, M$ // bound message, Section 3
  $md \leftarrow \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}, \widetilde{M})$
  $root \leftarrow \mathrm{FXMSS.pkFromSig}(md, (z^\star, i^\star), \sigma_{\mathsf{tree}}, \mathsf{PK.seed})$
  return $root \neq \perp \;\wedge\; root = \mathsf{PK.sf}$
end

Definition 3.1 (Honest execution; correctness)**.** Let $seed \in \{0,1\}^{3n}$, let structure be a structure of one of the two prescribed shapes (Section 2.5), and let $(\mathsf{SK}, \mathsf{PK}, \mathsf{ctr}_0)$ be the output of SHRINCS.KeyGen($seed$, structure). An *honest execution* of SHRINCS on a sequence of inputs (M_i, ctx_i, opt_rand_i), $i = 1, \ldots, q$, with $|M_i| \le 2^{61} - 384$ bytes and $|ctx_i| \le 255$ bytes, is the sequence of calls

$$(\sigma_i, \mathsf{ctr}_i) \leftarrow \mathsf{SHRINCS.Sign}(M_i,\, ctx_i,\, \mathsf{SK},\, st_{i-1},\, opt_rand_i), \qquad i = 1, \ldots, q,$$

where the state $st_{i-1} \in \{\mathsf{ctr}_{i-1}, \perp\}$ passed to the i-th call is the state returned by the previous call, except that the signer may replace it by $\perp$ at any point, as the recovery rules of Section 3 require after a loss of state or on a recovered device. SHRINCS is *correct* if, for every honest execution and every i with $\sigma_i \neq \perp$,

$$\mathsf{SHRINCS.Verify}(M_i,\, ctx_i,\, \sigma_i,\, \mathsf{PK}) = 1.$$

The execution *fails* if $\sigma_i = \perp$ for some i.

We use the idealization under which the expected cost of the counter search is estimated (Remark 4.4).

Assumption 3.1 (Grinding heuristic)**.** We model the values of $\mathbf{H}_{\mathrm{grind}}$ at pairwise distinct inputs as independent random variables, each uniformly distributed over $\{0,1\}^n$. In particular, at a fixed public seed, tweak, and digest, the 2^c encoded digests examined by grind are independent and uniform, and each satisfies the target-sum condition with probability

$$p_{\mathsf{sum}} \;=\; \frac{\left|\{\, x \in \{0,1\}^n : \sum_{i=0}^{l_{\mathsf{sf}}-1} a_i(x) = S_{wn} \,\}\right|}{2^n},$$

where $(a_0(x), \ldots, a_{l_{\mathsf{sf}}-1}(x))$ is the base-w digit vector of x.

The heuristic enters only the failure bound below and the expected cost of the counter search (Remark 4.4). The security analysis of Section 4 treats $\mathbf{H}_{\mathrm{grind}}$ through the S-TCR(Prop) notion and does not use the heuristic; it rests on Conjecture 3.1 below.

Lemma 3.1 (Correctness of SHRINCS)**.** *Let* structure *be a structure of one of the two prescribed shapes with $L = |\mathcal{A}|$ anchors ($L = \mathsf{depth} + 1$ for UXMSS, $L = 2^{\mathsf{depth}}$ for BXMSS), and consider an honest execution of SHRINCS in the sense of Definition 3.1.*

(i) (Verification of honest signatures.) *SHRINCS is correct: for every i with $\sigma_i \neq \perp$,*

$$\mathit{SHRINCS}.\mathsf{Verify}(M_i,\ ctx_i,\ \sigma_i,\ \mathsf{PK}) = 1.$$

Moreover, $\sigma_i = \perp$ occurs only on the stateful path, and only when the grinding step of WOTS+C returns $\perp$.

(ii) (Signing failure.) *Under Assumption 3.1, every stateful signing call of the execution returns $\perp$ with probability $(1 - p_{\mathsf{sum}})^{2^c}$, for every message, context, and key, and the execution fails with probability at most $L \cdot (1 - p_{\mathsf{sum}})^{2^c}$.*

For the SHRINCS parameters ($n = 128$, $w = 16$, $l_{\mathsf{sf}} = 32$, $S_{wn} = 240$, $c = 16$),

$$N = \sum_{j=0}^{15} (-1)^j \binom{32}{j} \binom{271 - 16j}{31} \approx 2^{121.96}$$

is the number of n-bit strings whose digits sum to S_{wn}, so $p_{\mathsf{sum}} = N/2^{128} \approx 2^{-6.04}$, i.e., one attempt in about 65.7 succeeds and the counter search performs $1/p_{\mathsf{sum}} \approx 66$ attempts in expectation. A stateful signing call fails with probability $(1 - p_{\mathsf{sum}})^{2^{16}} < 2^{-1450}$, and an execution with the largest admissible budget, $L = 2^{64}$, fails with probability below 2^{-1386}.

Proof. Fix a call of the execution, with input (M, ctx, opt_rand), state st, and output (σ, ctr'), and write $(\mathsf{shape}, \mathsf{depth}) = \mathsf{structure}$. Algorithm 2 takes the stateless branch if $st = \perp$ or if leaf selection on st returns $\perp$, and the stateful branch otherwise. On the stateless branch, slh.sign has no failing step: its wrapper requires $|ctx| \leq 255$, and the message-length cap keeps every SHA-256 input within its domain. On the stateful branch, FXMSS.sign fails only through grind: the selected position is an anchor (Step 1), and every sibling visited by the authentication-path loop lies in $\mathcal{A} \cup \mathcal{I}$ (Section 2.5), so all treeNode calls are well-defined. This proves the second sentence of claim (i); for the first, assume $\sigma \neq \perp$.

Step 1: leaf selection lands on an anchor. Let $st = \mathsf{ctr} \neq \perp$, $(z^\star, i^\star) = \mathsf{leafSelect}_{\mathsf{structure}}(\mathsf{ctr}) \neq \perp$, and $\delta^\star = h_{\mathsf{sf}} - z^\star$. For UXMSS, counters $0 \leq \mathsf{ctr} < \mathsf{depth}$ select $(\delta^\star, i^\star) = (\mathsf{ctr} + 1, 1)$ and $\mathsf{ctr} = \mathsf{depth}$ selects $(\mathsf{depth}, 0)$; for BXMSS, counters $0 \leq \mathsf{ctr} < 2^{\mathsf{depth}}$ select $(\mathsf{depth}, \mathsf{ctr})$. In all cases $(\delta^\star, i^\star) \in \mathcal{A}$ with $1 \leq \delta^\star \leq \mathsf{depth}$, and $i^\star \in \{0, 1\}$ for UXMSS, whereas $i^\star < 2^{\delta^\star}$ for BXMSS. Moreover, every proper ancestor of an anchor is an internal node: for UXMSS the ancestors of $(\delta, 1)$ and of $(\mathsf{depth}, 0)$ are the spine nodes $(\delta', 0)$ with $\delta' < \mathsf{depth}$; for BXMSS the ancestors of (depth, i) are the nodes $(\delta', \lfloor i/2^{\mathsf{depth}-\delta'} \rfloor)$ with $\delta' < \mathsf{depth}$, whose index is below $2^{\delta'}$.

Step 2: dispatch and format checks. A stateless signature is non-empty, begins with the indicator byte h_{sf}, and is routed to the stateless branch of Algorithm 3, which passes the remaining 5776 bytes, the output of slh.sign, to slh.verify. A stateful signature begins with the indicator byte $z^\star = h_{\mathsf{sf}} - \delta^\star$, where $1 \leq \delta^\star \leq 255$ (Step 1 and the depth bounds of the two shapes), so $z^\star < h_{\mathsf{sf}}$ and it is routed to the stateful branch, where the verifier recovers $z^\star$ and $\delta^\star$. The length check passes, since the signer emits exactly $1 + n/8 + \ell_{\mathsf{idx}}(\delta^\star)$ header bytes followed by $|\sigma_{\mathsf{tree}}| = c/8 + (l_{\mathsf{sf}} + \delta^\star)\, n/8$ bytes (Section 2.5). The index check $i^\star < 2^{\min(64, \delta^\star)}$ passes: $i^\star \leq 1$ and $\delta^\star \geq 1$ for UXMSS; $i^\star < 2^{\delta^\star}$ and $\delta^\star = \mathsf{depth} \leq 64$ for BXMSS. Since $2^{\min(64,\delta^\star)} \leq 2^{8\ell_{\mathsf{idx}}(\delta^\star)}$, the encoding $\mathsf{toByte}(i^\star, \ell_{\mathsf{idx}}(\delta^\star))$ is well-defined and the verifier parses back the signer's $i^\star$, and with it the signer's idx. The public-key length check passes as $|\mathsf{PK}| = 3n/8$.

Step 3: chain arithmetic and the one-time round trips. Fix a chain index i, the public seed, and an address **ADRS** of hash type, and abbreviate $c^{j,k}(x) = c^{j,k}(x, i, \mathsf{PK.seed}, \mathbf{ADRS})$. Unrolling the recursive definition of the chaining function (Section 2.3) gives, for $j + k \leq w - 1$,

$$c^{j,k}(x) = F_{j+k-1} \circ \cdots \circ F_j\,(x), \qquad F_t(y) := F(\mathsf{PK.seed}, \mathbf{ADRS}_t, y),$$

where $\mathbf{ADRS}_t$ is **ADRS** with the hash address set to t. Consequently, for every digit $0 \le a \le w-1$,

$$c^{a,\,w-1-a}\big(c^{0,a}(x)\big) = F_{w-2} \circ \cdots \circ F_0\,(x) = c^{0,\,w-1}(x).$$

For WOTS+C, write $\mathbf{ADRS}^{\mathsf{w}} = \mathbf{ADRS}^{\mathsf{w}}_{(z^\star,i^\star)}$, and let $\sigma_{\text{WOTS+C}} = (\mathsf{ctr}_{\mathsf{g}}, \sigma_0, \ldots, \sigma_{l_{\mathsf{sf}}-1})$ be the WOTS+C signature of md under $\mathsf{SK}_{(z^\star,i^\star)}$, which is not $\perp$. The signer's grind returned a counter $\mathsf{ctr}_{\mathsf{g}}$ whose encoded digest d has digit sum S_{wn}; WOTS+C.$\mathsf{pkFromSig}$ recomputes d from the same inputs, so its target-sum check passes (it accepts any valid counter) and it obtains the same digits $(a_0, \ldots, a_{l_{\mathsf{sf}}-1})$. Since the signer re-derives $\mathsf{sk}_i = \mathbf{PRF}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}_i)$ exactly as key generation does, $\sigma_i = c^{0,a_i}(\mathsf{sk}_i)$ and hence $\mathsf{pk}'_i = c^{a_i,\,w-1-a_i}(\sigma_i) = c^{0,\,w-1}(\mathsf{sk}_i) = \mathsf{pk}_i$ for every chain (with $\mathbf{ADRS} = \mathbf{ADRS}^{\mathsf{h}}$). Both sides derive $\mathbf{ADRS}^{\mathsf{h}}$ and $\mathbf{ADRS}^{\mathsf{pk}}$ from $\mathbf{ADRS}^{\mathsf{w}}$, which the verifier rebuilds from the recovered $(z^\star, i^\star)$; the verifier's copy has zero structure bytes, but the two derived addresses coincide on both sides, since the type change to `SF_WOTS_C_HASH` or `SF_WOTS_C_PK` zeroes the payload (Table 3) and $\mathbf{H}_{\text{grind}}$ absorbs only the first 10 bytes of $\mathbf{ADRS}^{\mathsf{g}}$, so

$$\begin{aligned}&\text{WOTS+C.}\mathsf{pkFromSig}\big(md,\ \sigma_{\text{WOTS+C}},\ \mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{w}}\big)\\ &\quad = \text{WOTS+C.}\mathsf{kg}\big(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}^{\mathsf{w}}\big).\mathsf{pk}.\end{aligned}$$

The same argument, with the checksum encoding $(b_0, \ldots, b_{l_{\mathsf{sl}}-1})$ of m in place of the grinded digits, gives

$$\begin{aligned}&\text{WOTS-TW.}\mathsf{pkFromSig}\big(\mathsf{m},\ \text{WOTS-TW.}\mathsf{sign}(\mathsf{m}, \mathsf{SK}_q),\ \mathsf{PK.seed},\ \mathbf{ADRS}_q\big)\\ &\quad = \text{WOTS-TW.}\mathsf{kg}\big(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}_q\big).\mathsf{pk}.\end{aligned}$$

Step 4: Merkle round trips. Let $\mathsf{node}(z, \ell)$ be a family of values indexed by height and horizontal index such that, for every parent position $(z+1, \ell)$ visited below,

$$\mathsf{node}(z+1, \ell) = H\big(\mathsf{PK.seed},\ \mathbf{ADRS}^{\mathsf{tr}}_{z+1,\ell},\ \mathsf{node}(z, 2\ell) \,\|\, \mathsf{node}(z, 2\ell+1)\big).$$

Suppose a verifier starts from $N = \mathsf{node}(z_0, \ell_0)$ and, for $k = 1, 2, \ldots$, combines its current value at position $(z, \ell) = (z_0 + k - 1,\ \lfloor \ell_0 / 2^{k-1} \rfloor)$ with the sibling $A_k = \mathsf{node}(z, \ell \oplus 1)$ as $N \leftarrow H(\mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{tr}}_{z+1,\lfloor \ell/2 \rfloor}, N \| A_k)$ if ℓ is even and as $N \leftarrow H(\mathsf{PK.seed}, \mathbf{ADRS}^{\mathsf{tr}}_{z+1,\lfloor \ell/2 \rfloor}, A_k \| N)$ if ℓ is odd. Then after k steps $N = \mathsf{node}(z_0 + k, \lfloor \ell_0 / 2^k \rfloor)$, by induction on k: for even ℓ the pair $\big((z, \ell), (z, \ell+1)\big)$ is the (left, right) child pair of $(z+1, \ell/2)$, for odd ℓ the pair $\big((z, \ell-1), (z, \ell)\big)$ is that of $(z+1, \lfloor \ell/2 \rfloor)$, and in both cases the combination is the defining hash of the parent. This is the walk of FXMSS.$\mathsf{pkFromSig}$ with $\mathsf{node} = \mathsf{treeNode}_{\mathsf{structure}}$ and the signer's siblings $A_k = \mathsf{treeNode}_{\mathsf{structure}}(z, i \oplus 1)$, where the recursion holds at every visited parent because every proper ancestor of an anchor is internal (Step 1); of XMSS.$\mathsf{pkFromSig}$ with $\mathsf{node} = N_{z,j}$ of Section 2.5 and $A_z = N_{z,\lfloor q/2^z \rfloor \oplus 1}$; and, for each i, of FORS.$\mathsf{pkFromSig}$ with $\mathsf{node} = N^{(i)}_{z,\ell}$ of Section 2.6 and $A^{(i)}_z = N^{(i)}_{z,\lfloor idx_i/2^z \rfloor \oplus 1}$, the address carrying the global index $i \cdot 2^{a-z-1} + \lfloor c/2 \rfloor$ of the parent position.

Step 5: the stateful branch accepts. The verifier forms idx from the recovered $z^\star$ and the transmitted $i^\star$, and $\widetilde{M}$ from ctx, $\mathsf{PK.sl}$, and M, so with the transmitted R its digest $md = \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}, \widetilde{M})$ equals the signer's. By Step 3 the recovered anchor value is WOTS+C.$\mathsf{kg}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS}^{\mathsf{w}}).\mathsf{pk} = \mathsf{treeNode}_{\mathsf{structure}}(z^\star, i^\star)$, as $(\delta^\star, i^\star) \in \mathcal{A}$. Signer and verifier traverse the same positions from $(z^\star, i^\star)$ under the parent map $(z, i) \mapsto (z+1, \lfloor i/2 \rfloor)$, so by Step 4 the verifier holds $\mathsf{treeNode}_{\mathsf{structure}}(h_{\mathsf{sf}}, 0) = \mathsf{PK.sf}$ after $\delta^\star$ steps, since $\lfloor i^\star / 2^{\delta^\star} \rfloor = 0$ by $i^\star < 2^{\delta^\star}$ (Step 1). Hence Algorithm 3 returns 1.

Step 6: the stateless branch accepts. The wrapper forms the same string M' from ctx and $\mathsf{PK.sf} \| M$ on both sides (Section 2.7), and the verifier recomputes $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sl}, M')$ from the transmitted R and parses it as the signer did, obtaining the same $(md, i_{\mathsf{tree}}, i_{\mathsf{leaf}})$ and initial

ADRS. The signer obtains $\mathsf{pk}_{\mathrm{FORS}}$ by calling FORS.pkFromSig on $(md, \sigma_{\mathrm{FORS}}, \mathsf{PK.seed}, \mathbf{ADRS})$, and, starting from $\mathsf{m}_0 = \mathsf{pk}_{\mathrm{FORS}}$, defines the message of each hypertree layer $j+1$ by applying XMSS.pkFromSig to $(\mathsf{m}_j, \sigma^{(j)}, \mathsf{PK.seed}, \mathbf{ADRS}_j, i_{\mathsf{leaf},j})$; the verifier applies the same calls to the same inputs with the same index updates, so by induction on j it recovers $\mathsf{pk}_{\mathrm{FORS}}$ and $\mathsf{m}_1, \ldots, \mathsf{m}_d$. For the last layer, $\sigma^{(d-1)}$ is the XMSS signature of m_{d-1} under leaf $i_{\mathsf{leaf},d-1}$ of the tree at layer $d-1$ with tree address $i_{\mathsf{tree},d-1}$, so by Steps 3 and 4 m_d is the root $N_{h',0}$ of that tree. Since $i_{\mathsf{tree},0} < 2^{h_{\mathsf{sl}}-h'} = 2^{(d-1)h'}$ and each layer divides the tree index by $2^{h'}$, $i_{\mathsf{tree},d-1} = 0$, and m_d is the root of the layer-$(d-1)$ tree 0, which Algorithm 1 stored as $\mathsf{PK.sl}$. Hence $\mathsf{slh.verify}$ returns 1, and so does Algorithm 3. Together with Step 5 this proves claim (i).

Claim (ii). A stateful call at anchor $(z^\star, i^\star)$ with digest md evaluates $\mathbf{H}_{\mathrm{grind}}$ at the inputs $(\mathsf{PK.seed}, T, md, \mathsf{ctr_g})$, $\mathsf{ctr_g} = 0, 1, \ldots, 2^c - 1$, where $T = \mathbf{ADRS}^{\mathsf{g}}_{(z^\star,i^\star)}[0{:}10]$, stopping at the first success. These 2^c inputs are pairwise distinct, so under Assumption 3.1 the encoded digests are independent and uniform, whatever the values of $\mathsf{PK.seed}$, T, and md, and the call returns $\perp$ exactly when all 2^c attempts fail, with probability $(1-p_{\mathsf{sum}})^{2^c}$. The execution contains at most L stateful calls: Algorithm 2 increments the counter on every stateful call, including a failing one, the counter never decreases (Section 3), and $\mathsf{leafSelect}_{\mathsf{structure}}$ returns $\perp$ once the counter reaches L, routing all further calls to the stateless path. A union bound gives $L \cdot (1-p_{\mathsf{sum}})^{2^c}$. For the concrete numbers, $p_{\mathsf{sum}} = N/2^{128} \approx 0.015226$ and $2^{16}\log_2(1-p_{\mathsf{sum}}) \approx -1450.7$, so $(1-p_{\mathsf{sum}})^{2^{16}} < 2^{-1450}$, and $L \le 2^{64}$ gives 2^{-1386}; a successful search stops after a geometrically distributed number of attempts with mean $1/p_{\mathsf{sum}} \approx 65.7$, in line with Remark 4.4. □

The failure bound is specific to the parameter set: it is the value p_{sum} of the target-sum predicate, together with the counter length c, that makes exhaustion of the counter space a negligible event, and other choices of (n, w, S_{wn}, c) can give substantially weaker bounds. The statement of Lemma 3.1 covers the two prescribed shapes; its proof carries over to a future shape (Section 2.5), with the $\mathsf{leafSelect}$ of that shape in place of the two formulas used in Step 1 and the depth and index bounds of its anchors in place of those of the two shapes in Step 2: every valid structure has the two properties the proof uses (every proper ancestor of an anchor is an internal node, and every sibling along an anchor's path to the root lies in $\mathcal{A} \cup \mathcal{I}$), and the $\mathsf{leafSelect}$ of a future shape has the properties required in the footnote to its definition.

The security analysis needs more than a small failure probability: its reductions and Remark 4.6 use that no stateful signing call fails at all. We therefore state the corresponding property of $\mathbf{H}_{\mathrm{grind}}$ as a conjecture, and bound its failure in the ideal model.

Conjecture 3.1 (Grind-completeness of $\mathbf{H}_{\mathrm{grind}}$)**.** *For every public seed* $\mathsf{PK.seed} \in \{0,1\}^n$*, every grinding address* $T \in \mathcal{T}_{\mathsf{g}}$ *(Section 2.2), and every digest* $md \in \{0,1\}^{2n}$*, some counter* $\mathsf{ctr} \in \{0, \ldots, 2^c - 1\}$ *gives an encoded digest* $\mathbf{H}_{\mathrm{grind}}(\mathsf{PK.seed}, T[0{:}10], md, \mathsf{ctr})$ *whose base-w digits sum to* S_{wn}*.*

Lemma 3.2 (Grind-completeness in the ideal model)**.** *Model* $\mathbf{H}_{\mathrm{grind}}$ *as in Assumption 4.1, as part of a uniformly random collection. Then Conjecture 3.1 fails with probability at most*

$$\varepsilon_{\mathsf{gc}} \;:=\; 2^n \cdot 2^{72} \cdot 2^{2n} \cdot (1-p_{\mathsf{sum}})^{2^c},$$

and for the SHRINCS parameters $\varepsilon_{\mathsf{gc}} < 2^{-994}$*.*

Proof. Fix $(\mathsf{PK.seed}, T, md)$. The 2^c inputs of the counter search differ in the counter, so in the ideal model their values are independent and uniform, and each meets the target sum with probability p_{sum}; all of them fail with probability $(1-p_{\mathsf{sum}})^{2^c}$. There are 2^n seeds, at most 2^{72} grinding addresses, since a grinding address is determined by its 1-byte node height and 8-byte node index (its type byte is fixed and its payload is zero), and 2^{2n} digests, and a union bound gives $\varepsilon_{\mathsf{gc}}$. For the concrete value, $2^{16}\log_2(1-p_{\mathsf{sum}}) \approx -1450.70$, so $\log_2 \varepsilon_{\mathsf{gc}} \approx 456 - 1450.70 = -994.70$. □

4 Security Analysis

In this section we prove the EU-CMA security of SHRINCS by reduction to the security properties of the underlying tweakable hash functions and pseudorandom functions. The argument follows the proof paradigm of [BHK+19] and the corrected proof of [HK22], and reuses the SPHINCS+C results of [HKRY23] for the analysis of the WOTS+C block. The proofs presented here closely follow the security proofs of the original schemes; the security of SPHINCS$^+$ and plain XMSS has moreover been formally verified [BDH+24, BDG+23]. One adaptation of the EU-CMA model is needed: the adversary chooses an arbitrary valid FXMSS structure before the public key is generated. Giving this choice to the adversary establishes the bound for every admissible structure at once, so the signer's freedom to choose the shape of the stateful tree (Section 2.5) cannot weaken the scheme; see the discussion preceding Definition 4.15.

4.1 Hash Functions

In SHRINCS we use keyed hash functions and tweakable hash functions. Both are implemented using SHA-256 [Nat15]. The main difference between them is conceptual.

A keyed hash function is a cryptographic hash function that takes both a message and a key as input to produce a hash value.

Definition 4.1 (Keyed hash functions)**.** Let $\mathcal{M}$ be the message space, $\mathcal{K}$ be the key space, and $\mathcal{N}$ the output space. A keyed hash function is an efficient function

$$F : \mathcal{K} \times \mathcal{M} \to \mathcal{N},\ y \leftarrow F(k, \mathsf{m})$$

generating a digest y out of a key k and a message m.

We will also write $F_k(\mathsf{m})$ to denote $F(k, \mathsf{m})$.

A *tweakable hash function* takes three arguments: public parameters P and context information in the form of a tweak T in addition to the message input m.

Definition 4.2 (Tweakable hash function [BHK+19])**.** Let $\mathcal{P}$ be the public parameter space, $\mathcal{T}$ the tweak space, $\mathcal{M}$ the message space, and $\mathcal{N}$ the output space. A *tweakable* hash function is an efficient function

$$\mathbf{Th} : \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{N},\ y \leftarrow \mathbf{Th}(P, T, \mathsf{m})$$

mapping a message m to a hash value y using a function key called public parameter $P \in \mathcal{P}$ and a tweak $T \in \mathcal{T}$.

We sometimes write $\mathsf{Th}_{P,T}(M)$ for $\mathsf{Th}(P, T, M)$. In SHRINCS, the public seed PK.seed serves as P and a 22-byte address structure **ADRS** encodes the tweak, uniquely identifying each hash call within the scheme.

4.2 Security notions of hash functions

In this subsection we present the properties that are used in the security proof of SHRINCS. We present the notions in order of increasing complexity; each definition states whether it concerns a keyed or a tweakable hash function.

Keyed hash functions are often used as pseudorandom functions (PRFs), meaning that without knowledge of the key (which remains secret), an adversary cannot distinguish their output from that of a truly random function. In SHRINCS, this property is required from the function **PRF**, which derives the secret key elements, and from $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$, which derive the message randomizers of the stateless and the stateful path, respectively.

Definition 4.3 (Pseudorandom Function (PRF))**.** Let $F : \mathcal{K} \times \mathcal{M} \to \mathcal{N}$ be a KHF. Let $\mathcal{G}(\mathcal{M}, \mathcal{N})$ be the set of all functions from $\mathcal{M}$ to $\mathcal{N}$. Let $\mathcal{A}$ be an algorithm, and let $p \in \mathbb{N}$ be a bound on its oracle queries. Consider the following experiment $\mathbf{PRF}_{F,p}(\mathcal{A})$:

1. Generate a random key $k \leftarrow_{\$} \mathcal{K}$.
2. Generate a random function $G \leftarrow_{\$} \mathcal{G}(\mathcal{M}, \mathcal{N})$.
3. Generate a random bit $b \leftarrow_{\$} \{0,1\}$.
4. Run $\mathcal{A}$ with (classical) access to an oracle $\mathcal{O}(\cdot, b)$ that takes $\mathsf{m} \in \mathcal{M}$ as input and works as follows:
   - If $\mathcal{A}$ has already made p queries, return $\perp$.
   - If $b = 0$, return $y = F(k, \mathsf{m})$.
   - If $b = 1$, return $y = G(\mathsf{m})$.
5. Obtain an output from $\mathcal{A}$: $b' \leftarrow \mathcal{A}^{\mathcal{O}(\cdot,b)}()$, where $b' \in \{0,1\}$.
6. Output b'.

For any such algorithm $\mathcal{A}$, we define the advantage against pseudorandomness of F as:

$$\mathsf{Adv}^{\mathrm{PRF}}_{F,p}(\mathcal{A}) = |\Pr[\mathbf{PRF}_{F,p}(\mathcal{A}) \Rightarrow 1 \mid b = 1] - \Pr[\mathbf{PRF}_{F,p}(\mathcal{A}) \Rightarrow 1 \mid b = 0]|.$$

The oracle access in the PRF experiment is classical, and p caps the number of oracle queries; this matches the use of the experiment in this document, where the reductions query the oracle classically, once per derived secret value or randomizer. The generic PRF bounds of Table 6 do not depend on p. In the ideal model, where F is a random oracle taking the hidden key as part of its input, the real and the ideal oracle are both random functions, so the oracle answers are distributed identically in the two cases until one of the adversary's own evaluations of F uses the hidden key. The advantage is therefore driven by the adversary's evaluations of the public function $F(\cdot,\cdot)$ at keys of its own choice (a key search, classical or quantum with the adversary), which are counted by the time ξ of Section 4.3, and not by the challenge queries counted by p.

In the following interactive notions, we typically divide the algorithm into two distinct stages that share a state. The first stage involves setting up challenges, which requires the algorithm to make classical queries to the challenger, as the challenger possesses certain secret information. Once the challenges have been set, the algorithm proceeds to the second stage, where it gains access to the previously hidden information. However, during the second stage, the algorithm does not have the ability to interact with the challenge oracle anymore.

The first interactive notion is single-function, multi-target target collision resistance, the property we rely on most. In SHRINCS, SM-TCR is required from the functions computing the WOTS-TW and WOTS+C chains, the compression of WOTS-TW and WOTS+C public keys, the Merkle-tree nodes, the leaves of the FORS trees, and the compression of FORS public keys. Essentially, this property guarantees that a forgery cannot deviate from the honest structure of the SHRINCS tree: we show that the adversary must output exactly the values we would have computed for the forged message, as any deviation yields a collision breaking the SM-TCR assumption for one of these functions. The distinct address of every hash call serves as the tweak, which lets a reduction embed one target per call.

Definition 4.4 (SM-TCR — Single-function Multi-target Target-Collision Resistance [BHK+19])**.** Let $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ be a tweakable hash function. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}TCR}_{\mathbf{Th},p}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Generate a random public parameter $P \leftarrow_{\$} \mathcal{P}$.

3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with classical access to an oracle $\mathcal{O}(\cdot,\cdot)$ that takes $T \in \mathcal{T}$ and $\mathsf{m} \in \mathcal{M}$ as inputs and works as follows:
   - If $|Q| \geq p$ or there is an $\mathsf{m}' \in \mathcal{M}$ with $(T, \mathsf{m}') \in Q$, return $\perp$.
   - Insert (T, m) into the list Q.
   - Return $\mathbf{Th}(P, T, \mathsf{m})$.
4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without the oracle access.
5. Obtain an output from $\mathcal{A}_2 : (j, \mathsf{m}^*) \leftarrow \mathcal{A}_2(P)$ with $\mathsf{m}^* \in \mathcal{M}, j \in [|Q|]$. Denote the jth entry in Q by (T_j, m_j).
6. Output 1 if $\mathbf{Th}(P, T_j, \mathsf{m}_j) = \mathbf{Th}(P, T_j, \mathsf{m}^*)$ and $\mathsf{m}^* \neq \mathsf{m}_j$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability against single-function, multi-target target collision resistance of a tweakable hash function as:

$$\mathrm{Succ}^{\text{SM-TCR}}_{\mathbf{Th},p}(\mathcal{A}) := \Pr[\mathbf{SM\text{-}TCR}_{\mathbf{Th},p}(\mathcal{A}) \Rightarrow 1].$$

The next notion is the multi-target analogue of preimage resistance. Here the challenge oracle returns images of uniformly random inputs that remain hidden from the adversary, who must later invert one of them. In the proof of SHRINCS, SM-PRE bounds the event that a forgery reveals a preimage of a WOTS+C or WOTS-TW chain value that the honest signer never disclosed.

Definition 4.5 (SM-PRE — Single-function Multi-target Preimage Resistance [BHK+19])**.** Let $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ be a tweakable hash function. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}PRE}_{\mathbf{Th},p}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Generate a random public parameter $P \leftarrow_{\$} \mathcal{P}$.
3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with classical access to an oracle $\mathcal{O}(\cdot)$ that takes $T \in \mathcal{T}$ as input and works as follows:
   - If $|Q| \geq p$ or there is an $y' \in \mathcal{N}$ with $(T, y') \in Q$, return $\perp$.
   - Sample $\mathsf{m} \leftarrow_{\$} \mathcal{M}$.
   - Compute $y = \mathbf{Th}(P, T, \mathsf{m})$.
   - Insert (T, y) into the list Q.
   - Return y.
4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without the oracle access.
5. Obtain an output from $\mathcal{A}_2 : (j, \mathsf{m}^*) \leftarrow \mathcal{A}_2(P)$ with $\mathsf{m}^* \in \mathcal{M}, j \in [|Q|]$. Denote the jth entry in Q by (T_j, y_j).
6. Output 1 if $\mathbf{Th}(P, T_j, \mathsf{m}^*) = y_j$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability against single-function, multi-target preimage resistance of a tweakable hash function as:

$$\mathrm{Succ}^{\text{SM-PRE}}_{\mathbf{Th},p}(\mathcal{A}) := \Pr[\mathbf{SM\text{-}PRE}_{\mathbf{Th},p}(\mathcal{A}) \Rightarrow 1].$$

Undetectability notion states that outputs of a tweakable hash function on secret uniformly random inputs are indistinguishable from uniformly random strings. It is the tool that enables the preimage embedding in the proof: the unrevealed WOTS+C and WOTS-TW chain values are replaced by independently uniform strings via a hybrid over the w chain positions, so that the SM-PRE challenges can be embedded.

Definition 4.6 (SM-UD — Single-function Multi-target Undetectability [BHK+19], [HK22, DKKW25])**.** Let $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ be a tweakable hash function. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}UD}_{\mathbf{Th},p}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Sample $b \leftarrow_{\$} \{0,1\}$ and $P \leftarrow_{\$} \mathcal{P}$.
3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with (classical) access to an oracle $\mathcal{O}(\cdot, b)$ that takes $T \in \mathcal{T}$ as input and works as follows:
   - If $|Q| \geq p$ or $T \in Q$, return $\perp$.
   - Insert T into the list Q.
   - If $b = 0$, sample $\mathsf{m} \leftarrow_{\$} \mathcal{M}$ and return $y := \mathbf{Th}(P, T, \mathsf{m})$.
   - If $b = 1$, return $y \leftarrow_{\$} \mathcal{N}$.
4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without the oracle access.
5. Obtain an output from $\mathcal{A}_2$: $b' \leftarrow \mathcal{A}_2(P)$, where $b' \in \{0,1\}$.
6. Output b'.

For any such algorithm $\mathcal{A}$, we define the advantage against single-function, multi-target undetectability of a tweakable hash function as:

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

The security proof of FXMSS additionally uses the following variant of SM-TCR. Because the structure of the stateful tree is flexible, a node of the imaginary complete binary tree of FXMSS can play one of two roles: it is either the compression of a WOTS+C public key, or a regular Merkle-tree node obtained by hashing its two children. The two roles use distinct address types, so a forgery may produce a collision across two *distinct* tweaks that refer to the same node. Crucially, the paired tweaks never appear in the scheme simultaneously: each node takes exactly one of the two roles, and they are in one-to-one correspondence. The definition below reflects this: the affected part of the tweak space splits into two halves $\mathcal{T}_2$ and $\mathcal{T}_3$ (the internal-node and the WOTS+C public-key-compression addresses, respectively), related by a bijection ω that pairs the tweaks referring to the same node. A challenge may be set at a tweak from one half, making the paired tweak from the other half unavailable, while the solution, i.e. the collision, may use either of the paired tweaks. In SHRINCS the notion is instantiated with the tweakable hash function $\mathsf{Th}_{2 \cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda$, combined from two members of the collection defined below (Definition 4.12): at every tweak it coincides with $\mathbf{H}$ on messages of length $2n$ and with $\mathsf{Th}_{l_{\mathsf{sf}}}$ on messages of length $l_{\mathsf{sf}} \cdot n$. The scheme uses it as the merge of the two node roles, with $2n$-bit messages on the tweaks in $\mathcal{T}_2$ and $l_{\mathsf{sf}} \cdot n$-bit messages on the tweaks in $\mathcal{T}_3$. The hardness of this notion is analyzed in Section E.

Definition 4.7 (Single-Function, Multi-Target Target Collision Resistance with Exclusive Tweaks (SM-TCR-EXCL))**.** Let $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ be a tweakable hash function. Let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$, where $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$ if $i \neq j$ and $|\mathcal{T}_2| = |\mathcal{T}_3|$. Let $\omega : \mathcal{T}_2 \to \mathcal{T}_3$ be a bijection. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}_1 \cup \mathcal{T}_2|]$. Consider the following experiment $\mathbf{SM\text{-}TCR\text{-}EXCL}_{\mathbf{Th},p}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Generate a random public parameter $P \leftarrow_{\$} \mathcal{P}$.
3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with classical access to an oracle $\mathcal{O}(\cdot,\cdot)$ that takes $T \in \mathcal{T}$ and $\mathsf{m} \in \mathcal{M}$ as inputs and works as follows:
   - If $|Q| \geq p$ return $\perp$.
   - If $T \in \mathcal{T}$ and there is an $\mathsf{m}' \in \mathcal{M}$ with $(T, \mathsf{m}') \in Q$ return $\perp$.
   - If $T \in \mathcal{T}_2$ and there is an $\mathsf{m}' \in \mathcal{M}$ with $(\omega(T), \mathsf{m}') \in Q$ return $\perp$.
   - If $T \in \mathcal{T}_3$ and there is an $\mathsf{m}' \in \mathcal{M}$ with $(\omega^{-1}(T), \mathsf{m}') \in Q$ return $\perp$.
   - Insert (T, m) into the list Q
   - Return $\mathbf{Th}(P, T, \mathsf{m})$.
4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without the oracle access.
5. Obtain an output from $\mathcal{A}_2 : (j, T^*, \mathsf{m}^*) \leftarrow \mathcal{A}_2(P)$ with $j \in [|Q|]$, $T^* \in \mathcal{T}$ and $\mathsf{m}^* \in \mathcal{M}$. Denote the jth entry in Q by (T_j, m_j).
6. If $T_j \in \mathcal{T}_1$ and $T^* \neq T_j$: Output 0.
7. If $T_j \in \mathcal{T}_2$ and ($T^* \neq T_j$ and $T^* \neq \omega(T_j)$): Output 0.
8. If $T_j \in \mathcal{T}_3$ and ($T^* \neq T_j$ and $T^* \neq \omega^{-1}(T_j)$): Output 0.
9. Output 1 if $\mathbf{Th}(P, T_j, \mathsf{m}_j) = \mathbf{Th}(P, T^*, \mathsf{m}^*)$ and $\mathsf{m}^* \neq \mathsf{m}_j$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability against single-function, multi-target target collision resistance with exclusive tweaks of a tweakable hash function as:

$$\mathrm{Succ}^{\text{SM-TCR-EXCL}}_{\mathbf{Th},p}(\mathcal{A}) := \Pr[\mathbf{SM\text{-}TCR\text{-}EXCL}_{\mathbf{Th},p}(\mathcal{A}) \Rightarrow 1].$$

In the stateful path, some hash values are produced by counter search: the signer iterates a counter until the digest satisfies a structural property. The following notion captures collisions on such counter-searched digests. In SHRINCS, Prop is instantiated with the WOTS+C target-sum condition: the encoded digest's base-w digits sum to the target S_{wn}, yielding S-TCR(+C).

Definition 4.8 (S-TCR(Prop) — Special Target-Collision Resistance)**.** Let $\mathbf{Th}$ be a THF. This notion is a variant of SM-TCR parameterized by a predicate Prop on the output space. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{S\text{-}TCR}(\mathsf{Prop})_{\mathbf{Th},p}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Generate a random public parameter $P \leftarrow_{\$} \mathcal{P}$.
3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with classical access to an oracle $\mathcal{O}^{\mathsf{Prop}}(\cdot,\cdot)$ that takes $T \in \mathcal{T}$ and $\mathsf{m} \in \mathcal{M}$ as inputs and works as follows:
   - If $|Q| \geq p$ or there is an $\mathsf{m}' \in \mathcal{M}$ with $(T, \mathsf{m}') \in Q$, return $\perp$.

- Find the smallest $i \in \mathbb{N}$ with $\mathsf{Prop}(\mathbf{Th}(P, T, \mathsf{m}\|i)) = 1$ and set $y = \mathbf{Th}(P, T, \mathsf{m}\|i)$.
- Insert $\{(T, \mathsf{m}), (y, i)\}$ into the list Q.
- Return (y, i).

4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without the oracle access.

5. Obtain an output from $\mathcal{A}_2 : (j, \mathsf{m}^*, ctr) \leftarrow \mathcal{A}_2(P)$ with $\mathsf{m}^* \in \mathcal{M}$, $j \in [|Q|]$, and $ctr \in \mathbb{N}$. Denote the jth entry in Q by $\{(T_j, \mathsf{m}_j), (y_j, i_j)\}$.

6. Output 1 if $\mathbf{Th}(P, T_j, \mathsf{m}_j\|i_j) = \mathbf{Th}(P, T_j, \mathsf{m}^*\|ctr)$ and $\mathsf{m}^* \neq \mathsf{m}_j$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability against special target collision resistance of a tweakable hash function as:

$$\mathsf{Succ}^{\text{S-TCR(Prop)}}_{\mathbf{Th},p}(\mathcal{A}) := \Pr[\mathbf{S\text{-}TCR}(\mathsf{Prop})_{\mathbf{Th},p}(\mathcal{A}) \Rightarrow 1].$$

We will use a Multi-Target Target Collision Resistance with Random Sampling notion to bound the success probability against FXMSS. The notion lets the challenge oracle internally sample fresh randomness ρ until the resulting digest satisfies a structural predicate Prop (or until a sampling budget K is exhausted). With $K = 1$ and $\mathsf{Prop} \equiv 1$ the oracle samples one ρ per query, matching the behaviour of the message hashing function $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ in SHRINCS.

Definition 4.9 (Multi-Target Target Collision Resistance with Random Sampling [DKKW25])**.** Let $K \in \mathbb{N}$ be an integer. Let $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R}) \to \mathcal{N}$ be a tweakable hash function, where the input is split into a message part $\mathsf{m} \in \mathcal{M}$ and a randomness part $\rho \in \mathcal{R}$. Let $\mathsf{Prop}\colon \mathcal{N} \to \{0, 1\}$ be a function that represents some property on the output space. Let $\mathcal{A}$ be an algorithm, and $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}rTCR}^K_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A})$:

1. Prepare an empty list Q.

2. Generate a random public parameter $P \leftarrow_{\$} \mathcal{P}$.

3. Run $\mathcal{A}$ with input P and (classical) access to an oracle $\mathcal{O}(\cdot, \cdot)$ that takes $T \in \mathcal{T}$ and $\mathsf{m} \in \mathcal{M}$ as inputs and works as follows:

   - If $|Q| \geq p$ or there is a tuple $(T, \mathsf{m}', \rho') \in Q$, for some m', ρ', then return $\perp$.
   - Otherwise, set $\mathsf{ctr} := 0$ and $y := \perp$. While $\mathsf{ctr} < K$ and $y = \perp$:
     (a) Sample $\rho \leftarrow_{\$} \mathcal{R}$.
     (b) Set $y := \mathbf{Th}(P, T, (\mathsf{m}\|\rho))$.
     (c) If $\mathsf{Prop}(y) = 1$: Insert (T, m, ρ) into Q.
     (d) Else: Set $y := \perp$, $\rho := \perp$.
     (e) Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
   - If $y = \perp$: Insert $(T, \mathsf{m}, \perp)$ into Q.
   - Return (y, ρ).

4. Obtain an output from $\mathcal{A}$: $(j, \mathsf{m}^*, \rho^*) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(P)$ with $\mathsf{m}^* \in \mathcal{M}$, $j \in [|Q|]$. Denote the jth entry in Q by $(T_j, \mathsf{m}_j, \rho_j)$.

5. Output 1 if $\mathbf{Th}(P, T_j, (\mathsf{m}_j\|\rho_j)) = \mathbf{Th}(P, T_j, (\mathsf{m}^*\|\rho^*))$ and $(\mathsf{m}^*, \rho^*) \neq (\mathsf{m}_j, \rho_j)$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability of any adversary $\mathcal{A}$ against Multi-Target Target Collision Resistance with Random Sampling SM-rTCR as:

$$\mathsf{Succ}^{\mathsf{SM\text{-}rTCR},K}_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A}) := \Pr[\mathbf{SM\text{-}rTCR}^K_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A}) \Rightarrow 1].$$

The key differences from SM-TCR are that the public parameter is immediately known to the adversary, the challenge oracle samples fresh randomness ρ, and may iterate until the digest satisfies Prop. The unique-tweak guard (T never repeats in Q) ensures the multi-target flavour of SM-TCR.

The last property for tweakable hash functions is a decisional variant of second-preimage resistance: the adversary must decide whether its chosen target has a second preimage under the tweakable hash function. Following [BH19], combining SM-DSPR with SM-TCR gives a tight bound on finding preimages in the setting where the adversary can reveal preimages of other targets. In SHRINCS, this bounds the recovery of unrevealed FORS secret values from a forgery in the stateless path, contributing the term $3 \cdot \mathrm{InSec}^{\text{SM-TCR}} + \mathrm{InSec}^{\text{SM-DSPR}}$.

Definition 4.10 (SM-DSPR — Single-function Multi-target Decisional Second-Preimage Resistance [BHK+19])**.** Let $\mathbf{Th}$ be a THF. The *second-preimage-exists predicate* is:

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

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}|]$. Consider the following experiment $\textbf{SM-DSPR}_{\mathbf{Th},p}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Generate a random public parameter $P \leftarrow_{\$} \mathcal{P}$.
3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with classical access to an oracle $\mathcal{O}(\cdot,\cdot)$ that takes $T \in \mathcal{T}$ and $\mathsf{m} \in \mathcal{M}$ as inputs and works as follows:
   - If $|Q| \geq p$ or there is an $\mathsf{m}' \in \mathcal{M}$ with $(T, \mathsf{m}') \in Q$, return $\perp$.
   - Insert (T, m) into the list Q.
   - Return $\mathbf{Th}(P, T, \mathsf{m})$.
4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without the oracle access.
5. Obtain an output from $\mathcal{A}_2 : (j, b) \leftarrow \mathcal{A}_2(P)$ with $j \in [|Q|]$ and $b \in \{0, 1\}$. Denote the jth entry in Q by (T_j, m_j).
6. Output 1 if $\mathrm{SP}_{P,T_j}(\mathsf{m}_j) = b$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, define

$$\begin{aligned} \mathrm{succ} &:= \Pr[\textbf{SM-DSPR}_{\mathbf{Th},p}(\mathcal{A}) \Rightarrow 1], \\ \mathrm{triv} &:= \Pr[\mathrm{SP}_{P,T_j}(\mathsf{m}_j) = 1], \end{aligned}$$

where the latter probability is taken over the same experiment. We define the advantage against single-function, multi-target decisional second-preimage resistance of a tweakable hash function as:

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

Subtracting triv removes the trivial strategy of always guessing $b = 1$: an adversary that never outputs $b = 0$ wins with probability triv, achieving zero advantage.

The last notion we present in this subsection is interleaved target-subset resilience, a notion for keyed hash functions. It captures the few-time nature of FORS within the hypertree: each signature reveals the FORS secret values at positions selected by the message digest, and a forgery on a fresh message whose positions are all covered by already-revealed ones is valid without breaking any other hash property. ITSR bounds the probability of this event; in SHRINCS it is applied to the message-digest function $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ of the stateless path, with one target per signing query.

Definition 4.11 (Interleaved Target-Subset Resilience (ITSR) [BHK+19])**.** Let $\mathrm{H} : \mathcal{K} \times \{0,1\}^{\alpha} \to \{0,1\}^{m}$ be a KHF. A deterministic mapping function $\mathrm{MAP}_{h,k,t} : \{0,1\}^{m} \to [0, 2^{h}-1] \times [0, t-1]^{k}$ interprets an m-bit string as a tree index $I \in [0, 2^{h}-1]$ and k leaf indices $(J_1, \ldots, J_k) \in [0, t-1]^{k}$; we write $G = \mathrm{MAP}_{h,k,t} \circ \mathrm{H}$ for the composed function and identify $G(K, M)$ with the set $\{(I, i, J_i) : i \in [k]\}$. The adversary wins by producing a *fresh* key-message pair whose entire leaf-index set is *covered* by the union of all previously observed outputs. Let $\mathcal{A}$ be an algorithm. Consider the following experiment $\mathbf{ITSR}_{\mathrm{H},q}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Run $\mathcal{A}$ with (classical) access to an oracle $\mathcal{O}(\cdot)$ that takes $M \in \{0,1\}^{\alpha}$ as input and works as follows:
   - If $|Q| \geq q$, return $\perp$.
   - Generate a random key $K \leftarrow_{\$} \mathcal{K}$.
   - Insert (K, M) into the list Q.
   - Return $(K, G(K, M))$.
3. Obtain an output from $\mathcal{A}$: $(K^*, M^*) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}()$, with $K^* \in \mathcal{K}$ and $M^* \in \{0,1\}^{\alpha}$.
4. Output 1 if $(K^*, M^*) \notin Q$ and $G(K^*, M^*) \subseteq \bigcup_{(K_i, M_i) \in Q} G(K_i, M_i)$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability against interleaved target-subset resilience of H as:

$$\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},q}(\mathcal{A}) := \Pr[\mathbf{ITSR}_{\mathrm{H},q}(\mathcal{A}) \Rightarrow 1].$$

4.3 Collections of Tweakable Hash Functions

SHRINCS uses tweakable hash functions with different message lengths. Following [HK22], we model them as a collection $\mathsf{Th}_{\lambda} = \{\mathsf{Th}_{m_i}\}$ of THFs that share the public parameter space $\mathcal{P}$ and the tweak space $\mathcal{T}$ and differ in the length m_i of the messages they process; in the scheme, all members are keyed by the same public parameter P, and we write $\mathsf{Th}_m \in \mathsf{Th}_{\lambda}$ to indicate membership. Each member is defined on a tweak set $\mathcal{T}_{m_i} \subseteq \mathcal{T}$, by default all of $\mathcal{T}$, and the lengths m_i are pairwise distinct, so that the length of a message determines the member. In SHRINCS the grinding hash $\mathbf{H}_{\mathrm{grind}}$ is the member with messages of $2n + c$ bits, and its tweak set is the set of addresses of type `SF_WOTS_C_GRIND` whose payload (bytes 10–21) is zero (Section 2.2). The shared parameter creates a gap between the scheme and the stand-alone notions above: in SHRINCS, the input of a hash call is computed from outputs of other members of the collection (or of the same member at different tweaks) under the same P, while in the stand-alone experiments the adversary learns P only after the challenge phase and hence cannot form such inputs for its challenge queries. As in [HK22, Def. 7], we close this gap by giving the first stage of the adversary oracle access to the collection.

Definition 4.12 (Security for members of a collection, following [HK22, Def. 7])**.** Let Th_{λ} be a collection of THFs as described above, and let $\mathsf{Th}_m \in \mathsf{Th}_{\lambda}$ be the member with message space $\{0,1\}^{m}$. Let $\mathsf{Th}_{\lambda}(P, \cdot, \cdot)$ denote the oracle that takes a tweak $T \in \mathcal{T}$ and a message m as input and returns $\mathsf{Th}_{m'}(P, T, \mathsf{m})$ for $m' = |\mathsf{m}|$ if $T \in \mathcal{T}_{m'}$, and $\perp$ otherwise. Consider a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the SM-TCR (SM-PRE, SM-UD, SM-TCR-EXCL, S-TCR(Prop), SM-DSPR) security of Th_m *as a member of the collection* Th_{λ}. We define the success probability (respectively, advantage) of $\mathcal{A}$ against the SM-TCR (SM-PRE, SM-UD, SM-TCR-EXCL, S-TCR(Prop), SM-DSPR) security of $\mathsf{Th}_m \in \mathsf{Th}_{\lambda}$ as its success probability (advantage) in the corresponding stand-alone experiment defined above, modified as follows: $\mathcal{A}_1$ is additionally given classical access to $\mathsf{Th}_{\lambda}(P, \cdot, \cdot)$, subject to the restriction that *tweaks are reserved by challenge queries*: no tweak used in a challenge query may appear in any collection-oracle query, to any

member of the collection. For SM-TCR-EXCL, a challenge query at T additionally reserves the partner tweak $\omega(T)$ (respectively, $\omega^{-1}(T)$).

The rule applies in both orders: a challenge query is refused if its tweak, or for SM-TCR-EXCL its partner tweak, already appears in a collection-oracle query.

The challenge oracle likewise refuses a tweak outside the tweak set $\mathcal{T}_m$ of the challenged member; for $\mathsf{Th}_{2\cup l_{\mathsf{sf}}}$, below, every tweak of $\mathcal{T}$ is admitted.

The function $\mathsf{Th}_{2\cup l_{\mathsf{sf}}}$ of SM-TCR-EXCL, combined from $\mathbf{H}$ and $\mathsf{Th}_{l_{\mathsf{sf}}}$ by message length (before Definition 4.7), is not itself a member; we write $\mathsf{Th}_{2\cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda$ for its SM-TCR-EXCL security in this experiment.

As an example, Section D.1 spells out the resulting experiment for SM-TCR; the experiments for the other properties are obtained in the same way. Both oracles are classical, so the reservation constraint refers to well-defined query transcripts. Reductions simulating SHRINCS comply with it automatically, since the addressing scheme assigns every hash call of the scheme a distinct address (and, for SM-TCR-EXCL, each node of the FXMSS tree takes exactly one of the two paired roles).

When attacking a member of a collection, we parameterize an adversary by three resources: the number p of challenge-oracle queries, which for all properties above equals the number of targets; the total number q_λ of collection-oracle queries, counted across all members of the collection; and the time ξ, measured in function evaluations. The time ξ counts the evaluations that $\mathcal{A}$ performs itself (evaluations of members of Th_λ, at any public parameter and any tweak), while each oracle interface carries its own query count. Here the interfaces are the challenge and collection oracles, counted by p and q_λ: during the first stage the public parameter is hidden, so these oracles evaluate the collection on the adversary's behalf. Computation that evaluates no function is charged to none of these resources: bookkeeping, table maintenance, and memory are uncharged. We measure time this way because it is the metric in which the generic bounds of Table 6 are proven: bounds for ideal (random) functions count oracle queries, and computation that makes no query does not affect them. Section 5.1 discusses the convention and its limits.

The same convention applies to the forgers in the scheme-level theorems: their time counts their own hash evaluations, including their (possibly quantum) queries to the random oracle for the message hash, whose number on that interface we additionally track as q_h (so $q_h \le \xi$); the interaction with the signing oracle, which requires the secret key, is counted separately by q_s.

We work with post-quantum, single-function, multi-target versions of the properties above, where adversaries may query at most one challenge per tweak (DT restriction, which we omit from the notation since all our properties carry it). The *insecurity* of property Prop of a hash function F for parameter p against time-ξ adversaries is:

$$\mathrm{InSec}^{\mathrm{Prop}}(F;\, \xi,\, p) = \max_{\mathcal{A}} \big\{ \mathrm{Succ}^{\mathrm{Prop}}_{F,p}(\mathcal{A}) \text{ or } \mathrm{Adv}^{\mathrm{Prop}}_{F,p}(\mathcal{A}) \big\},$$

where the maximum ranges over all adversaries running in time ξ, for quantum adversaries over the halting ones (Remark C.1). For a member of a collection, the insecurity carries q_λ as a third parameter:

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda) = \max_{\mathcal{A}} \big\{ \mathrm{Succ}^{\mathrm{Prop}}_{\mathsf{Th}_m \in \mathsf{Th}_\lambda,\, p}(\mathcal{A}) \text{ or } \mathrm{Adv}^{\mathrm{Prop}}_{\mathsf{Th}_m \in \mathsf{Th}_\lambda,\, p}(\mathcal{A}) \big\},$$

where the maximum ranges over all adversaries[8] that run in time ξ, make at most p challenge-oracle queries, and make at most q_λ collection-oracle queries.

The same rule gives keyed hash functions an optional third parameter. For a keyed hash function G that all parties evaluate by oracle queries (in SHRINCS, the message-digest functions $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$), we track the adversary's evaluations of G itself: $\mathrm{InSec}^{\mathrm{Prop}}(G;\, \xi,\, p,\, q_G)$ takes

[8] As for every insecurity in this document, the maximum over quantum adversaries ranges over halting ones (Remark C.1).

the maximum over adversaries[9] that run in time ξ, make at most p challenge queries, and evaluate G at most $q_G \leq \xi$ times; omitting the parameter sets $q_G = \xi$. Known generic bounds on these insecurities are collected in Table 6; they hold when the hash function is modeled as a (quantum) random oracle, are proven for the stand-alone notions, except the SM-DSPR row, which is conjectured [BHK+19, BH19] (Section 7), and depend only on ξ and p. Conjecture 4.1 and Lemma 4.2 relate the remaining parameters q_λ and q_G to these bounds.

The reductions in our security proof simulate SHRINCS for a forger, and this simulation recomputes large parts of the SHRINCS structure (for example all WOTS-TW chains of a hypertree). Under the accounting above, the recomputation consists of collection-oracle queries at non-challenge tweaks: the resulting adversaries have a large q_λ, while their time ξ stays close to the forger's. This accounting is justified by the following conjecture, which goes back to [HK22, Sect. 8.1]: access to the collection at tweaks disjoint from the challenge tweaks does not help the adversary.

Remark 4.1. All queries that the reductions of Section 4 issue to the challenge and collection oracles are classical, since those interfaces are classical by definition, so the budgets $q_{\lambda,i}$ count classical queries, as Conjecture 4.1 below requires. Quantum access arises only on the message-hash interface, where the reductions relay or simulate the forger's random-oracle queries; the forger's own evaluations of the tweakable hash functions are its own work and are charged to ξ.

Conjecture 4.1 (Collection-oracle queries do not affect insecurity; cf. [HK22, Sect. 8.1])**.** *Let* Th_λ *be the collection of tweakable hash functions used in SHRINCS, and let* Prop *be one of the properties of Definition 4.12. Then for every member* $\mathsf{Th}_m \in \mathsf{Th}_\lambda$*, and for* $\mathsf{Th}_m := \mathsf{Th}_{2\cup l_{\mathsf{sf}}}$ *(Definition 4.12), and all* ξ*,* p*, and* q_λ*,*

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda) = \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p).$$

The direction $\geq$ holds unconditionally, since an adversary may ignore the collection oracle; the conjectured content is the direction $\leq$. The supporting argument of [HK22] treats the ideal case: for a random tweakable function, every tweak defines an independent random function, so classical evaluations at tweaks disjoint from the challenge tweaks reveal nothing about the targets. In the ideal model the conjecture holds up to a parameter-guessing term, both for classical adversaries and for adversaries that evaluate the collection in superposition (Lemma 4.1 below); for the concrete collection it remains an assumption (Remark 5.1).

Corollary 4.1. *Assume Conjecture 4.1. Then every upper bound on the stand-alone insecurity* $\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p)$*, including the generic bounds of Table 6, applies to* $\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda)$ *for every* q_λ*. In particular, the simulation work that a reduction routes through the collection oracle does not degrade the resulting security bound. Here, as in Conjecture 4.1,* Th_m *is a member of* Th_λ *or* $\mathsf{Th}_{2\cup l_{\mathsf{sf}}}$*.*

For ideal collections the conjecture holds up to a parameter-guessing term, both against classical adversaries and against adversaries that evaluate the collection themselves in superposition. We discuss the applicability of the lemma in Section 5.4; the proof, common to both classes, is in Section D.2.

Lemma 4.1 (Collection queries are harmless in the ideal model)**.** *Model the collection* Th_λ *as random oracles: for every member* $\mathsf{Th}_m \in \mathsf{Th}_\lambda$*, every* $P \in \mathcal{P}$*, and every* $T \in \mathcal{T}$*, let* $\mathsf{Th}_m(P, T, \cdot)$ *be an independent uniformly random function, which all parties evaluate by oracle queries. Let* Prop *be one of the properties of Definition 4.12, measure the time* ξ *in function evaluations, as fixed in Section 4.3, and let the adversaries access the challenge and collection oracles classically, as Definition 4.12 prescribes. Let* Th_m *be a member of* Th_λ *or* $\mathsf{Th}_{2\cup l_{\mathsf{sf}}}$ *(Definition 4.12). Then*

[9] As for every insecurity in this document, the maximum over quantum adversaries ranges over halting ones (Remark C.1).

for all ξ, p, and q_λ, and for each of the two adversary classes below, with both sides of every inequality ranging over the same class,

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p) \;\le\; \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda),$$

and, in the other direction,

(i) for classical adversaries,

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda) \;\le\; \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p) \;+\; \frac{2\xi}{|\mathcal{P}|};$$

(ii) for adversaries that evaluate the collection themselves with quantum access, with output space $\{0,1\}^\nu$ and the XOR oracle (Remark C.2), counted by ξ, where $\xi_1 \le \xi$ is the number of own evaluations performed in the first stage,

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda) \;\le\; \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p) \;+\; \frac{2\,\xi_1}{\sqrt{|\mathcal{P}|}},$$

where the additive term is $4\,\xi_1/\sqrt{|\mathcal{P}|}$ for the advantage notions SM-UD and SM-DSPR.

The additive terms are parameter-guessing artifacts of the proofs.

Conjecture 4.1 concerns the collection interface, and its scope is the properties of Definition 4.12. The message-hash terms of our theorems lie outside this machinery: the SM-rTCR and reprogramming terms of Theorem 4.2 and the ITSR term of Theorem 4.3 are played against the keyed functions $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, and their experiments carry no collection oracle. The reductions behind the FXMSS terms hold the public parameter from the start, so their simulation work is charged to time; this is the term $\tilde{\xi}' \le \xi + t_{\mathsf{sim}}$ of Theorem 4.2. A second separation principle applies to these terms: the following lemma rests on the independence of the evaluated functions. The generic bounds for these properties count queries to the message-digest function alone, and evaluations of other, independently modeled functions leave them unchanged.

Lemma 4.2 (Evaluations of independent functions are harmless in the ideal model)**.** *Let G be a keyed hash function modeled as a random oracle, and model every further function that an adversary may evaluate as a random oracle independent of G; all parties evaluate all functions by oracle queries, and time is measured in function evaluations. Let* Prop *be a security experiment for G whose challenger, auxiliary oracles, and win condition evaluate G only;* SM-rTCR *(Definition 4.9), adaptive reprogramming (Definition C.1) and its distinct-inputs variant (Definition C.2), and* ITSR *(Definition 4.11) are of this form. Then for all ξ, p, and $q_G \le \xi$,*

$$\mathrm{InSec}^{\mathrm{Prop}}(G;\, \xi,\, p,\, q_G) = \mathrm{InSec}^{\mathrm{Prop}}(G;\, q_G,\, p),$$

with any further query parameters of the experiment unchanged on both sides: evaluations of the independent functions may be omitted from the accounting. The statement covers classical adversaries and adversaries that evaluate the oracles in superposition.

The proof is an exact simulation argument, simpler than its collection counterpart Lemma 4.1, and is given in Section D.3. As with the collection interface, ideality is essential: for the concrete functions, the independence of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ from the tweakable hash calls and the remaining keyed functions is an assumption about the SHA-256 instantiation, stated as Assumption 4.1 below and supported by Lemma 4.3. Theorem 4.2 displays the $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$-evaluation counts of its reductions as query parameters for exactly this purpose, and Section 6.3 instantiates the SM-rTCR and reprogramming rows through the lemma; for adaptive reprogramming the role of q_G is played by the oracle-query count that the notion already carries. The same argument carries the ITSR term of Theorem 4.3, and there it does far more work. The reduction behind that term builds the stateless path and answers up to 2^{40} signing queries itself, about 2^{61} hash evaluations, while it queries $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ only $\tilde{q}^{\mathsf{sl}}_h \le q_h + 1$ times. By the lemma only the latter count enters the bound.

4.4 Modeling the Instantiated Functions

The ideal-model results above treat the collection and the keyed functions as independent random objects. In SHRINCS, however, every function is built from SHA-256: most make a single call on a structured input, while the message hashes and the HMAC-based randomizers make two nested calls. Their inputs are kept apart by the domain separation of Section 2.2. We condense that discussion into Lemma 4.3, which names the only events in which calls from different functions can receive the same SHA-256 input, and Assumption 4.1, which models the functions as independent. Three parts of the analysis rest on this assumption: Conjecture 4.1 applied to the concrete collection, Lemma 4.2, and the separation of the two paths in the SHRINCS proof. The assumption separates distinct functions only; separation *within* the collection, across tweaks, is left to Conjecture 4.1.

The model idealizes each named function as a whole and leaves its internal SHA-256 calls outside. A message hash therefore enters the model as a single random function, in the same way that a single SHA-256 call enters without its compression structure and $\mathbf{PRF}_{\mathbf{msg}}$ without its inner HMAC calls. The internal calls do connect the functions in ways the model cannot see. Remark 4.2 examines these connections and explains why we do not expect them to break Assumption 4.1. The analysis in this document, however, covers only the model in which the named functions are random.

Lemma 4.3 (Domain separation of the instantiated functions)**.** *Instantiate the functions of SHRINCS as in Section 2.2: the members F, H, Th_k, $\mathsf{Th}_{l_{\mathsf{sl}}}$, $\mathsf{Th}_{l_{\mathsf{sf}}}$, and $\mathbf{H}_{\mathrm{grind}}$ of the collection Th_λ, the function $\mathbf{PRF}$, the message hashes $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, and the randomizer functions $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$. Evaluations of the collection members, $\mathbf{H}_{\mathrm{grind}}$ included, and $\mathbf{PRF}$ perform one SHA-256 call each; evaluations of the message hashes and the randomizer functions perform two nested calls. These functions are domain separated on inputs whose public seed is that of the generated key, whose root, for each message hash, is the generated root of its own path, and whose address type, for the functions that take one, is one of the function's own types (Table 2); the randomizer, message, idx, counter, remaining address fields, and every other argument are arbitrary: two SHA-256 calls made by two different functions, or by one member of Th_λ at two different tweaks, on such inputs receive the same input only in the following events:*

- *(i) the inner calls of the two message hashes receive the same input; this requires $\mathsf{PK.sl} = \mathsf{PK.sf}$;*
- *(ii) a call of a message hash and a padded-seed call (Th_λ, $\mathbf{PRF}$, $\mathbf{H}_{\mathrm{grind}}$) or an HMAC call of a randomizer function receive the same input; this requires $\mathsf{PK.seed}$ and one of the two path roots to equal matching 16-byte constants simultaneously;*
- *(iii) an outer call of one message hash and an inner call of the other receive the same input; this requires an evaluated digest whose first half equals the other path's root. Within the scheme's own calls the message wrappers strengthen the requirement in both directions (Remark 4.2):*
  - *(iii.a) the outer call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ never matches an inner call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$: the wrapped message makes every honest inner string of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ at least 75 bytes long, longer than both outer strings (68 and 73 bytes);*
  - *(iii.b) the outer call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ matches an inner call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ only if, in addition to the digest half equal to $\mathsf{PK.sl}$, the wrapper's prefix bytes and the embedded root $\mathsf{PK.sf}$ reappear inside the digest and idx bytes.*

The proof, which lays out the SHA-256 input of every function byte by byte, is given in Section D.4.

Remark 4.2 (Outer-inner coincidences of the message hashes)**.** Event (iii) deserves a closer look, both for what the scheme does about it and for what makes it real. Within the scheme's

own calls the wrappers of Lemma 4.3 (iii.a) and (iii.b) close most of the surface. Direction (iii.a) has probability zero: the honest inner strings of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ are longer than both outer strings, so no evaluation can produce the coincidence. Direction (iii.b) compounds four independent conditions: the digest half equal to $\mathsf{PK.sl}$ (2^{-n}), the wrapper's leading zero byte (2^{-8}), a length byte among the at most eight admissible values (2^{-5}), and the embedded 16-byte $\mathsf{PK.sf}$ reappearing inside the remaining digest and idx bytes (2^{-n}); together at most 2^{-2n-13} per evaluated call, so the scheme's own $q_s + 1$ evaluations contribute at most $(q_s + 1) \cdot 2^{-2n-13}$.

The functions as defined, however, take an arbitrary message argument, and the adversary's direct queries are not wrapped; so we discuss unwrapped case here for completeness. Through the function interfaces alone it is invisible: the bytes linking the outer call of one message hash to the inner family of the other are an inner digest, which neither oracle reveals, so even when the event occurs its holder cannot name the colliding input. What surfaces it is that SHA-256 is public. The adversary computes inner digests of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ directly, at one SHA call per attempt. Once an attempt yields $h^{\mathsf{sf}}_{\mathrm{in}}[0{:}16] = \mathsf{PK.sl}$, the outer string $R \,\|\, \mathsf{PK.seed} \,\|\, h^{\mathsf{sf}}_{\mathrm{in}} \,\|\, \mathsf{idx}$ is bytewise the inner string of the call $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sl}, M')$ on the message $M' = h^{\mathsf{sf}}_{\mathrm{in}}[16{:}32] \,\|\, \mathsf{idx}$. The inner digest of that call therefore equals the $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ output y^*, and one further SHA call predicts the full output, $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sl}, M') = \mathsf{SHA\text{-}256}(R \,\|\, \mathsf{PK.seed} \,\|\, y^* \,\|\, 0^4)$, a relation that independent random oracles satisfy with probability 2^{-2n}. In the swapped direction, an attempt with $h^{\mathsf{sl}}_{\mathrm{in}}[0{:}16] = \mathsf{PK.sf}$ makes the outer string $R \,\|\, \mathsf{PK.seed} \,\|\, h^{\mathsf{sl}}_{\mathrm{in}} \,\|\, 0^4$ bytewise the inner string of the call $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}', M')$ with $\mathsf{idx}' = h^{\mathsf{sl}}_{\mathrm{in}}[16{:}25]$ and $M' = h^{\mathsf{sl}}_{\mathrm{in}}[25{:}32] \,\|\, 0^4$, and the prediction reads $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}', M') = \mathsf{SHA\text{-}256}(R \,\|\, \mathsf{PK.seed} \,\|\, y^* \,\|\, \mathsf{idx}')$ for the $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ output y^*; here digest bytes serve as the leaf identifier, which the functions as defined permit. An attempt succeeds with probability $\approx 2^{-n}$ classically, and q quantum evaluations succeed with probability roughly $8(q+1)^2 \cdot 2^{-n}$ [HRS16]; the attempts absorb $\mathsf{PK.seed}$ and the root, so precomputation does not apply.

The modeling of Assumption 4.1 hides the internal calls from the model. This is the sense in which the assumption has the same standing as modeling SHA-256, or the HMAC calls of $\mathbf{PRF}_{\mathbf{msg}}$, as ideal.

Assumption 4.1 (Independence of the instantiated functions)**.** We model the functions of $\mathsf{SHRINCS}$ as five mutually independent random objects: the collection Th_λ as one block (its members include the grinding hash $\mathbf{H}_{\mathrm{grind}}$, Section 4.3), and each of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, $\mathbf{PRF}$, and the combined randomizer function $\mathbf{PRF}_{\mathbf{msg}}$ (Section 2.2) as an independent uniformly random function. The events of Lemma 4.3 are where the instantiation can contradict this modeling, and their probabilities rest on two explicit distributional premises: $\mathsf{PK.seed}$ is uniform by definition of key generation, and the roots $\mathsf{PK.sl}$ and $\mathsf{PK.sf}$ are assumed to also be independent uniform n-bit values. Under these premises, event (i), the collision of the two roots, has probability 2^{-n}, and the ten pairings of event (ii), each requiring two of the listed values to equal prescribed 16-byte constants, have total probability at most $10 \cdot 2^{-2n}$. Every bound in this document therefore holds up to the additive term

$$\varepsilon_{\mathrm{ds}} \;\le\; 2^{-n} + 10 \cdot 2^{-2n},$$

charged once, in Theorem 4.1. Event (iii) is not charged: it concerns the internal instantiation of the functions, which the modeling does not resolve.[10]

We discuss the problem of reduction accounting further in Section 5.

4.5 Signature Security

In this subsection we give formal definitions of signature schemes and their security requirements. We start with the standard definition for signature schemes.

[10] Remark 4.2 discusses the event and why the modeling may absorb it.

Definition 4.13 (Digital Signature Scheme)**.** A *digital signature scheme* $\mathsf{Dss} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Verify})$ is a triple of probabilistic polynomial-time algorithms: $\mathsf{Kg}(1^n)$ outputs $(\mathsf{sk}, \mathsf{pk})$; $\mathsf{Sign}(\mathsf{sk}, M)$ outputs a signature σ; $\mathsf{Verify}(\mathsf{pk}, \sigma, M)$ outputs 1 iff σ is valid.

The standard security requirement for the signature scheme is existential unforgeability under chosen message attack. The adversary, given a public key, may query a signing oracle with messages and receives signatures for them in response. The task is to produce a signature on a new message that has not been queried before.

Definition 4.14 (EU-CMA)**.** The existential unforgeability under adaptive chosen message attacks (EU-CMA) success probability of an adversary $\mathcal{A}$ making q signing queries is:

$$\mathrm{Succ}^{\text{EU-CMA}}_{\mathsf{Dss},q}(\mathcal{A}) = \Pr\big[(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Kg}(1^n);\ (M^*,\sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk}) : \mathsf{Verify}(\mathsf{pk},\sigma^*,M^*) = 1 \wedge M^* \notin \{M_i\}_{i=1}^{q}\big].$$

We adapt the EU-CMA game to the SHRINCS setting.

Adversarially chosen tree structure. In SHRINCS the signer is free to choose the shape of the stateful Merkle tree (e.g. balanced or unbalanced, with any admissible height). To show that no choice of structure creates an attack vector, we give the adversary the power to make this choice on behalf of the signer: a first stage $\mathcal{A}_1$ outputs a shape descriptor Π (not necessarily B or U) before the public key is generated. The scheme parameters allow trees of depth up to 255, so $\mathcal{A}_1$ may in principle specify a structure that cannot be built efficiently; in that case key generation does not terminate and the adversary trivially fails to forge the signature. We therefore implicitly restrict to descriptors whose key generation is efficient.

Adversarially chosen signing path. Each signing query carries an extra bit $b \in \{0,1\}$ that selects whether the oracle uses the stateful path ($b = 0$) or the stateless path ($b = 1$). This models the fact that in SHRINCS either path may produce a valid signature for any message, and an adversary should not be able to exploit the choice of path.

Contexts. SHRINCS signs a message together with a caller-chosen context string of at most 255 bytes, and both paths digest the injectively encoded pair: the bound message $\widetilde{M}$ on the stateful path, and the ctx-wrapped SLH-DSA input on the stateless path (Section 3). For the security model we regard this encoded pair as *the* message: the queries M_i and the forgery M^* below range over pairs (ctx, M), identified with their encodings, and freshness of the forged pair coincides with freshness of the encoded string, since the encoding is injective. In particular, reusing a queried message under a different context is a fresh forgery in the model, so the bounds of Section 4 also certify cross-context unforgeability: a signature issued with one context verifies under another only through a forgery.

Definition 4.15 (SHRINCS EU-CMA)**.** Let $\mathsf{SHRINCS} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Verify})$ be the SHRINCS signature scheme. A two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ plays the following game with q_s signing queries.

1. *Structure phase.* $\mathcal{A}_1$ outputs a shape descriptor Π for the stateful path (not necessarily with $\Pi \in \{\mathsf{B}, \mathsf{U}\}$) that describes a valid structure (Section 2.5); a shape other than B and U is a future shape in the sense of Section 2.5, and Π then also specifies its anchor and internal-node sets and its map $\mathsf{leafSelect}_\Pi$, which must have the properties required there.

2. *Key generation.* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Kg}(1^n, \Pi)$.

3. *Signing phase.* $\mathcal{A}_2(\Pi, \mathsf{pk})$ makes q_s adaptive queries (M_i, b_i) to a signing oracle $\mathcal{O}_{\mathsf{sk}}(\cdot,\cdot)$. On query (M_i, b_i) with $b_i \in \{0,1\}$:

   - If $b_i = 0$ *(stateful path)*: return $\mathsf{Sign}_{\mathsf{sf}}(\mathsf{sk}, M_i)$ using the next unused stateful key; return $\perp$ if all stateful keys are exhausted.

- If $b_i = 1$ *(stateless path)*: return $\mathsf{Sign}_{\mathsf{sl}}(\mathsf{sk}, M_i)$.

4. *Forgery.* $\mathcal{A}_2$ outputs (M^*, σ^*).

The SHRINCS EU-CMA success probability of $\mathcal{A}$ is:

$$\mathrm{Succ}^{\text{SHRINCS-EU-CMA}}_{\text{SHRINCS},q_s}(\mathcal{A}) = \Pr\big[(\Pi) \leftarrow \mathcal{A}_1();\ (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Kg}(1^n,\Pi);\ (M^*,\sigma^*) \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{sk}}(\cdot,\cdot)}(\Pi,\mathsf{pk}) : \mathsf{Verify}(\mathsf{pk},\sigma^*,M^*) = 1 \ \wedge\ M^* \notin \{M_i\}_{i=1}^{q_s}\big].$$

4.6 Main Security Theorems

We show that the EU-CMA security of SHRINCS reduces to the properties of the hash functions used by FXMSS on the stateful path and by SLH-DSA on the stateless path. We first sketch the argument at the level of the two components (Remark 4.3) and then state the precise bound in Theorem 4.1. Theorems 4.2 and 4.3 give the bounds of the two paths as stand-alone schemes; the proof of Theorem 4.1 reuses their reductions and embeds the other path into them.

Remark 4.3 (Component-level view of SHRINCS security, informal)**.** A SHRINCS public key contains one root for each component, and a forgery verifies against exactly one of them. A forger therefore breaks either SLH-DSA under the stateless key or FXMSS under the stateful key, and in the notation of Theorem 4.1 one expects a bound of the form

$$\mathrm{Succ}^{\text{SHRINCS-EU-CMA}}_{\text{SHRINCS},q_s}(\mathcal{A}) \le \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF};\,\xi,\,q_1) + \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF_{msg}};\,\xi,\,q_s) + \mathrm{InSec}^{\text{EU-CMA}}(\text{SLH-DSA};\,\xi',\,q_s^{\mathsf{sl}}) + \mathrm{InSec}^{\text{EU-CMA}}(\text{FXMSS};\,\xi',\,q_s^{\mathsf{sf}}) \ +\ \varepsilon_{\mathrm{ds}}\,,$$

where ξ' is ξ plus the cost of building and signing the other path. The two PRF terms allow us to split the paths. After the secret elements and randomizers are replaced by uniform values, the two paths are independent, so a reduction can take the experiment of one component as its challenge and simulate the other path itself.

Theorem 4.1 does not use this form. It expands each component term into the properties of the underlying hash functions, for two reasons. First, in the games of the proof both paths run with uniform secret elements and randomizers, so the component terms would bound idealized experiments and not the schemes of Theorems 4.2 and 4.3. Second, both paths make their tweakable-hash calls, including the counter search of $\mathbf{H}_{\mathrm{grind}}$, to the same collection Th_λ, and the collection terms of each path are played against this collection. A reduction that uses a component bound as a black box must simulate the other path in its second stage, and once the component bound is expanded, the cost of that simulation appears in the time of every collection term. We therefore open both component proofs and modify their reductions.

In Theorem 4.1 the two PRF terms and $\varepsilon_{\mathrm{ds}}$ stay as above. The stateful component term becomes the SM-rTCR term, the reprogramming term, and the seven collection terms of Theorem 4.2. The stateless component term becomes the ten summands of Theorem 4.3 other than its two PRF terms.

Theorem 4.1 (SHRINCS EU-CMA Security)**.** *Let n, w, h_{sf}, h_{sl}, d, m, t, k be the parameters of SHRINCS (Sections 2.5 and 2.7) and let Th_λ be the collection of tweakable hash functions (Section 4.3) used throughout, of which $\mathbf{H}_{\mathrm{grind}}$ is a member. Model $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as a (quantum) random oracle, and every function of the scheme as in Assumption 4.1, and assume Conjecture 3.1 (see Lemma 3.2). For any classical or quantum adversary $\mathcal{A}$ running in time ξ that makes at most q_s*

signing queries and at most q_h random-oracle queries to each of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$:

$$
\begin{aligned}
\mathrm{Succ}^{\textit{SHRINCS-EU-CMA}}_{\textit{SHRINCS},q_s}(\mathcal{A}) \le\ & \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF};\, \xi,\, q_1) + \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}_{\mathbf{msg}};\, \xi,\, q_s) \\
& + \mathrm{InSec}^{\mathsf{SM\text{-}rTCR},1}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}},\, \mathbf{1};\, \tilde{\xi}'_{\mathsf{sf}},\, q_s^{\mathsf{sf}},\, \tilde{q}_h^{\mathsf{sf}}) + \mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}};\, \tilde{\xi}'_{\mathsf{sf}},\, q_s^{\mathsf{sf}},\, q_s^{\mathsf{sf}} + q_h + 1) \\
& + \mathrm{InSec}^{\text{S-TCR(Prop)}}(\mathbf{H}_{\mathrm{grind}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_s^{\mathsf{sf}},\, \tilde{q}_{\lambda,0}) + w \cdot \mathrm{InSec}^{\mathrm{SM\text{-}UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_1^{\mathsf{sf}},\, \tilde{q}_{\lambda,1}) \\
& + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_2^{\mathsf{sf}},\, \tilde{q}_{\lambda,2}) + \mathrm{InSec}^{\mathrm{SM\text{-}PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_1^{\mathsf{sf}},\, \tilde{q}_{\lambda,1}) \\
& + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_4^{\mathsf{sf}},\, \tilde{q}_{\lambda,4}) + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_3^{\mathsf{sf}},\, \tilde{q}_{\lambda,3}) \\
& + \mathrm{InSec}^{\mathrm{SM\text{-}TCR\text{-}EXCL}}(\mathsf{Th}_{2\cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_5^{\mathsf{sf}},\, \tilde{q}_{\lambda,5}) \\
& + \mathrm{InSec}^{\mathrm{ITSR}}(\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}};\, \tilde{\xi}''_{\mathsf{sl}},\, q_s^{\mathsf{sl}},\, \tilde{q}_h^{\mathsf{sl}}) + w \cdot \mathrm{InSec}^{\mathrm{SM\text{-}UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_2^{\mathsf{sl}},\, q_\lambda) \\
& + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_3^{\mathsf{sl}},\, q_\lambda) + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_7^{\mathsf{sl}},\, q_\lambda) \\
& + \mathrm{InSec}^{\mathrm{SM\text{-}PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_2^{\mathsf{sl}},\, q_\lambda) \\
& + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_4^{\mathsf{sl}},\, q_\lambda) + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathsf{Th}_k \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_5^{\mathsf{sl}},\, q_\lambda) \\
& + \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathsf{Th}_{l_{\mathsf{sl}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_6^{\mathsf{sl}},\, q_\lambda) \\
& + 3 \cdot \mathrm{InSec}^{\mathrm{SM\text{-}TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_8^{\mathsf{sl}},\, q_\lambda) + \mathrm{InSec}^{\mathrm{SM\text{-}DSPR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_8^{\mathsf{sl}},\, q_\lambda) \ +\ \varepsilon_{\mathrm{ds}}\,,
\end{aligned}
$$

where q_s^{sl} and q_s^{sf}, with $q_s^{\mathsf{sl}} + q_s^{\mathsf{sf}} = q_s$, are the numbers of stateless and stateful signing queries; q_1 counts the secret-key elements derived from SK.seed *on both paths; $q_1^{\mathsf{sf}}, \ldots, q_5^{\mathsf{sf}}$ and $q_{\lambda,0}, \ldots, q_{\lambda,5}$ are the query counts and collection budgets of Theorem 4.2 at $q_s = q_s^{\mathsf{sf}}$; $q_2^{\mathsf{sl}}, \ldots, q_8^{\mathsf{sl}}$ are the query counts of Theorem 4.3 at $q_s = q_s^{\mathsf{sl}}$; $\mathbf{PRF}_{\mathbf{msg}}$ is the combined randomizer function of Section 2.2; and $\varepsilon_{\mathrm{ds}}$ collects the domain-separation events of Assumption 4.1. The FXMSS instance carries the structure chosen by $\mathcal{A}$ (Definition 4.15), with L anchors and I internal nodes, so that $q_s^{\mathsf{sf}} \le L$. The terms on the right-hand side are evaluated at $q_s^{\mathsf{sf}} := L$, hence at $q_s := q_s^{\mathsf{sl}} + L$ in the $\mathbf{PRF}_{\mathbf{msg}}$ term; by Remark 4.6, applied to the stateful path, this bounds every adversary with $q_s^{\mathsf{sf}} \le L$. The resources of the reductions are*

$$
\begin{aligned}
\tilde{\xi}'_{\mathsf{sf}} &\le \xi + t_{\mathsf{sim}} + t^{\mathsf{sl}}_{\mathsf{sim}}, & \tilde{q}_h^{\mathsf{sf}} &\le q_h + 1, \\
\tilde{\xi}_{\mathsf{sf}} &\le \xi + V_{\mathsf{sf}}, & \tilde{q}_{\lambda,i} &\le q_{\lambda,i} + Q^{\mathsf{sl}}_{\Lambda}, \\
\tilde{\xi}_{\mathsf{sl}} &\le \xi + V_{\mathsf{sl}}, & q_\lambda &\le Q^{\mathsf{sl}}_{\Lambda} + L(l_{\mathsf{sf}} w + 1) + I + q_g, \\
\tilde{\xi}''_{\mathsf{sl}} &\le \xi + t^{\mathsf{sl}}_{\mathsf{sim}} + L(l_{\mathsf{sf}} w + 1) + I + q_g, & \tilde{q}_h^{\mathsf{sl}} &\le q_h + 1,
\end{aligned}
$$

with t_{sim} as in Theorem 4.2, $t^{\mathsf{sl}}_{\mathsf{sim}}$ and $Q^{\mathsf{sl}}_{\Lambda}$ as in Theorem 4.3 at $q_s = q_s^{\mathsf{sl}}$, q_g the grinding budget of Remark 4.4, and $\tilde{q}_h^{\mathsf{sf}}$ and $\tilde{q}_h^{\mathsf{sl}}$ counting the reductions' own evaluations of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, and V_{sf} and V_{sl} the tweakable-hash evaluations of the verification of the forgery (Lemma 5.1).

Each PRF replacement is charged exactly once: the stateless terms of Theorem 4.1 exclude the two PRF terms of Theorem 4.3, and the stateful terms are those of Theorem 4.2, whose experiment idealizes both the secret elements and the randomizers and which therefore has no PRF terms.

Only the stateful message hash is programmed in the proof, by the reprogramming hop (Claim 4.14). The stateless message hash is never programmed. Its security property enters through the ITSR notion (Definition 4.11), which is a standard-model notion. As a function, the stateless message hash is modeled by Assumption 4.1, like every other function, as an independent random function; the reductions of the stateful and stateless hops therefore simulate it as a random oracle for the forger's queries, Lemma 4.2 discharges their own evaluations of it, and this modeling supplies the query parameter $\tilde{q}_h^{\mathsf{sl}}$. The full proof is given in Section 4.8.

Theorem 4.2 (FXMSS EU-CMA Security)**.** *Let n, w, h_{sf} be the parameters of FXMSS (Section 2.5) and let l_{sf} be the number of chains per WOTS+C instance. Consider an adversary that*

commits to a structure with L anchors and I internal nodes (Section 2.5); the pairwise-distinct signing positions imply $q_s \le L$. The EU-CMA experiment idealizes key generation: the challenger samples every WOTS+C secret-key element uniformly at random, and hands the adversary the full SHRINCS public key, including $\mathsf{PK.sl}$ *(see Definition 4.16). Modeling the stateful message hash $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as a (quantum) random oracle and assuming Conjecture 3.1 (see Lemma 3.2), for any such adversary $\mathcal{A}$ running in time ξ that makes at most q_s signing queries and at most q_h random-oracle queries to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$:*

$$
\begin{aligned}
&\mathrm{InSec}^{\mathit{EU\text{-}CMA}}(\mathit{FXMSS};\, \xi,\, q_s) \le \\
&\quad \mathrm{InSec}^{\mathsf{SM\text{-}rTCR},1}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}},\, \mathbf{1};\, \tilde{\xi}',\, q_s,\, \tilde{q}_h) + \mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}};\, \tilde{\xi}',\, q_s,\, q_s + q_h + 1) \\
&\quad + \mathrm{InSec}^{\text{S-TCR(Prop)}}(\mathbf{H}_{\mathrm{grind}} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_s,\, q_{\lambda,0}) + w \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1}) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_2,\, q_{\lambda,2}) + \mathrm{InSec}^{\text{SM-PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1}) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_4,\, q_{\lambda,4}) + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_3,\, q_{\lambda,3}) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR-EXCL}}(\mathsf{Th}_{2 \cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_5,\, q_{\lambda,5})\,,
\end{aligned}
$$

The terms on the right-hand side are evaluated at $q_s := L$; by Remark 4.6 this bounds every adversary with $q_s \le L$. Here $q_1 \le L \cdot l_{\mathsf{sf}}$, $q_2 \le L \cdot l_{\mathsf{sf}} \cdot w$, $q_3 \le I$, $q_4 \le L$, and $q_5 \le L + I$. The collection budgets (Section 4.3) and the resources of the corresponding reductions satisfy

$$
\begin{aligned}
&q_{\lambda,0} \le L(l_{\mathsf{sf}} w + 1) + I + q_{\mathsf{kg}}, && q_{\lambda,1} \le L(l_{\mathsf{sf}} w + 1) + I + q_g + q_{\mathsf{kg}}, \\
&q_{\lambda,2} \le L + I + q_g + q_{\mathsf{kg}}, && q_{\lambda,3} \le L(l_{\mathsf{sf}} w + 1) + q_g + q_{\mathsf{kg}}, \\
&q_{\lambda,4} \le L l_{\mathsf{sf}} w + I + q_g + q_{\mathsf{kg}}, && q_{\lambda,5} \le L l_{\mathsf{sf}} w + q_g + q_{\mathsf{kg}}, \\
&\tilde{\xi} \le \xi + V_{\mathsf{sf}}, && \tilde{\xi}' \le \xi + t_{\mathsf{sim}} \quad \textit{with } t_{\mathsf{sim}} \le L(l_{\mathsf{sf}} w + 1) + I + q_g + q_{\mathsf{kg}}, \\
&\tilde{q}_h \le q_h + 1, &&
\end{aligned}
$$

where q_g is the grinding budget (Remark 4.4), $q_{\mathsf{kg}} \le 2^{19}$ counts the key generation of the stateless path, t_{sim} is the honest key generation and signing performed by the SM-rTCR *and reprogramming reductions, $V_{\mathsf{sf}} := l_{\mathsf{sf}}(w-1) - S_{wn} + 2 + h_{\mathsf{sf}}$ of the collection terms is the number of tweakable-hash evaluations of the verification of the forgery, split per function in Lemma 5.1, and $\tilde{q}_h$ counts the reductions' own evaluations of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$. The predicate* Prop *for the* S-TCR(Prop) *bound is the WOTS+C target-sum condition (Section 2.4): the encoded digest's l_{sf} base-w digits sum to the target S_{wn}. For the prescribed shapes, $I = L - 1$ with $L = \mathsf{depth} + 1$ (UXMSS) and $L = 2^{\mathsf{depth}}$ (BXMSS); the same relation holds for every valid structure, future shapes included (Section 2.5).*

Under the accounting of Section 4.3 the collection budgets $q_{\lambda,i}$ do not affect the achievable insecurities (Conjecture 4.1 and Corollary 4.1), and the collection terms do not degrade with the size of the structure. The sole L-dependent time is $\tilde{\xi}'$ in the SM-rTCR and reprogramming terms, whose reductions hold the public parameter from the start and thus pay the honest simulation cost t_{sim} in time. This cost is bounded by the feasibility of key generation and signing (cf. the magnitudes discussion in Section 4.3), and it consists of evaluations of the tweakable hash functions and $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$, modeled independently of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ (Assumption 4.1); by Lemma 4.2 these terms depend on $\tilde{\xi}'$ only through the query parameters $\tilde{q}_h$ and $q_s + q_h + 1$. Within Theorem 4.1 the SM-rTCR term and the seven collection terms enter at the resources of Claims 4.13 and 4.15, and the reprogramming term once, shared with the stateless route (Claim 4.14).

Remark 4.4. The grinding budget q_g counts the reductions' evaluations of $\mathbf{H}_{\mathrm{grind}}$ through the collection oracle when they reproduce the counter search of the $\mathcal{O}^{\mathsf{Prop}}$ oracle themselves. An attempt succeeds when the encoded digest meets the target sum S_{wn}; for our parameters ($l_{\mathsf{sf}} = 32$ base-w digits, $w = 16$, $S_{wn} = 240$) the success probability per attempt is approximately $1/65.7$, so the expectation of q_g is about $65.7\,L \approx 2^{6.04} L$ (Lemma 3.1); by Conjecture 3.1 every counter search stops below 2^c, so $q_g \le 2^c L$ in every execution, which is the budget used in the bounds.

The stateless path enters Theorem 4.1 through the hash-function terms of the SLH-DSA bound. We quote that bound, restated under the accounting of Section 4.3; its proof is in [HK22]. Theorem 4.1 uses its ten summands other than the two PRF terms, at the resources of the augmented reductions of Claim 4.16.

Theorem 4.3 (SLH-DSA EU-CMA Security [HK22])**.** *Let $n, w, h_{\mathsf{sl}}, d, m, t, k$ be the parameters of Section 2.7, with the values defined in Table 1, and let l_{sl} be the number of chains per WOTS-TW instance. For any adversary $\mathcal{A}$ running in time ξ that makes at most q_s signing queries and at most q_h queries to $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$:*

$$
\begin{aligned}
&\mathrm{InSec}^{\mathit{EU\text{-}CMA}}(\mathit{SLH\text{-}DSA};\, \xi,\, q_s) \le \\
&\quad \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF};\, \xi,\, q_1) + \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}};\, \xi,\, q_s) \\
&\quad + \mathrm{InSec}^{\mathrm{ITSR}}(\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}};\, \tilde{\xi}'_{\mathsf{sl}},\, q_s,\, \tilde{q}^{\mathsf{sl}}_h) + w \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_2,\, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_3,\, q_\lambda) + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_7,\, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_2,\, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_4,\, q_\lambda) + \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_k \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_5,\, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_{l_{\mathsf{sl}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_6,\, q_\lambda) \\
&\quad + 3 \cdot \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_8,\, q_\lambda) + \mathrm{InSec}^{\text{SM-DSPR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sl}},\, q_8,\, q_\lambda)\,,
\end{aligned}
$$

where $q_1 < 2^{h_{\mathsf{sl}}+1}(kt + l_{\mathsf{sl}})$, $q_2 < 2^{h_{\mathsf{sl}}+1} \cdot l_{\mathsf{sl}}$, $q_3 < 2^{h_{\mathsf{sl}}+1} \cdot l_{\mathsf{sl}} \cdot w$, $q_4 < 2^{h_{\mathsf{sl}}+1} k \cdot 2t$, $q_5 < 2^{h_{\mathsf{sl}}}$, $q_6 < 2^{h_{\mathsf{sl}}+1}$, $q_7 < 2^{h_{\mathsf{sl}}+1} kt$, and $q_8 < 2^{h_{\mathsf{sl}}} \cdot kt$. The collection budget and the resources of the reductions satisfy

$$
\begin{aligned}
&q_\lambda \le Q^{\mathsf{sl}}_\Lambda := 2^{h_{\mathsf{sl}}+1}(l_{\mathsf{sl}} w + 2) + 2^{h_{\mathsf{sl}}}(2kt + 1), && \tilde{\xi}_{\mathsf{sl}} \le \xi + V_{\mathsf{sl}}, \\
&\tilde{\xi}'_{\mathsf{sl}} \le \xi + t^{\mathsf{sl}}_{\mathsf{sim}} \quad \textit{with } t^{\mathsf{sl}}_{\mathsf{sim}} \le 2^{18.2} + q_s \cdot 2^{20.7}, && \tilde{q}^{\mathsf{sl}}_h \le q_h + 1,
\end{aligned}
$$

where $Q^{\mathsf{sl}}_\Lambda \approx 2^{62}$ is the size of the complete virtual structure, $t^{\mathsf{sl}}_{\mathsf{sim}}$ is the honest key generation and signing performed by the ITSR *reduction (Table 4), $V_{\mathsf{sl}} := k(a+1) + 1 + d\,(s_{\max} + h' + 1)$ of the collection terms is the number of tweakable-hash evaluations of the verification of the forgery, with $s_{\max}$ the largest chain work of a WOTS-TW encoding, split per function in Lemma 5.1, and $\tilde{q}^{\mathsf{sl}}_h$ counts the reduction's own evaluations of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$.*

The statement is the corrected SPHINCS$^+$ bound of [HK22], applied to the SLH-DSA algorithm set of Section 2.7 and restated under the accounting of Section 4.3, with the parameters of Table 1 and $q_s \le 2^{40}$.

The restatement follows from an inspection of the reductions behind the proof. They are two-stage adversaries that commit their challenge targets before the public parameter is revealed and then complete the remaining virtual structure, all WOTS-TW chains, XMSS nodes, and FORS trees. Every completion call uses an address distinct from every challenge address, so routing the completion through the collection oracle satisfies the reservation rule of Definition 4.12. The budget Q^{sl}_Λ is the exact structure count of the monolithic SM-TCR reduction of [BHK+19], dominated by the FORS trees; the per-term budgets are slightly smaller, since they exclude the challenged positions, and we do not track them because the budgets do not affect the instantiated bounds (Corollary 4.1). The ITSR reduction holds the public parameter from the start and performs the honest key generation and signing itself, which is the time $t^{\mathsf{sl}}_{\mathsf{sim}}$; the per-signature digests are served by its challenge oracle, so its own evaluations of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ are the forger's queries and one verification of the forgery. Besides this restatement of the resources, Theorem 4.3 has two SM-TCR terms for $\mathbf{F}$ where [HK22] has one (Remark 4.5).

Remark 4.5 (The merged SM-TCR term of [HK22])**.** [HK22, Theorem 3] merges the two SM-TCR terms for $\mathbf{F}$ of Theorem 4.3 into the single term $\mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \xi,\, q_3 + q_7)$. Write $\mathbf{HK}.x$ for the game GAME.x of their proof of Theorem 3, so $\mathbf{HK}.\mathbf{3}, \mathbf{HK}.\mathbf{3}.\mathbf{1}, \ldots, \mathbf{HK}.\mathbf{3}.\mathbf{4}, \mathbf{HK}.\mathbf{4}$ are

their GAME.3, GAME.3.1, ..., GAME.4. Their proof bounds the q_7 event, a second preimage for $\mathbf{F}$ in a FORS leaf, at the hop from **HK.3.3** to **HK.3.4**. It bounds the q_3 event, a collision in a WOTS-TW chain, inside the proof of [HK22, Theorem 2], after the SM-UD hop has replaced the chain values. InSec is a maximum over adversaries, so adding the two bounds into one term needs a single reduction that handles both events in the same game, and the proof does not give one. A reordered game sequence might give one; we have not checked this.

4.7 Proof of Theorem 4.2 (FXMSS Security)

We prove the EU-CMA security of FXMSS via a sequence of game hops. The new content of the proof is mostly limited to the handling of the flexible structure; every other hop follows the reductions of the SPHINCS$^+$ proofs [BHK+19, HK22], in the modular presentation of [BDG+23], together with the grinding (counter search) hop of [HKRY23, Theorems C.2 and D.1] for the WOTS+C counter selection. We spell out the full sequence, so that the proof can be read as a self-contained result. The proof exploits the unique-tweak property of the addressing scheme: even though the signer constructs the tree according to the structure committed by $\mathcal{A}$, every Th-call across the entire reduction uses a distinct address, so each call is a separate target for the underlying multi-target security notions.

The intuition is the standard XMSS argument: any valid forgery either (i) constitutes a WOTS+C forgery at some leaf (used or unused) or (ii) yields, somewhere along the authentication path, an internal node distinct from the one the honest signer would compute, which is a SM-TCR collision against $\mathbf{H} \in \mathsf{Th}_\lambda$. The flexible structure adds a third case, which earlier proofs, written for fixed trees, never had to address: a forgery may follow a structure that differs from the one the signer committed to. We handle this case using the SM-TCR-EXCL property (Definition 4.7).

We base the experiment on the SHRINCS game of Definition 4.15 with the key material idealized and the signing oracle restricted to the stateful path. One further ability is given to the adversary: while in SHRINCS the WOTS+C instances are consumed in a fixed order, the order is not important for the security, so we let each signing query name the leaf, and hence the WOTS+C instance, that signs the message. The adversary still receives the full SHRINCS public key, as the stateful path is attacked in the presence of the stateless one.

Definition 4.16 (FXMSS EU-CMA)**.** Let FXMSS be the stateful component of SHRINCS (Section 2.5). A two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ plays the game of Definition 4.15 with up to q_s signing queries, modified in three places.

1. *Idealized key generation.* On the descriptor Π submitted by $\mathcal{A}_1$ (for the prescribed shapes, $\Pi = \mathsf{structure} = (\mathsf{shape}, \mathsf{depth})$), which determines the anchor and internal-node sets, of sizes L and I (Section 2.5), the challenger samples $\mathsf{PK.seed} \leftarrow_\$ \{0,1\}^n$ and every WOTS+C secret-key element of the structure independently and uniformly at random from $\{0,1\}^n$ (one per chain per anchor, in place of the $\mathbf{PRF}(\mathsf{SK.seed}, \cdot)$ derivation of Section 2.4) and runs the FXMSS key-generation procedure on these elements, obtaining the root $\mathsf{PK.sf}$ of the constructed tree. It additionally runs the key generation of the stateless path (Section 3) on uniformly sampled secret elements under the same $\mathsf{PK.seed}$, obtaining $\mathsf{PK.sl}$, and hands $\mathcal{A}_2$ the full SHRINCS public key $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$. We denote this procedure by $(\mathsf{sk}, \mathsf{pk}) \leftarrow \widetilde{\mathsf{Kg}}(1^n, \Pi)$.

2. *Stateful path with adversarially chosen leaves.* A signing query is a pair $(M_i, (z_i^\star, i_i^\star))$ naming a leaf of the tree matching the submitted structure, in place of the path bit; the queried leaf identifiers are pairwise distinct, a query outside the structure is answered by $\perp$ (so without loss of generality every query names a leaf of the structure), and the oracle replies with $\sigma_i \leftarrow \mathrm{FXMSS.Sign}(\mathsf{sk}, M_i, (z_i^\star, i_i^\star))$, sampling the per-signature randomizer uniformly at random.

3. *Forgery.* $\mathcal{A}_2$ outputs $(M^*, (z^\star, i^\star), \sigma^*)$.

The FXMSS EU-CMA success probability of $\mathcal{A}$ is:

$$\mathrm{Succ}^{\text{FXMSS-EU-CMA}}_{\text{FXMSS},q_s}(\mathcal{A}) = \Pr\big[\Pi \leftarrow \mathcal{A}_1();\ (\mathsf{sk},\mathsf{pk}) \leftarrow \widetilde{\mathsf{Kg}}(1^n,\Pi);\ (M^*,\,(z^\star,i^\star),\,\sigma^*) \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{sk}}(\cdot,\cdot)}(\Pi,\,\mathsf{pk}) :\ \text{FXMSS.Verify}(\mathsf{pk},\,M^*,\,(z^\star,i^\star),\,\sigma^*) = 1 \ \wedge\ M^* \notin \{M_i\}_{i=1}^{q_s}\big].$$

Game sequence. We write $\mathrm{Succ}^{\mathbf{F}.i}(\mathcal{A})$ for the success probability of $\mathcal{A}$ in $\mathbf{F}.i$. Throughout the game sequence we denote the queried messages by $\widetilde{M}_i$ and the forgery's message by $\widetilde{M}^*$: the experiment signs arbitrary strings, and in the composition of Section 3 the submitted strings are exactly the bound messages of Algorithm 2, whose injective encoding maps distinct signed pairs, and a fresh forged pair, to pairwise distinct strings.

F.0 is the original FXMSS EU-CMA experiment of Definition 4.16, so $\mathrm{Succ}^{\mathbf{F}.0}(\mathcal{A}) = \mathrm{Succ}^{\text{FXMSS-EU-CMA}}_{\text{FXMSS},q_s}(\mathcal{A})$. The adversarial possibilities are split into two sets: either it produces a forgery for an index that is a leaf in the submitted structure, or it produces a forgery for an index that is not a leaf in the submitted structure. The second case has not been covered in previous proofs, since the verifier for plain XMSS verified the signature with the restrictions on the structure embedded into the algorithm. We will start by bounding the cases where the adversary matches the structure and then bound the forgery attempt using a different structure than the signer.

Remark 4.6 (All signing leaves are queried)**.** Without loss of generality we consider forgers that use all of their q_s signing queries, one per signing leaf of the committed structure. For an XMSS-style stateful scheme such as FXMSS the signing leaves are the anchors, so this means $q_s = L$. A forger $\mathcal{A}$ that stops after $q'_s < q_s$ queries is completed by a wrapper $\mathcal{A}'$ that runs $\mathcal{A}$, receives its forgery $(\widetilde{M}^*,\,(z^\star,i^\star),\,\sigma^*)$, and only then issues the remaining $q_s - q'_s$ signing queries on the still unqueried leaves, choosing messages that differ from $\widetilde{M}^*$, before outputting $\mathcal{A}$'s forgery unchanged. The forgery is valid for $\mathcal{A}'$ exactly when it is valid for $\mathcal{A}$. The additional queries add only their issuing cost to the forger's running time: the signing work is performed by the challenger (in the reductions below, by the simulator, where it is part of t_{sim}), and $\mathcal{A}'$ makes no further queries to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, so q_h is unchanged. The same wrapper applies to the stateful path of a SHRINCS forger (Definition 4.15). There the anchors are chosen by the signer through leafSelect, so the wrapper counts the stateful queries of $\mathcal{A}$ that are answered by a signature, answers by $\perp$ once all L anchors are used, and, after the output of $\mathcal{A}$, issues stateful queries on messages that differ from M^* until every anchor has been used, before outputting the forgery of $\mathcal{A}$ unchanged. The bound message $\widetilde{M}$ contains M (Algorithm 2), so these queries have bound messages different from $\widetilde{M}^*$. The stateless queries of $\mathcal{A}$ are forwarded unchanged. Hence, without loss of generality, a SHRINCS forger issues exactly L stateful signing queries, none of them answered by $\perp$ (under Conjecture 3.1 no counter search fails), and $q^{\mathsf{sf}}_s = L$ in every execution.

F.0 → F.1: Abort on $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ collision with the same index. F.1 is identical to **F.0** except the game additionally aborts if $\mathcal{A}$'s forgery $(\widetilde{M}^*,\,(z^\star,i^\star),\,\sigma^*)$ satisfies $(z^\star,i^\star) = (z^\star_j,i^\star_j)$ for some signing query j (a used-leaf forgery) and

$$\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R^*,\,\mathsf{PK.seed},\,\mathsf{PK.sf},\,\mathsf{idx}_j,\,\widetilde{M}^*) = \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_j,\,\mathsf{PK.seed},\,\mathsf{PK.sf},\,\mathsf{idx}_j,\,\widetilde{M}_j),$$

where R^* is the randomness extracted from σ^*, R_j is the randomness in σ_j, and $\mathsf{idx}_j = \mathsf{toByte}(z^\star_j,1)\|\mathsf{toByte}(i^\star_j,8)$ is the 9-byte leaf identifier shared by both calls (since the used-leaf condition forces $(z^\star,i^\star) = (z^\star_j,i^\star_j)$).

We bound this event with the SM-rTCR notion of Definition 4.9 applied to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$. View $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as a tweakable hash with public parameter $P = \mathsf{PK.seed}$, tweak $T = (\mathsf{PK.sf},\,\mathsf{idx})$, message $\mathsf{m} = M$, and randomness $\rho = R \in \{0,1\}^n$. The message slot within SHRINCS receives the bound message $\widetilde{M}$. Hence, the reduction must generate the stateless part upon receiving the parameter

Algorithm 4: Reduction $\mathcal{B}_0$ against SM-rTCR of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$

Input : Public parameter P and oracle $\mathcal{O}(\cdot,\cdot)$ of the SM-rTCR challenger ($K = 1$, $\mathsf{Prop} \equiv 1$); EU-CMA forger $\mathcal{A}$ against FXMSS
Output : An SM-rTCR collision $(j,\ M^*,\ R^*)$ or fail

Set $\mathsf{PK.seed} \leftarrow P$.
Run $\mathcal{A}$ to obtain the structure descriptor.
Sample every WOTS+C secret-key element of the structure uniformly at random and run the FXMSS key-generation procedure on these elements with $\mathsf{PK.seed}$ to obtain the root $\mathsf{PK.sf}$
`// matches the honest key-generation distribution of` **F.0**
Run Key Generation for the stateless part of the SHRINCS scheme using $\mathsf{PK.seed}$ to obtain $\mathsf{PK.sl}$. Return $\mathsf{pk} = (\mathsf{PK.seed},\ \mathsf{PK.sl},\ \mathsf{PK.sf})$ to $\mathcal{A}$.
for *each signing query* $(\widetilde{M}_i,\ (z_i^\star, j_i^\star))$ **do**
  Form the tweak $\mathsf{idx}_i = \mathsf{toByte}(z_i^\star, 1) \,\|\, \mathsf{toByte}(j_i^\star, 8)$.
  Query $\mathcal{O}((\mathsf{PK.sf},\ \mathsf{idx}_i),\ \widetilde{M}_i)$ to obtain $(y_i,\ R_i)$ `//` $y_i = \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_i, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i)$
  Run the rest of the FXMSS signing procedure on the digest y_i and reply with the signature σ_i, which contains R_i.
end
Obtain the forgery $(\widetilde{M}^*,\ (z^\star, i^\star),\ \sigma^*)$ from $\mathcal{A}$ and extract the randomness R^* from σ^*.
if *σ^* is valid, $(z^\star, i^\star)$ equals the queried leaf of some signing query j,* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R^*, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M}^*) = y_j$*, and* $(\widetilde{M}^*, R^*) \neq (\widetilde{M}_j, R_j)$ **then**
  return $(j,\ \widetilde{M}^*,\ R^*)$
else
  return *fail*
end

$P = \mathsf{PK.seed}$. All queries remain pairwise distinct through idx. Instantiate the predicate as $\mathsf{Prop} \equiv 1$ and the sampling budget as $K = 1$, so that on each query the oracle samples a single ρ and always inserts (T, m, ρ) into Q, which is exactly matching the way $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ is used during signing.

The reduction $\mathcal{B}_0$ is given in Algorithm 4; it simulates **F.0** for $\mathcal{A}$ perfectly while embedding the SM-rTCR challenges into the signing queries.

The signing queries provide at most q_s SM-rTCR challenges with pairwise distinct tweaks $(\mathsf{PK.sf}, \mathsf{idx}_i)$, and the abort event is precisely an SM-rTCR collision: a pair $(\widetilde{M}^*, R^*) \neq (\widetilde{M}_j, R_j)$ with matching $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$-output at the j-th tweak.

Claim 4.1. $\left|\mathrm{Succ}^{\mathbf{F.0}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.1}}(\mathcal{A})\right| \leq \mathrm{InSec}^{\mathsf{SM\text{-}rTCR},1}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}},\ \mathbf{1};\ \tilde{\xi}',\ q_s,\ \tilde{q}_h)$*, where* $\tilde{\xi}' \leq \xi + t_{\mathsf{sim}}$ *accounts for the honest key generation and signing performed by* $\mathcal{B}_0$*, and* $\tilde{q}_h \leq q_h + 1$ *counts* $\mathcal{B}_0$*'s evaluations of* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$*: the forger's queries and one verification of the forgery.*

F.1 → F.2: Adaptive reprogramming of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$. In this game-hop we model $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as a random oracle. In **F.1** signing of $\widetilde{M}_i$ at index $(z_i^\star, j_i^\star)$ proceeds by sampling randomness $R_i \leftarrow_{\$} \mathcal{R} = \{0,1\}^n$, computing the $2n$-bit digest $y_i \leftarrow \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_i, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i)$, and then invoking WOTS+C on y_i. In **F.2** the challenger pre-samples, at setup, q_s uniformly random digests $y_1, \ldots, y_{q_s} \leftarrow_{\$} \{0,1\}^{2n}$, one per signing leaf of the committed structure, indexed by a canonical enumeration of the leaves ($q_s = L$ by Remark 4.6), and assigns y_i to the i-th leaf. On the query naming the leaf $(z_i^\star, j_i^\star)$, and message $\widetilde{M}_i$ the challenger samples $R_i \leftarrow_{\$} \mathcal{R}$, *reprograms* the random oracle so that $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_i, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i) := y_i$, uses y_i as the digest, and includes R_i in the signature. The two games are the two sides of the experiment of Definition C.2, a variant with distinct inputs, defined in this document, of the adaptive-reprogramming experiment of [GHHM21]: the leaf identifiers of the signing queries are pairwise distinct, so no input is reprogrammed twice. The distinguishing advantage between **F.1** and **F.2** is at most $\mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}};\ \tilde{\xi}',\ q_s,\ q_s + q_h + 1)$, where q_h is the number of $\mathcal{A}$'s queries to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$

Algorithm 5: Distinguisher $\mathcal{B}_1$ against adaptive reprogramming of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$

Input : Random oracle $\mathcal{O}_b(\cdot,\cdot)$ and oracle $\mathsf{Reprogram}(\cdot)$ of the $\mathbf{DI\text{-}Repro}_b$ challenger (Definition C.2) with $X_1 = \mathcal{R}$; EU-CMA forger $\mathcal{A}$ against FXMSS

Output : A bit b'

Sample $\mathsf{PK.seed} \leftarrow_{\$} \{0,1\}^n$.
Run $\mathcal{A}$ to obtain the structure descriptor.
Sample every WOTS+C secret-key element of the structure uniformly at random and run the FXMSS key-generation procedure on these elements with $\mathsf{PK.seed}$ to obtain the root $\mathsf{PK.sf}$
`// matches the honest key-generation distribution of` $\mathbf{F.1}$
Run Key Generation for the stateless part of the SHRINCS scheme using $\mathsf{PK.seed}$ to obtain $\mathsf{PK.sl}$.
Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ with $\mathcal{O}_b$.
for *each signing query* $(\widetilde{M}_i, (z_i^\star, j_i^\star))$ **do**
  Form the tweak $\mathsf{idx}_i = \mathsf{toByte}(z_i^\star, 1) \,\|\, \mathsf{toByte}(j_i^\star, 8)$.
  Call $R_i \leftarrow \mathsf{Reprogram}(\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i)$.
  Query $y_i \leftarrow \mathcal{O}_b(R_i, (\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i))$ // $y_i = \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_i, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i)$
  Run the rest of the FXMSS signing procedure on the digest y_i and reply with the signature σ_i, which contains R_i.
end
Obtain the forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ from $\mathcal{A}$.
if σ^* *is valid,* $\widetilde{M}^* \neq \widetilde{M}_j$, $j \in [1; q_s]$, *and the abort condition of* $\mathbf{F.1}$ *does not hold* **then**
  return $b' = 1$
else
  return $b' = 0$
end

besides signature queries. After $\mathbf{F.2}$ the digests $\{y_i\}_{i=1}^{q_s}$ are known to the simulator at setup time: the adaptive choice of messages by $\mathcal{A}$ is decoupled from the digests fed to WOTS+C.

The distinguisher $\mathcal{B}_1$ is given in Algorithm 5. It plays the reprogramming game with distinct inputs of Definition C.2 instantiated with $X_1 = \mathcal{R}$, the randomness space of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, and X_2 the set of the remaining inputs $(\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}, \widetilde{M})$. Every signing query is answered through the $\mathsf{Reprogram}$ oracle.The guard of Definition C.2 never fires: the signing queries name pairwise distinct leaf identifiers, all of them leaves of the structure, as Definition 4.16 assumes without loss of generality, so the inputs $(\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i)$ of the calls are pairwise distinct and no call is answered by $\perp$. If the challenge bit is $b = 0$, the returned randomness R_i is uniform and $\mathcal{O}_0$ is left untouched, so the digest y_i is the honest hash value and $\mathcal{A}$'s view is exactly $\mathbf{F.1}$. If $b = 1$, then $\mathcal{O}_1$ is reprogrammed to a fresh uniform digest at the returned randomness, so $\mathcal{A}$'s view is exactly $\mathbf{F.2}$. Since $\mathcal{B}_1$ outputs 1 precisely when $\mathcal{A}$ wins the simulated game, its distinguishing advantage equals $\left|\mathrm{Succ}^{\mathbf{F.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.2}}(\mathcal{A})\right|$.

Claim 4.2. *Modeling* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ *as a random oracle,*

$$\left|\mathrm{Succ}^{\mathbf{F.1}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.2}}(\mathcal{A})\right| \;\leq\; \mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}; \tilde{\xi}', q_s, q_s + q_h + 1),$$

where q_h *denotes the number of* $\mathcal{A}$*'s queries to* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ *besides the signature queries, and* $\tilde{\xi}' \leq \xi + t_{\mathsf{sim}}$ *accounts for the honest key generation and signing performed by the reduction.*

Concrete bounds for this term are collected in Corollary C.3. That corollary also bounds the advantage of $\mathcal{B}_1$ directly, using the query pattern of Algorithm 5, which its proof counts.

Remark 4.7. Modeling $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as a random oracle independent of the tweakable-hash calls and of the remaining keyed functions is an instance of Assumption 4.1; the layout analysis behind it is recorded in the domain-separation discussion of Section 2.2 and in Lemma 4.3.

F.2 → F.3: Abort on $\mathbf{H}_{\text{grind}}$ collision with the same index. In **F.2** the WOTS+C instance at the i-th queried index internally grinds a counter c_i until the encoded digest $d_i = \mathbf{H}_{\text{grind}}(\mathsf{PK.seed}, \mathbf{ADRS}_i^{\mathsf{g}}[0{:}10], y_i, c_i)$ satisfies the WOTS+C target-sum predicate Prop (Section 2.4); the tweak is the grinding address, named by its 10-byte prefix (Section 2.2), which uniquely identifies the queried index via its height, index and type fields. In **F.3** the challenger instead *queries* the external S-TCR(Prop) oracle $\mathcal{O}^{\mathsf{Prop}}$ on $(\mathbf{ADRS}_i^{\mathsf{g}}[0{:}10], y_i)$ (which is well-defined because the digests y_i are now known at setup), and adopts the returned (d_i, c_i) as the encoded digest and witness counter, and it aborts if $\mathcal{A}$'s forgery is at the leaf of some signing query j and carries a message digest $y^* \neq y_j$ and a counter c^* with $\mathbf{H}_{\text{grind}}(\mathsf{PK.seed}, \mathbf{ADRS}_j^{\mathsf{g}}[0{:}10], y^*, c^*) = d_j$, where $y^* = \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R^*, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M^*})$ and R^*, c^* are read from the forgery. On the abort event, $(y^*, c^*) \neq (y_j, c_j)$ since $y^* \neq y_j$, and, since the tweaks $\mathbf{ADRS}_i^{\mathsf{g}}[0{:}10]$ are pairwise distinct, the forgery contains an S-TCR(Prop) collision at the j-th tweak: the two pairs satisfy $\mathbf{H}_{\text{grind}}(\mathsf{PK.seed}, \mathbf{ADRS}_j^{\mathsf{g}}[0{:}10], y^*, c^*) = \mathbf{H}_{\text{grind}}(\mathsf{PK.seed}, \mathbf{ADRS}_j^{\mathsf{g}}[0{:}10], y_j, c_j)$ (see [HKRY23, Theorem D.1]).

The reduction $\mathcal{B}_2$ is given in Algorithm 6. The two-stage structure of Definition 4.8 requires all targets to be committed before the public parameter is revealed, and after **F.2** this is possible: the digests y_i are sampled at setup, independently of $\mathcal{A}$'s messages, so $\mathcal{B}_2$ queries all q_s targets first, builds both trees through the collection oracle Th_λ (Section 4.3), and only then obtains $P = \mathsf{PK.seed}$.

The oracle queries commit q_s targets with pairwise-distinct tweaks, the simulation of **F.2** is perfect, and the abort event of **F.3** is precisely an S-TCR(Prop) collision: a pair (y^*, c^*) with $y^* \neq y_j$ and matching $\mathbf{H}_{\text{grind}}$-output at the j-th tweak.

Remark 4.8. The scheme caps the counter search at $2^c = 2^{16}$ attempts and returns $\perp$ on exhaustion, while the S-TCR(Prop) oracle of Definition 4.8 searches unboundedly for the smallest valid counter. On non-$\perp$ signatures the two coincide (the scheme also takes the smallest valid counter), and under Conjecture 3.1 no signature is $\perp$: the smallest valid counter is always below 2^c, so the oracle and the signer agree on every input.

Claim 4.3. $\left|\mathrm{Succ}^{\mathbf{F.2}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.3}}(\mathcal{A})\right| \leq \mathrm{InSec}^{\text{S-TCR(Prop)}}(\mathbf{H}_{\text{grind}} \in \mathsf{Th}_\lambda; \tilde{\xi}, q_s, q_{\lambda,0})$, *where* Prop *is the WOTS+C target-sum predicate* $\sum_{i=0}^{l_{\mathsf{sf}}-1} a_i = S_{wn}$, $q_{\lambda,0} \leq L(l_{\mathsf{sf}} w + 1) + I + q_{\mathsf{kg}}$, *and* $\tilde{\xi} \leq \xi$ *is the time of the reduction.*

F.3 → F.4: SM-UD chain-value replacement. After **F.3** every WOTS+C secret-key element is uniform (by the definition of the experiment) and the encoded digests d_j together with their witness counters c_j are pinned by the $\mathcal{O}^{\mathsf{Prop}}$ oracle. Recall that, without loss of generality, the adversary queries all L signing leaves (Remark 4.6); we enumerate the anchors canonically as $j = 1, \ldots, L$, matching the assignment of the digests y_j in **F.2**. We write $\mathbf{ADRS}_j^{\mathsf{h}}$ for the type-`SF_WOTS_C_HASH` address of the WOTS+C instance at the j-th anchor and $(a_{j,0}, \ldots, a_{j,l_{\mathsf{sf}}-1})$ for the base-w digits of d_j. The response to the j-th signing query reveals the chain values

$$\sigma_{j,i} = c^{0,a_{j,i}}(\mathsf{sk}_{j,i}, i, \mathsf{PK.seed}, \mathbf{ADRS}_j^{\mathsf{h}}), \quad i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}.$$

F.4 changes how these blocks are produced. For every chain the challenger samples a fresh value $u_{j,i} \leftarrow_{\$} \{0,1\}^n$ and sets

$$\sigma_{j,i} = \begin{cases} c^{a_{j,i}-1,1}(u_{j,i}, i, \mathsf{PK.seed}, \mathbf{ADRS}_j^{\mathsf{h}}), & \text{if } a_{j,i} > 0, \\ u_{j,i}, & \text{if } a_{j,i} = 0, \end{cases}$$

i.e., every signature block with a positive digit is the hash of a fresh uniform value placed at position $a_{j,i} - 1$. The public-key elements are completed from the blocks as in verification, $\mathsf{pk}_{j,i} = c^{a_{j,i}, w-1-a_{j,i}}(\sigma_{j,i}, i, \mathsf{PK.seed}, \mathbf{ADRS}_j^{\mathsf{h}})$, and the rest of the experiment is unchanged. The reason we consider this game is that a hash of a secret uniform value is exactly the shape of an

Algorithm 6: Reduction $\mathcal{B}_2$ against S-TCR(Prop) of $\mathbf{H}_{\text{grind}}$

Input : Oracle $\mathcal{O}^{\mathsf{Prop}}(\cdot,\cdot)$ of the S-TCR(Prop) challenger (Definition 4.8) for $\mathbf{H}_{\text{grind}} \in \mathsf{Th}_\lambda$, with Prop the WOTS+C target-sum predicate, and collection oracle Th_λ (Section 4.3); EU-CMA forger $\mathcal{A}$ against FXMSS

Output : An S-TCR(Prop) collision $(j,\, y^*,\, c^*)$ or fail

Run $\mathcal{A}$ to obtain the structure descriptor; enumerate the signing leaves of the structure as $i = 1, \ldots, q_s$ in a canonical order.

Sample the digests $y_1, \ldots, y_{q_s} \leftarrow_{\$} \{0,1\}^{2n}$ and assign y_i to the i-th signing leaf `// matches F.2`

for $i = 1, \ldots, q_s$ **do**

  Query $(d_i,\, c_i) \leftarrow \mathcal{O}^{\mathsf{Prop}}(\mathbf{ADRS}_i^{\mathsf{g}}[0{:}10],\, y_i)$, where $\mathbf{ADRS}_i^{\mathsf{g}}[0{:}10]$ is the grinding-address prefix of the i-th signing leaf, determined by the structure `//` $d_i = \mathbf{H}_{\text{grind}}(P, \mathbf{ADRS}_i^{\mathsf{g}}[0{:}10], y_i, c_i)$ `satisfies Prop, with` c_i `the smallest such counter`

end

Sample every WOTS+C secret-key element of the structure uniformly at random and run the FXMSS key-generation procedure on these elements, evaluating every hash call through the collection oracle Th_λ, to obtain the root $\mathsf{PK.sf}$.

Run Key Generation for the stateless part of the SHRINCS scheme, evaluating every hash call through Th_λ, to obtain $\mathsf{PK.sl}$.

Signal the S-TCR(Prop) challenger to proceed to the second stage and receive the public parameter P; set $\mathsf{PK.seed} \leftarrow P$.

Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ by simulating the random oracle.

for *each signing query* $(\widetilde{M}_i,\, (z_i^\star, j_i^\star))$*, where* i *is the canonical index of the queried leaf* **do**

  Form the tweak $\mathsf{idx}_i = \mathsf{toByte}(z_i^\star, 1) \,\|\, \mathsf{toByte}(j_i^\star, 8)$.

  Sample $R_i \leftarrow_{\$} \mathcal{R}$ and reprogram $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}(R_i,\, \mathsf{PK.seed},\, \mathsf{PK.sf},\, \mathsf{idx}_i,\, \widetilde{M}_i) := y_i$ as in **F.2**.

  Run the rest of the FXMSS signing procedure with the encoded digest d_i and counter c_i in place of the counter search, and reply with the signature σ_i.

end

Obtain the forgery $(\widetilde{M}^*,\, (z^\star, i^\star),\, \sigma^*)$ from $\mathcal{A}$ and extract the randomness R^* and the counter c^* from σ^*.

if $(z^\star, i^\star)$ *equals the queried leaf of some signing query* j *and* $y^* := \mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}(R^*,\, \mathsf{PK.seed},\, \mathsf{PK.sf},\, \mathsf{idx}_j,\, \widetilde{M}^*) \neq y_j$ **then**

  return $(j,\, y^*,\, c^*)$

else

  return *fail*

end

SM-PRE challenge (Definition 4.5): the hop **F.5** → **F.6** places preimage challenges at precisely these positions. Such challenges are hashes of random elements rather than hashes of chain values. These two distributions may be different enough for the adversary to notice and break the reduction. So we bound this change with the undetectability property.

To compare the two games, we describe them through the joint distribution of the chain values one step before the signature blocks. For every chain with $a_{j,i} > 0$ let $\phi_{j,i}$ denote the value at position $a_{j,i} - 1$, so that $\sigma_{j,i} = c^{a_{j,i}-1,\,1}(\phi_{j,i},\, i,\, \mathsf{PK.seed},\, \mathbf{ADRS}_j^{\mathsf{h}})$ in either game; blocks with $a_{j,i} = 0$ are fresh uniform values in both games and play no role in the comparison. In **F.3** the tuple $\{\phi_{j,i}\}$ follows the distribution

$$D_{Kg} = \left\{\phi_{j,i} = c^{0,\,a_{j,i}-1}(u_{j,i},\, i,\, \mathsf{PK.seed},\, \mathbf{ADRS}_j^{\mathsf{h}})\right\}, \quad u_{j,i} \leftarrow_{\$} \{0,1\}^n,$$

where the uniform $u_{j,i}$ plays the role of the secret-key element $\mathsf{sk}_{j,i}$, while in **F.4** it follows $D_0 = \{\phi_{j,i} = u_{j,i}\}$ with $u_{j,i} \leftarrow_{\$} \{0,1\}^n$. We can hence deduce that

$$\left|\mathrm{Succ}^{\mathbf{F.3}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.4}}(\mathcal{A})\right| \leq \left|\mathrm{Succ}^{D_{Kg}}(\mathcal{A}) - \mathrm{Succ}^{D_0}(\mathcal{A})\right|.$$

We need to bound $\left|\mathrm{Succ}^{D_{Kg}}(\mathcal{A}) - \mathrm{Succ}^{D_0}(\mathcal{A})\right|$. To do so we use the hybrid argument of [HK22, Theorem 1, Claim 2], we interpolate over the chain depth: for $\gamma \in \{0, \ldots, w-2\}$ define

$$H_\gamma = \left\{\phi_{j,i} = c^{\gamma, a_{j,i}-1-\gamma}(u_{j,i},\, i,\, \mathsf{PK.seed},\, \mathbf{ADRS}^{\mathsf{h}}_j)\right\}, \quad u_{j,i} \leftarrow_{\$} \{0,1\}^n,$$

recalling that the chaining function returns its input unchanged when the iteration count is non-positive (Section 2.3). In these hybrids we have uniform values up to position γ and evaluation of the chaining function from $\gamma+1$ till the end of the chain. Then $H_0 = D_{Kg}$, and $H_{w-2} = D_0$. We have $\mathbf{F.3}_0 = \mathbf{F.3}$ and $\mathbf{F.3}_{w-2} = \mathbf{F.4}$, and according to the hybrid argument we get:

$$\left|\mathrm{Succ}^{\mathbf{F.3}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.4}}(\mathcal{A})\right| \le \sum_{\gamma=0}^{w-3} \left|\mathrm{Succ}^{\mathbf{F.3}_\gamma}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.3}_{\gamma+1}}(\mathcal{A})\right| \le w \cdot \max_\gamma \left|\mathrm{Succ}^{\mathbf{F.3}_\gamma}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.3}_{\gamma+1}}(\mathcal{A})\right|.$$

We now need to show how to use the SM-UD oracle to construct the hybrid H_γ or $H_{\gamma+1}$. The idea is that hybrid H_γ has all the uniform values till γ, while $H_{\gamma+1}$ till $\gamma+1$. The difference is only in the position $\gamma+1$, where it is either a uniformly random value, or a hash of a uniformly random value. Hence to construct one of the hybrids we can query the SM-UD oracle with the tweak corresponding to the chain i and position $\gamma+1$. If SM-UD oracle returns uniformly random values, we get the distribution of H_γ, if it returns the hash of a uniformly random value we get the distribution of $H_{\gamma+1}$. If there is a significant change is success then we can build a reduction that breaks SM-UD. An example for a single WOTS instance can be found in Fig. 3.

The reduction $\mathcal{B}_3$ is given in Algorithm 7; it takes the transition index γ as input and plays the SM-UD experiment against $\mathbf{F} \in \mathsf{Th}_\lambda$. Depending on the challenge bit b, the oracle response for a position-γ tweak is either the hash of a secret uniform value, i.e., the position-$(\gamma+1)$ value of a chain whose position-γ value is uniform, or a fresh uniform value. $\mathcal{B}_3$ grinds the digests first and places SM-UD challenges only at the position-γ tweaks of the chains whose digit satisfies $a_{j,i} - 1 > \gamma$, since these are the only chains whose construction depends on the challenge bit. All remaining first-stage evaluations (the counter search, the chain completions at positions above γ, the public-key compressions, the tree nodes, and the stateless key generation) run through the collection oracle Th_λ, at tweaks disjoint from the challenge tweaks.

Since $\mathcal{B}_3$ outputs 1 exactly when $\mathcal{A}$ wins the simulated game, for every γ we obtain

$$\left|\mathrm{Succ}^{\mathbf{F.3}_\gamma}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.3}_{\gamma+1}}(\mathcal{A})\right| = \mathsf{Adv}^{\text{SM-UD}}_{\mathbf{F}\in\mathsf{Th}_\lambda,\, q_1}(\mathcal{B}_3) \le \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1}).$$

Summing over the $w-2$ transitions bounds the distance between **F.3** and **F.4** by $(w-2) \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1})$, which we relax to $w \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1})$ in the claim below.

Claim 4.4. $\left|\mathrm{Succ}^{\mathbf{F.3}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.4}}(\mathcal{A})\right| \le w \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1})$, *where* $q_1 \le L \cdot l_{\mathsf{sf}}$, $q_{\lambda,1} \le L(l_{\mathsf{sf}} w + 1) + I + q_g + q_{\mathsf{kg}}$, *and* $\tilde{\xi} \le \xi + V_{\mathsf{sf}}$ *is the time of the reduction.*

F.4 → F.5: Abort on chain collisions. **F.5** is identical to **F.4** except the game aborts whenever $\mathcal{A}$'s forgery reaches a chain value via an input distinct from the honest one but with matching **F**-output. If the uncompressed public key of the WOTS+C instance is the same, then the adversary has two possibilities for a forgery. Due to the target-sum constraint, there must be at least one chain longer than in the queried signature. There are two options for building such a chain: either it matches the values that the honest signer uses for that chain, or at least some of the values differ. Since the uncompressed public key element is the same in both cases, the dishonest and honest chains must collide at some level, which constitutes a chain-level collision against $\mathbf{F} \in \mathsf{Th}_\lambda$. One can build a reduction that uses a SM-TCR oracle to construct all of the WOTS+C instances (see [HK22, Theorem 1, Claim 3]). Any collision within a WOTS+C chain is a solution for some SM-TCR challenge. Any such event extracts an SM-TCR collision at a chain hash position; counting all chain hash positions across all anchors of the structure gives $q_2 \le L \cdot l_{\mathsf{sf}} \cdot w$ targets.

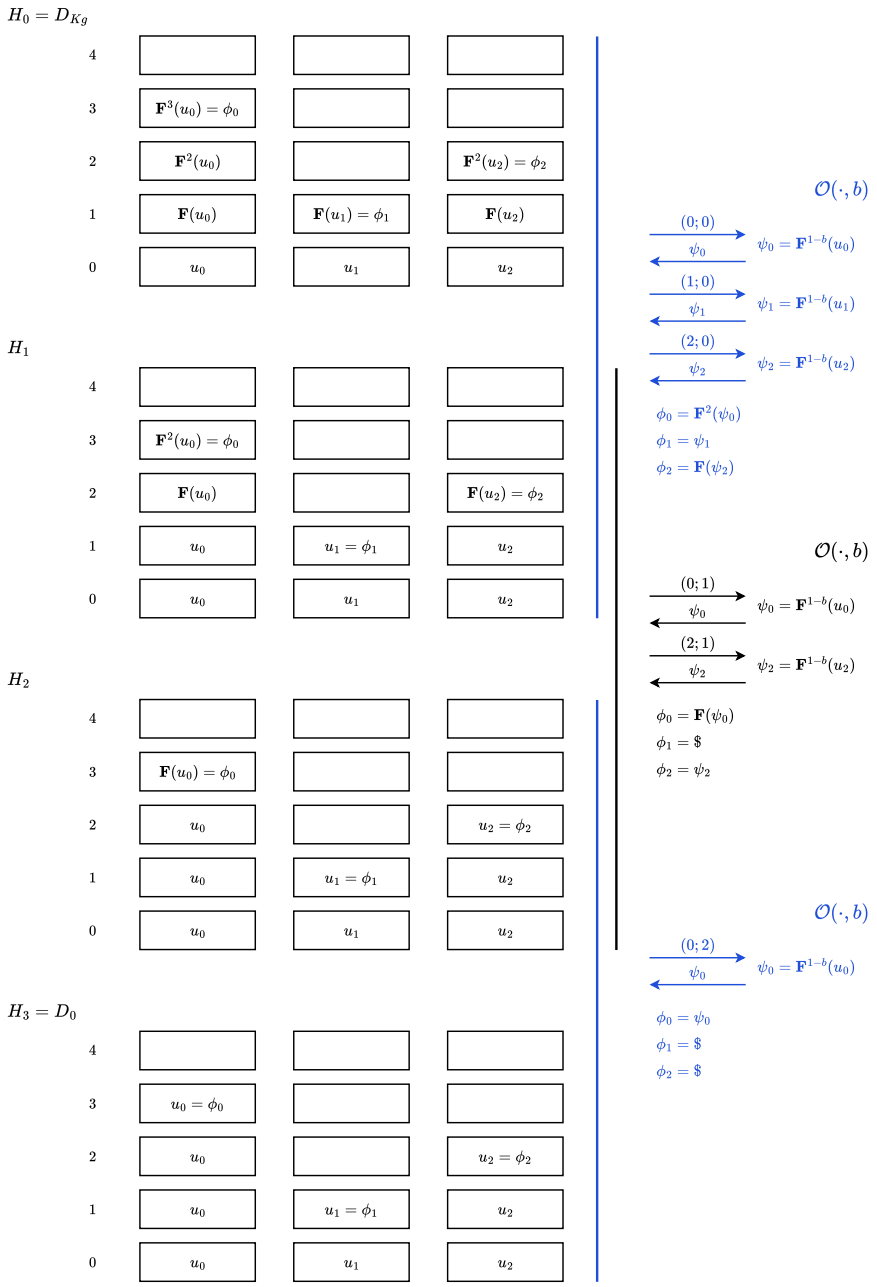


Figure 3: An example of SM-UD hybrid steps.

The reduction $\mathcal{B}_4$ is given in Algorithm 8. The chains are built through the challenge oracle itself, feeding each response as the message of the next query; all remaining stage-one evaluations run through the collection oracle Th_λ (Section 4.3). In particular, since the digits $a_{j,i}$ determine where the chains start, $\mathcal{B}_4$ grinds the counters before its chain queries by viewing $\mathbf{H}_{\text{grind}}$ as a member of the collection (with its address of type `SF_WOTS_C_GRIND` as its tweak, Section 4.3). The tweak-reservation rule is respected: none of the collection-oracle tweaks coincide with the type-`SF_WOTS_C_HASH` chain addresses used for the challenges.

Claim 4.5. $\left|\mathrm{Succ}^{\mathbf{F}.4}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F}.5}(\mathcal{A})\right| \leq \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}, q_2, q_{\lambda,2})$, *where* $q_2 \leq L \cdot l_{\mathsf{sf}} \cdot w$, $q_{\lambda,2} \leq L + I + q_g + q_{\mathsf{kg}}$, *and* $\tilde{\xi} \leq \xi + V_{\mathsf{sf}}$ *is the time of the reduction.*

F.5 → F.6: Abort on chain inversions. After **F.5** the only way to produce a WOTS+C forgery on a used leaf without changing the uncompressed public key of some WOTS+C instance is to invert a revealed chain output (i.e., move *backwards* along some chain). Note that the

switch in **F.4** arranged for every signature block in every WOTS+C instance to be the result of hashing a uniformly random value. This matches the SM-PRE game, so we use the SM-PRE oracle to compute those challenges and plug them into the WOTS+C chains. **F.6** aborts if a preimage for one of the challenges is revealed, reducing to SM-PRE of $\mathbf{F} \in \mathsf{Th}_\lambda$ with $q_1 \le L \cdot l_{\mathsf{sf}}$ targets (see [HK22, Theorem 1, Claim 4]).

The reduction $\mathcal{B}_5$ is given in Algorithm 9. Every signature block with a positive digit is obtained directly from the SM-PRE challenge oracle at the tweak $\mathbf{ADRS}_{j,i}^{a_{j,i}-1}$, so the oracle's hidden uniform preimage plays the role of the fresh value of **F.4**; the counter search and all completion work (the chain steps above the challenge positions, the public-key compressions, the tree nodes, and the stateless key generation) run through the collection oracle Th_λ as in $\mathcal{B}_4$. The tweak-reservation rule is again respected: the challenge tweaks sit at the hash addresses $a_{j,i} - 1$, which no collection query touches.

Claim 4.6. $\left|\mathrm{Succ}^{\mathbf{F.5}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.6}}(\mathcal{A})\right| \le \mathrm{InSec}^{\text{SM-PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}, q_1, q_{\lambda,1})$, *where* $q_{\lambda,1} \le L(l_{\mathsf{sf}} w + 1) + I + q_g + q_{\mathsf{kg}}$ *and* $\tilde{\xi} \le \xi + V_{\mathsf{sf}}$ *is the time of the reduction.*

F.6 → F.7: Abort on WOTS+C public-key compression collisions. Up to **F.6** we have ruled out every forgery on a used leaf in which the recovered *uncompressed* WOTS+C public key (the tuple of l_{sf} chain endpoints) equals the honest one: chain collisions are excluded by **F.5** and chain inversions by **F.6**. The remaining attack surface at a used leaf is therefore a forgery whose chain endpoints differ from the honest endpoints at some position but which, after $\mathsf{Th}_{l_{\mathsf{sf}}}$-compression, still yields the honest WOTS+C compressed public key pk. **F.7** aborts whenever such an event occurs. This is an SM-TCR collision against $\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda$ at the leaf's WOTS+C public-key-compression address: the honest uncompressed tuple is the embedded target, and $\mathcal{A}$'s forgery supplies a distinct tuple with matching $\mathsf{Th}_{l_{\mathsf{sf}}}$-output. We use Th_λ (see Section 4.3) to build each WOTS+C instance and then use the SM-TCR oracle to compress the public key elements. There is one $\mathsf{Th}_{l_{\mathsf{sf}}}$ call per WOTS+C instance in the structure, giving $q_4 \le L$ targets, all with pairwise-distinct addresses (unique tweak).

The reduction $\mathcal{B}_6$ is given in Algorithm 10. It builds every WOTS+C instance through the collection oracle Th_λ (the counter search, the chains started from the fresh uniform values of **F.4**, the internal tree nodes, and the stateless key generation), and reserves the challenge oracle for the L public-key compressions, one per WOTS+C instance. The tweak-reservation rule is respected: the challenge tweaks are the type-`SF_WOTS_C_PK` compression addresses, which no query to Th_λ touches.

Claim 4.7. $\left|\mathrm{Succ}^{\mathbf{F.6}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.7}}(\mathcal{A})\right| \le \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda; \tilde{\xi}, q_4, q_{\lambda,4})$, *where* $q_4 \le L$, $q_{\lambda,4} \le L l_{\mathsf{sf}} w + I + q_g + q_{\mathsf{kg}}$, *and* $\tilde{\xi} \le \xi + V_{\mathsf{sf}}$ *is the time of the reduction.*

F.7 → F.8: Abort on tree-node collisions. After **F.7**, no WOTS+C forgery of any honest instance is possible at any leaf. What is still possible for the adversary is to use a different WOTS+C instance in place of the honest one. The new instance hence has a different compressed public key. As the root of the tree must be the same as the honest one (the verification checks whether $\mathsf{PK.sf}^* = \mathsf{PK.sf}$), by a pigeonhole argument there must be a collision somewhere in the tree. **F.8** aborts on this event. The reduction embeds the SM-TCR challenge across all internal tree-node computations and builds the rest with Th_λ oracle: there are $q_3 \le I$ such targets, one per internal node of the structure.

The reduction $\mathcal{B}_7$ is given in Algorithm 11. It builds the WOTS+C instances and their compressed public keys through the collection oracle Th_λ and reserves the challenge oracle for the internal nodes: the tree is assembled bottom-up, each internal node being one challenge query on the concatenation of its two children at the node's address, so the root $\mathsf{PK.sf}$ is the output of the last challenge query. The tweak-reservation rule is respected: the challenge tweaks are the type-`SF_FXMSS_TREE` internal-node addresses, which no collection query touches.

Claim 4.8. $\left|\mathrm{Succ}^{\mathbf{F.7}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.8}}(\mathcal{A})\right| \leq \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda; \tilde{\xi}, q_3, q_{\lambda,3})$, *where* $q_3 \leq I$, $q_{\lambda,3} \leq L(l_{\mathsf{sf}}w + 1) + q_g + q_{\mathsf{kg}}$, *and* $\tilde{\xi} \leq \xi + V_{\mathsf{sf}}$ *is the time of the reduction.*

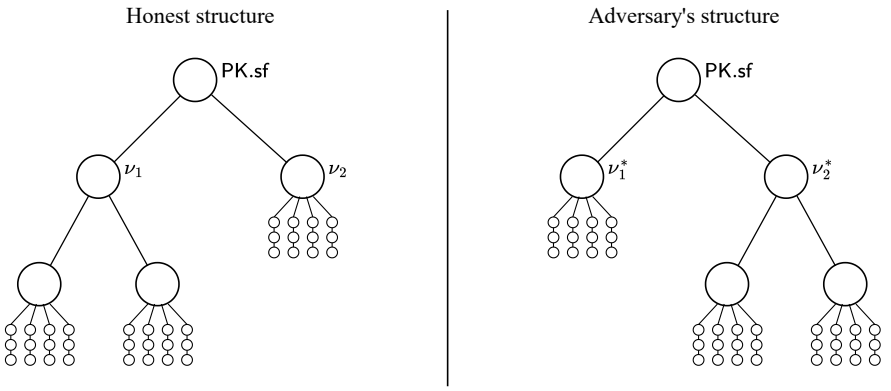
F.8 → F.9: Abort on tree-node collisions across mismatched structures. The hops up to **F.8** all assume that the adversary's verification path commits to the *same* imaginary-tree structure as the honest signer: every leaf position is processed via a WOTS+C public-key-compression call, every internal-node position via a tree-node hash, and the addresses used at each position match those used by the honest signer. The FXMSS verifier, however, does not test the shape of the imaginary tree directly. The verifier only recomputes the root from the supplied authentication path and accepts whenever the result matches $\mathsf{PK.sf}$. This leaves one final attack route: $\mathcal{A}$ can produce a forgery whose authentication path *deviates* from the honest structure at some node, either by placing a WOTS+C instance where the honest signer would compute an internal-node hash, or by placing an internal-node hash where the honest signer would compute a WOTS+C public-key compression (see Fig. 4). In case of such mismatch, the adversary's hash call uses an address whose *type byte* differs from the type byte the honest signer uses at the same node identifier (z, i), so a collision at this position is not captured by plain SM-TCR, since the tweaks on the two sides of the collision are not equal. **F.9** aborts on any such structured-mismatched collision.

We bound this remaining gap with the SM-TCR-EXCL notion of Definition 4.7, the SM-TCR variant adjusted to this scenario; its analysis is given in Section E. Partition the 22-byte tweak space as $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$, where $\mathcal{T}_2$ collects the internal-node addresses (type byte `SF_FXMSS_TREE`, the $\mathbf{H}$ tweaks), $\mathcal{T}_3$ collects the WOTS+C public-key-compression addresses (type byte `SF_WOTS_C_PK`, the $\mathsf{Th}_{l_{\mathsf{sf}}}$ tweaks), and the bijection $\omega : \mathcal{T}_2 \to \mathcal{T}_3$ pairs the two addresses that share the same 9-byte node-identifier prefix (z, i), i.e., ω replaces the type byte `SF_FXMSS_TREE` by `SF_WOTS_C_PK`. The SM-TCR-EXCL game lets the reduction set a challenge at any tweak in $\mathcal{T}_2$ or $\mathcal{T}_3$, but not both at the same time (the type the honest signer uses at that node) while permitting the colliding tweak T^* to be either T_j itself or its bijection partner $\omega(T_j)$ (or $\omega^{-1}(T_j)$ when the honest type is the leaf-compression one). This matches the structured-mismatched collision described above. The reduction embeds one SM-TCR-EXCL challenge per node of the honest structure, simulates the rest of the experiment honestly via the collection oracle Th_λ, and on a structured-mismatched forgery extracts the colliding tweak/message pair as a SM-TCR-EXCL solution. Embedding challenges only at the nodes of the structure suffices even though the verifier accepts an arbitrary in-bounds anchor position $(z^\star, i^\star)$ given by the indicator byte and the index field of the signature (Section 3), so the forgery's verification path may traverse positions that carry no honest node at all: the recomputed path terminates in the honest root $\mathsf{PK.sf}$, hence at the *first* position (walking towards the root) where the recomputed value agrees with an honest node value, the colliding hash call has an honest structure node on its target side, and that is where we placed the challenge. There is one challenge per node of the structure, giving $q_5 \leq L + I$ targets.

The reduction $\mathcal{B}_8$ is given in Algorithm 12. It is the reduction $\mathcal{B}_6$ with both node roles turned into challenges: the L public-key compressions are challenge queries at their type-`SF_WOTS_C_PK` addresses (in $\mathcal{T}_3$) and the I internal nodes at their type-`SF_FXMSS_TREE` addresses (in $\mathcal{T}_2$), while the counter search, the chains, and the stateless key generation again run through the collection oracle Th_λ. The exclusivity constraint of Definition 4.7 is satisfied because the honest structure uses exactly one role per node identifier, and the tweak-reservation rule holds because the collection queries use only chain and grinding tweaks, which lie in $\mathcal{T}_1$.

Claim 4.9. $\left|\mathrm{Succ}^{\mathbf{F.8}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F.9}}(\mathcal{A})\right| \leq \mathrm{InSec}^{\text{SM-TCR-EXCL}}(\mathsf{Th}_{2\cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda; \tilde{\xi}, q_5, q_{\lambda,5})$, *where* $q_5 \leq L + I$, $q_{\lambda,5} \leq Ll_{\mathsf{sf}}w + q_g + q_{\mathsf{kg}}$, $\tilde{\xi} \leq \xi + V_{\mathsf{sf}}$ *is the time of the reduction, and the tweak partition* $(\mathcal{T}_2, \mathcal{T}_3, \omega)$ *pairs each internal-node address with the WOTS+C public-key-compression address sharing the same node identifier.*

F.9: success probability is zero. After **F.9** every winning route has been excluded. Hence



Option 1. The adversary has a WOTS+C instance in place of a tree node: ν_1 is a tree node, ν_1^* is a compressed WOTS+C public key.

$$\mathsf{Th}(\mathsf{PK.seed},\,(z,i)\,\|\,\mathtt{SF_WOTS_C_PK},\,x_0^*\|\cdots\|x_{l_{sf}-1}^*) = \mathsf{Th}(\mathsf{PK.seed},\,(z,i)\,\|\,\mathtt{SF_FXMSS_TREE},\,x_0\|x_1)$$

Option 2. The adversary has a tree node in place of a WOTS+C instance: ν_2 is a compressed WOTS+C public key, ν_2^* is a tree node.

$$\mathsf{Th}(\mathsf{PK.seed},\,(z,i)\,\|\,\mathtt{SF_FXMSS_TREE},\,x_0^*\|x_1^*) = \mathsf{Th}(\mathsf{PK.seed},\,(z,i)\,\|\,\mathtt{SF_WOTS_C_PK},\,x_0\|\cdots\|x_{l_{sf}-1})$$

Figure 4: An example different structures collision.

$\mathrm{Succ}^{\mathbf{F.9}}(\mathcal{A}) = 0$, and combining the bounds of all hops via a telescoping sum yields the statement of Theorem 4.2. The proof uses Conjecture 3.1, which the theorem assumes, in Remark 4.6 and in the hop $\mathbf{F.2} \to \mathbf{F.3}$.

Claim 4.10. $\mathrm{Succ}^{\mathbf{F.9}}(\mathcal{A}) = 0$.

Proof. Fix a run of **F.9** in which $\mathcal{A}$ outputs a forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ that verifies and whose bound message is fresh; we show that one of the abort events of **F.1**–**F.9** occurs. By Remark 4.6 every anchor has been signed, and every bound message $\widetilde{M}_j$ differs from $\widetilde{M}^*$: for the queries of $\mathcal{A}$ by freshness, for the queries added by the wrapper by construction. Let $v_0 = (z^\star, i^\star), v_1, \ldots, v_k$ be the path that the verifier recomputes, with v_k the root, let N_0 be the compressed WOTS+C public key it recomputes at v_0, and let N_t, for $t \geq 1$, be its tree-node value at v_t; since the forgery verifies, $N_k = \mathsf{PK.sf}$.

Walk up the path from v_0 to the root v_k. The root lies in $\mathcal{I}$, and since the structure is valid (Section 2.5), the parent of every other node of $\mathcal{A} \cup \mathcal{I}$ lies in $\mathcal{I}$. Hence, once the path enters $\mathcal{A} \cup \mathcal{I}$, it stays there: let t^* be the least index with $v_{t^*} \in \mathcal{A} \cup \mathcal{I}$, so that $v_{t^*}, \ldots, v_k$ all lie in $\mathcal{A} \cup \mathcal{I}$. Every node above v_{t^*} is the parent of a node in $\mathcal{A} \cup \mathcal{I}$, so $v_t \in \mathcal{I}$ for $t > t^*$. If $t^* > 0$, the entry node v_{t^*} is an anchor: a node of $\mathcal{I}$ has both children in $\mathcal{A} \cup \mathcal{I}$, so if v_{t^*} were internal, v_{t^*-1} would already lie in $\mathcal{A} \cup \mathcal{I}$. Along this segment, compare the values the verifier recomputes with the honest ones. They agree at the root, since $N_k = \mathsf{PK.sf}$. Let $s \geq t^*$ be the least index at which N_s equals the honest value at v_s; such an s exists, since $s = k$ qualifies.

Case $s > t^*$. At $v_s \in \mathcal{I}$ the verifier and the honest signer evaluate $\mathbf{H}$ at the same tree-node address, on different inputs, since N_{s-1} differs from the honest value at v_{s-1}, and with the same output: this is the event of **F.8**.

Case $s = t^*$, with a change of role. If $t^* > 0$, the verifier evaluates $\mathbf{H}$ at the anchor v_{t^*}, where the honest signer compresses a WOTS+C public key; if $t^* = 0$ and $v_0 \in \mathcal{I}$, the verifier compresses a WOTS+C public key where the honest signer evaluates $\mathbf{H}$. Both calls are at the same node identifier with the two partner types, on inputs of different lengths ($2n$ against $l_{\mathsf{sf}} n$ bits), and have the same output: this is the event of **F.9**.

Case $s = t^* = 0$ and $v_0 \in \mathcal{A}$. Then v_0 is the anchor of some signing query j, and the recovered compressed public key is the honest one. Let y^* and d^* be the message digest and the encoded

digest recovered from the forgery, with digits a_i^*. If $y^* = y_j$, the inputs $(R^*, \widetilde{M^*}) \neq (R_j, \widetilde{M_j})$ collide under $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ at idx_j: the event of $\mathbf{F}.\mathbf{1}$. If $y^* \neq y_j$ and $d^* = d_j$, the pairs $(y^*, c^*) \neq (y_j, c_j)$ collide under $\mathbf{H}_{\text{grind}}$ at the address of anchor j, and both satisfy Prop: the event of $\mathbf{F}.\mathbf{3}$. Otherwise $d^* \neq d_j$, so the digit vectors differ, since d is determined by its l_{sf} base-w digits; the verifier accepts only digests whose digits sum to S_{wn}, so some chain i has $a_i^* < a_{j,i}$. If the recovered uncompressed public key differs from the honest one, the event of $\mathbf{F}.\mathbf{7}$ occurs. Otherwise chain i, recomputed from σ_i^*, ends in the honest endpoint. If its value at position $a_{j,i}$ is the revealed block $\sigma_{j,i}$, its value at position $a_{j,i} - 1$ is an $\mathbf{F}$-preimage of $\sigma_{j,i}$ at the honest chain address: the event of $\mathbf{F}.\mathbf{6}$. If not, the recomputed and the honest chain differ at position $a_{j,i}$ and agree at the endpoint, so some step maps distinct inputs at the same chain address to the same output: the event of $\mathbf{F}.\mathbf{5}$.

In every case $\mathbf{F}.\mathbf{9}$ aborts, so $\mathrm{Succ}^{\mathbf{F}.\mathbf{9}}(\mathcal{A}) = 0$. □

The square-root-free form of the reprogramming hop (Corollary C.3) gives the following variant of the theorem for quantum adversaries.

Corollary 4.2 (FXMSS EU-CMA security against quantum adversaries, square-root-free reprogramming term)**.** *In the setting of Theorem 4.2, for a halting (Remark C.1) quantum adversary $\mathcal{A}$ with $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ modeled as a quantum random oracle, and with $S := q_s\, q_h + q_s(q_s - 1)/2 + 1$,*

$$\begin{aligned}
\mathrm{InSec}^{\mathit{EU\text{-}CMA}}(\mathit{FXMSS};\, \xi,\, q_s) \le\ & \mathrm{InSec}^{\mathsf{SM\text{-}rTCR},1}(\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}},\, \mathbf{1};\, \tilde{\xi}',\, q_s,\, \tilde{q}_h) + 32\,\frac{S}{2^n} \\
& + 2\Big(\mathrm{InSec}^{\text{S-TCR(Prop)}}(\mathbf{H}_{\text{grind}} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_s,\, q_{\lambda,0}) + w \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1}) \\
& \quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_2,\, q_{\lambda,2}) + \mathrm{InSec}^{\text{SM-PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_1,\, q_{\lambda,1}) \\
& \quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_4,\, q_{\lambda,4}) + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_3,\, q_{\lambda,3}) \\
& \quad + \mathrm{InSec}^{\text{SM-TCR-EXCL}}(\mathsf{Th}_{2 \cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi},\, q_5,\, q_{\lambda,5})\Big)\,,
\end{aligned}$$

with the same query and time parameters as in Theorem 4.2, and with $q_s := L$ in S and in the terms.

Proof. The proof above bounds $\mathrm{Succ}^{\mathbf{F}.\mathbf{0}}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{F}.\mathbf{1}}(\mathcal{A})$ by the SM-rTCR term (Claim 4.1), and the telescoping sum of Claims 4.3 to 4.10 bounds $\mathrm{Succ}^{\mathbf{F}.\mathbf{2}}(\mathcal{A})$ by the sum inside the parentheses. For the hop from $\mathbf{F}.\mathbf{1}$ to $\mathbf{F}.\mathbf{2}$ we use the last inequality of Corollary C.3 for the distinguisher $\mathcal{B}_1$ of Algorithm 5, which outputs 1 in $\mathbf{DI\text{-}Repro}_0$ exactly when $\mathbf{F}.\mathbf{1}$ succeeds and in $\mathbf{DI\text{-}Repro}_1$ exactly when $\mathbf{F}.\mathbf{2}$ succeeds: $\mathrm{Succ}^{\mathbf{F}.\mathbf{1}}(\mathcal{A}) \le 2\,\mathrm{Succ}^{\mathbf{F}.\mathbf{2}}(\mathcal{A}) + 32\,S/2^n$, in place of Claim 4.2. Theorem 4.2 with the bound of Lemma C.3 ($e = 1$) in Corollary C.3 is the additive alternative; Section 6 instantiates both. □

4.8 Proof of Theorem 4.1 (SHRINCS Security)

We prove EU-CMA security of SHRINCS via a sequence of six game hops. The first hop replaces every PRF-derived secret element of both paths with a uniformly random value; the second hop strips the message-randomizer PRFs. The third hop aborts on a stateful-branch forgery that reuses a signed position with a colliding digest of $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$, the SM-rTCR event of the FXMSS proof. The fourth hop reprograms $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$, so that the digests of the stateful signatures, and with them the counter search of $\mathbf{H}_{\text{grind}}$, no longer depend on the forger's messages; this hop serves both forgery routes. The fifth and sixth hops bound the stateful and the stateless forgery routes by the properties of the hash functions behind FXMSS and SLH-DSA, reusing the reductions of the two component proofs with the other path embedded into their first stage. Throughout this proof we write $\mathrm{Succ}^{i}(\mathcal{A})$ as shorthand for $\mathrm{Succ}^{\mathbf{GAME}.i}(\mathcal{A})$.

GAME.0. **GAME.0** is the SHRINCS EU-CMA experiment of Definition 4.15: the first stage of $\mathcal{A}$ outputs a structure descriptor Π for the stateful path, and the challenger samples

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SHRINCS.KeyGen}(1^n, \Pi)$ with $\mathsf{pk} = \mathsf{PK.seed} \| \mathsf{PK.sl} \| \mathsf{PK.sf}$, where $\mathsf{PK.sl}$ is the SLH-DSA hypertree root and $\mathsf{PK.sf}$ the FXMSS root (Section 3). The adversary makes up to q_s signing queries, each marked with a path bit selecting the stateless or the stateful signing oracle, and wins by producing a valid forgery (M^*, σ^*) with $M^* \notin \{M_i\}_{i=1}^{q_s}$. By Remark 4.6, applied to the stateful path, we assume without loss of generality that $\mathcal{A}$ issues exactly L stateful signing queries, one per anchor of Π, none of them answered by $\perp$; hence $q_s^{\mathsf{sf}} = L$ in every execution, and no stateful signing query has $pos_i = \perp$. By definition $\mathrm{Succ}^0(\mathcal{A}) = \mathrm{Succ}^{\mathsf{SHRINCS\text{-}EU\text{-}CMA}}_{\mathsf{SHRINCS},q_s}(\mathcal{A})$. All games of this proof are played in the model of Assumption 4.1, with $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ a (quantum) random oracle; the additive $\varepsilon_{\mathrm{ds}}$ that Assumption 4.1 attaches to this modeling is charged once in Theorem 4.1, outside the sequence of games, and no hop below charges it.

GAME.0 → GAME.1: Replace $\mathbf{PRF}(\mathsf{SK.seed}, \cdot)$ outputs with uniform random values. **GAME.1** is identical to **GAME.0** except that every secret-key element of the SHRINCS key, the WOTS-TW, WOTS+C, and FORS elements of both paths, is sampled independently and uniformly at random from $\{0,1\}^n$ instead of being derived as $\mathbf{PRF}(\mathsf{SK.seed}, \mathbf{ADRS})$. This hop covers the secret elements of both paths, so no PRF-derived secret remains afterwards. A standard reduction $\mathcal{B}_1$ samples $\mathsf{PK.seed}$ and $\mathsf{SK.prf}$ honestly and, wherever key generation would call $\mathbf{PRF}(\mathsf{SK.seed}, \mathbf{ADRS})$, queries its oracle at **ADRS** instead; the queries are classical and pairwise distinct by the unique-address discipline. If the oracle implements $\mathbf{PRF}(S, \cdot)$ for a uniform key S, the simulation is **GAME.0**; if it is a uniformly random function, it is **GAME.1**.

Claim 4.11. $\left|\mathrm{Succ}^0(\mathcal{A}) - \mathrm{Succ}^1(\mathcal{A})\right| \le \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \xi, q_1)$, *where q_1 is the total number of secret-key elements across both paths of the SHRINCS key.*

GAME.1 → GAME.2: Replace $\mathbf{PRF_{msg}}$ outputs with uniform random values. **GAME.2** is identical to **GAME.1** except that on every signing query, on either path, the randomizer R that feeds the message hash is sampled uniformly from $\{0,1\}^n$ instead of being derived by $\mathbf{PRF_{msg}}$ (Section 2.2).[11] Both derivations are calls of the combined randomizer function $\mathbf{PRF_{msg}}$ under the single key $\mathsf{SK.prf}$, so we replace it once, for both paths together: a standard reduction $\mathcal{B}_2$ replaces every $\mathbf{PRF_{msg}}$ call, on either path, by a query to a PRF-or-random oracle and simulates the rest of the experiment honestly, using the uniform secret elements of **GAME.1**, which it samples itself. The selector keeps the inputs of the two paths distinct, so when the oracle is a random function the randomizers of both paths are uniform, independent across distinct inputs, and equal on a repeated input, as **GAME.2** prescribes.

Claim 4.12. $\left|\mathrm{Succ}^1(\mathcal{A}) - \mathrm{Succ}^2(\mathcal{A})\right| \le \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF_{msg}}; \xi, q_s)$, *where $q_s = q_s^{\mathsf{sl}} + q_s^{\mathsf{sf}}$ counts the signing queries of both paths.*

Remark 4.9 (PRF accounting)**.** The reductions $\mathcal{B}_1$ and $\mathcal{B}_2$ query the PRF oracle classically, once per derived secret element (q_1) or per signing query (q_s), and never evaluate the challenged family at a key of their own: what they build from the oracle's answers consists of tweakable-hash evaluations only. Under Table 6 the two terms therefore instantiate at the forger's budget, $\xi/|\mathcal{K}|$ classically and $2\xi/\sqrt{|\mathcal{K}|}$ quantumly (see the discussion after Definition 4.3).

GAME.2 → GAME.3: Abort on a stateful-branch forgery with an $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ collision at a signed position. The SHRINCS verifier dispatches on the indicator byte b of the signature: a forgery verifies either via the stateless branch ($b = h_{\mathsf{sf}}$) or via the stateful branch ($b < h_{\mathsf{sf}}$), and

[11] The randomizer is sampled lazily and cached: on the first signing query with a given input of $\mathbf{PRF_{msg}}$, R is drawn uniformly from $\{0,1\}^n$ and stored, and a later signing query with the same input receives the stored R. On the stateful path the inputs contain the pairwise distinct indices idx, so no input repeats. On the stateless path an input repeats exactly when a signing query repeats both the message, i.e., the pair (ctx, M) of Definition 4.15, and *opt_rand*: in the deterministic mode, *opt_rand* = $\mathsf{PK.seed}$, this is every repeated message; in the hedged mode it is a repeated message whose fresh *opt_rand* coincides with an earlier one. In both modes the cached value is the one the scheme with a random $\mathbf{PRF_{msg}}$ would return, so the reading adds no term.

never via both. **GAME.3** is identical to **GAME.2** except that the game aborts if $\mathcal{A}$'s forgery verifies via the stateful branch at a position $(z^\star, i^\star)$ that was signed in some stateful query j and

$$\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R^*,\, \mathsf{PK.seed},\, \mathsf{PK.sf},\, \mathsf{idx}_j,\, \widetilde{M}^*) = \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_j,\, \mathsf{PK.seed},\, \mathsf{PK.sf},\, \mathsf{idx}_j,\, \widetilde{M}_j),$$

where R^* is the randomizer in σ^*, $\widetilde{M}^*$ the bound message of the forgery, and R_j, $\widetilde{M}_j$, idx_j those of query j. This is the abort event of the hop $\mathbf{F.0} \to \mathbf{F.1}$ of Section 4.7, applied to the stateful branch.

The reduction $\mathcal{B}_3$ is the reduction $\mathcal{B}_0$ of Algorithm 4 extended by the stateless path. It receives $P = \mathsf{PK.seed}$ from the SM-RTCR challenger, runs the structure stage of $\mathcal{A}$, samples the secret elements of both paths uniformly at random (**GAME.1**), runs the key generation of both paths itself, and hands $\mathcal{A}$ the key $\mathsf{pk} = \mathsf{PK.seed} \| \mathsf{PK.sl} \| \mathsf{PK.sf}$. It answers stateless signing queries itself with uniform randomizers (**GAME.2**). On a stateful signing query at the anchor $pos_i = \mathsf{leafSelect}_\Pi(\mathsf{ctr})$ it queries its SM-RTCR oracle at the tweak $(\mathsf{PK.sf}, \mathsf{idx}_i)$ on the message $\widetilde{M}_i$, receives (y_i, R_i), and runs the rest of the stateful signing procedure on the digest y_i; the tweaks are pairwise distinct because leafSelect is injective, and the oracle's uniform R_i is the uniform randomizer of **GAME.2**. On a forgery that meets the abort condition, $\mathcal{B}_3$ outputs $(j, \widetilde{M}^*, R^*)$. The pair $(\widetilde{M}^*, R^*)$ differs from $(\widetilde{M}_j, R_j)$ because M^* is fresh and the bound-message encoding is injective, so the output is an SM-RTCR collision at the j-th tweak.

Claim 4.13. $\left|\mathrm{Succ}^2(\mathcal{A}) - \mathrm{Succ}^3(\mathcal{A})\right| \le \mathrm{InSec}^{\mathsf{SM\text{-}rTCR},1}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}, \mathbf{1}; \tilde{\xi}'_{\mathsf{sf}}, q_s^{\mathsf{sf}}, \tilde{q}_h^{\mathsf{sf}})$, *where* $\tilde{\xi}'_{\mathsf{sf}} \le \xi + t_{\mathsf{sim}} + t^{\mathsf{sl}}_{\mathsf{sim}}$ *accounts for the honest key generation and signing of both paths performed by* $\mathcal{B}_3$, *with* t_{sim} *as in Theorem 4.2 and* $t^{\mathsf{sl}}_{\mathsf{sim}}$ *as in Theorem 4.3, and* $\tilde{q}_h^{\mathsf{sf}} \le q_h + 1$ *counts* $\mathcal{B}_3$*'s evaluations of* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$*: the forger's queries and one verification of the forgery.*

The $t_{\mathsf{sim}} + t^{\mathsf{sl}}_{\mathsf{sim}}$ evaluations concern Th_λ and $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ only, which are modeled independently of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ (Assumption 4.1); by Lemma 4.2 the term depends on $\tilde{\xi}'_{\mathsf{sf}}$ only through its query parameters.

GAME.3 → GAME.4: Adaptive reprogramming of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$. In **GAME.3** the digest of a stateful signature is $y_i = \mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_i, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i)$, so it depends on the forger's message, and so does the counter c_i that the signer finds by evaluating $\mathbf{H}_{\mathrm{grind}}$ on y_i until the target sum is met. The function $\mathbf{H}_{\mathrm{grind}}$ is a tweakable hash function and a member of Th_λ. A reduction against a member of Th_λ that receives P only in its second stage could therefore reproduce this counter search only by its own evaluations of Th_λ, which are charged to its time. We remove the dependence in the same way as the FXMSS proof does in the hop $\mathbf{F.1} \to \mathbf{F.2}$: we fix the digests at setup and reprogram $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ to them. The counter search then depends on the structure alone, and the reductions of the next two hops perform it before they receive P.

In this hop we use that $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ is modeled as a random oracle (Assumption 4.1 and Remark 4.7). **GAME.4** is identical to **GAME.3** except for the digests of the stateful signatures. At setup the challenger samples one uniformly random digest $y_{pos} \leftarrow_\$ \{0,1\}^{2n}$ for every anchor pos of the structure Π, L digests in total. On a stateful signing query M_i at the anchor $pos_i = \mathsf{leafSelect}_\Pi(\mathsf{ctr})$ it samples $R_i \leftarrow_\$ \mathcal{R} = \{0,1\}^n$, *reprograms* the random oracle so that $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_i, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i) := y_{pos_i}$, uses $y_i = y_{pos_i}$ as the digest, and continues as the stateful branch of Algorithm 2 does. Every anchor is used at most once, since leafSelect is injective, so the digests of distinct queries are independent. A query with $pos_i = \bot$ is answered as in **GAME.3**, and nothing is reprogrammed. The abort condition of **GAME.3** is kept.

The distinguisher $\mathcal{B}_4$ plays the reprogramming game with distinct inputs of Definition C.2 with $X_1 = \mathcal{R}$ and X_2 the set of the remaining inputs $(\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}, \widetilde{M})$; it is the distinguisher $\mathcal{B}_1$ of Algorithm 5 extended by the stateless path. The guard of Definition C.2 never fires, because leafSelect is injective and the inputs of the calls therefore carry pairwise distinct idx_i. $\mathcal{B}_4$ samples PK.seed, runs the structure stage of $\mathcal{A}$ to obtain Π, samples the secret elements of both paths uniformly at random (**GAME.1**), performs the key generation of both paths

itself, and hands $\mathcal{A}$ the key $\mathsf{pk} = \mathsf{PK.seed}\|\mathsf{PK.sl}\|\mathsf{PK.sf}$. It answers the queries of $\mathcal{A}$ to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ with its oracle $\mathcal{O}_b$, and it answers the stateless signing queries itself with uniform randomizers (**GAME.2**); these evaluate $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and Th_λ only. On a stateful signing query it calls $R_i \leftarrow \mathsf{Reprogram}(\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i)$, reads $y_i \leftarrow \mathcal{O}_b(R_i, (\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i))$, and runs the rest of the stateful signing procedure on y_i, including the counter search; on a query with $pos_i = \bot$ it makes no call to $\mathsf{Reprogram}$. $\mathcal{B}_4$ outputs 1 exactly when $\mathcal{A}$ outputs a fresh valid forgery on which the abort of **GAME.3** does not fire. A forgery on the stateless branch is verified with evaluations of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and Th_λ only; a forgery on the stateful branch needs one query to $\mathcal{O}_b$, at $(R^*, (\mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}^*, \widetilde{M}^*))$, which also decides the abort condition of **GAME.3**. If $b = 0$ the view of $\mathcal{A}$ is that of **GAME.3**, and if $b = 1$ it is that of **GAME.4**.

Claim 4.14. *Modeling* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ *as a (quantum) random oracle,*

$$\left|\mathrm{Succ}^3(\mathcal{A}) - \mathrm{Succ}^4(\mathcal{A})\right| \;\le\; \mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}; \tilde{\xi}'_{\mathsf{sf}}, q^{\mathsf{sf}}_s, q^{\mathsf{sf}}_s + q_h + 1),$$

where q_h *denotes the number of* $\mathcal{A}$*'s queries to* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ *besides the signing queries, and* $\tilde{\xi}'_{\mathsf{sf}} \le \xi + t_{\mathsf{sim}} + t^{\mathsf{sl}}_{\mathsf{sim}}$ *accounts for the honest key generation and signing of both paths performed by* $\mathcal{B}_4$*, with* t_{sim} *as in Theorem 4.2 and* $t^{\mathsf{sl}}_{\mathsf{sim}}$ *as in Theorem 4.3.*

$\mathcal{B}_4$ issues q^{sf}_s calls to $\mathsf{Reprogram}$ and at most $q^{\mathsf{sf}}_s + q_h + 1$ queries to $\mathcal{O}_b$: the forwarded queries of $\mathcal{A}$, one read per stateful signing query, and at most one final query to verify the forgery, the query pattern of $\mathcal{B}_1$ (Algorithm 5). The $t_{\mathsf{sim}} + t^{\mathsf{sl}}_{\mathsf{sim}}$ evaluations concern Th_λ and $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ only, which are modeled independently of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ (Assumption 4.1); by Lemma 4.2 the term depends on $\tilde{\xi}'_{\mathsf{sf}}$ only through its query parameters. The concrete bounds are those of Corollary C.3 at $q_s = q^{\mathsf{sf}}_s$, in the ROM and in the QROM.

By Remark 4.6, applied to the stateful path at **GAME.0**, $\mathcal{A}$ issues exactly L stateful signing queries, none answered by $\bot$, so $\mathcal{B}_4$ issues $q^{\mathsf{sf}}_s = L$ calls to $\mathsf{Reprogram}$ in every execution, each followed by one read-back, and all of them precede the final query, exactly as for $\mathcal{B}_1$. The signing queries that the wrapper of Remark 4.6 adds after the output of $\mathcal{A}$ are answered by $\mathcal{B}_4$ like every other stateful query. Their inputs carry anchors not used before and bound messages different from $\widetilde{M}^*$, so none of them is answered by $\bot$, none reprograms the point of the final query, and $\mathcal{B}_4$ outputs 1 in $\mathbf{DI\text{-}Repro}_0$ exactly when **GAME.3** succeeds and in $\mathbf{DI\text{-}Repro}_1$ exactly when **GAME.4** succeeds. Each call is preceded by at most $q_h + i - 1$ queries, and the counting in the proof of that corollary gives $\sum_i \hat{q}_i \le S$ for $\mathcal{B}_4$ as for $\mathcal{B}_1$. No padding call of Remark C.4 is needed.

GAME.4 → GAME.5: Abort on a stateful-branch forgery. **GAME.5** is identical to **GAME.4** except that the game aborts if $\mathcal{A}$'s forgery verifies via the stateful branch, i.e., the root recovered from σ^* equals $\mathsf{PK.sf}$, so that σ^* is a valid FXMSS signature on $\widetilde{M}^*$ under $(\mathsf{PK.seed}, \mathsf{PK.sf})$.

Write W_{sf} for the event that $\mathcal{A}$ wins **GAME.4** with a forgery that verifies via the stateful branch. **GAME.5** aborts on every forgery that verifies via the stateful branch, so a forgery that wins **GAME.4** is lost in **GAME.5** exactly when W_{sf} occurs, and $\mathrm{Succ}^4(\mathcal{A}) - \mathrm{Succ}^5(\mathcal{A}) = \Pr_{\mathbf{GAME.4}}[W_{\mathsf{sf}}]$.

The augmented games. The games of Section 4.7 have no stateless path, so we extend them. For $k \in \{2, \ldots, 9\}$ let $\mathbf{F}.k^+$ be the game $\mathbf{F}.k$ of Section 4.7 for the structure Π, extended by the stateless path of **GAME.4**: the challenger also samples the stateless secret elements uniformly at random, gives $\mathcal{A}$ the key $\mathsf{pk} = \mathsf{PK.seed}\|\mathsf{PK.sl}\|\mathsf{PK.sf}$, and answers stateless signing queries as **GAME.2** does, with the cached randomizers and its own random oracle $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$. The success event of $\mathbf{F}.k^+$ is that $\mathcal{A}$'s forgery is fresh with respect to the signing queries of both paths, verifies via the stateful branch, and triggers none of the aborts of $\mathbf{F}.1$ to $\mathbf{F}.k$. Restricted to the stateful signing oracle and the stateful verification branch, **GAME.4** is the game $\mathbf{F}.2$ for the structure Π: its secret elements are uniform since **GAME.1**, its randomizers since **GAME.2**, the abort of

GAME.3 is the SM-rTCR abort of **F.1**, and the digests are fixed at setup and reprogrammed as in **F.2**. With the stateless path added, **GAME.4** is $\mathbf{F}.\mathbf{2}^{+}$ played by the forger $\mathcal{A}_{\mathsf{F}}$ that runs $\mathcal{A}$, forms $\widetilde{M}$ from the context, $\mathsf{PK.sl}$ and the message of each stateful query, and names the leaves in the signer's order, and W_{sf} is its success event, so $\Pr_{\mathbf{GAME.4}}[W_{\mathsf{sf}}] = \mathrm{Succ}^{\mathbf{F}.\mathbf{2}^{+}}(\mathcal{A}_{\mathsf{F}})$. Each of Claims 4.3 to 4.9 holds with $\mathbf{F}.k^{+}$ in place of $\mathbf{F}.k$ and with the augmented reduction below in place of the original one, and Claim 4.10 holds for $\mathbf{F}.\mathbf{9}^{+}$, since its proof uses only the stateful branch of the verification and the honest stateful values. The reductions $\mathcal{B}_2, \ldots, \mathcal{B}_8$ of Section 4.7 do not simulate the stateless path of these games. As in the stateless hop below, we cannot leave it to time: an opaque reduction that simulated the stateless path in its second stage would charge up to $t^{\mathsf{sl}}_{\mathsf{sim}} \leq 2^{61}$ evaluations of Th_λ to every collection term.

The augmented reductions. Every reduction of the hops $\mathbf{F}.\mathbf{2} \to \mathbf{F}.\mathbf{9}$ is a two-stage adversary against a member of Th_λ (Definition 4.12) whose first stage runs the structure stage of the forger, samples the values its game prescribes (the WOTS+C secret elements, or the fresh chain values from **F.4** on), and builds the stateful structure through the collection oracle before it receives P. We augment each of them in the same way. In the first stage it additionally samples the stateless secret elements uniformly at random and builds the complete virtual hypertree, all WOTS-TW chains, XMSS nodes, and FORS trees, through the collection oracle at the stateless addresses, extending the stateless key generation it already performs to obtain $\mathsf{PK.sl}$ (q_{kg} in Theorem 4.2). The stateless structure is a function of the uniform secret elements and P alone, so it is available before P is released, and the reservation rule of Definition 4.12 is respected: the challenge tweaks of every stateful reduction carry the stateful address types, while every stateless call carries a stateless type at the same byte offset (Lemma 4.3 and Table 2). In the second stage it holds P, hands $\mathcal{A}$ the key $\mathsf{pk} = \mathsf{PK.seed} \| \mathsf{PK.sl} \| \mathsf{PK.sf}$, and answers a stateless signing query by taking R as in **GAME.2**: a fresh $R \leftarrow_{\$} \{0,1\}^n$ on a new $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ input and the stored R on a repeated one, so that a repeated input receives the earlier signature; evaluating the digest under its own simulation of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, which is independent of Th_λ and of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ (Assumption 4.1), and assembling the signature from the stored values; the stateless path has no counter search, so the second stage evaluates no member of Th_λ for it, and its evaluations of the simulated $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, an independent random function under Assumption 4.1, are bookkeeping, which Section 4.3 does not charge. In the output step of every augmented reduction, "σ^* is valid" means that σ^* verifies via the stateful branch. The reduction reads the indicator byte and runs only the stateful verification, so its own evaluations stay within V_{sf}. The event it tests concerns the stateful branch only; for the distinguisher $\mathcal{B}_3$ of the SM-UD hybrids it is the success event of its game, and for the reductions that extract it is the abort event of the next game. The extraction step of each reduction is otherwise unchanged, for three reasons. First, the verification of a stateful-branch forgery evaluates the collection only at stateful address types, so it meets the challenge tweaks exactly as in Section 4.7, and the stateless calls, which carry stateless types, meet none of them. Second, the forged digest is computed under the reduction's own simulation of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, as in the original reduction. Third, freshness transfers: a forgery that is fresh with respect to both paths has $\widetilde{M}^* \neq \widetilde{M}_j$ for every stateful query j, since the bound-message encoding is injective (Section 3). The forger's view under each augmented reduction is that of the game its original reduction simulates, extended by the stateless path: $\mathbf{F}.k^{+}$ for the reductions that extract, and the corresponding hybrid of $\mathbf{F}.\mathbf{3}^{+}$ and $\mathbf{F}.\mathbf{4}^{+}$ for $\mathcal{B}_3$.

The resources of the augmented reductions differ from those in Theorem 4.2 as follows. The collection budget of every collection term grows by the size of the stateless structure, $\tilde{q}_{\lambda,i} \leq q_{\lambda,i} + Q^{\mathsf{sl}}_{\Lambda}$ with $Q^{\mathsf{sl}}_{\Lambda} \approx 2^{62}$ as in Theorem 4.3, and their time stays $\tilde{\xi}_{\mathsf{sf}} \leq \xi + V_{\mathsf{sf}}$: the stateless simulation adds no evaluation of a member of Th_λ (its simulation of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ is bookkeeping, as above). The budgets are discharged by Corollary 4.1, so none of these additions moves the instantiated bounds.

Claim 4.15. *Modeling* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ *as a (quantum) random oracle and* $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ *as in Assumption 4.1,*

$$\begin{aligned}\left|\mathrm{Succ}^4(\mathcal{A}) - \mathrm{Succ}^5(\mathcal{A})\right| \;\le\; & \mathrm{InSec}^{\text{S-TCR(Prop)}}(\mathbf{H}_{\mathrm{grind}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_s^{\mathsf{sf}},\, \tilde{q}_{\lambda,0}) \\ & + w \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_1^{\mathsf{sf}},\, \tilde{q}_{\lambda,1}) + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_2^{\mathsf{sf}},\, \tilde{q}_{\lambda,2}) \\ & + \mathrm{InSec}^{\text{SM-PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_1^{\mathsf{sf}},\, \tilde{q}_{\lambda,1}) + \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_4^{\mathsf{sf}},\, \tilde{q}_{\lambda,4}) \\ & + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_3^{\mathsf{sf}},\, \tilde{q}_{\lambda,3}) + \mathrm{InSec}^{\text{SM-TCR-EXCL}}(\mathsf{Th}_{2\cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda;\, \tilde{\xi}_{\mathsf{sf}},\, q_5^{\mathsf{sf}},\, \tilde{q}_{\lambda,5}),\end{aligned}$$

the seven collection summands of Theorem 4.2, with its query counts $q_1^{\mathsf{sf}}, \ldots, q_5^{\mathsf{sf}}$ *and budgets* $q_{\lambda,i}$ *at* $q_s = q_s^{\mathsf{sf}}$*, at the resources of the augmented reductions:* $\tilde{q}_{\lambda,i} \le q_{\lambda,i} + Q_\Lambda^{\mathsf{sl}}$ *and* $\tilde{\xi}_{\mathsf{sf}} \le \xi + V_{\mathsf{sf}}$*.*

The bound is the sum of Claims 4.3 to 4.10 for the augmented reductions.

GAME.5 → GAME.6: Abort on a stateless-branch forgery. GAME.6 is identical to **GAME.5** except the game aborts if $\mathcal{A}$'s forgery (M^*, σ^*) verifies via the *stateless* branch, i.e., σ^* is a valid SLH-DSA signature on the message $\mathsf{PK.sf} \| M^*$ with the forgery's context under the embedded SLH-DSA public key $(\mathsf{PK.seed}, \mathsf{PK.sl})$.

A forgery that verifies via the stateless branch is an SLH-DSA forgery under the stateless key $(\mathsf{PK.seed}, \mathsf{PK.sl})$. For the two reasons given in Remark 4.3 we do not reduce to Theorem 4.3 as a black box; as in the stateful hop above, we open its proof and embed the other path into its reductions.

The proof of Theorem 4.3 in [HK22] begins with two PRF hops, after which every secret element is uniform and the randomizer R is drawn by a random function of the $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ input; the remaining hops bound the insecurity of that idealized experiment by the ten summands of Theorem 4.3 other than the two PRF terms, at the resources of our restatement. The stateless path of **GAME.5**, restricted to the stateless signing oracle and the stateless verification branch, is exactly this idealized experiment on the messages $\mathsf{PK.sf} \| M$ with the queries' contexts: its secret elements are uniform since **GAME.1** and its randomizers since **GAME.2**.

Write W_{sl} for the event that $\mathcal{A}$ wins **GAME.5** with a forgery that verifies via the stateless branch. **GAME.6** aborts on every forgery that verifies via the stateless branch, so a forgery that wins **GAME.5** is lost in **GAME.6** exactly when W_{sl} occurs, and $\mathrm{Succ}^5(\mathcal{A}) - \mathrm{Succ}^6(\mathcal{A}) = \Pr_{\mathbf{GAME.5}}[W_{\mathsf{sl}}]$.

The augmented games. For every game $\mathbf{HK}.x$ of the proof of Theorem 4.3 from the second PRF hop on, $x \in \{2, 3, 3.1, \ldots, 3.4, 4\}$ (Remark 4.5), let $\mathbf{HK}.x^+$ be $\mathbf{HK}.x$ extended by the stateful path of **GAME.5**: the challenger also runs the structure stage of $\mathcal{A}$, samples the WOTS+C secret elements uniformly at random, fixes the stateful digests at setup and programs them into its own random oracle $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as in **GAME.4**, gives $\mathcal{A}$ the key $\mathsf{pk} = \mathsf{PK.seed} \| \mathsf{PK.sl} \| \mathsf{PK.sf}$, and answers stateful signing queries as **GAME.5** does. The success event of $\mathbf{HK}.x^+$ is that $\mathcal{A}$'s forgery is fresh with respect to the signing queries of both paths, verifies via the stateless branch, and triggers none of the aborts of the games of that proof up to $\mathbf{HK}.x$; a stateless-branch forgery triggers none of the aborts of **GAME.3** and **GAME.5**, which concern the stateful branch. For the first of these games, $\mathbf{HK}.\mathbf{2}^+$ is **GAME.5** and its success event is W_{sl}. The step from $\mathbf{HK}.\mathbf{3}.\mathbf{4}$ to $\mathbf{HK}.\mathbf{4}$ bounds the probability of a WOTS-TW forgery at one of the instances through the games of [HK22, Theorem 2], a d-EU-naCMA experiment; we extend those games by the stateful path in the same way, and their success event is a fresh stateless-branch forgery that contains a WOTS-TW forgery at one of the instances. The hops and final bounds of that proof, applied to the extended games with the augmented reductions below in place of the original ones, give the ten summands: the hop to $\mathbf{HK}.\mathbf{3}$, which [HK22] take from [BHK+19], gives the ITSR term; the hops from $\mathbf{HK}.\mathbf{3}$ to $\mathbf{HK}.\mathbf{3}.\mathbf{3}$ give the SM-TCR terms for $\mathbf{H}$, Th_k, and $\mathsf{Th}_{l_{\mathsf{sl}}}$, and the hop from $\mathbf{HK}.\mathbf{3}.\mathbf{3}$ to $\mathbf{HK}.\mathbf{3}.\mathbf{4}$ the SM-TCR term for q_7; inside [HK22, Theorem 2], w hybrid hops give the SM-UD term, the following hop the SM-TCR term for q_3, and its final bound the SM-PRE term; and the final bound of the proof, from [BHK+19, Claim 23], gives the three SM-TCR terms for q_8 and the SM-DSPR term.

The augmented reductions. The reductions behind the collection terms are two-stage adversaries against a member of Th_λ (Definition 4.12): the first stage commits its challenge targets and completes the virtual hypertree through the collection oracle at tweaks disjoint from the challenge tweaks, and the second stage, holding P, answers the forger's signing queries from the stored values and extracts from the forgery. The reduction behind the ITSR term holds P from the start and performs the stateless key generation and signing itself, with the per-signature digests served by its challenge oracle. Each of these reductions keeps the cache of **GAME.2** for the stateless randomizers: a stateless signing query that repeats the $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ input of an earlier one receives the earlier signature. For the ITSR reduction this replay issues no new challenge query; on pairwise distinct inputs the fresh keys of its challenge oracle (Definition 4.11) are distributed as the cached randomizers of **GAME.2**, and the number of challenge queries stays at most q_s^{sl}. We augment each of them to simulate the full **GAME.5** instead of its stateless path alone, in the same way for all of them.

In the first stage, the augmented reduction additionally builds the stateful path, in three steps. First, it runs the structure stage $\mathcal{A}_1$ of the forger to learn Π. Second, it samples every WOTS+C secret element of Π uniformly at random and runs the FXMSS key-generation procedure on these elements through the collection oracle at the stateful addresses, to obtain $\mathsf{PK.sf}$. Third, it samples the digests $y_{pos} \leftarrow_\$ \{0,1\}^{2n}$ of **GAME.4**, one per anchor of Π, and performs the counter search for each of them by querying $\mathbf{H}_{\text{grind}}$ through the collection oracle at the grinding address of the anchor, on y_{pos} and the counters in the order of the signer, and stores the counter and the encoded digest; this is the counter search of the FXMSS reductions and costs the same grinding budget q_g (Remark 4.4), and by Conjecture 3.1 it succeeds within the counter cap.

All three steps run through Th_λ before P is released, because nothing in the stateful path depends on the forger's messages any more: the secret elements are uniform since **GAME.1**, and the digests, and with them the counters, are fixed at setup since **GAME.4**. The reservation rule of Definition 4.12 is respected because the stateless challenge tweaks and the stateful calls, those of $\mathbf{H}_{\text{grind}}$ included, carry different address types at the same byte offset (Lemma 4.3 and Table 2).

In the second stage, the reduction holds P, hands $\mathcal{A}$ the key $\mathsf{pk} = \mathsf{PK.seed}\|\mathsf{PK.sl}\|\mathsf{PK.sf}$, and answers a stateful signing query at the anchor pos_i as **GAME.5** prescribes: it samples $R_i \leftarrow_\$ \{0,1\}^n$, programs its own simulation of the random oracle $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, which is independent of Th_λ and of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ (Assumption 4.1), so that $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_i, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_i, \widetilde{M}_i) := y_{pos_i}$, and assembles the signature from the stored values. The second stage evaluates no member of Th_λ for the stateful path (its simulation of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ is bookkeeping, Section 4.3); in particular it never evaluates $\mathbf{H}_{\text{grind}}$, so the membership of $\mathbf{H}_{\text{grind}}$ in Th_λ costs the collection terms collection queries and not time. The ITSR reduction, which holds P throughout, performs the same stateful key generation and counter search by its own evaluations and programs $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ in the same way. In the output step of every augmented reduction, a valid forgery means one that verifies via the stateless branch.

The reduction reads the indicator byte and runs only the stateless verification, so its own evaluations stay within V_{sl}, or, for the ITSR reduction, within the resources $\tilde{\xi}''_{\mathsf{sl}}$ and $\tilde{q}^{\mathsf{sl}}_h$ of Theorem 4.1. The event it tests concerns the stateless branch only; for the distinguishers behind the SM-UD term it is the success event of its game, for the reductions behind a hop it is the abort event of the next game, and for the reduction behind the SM-PRE term it is the success event of the last extended game of [HK22, Theorem 2].

For the reductions behind the final bound of [BHK+19, Claim 23], one SM-TCR reduction for $\mathbf{F}$, whose success probability is counted three times in the SM-TCR terms for q_8, and one SM-DSPR reduction for the SM-DSPR term, it is the event E that the forgery is fresh, verifies via the stateless branch, and its FORS part contains, at a leaf position j revealed by no signing query, a value x' with $\mathbf{F}(P, T_j, x')$ equal to the stored leaf value at the leaf tweak T_j, whose committed secret value is x_j.

The extraction step of each reduction is otherwise unchanged, for three reasons. First, the

verification of a stateless-branch forgery evaluates the collection only at stateless address types, so it meets the challenge tweaks exactly as in the proof of Theorem 4.3, and the stateful calls, which carry stateful types, meet none of them. Second, the stateless digests are computed under the reduction's own simulation of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, which Assumption 4.1 models as a random function independent of Th_λ (the reductions of [HK22] evaluate the message hash directly), or, for the ITSR reduction, by its challenge oracle; the stateful path never evaluates $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$. Third, freshness transfers: a forgery that is fresh with respect to both paths is fresh for the embedded SLH-DSA instance, since the stateless message $\mathsf{PK.sf}\|M$ and its context determine (ctx, M) (Section 3). The forger's view under each augmented reduction is that of the game its original reduction simulates, extended by the stateful path: $\mathbf{HK}.x^+$ for a reduction that simulates $\mathbf{HK}.x$, the extended games of [HK22, Theorem 2] with the chain values replaced for the SM-TCR reduction for q_3 and the SM-PRE reduction, and the corresponding hybrid, extended in the same way, for the hybrid reductions behind the SM-UD term.

The resources of the augmented reductions differ from those in Theorem 4.3 as follows. The collection budget of every collection term grows by the size of the stateful structure and by the grinding budget, $q_\lambda \le Q^{\mathsf{sl}}_\Lambda + L(l_{\mathsf{sf}} w + 1) + I + q_g$, an addition below 2^{37} against $Q^{\mathsf{sl}}_\Lambda \approx 2^{62}$ for $L = 2^{20}$ even with $q_g \le 2^{16} L$ by Conjecture 3.1, discharged by Corollary 4.1; their time stays $\tilde{\xi}_{\mathsf{sl}} \le \xi + V_{\mathsf{sl}}$, since the stateful simulation adds no evaluation of a member of Th_λ (its simulation of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ is bookkeeping, Section 4.3).

Claim 4.16. *Modeling* $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ *and* $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ *as (quantum) random oracles,*

$$
\begin{aligned}
&\left|\mathrm{Succ}^5(\mathcal{A}) - \mathrm{Succ}^6(\mathcal{A})\right| \le \\
&\quad \mathrm{InSec}^{\mathrm{ITSR}}(\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}; \tilde{\xi}''_{\mathsf{sl}}, q^{\mathsf{sl}}_s, \tilde{q}^{\mathsf{sl}}_h) + w \cdot \mathrm{InSec}^{\text{SM-UD}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_2, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_3, q_\lambda) + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_7, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-PRE}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_2, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{H} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_4, q_\lambda) + \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_k \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_5, q_\lambda) \\
&\quad + \mathrm{InSec}^{\text{SM-TCR}}(\mathsf{Th}_{l_{\mathsf{sl}}} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_6, q_\lambda) \\
&\quad + 3 \cdot \mathrm{InSec}^{\text{SM-TCR}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_8, q_\lambda) + \mathrm{InSec}^{\text{SM-DSPR}}(\mathbf{F} \in \mathsf{Th}_\lambda; \tilde{\xi}_{\mathsf{sl}}, q^{\mathsf{sl}}_8, q_\lambda)\,,
\end{aligned}
$$

the ten summands of Theorem 4.3 other than its two PRF *terms, with the query counts* $q^{\mathsf{sl}}_2, \ldots, q^{\mathsf{sl}}_8$ *of that theorem at* $q_s = q^{\mathsf{sl}}_s$ *and at the resources of the augmented reductions:* $q_\lambda \le Q^{\mathsf{sl}}_\Lambda + L(l_{\mathsf{sf}} w + 1) + I + q_g$ *and* $\tilde{\xi}_{\mathsf{sl}} \le \xi + V_{\mathsf{sl}}$ *for the collection terms, and* $\tilde{\xi}''_{\mathsf{sl}} \le \xi + t^{\mathsf{sl}}_{\mathsf{sim}} + L(l_{\mathsf{sf}} w + 1) + I + q_g$ *with* $\tilde{q}^{\mathsf{sl}}_h \le q_h + 1$ *evaluations of* $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ *for the* ITSR *term.*

GAME.6: success probability is zero. A valid forgery must pass SHRINCS.Verify, which accepts only via one of the two branches selected by the indicator byte: the stateless branch is excluded by **GAME.6** and the stateful branch by **GAME.5**, whose abort conditions every later game keeps. Hence $\mathrm{Succ}^6(\mathcal{A}) = 0$, and combining the bounds of all hops (Claims 4.11 to 4.16) via a telescoping sum yields the statement of Theorem 4.1, with the additive $\varepsilon_{\mathrm{ds}}$ charged by the modeling of Assumption 4.1:

$$
\begin{aligned}
\mathrm{InSec}^{\text{EU-CMA}}(\mathsf{SHRINCS}; \xi, q_s) \le\ & \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}; \xi, q_1) + \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}_{\mathbf{msg}}; \xi, q_s) \\
& + \mathrm{InSec}^{\mathsf{SM\text{-}rTCR},1}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}, \mathbf{1}; \tilde{\xi}'_{\mathsf{sf}}, q^{\mathsf{sf}}_s, \tilde{q}^{\mathsf{sf}}_h) \\
& + \mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}; \tilde{\xi}'_{\mathsf{sf}}, q^{\mathsf{sf}}_s, q^{\mathsf{sf}}_s + q_h + 1) + \Sigma_{\mathsf{sf}} + \Sigma_{\mathsf{sl}} + \varepsilon_{\mathrm{ds}}\,,
\end{aligned}
$$

where Σ_{sf} and Σ_{sl} denote the sums on the right-hand sides of Claims 4.15 and 4.16. The proof uses Conjecture 3.1, which the theorem assumes, where it needs that no stateful signing call fails: in Remark 4.6, in the hop $\mathbf{F.2} \to \mathbf{F.3}$ of the embedded FXMSS proof, and in the counter searches of the augmented reductions.

5 Discussion: Reduction Accounting and What the Bounds Certify

The theorems of this document reduce the security of the schemes to properties of the underlying hash functions, and every InSec term in Theorems 4.1 to 4.3 carries explicit resource parameters. Such terms guide the choice of parameters only if we track how the resources of a forger translate into the resources of the adversaries built from it against the underlying properties. This section examines that translation: it accounts for the resources of the reductions and discusses the consequences.

We argue that, read plainly, the accounting of the tight SPHINCS$^+$ proof of [HK22] is flawed, and that the straightforward repair gives a significantly weaker certified bound for the standardized $h = 63$ parameter set with $n = 128$, already in the classical instantiation (Section 5.3).

The response of the preceding sections is the collection-oracle accounting of Section 4.3 together with the transfer conjecture Conjecture 4.1. Here we justify that choice by comparing it with the straightforward repair, which charges all of the reductions' work as attack queries. We fix the resources a concrete statement counts and the cost of an attack on SHRINCS itself (Sections 5.1 and 5.2), restate the flaw in a self-contained way and name the two accountings (Section 5.3), and discuss why the adopted accounting is sound in the ideal model (Lemma 4.1, proved in Section D) and exhibit a degenerate collection for which it fails (Section 5.4). Section 6.8 compares the two accountings at our parameters.

5.1 Concrete Security and Its Resources

Concrete (or exact) security, going back to Bellare and Rogaway [BR96], replaces the asymptotic statement "if P is hard then the scheme is secure" by an explicit transformation of any forger with resources $(\xi, q_s, \ldots)$ and success probability ε into a primitive adversary with resources $(\xi', p, \ldots)$ and success probability ε'. Its practical content lies in this resource translation: a reduction that turns a time-ξ forger into a time-$2^{60}\xi$ primitive adversary certifies far less than a tight one, although both support the same asymptotic claim. The critiques of Koblitz and Menezes [KM07] are largely aimed at interpretations of provable-security claims that ignore this translation; this section is one more case study of that kind, arising this time from the theorem statements themselves.

The insecurities $\mathrm{InSec}^{\mathrm{Prop}}$ and their resources are defined in Section 4.3. The honest reading, which we adopt, is Rogaway's: theorems of this document are *explicit* reductions, and a bound of the form $\mathrm{InSec}^{\text{EU-CMA}} \le \sum_i \mathrm{InSec}^{\mathrm{Prop}_i}$ asserts that anyone who exhibits a forger thereby exhibits, constructively and with stated resources, a breaker of one of the listed properties. The InSec terms are placeholders, instantiated either by the generic bounds of Table 6, which are proven theorems about ideal (random) functions except the conjectured SM-DSPR row, or by cryptanalytic estimates for the concrete function.

We argue that a *relative* statement such as Conjecture 4.1 remains meaningful under this reading: it does not assert that any insecurity is small, only that a specific added interface does not move the maximum, and it is falsifiable for a concrete collection (Remark 5.1 exhibits a collection for which it fails).

What a reduction must deliver. The informal statement of the goal, that a forger with resources ξ and success ε be turned into a primitive adversary with resources close to ξ and success close to ε, asks for more than we need. Consider a theorem of the form

$$\mathrm{InSec}^{\text{EU-CMA}}(\xi) \;\le\; \sum_i \mathrm{InSec}^{\mathrm{Prop}_i}(\xi + t'_i).$$

What it certifies depends on how each $\mathrm{InSec}^{\mathrm{Prop}_i}$ *grows* with the extra resources t'_i. The size of t'_i on its own says nothing. If the bound we substitute for a term ignores some resource, the

reduction may spend as much of that resource as it likes and the bound does not move; if the bound is sensitive to a resource, even a small amount of it can be expensive. The analysis below gives an explicit treatment of the resources used, and presents which of them the reduction inflates, and how that affects security.

Under the accounting of Section 4.3 the resources separate. Local computation that evaluates no member of the collection is uncharged, and it is uncharged because no bound of Table 6 depends on it: these are statements about ideal functions in which success is bounded by the number of queries alone, so an adversary may spend unbounded machine time between its queries and gain nothing by it. The bookkeeping a reduction performs around its oracle calls (assembling signatures from oracle outputs, storing random keys, maintaining the tables of Lemma 4.1) may therefore be arbitrarily large while the certified bound stays the same.

The resource vocabulary. A concrete statement must first fix which resources it counts: the adversary's computation, its memory, its parallelism, and its precomputation. This document counts computation as in Section 4.3: the adversary's own hash evaluations, with separate counts for signing, challenge, collection, and random-oracle queries. Memory, parallelism, and precomputation are not charged.

We return to memory and to the verification of the forgery in Section 5.2; otherwise this section stays in the query-counting model, which is where the flaw of Section 5.3 arises.

5.2 What an Attack Costs

Before auditing the running time of the reductions (Section 5.3), we fix intuitions about the object being bounded: the cost of an attack on SHRINCS itself. The EU-CMA experiment (Definition 4.14) gives the adversary a key at no cost and a signing oracle at unit cost per query, and it charges only the adversary's own computation. The honest algorithms are still executed in real world: the signer generates the key and the signatures, and the verifier checks the forgery. Which of this work is charged to the forger is a choice, and a recent work [KWW26] makes the choice explicit. They define bit security relative to a *counting mode*: the set of query counts charged as the forger's work, chosen among the forger's own random-oracle queries, its signing queries, and the random-oracle queries made by key generation, by signing, and by the verification of the forgery. For two instances of XMSS they prove the exact level in every mode and give matching attacks. In the most conservative mode the attacks make at most one query of their own and rely on the evaluations of the verifier: an honest signature with one value replaced makes every later evaluation on the path to the root fresh, and each of them may hit the honest node. They sketch the same attacks for SLH-DSA and for SHRINCS [KWW26, Section 7]. Their analysis is in the classical random-oracle model only and does not cover quantum adversaries. We go through the honest work in turn, key generation and signing, the verification of the forgery, and memory, and for each we ask who performs it and who is charged for it.

Signing queries are not free for the world. Table 4 collects the honest computation costs of SHRINCS, counted in hash calls (SHA-256 compression evaluations agree with these figures up to small factors; each chain step, **PRF** call, and node hash is one call). The per-signature costs matter most for the present discussion. A stateless signature costs about $2^{20.7}$ hash calls, dominated by rebuilding one XMSS tree per hypertree layer. A stateful signature costs about $2^{8.4}$ for a signer that keeps the committed structure in memory; a signer that stores only the seed and the state counter rebuilds the structure on every signature, at the key-generation cost of about $2^{29.0}$ hash calls for BXMSS-20 and about $2^{17.0}$ for UXMSS. The stateful path is cheap in computation only where the signer pays in storage (about 32 MiB for BXMSS-20); the stateless path cannot make the same trade, since its virtual structure has 2^{45} hypertree leaves.

Consequently, an adversary that actually elicits the full stateless budget of $q_s = 2^{40}$ signatures has caused about 2^{61} hash evaluations, performed by the signer rather than by the adversary, but performed nonetheless. For the standardized "s" parameter sets of SPHINCS$^+$ with $q_s = 2^{64}$ the same arithmetic gives more than 2^{84} hash evaluations: an attack "of cost 2^{64}" that needs the

full signing budget comes with a factor of about 2^{21}. The stateful is simillar but at a smaller scale: exhausting all $L = 2^{20}$ anchors of BXMSS-20 costs the signer about $2^{28.4}$ hash calls with the stored tree, but about 2^{49} without it.

The unit-cost convention is nevertheless standard for the security definition, since signatures may be obtained from many signers, harvested from protocols, or accumulated over years, and none of that work is the adversary's own; the hash evaluations of signing lie outside both counting modes that this document uses. The comparison is still worth keeping in mind.

Table 4: Honest computation costs of SHRINCS in hash calls for the parameters of Table 1. Stateful key generation costs $l_{\mathsf{sf}}w + 1 = 513$ calls per anchor plus one per internal node; the stateless part costs $l_{\mathsf{sl}}w + 1 = 561$ calls per WOTS-TW instance of the top XMSS tree. Keeping the tree in memory takes $2L - 1$ nodes of 16 bytes (about 32 MiB for BXMSS-20, under 8 KiB for UXMSS); grinding takes $\approx 2^6$ expected attempts (Remark 4.4). $1 \le q \le 255$ is the signature counter in UXMSS of depth 255.

Operation	Hash calls	Dominant part
Kg, stateful part, BXMSS depth 20	$514 \cdot 2^{20} \approx 2^{29.0}$	$L = 2^{20}$ WOTS+C instances
Kg, stateful part, UXMSS depth 255	$\approx 2^{17.0}$	$L = 256$ WOTS+C instances
Kg, stateless part	$287{,}743 \approx 2^{18.1}$	top XMSS tree, 2^9 instances
Sign, stateful path, tree in memory	$\approx 2^{8.4}$	32 **PRF** + 240 chain + grinding
Sign, stateful path, seed only	$\approx 2^{29.0}$ (BXMSS-20) / $\approx 2^{17.0}$ (UXMSS)	committed-structure rebuild
Sign, stateless path	$\approx 2^{20.7}$	$d = 5$ XMSS tree rebuilds
Verify, stateful path	$242 + q$ (UXMSS) / 262 (BXMSS-20)	chain completion
Verify, stateless path	$\approx 2^{11.42}$	WOTS-TW chain completion

The verification of the forgery. The forger also chooses how much work the verification of its forgery causes. The verifier does not check the anchor depth $\delta^\star$ against any structure (Section 2.5), so a stateful forgery against a BXMSS-20 key may name the depth $h_{\mathsf{sf}} = 255$ and cause 497 tweakable-hash evaluations instead of the 262 of an honest signature (Table 4). Every generic bound of Table 6 is a function of the number of queries to the function in question, so we split the count by function.

Lemma 5.1 (Verification work)**.** *For a function f of the scheme, let V_{sf}^{f} (respectively V_{sl}^{f}) be the largest number of evaluations of f that one run of the verification algorithm (Algorithm 3) makes on a stateful (respectively stateless) signature. The maximum is over all public keys, messages, and signatures, and on the stateful branch over every anchor depth $\delta^\star \in \{1, \dots, h_{\mathsf{sf}}\}$: the signature chooses $\delta^\star$, and the verifier does not check it against any structure (Section 2.5). Neither path evaluates* **PRF** *or a randomizer function, so $V_{\mathsf{sf}}^{\mathrm{PRF}} = V_{\mathsf{sl}}^{\mathrm{PRF}} = 0$.*

- *Stateful path: $V_{\mathsf{sf}}^{\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}} = V_{\mathsf{sf}}^{\mathbf{H}_{\mathrm{grind}}} = V_{\mathsf{sf}}^{\mathsf{Th}_{l_{\mathsf{sf}}}} = 1$, $V_{\mathsf{sf}}^{\mathbf{F}} = l_{\mathsf{sf}}(w-1) - S_{wn}$, $V_{\mathsf{sf}}^{\mathbf{H}} = h_{\mathsf{sf}}$, and $V_{\mathsf{sf}}^{\mathsf{Th}_{2 \cup l_{\mathsf{sf}}}} = 1 + h_{\mathsf{sf}}$; at a fixed anchor depth $\delta^\star$ the same counts hold with h_{sf} replaced by $\delta^\star$. The tweakable-hash evaluations total*

$$V_{\mathsf{sf}} := l_{\mathsf{sf}}(w-1) - S_{wn} + 2 + h_{\mathsf{sf}},$$

  which is 497 at the parameters of Table 1 ($242 + \delta^\star$ at depth $\delta^\star$).

- *Stateless path: $V_{\mathsf{sl}}^{\mathbf{H}_{\mathbf{msg}}^{\mathsf{sl}}} = V_{\mathsf{sl}}^{\mathsf{Th}_k} = 1$, $V_{\mathsf{sl}}^{\mathbf{F}} \le k + d\, s_{\max}$, $V_{\mathsf{sl}}^{\mathbf{H}} = ka + dh'$, and $V_{\mathsf{sl}}^{\mathsf{Th}_{l_{\mathsf{sl}}}} = d$. Here $s_{\max}$ is the largest chain work $\sum_i (w-1-b_i)$ of a WOTS-TW encoding. The tweakable-hash evaluations total at most*

$$V_{\mathsf{sl}} := k(a+1) + 1 + d\,(s_{\max} + h' + 1).$$

At the parameters of Table 1, $s_{\max} = 510$, attained only at checksum value 480, so $V_{\mathsf{sl}}^{\mathbf{F}} \le 2560$, $V_{\mathsf{sl}}^{\mathbf{H}} = 175$, $V_{\mathsf{sl}}^{\mathsf{Th}_{l_{\mathsf{sl}}}} = 5$, and $V_{\mathsf{sl}} = 2741$. The bound on $V_{\mathsf{sl}}^{\mathbf{F}}$ is attained only if all d layer messages encode with checksum 480; we use it as an upper bound only.

Proof. Stateful path. The verifier evaluates $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ once for the digest (Section 3) and $\mathbf{H}_{\mathrm{grind}}$ once for the digit vector (Section 2.4). It rejects a digit vector whose sum differs from S_{wn} before any chain step. Otherwise it completes the chains with $\sum_i (w-1-a_i) = l_{\mathsf{sf}}(w-1) - S_{wn}$ calls to $\mathbf{F}$, which is independent of the message, and compresses the chain ends with one call to $\mathsf{Th}_{l_{\mathsf{sf}}}$. The root recovery of FXMSS.pkFromSig makes $\delta^\star \le h_{\mathsf{sf}}$ calls to $\mathbf{H}$. $\mathsf{Th}_{2\cup l_{\mathsf{sf}}}$ coincides with $\mathbf{H}$ on node tweaks and with $\mathsf{Th}_{l_{\mathsf{sf}}}$ on compression tweaks (the instantiation before Definition 4.7), so it is evaluated $1+\delta^\star$ times.

Stateless path. The verifier evaluates $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sl}}$ once. FORS.pkFromSig makes one call to $\mathbf{F}$ and a calls to $\mathbf{H}$ per tree, and one call to Th_k. Each of the d layers makes one call to WOTS-TW.pkFromSig, which makes $\sum_i (w-1-b_i)$ calls to $\mathbf{F}$ and one to $\mathsf{Th}_{l_{\mathsf{sl}}}$, and h' calls to $\mathbf{H}$ on the authentication path. With $C = \sum_{i<l_1}(w-1-a_i) \in \{0,\dots,l_1(w-1)\}$ the checksum of Section 2.3, the chain work of a layer is $C + \sum_{j<l_2}(w-1-C_j)$, a function of C alone, with the maximum 510, at $C = 480$. □

In the theorems this work belongs to the reductions: each collection reduction of Section 4 except the S-TCR(Prop) reduction of Claim 4.3 recomputes the verification of the forgery in its second stage, after the public parameter is revealed, and the theorems state these evaluations as the additive V_{sf} and V_{sl} in the times of the collection terms, an upper bound for the exception. The ITSR and SM-rTCR reductions have no second stage: they count the one evaluation of the message hash for the verification in $\tilde{q}_h \le q_h + 1$ and discharge their other evaluations by Lemma 4.2. The reprogramming distinguishers have no second stage either; their query that verifies the forgery is the $+1$ in the query count $q_s + q_h + 1$ of their class in Corollary C.3.

How the verification of the forgery is charged changes what a bound certifies when the forger makes few queries. We usconsider the two of the modes of [KWW26].

- *Verification free to the forger.* This is their most conservative mode, which charges only the forger's own queries, and it is the mode this document aims at. The time ξ counts the forger's own evaluations (Section 4.3), and the verification stays work of the reductions, as above. A classical bound then has the form $\xi/2^Y + c$, and the verification enters the constant c only (Section 6.4).

- *Verification counted as the forger's work.* Here the evaluations of the verification are added to the forger's work. This mirrors the signing convention from the other side: the signatures a forger uses may come from anywhere, but the verification of its forgery is caused by the forgery itself, and no forgery is complete before it is verified.

Memory. Our InSec terms carry no memory parameter. For the forger this costs nothing in the ideal model: the generic bounds of Table 6 for the tweakable-hash properties contain no challenge count p, so an adversary that records many targets does, generically, no better than one that keeps a single target; the keyed message-hash properties do carry p, but there p is the signing budget rather than a structure-scale count.

For the reduction the picture is a bit different: it stores not the targets it attacks but the structure itself. The reductions of this document assemble the complete structure from challenge and collection oracle values before the forger starts the attack. Unlike the honest signer, who regenerates everything from its seed, a reduction cannot recompute what the challenger sampled: the challenge oracle issues each target once. For FXMSS the requirement is modest: the $514L$ structure values occupy about 8 GiB for BXMSS-20 and 2 MiB for UXMSS. For the stateless part the reduction must store the complete virtual structure: about 2^{54} values for the WOTS-TW chains alone and about 2^{62} with the FORS trees, i.e., hundreds of petabytes to tens of exabytes.

The reduction's memory therefore grows with the size of the structure and independent of the resources of the forger; Section 5.3 shows that the same holds for its running time. For the bounds of this document this is acceptable, since the target problems are not memory-sensitive. An instantiation of the InSec terms by a memory-sensitive assumption, or by a cost model that charges local computation, would have to revisit this point (see Remark D.1).

5.3 The Runtime in the SPHINCS$^+$ Accounting

We now restate the misaccounting that motivates this discussion. The tight SPHINCS$^+$ theorem of [HK22] is assembled from a multi-instance WOTS-TW theorem. The WOTS-TW theorem, for D one-time instances, hands its reductions the time budget

$$\tilde{\xi} = \xi + D\ell w,$$

where ξ is the forger's time: the reductions build the public keys of all D instances, and completing the hash chains of one instance costs ℓw, so the additional work is genuinely performed. The top-level theorem, however, quotes all four WOTS-TW primitive terms at the forger's time ξ. The printed same-time statement is not established by the written proof.

The natural repair is to replace ξ by $\tilde{\xi}$ in the affected terms. This, however, weakens the certified bound, already for classical adversaries. At the $h = 63$ parameter sets of FIPS 205 [Nat24] the complete virtual hypertree contains $D > 2^{63}$ WOTS-TW instances, so $\tilde{\xi} > D\ell w > 2^{72}$, and the corrected SM-UD summand, with its coefficient $w = 16$ and the classical bound of Table 6, evaluates to $w(\xi + D\ell w)/2^n \approx 2^{-52} + \xi/2^{124}$ for $n = 128$ (SLH-DSA-128s). The consequence is easiest to see at $\xi = 0$: for a forger that makes no hash evaluations of its own, the corrected theorem certifies only 2^{-52} at $n = 128$. The adversary's own queries, however, enter only through the term $\xi/2^{124}$, so driving the success probability toward 1 still requires on the order of 2^n evaluations. Counting also the FORS instances that the enclosing SPHINCS$^+$-level reduction builds (Theorem 4.3), $2^{63} \cdot k \cdot 2t \approx 2^{80}$ calls at SLH-DSA-128s ($k = 14$, $t = 2^{12}$), raises the floor of this single summand to $w \cdot 2^{80}/2^{128} \approx 2^{-44}$. The quantum case needs a separate treatment, which we do not attempt here. Neither observation demonstrates an attack on SPHINCS$^+$. These observations only state that a more careful reading of the theorem statement is needed.

Note that the loss in security is *structurally harmless*: the $D\ell w$ evaluations complete hash chains at tweaks disjoint from every challenge tweak, and the supporting argument of [HK22] already discussed that for a random tweakable function such evaluations reveal nothing about the targets. The same argument applies to most of the other properties. The accounting problem is that the time-parameterized notion cannot see this distinction: once the work is charged to $\tilde{\xi}$, the notion must tolerate an adversary who spends the same budget *at* the challenge tweaks, against the targets themselves.

SHRINCS inherits the same problem. Its stateless path rests on the SPHINCS$^+$ proof of [HK22], whose reductions complete the entire virtual structure, which is why Theorem 4.3 restates that bound under the accounting of Section 4.3 instead of quoting it as printed. Our own FXMSS reductions have the same completion-work pattern: the budgets $q_{\lambda,i}$ of Theorem 4.2 consist of the committed structure, the stateless key generation, and the grinding budget q_g (Section 6.8 gives the magnitudes). A full accounting of the stateless term would carry $q_\lambda \approx 2^{62}$, dominated by the FORS trees (cf. the note at Theorem 4.3). With the classical generic bounds and our parameters, this full accounting changes no coefficient of ξ and adds only a constant, about $2^{-60.0}$ on the stateless path (Section 6.8). The solution adopted in the preceding sections is to refine the notion: the collection oracle of Definition 4.12 gives the harmless work its own metered interface, disjoint from the challenge tweaks by construction, and Conjecture 4.1 asserts that this interface does not move the insecurity. The PRF terms sit outside this machinery: the generic PRF bounds count only the adversary's evaluations of the challenged public family, the hidden-key oracle queries are provably worthless in the idealized model, and the reductions of this document never evaluate the challenged family themselves (Definition 4.3 and Remark 4.9). The

message-hash terms reductions hold the public parameter from the start and perform the honest key generation and signing themselves (the times t_{sim} and $t^{\mathsf{sl}}_{\mathsf{sim}}$ of Theorems 4.2 and 4.3). The message hash is modeled as a random oracle independent of the other functions (Assumption 4.1), Lemma 4.2 shows that such evaluations do not move the insecurity.

We name the two accountings, since Section 6 evaluates the bounds under both.

Definition 5.1 (Accounting conventions)**.** Fix a reduction that runs a forger with time ξ and performs q_λ evaluations of the collection at non-challenge tweaks, plus t_{own} further evaluations of its own. We call the following two instantiations of a primitive term:

- *Convention I (interface-separated; used in this document).* The term is $\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda; \xi', p, q_\lambda)$ with $\xi' = \xi + t_{\text{own}}$, and a generic bound from Table 6 is applied with query count $q := \xi'$[12], justified by Conjecture 4.1 via Corollary 4.1; for classical and quantum adversaries, Lemma 4.1 establishes the conjecture in the ideal model up to its additive term.

- *Convention M (monolithic).* Nothing is assumed to be harmless: a generic bound is applied with the total evaluation count $q := \tilde{\xi} = \xi + t_{\text{own}} + q_\lambda$, i.e., every evaluation the constructed adversary causes, through whatever interface, is treated as an attack query.

5.4 When Is the Collection Oracle Harmless?

The intuition behind Conjecture 4.1 is the following two-oracle picture. Suppose an adversary must find a preimage under a random oracle G_1 and is additionally given an independent random oracle G_2. Queries to G_2, even a million, or 2^{128}, cannot help: the joint distribution of the challenge and of everything the adversary sees through G_1 is unchanged whether G_2 exists or not, so any success probability achievable with G_2 is achievable without it, for example by simulating G_2 with random coin tosses. For a random tweakable function, distinct tweaks mimic such independent oracles, and the collection oracle, restricted to tweaks disjoint from the challenge tweaks, follows the same idea.

Harmlessness is thus a property of *where* the work happens: the same 2^{60} budget spent at the challenge tweaks erodes preimage resistance classically, and quantum queries would amplify the erosion quadratically. An accounting that leaves the simulation work uncounted is therefore sound only together with an interface that *certifies* the work as harmless, and in Definition 4.12 the certificate is the tweak-reservation rule.

Remark 5.1 (Ideality is essential)**.** Lemma 4.1 is a statement about ideal collections, and its proof leans on the independence of the per-tweak functions. For a concrete collection, Conjecture 4.1 is a real assumption, and it is important to mark that it can *fail*: the following construction, degenerate by design, satisfies the syntax of Definition 4.12 while making collection queries as valuable as challenge queries. Intuitively, we make every tweak the *same* oracle. Let $G_m \colon \mathcal{P} \times \{0,1\}^m \to \{0,1\}^n$ be independent random functions, one per message length, and define the *tweak-blind* collection $\mathsf{Th}^\flat_\lambda = \{\mathsf{Th}^\flat_m\}$ by

$$\mathsf{Th}^\flat_m(P, T, x) \;=\; G_m(P, x) \qquad \text{for all } T \in \mathcal{T}.$$

Tweak reservation now protects nothing: the collection oracle at any unreserved tweak evaluates the same function that the challenges live on. Collection queries now help against the properties directly. About $2^{n/2}$ of them at a single unreserved tweak find, by the birthday bound, a pair

[12] For the two-stage properties, the $|\mathcal{P}|$ summands of Table 6 count only the evaluations made before the public parameter is revealed (see its caption). We read the collection terms accordingly: the maximum in InSec ranges over the adversaries of time ξ' that make at most ξ of their own evaluations in the first stage, and the $|\mathcal{P}|$ summands are applied with $q := \xi$. The reductions of Section 4 meet this, since they verify the forgery, if at all, in their second stage (Section 5.2).

$x \neq x'$ with $G_m(P, x) = G_m(P, x')$; since every tweak evaluates the same function, challenging x at a fresh tweak afterwards turns the pair into a target collision, and SM-TCR of a member of $\mathsf{Th}^{\flat}_{\lambda}$ falls at birthday cost with no own evaluations at all.

Conjecture 4.1 is a statement about the *concrete* collection. For the SHRINCS instantiation, every hash call absorbs its address (Figs. 5 to 11), so distinct tweaks produce distinct SHA-256 inputs, and the per-tweak functions are as independent as SHA-256 makes them; a violation of the conjecture would be a related-input distinguisher for SHA-256 of a rather specific shape. Lemma 4.3 and Assumption 4.1 record this modeling step for the whole scheme.

6 Concrete Security of the Proposed Parameters

The theorems of Section 4 bound the forging probability by a sum of terms. This section turns these sums into numbers for the parameters of Table 1, using the generic bounds of Table 6 and the sharper bounds of the appendices, instantiated in Section 6.5, all of which hold when the hash functions are modeled as (quantum) random oracles. We do this for the two paths separately (Theorems 4.2 and 4.3) and for the full scheme (Theorem 4.1), for classical and for quantum adversaries. The figures rest on three conjectures. Every bound used is proven except the SM-DSPR row of Table 6, which is conjectured (Section 7) and enters the stateless and full-scheme figures. The stateful and full-scheme figures assume Conjecture 3.1, as Theorems 4.1 and 4.2 do. The figures computed under Convention I (Section 6.3) assume Conjecture 4.1; those of Section 6.8, which charge the collection work, do not.

Reprogramming terms and core terms. Three summands originate in one proof step, the reprogramming of the message digests: the randomizer part of the ITSR bound, the randomizer part of the SM-rTCR bound, and the reprogramming hop of Claims 4.2 and 4.14. We call them the *reprogramming* terms; Section 6.5 explains the step and instantiates their bounds. Every other summand we call a *core* term: the PRF terms, the search part of ITSR, the hash part of SM-rTCR, the collection terms of the tweakable hash functions, and, for the full scheme, $\varepsilon_{\mathrm{ds}}$. We separate the two kinds because the bound over the core terms alone limits what a sharper treatment of the reprogramming step could still gain: less than 0.1 bits classically, and no $\xi^\star$ quantumly, where the minimum over the forms already attains the core value; only the value of the bound at $\xi = 0$ could still drop. It also isolates the comparison of Remark 6.1 with the earlier reprogramming lemmas.

6.1 Summary of Results

The classical bounds are affine in ξ up to the pull part of Theorem F.1, and we report them as $\xi/2^Y$ plus an additive constant; Y is the bit security in the query-counting sense of Section 5.1. The quantum bounds contain $\sqrt{\xi}$, ξ, and ξ^2 terms, so no single Y describes them; we report their form, their value at $\xi = 0$, and the query count $\xi^\star$ at which the bound reaches 1, which is the quantum analogue of Y. Table 5 reports every bound by $\log_2 \xi^\star$; for a classical bound $\xi/2^Y + c$ with small c this is Y.

Table 5 states the bounds with all terms; the rest of the section derives them. Classically, every bound certifies between 121.4 and 122.7 bits. The loss against the 128-bit hash is the sum of the coefficients of the core terms, which alone give 122.23, 122.83, and 121.50 bits on the stateless path, the stateful path (either shape), and the full scheme; the reprogramming terms add less than 0.1 bits on every path.

Quantumly, every bound with $e = 1$, and the minimum over the forms, reaches 1 at $\xi^\star = 2^{58.0}$ to $2^{59.2}$. The core terms set this value through two kinds of summands: linear summands $\xi/2^{n/2}$, and quadratic summands $\xi^2/2^n$. The quantum reprogramming terms are bounded with the lemma of Liao, Ge, Xue, and Zhou [LGXZ26] (Lemma C.3), in one of two forms, $e = 1$ or $e = 1/2$. These terms matter mainly for a forger that makes few hash queries, so we report two numbers:

$\xi^\star$, and the value of the bound at $\xi = 0$. With $e = 1$, $\xi^\star$ keeps its core value, and the bound at $\xi = 0$ can reach $2^{-22.4}$. With $e = 1/2$, the bound at $\xi = 0$ drops to at most $2^{-42.9}$, but $\xi^\star$ loses 0.8 bits on the stateful path and 0.9 bits for the full scheme.[13]

			Quantum		
Bound	Shape	Classical	$e = 1$	$e = 1/2$	minimum
Theorem 4.3 (stateless)	—	122.23 (−111.4)	58.5 (−22.4)	58.5 (−43.9)	58.5 (−43.9)
Theorem 4.2 (stateful)	BXMSS-20	122.75 (−114.2)	59.2 (−41.4)	58.4 (−82.9)	59.2 (−82.9)
	UXMSS-256	122.75 (−114.2)	59.2 (−53.4)	58.4 (−101.3)	59.2 (−101.3)
Theorem 4.1 (full)	BXMSS-20	121.47 (−111.2)	58.0 (−22.4)	57.1 (−42.9)	58.0 (−42.9)
	UXMSS-256	121.47 (−111.2)	58.0 (−22.4)	57.1 (−42.9)	58.0 (−43.9)

Table 5: All terms, with $q_s^{\mathsf{sl}} = 2^{40}$ and $q_s^{\mathsf{sf}} = L$: $\log_2 \xi^\star$, where $\xi^\star$ is the hash-query count at which the bound reaches 1, and in parentheses $\log_2$ of the bound at $\xi = 0$, with the verification of the forgery charged and $\varepsilon_{\mathrm{ds}}$ included (Section 6.3). Classical reprogramming terms bounded as in Section 6.5. Quantum reprogramming terms bounded with Lemma C.3 ($e = 1$) or Corollary C.1 ($e = 1/2$); "minimum" is the pointwise minimum over all combinations of the two forms at the three places of Section 6.5. Closed forms: Claims 6.1, 6.2 and 6.4. Each $\log_2 \xi^\star$ is rounded down and each $\log_2$ of a value at $\xi = 0$ up, so every entry is a bound (for the stateless and full-scheme rows, under the conjectured SM-DSPR row of Table 6); classical $\log_2 \xi^\star$ is given to two decimals, all other entries to one.

Remark 6.1 (The bounds with earlier results)**.** We compare Table 5 with four earlier bounds that it replaces. Classically, Theorem F.1 replaces Theorem F.2 for ITSR, and the bounds of Section 6.5 replace Lemma C.2 at the randomizer part of SM-rTCR and at the reprogramming hop. Quantumly, Lemma C.3 replaces Lemma C.1 [GHHM21] at the three places of Section 6.5, and the quadratic SM-UD and SM-PRE bounds of Table 6 replace the earlier bounds of its ‡ note. The gains do not add up, so we switch one result back at a time and keep the other three; entries are in the notation of the table, and quantum gains are taken against the column "minimum" [14].

- With Theorem F.2, the classical entry is 88.00 (−48.0) on the stateless path and for the full scheme with either shape, so Theorem F.1 adds 34.23 and 33.47 bits. The stateful path does not use it.
- With Lemma C.2 at the randomizer part of SM-rTCR and at the hop, the classical entries are 107.00 (−87.0) and 118.90 (−110.9) on the stateful path with BXMSS-20 and UXMSS-256, and 107.00 (−87.0) and 118.77 (−110.1) for the full scheme, so the bounds of Section 6.5 add 15.75 and 3.85 bits on the stateful path and 14.47 and 2.70 bits for the full scheme. The stateless path has neither term.
- With [GHHM21] at the three places, the quantum entries are 46.8 (−3.4) on the stateless path and for the full scheme with either shape, and 59.3 (−32.4) and 59.3 (−50.4) on the stateful path with BXMSS-20 and UXMSS-256. Lemma C.3 thus adds 11.7 bits of $\xi^\star$ on the stateless path and 11.2 bits for the full scheme, where it also removes the ξ-independent floor of $2^{-3.4}$; on the stateful path it leaves $\xi^\star$ unchanged up to 0.001 bits and lowers the values at $\xi = 0$.

[13] With the constants $2\sqrt{2}$ and 16 of Remark C.3 in place of 4 and 32, every quantum value at $\xi = 0$ in Table 5 would be about 1 bit lower in the columns $e = 1/2$ and "minimum", and about 0.5 bits lower in the column $e = 1$, except the stateful path with UXMSS-256 in the columns $e = 1/2$ and "minimum", where it would be 0.01 bits lower; no $\xi^\star$ and no classical entry would change (Section 7).

[14] The entries of the earlier bounds are rounded to nearest; the gains are computed from the exact values and rounded down.

- With the earlier SM-UD and SM-PRE bounds, the quantum entries are 56.1 (−43.4) on the stateless path, 56.2 (−46.4) and 56.2 (−47.4) on the stateful path with BXMSS-20 and UXMSS-256, and 55.1 (−42.3) and 55.1 (−43.3) for the full scheme, so the quadratic bounds add 2.4, 3.0, and 2.8 bits of $\xi^\star$.

With all four results switched back, the classical entries are 88.00 (−48.0) on the stateless path and for the full scheme with either shape, and 107.00 (−87.0) and 118.90 (−110.9) on the stateful path with BXMSS-20 and UXMSS-256; the quantum entries are 46.8 (−3.4) on the stateless path and the full scheme, and 56.2 (−32.4) and 56.2 (−47.2) on the stateful path with BXMSS-20 and UXMSS-256.

6.2 Generic Security

In this subsection, we collect the generic bounds for the hash function properties used in the security proof of SHRINCS. Table 6 lists, for each property, an upper bound on the success probability (or advantage) of classical and quantum adversaries when the underlying hash function is modeled as a classical or quantum random oracle, respectively.

	Classical Bound	Quantum Bound	Reference
$\mathsf{Adv}^{\mathrm{PRF}}_{F,p}(\mathcal{A})$	$\frac{q}{\lvert\mathcal{K}\rvert}$	$\frac{2q}{\sqrt{\lvert\mathcal{K}\rvert}}$	[BHK+19], [SXY17, Lemma 2.2], [XY18, Lemma 2.1]
$\mathrm{Succ}^{\text{SM-TCR}}_{\mathbf{Th},p}(\mathcal{A})$	$\frac{2q+1}{\lvert\mathcal{N}\rvert}+\frac{2q}{\lvert\mathcal{P}\rvert}$	$\frac{32(q+1)^2}{\lvert\mathcal{N}\rvert}+\frac{4q}{\sqrt{\lvert\mathcal{P}\rvert}}$	[HK22]†
$\mathrm{Succ}^{\text{SM-PRE}}_{\mathbf{Th},p}(\mathcal{A})$	$\frac{q+1}{\lvert\mathcal{N}\rvert}+\frac{q+1}{\lvert\mathcal{M}\rvert}$	$\frac{8(q+1)^2}{\lvert\mathcal{N}\rvert}+\frac{4(q+1)^2}{\lvert\mathcal{M}\rvert}$	[Kud25, Theorem 6.2.10], [Kud26b]‡
$\mathsf{Adv}^{\text{SM-UD}}_{\mathbf{Th},p}(\mathcal{A})$	$\frac{q}{\lvert\mathcal{M}\rvert}$	$\frac{4q^2}{\lvert\mathcal{M}\rvert}$	[DSS05, Hül13, HK22], [Kud26b]‡
$\mathrm{Succ}^{\text{SM-TCR-EXCL}}_{\mathbf{Th},p}(\mathcal{A})$	$\frac{q+1}{\lvert\mathcal{N}\rvert}+\frac{q}{\lvert\mathcal{P}\rvert}$	$\frac{8(q+1)^2}{\lvert\mathcal{N}\rvert}+\frac{2q}{\sqrt{\lvert\mathcal{P}\rvert}}$	Section E
$\mathrm{Succ}^{\text{S-TCR(+C)}}_{\mathbf{Th},p}(\mathcal{A})$	$\frac{2q+1}{\lvert\mathcal{N}\rvert}+\frac{2q}{\lvert\mathcal{P}\rvert}$	$\frac{32(q+1)^2}{\lvert\mathcal{N}\rvert}+\frac{4q}{\sqrt{\lvert\mathcal{P}\rvert}}$	[HKRY23], [Kud25, Section 6.2.4]†
$\mathsf{Adv}^{\text{SM-DSPR}}_{\mathbf{Th},p}(\mathcal{A})$	$\frac{2q+1}{\lvert\mathcal{N}\rvert}$	$\frac{32(q+1)^2}{\lvert\mathcal{N}\rvert}$	Conjectured [BHK+19, BH19]
$\mathrm{Succ}^{\mathsf{SM\text{-}rTCR},K}_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A})$, $e=1$	$\frac{q+1}{\lvert\mathcal{N}\rvert}+\frac{Kq+pK(K-1)/2}{\lvert\mathcal{R}\rvert}$	$\frac{8(q+1)^2}{\lvert\mathcal{N}\rvert}+4\sqrt{\frac{S'}{\lvert\mathcal{R}\rvert}}$	Corollary G.1, Theorem G.1
$\mathrm{Succ}^{\mathsf{SM\text{-}rTCR},K}_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A})$, $e=1/2$		$\frac{16(q+1)^2}{\lvert\mathcal{N}\rvert}+\frac{32\,S'}{\lvert\mathcal{R}\rvert}$	
$\mathrm{Succ}^{\text{ITSR}}_{\mathrm{H},p}(\mathcal{A})$, $e=1$	$(q+1)\Gamma+P_{h,k,t}(q,p)$	$4\sqrt{\frac{S}{\lvert\mathcal{K}\rvert}}+8(q+1)^2\Gamma$	Theorem F.1 and Remark F.1, Theorem F.3
$\mathrm{Succ}^{\text{ITSR}}_{\mathrm{H},p}(\mathcal{A})$, $e=1/2$		$\frac{32\,S}{\lvert\mathcal{K}\rvert}+16(q+1)^2\Gamma$	

Table 6: Generic bounds in the classical and quantum random-oracle models for a tweakable hash function $\mathbf{Th}\colon \mathcal{P}\times\mathcal{T}\times\mathcal{M}\to\mathcal{N}$ and keyed functions F and H with key space $\mathcal{K}$; q counts hash queries and p challenge queries. For SM-rTCR, $\mathcal{M}=\mathcal{M}_0\times\mathcal{R}$ with K hash queries per challenge; SM-DSPR is conjectured for length-preserving functions; $\Gamma:=\Gamma_{h,k,t}(p)$ (Lemma F.1), and $P_{h,k,t}$ is the pull part of Theorem F.1, which needs equally many preimages per digest under $\mathrm{MAP}_{h,k,t}$ (Remark F.1), as in SHRINCS. In the rows $e=1$ and $e=1/2$ the quantum bound uses Lemma C.3 or Corollary C.1 and holds for halting adversaries (Remark C.1), with $S:=pq+p(p-1)/2+1$ and $S':=pKq+pK(pK-1)/2+1$; the classical bound does not depend on e. For the two-stage properties, q counts the queries of both stages in the $|\mathcal{N}|$ and $|\mathcal{M}|$ summands and only those of the first stage in the $|\mathcal{P}|$ summands (Lemmas C.7 and C.8); the quantum bounds assume bit-string output spaces with XOR oracles (Remark C.2), as in SHRINCS. †Corrected (Remark C.7): the cited proofs use the ZeroDist bound of [HRS16], which does not hold; we apply Lemma C.8 to their ZeroDist adversaries, which make $2q$ queries. ‡Earlier quantum bounds: $12q/\sqrt{|\mathcal{M}|}$ for SM-UD [HK22, Lemma 3], used in the proof of SPHINCS$^+$, and $8(q+1)^2/|\mathcal{N}|+12(q+1)/\sqrt{|\mathcal{M}|}$ for SM-PRE [Kud25, Theorem 6.2.10].

6.3 Conventions

Resources. Following Section 4.3, ξ is the forger's own hash-evaluation count. Every bound of Table 6 and Section 6.5 is a function of the number q of queries to the hash function in question, and we instantiate each of them with $q:=\xi$: the forger may spend its entire budget on the

one function that the term concerns. The challenge counts are those of the theorems. On the stateless path the ITSR term has $p = q_s^{\mathsf{sl}} = 2^{40}$, the signing budget the parameter set is calibrated for. On the stateful path the SM-rTCR term and the reprogramming hop have $p = q_s^{\mathsf{sf}} = L$ and $K = 1$, and we take the two prescribed shapes BXMSS-20 ($L = 2^{20}$) and UXMSS-256 ($L = 2^8$). The remaining terms contain no p, so their challenge counts $q_1, \ldots, q_8$ do not affect the numbers.
Reduction work. We follow Convention I of Definition 5.1. The collection budgets $q_{\lambda,i}$ of the reductions are not added to ξ (Corollary 4.1), but the reductions' own evaluations are. These are the verification of the forgery, which every collection reduction but the S-TCR(Prop) one of Claim 4.3 recomputes; the theorems charge it to all collection terms (Lemma 5.1). Each collection term is therefore instantiated at $q := \xi + V$, with the tweakable-hash evaluations $V = V_{\mathsf{sf}} = 497$ of the verification on the stateful path, at the worst anchor depth $\delta^\star = h_{\mathsf{sf}} = 255$, and $V = V_{\mathsf{sl}} = 2741$ on the stateless path (Lemma 5.1). These are the times $\tilde{\xi}_{\mathsf{sf}}$ and $\tilde{\xi}_{\mathsf{sl}}$ of the collection terms in Theorems 4.1 to 4.3, which count the evaluations of every member of Th_λ (Section 4.3).[15]

We count the verification as free to the forger, the conservative mode of Section 5.2. Three kinds of summands receive no charge: the PRF terms, since the verification evaluates no PRF; the reprogramming hop, since the final verification query comes after the last reprogramming call and enters no reprogramming count (Remark C.6 classically, Corollaries C.3 and 4.2 quantumly); and the $|\mathcal{P}|$ summands of the SM-TCR-type rows, $2q/|\mathcal{P}|$ and $q/|\mathcal{P}|$ classically and those of scale 2^{-64} quantumly, since Lemmas C.7 and C.8 count only the queries of the first stage, before the public parameter is revealed, the second stage of ZeroDist having no oracle (Definition C.4), and the verification runs in the second stage (Section 6.4). Charging the hop as well would change no entry of the tables. The verification changes no $\xi^\star$ of the tables; it raises the values at $\xi = 0$, and Section 6.4 gives the added constant exactly.

The reductions behind the message-hash terms (ITSR, SM-rTCR, and the reprogramming hop) run the honest key generation and signing, and they make $\tilde{q}_h \le q_h + 1 \le \xi + 1$ queries to the message-digest function. The time of key generation and signing does not enter these terms: by Lemma 4.2, they depend on the reduction only through its query parameters. The extra query is the evaluation of the message digest in the verification, $V^{\mathbf{H}_{\mathbf{msg}}} = 1$, and we instantiate the ITSR and SM-rTCR terms at $q := \xi + 1$; the hop stays at $q := \xi$, as above. The full-scheme numbers include the modeling charge $\varepsilon_{\mathrm{ds}} \le 2^{-128} + 10 \cdot 2^{-256}$ of Theorem 4.1 (Assumption 4.1).
Spaces. Every space has size $2^n = 2^{128}$: the public parameter, key, message, and output spaces of the tweakable hash functions, the keys of $\mathbf{PRF}$, $\mathbf{PRF}_{\mathbf{msg}}^{\mathsf{sl}}$, and $\mathbf{PRF}_{\mathbf{msg}}^{\mathsf{sf}}$, and the randomizer space $\mathcal{R}$ of both message-digest functions. The one exception is the output of $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$, which is $2n = 256$ bits (Section 2.2), so $|\mathcal{N}| = 2^{256}$ in the hash part of the SM-rTCR term. The ITSR terms use $\Gamma = \Gamma_{45,10,2^{13}}(2^{40}) \le 2^{-130.23}$ (Lemma F.1).

6.4 Additive Constants and the Verification of the Forgery

Every classical bound of Claims 6.1 and 6.2 has the form $\xi/2^Y + c$, up to the term ν of scale 2^{-130} and below (Section 6.1). The tables of this section include the verification of the forgery in c (Section 6.3). We now give c exactly, in the mode of Section 5.2 in which the verification is free to the forger, the conservative mode the tables use; if the verification is instead counted as the forger's work, the same bound is at most $(\xi + V + 1)/2^Y + c_0$ (Section 5.2).
The two parts of the constant. We write $c = c_0 + c_V$. The part c_0 is the value at $\xi = 0$ without the verification, made of the constants of the generic bounds. On the stateful path it is $c_{0,\mathsf{sf}} = 7/2^{128} + 2^{-256} \le 2^{-125.19}$: one unit from each of the S-TCR(Prop) row, the three SM-TCR rows, and the SM-TCR-EXCL row, two from the SM-PRE row, and 2^{-256} from the hash part of the

[15] Charging each term only the evaluations of its own function, V^f (Lemma 5.1), would lower the classical values at $\xi = 0$ in Table 5 to $2^{-111.7}$, $2^{-115.5}$, and $2^{-111.6}$, and the quantum UXMSS-256 value with $e = 1/2$ and the minimum to $2^{-103.7}$, and change nothing else; it needs a per-member reading of ξ, which Conjecture 4.1 does not make.

SM-rTCR term. On the stateless path it is $c_{0,\mathsf{sl}} = 11/2^{128} + \Gamma \le 2^{-124.51}$: one unit from each of the five grouped SM-TCR summands, three from the q_8 summand, two from SM-PRE, one from SM-DSPR, and Γ from the search part of ITSR; the pull part vanishes at $q = 0$.

The part c_V charges the verification. Each collection term is instantiated at $q := \xi + V$ instead of $q := \xi$, with $V = V_{\mathsf{sf}}$ or V_{sl} from Lemma 5.1 (Section 6.3). The exception are its $|\mathcal{P}|$ summands, which stay at $q := \xi$. If the other summands of the term grow by $a/2^{128}$ per query, the term therefore grows by $aV/2^{128}$. Three kinds of terms are charged differently. The PRF terms receive nothing, since $V^{\mathrm{PRF}} = 0$. The reprogramming hop receives nothing classically, since the final verification query comes after the last reprogramming call (Remark C.6). The terms in the message hash receive its one evaluation: one unit for the randomizer part of SM-rTCR and 2^{-256} for its hash part, and on the stateless path Γ for the search part of ITSR and $P_{45,10,2^{13}}(1, 2^{40}) < 2^{-255.9}$ for the pull part. On the stateful path the collection terms have the slope 36 (Section 6.6), of which 9 units come from the $|\mathcal{P}|$ summands and are not charged, and the verifier accepts every anchor depth $\delta^\star \le h_{\mathsf{sf}}$ whatever the structure, so $V_{\mathsf{sf}} = 497$ at the worst case $\delta^\star = 255$ applies to both shapes:

$$c_{V,\mathsf{sf}} = \frac{27 \cdot 497 + \underbrace{1}_{\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}}}{2^{128}} + 2^{-256} = \frac{13420}{2^{128}} + 2^{-256} \le 2^{-114.28}.$$

On the stateless path the collection terms have the slope 52 (Section 6.6, without the PRF terms), of which 16 units come from the $|\mathcal{P}|$ summands, and

$$c_{V,\mathsf{sl}} = \frac{36 \cdot 2741}{2^{128}} + \Gamma + P_{45,10,2^{13}}(1, 2^{40}) = \frac{98676}{2^{128}} + \Gamma + P_{45,10,2^{13}}(1, 2^{40}) \le 2^{-111.40}.$$

Charging each term only the evaluations of its own function would give 5571 and 77162 units instead (footnote in Section 6.3). On the full scheme each term receives the verification of its own path, so $c_V = c_{V,\mathsf{sf}} + c_{V,\mathsf{sl}} = 112096/2^{128} + \ldots \le 2^{-111.22}$, with $c_0 = 18/2^{128} + \Gamma + 2^{-256} \le 2^{-123.81}$ and $\varepsilon_{\mathrm{ds}} \le 2^{-128} + 10 \cdot 2^{-256}$. With $a = 38$ and $a = 54$ the total slopes of the stateful and stateless bounds (Claims 6.1 and 6.2, with the PRF terms), the constant c_V dominates the linear part for $\xi < c_V \cdot 2^{128}/a$, i.e., up to about 353 queries on the stateful path and about 1827 on the stateless path.

The constant c_V is a cost of the reductions, not a known attack. For EU-CMA, [KWW26, Section 7.2] sketches attacks with one query of the forger that succeed with probability about $45/2^{128} \approx 2^{-122.5}$ on the stateless path and about $2p_T \cdot 256/2^{128} \approx 2^{-125.04}$ on the stateful path. Here $p_T \approx 2^{-6.04}$ is the probability that a fresh randomizer gives a valid target-sum encoding. The constants $c = c_0 + c_V \le 2^{-111.40}$ and $2^{-114.28}$ of the two paths lie about $2^{11.1}$ and $2^{10.8}$ above these values.

Quantum adversaries. The verification is classical: the reduction evaluates the functions on inputs fixed by the forgery, after the forger has stoped. We charge these evaluations as quantum queries, which overestimates their cost, and separate three kinds of summands of the quantum bounds of Tables 6 and 7.

1. The summands of scale 2^{-128}, quadratic in q, of the SM-TCR-type, SM-PRE, and SM-DSPR rows, and the $w \cdot$SM-UD term $64\xi^2/2^{128}$; the quadratic SM-UD and SM-PRE bounds count the queries of both stages, so the verification is charged to them in full. Charged at $q := \xi + V$ instead of $q := \xi$, they add less than $2^{-96.6}$ on the stateless path and $2^{-102.3}$ on the stateful path to the value of the bound at $\xi = 0$.

2. The summands of scale 2^{-64} in $|\mathcal{P}|$ of the SM-TCR-type rows. They receive no charge, since Lemma C.8 counts only the queries of the first stage and the verification runs in the second (Section 6.3).

3. The summands in the message hash, which receive its one evaluation. The quadratic parts $8(\xi+1)^2\Gamma$ of ITSR and $8(\xi+1)^2/2^{256}$ of SM-rTCR grow at $\xi=0$ by $24\Gamma < 2^{-125.6}$ and $24/2^{256} < 2^{-251}$. A randomizer part $4\sqrt{S/2^n}$, with $S = pq + p(p-1)/2 + 1$, grows from $q=0$ to $q=1$ by less than $2^{-62.5}$ for $p \in \{2^8, 2^{20}, 2^{40}\}$ (the shapes UXMSS-256 and BXMSS-20, and the stateless signing budget), since S grows by p; in the form with $e=1/2$ the growth is $32p/2^{128} \le 2^{-83}$.

The charges change no quantum $\xi^\star$: they shift ξ by at most V, which is negligible at $\xi^\star \ge 2^{57}$. They do raise the values at $\xi=0$, and Table 5 and Claims 6.1, 6.2 and 6.4 include them. With the minimum over the forms, every stateful value at $\xi=0$ is at most about 2^{-83}, far below 2^{-64}, so the charges do not matter there. On the stateless path the value at $\xi=0$ is set by the randomizer part of the ITSR term with $p=2^{40}$ signing queries, about 2^{-44}. For the full scheme it is about 2^{-43} with BXMSS-20, where the minimum at $\xi=0$ uses $e=1/2$ at the hop and so doubles the stateless terms, and about 2^{-44} with UXMSS-256, where it uses $e=1$ at the hop. The verification adds about 2^{-83} to the stateless value, almost all of it from the one message-hash query of item 3, about twice that for the full scheme with BXMSS-20 and about the same with UXMSS-256.

6.5 The Reprogramming Terms

The reductions behind the ITSR, SM-rTCR, and S-TCR(Prop) terms need every message digest fixed before the forger runs (Theorems F.2 and G.1 and Algorithm 6), whereas the forger chooses its messages adaptively. The proofs therefore replace the digest of each signing query by a uniform value and program it into the random oracle at the fresh randomizer of that signature. This reprogramming step costs the probability that the forger notices the replacement, which requires a query at the reprogrammed point before the randomizer is revealed. With the generic reprogramming lemmas it grows with the signing budget and, classically, dominates every bound of this section (Remark 6.1). Classically, the bounds used here charge each query against at most one reprogrammed point (Corollaries C.3 and G.1) or, for ITSR, avoid the reprogramming step (Theorem F.1). Table 7 lists the bounds used at the three places of the reprogramming step; for quantum adversaries each place has two forms, explained below.

Place	Classical	Quantum, $e=1$	Quantum, $e=1/2$
ITSR, randomizer part	$P_{45,10,2^{13}}(q, 2^{40})$ (Remark F.1)	$4\sqrt{S_{\mathsf{sl}}/2^n}$	$32\,S_{\mathsf{sl}}/2^n$, search part doubled
SM-rTCR ($K=1$), randomizer part	$q/2^n$ (Corollary G.1)	$4\sqrt{S_L/2^n}$	$32\,S_L/2^n$, hash part doubled
reprogramming hop ($\mathcal{B}_1$, $\mathcal{B}_4$)	$q/2^n$ (Corollary C.3)	$4\sqrt{S_L/2^n}$	$32\,S_L/2^n$, every later term doubled

Table 7: Bounds at the three places of the reprogramming step, in the number q of queries to the message-digest function: $q := \xi+1$ in the first two rows and $q := q_h := \xi$ at the hop (Section 6.3). $S_{\mathsf{sl}} := 2^{40}q + 2^{40}(2^{40}-1)/2 + 1$ is S of Theorem F.3, and $S_L := Lq + L(L-1)/2 + 1$ is S of Corollary C.3 and S' with $K=1$ of Theorem G.1; $P_{h,k,t}$ is the pull part of Theorem F.1. Quantum forms: Lemma C.3 ($e=1$) and Corollary C.1 ($e=1/2$). The hop row bounds the advantages of $\mathcal{B}_1$ (Algorithm 5) and $\mathcal{B}_4$ (Claim 4.14).

Quantum bounds: the two forms. Quantumly, each of the three places is bounded with Lemma C.3, with $e=1$ or, through Corollary C.1, with $e=1/2$. With $e=1$ the lemma bounds the advantage of a distinguisher, which enters the proofs additively. With $e=1/2$, Corollary C.1 relates the two output probabilities of a single distinguisher. We use it only at hops whose reduction simulates both games exactly, so that these output probabilities are the success probabilities in the two games. Every distinguisher for the reprogramming experiments of this document is of this kind, and for such a hop with $\sum_j \hat{q}_j \le S$

$$\mathrm{Succ}^{i}(\mathcal{A}) \;\le\; 2\,\mathrm{Succ}^{i+1}(\mathcal{A}) + 32\,\frac{S}{|X_1|}\,.$$

Since this bounds Succ^i by an increasing function of Succ^{i+1}, a sequence of games still gives an upper bound, and the factor 2 falls on every term bounded after the hop (Remark C.5). The choices at the three places are independent, so each of the eight combinations gives a valid bound, and so does their pointwise minimum; Section 6.6 gives the combinations that attain the minimum at the parameters of Table 1.

6.6 The Two Paths

We now write the bounds of Theorems 4.2 and 4.3 at the proposed parameters as explicit functions of ξ: $\xi/2^Y + c$ for classical adversaries and $\xi^2/2^Y + \xi/2^X + c'$ for quantum adversaries, each with exact integer coefficients and up to one further term ν, which moves $\log_2 \xi^\star$ by less than 0.01 and changes no value at $\xi = 0$ at the printed precision. The coefficients collect the rows of Table 6 and the reprogramming terms of Table 7, instantiated as in Section 6.3: $q := \xi + V$ in the collection terms, except their $|\mathcal{P}|$ summands, which stay at $q := \xi$, and $q := \xi + 1$ in the message-hash terms. Quantumly, every combination of the two forms of Section 6.5 gives a bound, and the bound we state is their minimum. Inside the SM-rTCR proof we use $e = 1/2$, which gives the smaller bound at every ξ at which some combination is below 1. Inside the ITSR proof we use $e = 1/2$ as well; it gives the smaller bound up to $\xi \approx 2^{56.8}$, and above that the form $e = 1$ is slightly smaller but moves $\xi^\star$ by less than 0.01 bits. At the hop the two forms trade off: $e = 1$ adds the square-root term $4\sqrt{S/2^{128}}$, which grows with L, and $e = 1/2$ adds only $32\,S/2^{128}$ but doubles every term behind the hop. On the stateful path and for the full scheme we therefore give the two combinations with $e = 1/2$ inside the property proofs and either form at the hop, and take their minimum; on the stateless path, which has no hop, one combination. In each case the result equals, at the printed precision, the column "minimum" of Table 5. The combination with $e = 1$ at the hop has a square-root term, which we split by $\sqrt{a+b} \le \sqrt{a} + \sqrt{b}$ into a term $\sqrt{\xi}/2^Z$ and a constant. In the rounded forms the exponents Y, X, Z are rounded down and the logarithms of the constants up, so the rounded forms are bounds as well.

Stateless path. Each coefficient is a sum over the terms of Theorem 4.3, in units of 2^{-128} unless stated otherwise.

- *Classical slope:* $54 = 2 + 16 + 32 + 2 + 2$: 2 from the two PRF terms, 16 from $w \cdot$ SM-UD, 32 from the SM-TCR-type summands (4 from each of the five grouped ones and 12 from the q_8 summand with its factor 3), 2 from SM-PRE and 2 from SM-DSPR. The collection terms alone have the slope 52, of which 16 units come from their $|\mathcal{P}|$ summands.
- *Classical constant:* $98687 = 11 + 36 \cdot 2741$: $c_{0,\mathsf{sl}}$ and the verification, 36 charged slope units times V_{sl} (Section 6.4); the Γ and pull parts are in $\nu^{\mathrm{c}}_{\mathsf{sl}}$.
- *Quantum, linear:* the coefficient 36 of $\xi/2^{64}$ is 4 from the PRF terms and 32 from the $|\mathcal{P}|$ summands of the SM-TCR-type rows.
- *Quantum, quadratic:* the coefficient 364 of $\xi^2/2^{128}$ is 64 from $w \cdot$ SM-UD, $256 = 5 \cdot 32 + 96$ from the SM-TCR-type rows, 12 from SM-PRE and 32 from SM-DSPR.
- *Randomizer part:* the ITSR term with $e = 1/2$ contributes $32\,S_{\mathsf{sl}}(\xi+1) = 2^{45}\xi + 2^{84} + 2^{44} + 32$; the remaining $1996048\,\xi$ and $2736406384 = 2736406416 - 32$ come from the collection terms at $q := \xi + V_{\mathsf{sl}}$.

We give only the form with $e = 1/2$, which is the smaller of the two forms up to $\xi \approx 2^{56.8}$. Above that the form with $e = 1$ is slightly smaller, but it reaches 1 at a value of $\xi^\star$ less than 0.01 bits larger.

Claim 6.1 (Bound of Theorem 4.3 at $q_s = 2^{40}$)**.** *Instantiate Theorem 4.3 under Convention I, with ξ standing for the running time and every hash-query count of the forger, $q := \xi + V_{\mathsf{sl}}$,*

$V_{\mathsf{sl}} = 2741$, *in the collection terms except their* $|\mathcal{P}|$ *summands,* $q := \xi + 1$ *in the* ITSR *term, the collection terms bounded by Table 6 with the spaces of Section 6.3 (*$2^n = 2^{128}$, $w = 16$, $\Gamma \le 2^{-130.23}$*), and the reprogramming step of the* ITSR *proof bounded as in Table 7. The* SM-DSPR *term is bounded by the SM-DSPR row of Table 6, which is conjectured, so the claim rests on that conjecture. Then the success probability of every adversary with* ξ *hash queries and* 2^{40} *signing queries is at most, for a classical adversary in the ROM,*

$$\frac{54\,\xi + 98687}{2^{128}} + \nu^{\mathrm{c}}_{\mathsf{sl}}(\xi) \;\le\; \frac{\xi}{2^{122.24}} + 2^{-111.40} + \nu^{\mathrm{c}}_{\mathsf{sl}}(\xi), \tag{1}$$

and, for a halting (Remark C.1) quantum adversary in the QROM, with $e = 1/2$ *in the* ITSR *proof,*

$$\begin{aligned}\frac{36\,\xi}{2^{64}} + \frac{364\,\xi^2 + (2^{45} + 1996048)\,\xi + 2^{84} + 2^{44} + 2736406416}{2^{128}} + \nu^{\mathrm{q}}_{\mathsf{sl}}(\xi) \\ \le\; \frac{\xi^2}{2^{119.49}} + \frac{\xi}{2^{58.83}} + 2^{-43.99} + \nu^{\mathrm{q}}_{\mathsf{sl}}(\xi),\end{aligned} \tag{2}$$

with $\nu^{\mathrm{c}}_{\mathsf{sl}}(\xi) := (\xi + 2)\,\Gamma + P_{45,10,2^{13}}(\xi + 1, 2^{40})$ *and* $\nu^{\mathrm{q}}_{\mathsf{sl}}(\xi) := 16(\xi + 2)^2\,\Gamma$*, where the pull part* $P_{45,10,2^{13}}(\xi + 1, 2^{40})$ *of Theorem F.1 has the leading summand at most* $(\xi + 1)(\xi + 2)/2^{256.98}$ *(Remark F.2). The bounds are* $2^{-111.4}$ *and* $2^{-43.9}$ *at* $\xi = 0$ *and reach* 1 *at* $\xi = 2^{122.23}$ *and* $\xi^\star = 2^{58.5}$*; the terms* ν *move* $\log_2 \xi^\star$ *by less than* 0.01 *and change neither value at* $\xi = 0$*.*

Stateful path. Each coefficient is again a sum over the terms of Theorem 4.2, in units of 2^{-128} unless stated otherwise.

- *Classical slope:* $38 = 16 + 4 \cdot 4 + 2 + 2 + 1 + 1$: 16 from $w \cdot$ SM-UD, 4 from S-TCR(Prop) and from each of the three grouped SM-TCR summands, 2 from each of SM-TCR-EXCL and SM-PRE, and 1 from each of the two reprogramming terms. It does not depend on L: under Convention I the collection budgets do not enter, and the classical reprogramming terms are charged per challenge (Section 6.5). The collection terms alone have the slope 36, of which 9 units come from their $|\mathcal{P}|$ summands; the core terms give $(36\xi + 13426)/2^{128} \approx \xi/2^{122.83}$, the core level of Section 6.1.

- *Classical constant:* $13427 = 7 + 27 \cdot 497 + 1$: $c_{0,\mathsf{sf}}$, the verification (27 charged slope units times V_{sf}) and the SM-rTCR randomizer part (Section 6.4); the 2^{-256} parts are in $\nu^{\mathrm{c}}_{\mathsf{sf}}$.

- *Quantum, linear:* the coefficient 18 of $\xi/2^{64}$ is 4 from S-TCR(Prop) and from each of the three grouped SM-TCR summands and 2 from SM-TCR-EXCL, all from their $|\mathcal{P}|$ summands.

- *Quantum, quadratic:* the coefficient 212 of $\xi^2/2^{128}$ is 32 from S-TCR(Prop), 64 from $w \cdot$ SM-UD, 96 from the three SM-TCR rows, 12 from SM-PRE and 8 from SM-TCR-EXCL.

- *Terms in L:* these are the reprogramming terms: $32\,S_L(\xi+1) = 32L\xi + 16L^2 + 16L + 32$ from the SM-rTCR term with $e = 1/2$, and from the hop $32\,S_L(\xi)$ with $e = 1/2$ or $4\sqrt{S_L(\xi)/2^{128}}$ with $e = 1$.

- *Doubling:* with $e = 1/2$ at the hop the seven collection terms behind it are doubled (Corollary 4.2): $18 \to 36$ and $212 \to 424$. The SM-rTCR term precedes the hop and is not doubled; the factor 2 on its hash part, in $\nu^{\mathrm{q}}_{\mathsf{sf}}$, comes from the form $e = 1/2$ inside the SM-rTCR proof (Table 7).

Claim 6.2 (Bound of Theorem 4.2 at $q_s = L$)**.** *Assume Conjecture 3.1, as Theorem 4.2 does. Instantiate Theorem 4.2 with* $K = 1$ *under Convention I, with* ξ *standing for the running time and every hash-query count of the forger,* $q := \xi + V_{\mathsf{sf}}$*,* $V_{\mathsf{sf}} = 497$ *at the worst anchor depth* $\delta^\star = 255$*, in the collection terms except their* $|\mathcal{P}|$ *summands,* $q := \xi + 1$ *in the* SM-rTCR *term,*

$q := q_h := \xi$ at the hop, the collection terms bounded by Table 6 with the spaces of Section 6.3 ($2^n = 2^{128}$, $|\mathcal{N}| = 2^{256}$ for $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, $w = 16$), the reprogramming step of the SM-rTCR *proof bounded as in Table 7, and the reprogramming hop bounded through the advantage of the distinguisher $\mathcal{B}_1$ of the proof (Corollary C.3; with $e = 1/2$, Corollary 4.2). Let $\nu^{\mathrm{c}}_{\mathsf{sf}}(\xi) := (\xi+2)/2^{256}$ and $\nu^{\mathrm{q}}_{\mathsf{sf}}(\xi) := 16(\xi+2)^2/2^{256}$. Then the success probability of every adversary with ξ hash queries and L signing queries is at most, for a classical adversary in the ROM and every L,*

$$\frac{38\,\xi + 13427}{2^{128}} + \nu^{\mathrm{c}}_{\mathsf{sf}}(\xi) \;\le\; \frac{\xi}{2^{122.75}} + 2^{-114.28} + \nu^{\mathrm{c}}_{\mathsf{sf}}(\xi), \tag{3}$$

and, for a halting (Remark C.1) quantum adversary in the QROM, the minimum of the bound with $e = 1/2$ at both places,

$$\frac{36\,\xi}{2^{64}} + \frac{424\,\xi^2 + (64L + 422048)\,\xi + 32L^2 + 105026400}{2^{128}} + \nu^{\mathrm{q}}_{\mathsf{sf}}(\xi), \tag{4}$$

and the bound with $e = 1$ at the hop and $e = 1/2$ in the SM-rTCR *proof,*

$$\begin{aligned}\frac{18\,\xi}{2^{64}} + \frac{212\,\xi^2 + (32L + 211024)\,\xi + 16L^2 + 16L + 52513200}{2^{128}} \\ + \frac{4\sqrt{L}\,\sqrt{\xi}}{2^{64}} + 4\sqrt{\frac{L(L-1)/2 + 1}{2^{128}}} + \nu^{\mathrm{q}}_{\mathsf{sf}}(\xi).\end{aligned} \tag{5}$$

For the two prescribed shapes, BXMSS-20 ($L = 2^{20}$) and UXMSS-256 ($L = 2^8$), these are at most, in the order Eq. (4), Eq. (5),

$$\begin{aligned}\textit{BXMSS}\text{-}20\colon\quad & \frac{\xi^2}{2^{119.27}} + \frac{\xi}{2^{58.83}} + 2^{-82.99} + \nu^{\mathrm{q}}_{\mathsf{sf}}(\xi) \\ & \frac{\xi^2}{2^{120.27}} + \frac{\xi}{2^{59.83}} + \frac{\sqrt{\xi}}{2^{52}} + 2^{-42.50} + \nu^{\mathrm{q}}_{\mathsf{sf}}(\xi) \\ \textit{UXMSS}\text{-}256\colon\quad & \frac{\xi^2}{2^{119.27}} + \frac{\xi}{2^{58.83}} + 2^{-101.32} + \nu^{\mathrm{q}}_{\mathsf{sf}}(\xi) \\ & \frac{\xi^2}{2^{120.27}} + \frac{\xi}{2^{59.83}} + \frac{\sqrt{\xi}}{2^{58}} + 2^{-54.50} + \nu^{\mathrm{q}}_{\mathsf{sf}}(\xi)\end{aligned}$$

The classical bound is $2^{-114.2}$ at $\xi = 0$ and reaches 1 at $\xi = 2^{122.75}$. The quantum minimum is Eq. (4) for $\xi \le 286586 \approx 2^{18.1}$ (BXMSS-20) and $\xi \le 69$ (UXMSS-256) and Eq. (5) above; it is $2^{-82.9}$ and $2^{-101.3}$ at $\xi = 0$ and reaches 1 at $\xi^\star = 2^{59.2}$ for both shapes, and Eq. (4) alone at $2^{58.4}$. The terms ν move $\log_2 \xi^\star$ by less than 0.01 and change no value at $\xi = 0$.

6.7 The Full Scheme

Theorem 4.1 consists of two top-level PRF terms, for **PRF** and for the combined $\mathbf{PRF}_{\mathbf{msg}}$, the SM-rTCR term and the seven collection terms of Theorem 4.2 at $q_s^{\mathsf{sf}} = L$, one reprogramming hop at $q_s^{\mathsf{sf}} = L$, shared by the two paths (Claim 4.14) and bounded for its distinguisher $\mathcal{B}_4$, and the ten summands of Theorem 4.3 other than its two PRF terms. The two PRF terms contribute $2\xi/2^{128}$ classically and $4\xi/2^{64}$ quantumly, and every PRF replacement is counted once. Over the core terms alone the classical bound is $(90\xi + 112114)/2^{128} + (\xi+2)\Gamma \approx \xi/2^{121.50}$, with $\varepsilon_{\mathrm{ds}}$ and up to summands of scale 2^{-256}, the core level of Section 6.1; the reprogramming terms complete it to Eq. (9). Quantumly the shared hop can be bounded with either form of Section 6.5, independently of the forms used inside the two property proofs.

Claim 6.3 (Full scheme, classical and quantum)**.** *Assume Conjecture 3.1, as Theorem 4.1 does. Model $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as a random oracle, quantum-accessible for a quantum adversary, and every function of the scheme as in Assumption 4.1, let $\mathcal{A}$ be a classical or halting (Remark C.1)*

quantum adversary with the resources of Theorem 4.1, where $q_s^{\mathsf{sf}} = L$ *by Remark 4.6, applied to the stateful path (as in Section 6.3), let* $S := q_s^{\mathsf{sf}}\, q_h + q_s^{\mathsf{sf}}(q_s^{\mathsf{sf}} - 1)/2 + 1$, *let* Σ_{sf} *be the sum of the seven stateful collection terms and* Σ_{sl} *the sum of the ten stateless summands of Theorem 4.1, with the resources stated there, i.e., the right-hand sides of Claims 4.15 and 4.16, and let* $T_0 := \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF};\, \xi,\, q_1) + \mathrm{InSec}^{\mathrm{PRF}}(\mathbf{PRF}_{\mathbf{msg}};\, \xi,\, q_s) + \mathrm{InSec}^{\mathsf{SM\text{-}rTCR},1}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}},\, \mathbf{1};\, \tilde{\xi}'_{\mathsf{sf}},\, q_s^{\mathsf{sf}},\, \tilde{q}_h^{\mathsf{sf}})$. *Then* $\mathrm{Succ}^{\mathit{SHRINCS\text{-}EU\text{-}CMA}}_{\mathit{SHRINCS},q_s}(\mathcal{A})$ *is at most*

$$T_0 + q_h/2^n + \Sigma_{\mathsf{sf}} + \Sigma_{\mathsf{sl}} + \varepsilon_{\mathrm{ds}} \qquad \textit{classical adversary, ROM,} \tag{6}$$

$$T_0 + 4\sqrt{S/2^n} + \Sigma_{\mathsf{sf}} + \Sigma_{\mathsf{sl}} + \varepsilon_{\mathrm{ds}} \qquad \textit{quantum adversary, QROM, } e = 1 \textit{ at the shared hop,} \tag{7}$$

$$T_0 + 32\, S/2^n + 2\left(\Sigma_{\mathsf{sf}} + \Sigma_{\mathsf{sl}}\right) + \varepsilon_{\mathrm{ds}} \qquad \textit{quantum adversary, QROM, } e = 1/2 \textit{ at the shared hop.} \tag{8}$$

Proof. In the proof of Theorem 4.1 (Section 4.8), the hops from **GAME.0** to **GAME.3** bound $\mathrm{Succ}^0(\mathcal{A}) - \mathrm{Succ}^3(\mathcal{A})$ by T_0, and the hops from **GAME.4** to **GAME.6** bound $\mathrm{Succ}^4(\mathcal{A})$ by $\Sigma_{\mathsf{sf}} + \Sigma_{\mathsf{sl}}$, since **GAME.6** cannot be won; $\varepsilon_{\mathrm{ds}}$ is charged once, outside the sequence of games. The remaining hop, from **GAME.3** to **GAME.4**, is bounded through the distinguisher $\mathcal{B}_4$ defined before Claim 4.14, which Corollary C.3 covers. $\mathcal{B}_4$ outputs 1 in $\mathbf{DI\text{-}Repro}_0$ exactly when **GAME.3** succeeds and in $\mathbf{DI\text{-}Repro}_1$ exactly when **GAME.4** succeeds, so $|\mathrm{Succ}^3(\mathcal{A}) - \mathrm{Succ}^4(\mathcal{A})| = \mathsf{Adv}^{\mathsf{DI\text{-}Repro}}(\mathcal{B}_4)$. The bounds of Corollary C.3 on this advantage, $q_h/2^n$ in the ROM and $4\sqrt{S/2^n}$ in the QROM, give Eqs. (6) and (7). For Eq. (8), the same exact simulation lets us use the last inequality of Corollary C.3 gives $\mathrm{Succ}^3(\mathcal{A}) \le 2\,\mathrm{Succ}^4(\mathcal{A}) + 32\, S/2^n$, and Remark C.5 with $T_{\mathsf{before}} = T_0$ and $T_{\mathsf{after}} = \Sigma_{\mathsf{sf}} + \Sigma_{\mathsf{sl}}$ gives the bound, as in the proof of Corollary 4.2. □

Claim 6.4 (Bound of Theorem 4.1 at the proposed parameters)**.** *Instantiate Claim 6.3 as Claims 6.1 and 6.2 instantiate their theorems, each term with the counts and bounds of its own path, the shared hop bounded through the distinguisher* $\mathcal{B}_4$ *as in Claim 6.3,* $e = 1/2$ *inside both property proofs, and* $\varepsilon_{\mathrm{ds}} \le 2^{-128} + 10 \cdot 2^{-256}$. *Let* $\nu^{\mathrm{c}}_{\mathsf{full}}(\xi) := (\xi+2)\,\Gamma + P_{45,10,2^{13}}(\xi+1, 2^{40}) + (\xi+12)/2^{256}$ *and, for* $f \in \{1,2\}$, $\nu^{\mathrm{q}}_f(\xi) := 16f(\xi+2)^2\,\Gamma + (16(\xi+2)^2+10)/2^{256}$. *As in Claim 6.1, the* SM-DSPR *term is bounded by the conjectured SM-DSPR row of Table 6, so the claim rests on that conjecture. Then the success probability of every adversary with the resources of Theorem 4.1 is at most, for a classical adversary in the ROM and every L,*

$$\frac{92\,\xi + 112115}{2^{128}} + \nu^{\mathrm{c}}_{\mathsf{full}}(\xi) \;\le\; \frac{\xi}{2^{121.47}} + 2^{-111.22} + \nu^{\mathrm{c}}_{\mathsf{full}}(\xi), \tag{9}$$

and, for a halting (Remark C.1) quantum adversary in the QROM, the minimum of the bound with $e = 1/2$ *at the hop,*

$$\frac{104\,\xi}{2^{64}} + \frac{1152\,\xi^2 + (2^{46} + 64L + 4414144)\,\xi}{2^{128}} + \frac{2^{85} + 2^{45} + 32L^2 + 5577839233}{2^{128}} + \nu^{\mathrm{q}}_2(\xi), \tag{10}$$

and the bound with $e = 1$ *at the hop,*

$$\frac{54\,\xi}{2^{64}} + \frac{576\,\xi^2 + (2^{45} + 32L + 2207072)\,\xi + 2^{84} + 2^{44} + 16L^2 + 16L + 2788919617}{2^{128}} + \frac{4\sqrt{L}\,\sqrt{\xi}}{2^{64}} + 4\sqrt{\frac{L(L-1)/2 + 1}{2^{128}}} + \nu^{\mathrm{q}}_1(\xi). \tag{11}$$

For the two prescribed shapes these are at most, in the order Eq. (10), Eq. (11),

$$\begin{aligned}
\textit{BXMSS}\text{-}20\colon\quad & \frac{\xi^2}{2^{117.83}}+\frac{\xi}{2^{57.29}}+2^{-42.99}+\nu_2^{\mathrm{q}}(\xi)\\
& \frac{\xi^2}{2^{118.83}}+\frac{\xi}{2^{58.24}}+\frac{\sqrt{\xi}}{2^{52}}+2^{-42.06}+\nu_1^{\mathrm{q}}(\xi)\\
\textit{UXMSS}\text{-}256\colon\quad & \frac{\xi^2}{2^{117.83}}+\frac{\xi}{2^{57.29}}+2^{-42.99}+\nu_2^{\mathrm{q}}(\xi)\\
& \frac{\xi^2}{2^{118.83}}+\frac{\xi}{2^{58.24}}+\frac{\sqrt{\xi}}{2^{58}}+2^{-43.99}+\nu_1^{\mathrm{q}}(\xi)
\end{aligned}$$

The classical bound is $2^{-111.2}$ at $\xi=0$ and reaches 1 at $\xi=2^{121.47}$. The quantum minimum is $2^{-42.9}$ (BXMSS-20) and $2^{-43.9}$ (UXMSS-256) at $\xi=0$ and reaches 1 at $\xi^\star=2^{58.0}$ for both shapes, and Eq. (10) alone at $2^{57.1}$; for BXMSS-20 the minimum switches from Eq. (10) to Eq. (11) at $\xi\approx 2^{15.8}$. The terms ν move $\log_2\xi^\star$ by less than 0.01 and change no value at $\xi=0$.

6.8 Reduction Work under Convention M

The tables above follow Convention I, which does not charge the collection work $q_{\lambda,i}$ of the reductions to the adversary; Convention M (Definition 5.1) does, and we give its classical figures only. Convention M counts the collection queries as attack queries, so each collection term is instantiated at $q:=\xi+V+q_{\lambda,i}$. The exception are again the $|\mathcal{P}|$ summands of the classical SM-TCR-type rows, which count only the queries of the first stage, before the public parameter is revealed (Section 6.3). The collection queries still belong to the first stage, the verification does not, so these summands are instantiated at $\xi+q_{\lambda,i}$. Every classical row is affine in q, so both replacements leave every coefficient of ξ unchanged and add a constant of at most $cB/2^{128}$, where $c\xi/2^{128}$ is the sum of the linear parts of the collection terms and $B:=\max_i q_{\lambda,i}$ is the largest collection budget.

Stateful path. For stand-alone FXMSS every core term of Theorem 4.2 except the hash part of the SM-rTCR term, of scale 2^{-256}, is a collection term, so $c=36$. By Theorem 4.2 the budget is at most $B=L(l_{\mathsf{sf}}w+1)+I+q_{\mathsf{kg}}+q_g$. Here q_{kg} is the honest stateless key generation, 287,743 calls (Table 4); the theorem's rounded bound $q_{\mathsf{kg}}\le 2^{19}$ would move the UXMSS-256 budget slightly above $2^{24.04}$. The grinding budget q_g can be read in two ways. Its expectation is about $65.7\,L$, at most 2^7L (Remark 4.4), but in a single run it is bounded only by the counter cap, $q_g\le 2^{16}L$, which holds in every run by Conjecture 3.1. Table 8 gives both readings; only the cap reading is a valid budget, since an insecurity is a maximum over all adversaries with at most that many collection queries, and the expected reading, which rests on Assumption 3.1, is given for comparison.

Stateless path. The collection terms are the core terms of Theorem 4.3 except the two PRF terms and the search part of ITSR, so $c=52$. The budget is $B=Q_\Lambda^{\mathsf{sl}}=2^{45}\cdot 164{,}965\approx 2^{62.33}$ (Theorem 4.3).

Path	q_g	B	$c\,B/2^{128}$	Gap
Stateless ($c = 52$)	—	$\approx 2^{62.33}$	$2^{-59.96}$	51.4
Stateful, BXMSS-20	cap $2^{16}L$	$< 2^{36.02}$	$2^{-86.81}$	27.5
($c = 36$)	expected $\leq 2^{7}L$	$\approx 2^{29.3}$	$2^{-93.50}$	20.8
Stateful, UXMSS-256	cap $2^{16}L$	$< 2^{24.04}$	$2^{-98.79}$	15.5
($c = 36$)	expected $\leq 2^{7}L$	$\approx 2^{18.8}$	$2^{-104.04}$	10.2

Table 8: The constant $c\,B/2^{128}$ that Convention M adds to the classical bounds, for the collection budget B and, on the stateful path, both readings of the grinding budget q_g. The last column gives, in bits, how far the constant lies above the classical bound at $\xi = 0$ of Table 5, which is $\approx 2^{-111.41}$ on the stateless path and $\approx 2^{-114.29}$ on the stateful path. The constants are rounded up to two decimals; the gaps are computed from the exact values and rounded to nearest.

Since the forger's own share $c\,\xi/2^{128}$ of the collection terms exceeds $c\,B/2^{128}$ as soon as $\xi > B$, Convention M raises the classical bounds only for small ξ, and no classical $\xi^\star$ of Table 5 and none of the classical core-only levels of Section 6.1 changes.

The stateful figures are for FXMSS as a stand-alone scheme. Within Theorem 4.1 the stateful collection budgets also carry Q^{sl}_Λ (Claim 4.15), so in the full scheme the stateless budget sets the Convention M constant of both paths. Second, Convention M is applied here to the collection terms only; the simulation time t_{sim} of the message-hash terms stays discharged by Lemma 4.2.

7 Open Questions and Work in Progress

The bounds of Section 6 rest on four results that we expect to improve. We list them here, together with what an improvement would change and the route we are following. The work in progress named below consists of drafts; it is not part of this document, nothing in this document depends on it, and it has not been checked to the standard of the rest of the document.

Constants of the reprogramming lemma. The quantum reprogramming terms use Lemma C.3 and Corollary C.1 with the constants 4 and 32. They rest on the derivation of [LGX25, Theorem 5] that [LGX25] write out (paragraph *Constant* of the proof of Lemma C.3). With the printed constant of their Theorem 5, the same argument gives $2\sqrt{2}$ and 16 (Remark C.3). We are working on a proof of smaller constants, down to 2 and 8. From 4 to 2 the reprogramming terms with $e = 1$ halve, and from 32 to 8 those with $e = 1/2$ shrink by a factor 4. Since the column "minimum" of Table 5 already attains the $\xi^\star$ of the core terms (Section 6.1), smaller constants cannot raise any $\xi^\star$ of that column; they lower the values of the bounds at $\xi = 0$.

Removing the SM-DSPR conjecture. The bound of Theorem 4.3 contains an SM-DSPR term, and the SM-DSPR row of Table 6 is conjectured [BHK+19, BH19]. For a single target, [BH19] prove a quantum bound of the same form; for p targets the known argument loses a factor p, which makes the bound vacuous at the roughly 2^{61} targets of the stateless path. We are working on proven bounds without this factor that match the conjectured row, for the experiment of Definition 4.10, in which the first stage does not know P, and for a member of the collection (Section 4.3). Once such bounds are in place, the SM-DSPR row of Table 6 can cite a proof instead of a conjecture, and the stateless path, and with it the full scheme, no longer rests on a conjecture about SM-DSPR. The SM-DSPR term is a small share of the stateless bound, so we expect the figures of Section 6 to change little, but we will not rely on a conjecture anymore.

A tighter quantum bound for ITSR. Classically, Theorem F.1 bounds ITSR without reprogramming the message hash. The quantum bound Theorem F.3 still replaces the digests of the challenge queries by reprogramming, and its randomizer part sets the value of the stateless bound at $\xi = 0$ (Section 6.4). We follow a route through a compressed-oracle argument in which no adaptive reprogramming step appears. In its current draft form it avoids the reprogramming

term but has a larger constant on the search term and needs an additional hypothesis. The goal is a quantum analogue of Theorem F.1, with a matching attack for each of its terms.

Quantum reprogramming at distinct inputs. The reprogramming hop of the stateful path reprograms the message hash at pairwise distinct inputs (Definition C.2). Classically, this structure gives the bound of Corollary C.2, in which each query is charged against at most one reprogrammed point. Quantumly, the bound is only inherited from Lemma C.3 through Lemma C.5, which does not use that the inputs are distinct. A quantum bound that uses it would shrink the hop term and its dependence on the number L of leaves, i.e., the contribution of the hop to the terms in $\sqrt{L\xi}$, $L\xi$, and L^2 of the stateful forms in Claim 6.2.

References

[ACFK17] Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Memory-tight reductions. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 101–132, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[AHU18] Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. Cryptology ePrint Archive, Report 2018/904, 2018.

[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, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 421–454, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[BDH11] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 117–129, Tapei, Taiwan, November 29 – December 2 2011. Springer Berlin Heidelberg, Germany.

[BDH+24] Manuel Barbosa, François Dupressoir, Andreas Hülsing, Matthias Meijers, and Pierre-Yves Strub. A tight security proof for SPHINCS$^+$, formally verified. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part IV*, volume 15487 of *Lecture Notes in Computer Science*, pages 35–67, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.

[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, Cham, Switzerland.

[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.

[BHRvV21] Joppe W. Bos, Andreas Hülsing, Joost Renes, and Christine van Vredendaal. Rapidly verifiable XMSS signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):137–168, 2021.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416, Saragossa, Spain, May 12–16, 1996. Springer Berlin Heidelberg, Germany.

[DKKW25] Justin Drake, Dmitry Khovratovich, Mikhail A. Kudinov, and Benedikt Wagner. Hash-based multi-signatures for post-quantum ethereum. *IACR Communications in Cryptology (CiC)*, 2(1):13, 2025.

[DSS05] C. Dods, Nigel P. Smart, and Martijn Stam. Hash based digital signature schemes. In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 96–115, Cirencester, UK, December 19–21, 2005. Springer Berlin Heidelberg, Germany.

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

[GT20] Ashrujit Ghoshal and Stefano Tessaro. On the memory-tightness of hashed ElGamal. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 33–62, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[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, Cham, Switzerland.

[HKRY23] Andreas Hülsing, Mikhail A. Kudinov, Eyal Ronen, and Eylon Yogev. SPHINCS+C: Compressing SPHINCS+ with (almost) no cost. In *2023 IEEE Symposium on Security and Privacy*, pages 1435–1453, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[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 Berlin Heidelberg, Germany.

[Hül13] Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13: 6th International Conference on Cryptology in Africa*, volume 7918 of *Lecture Notes in Computer Science*, pages 173–188, Cairo, Egypt, June 22–24, 2013. Springer Berlin Heidelberg, Germany.

[KM07] Neal Koblitz and Alfred J. Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, January 2007.

[KN25] Mikhail Kudinov and Jonas Nick. Hash-based signature schemes for bitcoin. Cryptology ePrint Archive, Report 2025/2203, 2025.

[Kud25] Mikhail Kudinov. *Hash Functions in a Post-Quantum World*. PhD thesis, Eindhoven University of Technology, 2025.

[Kud26a] Mikhail Kudinov. Interleaved target subset resilience without reprogramming – classical bounds with exact constants. GitHub repository, https://github.com/SHRINCS/Security-Analysis, 2026.

[Kud26b] Mikhail Kudinov. A quadratic quantum bound for multi-target undetectability of tweakable hash functions. GitHub repository, https://github.com/SHRINCS/Security-Analysis, 2026.

[KWW26] Mikhail Kudinov, Benedikt Wagner, and Emile Wambsgans. The exact bit security of hash-based signatures. Cryptology ePrint Archive, Paper 2026/2626, 2026. Version of 2026-10-24.

[LGX25] Heming Liao, Jiangxia Ge, and Rui Xue. Non-adaptive one-way to hiding not only implies adaptive quantum reprogramming, but also does better. Cryptology ePrint Archive, Report 2025/1998, 2025.

[LGXZ26] Heming Liao, Jiangxia Ge, Rui Xue, and Xiaogang Zhou. Non-adaptive one-way to hiding not only implies adaptive quantum reprogramming, but also does better. In Shi Bai and Edoardo Persichetti, editors, *PKC 2026, Part II*, volume 16552 of *LNCS*, pages 96–124. Springer, Cham, 2026.

[MKJR16] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA cryptography specifications version 2.2. RFC 8017, RFC Editor, November 2016. Obsoletes RFC 3447.

[Nat08] National Institute of Standards and Technology. The keyed-hash message authentication code (HMAC). Federal Information Processing Standards Publication FIPS PUB 198-1, U.S. Department of Commerce, Gaithersburg, MD, July 2008.

[Nat15] National Institute of Standards and Technology. Secure hash standard (SHS). FIPS PUB Federal Information Processing Standards Publication 180-4, U.S. Department of Commerce, Gaithersburg, MD, August 2015.

[Nat24] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Federal Information Processing Standards Publication FIPS PUB 205, U.S. Department of Commerce, Gaithersburg, MD, August 2024.

[Nic25] Jonas Nick. SHRINCS: 324-byte stateful post-quantum signatures with static backups. Delving Bitcoin forum post, https://delvingbitcoin.org/t/shrincs-324-byte-stateful-post-quantum-signatures-with-static-backups/2158, December 2025.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[SXY17] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. Cryptology ePrint Archive, Report 2017/1005, 2017.

[The25] The SHRINCS Signature Algorithm Working Group. shrincs-bip: The SHRINCS specification staging area. GitHub repository, https://github.com/SHRINCS/shrincs-bip, 2025. Work in progress.

[XY18] Keita Xagawa and Takashi Yamakawa. (Tightly) QCCA-secure key-encapsulation mechanism in the quantum random oracle model. Cryptology ePrint Archive, Report 2018/838, 2018.

[YTA23] Quan Yuan, Mehdi Tibouchi, and Masayuki Abe. Quantum-access security of hash-based signature schemes. In Leonie Simpson and Mir Ali Rezazadeh Baee, editors, *ACISP 23: 28th Australasian Conference on Information Security and Privacy*, volume 13915 of *Lecture Notes in Computer Science*, pages 343–380, Brisbane, QLD, Australia, July 5–7, 2023. Springer, Cham, Switzerland.

[Zha12] Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 758–775, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

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

A HMAC

HMAC is a message authentication code constructed from a hash function H with block size B and output length L [Nat08]. Given a key K and a message text, it is defined as

$$\mathsf{HMAC}(K, \mathsf{text}) = H\big((K_0 \oplus \mathsf{opad}) \parallel H((K_0 \oplus \mathsf{ipad}) \parallel \mathsf{text})\big),$$

where $\mathsf{ipad} = \mathtt{0x36}^B$ and $\mathsf{opad} = \mathtt{0x5c}^B$ are B-byte padding constants, and K_0 is the key K normalized to B bytes:

- if $|K| = B$, set $K_0 = K$;
- if $|K| > B$, set $K_0 = H(K) \parallel 0^{B-L}$;
- if $|K| < B$, set $K_0 = K \parallel 0^{B-|K|}$.

Step-by-step computation. Given K_0, the HMAC value is computed as follows:

1. Compute the inner-padded key: $K_0 \oplus \mathsf{ipad}$.
2. Compute the inner hash: $h_{\text{in}} = H\big((K_0 \oplus \mathsf{ipad}) \parallel \mathsf{text}\big)$.
3. Compute the outer-padded key: $K_0 \oplus \mathsf{opad}$.
4. Output $H\big((K_0 \oplus \mathsf{opad}) \parallel h_{\text{in}}\big)$.

B MGF1

A mask generation function (MGF) takes an octet string of variable length and a desired output length as input and produces a pseudorandom octet string of that length [MKJR16]. The output is deterministic and should be pseudorandom: given part of the output without the input, predicting another part should be infeasible.

MGF1. MGF1 is parameterized by a hash function Hash with output length hLen octets. It takes as input a seed $\mathsf{mgfSeed}$ and an intended mask length $\mathsf{maskLen} \le 2^{32}\,\mathsf{hLen}$, and produces an octet string mask of length $\mathsf{maskLen}$.

Given these inputs, MGF1 builds a long pseudorandom string T by iterating the hash function over the seed concatenated with a 4-byte counter, then truncates to the desired length:

1. If $\mathsf{maskLen} > 2^{32}\,\mathsf{hLen}$, abort with error `"mask too long"`.
2. Set $T \leftarrow \varepsilon$ (the empty string).
3. For each $\mathsf{counter} \in \{0, 1, \ldots, \lceil \mathsf{maskLen}/\mathsf{hLen} \rceil - 1\}$, compute
$$T \leftarrow T \parallel \mathrm{Hash}\big(\mathsf{mgfSeed} \parallel \mathrm{I2OSP}(\mathsf{counter}, 4)\big).$$
4. Output the leading $\mathsf{maskLen}$ octets of T as the mask.

I2OSP (Integer to Octet String Primitive) converts a non-negative integer to its big-endian octet string representation of a fixed length [MKJR16]. The formal description is given below.

I2OSP. I2OSP takes as input a non-negative integer x and an intended output length xLen, and produces an octet string X of length xLen.

If $x \geq 256^{\mathsf{xLen}}$, the integer is too large to encode and the procedure aborts with error `"integer too large"`. Otherwise, write x in its unique base-256 representation

$$x = \sum_{i=0}^{\mathsf{xLen}-1} x_i \cdot 256^i, \qquad 0 \leq x_i < 256,$$

and output the big-endian octet string $X = x_{\mathsf{xLen}-1} \parallel \cdots \parallel x_1 \parallel x_0$.

C (Quantum) Random Oracle Tools

The generic bounds of this document in the classical and the quantum random oracle model rest on three kinds of experiment, which this section collects together with their bounds. The first is adaptive reprogramming of a random oracle, together with a variant in which every call carries a different input. The proofs use it whenever a reduction answers the signing queries (Section 4) or the challenge queries (Sections F and G) by programming the message hash. The second is the search for an input that a sparse random boolean function maps to 1. The third is a two-stage experiment in which a point function on the public parameters must be told apart from the zero function; it serves the properties in which the public parameter is revealed only after the challenges (Sections D and E).

Adaptive Reprogramming. A reprogramming experiment lets a distinguisher interact with a random oracle that may change while it runs. We follow the formalization of Grilo, Hövelmanns, Hülsing, and Majenz [GHHM21]. The caller fixes one part x_2 of the input, the experiment draws the other part x_1 uniformly at random, and the value at (x_1, x_2) is either replaced by a fresh uniform value or left as it is. The definition is followed by the quantum bound of [GHHM21] and by an elementary classical counterpart.

Definition C.1 (Adaptive reprogramming [GHHM21])**.** Let X_1, X_2, and Y be finite sets, and let $\mathcal{A}$ be a stateful algorithm. Let $R,\ q \in \mathbb{N}$. Consider the following experiment $\mathbf{Repro_b}$:

- Sample a random oracle $\mathcal{O}_0 \leftarrow_\$ Y^{X_1 \times X_2}$, i.e., $\mathcal{O}_0 \colon X_1 \times X_2 \to Y$.
- Define a copy of $\mathcal{O}_0$ as $\mathcal{O}_1 := \mathcal{O}_0$.
- Run $\mathcal{A}$ with (classical or quantum) access to $\mathcal{O}_b$ and classical access to oracle $\mathsf{Reprogram} \colon X_2 \to X_1$, where $\mathcal{A}$ is allowed to make up to q queries to $\mathcal{O}_b$ and up to R queries to $\mathsf{Reprogram}$.
- Obtain from $\mathcal{A}$ a bit $b' \in \{0,1\}$ and output b'.

Here, the oracle $\mathsf{Reprogram}(x_2)$ is defined the following way:

1. Sample $(x_1, y) \leftarrow_\$ X_1 \times Y$.
2. $\mathcal{O}_1 := \mathcal{O}_1^{(x_1,x_2)\mapsto y}$, i.e., $\mathcal{O}_1$ is reprogrammed such that $\mathcal{O}_1(x_1, x_2) = y$.
3. Return x_1.

For any such algorithm $\mathcal{A}$, we define the advantage in the above experiment as

$$\mathrm{Succ}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) = \left|\Pr\left[\mathbf{Repro_0}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{Repro_1}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

If $\mathcal{A}$ is quantum and queries $\mathcal{O}_b$ in superposition, its advantage is bounded as follows.

Lemma C.1 ([GHHM21])**.** *Let X_1, X_2, and Y be finite sets, and let $\mathcal{A}$ be any algorithm in the game in Definition C.1. Assume that $\mathcal{A}$ issues R many classical calls to* Reprogram *and q many quantum queries to $\mathcal{O}_b$. Then, we have*

$$\mathsf{Adv}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) \leq \frac{3R}{2} \cdot \sqrt{\frac{q}{|X_1|}}.$$

For a classical distinguisher the bound is linear in the number of queries.

Lemma C.2. *Let X_1, X_2, and Y be finite sets, and let $\mathcal{A}$ be any classical algorithm in the game in Definition C.1. Assume that $\mathcal{A}$ issues R many classical calls to* Reprogram *and q many classical queries to $\mathcal{O}_b$. Then, we have*

$$\mathsf{Adv}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) \leq \frac{R \cdot q}{|X_1|}.$$

Proof. The two games can be told apart only if $\mathcal{A}$ has queried the random oracle at a reprogrammed seed x_1 *before* the call to Reprogram that selects this seed. Each seed is drawn uniformly from X_1, so it coincides with one of the at most q earlier queries with probability at most $q/|X_1|$. The union bound over the R calls gives the claim. □

A tighter quantum bound. The bound of Lemma C.1 pays for the R reprogrammings: a hybrid argument over the R reprogramming points sums R terms of the form $\sqrt{q/|X_1|}$. Liao, Ge, Xue, and Zhou [LGXZ26] (full version [LGX25]) do a more carefull analysis of the multitarget case. Their theorem is stated for the reprogramming framework of [GHHM21] and also for square roots of the two probabilities, which is the form a search-type game can use without a square root in the final bound. We state the instantiation for the experiment of Definition C.1. The count is finer than in Lemma C.1: only the queries issued *before* a reprogramming can help to detect it.

Remark C.1 (Halting quantum adversaries)**.** Throughout this document, quantum adversaries are algorithms in the standard query model that halt with probability 1 in every game in which they are run, including the intermediate games of the proofs, and the algorithms that the reductions build from them inherit both properties. We call these algorithms *halting*; they are the class to which Lemma C.3 applies. Every insecurity InSec against quantum adversaries is the maximum over halting ones (Section 4.3); the statements that rest on Lemma C.3 or Corollary C.1 name the class explicitly, since only they need it.

Remark C.2 (Quantum oracle access)**.** Every function queried in superposition in this document has output space $\{0,1\}^\nu$ for some ν ($\nu = 1$ for the Boolean challenge functions of ZeroDist and Avg-Search) and is accessed through the standard oracle with bitwise XOR. When the answer of a reduction to a query has the form $h(x) \oplus L_x\big(f(s(x))\big)$, with s, h and the GF(2)-linear maps L_x independent of f, the reduction can answer it with one query to f [Kud26b].

Lemma C.3 (Adapted from the proof of [LGX25, Theorem 7])**.** *Let X_1, X_2, and Y be finite sets, where, for some $\mu, \nu \in \mathbb{N}$, Y is identified with $\{0,1\}^\nu$ and $X_1 \times X_2$ with a subset of $\{0,1\}^\mu$ by an injective map, and let $\mathcal{A}$ be any quantum algorithm in the standard query model in the game of Definition C.1 with quantum access to $\mathcal{O}_b$ that issues at most R classical calls to* Reprogram *and halts with probability 1 in* $\mathbf{Repro_0}$ *and in* $\mathbf{Repro_1}$*. An algorithm that has issued fewer than R calls when it produces its output continues with the remaining calls, on an arbitrary input, without changing its output. For $j \in [R]$ let $\hat{q}_j$ be an upper bound, over all executions of* $\mathbf{Repro_0}$ *and of* $\mathbf{Repro_1}$*, on the number of queries to $\mathcal{O}_b$ issued before the j-th call to* Reprogram*. Then, for $e \in \{1, 1/2\}$,*

$$\left|\Pr\left[\mathbf{Repro_0}(\mathcal{A}) \Rightarrow 1\right]^e - \Pr\left[\mathbf{Repro_1}(\mathcal{A}) \Rightarrow 1\right]^e\right| \leq 4 \cdot \sqrt{\frac{\sum_{j=1}^{R} \hat{q}_j}{|X_1|}}\,.$$

In particular, if $\mathcal{A}$ issues at most q queries to $\mathcal{O}_b$ in total in every such execution, then $\sum_j \hat{q}_j \le R\,q$ and the right-hand side is at most $4\sqrt{Rq/|X_1|}$.

Instantiation. The experiment of Definition C.1 is an instance of the reprogramming framework of [GHHM21] as restated in [LGX25, Section 4, Figure 3]: the j-th call $\mathsf{Reprogram}(x_2)$ corresponds to a challenge query whose distribution over the reprogrammed point is uniform on $X_1 \times \{x_2\}$ and whose new value y is uniform on Y, and the caller learns the point x_1 but not the value, which it may read through $\mathcal{O}_b$ like any other query. The maximal probability of any point under the j-th distribution is $1/|X_1|$, for every j, for every choice of x_2, and in both games, so the quantity $p^j_{\max}$ of [LGX25, Theorem 7] equals $1/|X_1|$ regardless of how the expectation in its definition is taken.

Bit strings. The theorem is stated for functions $H\colon \{0,1\}^\mu \to \{0,1\}^\nu$ (its m and n). Let enc be the injective map of the statement and $\beta\colon Y \to \{0,1\}^\nu$ the identification, and for $H\colon \{0,1\}^\mu \to \{0,1\}^\nu$ let $\mathcal{O}_H(d) := \beta^{-1}(H(\mathsf{enc}(d)))$. A uniformly random H, restricted to the image of enc, is a uniformly random function on that image, so $\mathcal{O}_H$ is uniform on $Y^{X_1 \times X_2}$. An algorithm for the game of Definition C.1 becomes an algorithm in the framework by computing $\mathsf{enc}(d)$ into a fresh register before each query and uncomputing it afterwards, one query per query. Its j-th call becomes a challenge query with the uniform distribution on $\mathsf{enc}(X_1 \times \{x_2\})$, and the returned point is decoded to x_1. Reprogramming H_1 at $\mathsf{enc}(x_1, x_2)$ is $\mathsf{Reprogram}$ acting on $\mathcal{O}_{H_1}$, and points outside the image of enc are never queried or reprogrammed. Output probabilities, query counts, and $p^j_{\max} = 1/|X_1|$ are unchanged. $|X_1|$ need not be a power of 2, since the framework samples the challenge point from an arbitrary distribution.[16]

Running total. The theorem is stated for an algorithm whose i-th stage makes a fixed number q_i of queries, with $\hat{q}_j := q_0 + \cdots + q_{j-1}$. Applied as stated to an algorithm whose queries fall into different stages in different executions, it gives only the sum of the per-stage maxima. The running total of the statement follows from the proof of [LGX25, Theorem 7] in three steps.

First, we reduce to algorithms that stop after a fixed number of steps, which is what the proof of the theorem needs. For $T \in \mathbb{N}$, let $\mathcal{A}_T$ run $\mathcal{A}$ for at most T steps; if $\mathcal{A}$ has not halted by then, $\mathcal{A}_T$ outputs 0 and makes the calls to $\mathsf{Reprogram}$ that $\mathcal{A}$ has not made, without further queries. Stopping $\mathcal{A}$ early adds no queries, so $\mathcal{A}_T$ satisfies the hypotheses with the same $\hat{q}_j$ (since $\mathcal{A}$ halts with probability 1, every run of $\mathcal{A}_T$ begins like a run of $\mathcal{A}$ to which the bound $\hat{q}_j$ applies). Since $\mathcal{A}$ halts with probability 1, the output probabilities of $\mathcal{A}_T$ converge to those of $\mathcal{A}$ in both games as T grows, so the bound for every $\mathcal{A}_T$ gives the bound for $\mathcal{A}$.

Second, we put $\mathcal{A}$ into the model of the theorem, which requires every stage to make a fixed number of queries. After the real queries of a stage, $\mathcal{A}$ makes dummy queries on a fresh register pair in the state $|0^\mu\rangle\,|+\rangle$, where $|+\rangle$ is the uniform superposition over $\{0,1\}^\nu$, until the stage reaches its maximal number of queries over all executions of both games. A dummy query changes nothing, since adding any value to $|+\rangle$ leaves it unchanged, so output probabilities and challenge distributions stay the same. Applied as stated, the theorem would count the dummy queries; the third step shows that they do not count. For that step we also replace every measurement of $\mathcal{A}$ by a copy into a fresh register. The state of the whole system is then a superposition of classical configurations of all registers, and each configuration that appears with nonzero amplitude records how many real queries have been made so far.

Third, we show that the dummy queries do not count. The proof of the theorem is a quantum version of a classical fact: a reprogramming at a uniformly random point can only be noticed if the algorithm has queried that point before, and k queries touch at most k points, so this happens with probability at most $k/|X_1|$. In the proof, the touched points are recorded in a table of registers, one for each input, which start at 0^ν. At the j-th challenge the proof uses only that at most $\hat{q}_j$ entries are non-zero; this is the only place where the query counts enter. A real query

[16] In the applications, X_2 contains a message of variable length, e.g., the bound message $\widetilde{M}$ on the stateful path. The set is finite: SHA-256 accepts inputs of fewer than 2^{64} bits, and Algorithm 2 restricts messages to at most $2^{61} - 384$ bytes.

makes at most one entry non-zero, a dummy query changes none, and the reprogramming steps can only reset entries to 0^ν. So in every configuration the number of non-zero entries is at most the number of real queries made. By hypothesis this number is at most $\hat{q}_j$ in every execution of $\mathbf{Repro_0}$ and of $\mathbf{Repro_1}$. The proof replaces these two experiments by two equivalent games, in which the algorithm behaves exactly as in the original experiments. In particular, the number of real queries it makes before its j-th call has the same distribution, so it is at most $\hat{q}_j$ there as well. One of these games is equivalent to $\mathbf{Repro_0}$ and the other to $\mathbf{Repro_1}$, so the bound is needed in both. Hence at most $\hat{q}_j$ entries are non-zero at the j-th challenge, which is all the proof needs.

Constant. We state the lemma with the constant 4 instead of the $2\sqrt{2}$ of [LGX25, Theorem 7]. Their proof uses [LGX25, Theorem 5] once, and for that theorem [LGX25] write out only a derivation that loses a factor of at most $\sqrt{2}$; we use the bound that this derivation gives. □

Remark C.3 (The printed constant of [LGX25])**.** If [LGX25, Theorem 5] holds as printed, the same argument gives Lemma C.3 with $2\sqrt{2}$ and Corollary C.1 with 16; Section 7 discusses these forms.

Remark C.4 (Padding before the final query)**.** In the applications of Lemma C.3 in this document the algorithm forwards at most q queries of an adversary, issues one query of its own after each of its genuine calls to read the reprogrammed value, and issues at most one further query, after the output of the adversary. Before its j-th call the algorithm has issued at most q forwarded queries and at most $j-1$ read-backs, one per earlier genuine call, whether the j-th call is genuine or a padding call. Hence $\hat{q}_j \le q + j - 1$ for every j and every execution of either game, since the count does not depend on the answers of $\mathcal{O}_b$, and

$$\sum_{j=1}^{R} \hat{q}_j \;\le\; R\,q + \frac{R(R-1)}{2} \;<\; R\,q + \frac{R(R-1)}{2} + 1\,.$$

The form with $e = 1/2$ converts into an additive bound on the probability itself.

Corollary C.1. *In the setting of Lemma C.3,*

$$\Pr\left[\mathbf{Repro_0}(\mathcal{A}) \Rightarrow 1\right] \le 2 \cdot \Pr\left[\mathbf{Repro_1}(\mathcal{A}) \Rightarrow 1\right] + 32 \cdot \frac{\sum_{j=1}^{R} \hat{q}_j}{|X_1|}\,,$$

and the same holds with the two games exchanged.

Proof. Write $a = \sqrt{\Pr[\mathbf{Repro_1}(\mathcal{A}) \Rightarrow 1]}$ and $\varepsilon = 4\sqrt{\sum_j \hat{q}_j/|X_1|}$. Lemma C.3 with $e = 1/2$ gives $\sqrt{\Pr[\mathbf{Repro_0}(\mathcal{A}) \Rightarrow 1]} \le a + \varepsilon$, and $(a+\varepsilon)^2 \le 2a^2 + 2\varepsilon^2$ since $2a^2 + 2\varepsilon^2 - (a+\varepsilon)^2 = (a-\varepsilon)^2 \ge 0$. Finally $2\varepsilon^2 = 32 \sum_j \hat{q}_j/|X_1|$. □

When a reduction uses the reprogrammed game to bound the success probability of a search-type experiment, Corollary C.1 costs a factor 2 on the bound of the reprogrammed game but no square root on the reprogramming term; Lemma C.3 with $e = 1$ keeps the two contributions additive. Both forms are instantiated in Section 6.

Using the form $e = 1/2$ in a game sequence. The two forms bound different quantities. With $e = 1$, Lemma C.3 bounds the advantage of a distinguisher $\mathcal{B}$, that is, the difference $|\Pr[\mathbf{Repro_0}(\mathcal{B}) \Rightarrow 1] - \Pr[\mathbf{Repro_1}(\mathcal{B}) \Rightarrow 1]|$. With $e = 1/2$, it bounds the difference of the square roots of these two probabilities, for one fixed $\mathcal{B}$.

The form with $e = 1/2$ is of a different kind. Through Corollary C.1 it gives

$$\Pr\left[\mathbf{Repro_0}(\mathcal{B}) \Rightarrow 1\right] \;\le\; 2\,\Pr\left[\mathbf{Repro_1}(\mathcal{B}) \Rightarrow 1\right] + 32\,\frac{\sum_j \hat{q}_j}{|X_1|}$$

for one fixed distinguisher $\mathcal{B}$. This is a relation between the two output probabilities of $\mathcal{B}$, not a bound on their difference. It therefore cannot bound the insecurity term $\mathrm{InSec}^{\mathsf{DI\text{-}Repro}}$ of Theorems 4.1 and 4.2, which is the largest advantage of any distinguisher with the given resources. It says something about $\mathrm{Succ}^{i}(\mathcal{A})$ and $\mathrm{Succ}^{i+1}(\mathcal{A})$ only if the output probabilities of $\mathcal{B}$ equal these success probabilities.

The form is therefore used only at a game hop whose reduction $\mathcal{B}$ simulates both games exactly:

$$\Pr\left[\mathbf{Repro_0}(\mathcal{B}) \Rightarrow 1\right] = \mathrm{Succ}^{i}(\mathcal{A}) \qquad \text{and} \qquad \Pr\left[\mathbf{Repro_1}(\mathcal{B}) \Rightarrow 1\right] = \mathrm{Succ}^{i+1}(\mathcal{A})\,.$$

The inequalities $\mathrm{Succ}^{i}(\mathcal{A}) \le \Pr[\mathbf{Repro_0}(\mathcal{B}) \Rightarrow 1]$ and $\Pr[\mathbf{Repro_1}(\mathcal{B}) \Rightarrow 1] \le \mathrm{Succ}^{i+1}(\mathcal{A})$ would suffice, because the bound of Corollary C.1 is increasing in $\Pr[\mathbf{Repro_1}(\mathcal{B}) \Rightarrow 1]$.

All distinguishers for $\mathbf{Repro_b}$ and for the experiment of Definition C.2 in this document satisfy the equalities:

- $\mathcal{B}_2$ in the proof of Theorem F.2, and its padded variant $\mathcal{B}_2'$ in the proof of Theorem F.3;
- $\mathcal{C}$ in the proof of Theorem G.1, and its padded variant $\mathcal{C}'$;
- $\mathcal{B}_1$ of Algorithm 5;
- $\mathcal{B}_4$ of Claim 4.14.

Each of them runs $\mathcal{A}$ in a simulated game and outputs 1 exactly when $\mathcal{A}$ wins. For $b = 0$ the simulated game is the earlier game, and for $b = 1$ it is the later one. $\mathcal{B}_1$ and $\mathcal{B}_4$ play the experiment of Definition C.2 rather than $\mathbf{Repro_b}$ itself. By Lemma C.5, each is an algorithm for $\mathbf{Repro_b}$ with the same output probabilities, so they fit the same pattern. For such a hop with $\sum_j \hat{q}_j \le S$, Corollary C.1 gives

$$\mathrm{Succ}^{i}(\mathcal{A}) \;\le\; 2\,\mathrm{Succ}^{i+1}(\mathcal{A}) + 32\,\frac{S}{|X_1|}\,.$$

Remark C.5 (The form $e = 1/2$ in a sequence of games)**.** The inequality $\mathrm{Succ}^{i}(\mathcal{A}) \le 2\,\mathrm{Succ}^{i+1}(\mathcal{A}) + 32\,S/|X_1|$ above, for a hop whose reduction simulates both games exactly and with $\sum_j \hat{q}_j \le S$, bounds the earlier success probability by an increasing function of the later one, and no bound on their difference is needed: a sequence of games yields an upper bound on the success probability in the first game as long as every hop bounds Succ^{i} by an increasing function of Succ^{i+1}. If the terms of the hops before the reprogramming hop sum to T_{before}, those of the hops after it to T_{after}, and the last game cannot be won, then

$$\mathrm{Succ}^{0}(\mathcal{A}) \;\le\; T_{\mathsf{before}} + 32\,\frac{S}{|X_1|} + 2\,T_{\mathsf{after}}\,,$$

in place of $T_{\mathsf{before}} + 4\sqrt{S/|X_1|} + T_{\mathsf{after}}$ with $e = 1$.

In the proofs of Theorems F.3 and G.1 the reprogramming step is the first hop, and the factor 2 falls on what these proofs bound after it, the search term of Game 1: $8(q+1)^2\,\Gamma$ becomes $16(q+1)^2\,\Gamma$ in Eq. (15), and $8(q+1)^2/|\mathcal{N}|$ becomes $16(q+1)^2/|\mathcal{N}|$ in Eq. (18). The result is a bound on the success probability of every adversary against the property with the stated resources, i.e., a bound on the insecurity term. The proof of the scheme uses it as the value of the property term in an additive hop, so the factor 2 of the property proof does not reach any other term of the scheme's bound.

Adaptive reprogramming with distinct inputs. On the stateful path of SHRINCS every call to Reprogram carries a different second input: the input contains the leaf identifier, and a leaf signs once. For a classical adversary this changes the count. A reprogramming at (x_1, x_2) is visible only if the point was queried before the call, and a query names one second input, so it can serve against at most one call. We first state the classical count call by call for the experiment of Definition C.1, then define the experiment with distinct inputs as a property of its own, and bound it.

Lemma C.4 (Classical reprogramming, per-call count)**.** *Let X_1, X_2, and Y be finite sets, and let $\mathcal{A}$ be any classical algorithm in the game of Definition C.1 that issues at most R calls to* Reprogram*. For $j \in [R]$ let $x_2^{(j)}$ be the input of the j-th call, and let n_j be the number of seeds $x_1 \in X_1$ such that $\mathcal{A}$ has queried $\mathcal{O}_b$ at $(x_1, x_2^{(j)})$ before its j-th call; set $n_j := 0$ if $\mathcal{A}$ issues fewer than j calls. Then, with the expectation taken in either of the two games,*

$$\mathsf{Adv}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) \le \frac{1}{|X_1|} \sum_{j=1}^{R} \mathbb{E}[n_j]\,.$$

In particular, $n_j \le \hat{q}_j$ for every upper bound $\hat{q}_j$ as in Lemma C.3, and $n_j \le q$ for an algorithm with at most q queries, which gives Lemma C.2.

Proof. We sample the oracle lazily: the value at a point is sampled uniformly from Y when the point is queried for the first time, and recorded. In $\mathbf{Repro_1}$ a call $\mathsf{Reprogram}(x_2)$ that returns x_1 discards the value recorded at (x_1, x_2), if there is one, so that the next query at this point is answered with a fresh uniform value. This is the distribution of Definition C.1: the value y installed by the call is uniform, independent of everything else, and not seen by $\mathcal{A}$ before its next query at (x_1, x_2). In $\mathbf{Repro_0}$ the call leaves the records unchanged. The two experiments therefore differ only when a call discards a recorded value, and a value is recorded at a point only if the point was queried. Let Bad_j be the event that the point $(x_1^{(j)}, x_2^{(j)})$ of the j-th call was queried before that call. Until the first such event the two experiments proceed identically, so $\mathsf{Bad} := \bigvee_j \mathsf{Bad}_j$ has the same probability in both and $\mathsf{Adv}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) \le \Pr[\mathsf{Bad}]$. The seed $x_1^{(j)}$ is uniform over X_1 and independent of the view of $\mathcal{A}$ before the j-th call, which determines $x_2^{(j)}$ and the n_j seeds x_1 for which $(x_1, x_2^{(j)})$ was queried. Hence $\Pr[\mathsf{Bad}_j] = \mathbb{E}[n_j]/|X_1|$, and the union bound over j gives the claim. □

Definition C.2 (Adaptive reprogramming with distinct inputs)**.** Let X_1, X_2, Y, R, q, and $\mathcal{A}$ be as in Definition C.1. The experiment $\mathbf{DI\text{-}Repro}_b$ is the experiment $\mathbf{Repro_b}$ with one change. The challenger keeps a list $\mathcal{L} \subseteq X_2$ of submitted inputs, initially empty. On a call $\mathsf{Reprogram}(x_2)$ with $x_2 \in \mathcal{L}$ the oracle returns $\perp$ and neither samples nor reprograms; otherwise it inserts x_2 into $\mathcal{L}$ and proceeds as in Definition C.1. Every call counts toward R. We define

$$\mathsf{Adv}^{\mathsf{DI\text{-}Repro}}_{R,q}(\mathcal{A}) = \left|\Pr\left[\mathbf{DI\text{-}Repro}_0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathbf{DI\text{-}Repro}_1(\mathcal{A}) \Rightarrow 1\right]\right|.$$

For a keyed hash function $G\colon X_1 \times X_2 \to Y$ that is modeled as the random oracle of the experiment, $\mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(G; \xi, R, q)$ is the maximum of $\mathsf{Adv}^{\mathsf{DI\text{-}Repro}}_{R,q}(\mathcal{A})$ over all algorithms $\mathcal{A}$ (quantum ones halting, Remark C.1) that run in time ξ, issue at most R calls to Reprogram, and make at most q queries to $\mathcal{O}_b$.

The guard only restricts the adversary, so every bound for $\mathbf{Repro_b}$ carries over.

Lemma C.5. *For every algorithm $\mathcal{A}$ in the experiment of Definition C.2 there is an algorithm $\mathcal{A}'$ in the experiment of Definition C.1 with $\Pr[\mathbf{Repro_b}(\mathcal{A}') \Rightarrow 1] = \Pr[\mathbf{DI\text{-}Repro}_b(\mathcal{A}) \Rightarrow 1]$ for $b \in \{0, 1\}$. The calls of $\mathcal{A}'$ to* Reprogram *carry pairwise distinct inputs and are the calls of $\mathcal{A}$ that are not answered by $\perp$; the queries of $\mathcal{A}'$ to $\mathcal{O}_b$ are those of $\mathcal{A}$, with the same (classical*

or quantum) access; and its time, measured in function evaluations (Section 4.3), is that of $\mathcal{A}$. Consequently Lemmas C.1, C.2 and C.4 hold for $\mathbf{DI\text{-}Repro}_b$ *with the same right-hand sides, and so do Lemma C.3 and Corollary C.1 with $\sum_j \hat{q}_j$ replaced by $R\,q$. If no call of $\mathcal{A}$ is answered by $\perp$, then $\mathcal{A}' = \mathcal{A}$, and Lemma C.3 and Corollary C.1 hold with the counts $\hat{q}_j$ of $\mathcal{A}$, the completing calls of Lemma C.3 being issued in* $\mathbf{Repro_b}$.

Proof. $\mathcal{A}'$ runs $\mathcal{A}$ and keeps the list $\mathcal{L}$ itself: it answers a call with $x_2 \in \mathcal{L}$ by $\perp$ without calling its own oracle, and forwards every other call and every query. The view of $\mathcal{A}$ is that of $\mathbf{DI\text{-}Repro}_b$, and $\mathcal{A}'$ outputs what $\mathcal{A}$ outputs. The bounds of Lemmas C.1 and C.2 are non-decreasing in R, and the sum of Lemma C.4 for $\mathcal{A}'$ runs over a subset of the calls of $\mathcal{A}$ with non-negative terms, so the three bounds named first hold with the parameters of $\mathcal{A}$. When the guard fires, the j-th call of $\mathcal{A}'$ is a later call of $\mathcal{A}$, and the per-call counts $\hat{q}_j$ of $\mathcal{A}'$ can exceed those of $\mathcal{A}$; this is why the forms with $\sum_j \hat{q}_j$ are stated for $\mathcal{A}' = \mathcal{A}$ only, and with $\sum_j \hat{q}_j \le R\,q$ otherwise. □

Corollary C.2 (Classical bound for DI-Repro)**.** *Let $\mathcal{A}$ be a classical algorithm in the experiment of Definition C.2 that makes at most q queries to $\mathcal{O}_b$, and let q_{pre} be the number of its queries whose second input is submitted to* Reprogram *for the first time after the query. Then, with the expectation taken in either of the two games,*

$$\mathsf{Adv}^{\mathsf{DI\text{-}Repro}}_{R,q}(\mathcal{A}) \le \frac{\mathbb{E}[q_{\mathsf{pre}}]}{|X_1|} \le \frac{q}{|X_1|}\,,$$

and therefore $\mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(G;\,\xi,\,R,\,q) \le q/|X_1|$ *in the classical random oracle model, for every R.*

Proof. Apply Lemma C.4 to the algorithm $\mathcal{A}'$ of Lemma C.5. A query at (x_1, x_2) contributes to n_j only if $x_2^{(j)} = x_2$ and the j-th call comes after the query. The inputs $x_2^{(j)}$ are pairwise distinct, so there is at most one such j, and there is one only if the query is counted by q_{pre}. Hence $\sum_j n_j \le q_{\mathsf{pre}} \le q$. □

Remark C.6 (Read-back queries are free)**.** The reductions of this document have the shape described in Remark C.4: they forward at most q queries of an adversary, issue one query of their own after each genuine call to Reprogram, at the point that was just reprogrammed, and issue at most one further query after their last call. In the experiment of Definition C.2, and for a reduction whose calls are never answered by $\perp$, none of these own queries is counted by q_{pre}: a read-back carries the second input of a call that has already been made, and the final query comes after the last call. For such a reduction Corollary C.2 therefore gives the classical bound $q/|X_1|$ with q the number of forwarded queries alone. The same applies to forwarded queries at second inputs that have already been submitted, for example the queries by which a forger verifies the signatures it has received.

Corollary C.3 (Concrete bounds for the reprogramming term)**.** *Let $S := q_s\,q_h + q_s(q_s-1)/2 + 1$ and model $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as a random oracle with $|\mathcal{R}| = 2^n$. The insecurity of the class named in Theorem 4.2 satisfies*

$$\mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}};\,\tilde{\xi}',\,q_s,\,q_s+q_h+1) \le (q_s+q_h+1)/2^n \qquad \textit{in the ROM,}$$

$$\mathrm{InSec}^{\mathsf{DI\text{-}Repro}}(\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}};\,\tilde{\xi}',\,q_s,\,q_s+q_h+1) \le \min\Big\{\tfrac{3}{2}q_s\sqrt{(q_s+q_h+1)/2^n},\;4\sqrt{q_s(q_s+q_h+1)/2^n}\Big\} \qquad \textit{in the QROM.}$$

Let $\mathcal{B}$ be a distinguisher that forwards at most q_h queries, issues at most q_s calls, none of which is answered by $\perp$, reads the programmed value once after each call, and makes at most one further query, after its last call; assume in addition that $\mathcal{B}$ issues the same number of calls in every execution of either game, or makes no further query. The distinguishers $\mathcal{B}_1$ of Algorithm 5 and

$\mathcal{B}_4$ *of Claim 4.14, which are the ones the proofs use, are of this kind. Then*

$$\mathsf{Adv}^{\text{DI-Repro}}(\mathcal{B}) \le q_h/2^n \qquad \text{in the ROM ,}$$
$$\mathsf{Adv}^{\text{DI-Repro}}(\mathcal{B}) \le \min\Big\{\tfrac{3}{2} q_s \sqrt{(q_s+q_h+1)/2^n},\ 4\sqrt{S/2^n}\Big\} \qquad \text{in the QROM,}$$

and, in the QROM,

$$\Pr\left[\mathbf{DI\text{-}Repro}_0(\mathcal{B}) \Rightarrow 1\right] \le 2\Pr\left[\mathbf{DI\text{-}Repro}_1(\mathcal{B}) \Rightarrow 1\right] + 32\, S/2^n.$$

The QROM bounds that come from Lemma C.3 or Corollary C.1 are stated for halting distinguishers (Remark C.1).

Proof. By Lemma C.5, the bounds for $\mathbf{Repro_b}$ carry over to $\mathbf{DI\text{-}Repro}_b$. With $R = q_s$ and $q = q_s + q_h + 1$, the ROM line is Corollary C.2, and the QROM line follows from Lemma C.1 and from Lemma C.3 with $\sum_j \hat{q}_j \le R\,q = q_s(q_s+q_h+1)$. Lemma C.2 gives the weaker classical bound $q_s(q_s+q_h+1)/2^n$ for the class in the same way; we do not restate it.

The distinguisher $\mathcal{B}$. No call of $\mathcal{B}$ is answered by $\perp$, so by Lemma C.5 $\mathcal{B}$ is also an algorithm for $\mathbf{Repro_b}$ with the same output probabilities, and we count its queries there. Before its i-th call, $\mathcal{B}$ has forwarded at most q_h queries and has issued $i-1$ read-backs, one after each earlier call, and its further query comes after its last call. Hence $\hat{q}_i \le q_h + i - 1$ for every call of $\mathcal{B}$, in every execution of either game. The hypothesis on $\mathcal{B}$ leaves two cases. If $\mathcal{B}$ issues the same number $a \le q_s$ of calls in every execution of either game, Lemma C.3 is applied with $R = a$: all calls precede the further query, which is the situation of Remark C.4 with no padding call, no completing call is needed, and $\sum_i \hat{q}_i \le S - 1$. If $\mathcal{B}$ makes no further query, the lemma is applied with $R = q_s$: when $\mathcal{B}$ has issued a calls, the completing calls follow all of its at most $q_h + a$ queries, so $\hat{q}_i \le q_h + a \le q_h + i - 1$ for $i > a$, and again $\sum_i \hat{q}_i \le S - 1$. In total $\mathcal{B}$ issues at most $q_s + q_h + 1$ queries to $\mathcal{O}_b$ (q_h forwarded, q_s read-backs, and the further query), which is the count Lemma C.1 takes. Lemmas C.1 and C.3 give the QROM line and Corollary C.1 the last inequality, in both cases through Lemma C.5. The ROM line is Remark C.6, which involves no completing calls.

The distinguishers $\mathcal{B}_1$ and $\mathcal{B}_4$. $\mathcal{B}_1$ issues one call per signing query, hence $q_s = L$ calls in every execution (Remark 4.6), and its calls are never answered by $\perp$ because the signing queries name pairwise distinct leaf identifiers. Its further query verifies the forgery, so it makes $q_s + q_h + 1$ queries, as many as the class in Theorem 4.2 allows. It outputs 1 in $\mathbf{DI\text{-}Repro}_0$ exactly when **F.1** succeeds and in $\mathbf{DI\text{-}Repro}_1$ exactly when **F.2** succeeds (Claim 4.2). $\mathcal{B}_4$ does the same for **GAME.3** and **GAME.4**. By Remark 4.6, applied to the stateful path, it also issues $q_s = L$ calls in every execution, each followed by one read-back and all before the further query that verifies the forgery, so no padding call of Remark C.4 is involved (paragraph after Claim 4.14). □

HRS-Framework for Sets. Several bounds reduce to a search problem: given oracle access to a random boolean function that takes the value 1 only rarely, find an input on which it takes this value. Hülsing, Rijneveld, and Song [HRS16] bound the success probability of this search for functions on bit strings $\{0,1\}^c$. Their argument does not use the structure of bit strings, and we state the framework and its bound for functions on an arbitrary set $\mathcal{S}$.

Definition C.3 (HRS-Framework for Sets [HRS16])**.** Let $\mathcal{S}$ be a set, and let $\mathcal{F} = \{f\colon \mathcal{S} \to \{0,1\}\}$ be the collection of *all* functions that map elements of $\mathcal{S}$ to $\{0,1\}$. Let $\lambda \in [0,1]$ and $\varepsilon > 0$. Define a family of distributions D_λ on $\mathcal{F}$ such that a function $f \leftarrow D_\lambda$ drawn from D_λ satisfies

$$f\colon x \mapsto \begin{cases} 1 & \text{with probability } \lambda, \\ 0 & \text{with probability } 1-\lambda \end{cases} \qquad \text{for any } x \in \mathcal{S},$$

where all choices are made independently. The average case search problem $\mathsf{Avg\text{-}Search}_\lambda$ is the problem of finding an $x \in \mathcal{S}$ such that $f(x) = 1$ given (classical or quantum) oracle access to $f \leftarrow D_\lambda$. Namely, for any algorithm $\mathcal{A}$, we define

$$\mathrm{Succ}^{\mathsf{Avg\text{-}Search}_\lambda}(\mathcal{A}) := \Pr\left[f(x) = 1 \;\middle|\; f \leftarrow D_\lambda, \quad x \leftarrow \mathcal{A}^f(\cdot)\;\right].$$

Lemma C.6 ([HRS16], [BHRvV21]; for arbitrary sets [Kud25, Lemma 6.1.5])**.** *For any algorithm $\mathcal{A}$ that makes at most q queries to f, it holds that*

$$\mathrm{Succ}^{\mathsf{Avg\text{-}Search}_\lambda}(\mathcal{A}) \leq \begin{cases} \lambda(q+1), & \text{if } \mathcal{A} \text{ is a classical algorithm} \\ 8\lambda(q+1)^2, & \text{if } \mathcal{A} \text{ is a quantum algorithm with quantum access to } f \end{cases}$$

Public parameter reprogramming. Some properties, SM-TCR among them, fix all challenges before the adversary sees the public parameter: every challenge query is made while the parameter is still hidden. This order permits a cheaper simulation than reprogramming the tweakable hash function at each challenge input separately. The challenge queries are answered with an independent function, and once the public parameter has been chosen, this function is programmed into the tweakable hash function at that parameter. The experiment and the two lemmas below provide the bound on which this technique rests.

Definition C.4 (Distinguishing from Constant Zero)**.** Let $f_{P'} \colon \mathcal{P} \to \{0,1\}$ be the boolean function, with $f_{P'}(P) = 1$ if and only if $P' = P$. Let $f_0 \colon \mathcal{P} \to \{0,1\}$ be the boolean function for which $f_0(P) = 0$ for all $P \in \mathcal{P}$. Let $\mathcal{A}$ be a (stateful) algorithm, and consider the following experiment $\mathbf{ZeroDist}(\mathcal{A})$:

1. Generate a random public parameter $P' \leftarrow_\$ \mathcal{P}$.
2. Flip a random coin $b \leftarrow_\$ \{0,1\}$.
3. If $b = 1$ Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with access to an oracle $f_{P'}$.
4. If $b = 0$ Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with access to an oracle f_0.
5. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P', but without the oracle access.
6. Obtain an output from $\mathcal{A}_2$: $b' \leftarrow \mathcal{A}_2(P')$ and output b'.

For any such algorithm $\mathcal{A}$, we define the advantage in the above experiment as

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{A}) = \big|&\Pr\left[\mathbf{ZeroDist}(\mathcal{A}) \Rightarrow 1 \mid b = 1\right] - \\ &\Pr\left[\mathbf{ZeroDist}(\mathcal{A}) \Rightarrow 1 \mid b = 0\right]\big|.\end{aligned}$$

Lemma C.7 ([Kud25, Lemma 6.1.8])**.** *Let $\mathcal{A}$ be a classical algorithm that makes no more than q classical queries to its oracle. Then, we have*

$$\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{A}) \leq \frac{q}{|\mathcal{P}|}.$$

For a quantum adversary $\mathcal{A}$ whose first stage queries the boolean function in superposition, the bound on the advantage in $\mathbf{ZeroDist}(\mathcal{A})$ is linear in the number of queries and decreases only with $\sqrt{|\mathcal{P}|}$. The second stage learns P' while it still holds the quantum state of the first stage, so it distinguishes two pure states that it knows, and the distance of two pure states is linear in the difference of their amplitudes.

Lemma C.8. *Let $\mathcal{A}$ be a quantum algorithm whose first stage makes no more than q quantum queries to its oracle. Then, we have*

$$\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{A}) \leq \frac{2q}{\sqrt{|\mathcal{P}|}}.$$

The lemma is the one-way-to-hiding theorem [AHU18, Theorem 3] (see also [GHHM21, Theorem 8] and the algorithmic variant [HHK17, Lemma 4.1]) for $S = \{P'\}$, $H = f_0$, and $G = f_{P'}$, with P' given to the second stage only after the last query: the queries of $\mathcal{A}_1$ under f_0 do not depend on P', so each of them hits P' with probability $1/|\mathcal{P}|$.

Remark C.7 (The earlier quadratic bound)**.** The bound $4q^2/2^k$ for $\mathcal{P} = \{0,1\}^k$, stated as $(**)$ in [HRS16] for this experiment and used in [HK22, HKRY23, Kud25, DKKW25], does not hold: one query on the uniform superposition with the target qubit in $|-\rangle$, followed by the optimal measurement given P', achieves advantage $2\sqrt{|\mathcal{P}|-1}/|\mathcal{P}| > 4/|\mathcal{P}|$ for $|\mathcal{P}| > 5$, and Grover iterations show that Lemma C.8 is tight up to lower-order terms. The SM-TCR and S-TCR bounds of Table 6 are therefore stated with Lemma C.8.

Hidden-slice reprogramming. The two lemmas above bound the cost of the technique sketched before Definition C.4. In its simplest form, a two-stage adversary evaluates a tweakable hash function **Th** on its own and, in its first stage, makes classical challenge queries to $\mathbf{Th}(P', \cdot, \cdot)$ at a public parameter P' that it learns only in the second stage. A simulation answers these queries with an independent random function g instead and programs g into **Th** at P' when the second stage begins. The two executions can be told apart only through first-stage evaluations of **Th** at P', and this is what **ZeroDist** measures: in the proof below, a first-stage evaluation at P' is answered by g under $f_{P'}$ and by an independent function under f_0, and every other evaluation is answered alike in both cases. We state the lemma in the generality that we will use in Section D: for a collection Th_λ of tweakable hash functions (Section 4.3) in place of a single one, and for a classical interface Σ in place of the plain evaluation oracle, together with a predicate Φ that produces the reported bit.

Fix a collection $\mathsf{Th}_\lambda = \{\mathsf{Th}_{m_i}\}$ with parameter space $\mathcal{P}$, tweak space $\mathcal{T}$, and range $\mathcal{N}$. Let Σ be a classical stateful procedure that receives the queries of the first stage and answers them, evaluating the collection at one fixed public parameter, and let Φ be a classical predicate that likewise evaluates the collection at that parameter and produces the bit the experiment reports. Together (Σ, Φ) cover the challenge oracle of any property of Definition 4.12, the collection oracle beside it, and the winning condition. Consider the following two games for a two-stage algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, in which access to the collection is made explicit.

Game 0:
1. Sample $P' \leftarrow_\$ \mathcal{P}$.
2. Run $\mathcal{A}_1$ with access to the collection and classical access to Σ instantiated on the hidden slice:
$$\mathcal{A}_1^{\mathsf{Th}_\lambda(\cdot,\cdot,\cdot),\ \Sigma^{\mathsf{Th}_\lambda(P',\cdot,\cdot)}}(\ );$$
3. Run the second stage on input P', without access to Σ:
$$\mathcal{A}_2^{\mathsf{Th}_\lambda(\cdot,\cdot,\cdot)}(P').$$
4. Report $\Phi^{\mathsf{Th}_\lambda(P',\cdot,\cdot)}$ evaluated on the transcript of Σ and the output of $\mathcal{A}_2$.

Game 1:
1. Sample $P' \leftarrow_\$ \mathcal{P}$ and an independent family of uniformly random functions $g = \{g_{m_i}\}$ with $g_m \colon \mathcal{T} \times \{0,1\}^m \to \mathcal{N}$.
2. Run $\mathcal{A}_1$ with access to the collection and classical access to Σ instantiated on g:
$$\mathcal{A}_1^{\mathsf{Th}_\lambda(\cdot,\cdot,\cdot),\ \Sigma^{g(\cdot,\cdot)}}(\ );$$

3. When $\mathcal{A}$ signals to continue, construct $\mathsf{Th}'_\lambda = \{\mathsf{Th}'_{m_i}\}$ by

$$\mathsf{Th}'_m(P,T,x) = \begin{cases} \text{if } (P = P') : \text{ Return } g_m(T,x) \\ \text{Return } \mathsf{Th}_m(P,T,x) \end{cases}$$

and run the second stage on input P' with Th_λ replaced by Th'_λ:

$$\mathcal{A}_2^{\mathsf{Th}'_\lambda(\cdot,\cdot,\cdot)}(P').$$

4. Report $\Phi^{g(\cdot,\cdot)}$ evaluated on the transcript of Σ and the output of $\mathcal{A}_2$.

Lemma C.9 (Hidden-slice reprogramming)**.** *Consider **Game 0** and **Game 1** as described above, with the collection Th_λ modeled as (quantum) random oracles, and write $\Pr[\mathcal{A}^{G_i} \Rightarrow 1]$ for the probability that the ith game reports 1. Let $\mathcal{A}$ make at most q queries to $\mathsf{Th}_\lambda(\cdot,\cdot,\cdot)$ in its first stage. Then there is an algorithm $\mathcal{B}$ against* **ZeroDist** *(Definition C.4) making at most q queries to its oracle with*

$$\left|\Pr\left[\mathcal{A}^{G_0} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right]\right| \;\le\; \mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{B}).$$

If $\mathcal{A}$ queries $\mathsf{Th}_\lambda(\cdot,\cdot,\cdot)$ quantumly, with $\mathcal{N} = \{0,1\}^\nu$ and the XOR oracle (Remark C.2), then $\mathcal{B}$ is quantum and the difference is at most $2q/\sqrt{|\mathcal{P}|}$; if $\mathcal{A}$ queries classically then $\mathcal{B}$ is classical and the difference is at most $q/|\mathcal{P}|$.

Proof. Let $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ be an adversary for **ZeroDist**, where $\mathcal{B}_1$ has access to a boolean function f, either $f_{P'}$ or f_0, and $\mathcal{B}_2$ receives P'. $\mathcal{B}_1$ samples an independent family of uniformly random functions $\hat{\mathsf{Th}}_m \colon \mathcal{P} \times \mathcal{T} \times \{0,1\}^m \to \mathcal{N}$ and an independent family $g = \{g_{m_i}\}$ as above, and builds a collection from f by

$$\mathsf{Th}_m(P,T,x) = \begin{cases} \text{if } f(P) = 1 : \text{ Return } g_m(T,x) \\ \text{if } f(P) = 0 : \text{ Return } \hat{\mathsf{Th}}_m(P,T,x) \end{cases}$$

Each evaluation of this collection costs one query to f (Remark C.2), so answering the first stage of $\mathcal{A}$ costs at most q queries; when $\mathcal{A}$ queries in superposition, $\mathcal{B}_1$ queries f in superposition, which the experiment of Definition C.4 allows for a quantum adversary. Sampling $\hat{\mathsf{Th}}$ and g is the standard simulation of a quantum-accessible random oracle, by $2q$-wise independent functions or by the compressed-oracle technique [Zha12, Zha19].

$\mathcal{B}_1$ runs $\mathcal{A}_1$ on this collection, answering its mediated queries by running Σ on g. When $\mathcal{A}_1$ signals to continue, $\mathcal{B}$ obtains P' and assembles Th'_λ from $\hat{\mathsf{Th}}$ and g directly, without further queries to f. It runs $\mathcal{A}_2^{\mathsf{Th}'_\lambda(\cdot,\cdot,\cdot)}(P')$, evaluates Φ on g, and outputs the resulting bit.

We analyze the two cases.

- If $f = f_{P'}$, then $\mathsf{Th}_m(P',\cdot,\cdot) = g_m$ for every member, so Σ and Φ run on the true hidden slice and $\mathsf{Th}'_\lambda = \mathsf{Th}_\lambda$. This is **Game 0** for $\mathcal{A}$.
- If $f = f_0$, then the collection $\mathcal{A}_1$ queries is $\hat{\mathsf{Th}}$ everywhere and is independent of g, while Σ runs on g and g is programmed into the P'-slice for the second stage, with Φ evaluated on g. This is **Game 1** for $\mathcal{A}$.

Since $\mathcal{B}$ outputs what the game reports, its **ZeroDist** advantage equals the difference of the two game probabilities. The numerical bounds follow from Lemma C.8 in the quantum case and Lemma C.7 in the classical one. □

The single-function form is the instance with one member, $\mathsf{Th}_\lambda = \{\mathbf{Th}\}$, in which Σ is the challenge oracle of the property, which answers each challenge query (T, x) that its guards admit with the value of the hidden slice at (T, x) and records the query, and Φ is the winning condition; this is the form used for SM-TCR-EXCL in Section E.

Two features of the lemma are used in Sections D and E. First, q counts only the evaluations of the first stage: the second stage is simulated by $\mathcal{B}_2$, which knows P' and assembles Th'_λ from $\hat{\mathsf{Th}}$ and g without access to f. Second, in **Game 1** the functions g are independent of the collection that $\mathcal{A}_1$ queries directly, so the first stage learns about g only through Σ.

D The Collection and Independence Framework: Experiments and Proofs

This appendix collects the formal collection experiment and the proofs of the three lemmas behind the accounting of Section 4.3 and the modeling of Section 4.4: Lemma 4.1 (Section D.2), Lemma 4.2 (Section D.3), and Lemma 4.3 (Section D.4).

D.1 Formal Definition of Security for a Member of a Collection

In this subsection we spell out the experiment behind Definition 4.12 for one representative property. We state the SM-TCR experiment for a member Th_m of a collection Th_λ of THFs (cf. Section 4.3); the remaining properties of Definition 4.12 are extended in the same way. Compared to the stand-alone experiment of Definition 4.4, the changes are confined to the first stage: $\mathcal{A}_1$ is given a second oracle that evaluates the collection under the same public parameter, and the two oracles enforce the tweak-reservation rule by refusing offending queries. The second stage and the winning condition are unchanged.

Definition D.1 (SM-TCR for a member of a collection)**.** Let $\mathsf{Th}_\lambda = \{\mathsf{Th}_{m_i}\}$ be a collection of THFs with public parameter space $\mathcal{P}$ and tweak space $\mathcal{T}$, and let $\mathsf{Th}_m \in \mathsf{Th}_\lambda$ be the member with message space $\{0,1\}^m$. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}TCR}_{\mathsf{Th}_m \in \mathsf{Th}_\lambda, p}(\mathcal{A})$:

1. Prepare two empty lists Q and Q_λ.
2. Generate a random public parameter $P \leftarrow_\$ \mathcal{P}$.
3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with classical access to two oracles, the challenge oracle $\mathcal{O}(\cdot,\cdot)$ and the collection oracle $\mathsf{Th}_\lambda(P,\cdot,\cdot)$, that work as follows:
   - $\mathcal{O}(\cdot,\cdot)$ takes $T \in \mathcal{T}$ and $\mathsf{m} \in \{0,1\}^m$ as inputs:
     - If $|Q| \geq p$, or there is an $\mathsf{m}' \in \{0,1\}^m$ with $(T, \mathsf{m}') \in Q$, or $T \in Q_\lambda$, or $T \notin \mathcal{T}_m$, return $\perp$.
     - Insert (T, m) into the list Q.
     - Return $\mathsf{Th}_m(P, T, \mathsf{m})$.
   - $\mathsf{Th}_\lambda(P,\cdot,\cdot)$ takes $T \in \mathcal{T}$ and $\mathsf{m} \in \bigcup_i \{0,1\}^{m_i}$ as inputs:
     - If there is an $\mathsf{m}' \in \{0,1\}^m$ with $(T, \mathsf{m}') \in Q$, return $\perp$.
     - If $T \notin \mathcal{T}_{m'}$ for $m' = |\mathsf{m}|$, return $\perp$.
     - Insert T into the list Q_λ.
     - Return $\mathsf{Th}_{m'}(P, T, \mathsf{m})$ for $m' = |\mathsf{m}|$.
4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without access to either oracle.

5. Obtain an output from $\mathcal{A}_2 : (j, \mathsf{m}^*) \leftarrow \mathcal{A}_2(P)$ with $\mathsf{m}^* \in \{0,1\}^m, j \in [|Q|]$. Denote the jth entry in Q by (T_j, m_j).

6. Output 1 if $\mathsf{Th}_m(P, T_j, \mathsf{m}_j) = \mathsf{Th}_m(P, T_j, \mathsf{m}^*)$ and $\mathsf{m}^* \neq \mathsf{m}_j$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability against single-function, multi-target target collision resistance of Th_m as a member of the collection Th_λ as:

$$\mathrm{Succ}^{\text{SM-TCR}}_{\mathsf{Th}_m \in \mathsf{Th}_\lambda, p}(\mathcal{A}) := \Pr[\mathbf{SM\text{-}TCR}_{\mathsf{Th}_m \in \mathsf{Th}_\lambda, p}(\mathcal{A}) \Rightarrow 1].$$

D.2 Proof of the Collection Lemma

Lemma 4.1 (Collection queries are harmless in the ideal model)**.** *Model the collection* Th_λ *as random oracles: for every member* $\mathsf{Th}_m \in \mathsf{Th}_\lambda$*, every* $P \in \mathcal{P}$*, and every* $T \in \mathcal{T}$*, let* $\mathsf{Th}_m(P, T, \cdot)$ *be an independent uniformly random function, which all parties evaluate by oracle queries. Let* Prop *be one of the properties of Definition 4.12, measure the time* ξ *in function evaluations, as fixed in Section 4.3, and let the adversaries access the challenge and collection oracles classically, as Definition 4.12 prescribes. Let* Th_m *be a member of* Th_λ *or* $\mathsf{Th}_{2 \cup l_{\mathsf{sf}}}$ *(Definition 4.12). Then for all* ξ*,* p*, and* q_λ*, and for each of the two adversary classes below, with both sides of every inequality ranging over the same class,*

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p) \;\leq\; \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda),$$

and, in the other direction,

(i) for classical adversaries,

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda) \;\leq\; \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p) \;+\; \frac{2\xi}{|\mathcal{P}|};$$

(ii) for adversaries that evaluate the collection themselves with quantum access, with output space $\{0,1\}^\nu$ *and the XOR oracle (Remark C.2), counted by* ξ*, where* $\xi_1 \leq \xi$ *is the number of own evaluations performed in the first stage,*

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda) \;\leq\; \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p) \;+\; \frac{2\,\xi_1}{\sqrt{|\mathcal{P}|}},$$

where the additive term is $4\,\xi_1/\sqrt{|\mathcal{P}|}$ *for the advantage notions SM-UD and SM-DSPR.*

The additive terms are parameter-guessing artifacts of the proofs.

Every notion of Definition 4.12 grants the adversary the same kind of access: it evaluates the collection on its own, quantumly in part (ii), and accesses the challenge and collection oracles classically.

The proof answers collection queries from a lazily sampled table Λ, and the only thing that can expose this simulation is a first-stage own evaluation at the hidden parameter P'. For classical and quantum adversaries alike, the cost of this exposure is bounded by Lemma C.9, which bounds the cost of replacing an adversary's classical access to a hidden slice $\mathbf{Th}(P', \cdot, \cdot)$ by an independent random function that is programmed into the tweakable hash function when the second stage begins.

Here the hidden slice spans every member of the collection, and the classical interface Σ consists of the challenge oracle together with the collection oracle. Its challenge part may do more than evaluate: SM-UD tosses a bit, SM-PRE samples a hidden preimage, and S-TCR(Prop) grinds a counter. The games **Game 0** and **Game 1**, the probabilities $\Pr[\mathcal{A}^{G_i} \Rightarrow 1]$, the interface Σ, the predicate Φ, and the random family g below are those of Lemma C.9.

Proof. The lower bound holds because an adversary may ignore the collection oracle.

For the upper bounds, fix an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against $\mathsf{Th}_m \in \mathsf{Th}_\lambda$, classical or quantum, with resources (ξ, p, q_λ), and let $\xi_1 \le \xi$ be the number of its own evaluations in the first stage. Write $\epsilon(\xi_1)$ for the largest advantage $\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{B})$ of an adversary $\mathcal{B}$ of the same class as $\mathcal{A}$ that makes at most ξ_1 queries to its oracle.

The experiment of Definition 4.12 is an instance of the games of Lemma C.9: Σ is the challenge oracle of Prop together with the collection oracle, including the guards of Section D.1 that enforce the tweak-reservation rule, and Φ is the winning condition. The experiment as written is then **Game 0**; the proof moves to **Game 1**, removes the collection oracle there, and simulates the result in the stand-alone experiment.

*Step 1: from **Game 0** to **Game 1**.* By Lemma C.9,

$$\left|\Pr[\mathcal{A}^{G_0} \Rightarrow 1] - \Pr[\mathcal{A}^{G_1} \Rightarrow 1]\right| \le \epsilon(\xi_1).$$

*Step 2: removing the collection oracle in **Game 1**.* Without loss of generality $\mathcal{A}$ makes no query that the oracles refuse: whether a query is refused (the reservation rule, a repeated tweak, the bound p, a tweak outside the member's tweak set) is determined by the query and $\mathcal{A}$'s own earlier queries, so $\mathcal{A}$ can skip such a query and continue as if it had received $\perp$, with the same output distribution and no more queries. From $\mathcal{A}$ we construct an adversary $\mathcal{A}'$ that makes no collection queries and makes **Game 1** report 1 with exactly the same probability. Let R be the set of tweaks reserved by challenge queries; it is well defined because Σ is classical. The reservation rule keeps every collection-oracle tweak outside R, and for SM-TCR-EXCL, R also contains the partner tweak $\omega(T_j)$.

$\mathcal{A}'$ runs $\mathcal{A}$ and maintains a table Λ of entries $((m', T, x), y)$, initially empty.

- *Challenge queries.* $\mathcal{A}'$ relays them unchanged.
- *Collection queries.* A query to member $\mathsf{Th}_{m'}$ at tweak T on input x is answered with y if $((m', T, x), y) \in \Lambda$. Otherwise $\mathcal{A}'$ samples $y \leftarrow_\$ \{0,1\}^n$, stores $((m', T, x), y)$, and returns y.
- *Second stage.* $\mathcal{A}'$ receives P' and answers each own evaluation of $\mathcal{A}_2$ with its own oracle, patched at the recorded points: an evaluation of $\mathsf{Th}'_{m'}$ at (P', T, x) with $((m', T, x), y) \in \Lambda$ returns y, and every other evaluation is forwarded unchanged. Against a classical $\mathcal{A}_2$ the patch is a table lookup in front of each forwarded evaluation; against a quantum $\mathcal{A}_2$ it is the standard construction of an oracle that agrees with a given one except on a classical list of points, with one query per forwarded evaluation (Remark C.2).

The resources of $\mathcal{A}'$ are $(\xi, p, 0)$: it makes the challenge queries of $\mathcal{A}$, no collection queries, and at most as many own evaluations as $\mathcal{A}$, since answers served from Λ and the patch itself cost none.

Step 3: simulation in the stand-alone experiment. Since $\mathcal{A}'$ makes no collection queries, an adversary $\mathcal{A}''$ in the stand-alone experiment of Prop for Th_m simulates **Game 1** for $\mathcal{A}'$ exactly. Its oracle O is the random collection that the challenger of the experiment also evaluates.

- *First stage.* $\mathcal{A}''$ samples its own uniformly random collection $\widehat{\mathsf{Th}_\lambda}$, lazily against a classical $\mathcal{A}'$ and as a 2ξ-wise independent function against a quantum one [Zha12]. It runs $\mathcal{A}'_1$, answers each of its own evaluations from $\widehat{\mathsf{Th}_\lambda}$, and relays its challenge queries to the challenge oracle of the experiment.
- *Second stage.* $\mathcal{A}''$ receives P' and runs $\mathcal{A}'_2$. An own evaluation at a parameter $P \ne P'$ is answered from $\widehat{\mathsf{Th}_\lambda}$, an evaluation of Th_m at P' from O (for $\mathsf{Th}_m = \mathsf{Th}_{2 \cup l_{\mathsf{sf}}}$, an evaluation of $\mathbf{H}$ or of $\mathsf{Th}_{l_{\mathsf{sf}}}$ at P', since $\mathsf{Th}_{2 \cup l_{\mathsf{sf}}}$ is combined from the two), and an evaluation of any other member at P' from a fresh random function that $\mathcal{A}''$ simulates in the same way as $\widehat{\mathsf{Th}_\lambda}$. Finally, $\mathcal{A}''$ outputs what $\mathcal{A}'_2$ outputs.

The collection $\widehat{\mathsf{Th}_\lambda}$, the slice of Th_m in O at P', and the fresh functions are independent and distributed as Th, g_m (for $\mathsf{Th}_m = \mathsf{Th}_{2\cup l_{\mathsf{sf}}}$, the pair of slices of $\mathbf{H}$ and $\mathsf{Th}_{l_{\mathsf{sf}}}$, here and in the next sentence), and the remaining members of g in **Game 1**; the latter are read only in the second stage, since $\mathcal{A}'$ makes no collection queries. The stand-alone challenge oracle and winning condition read O only at P', as Σ and Φ read g_m. Hence the view of $\mathcal{A}'$ and the reported bit are identically distributed in both executions, and $\Pr[\mathcal{A}'^{G_1} \Rightarrow 1]$ is the probability that the stand-alone experiment reports 1 for $\mathcal{A}''$.

$\mathcal{A}''$ makes the challenge queries of $\mathcal{A}'$, no first-stage evaluations, and at most one query to O per evaluation of $\mathcal{A}'_2$; simulating $\widehat{\mathsf{Th}_\lambda}$ and the fresh functions evaluates no member of the collection and is not charged (Section 4.3). Its resources are therefore at most (ξ, p). Since O is the whole random collection, the count also holds against a quantum $\mathcal{A}'_2$, with one query to O at the queried input (Remark C.2).

Since Lemma C.9 bounds the shift of a single game probability and only Step 1 costs anything, a success-probability notion (SM-TCR, SM-PRE, S-TCR(Prop), SM-TCR-EXCL) loses at most $\epsilon(\xi_1)$. An advantage notion compares two probabilities: the two branches of SM-UD, or succ and triv of SM-DSPR. Fixing the branch inside Σ and Φ makes each an instance of **Game 0**. Step 1 then costs $\epsilon(\xi_1)$ per branch and Steps 2 and 3 stay exact, with a single $\mathcal{A}''$ serving both branches because it ignores the challenge bit, so the advantage shifts by at most $2\epsilon(\xi_1)$. Taking the maximum over $\mathcal{A}$,

$$\mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m \in \mathsf{Th}_\lambda;\, \xi,\, p,\, q_\lambda) \;\le\; \mathrm{InSec}^{\mathrm{Prop}}(\mathsf{Th}_m;\, \xi,\, p) \;+\; c \cdot \epsilon(\xi_1),$$

with $c = 1$ for the success-probability notions and $c = 2$ for the advantage notions. For quantum adversaries, Lemma C.8 gives $\epsilon(\xi_1) \le 2\xi_1/\sqrt{|\mathcal{P}|}$, hence the terms $2\xi_1/\sqrt{|\mathcal{P}|}$ and $4\xi_1/\sqrt{|\mathcal{P}|}$ of part (ii). For classical adversaries, Lemma C.7 gives $\epsilon(\xi_1) \le \xi_1/|\mathcal{P}|$, hence the terms $\xi_1/|\mathcal{P}|$ and $2\xi_1/|\mathcal{P}|$, both at most $2\xi/|\mathcal{P}|$ as in part (i). □

Remark D.1 (The conversion is query-tight but loose in computation and memory)**.** The simulators $\mathcal{A}'$ and $\mathcal{A}''$ preserve the charged resources of Section 4.3, but not the uncharged ones. $\mathcal{A}'$ stores the table Λ, with up to q_λ entries (2^{62} at the stateless-path magnitudes of Section 5.3), and spends on the order of q_λ steps on it. $\mathcal{A}''$ adds memory on the order of ξ and computation on the order of ξ, or ξ^2 against a quantum adversary. Lemma 4.1 is therefore memory-loose in the sense of [ACFK17], which is not always avoidable [GT20]. This is acceptable here because the target problems are not memory-sensitive (Section 5.2); a memory-sensitive assumption or a cost model that charges local computation (Section 5.1) would require revisiting it.

Note that this is a statement about ideal collections. For the concrete SHRINCS collection, Conjecture 4.1 remains an assumption.

D.3 Proof of the Independence Lemma

The two-oracle picture that opens Section 5.4 also covers the keyed message-hash terms, where independence holds between whole functions instead of tweaks of one collection. Lemma 4.2 formalizes it; since the games of Definitions C.1, 4.9 and 4.11 and Definition C.2 involve no hidden public parameter and no collection oracle, the simulation needs neither the reservation rule nor the parameter-guessing step of Lemma 4.1.

Lemma 4.2 (Evaluations of independent functions are harmless in the ideal model)**.** *Let G be a keyed hash function modeled as a random oracle, and model every further function that an adversary may evaluate as a random oracle independent of G; all parties evaluate all functions by oracle queries, and time is measured in function evaluations. Let* Prop *be a security experiment for G whose challenger, auxiliary oracles, and win condition evaluate G only;* SM-rTCR *(Definition 4.9), adaptive reprogramming (Definition C.1) and its distinct-inputs*

variant (Definition C.2), and ITSR *(Definition 4.11) are of this form. Then for all ξ, p, and $q_G \le \xi$,*

$$\mathrm{InSec}^{\mathrm{Prop}}(G;\, \xi,\, p,\, q_G) = \mathrm{InSec}^{\mathrm{Prop}}(G;\, q_G,\, p),$$

with any further query parameters of the experiment unchanged on both sides: evaluations of the independent functions may be omitted from the accounting. The statement covers classical adversaries and adversaries that evaluate the oracles in superposition.

Proof. The direction $\ge$ holds because an adversary that runs in time q_G satisfies the resource bounds (ξ, p, q_G). For the direction $\le$, fix an adversary $\mathcal{A}$ with resources (ξ, p, q_G). We construct $\mathcal{A}'$ that runs $\mathcal{A}$ and answers every evaluation of a function other than G itself: classically by lazy sampling with private coins, and for superposition access by simulating a fresh random oracle, which is standard using $2q$-wise independent functions or the compressed-oracle technique [Zha12, Zha19]. Interaction involving G [the adversary's own evaluations and the interfaces of the experiment] is forwarded unchanged. The experiment evaluates only G, so the challenger never reads the replaced functions, and the simulation covers the entire run, so no answer of a true oracle can disagree with a simulated one. The joint distribution of $\mathcal{A}$'s view and the challenger's coins is therefore unchanged, and $\mathcal{A}'$ succeeds exactly when $\mathcal{A}$ does. For ITSR, the win condition additionally applies the public map $\mathrm{MAP}_{h,k,t}$, which is no oracle and evaluates no function. $\mathcal{A}'$ evaluates only G, at most q_G times, and answers served from the simulation cost no evaluation, so $\mathcal{A}'$ runs in time q_G with the remaining resource parameters preserved, which concludes the proof. □

D.4 Proof of the Domain-Separation Lemma

Lemma 4.3 (Domain separation of the instantiated functions)**.** *Instantiate the functions of SHRINCS as in Section 2.2: the members F, H, Th_k, $\mathsf{Th}_{l_{\mathsf{sl}}}$, $\mathsf{Th}_{l_{\mathsf{sf}}}$, and $\mathbf{H}_{\mathrm{grind}}$ of the collection Th_λ, the function $\mathbf{PRF}$, the message hashes $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, and the randomizer functions $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$. Evaluations of the collection members, $\mathbf{H}_{\mathrm{grind}}$ included, and $\mathbf{PRF}$ perform one SHA-256 call each; evaluations of the message hashes and the randomizer functions perform two nested calls. These functions are domain separated on inputs whose public seed is that of the generated key, whose root, for each message hash, is the generated root of its own path, and whose address type, for the functions that take one, is one of the function's own types (Table 2); the randomizer, message, idx, counter, remaining address fields, and every other argument are arbitrary: two SHA-256 calls made by two different functions, or by one member of Th_λ at two different tweaks, on such inputs receive the same input only in the following events:*

- *(i) the inner calls of the two message hashes receive the same input; this requires $\mathsf{PK.sl} = \mathsf{PK.sf}$;*
- *(ii) a call of a message hash and a padded-seed call (Th_λ, $\mathbf{PRF}$, $\mathbf{H}_{\mathrm{grind}}$) or an HMAC call of a randomizer function receive the same input; this requires $\mathsf{PK.seed}$ and one of the two path roots to equal matching 16-byte constants simultaneously;*
- *(iii) an outer call of one message hash and an inner call of the other receive the same input; this requires an evaluated digest whose first half equals the other path's root. Within the scheme's own calls the message wrappers strengthen the requirement in both directions (Remark 4.2):*
  - *(iii.a) the outer call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ never matches an inner call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$: the wrapped message makes every honest inner string of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ at least 75 bytes long, longer than both outer strings (68 and 73 bytes);*
  - *(iii.b) the outer call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ matches an inner call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ only if, in addition to the digest half equal to $\mathsf{PK.sl}$, the wrapper's prefix bytes and the embedded root $\mathsf{PK.sf}$ reappear inside the digest and idx bytes.*

Proof. For each function, and for each member of Th_λ at each tweak, collect the inputs of all SHA-256 calls that its evaluations perform into one *input family*; the lemma states that these families are pairwise disjoint, except in events (i)–(iii). Fix two families and ask whether one byte string can lie in both. Two strings of distinct lengths never coincide, so a pair of families can only meet at a length that both attain. At equal length, the byte positions split into two kinds: *free* positions, which an evaluator may set at will (the randomizer R, message bytes, counters, addresses within a path, the idx input), and *fixed* positions, which carry key-generation values or structural constants ($\mathsf{PK.seed}$, the path roots, the zero padding, the pad-masked HMAC key blocks, the type bytes). Free positions agree at no cost, so an overlap requires all fixed positions to agree simultaneously. We now go through the pairs, in three groups of calls: padded seed (Th_λ, $\mathbf{PRF}$, $\mathbf{H}_{\mathrm{grind}}$), HMAC ($\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$), and message hash ($\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$).

The figures interleaved with this proof depict every SHA-256 invocation of $\mathsf{SHRINCS}$ at the byte level, one figure per call. In each figure, the top row lists the input fields in order, drawn to scale (field widths are proportional to their lengths in bytes, given in gray above each field), and the bottom row shows how the input splits into 64-byte SHA-256 blocks; dashed lines mark block boundaries that fall inside a field. The SHA-256 padding is drawn explicitly: the byte $\mathtt{0x80}$, a zero fill, and the final 8-byte length field (len), which holds the input length in bits. In these figures, exponents count *bytes*: 0^{48} denotes 48 zero bytes.

For the two-call functions ($\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$, and the HMAC-based $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$, $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$), h_{in} denotes the untruncated 32-byte digest of the corresponding inner call, following Section A. A slashed break elides part of the input: the variable-length message M of an inner call, or the run of 16-byte chunks in Th_{10}, Th_{32}, and Th_{35}. Fields after a break are schematic rather than to scale, the zero fill $0 \cdots 0$ completes the final block, and B denotes the resulting number of blocks. The inner-call figures show the message argument as $\mathsf{SHRINCS}$ evaluates it (Section 3): on either path the message field is the bound message

$$\mathsf{toByte}(0,1) \,\|\, \mathsf{toByte}(|ctx|,1) \,\|\, ctx \,\|\, root \,\|\, M,$$

where *root* is the other path's root ($\mathsf{PK.sl}$ on the stateful path, $\mathsf{PK.sf}$ on the stateless one), so each digest binds both roots and the caller's context; the randomizers absorb the same bound messages as their digests. In the inner-call figures, a brace beneath the block row marks this wrapped message: the bound message $\widetilde{M}$ on the stateful path (Section 3) and the SLH-DSA wrapper output M' on the stateless path (Section 2.7). The figures depict the default context $ctx = \varepsilon$, where the wrapper prefix is $\mathtt{0x00}^2$; a nonempty context inserts its bytes between the length byte and the root and shifts the subsequent offsets by $|ctx| \le 255$ bytes. In the padded-seed figures (Figs. 5 to 11), block 1 is the precomputed compression $P_{\mathsf{PK.seed}}$ of Section 2.2; in the HMAC figures, the key block 1 likewise depends only on $\mathsf{SK.prf}$ and can be precomputed.

Bytes 16–31 of the input are shaded in every figure. As discussed in the domain-separation paragraph of Section 2.2, these sixteen bytes already separate the three groups of calls: they are zero in every padded-seed call, they equal $\mathsf{PK.seed}$ in the four message-digest calls (Figs. 16 to 19), and they carry one of the pairwise-distinct non-zero constants $\mathtt{0x36}^{16}$, $\mathtt{0x5C}^{16}$, $\mathtt{0xC9}^{16}$, $\mathtt{0xA3}^{16}$ in the four HMAC calls (Figs. 12 to 15).

Padded seed against padded seed (Figs. 5 to 11). Every padded-seed string carries $\mathsf{PK.seed}$ at bytes 0–15 and the zero padding at bytes 16–63. Within one member, distinct tweaks give distinct strings: the tweak is the address, and the address occupies bytes 64–85 (bytes 64–73 for $\mathbf{H}_{\mathrm{grind}}$, whose tweak, an address of type $\mathtt{SF_WOTS_C_GRIND}$ with zero payload, enters through its 10-byte prefix). Across distinct functions of the group, the input lengths separate every pair except one: the lengths are 102 bytes for $F = \mathsf{Th}_1$ and for $\mathbf{PRF}$, 112 for $\mathbf{H}_{\mathrm{grind}}$, 118 for $H = \mathsf{Th}_2$, and 246, 598, 646 for Th_k, $\mathsf{Th}_{l_{\mathsf{sf}}}$, $\mathsf{Th}_{l_{\mathsf{sl}}}$. The remaining pair, $\mathbf{PRF}$ against F, is separated by the address type byte: $\mathbf{PRF}$ occurs only with the types $\mathtt{SL_WOTS_TW_PRF}$, $\mathtt{SL_FORS_PRF}$, and $\mathtt{SF_WOTS_C_PRF}$, none of which occurs with F (Fig. 6).

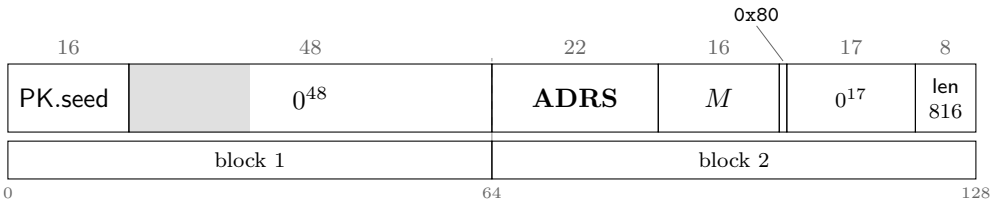


Figure 5: $F = \mathsf{Th}_1(\mathsf{PK.seed}, \mathbf{ADRS}, M)$: hash-chain steps (WOTS-TW, WOTS+C) and FORS leaf hashing. The input $\mathsf{PK.seed} \| 0^{48} \| \mathbf{ADRS} \| M$ is 102 bytes and occupies two blocks; block 1 is the precomputed $P_{\mathsf{PK.seed}}$, so each call costs one fresh compression.

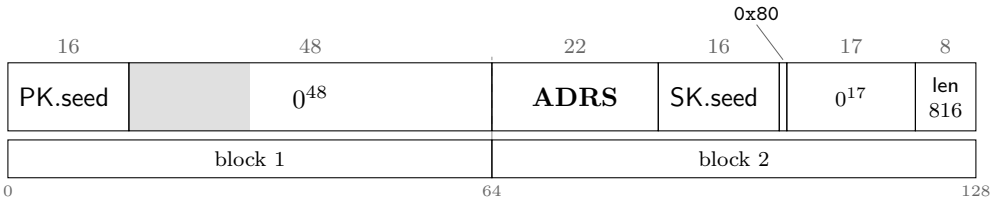


Figure 6: $\mathbf{PRF}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathbf{ADRS})$: derivation of WOTS-TW, WOTS+C, and FORS secret values. Identical 102-byte shape as F (Fig. 5), with SK.seed in place of M; the two families are separated by the address type byte (`SL_WOTS_TW_PRF`, `SL_FORS_PRF`, `SF_WOTS_C_PRF`).

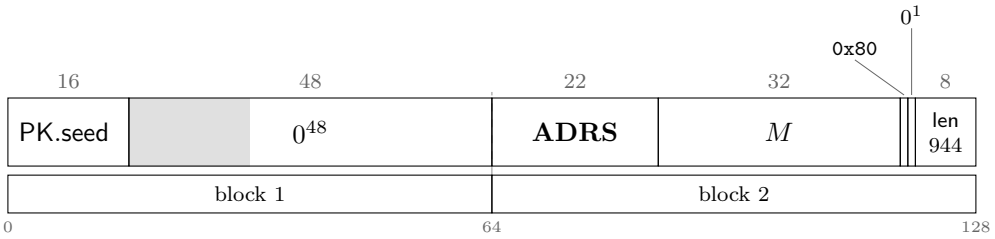


Figure 7: $H = \mathsf{Th}_2(\mathsf{PK.seed}, \mathbf{ADRS}, M)$: internal nodes of every Merkle tree (XMSS, FORS, FXMSS). The 118-byte input leaves room for a single zero byte of padding before the length field.

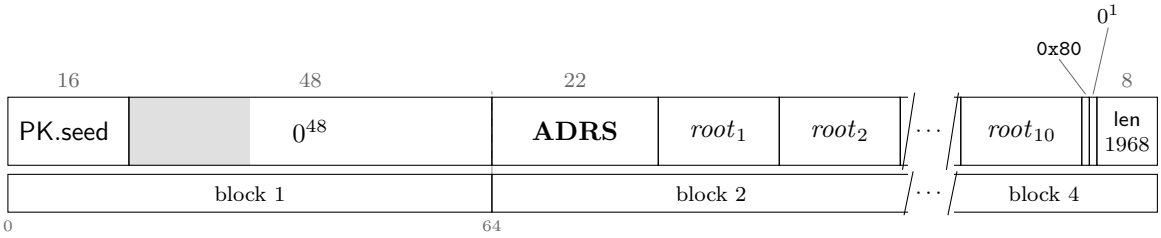


Figure 8: Th_{10}: compression of the $k = 10$ FORS roots (16 bytes each) into the FORS public key. The 246-byte input occupies four blocks (three fresh compressions after $P_{\mathsf{PK.seed}}$); $root_3$–$root_9$ and block 3 are elided, and the drawing is schematic across the break.

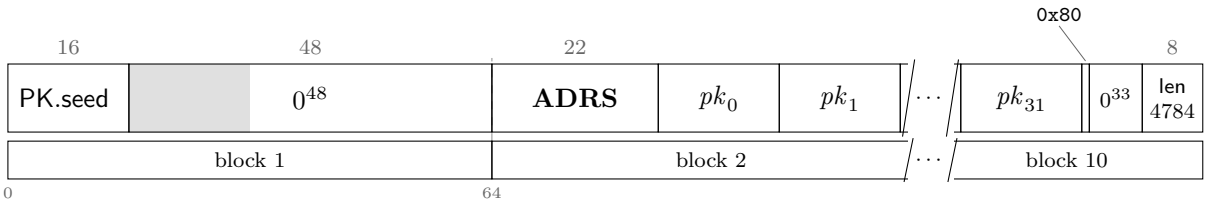


Figure 9: Th_{32}: compression of a WOTS+C public key ($l_{\mathsf{sf}} = 32$ chain-end values pk_i, 16 bytes each) on the stateful path. The 598-byte input occupies ten blocks; pk_2–pk_{30} and blocks 3–9 are elided, and the drawing is schematic across the break.

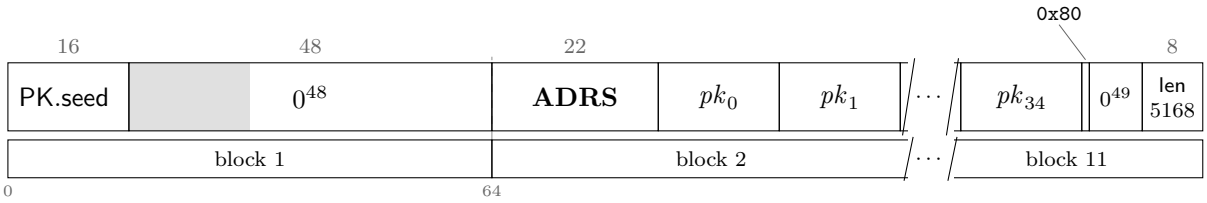


Figure 10: Th_{35}: compression of a WOTS-TW public key ($l_{\mathsf{sl}} = 35$ chain-end values pk_i, 16 bytes each) on the stateless path. The 646-byte input occupies eleven blocks; pk_2–pk_{33} and blocks 3–10 are elided, and the drawing is schematic across the break.

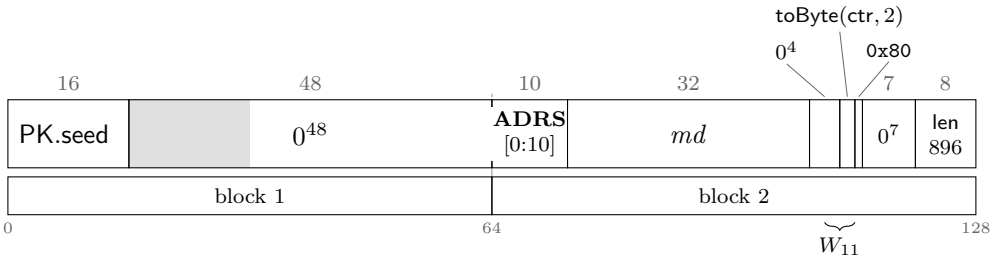


Figure 11: $\mathbf{H}_{\text{grind}}(\mathsf{PK.seed}, \mathbf{ADRS}[0{:}10], md, \mathsf{ctr})$: grinding of the stateful message digest. The 112-byte input fits in two blocks, so each attempt costs one fresh compression. The four zero bytes align ctr with a message-schedule word boundary: the two counter bytes are the only non-zero bytes of word W_{11} of block 2 (block bytes 44–47).

Padded seed against HMAC, and HMAC against HMAC (Figs. 12 to 15). Every HMAC string carries a masked key block at bytes 0–63, and its bytes 16–63 hold one of the four constants $\texttt{0x36}^{48}$ (inner call of $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$), $\texttt{0x5C}^{48}$ (outer call of $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$), $\texttt{0xC9}^{48}$ (inner call of $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$), $\texttt{0xA3}^{48}$ (outer call of $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$). A padded-seed string holds 0^{48} at the same bytes, so the two groups differ at bytes 16–63 of every string. Within the HMAC group, the only pair of families is $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ against $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$, and each of the four call-to-call comparisons meets two distinct constants at bytes 16–63, so the two families are disjoint unconditionally. (The inner and the outer call of one selector belong to the same family, so their coincidence lies outside the pairwise statement; it cannot occur regardless: ipad and opad differ in every byte, so the two masked key blocks differ at each of bytes 0–63.)

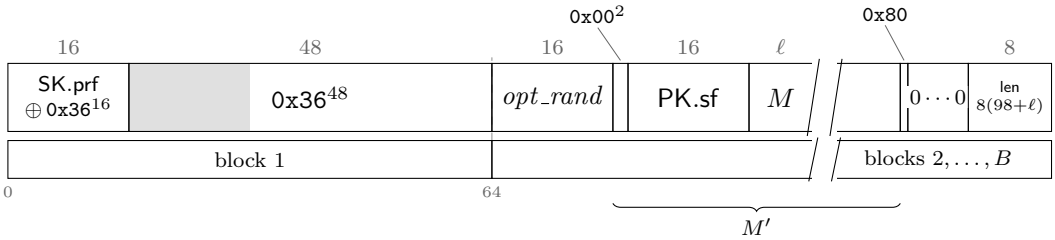


Figure 12: Inner call of $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$ (HMAC-SHA-256, Section A): block 1 is the ipad-masked key block $(\mathsf{SK.prf}\|0^{48}) \oplus \mathtt{0x36}^{64}$, followed by the HMAC text $\mathit{opt_rand}\|\mathtt{0x00}^2\|\mathsf{PK.sf}\|M$ — SLH-DSA derives the randomizer from the same wrapped message as in Fig. 16. The layout after the break is schematic.

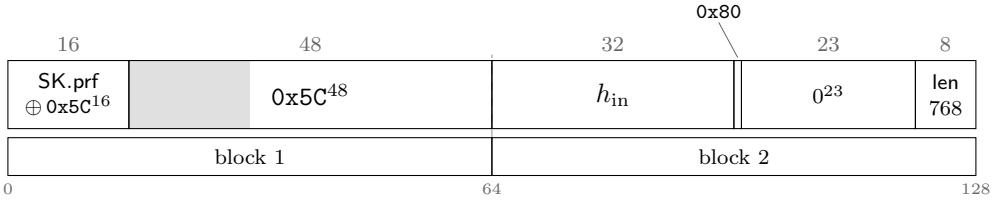


Figure 13: Outer call of $\mathbf{PRF}^{\mathsf{sl}}_{\mathbf{msg}}$: the opad-masked key block $(\mathsf{SK.prf}\|0^{48}) \oplus \mathtt{0x5C}^{64}$ followed by the 32-byte inner digest h_{in}; 96 bytes in total. Block 1 depends only on SK.prf and can be precomputed.

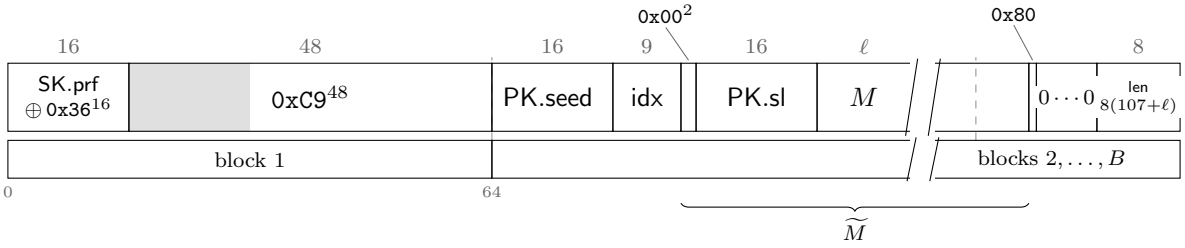


Figure 14: Inner call of $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$: the HMAC key is $\mathsf{SK.prf}\|\mathtt{0xFF}^{48}$, so the ipad-masked key block continues with $\mathtt{0xC9}^{48}$ ($\mathtt{0xC9} = \mathtt{0xFF} \oplus \mathtt{0x36}$); the HMAC text is $\mathsf{PK.seed}\|\mathsf{idx}\|\widetilde{M}$ with the bound message $\widetilde{M}$ shown at the default context $ctx = \varepsilon$: the randomizer absorbs the same bound message that feeds $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ (Fig. 18). The layout after the break is schematic.

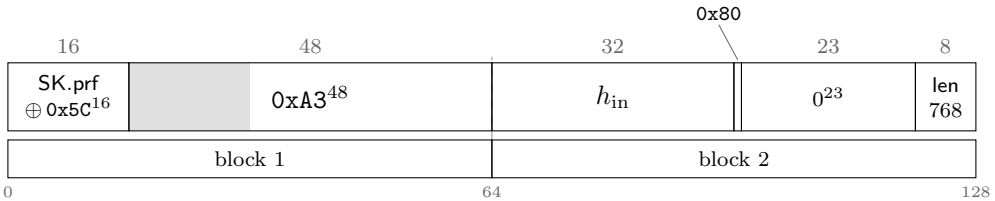


Figure 15: Outer call of $\mathbf{PRF}^{\mathsf{sf}}_{\mathbf{msg}}$: the opad-masked key block continues with $\mathtt{0xA3}^{48}$ ($\mathtt{0xA3} = \mathtt{0xFF} \oplus \mathtt{0x5C}$), followed by h_{in}; 96 bytes in total, as in Fig. 13, but with a distinct key block.

Message hash against padded seed and against HMAC (Figs. 16 to 19). The outer message-hash strings have fixed lengths 68 and 73 bytes, while every padded-seed string has at least 102 bytes and every HMAC string at least 80; the outer calls therefore meet neither group. The inner strings have free length, through the message, and are compared position by position. Against a padded-seed string: bytes 16–31 hold PK.seed on the inner side and zero padding on

the padded-seed side, forcing $\mathsf{PK.seed} = 0^{16}$; bytes 32–47 hold the own path root on the inner side and zero padding on the other, forcing additionally $\mathsf{PK.sl} = 0^{16}$ (for $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$) or $\mathsf{PK.sf} = 0^{16}$ (for $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$); bytes 0–15 cost nothing, since the inner side holds the free randomizer R, and every later position is free on at least one side. Against an HMAC string, the same two positions force PK.seed and the root onto the pad constant of that call: $\mathsf{PK.seed} = c^{16}$ and the root $= c^{16}$ for the c of that call. This enumerates ten pairings, each demanding two independent key-generation values to hit prescribed 16-byte constants: two with the padded-seed group ($\mathsf{PK.seed} = 0^{16}$ together with $\mathsf{PK.sl} = 0^{16}$, respectively $\mathsf{PK.sf} = 0^{16}$) and eight with the HMAC group (the four constants, for each of the two message hashes). Together they form event (ii).

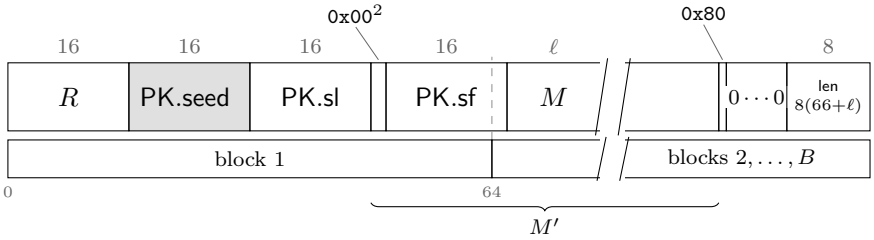


Figure 16: Inner call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ on the message SHRINCS actually evaluates: the stateless path signs $\mathsf{PK.sf}\|M$ (Algorithm 2), shown at the default context $ctx = \varepsilon$, so $h_{\text{in}} = \mathsf{SHA\text{-}256}(R\|\mathsf{PK.seed}\|\mathsf{PK.sl}\|\mathsf{0x00}^2\|\mathsf{PK.sf}\|M)$, where $\mathsf{0x00}^2$ is the wrapper prefix $\mathsf{toByte}(0,1)\|\mathsf{toByte}(|ctx|,1)$. The digest binds both roots, and block 1 ends inside the prepended PK.sf; the layout after the break is schematic.

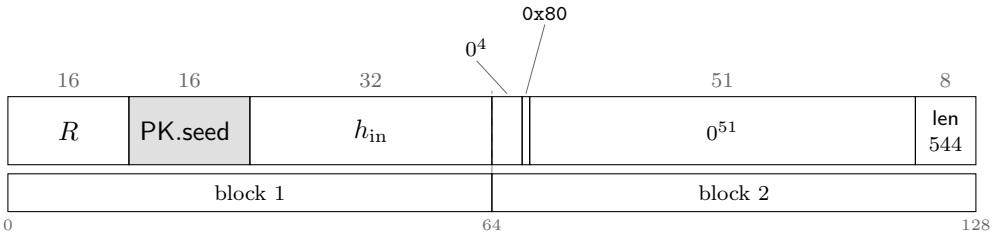


Figure 17: Outer call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$: $\mathsf{SHA\text{-}256}(R\|\mathsf{PK.seed}\|h_{\text{in}}\|0^4)$, 68 bytes. Block 1 is exactly $R\|\mathsf{PK.seed}\|h_{\text{in}}$; block 2 carries only the 4-byte MGF1 block counter $\mathsf{toByte}(0,4)$ and the SHA-256 padding.

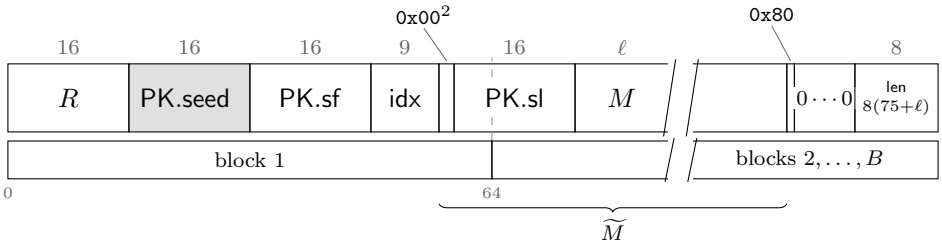


Figure 18: Inner call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ on the bound message SHRINCS actually evaluates (Algorithm 2), shown at the default context $ctx = \varepsilon$: $h_{\text{in}} = \mathsf{SHA\text{-}256}(R\|\mathsf{PK.seed}\|\mathsf{PK.sf}\|\mathsf{idx}\|\mathsf{0x00}^2\|\mathsf{PK.sl}\|M)$, where $\mathsf{0x00}^2$ is the wrapper prefix $\mathsf{toByte}(0,1)\|\mathsf{toByte}(|ctx|,1)$. The digest binds both roots, the caller's context, and the 9-byte leaf identifier idx, and block 1 ends inside the prepended PK.sl; the layout after the break is schematic.

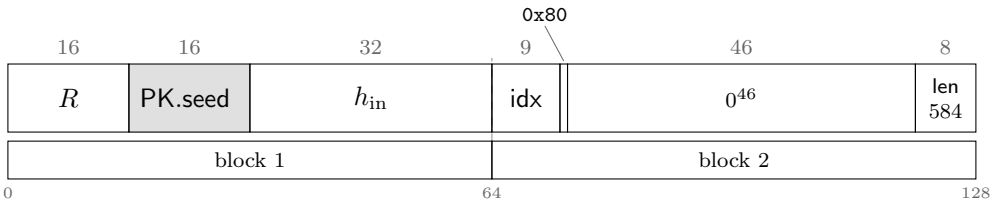


Figure 19: Outer call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$: SHA-256($R\|\mathsf{PK.seed}\|h_{\text{in}}\|\mathsf{idx}$), 73 bytes. Block 1 is exactly $R\|\mathsf{PK.seed}\|h_{\text{in}}$; block 2 carries only the 9-byte leaf identifier idx and the SHA-256 padding. Compare Fig. 17: the input lengths 68 and 73 keep the two outer calls distinct.

Message hash against message hash. The two outer calls have distinct fixed lengths, 68 and 73 bytes (Figs. 17 and 19). An inner call of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ and an inner call of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ (Figs. 16 and 18) hold the free randomizers at bytes 0–15 and the shared PK.seed at bytes 16–31; the lengths close through the free messages, and every position from byte 48 on is free on at least one side. The one obstruction sits at bytes 32–47, where the two strings hold their own roots; the strings coincide exactly when $\mathsf{PK.sl} = \mathsf{PK.sf}$. This is event (i).

The remaining pairwise case is an outer call of one message hash read as an inner call of the other (the analogous reading within a single message hash is a self-intersection of one family and concerns the internal structure of one modeled oracle rather than the separation of two; see Remark 4.2). An outer string is $R \,\|\, \mathsf{PK.seed} \,\|\, h_{\text{in}} \,\|\, t$, with h_{in} the 32-byte inner digest of the same evaluation at bytes 32–63 and t the trailing word (0^4 for $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$, idx for $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$). Consider first the calls of the scheme itself (Figs. 16 and 18): every honest inner call digests the bound message, whose wrapper places the prefix bytes and the opposite path's root in front of the message (Section 3). One direction is excluded by length alone: an honest inner string of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ carries the wrapped message of at least 18 bytes after idx and is therefore at least 75 bytes long, above both outer lengths; this settles (iii.a). In the other direction, reading an outer string of $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ as an honest inner string of $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ aligns bytes 32–47, where the outer side holds the digest half $h_{\text{in}}[0{:}16]$ and the inner side holds PK.sl, and bytes 48–72, where the inner side holds the wrapper's prefix bytes and the embedded PK.sf against $h_{\text{in}}[16{:}32] \,\|\, \mathsf{idx}$. The reading therefore requires the digest half to equal the root and, additionally, the wrapper bytes to reappear inside the digest and idx; this is (iii.b).

The wrapping, however, is applied by the scheme, and the definition of the functions does not enforce it: the message argument is arbitrary, so we discuss it here for completeness. For those, only the fixed positions constrain the reading: bytes 0–31 are free or shared as before, every position from byte 48 on is free on at least one side, and bytes 32–47 place the digest half $h_{\text{in}}[0{:}16]$ against the other path's root. Outer strings exist only for evaluated calls, since h_{in} is a computed digest, so without the wrapping the overlap occurs exactly when an evaluated digest half equals the other root, in either direction; this is the general condition of event (iii). This concludes the proof. □

E A Generic Bound for Multi-Target Target Collision Resistance with Exclusive Tweaks

This section bounds the success probability of an adversary against SM-TCR-EXCL. The argument follows the analysis of SM-TCR in [HK22], extended to the paired tweaks. Its main tool is the reprogramming of a (quantumly) accessible oracle at a hidden public parameter, whose cost is measured by the experiment of Definition C.4. For reference, we restate Definition 4.7.

Definition 4.7 (Single-Function, Multi-Target Target Collision Resistance with Exclusive Tweaks (SM-TCR-EXCL))**.** Let $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ be a tweakable hash function. Let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$, where $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$ if $i \neq j$ and $|\mathcal{T}_2| = |\mathcal{T}_3|$. Let $\omega : \mathcal{T}_2 \to \mathcal{T}_3$ be a bijection. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a

two-stage algorithm, where $\mathcal{A}_1$ and $\mathcal{A}_2$ share a state. Let $p \in [|\mathcal{T}_1 \cup \mathcal{T}_2|]$. Consider the following experiment $\textbf{SM-TCR-EXCL}_{\mathbf{Th},p}(\mathcal{A})$:

1. Prepare an empty list Q.
2. Generate a random public parameter $P \leftarrow_\$ \mathcal{P}$.
3. Run the first stage of $\mathcal{A}$ called $\mathcal{A}_1$ with classical access to an oracle $\mathcal{O}(\cdot,\cdot)$ that takes $T \in \mathcal{T}$ and $\mathsf{m} \in \mathcal{M}$ as inputs and works as follows:
   - If $|Q| \geq p$ return $\perp$.
   - If $T \in \mathcal{T}$ and there is an $\mathsf{m}' \in \mathcal{M}$ with $(T, \mathsf{m}') \in Q$ return $\perp$.
   - If $T \in \mathcal{T}_2$ and there is an $\mathsf{m}' \in \mathcal{M}$ with $(\omega(T), \mathsf{m}') \in Q$ return $\perp$.
   - If $T \in \mathcal{T}_3$ and there is an $\mathsf{m}' \in \mathcal{M}$ with $(\omega^{-1}(T), \mathsf{m}') \in Q$ return $\perp$.
   - Insert (T, m) into the list Q
   - Return $\mathbf{Th}(P, T, \mathsf{m})$.
4. When $\mathcal{A}$ signals to continue, then continue to the second stage of $\mathcal{A}$ called $\mathcal{A}_2$ with input P, but without the oracle access.
5. Obtain an output from $\mathcal{A}_2$: $(j, T^*, \mathsf{m}^*) \leftarrow \mathcal{A}_2(P)$ with $j \in [|Q|]$, $T^* \in \mathcal{T}$ and $\mathsf{m}^* \in \mathcal{M}$. Denote the jth entry in Q by (T_j, m_j).
6. If $T_j \in \mathcal{T}_1$ and $T^* \neq T_j$: Output 0.
7. If $T_j \in \mathcal{T}_2$ and ($T^* \neq T_j$ and $T^* \neq \omega(T_j)$): Output 0.
8. If $T_j \in \mathcal{T}_3$ and ($T^* \neq T_j$ and $T^* \neq \omega^{-1}(T_j)$): Output 0.
9. Output 1 if $\mathbf{Th}(P, T_j, \mathsf{m}_j) = \mathbf{Th}(P, T^*, \mathsf{m}^*)$ and $\mathsf{m}^* \neq \mathsf{m}_j$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the success probability against single-function, multi-target target collision resistance with exclusive tweaks of a tweakable hash function as:

$$\mathrm{Succ}^{\text{SM-TCR-EXCL}}_{\mathbf{Th},p}(\mathcal{A}) := \Pr[\textbf{SM-TCR-EXCL}_{\mathbf{Th},p}(\mathcal{A}) \Rightarrow 1].$$

The bound has two summands: one from the ZeroDist experiment and one from the search problem $\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}$ of Definition C.3. The proof takes two steps. In the first step, the challenge queries are answered by an independent random function instead of **Th**, and this function is programmed into **Th** at the public parameter when the adversary enters its second stage. The adversary can notice the change only by solving ZeroDist. In the second step, the independence of the new function is used to hide a search instance in it, so that every collision found by the adversary yields a solution of $\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}$.

The first step is an instance of the hidden-slice reprogramming of Lemma C.9 for the single tweakable hash function **Th**. In this instance, Σ is the challenge oracle of SM-TCR-EXCL together with its guards, which answers an admitted query (T_i, m_i) with the value of the hidden slice at (T_i, m_i), and Φ is the winning condition. The lemma bounds the cost of replacing the hidden slice by an independent random function g by the advantage of a ZeroDist adversary that makes at most q queries.

We now state and prove the bound.

Theorem E.1. *Let* $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ *be a tweakable hash function modeled as a (quantum) random oracle. Let* $\mathcal{A}$ *be an adversary against* SM-TCR-EXCL *for* $p \leq |\mathcal{T}|$ *targets that makes* q *queries to* **Th**. *Then there are an adversary* $\mathcal{B}$ *against* $\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}$ *that makes at most* q *queries to its challenge oracle and an adversary* $\mathcal{C}$ *against* ZeroDist *that makes at most* q *queries to its challenge oracle such that*

$$\mathrm{Succ}^{\text{SM-TCR-EXCL}}_{\mathbf{Th},p}(\mathcal{A}) \leq \mathrm{Succ}^{\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{C})$$

- *If $\mathcal{A}$ is a quantum adversary that makes quantum queries, with $\mathcal{N} = \{0,1\}^\nu$ and the XOR oracle (Remark C.2), then $\mathcal{B}$ and $\mathcal{C}$ are quantum, and*

$$\mathrm{Succ}^{\text{SM-TCR-EXCL}}_{\mathbf{Th},p}(\mathcal{A}) \le \frac{8(q+1)^2}{|\mathcal{N}|} + \frac{2q}{\sqrt{|\mathcal{P}|}}\,.$$

- *If $\mathcal{A}$ is a classical adversary that makes classical queries, then $\mathcal{B}$ and $\mathcal{C}$ are classical, and*

$$\mathrm{Succ}^{\text{SM-TCR-EXCL}}_{\mathbf{Th},p}(\mathcal{A}) \le \frac{q+1}{|\mathcal{N}|} + \frac{q}{|\mathcal{P}|}\,.$$

Proof. The proof is a sequence of two games. We write $\mathrm{Succ}^{\mathbf{Game}.i}_{\mathbf{Th},p}(\mathcal{A})$ for the probability that **Game.**i outputs 1. For a tweak $T \in \mathcal{T}$, let $\Omega(T)$ be the set of tweaks at which Definition 4.7 accepts a collision with a challenge at T: $\Omega(T) = \{T\}$ if $T \in \mathcal{T}_1$, $\Omega(T) = \{T, \omega(T)\}$ if $T \in \mathcal{T}_2$, and $\Omega(T) = \{T, \omega^{-1}(T)\}$ if $T \in \mathcal{T}_3$.

Game.0: This is the experiment $\mathbf{SM\text{-}TCR\text{-}EXCL}_{\mathbf{Th},p}(\mathcal{A})$ of Definition 4.7, so

$$\mathrm{Succ}^{\text{SM-TCR-EXCL}}_{\mathbf{Th},p}(\mathcal{A}) = \mathrm{Succ}^{\mathbf{Game.0}}_{\mathbf{Th},p}(\mathcal{A}).$$

Game.1: This is **Game 1** of Lemma C.9, instantiated with the experiment of **Game.0**. A random function $g\colon \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ answers the challenge queries: the admitted queries $\{(T_i, \mathsf{m}_i)\}_i$ receive $y_i = g(T_i, \mathsf{m}_i)$. When the second stage begins, g is programmed into **Th** at P', which yields the function $\mathbf{Th}'$ of the lemma. In short, **Game.1** is:

1. $P' \leftarrow_\$ \mathcal{P}$; $g \leftarrow_\$ \{g\colon \mathcal{T} \times \mathcal{M} \to \mathcal{N}\}$; $\mathcal{A}_1^{\mathbf{Th}(\cdot,\cdot,\cdot),\ \Sigma^{g(\cdot,\cdot)}}(\ )$, where Σ is the challenge oracle of Definition 4.7 with $\mathbf{Th}(P',\cdot,\cdot)$ replaced by g;
2. $(j, T^*, \mathsf{m}^*) \leftarrow \mathcal{A}_2^{\mathbf{Th}'(\cdot,\cdot,\cdot)}(P')$;
3. Output 1 if $T^* \in \Omega(T_j)$, $\mathbf{Th}'(P', T_j, \mathsf{m}_j) = \mathbf{Th}'(P', T^*, \mathsf{m}^*)$, and $\mathsf{m}^* \neq \mathsf{m}_j$; Output 0 otherwise.

The two games are **Game 0** and **Game 1** of Lemma C.9 for this choice of Σ and Φ, and $\mathcal{A}$ makes at most q queries to **Th** in its first stage. The lemma therefore provides an adversary $\mathcal{C}$ against ZeroDist that makes at most q queries, runs $\mathcal{A}$, and outputs the bit the game reports, such that

$$|\mathrm{Succ}^{\mathbf{Game.0}}_{\mathbf{Th},p}(\mathcal{A}) - \mathrm{Succ}^{\mathbf{Game.1}}_{\mathbf{Th},p}(\mathcal{A})| \le \mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{C}).$$

It remains to bound the success probability in **Game.1**, which is done by a reduction from $\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}$. Let $\mathcal{B}$ be the algorithm of Algorithm 13. It has oracle access to a function $f \leftarrow D_{1/|\mathcal{N}|}$ on $\{1,2,3\} \times [p] \times \mathcal{M}$ and runs **Game.1** for $\mathcal{A}$. The admitted challenge queries are answered with independent uniform values y_i, and the function g that $\mathcal{B}$ programs at P' takes the value y_i at the challenge input (T_i, m_i). At every other input (t, m) with $t \in \Omega(T_i)$ and $t \in \mathcal{T}_k$, the bit $f(k, i, \mathsf{m})$ is hidden in g: the value is y_i if the bit is 1 and a uniform element of $\mathcal{N} \setminus \{y_i\}$ otherwise. Since $f(k,i,\mathsf{m}) = 1$ holds with probability $1/|\mathcal{N}|$, each such value is uniform on $\mathcal{N}$. The guards of the challenge oracle make the sets $\Omega(T_i)$ of distinct challenges disjoint, so each input of g is governed by at most one challenge, and the values of g at distinct inputs are independent. Hence g is a uniformly random function, and $\mathcal{B}$ runs **Game.1** for $\mathcal{A}$ exactly. $\mathcal{B}$ queries f only when $\mathcal{A}_2$ queries $\mathbf{Th}'$ at P', and the input of such a query determines the single point (k, i, m) at which f is read. Each query of $\mathcal{A}$ thus costs $\mathcal{B}$ at most one query to f, counted as one invocation as in Lemma C.9 (Remark C.2), and $\mathcal{B}$ makes at most q queries in total.

Suppose that $\mathcal{A}$ wins **Game.1** with the output (j, T^*, m^*), and let k be the index with $T^* \in \mathcal{T}_k$. Then $T^* \in \Omega(T_j)$, $\mathsf{m}^* \neq \mathsf{m}_j$, and $\mathbf{Th}'(P', T^*, \mathsf{m}^*) = \mathbf{Th}'(P', T_j, \mathsf{m}_j) = y_j$. Since $\mathsf{m}^* \neq \mathsf{m}_j$, the input (T^*, m^*) is not the challenge input, so g takes the value y_j there only if $f(k, j, \mathsf{m}^*) = 1$; if

$f(k, j, \mathsf{m}^*) = 0$, the value is $g_j(k, \mathsf{m}^*) \neq y_j$. The output (k, j, m^*) of $\mathcal{B}$ is therefore a solution of $\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}$ whenever $\mathcal{A}$ wins, and

$$\mathrm{Succ}^{\mathbf{Game.1}}_{\mathbf{Th},p}(\mathcal{A}) \leq \mathrm{Succ}^{\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}}(\mathcal{B}).$$

The two inequalities together give the first bound of the theorem. For a quantum adversary, Lemma C.6 with $\lambda = 1/|\mathcal{N}|$ bounds the first summand by $8(q+1)^2/|\mathcal{N}|$, and Lemma C.9 bounds the second by $2q/\sqrt{|\mathcal{P}|}$. For a classical adversary, the same two lemmas give $(q+1)/|\mathcal{N}|$ and $q/|\mathcal{P}|$. □

F Generic Bounds for Interleaved Target-Subset Resilience

The stateless path relies on the interleaved target-subset resilience of the message digest $\mathbf{H}^{\mathsf{sl}}_{\mathbf{msg}}$ (Theorem 4.3). The generic quantum bound of [YTA23, Theorem 4] has the prefactor $8(q+p+2)^2$ on Γ, where q is the number of hash queries and p the number of challenge queries. Its proof, which follows that of Theorem 3 of the same paper, first reprograms the random oracle at the challenges, charging $q+p+1$ queries correctly, and then reduces to average-case search with the same count, although that reduction answers the challenge queries with targets sampled in advance, so they never reach the search oracle. With the count corrected, p enters the search part only through Γ. The search parts then match, without the challenge count, the classical generic-attack analysis of [BHK+19, Section 5], $(q+1)\Gamma$, and the quantum hardness conjectured in the table of [HK22].

We use the experiment of Definition 4.11 and, as in Table 6, write p for its challenge count and q for the number of queries of $\mathcal{A}$ to H, which is modeled as a random oracle with classical or quantum access. We assume that $\mathrm{MAP}_{h,k,t}$ maps the uniform distribution on $\{0,1\}^m$ to the uniform distribution on $[0, 2^h-1] \times [0, t-1]^k$. The byte-slicing map of FIPS 205 [Nat24], which SHRINCS uses on the stateless path (Section 3), has this property: every component is either a prefix of independent uniform bits or the residue of a uniform integer modulo a power of two. The probability Γ that a fresh uniform digest is covered by p uniform targets is computed in [BHK+19, Section 5], counting only targets with the same tree index.

Lemma F.1 ([BHK+19])**.** *Let $(I_0, J_{0,1}, \ldots, J_{0,k})$ and $(I_j, J_{j,1}, \ldots, J_{j,k})$ for $j \in [p]$ be independent and uniformly distributed over $[0, 2^h-1] \times [0, t-1]^k$. Then*

$$\Pr\Big[\{(I_0, i, J_{0,i})\}_{i\in[k]} \subseteq \bigcup_{j\in[p]} \{(I_j, i, J_{j,i})\}_{i\in[k]}\Big] \;=\; \Gamma_{h,k,t}(p)$$

$$:= \sum_{\gamma=0}^{p} \Big(1 - \big(1 - \tfrac{1}{t}\big)^{\gamma}\Big)^{k} \binom{p}{\gamma} \Big(1 - \tfrac{1}{2^h}\Big)^{p-\gamma} \frac{1}{2^{h\gamma}}\,.$$

The section gives two classical bounds and three quantum ones. Section F.1 states the classical bound of a separate study of ITSR [Kud26a], which avoids the reprogramming step; Section F.2 re-derives the bound of [YTA23] with the corrected count, classically and quantumly, and with the lemma of [LGXZ26] in place of that of [GHHM21]. The concrete bounds use Theorem F.1 classically and Theorem F.3 quantumly; the classical line of Theorem F.2 is the comparison point of Remark 6.1.

F.1 A Classical Bound without Reprogramming

The proof of Theorem F.2 first reprograms H at the p challenges and then reduces to average-case search. The first step costs $p(q+p+1)/|\mathcal{K}|$ classically: every one of the q queries of the adversary is charged against every one of the p challenges, although a query can only be detected by the challenge whose key it happens to hit. The following bound, from a separate study of

ITSR [Kud26a], avoids the reprogramming step. It charges a query at a future challenge point (a *pull*) by the probability that the pulled digest helps to cover a forgery, not by the probability that the pull happens. We state it for a random oracle whose outputs are digests; Remark F.1 transfers it to Definition 4.11, where the adversary queries the m-bit function H and $\mathrm{MAP}_{h,k,t}$ is applied afterwards.

Let $D := [0, 2^h - 1] \times [0, t-1]^k$ be the set of digests, identified with position sets as in Definition 4.11, let $\theta := 2^{-h}$, and for $j \in [0, k]$ let

$$\Gamma_j(p) := \sum_{g=0}^{p} \binom{p}{g} \theta^g (1-\theta)^{p-g} \big(1 - (1-1/t)^g\big)^j ,$$

so that $\Gamma_k(p) = \Gamma_{h,k,t}(p)$ of Lemma F.1 and $\Gamma_0(p) = 1$. We write $(p)_m = p(p-1)\cdots(p-m+1)$ for the falling factorial and $\left\{ {v \atop m} \right\}$ for the Stirling numbers of the second kind.

Theorem F.1 ([Kud26a, Theorem 2])**.** *Let $\mathrm{H}' \colon \mathcal{K} \times \{0,1\}^\alpha \to D$ be a uniformly random function. Let $\mathcal{A}$ be a classical adversary, computationally unbounded, that makes at most q classical queries to H' and at most p challenge queries, adaptively interleaved; on a challenge query M the challenger samples $K \leftarrow_{\$} \mathcal{K}$ independently of everything else and returns $(K, \mathrm{H}'(K, M))$, and repeated messages and repeated challenge pairs are allowed. $\mathcal{A}$ wins if it outputs a pair (K^*, M^*) that is not a challenge pair and whose digest $\mathrm{H}'(K^*, M^*)$ is covered by the digests of the challenge pairs, i.e., for every $i \in [k]$ some challenge digest has the same tree index and the same i-th leaf index. Then*

$$\Pr[\mathcal{A} \text{ wins}] \;\le\; (q+1)\,\Gamma_k(p) + P_{h,k,t}(q,p)\,,$$

where the pull part *is*

$$P_{h,k,t}(q,p) := \sum_{m=1}^{\min(k,q,p)} (q+1) \binom{q}{m} (p)_m \Big(\frac{\theta}{|\mathcal{K}|}\Big)^m \sum_{v=m}^{k} \binom{k}{v} m! \left\{ {v \atop m} \right\} t^{-v}\, \Gamma_{k-v}(p)\,.$$

The proof is in [Kud26a]. The search part $(q+1)\Gamma_k(p)$ is the mapping part of Theorem F.2. The pull part vanishes at $q = 0$. Let T_m be its summand with m pulls and A_m the inner sum over v in it. The summand T_m charges m distinct queries of the adversary that are later hit by challenges, each with probability $p/|\mathcal{K}|$ per query and each landing in the instance of the forgery with probability θ, times the probability A_m that the pulled digests complete a covered forgery. The expected number of pulls is thus $\rho := q\,p\,\theta/|\mathcal{K}|$, and each further pull multiplies the summand by a constant times ρ. Let $c_m := A_{m+1}/((m+1)A_m)$, which depends on (h,k,t,p) but not on q, and let $c := \max_{1 \le m \le k-1} c_m$. Then

$$\frac{T_{m+1}}{T_m} = \frac{q-m}{m+1} \cdot \frac{(p-m)\,\theta}{|\mathcal{K}|} \cdot \frac{A_{m+1}}{A_m} \;\le\; c_m\,\rho \;\le\; c\,\rho\,,$$

$$\text{so} \qquad T_m \le (c\rho)^{m-1}\, T_1 \qquad \text{and} \qquad \sum_{m \ge 2} T_m \;\le\; T_1\, \frac{c\rho}{1-c\rho}$$

whenever $c\rho < 1$, for every parameter set with $k \ge 2$. The maximum defining c is over finitely many values because the pull count m never exceeds k.

Remark F.1 (Transfer to Definition 4.11)**.** In Definition 4.11 the adversary queries the m-bit function $\mathrm{H} \colon \mathcal{K} \times \{0,1\}^\alpha \to \{0,1\}^m$ and sees its full output, whereas the adversary of Theorem F.1 sees only the digest $\mathrm{H}'(K, M) \in D$. The gap is closed when H is modeled as a random oracle and every digest $u \in D$ has the same number $|\mathrm{MAP}^{-1}_{h,k,t}(u)| = 2^m/|D|$ of preimages under $\mathrm{MAP}_{h,k,t}$. Let H' be a uniformly random function to D, and let the value y_x of H at every point x be sampled uniformly from the preimage set $\mathrm{MAP}^{-1}_{h,k,t}(\mathrm{H}'(x))$, independently for all points. Then $\Pr[y_x = s] = (1/|D|) \cdot (|D|/2^m) = 2^{-m}$ for every $s \in \{0,1\}^m$, so H is a uniformly random function to $\{0,1\}^m$, and $\mathrm{MAP}_{h,k,t} \circ \mathrm{H} = \mathrm{H}'$. Hence Theorem F.1 gives

$$\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},p}(\mathcal{A}) \;\le\; (q+1)\,\Gamma_{h,k,t}(p) + P_{h,k,t}(q,p)\,.$$

The mapping of SHRINCS satisfies the hypothesis of Remark F.1.

Remark F.2 (The pull part at the stateless parameters)**.** At the stateless parameters $(h, k, t) = (45, 10, 2^{13})$ of Table 1, with $p = 2^{40}$ challenge queries and $|\mathcal{K}| = 2^{128}$ (Section 6.3), $p\theta/|\mathcal{K}| = 2^{-133}$; the nine values $c_1, \ldots, c_9$ decrease from $c_1 = 2^{4.63}$ to $c_9 = 2^{-5.50}$, with $c_5, \ldots, c_9$ below 1. Hence the maximum is attained at $m = 1$, $c = c_1 = 2^{4.63}$, and $c\rho = q/2^{128.37}$. The summand $m = 1$ is at most $q(q+1)/2^{256.98}$.

F.2 Bounds via Reprogramming

Theorem F.2 (Generic security of ITSR)**.** *Let* $\mathrm{H}\colon \mathcal{K} \times \{0,1\}^{\alpha} \to \{0,1\}^{m}$ *be modeled as a random oracle and let* $\mathrm{MAP}_{h,k,t}$ *be as above. Let* $\mathcal{A}$ *be an adversary in the experiment of Definition 4.11 that makes at most* p *challenge queries and* q *queries to* H*, and write* $\Gamma = \Gamma_{h,k,t}(p)$*. Then*

$$\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},p}(\mathcal{A}) \le \frac{p\,(q+p+1)}{|\mathcal{K}|} + (q+1)\,\Gamma \qquad \text{if } \mathcal{A} \text{ is classical, and}$$

$$\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},p}(\mathcal{A}) \le \frac{3p}{2}\sqrt{\frac{q+p+1}{|\mathcal{K}|}} + 8\,(q+1)^2\,\Gamma \qquad \text{if } \mathcal{A} \text{ is quantum with quantum access to } \mathrm{H}.$$

Proof. We consider the following two games.

Game 0: The experiment $\mathbf{ITSR}_{\mathrm{H},p}(\mathcal{A})$ of Definition 4.11 with H a random oracle.

Game 1: As Game 0, except that the challenge oracle, on the j-th query M_j, samples $K_j \leftarrow_{\$} \mathcal{K}$ and $y_j \leftarrow_{\$} \{0,1\}^m$, reprograms $\mathrm{H}(K_j, M_j) := y_j$, and returns $(K_j, \mathrm{MAP}_{h,k,t}(y_j))$. The final check uses the reprogrammed H.

From Game 0 to Game 1. This is the first step of [YTA23, Theorem 4]. The algorithm $\mathcal{B}_2$ for the adaptive-reprogramming experiment of Definition C.1, with $X_1 = \mathcal{K}$, $X_2 = \{0,1\}^{\alpha}$, and $Y = \{0,1\}^m$, runs $\mathcal{A}$ and forwards its queries to H to its own oracle $\mathcal{O}_b$. It answers the j-th challenge query M_j with $(K_j, \mathrm{MAP}_{h,k,t}(y_j))$, where $K_j \leftarrow \mathsf{Reprogram}(M_j)$ and $y_j \leftarrow \mathcal{O}_b(K_j, M_j)$ is a classical query, and outputs 1 if and only if the output (K^*, M^*) of $\mathcal{A}$ wins, which it checks with one more query to $\mathcal{O}_b$. In $\mathbf{Repro_0}$ the oracle is never modified and K_j is a fresh uniform key, so $\mathcal{B}_2$ simulates Game 0; in $\mathbf{Repro_1}$ the value at (K_j, M_j) has just been set to a uniform y_j, so $\mathcal{B}_2$ simulates Game 1. In both cases $\mathcal{B}_2$ outputs 1 exactly when $\mathcal{A}$ wins, with at most p calls to $\mathsf{Reprogram}$ and $q+p+1$ queries to $\mathcal{O}_b$. By Lemmas C.1 and C.2,

$$\big|\Pr[\text{Game } 0 \Rightarrow 1] - \Pr[\text{Game } 1 \Rightarrow 1]\big| \;\le\; \mathsf{Adv}^{\mathsf{Repro}}_{p,\,q+p+1}(\mathcal{B}_2) \;\le\; \begin{cases} \dfrac{p\,(q+p+1)}{|\mathcal{K}|} & \text{classically,} \\[2ex] \dfrac{3p}{2}\sqrt{\dfrac{q+p+1}{|\mathcal{K}|}} & \text{quantumly.} \end{cases} \tag{12}$$

Bounding Game 1. In Game 1 the keys $K_1, \ldots, K_p$ and the digests $y_1, \ldots, y_p$ are sampled independently of $\mathcal{A}$ and of H, so they can be sampled before $\mathcal{A}$ is run; we call $\mathbf{y} = (y_1, \ldots, y_p)$ the targets. If $\mathcal{A}$ makes fewer than p challenge queries, the unused entries are discarded. For a target vector $\mathbf{y}$ define the *covered set*

$$C(\mathbf{y}) \;:=\; \Big\{ y' \in \{0,1\}^m \;:\; \mathrm{MAP}_{h,k,t}(y') \subseteq \bigcup_{j \in [p]} \mathrm{MAP}_{h,k,t}(y_j) \Big\} \qquad \text{and} \qquad \lambda_{\mathbf{y}} \;:=\; \frac{|C(\mathbf{y})|}{2^m}\,.$$

If $\mathcal{A}$ wins Game 1 then its output (K^*, M^*) is not in Q and $\mathrm{H}(K^*, M^*) \in C(\mathbf{y})$: the winning condition covers the forged positions by the answered challenges, which are a subset of all p targets.

Fix $\mathbf{y}$. We construct a reduction $\mathcal{B}_1$ that solves $\mathsf{Avg\text{-}Search}_{\lambda_{\mathbf{y}}}$ (Definition C.3) over the set $\mathcal{S} = \mathcal{K} \times \{0,1\}^{\alpha}$ with q queries. $\mathcal{B}_1$ has oracle access to $f \leftarrow D_{\lambda_{\mathbf{y}}}$. It samples the keys $K_1, \ldots, K_p \leftarrow_{\$} \mathcal{K}$ and two random functions, $I\colon \mathcal{S} \to C(\mathbf{y})$ and $g\colon \mathcal{S} \to \{0,1\}^m \setminus C(\mathbf{y})$, each uniform on its range. Let T be the set of pairs (K_j, M_j) of the challenge queries answered so far, initially empty. $\mathcal{B}_1$ provides $\mathcal{A}$ with the oracle

$$\widetilde{\mathrm{H}}(K,M) \;=\; \begin{cases} y_j & \text{if } (K,M) = (K_j,M_j) \in T, \\ I(K,M) & \text{if } f(K,M) = 1, \\ g(K,M) & \text{otherwise,} \end{cases}$$

evaluating the case distinction, I, and g itself and querying f once per evaluation; for a quantum $\mathcal{A}$ the evaluation is performed in superposition, with one quantum query to f per query of $\mathcal{A}$ (Remark C.2). On the j-th challenge query M_j, $\mathcal{B}_1$ inserts (K_j, M_j) into T and returns $(K_j, \mathrm{MAP}_{h,k,t}(y_j))$; this involves no query to f. When $\mathcal{A}$ outputs (K^*, M^*), $\mathcal{B}_1$ outputs the same pair.

The view of $\mathcal{A}$ is distributed as in Game 1 with targets $\mathbf{y}$. At a point outside T the value of $\widetilde{\mathrm{H}}$ is uniform over $C(\mathbf{y})$ with probability $\lambda_{\mathbf{y}}$ and uniform over its complement otherwise, independently across points, hence uniform over $\{0,1\}^m$; at a point of T it equals the uniform target y_j, installed exactly when Game 1 reprograms H at that point. If $\mathcal{A}$ wins, then $(K^*, M^*) \notin T$ and $\widetilde{\mathrm{H}}(K^*, M^*) \in C(\mathbf{y})$; since g never outputs an element of $C(\mathbf{y})$, this requires $f(K^*, M^*) = 1$, so $\mathcal{B}_1$ succeeds. $\mathcal{B}_1$ makes q queries to f, and Lemma C.6 gives

$$\Pr[\text{Game } 1 \Rightarrow 1 \mid \mathbf{y}] \;\le\; \mathrm{Succ}^{\mathsf{Avg\text{-}Search}_{\lambda_{\mathbf{y}}}}(\mathcal{B}_1) \;\le\; \begin{cases} (q+1)\,\lambda_{\mathbf{y}} & \text{classically,} \\ 8\,(q+1)^2\,\lambda_{\mathbf{y}} & \text{quantumly.} \end{cases}$$

It remains to average over the targets. For uniform $y' \leftarrow_{\$} \{0,1\}^m$ independent of $\mathbf{y}$,

$$\mathbb{E}_{\mathbf{y}}[\lambda_{\mathbf{y}}] \;=\; \Pr_{\mathbf{y},\,y'}\big[y' \in C(\mathbf{y})\big] \;=\; \Gamma_{h,k,t}(p)$$

by Lemma F.1, since $\mathrm{MAP}_{h,k,t}$ maps the independent uniform digests $y', y_1, \ldots, y_p$ to independent uniform position sets. The prefactors $(q+1)$ and $8(q+1)^2$ do not depend on $\mathbf{y}$, so

$$\Pr[\text{Game } 1 \Rightarrow 1] \;\le\; \begin{cases} (q+1)\,\Gamma & \text{classically,} \\ 8\,(q+1)^2\,\Gamma & \text{quantumly.} \end{cases} \tag{13}$$

Combining Eqs. (12) and (13) completes the proof. □

With Lemma C.3 in place of Lemma C.1, the first step moves the challenge count under the square root with a finer query count, and Corollary C.1 removes the root at the price of a factor 2 on the search term.

Theorem F.3 (Generic quantum security of ITSR, tighter reprogramming)**.** *In the setting of Theorem F.2, let $\mathcal{A}$ be a halting (Remark C.1) quantum adversary with quantum access to* H*, and write $S := p\,q + p(p-1)/2 + 1$. Then*

$$\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},p}(\mathcal{A}) \le 4\sqrt{\frac{S}{|\mathcal{K}|}} + 8\,(q+1)^2\,\Gamma\,, \tag{14}$$

$$\mathrm{Succ}^{\mathrm{ITSR}}_{\mathrm{H},p}(\mathcal{A}) \le 32\,\frac{S}{|\mathcal{K}|} + 16\,(q+1)^2\,\Gamma\,. \tag{15}$$

Proof. Consider Game 0 and Game 1 of the proof of Theorem F.2 and the algorithm $\mathcal{B}_2$ constructed there for Eq. (12). $\mathcal{B}_2$ issues one call to Reprogram per challenge query, hence $a \le p$ calls, where a is the number of challenge queries of $\mathcal{A}$ and depends on the execution. Let $\mathcal{B}_2'$

be $\mathcal{B}_2$ padded as in Remark C.4: when $\mathcal{A}$ outputs (K^*, M^*), $\mathcal{B}_2'$ issues $p-a$ further calls to Reprogram on messages that are pairwise distinct and distinct from M^* and from all challenge messages, reads nothing after them, and only then issues its verification query at (K^*, M^*). Every execution of $\mathcal{B}_2'$ issues exactly p calls, all before the verification query, and no completing call of Lemma C.3. Before its j-th call, $\mathcal{B}_2'$ has forwarded at most q queries of $\mathcal{A}$ to $\mathcal{O}_b$ and has issued at most $j-1$ classical queries of its own, one to read each earlier programmed digest, whether the j-th call answers a challenge query or pads. Hence $\hat{q}_j \le q + j - 1$ in the notation of Lemma C.3, in every execution of either game, and

$$\sum_{j=1}^{p} \hat{q}_j \;\le\; \sum_{j=1}^{p} (q + j - 1) \;=\; p\,q + \frac{p(p-1)}{2} \;<\; S$$

by Remark C.4. As in the proof of Theorem F.2, $\mathcal{B}_2$ outputs 1 in $\mathbf{Repro_0}$ exactly when Game 0 outputs 1, and in $\mathbf{Repro_1}$ exactly when Game 1 outputs 1. The padding calls come after the output of $\mathcal{A}$ and reprogram only points (x_1, M') with $M' \neq M^*$, whose values are never read, so they change neither the view of $\mathcal{A}$ nor the value $\mathcal{O}_b(K^*, M^*)$. Hence $\mathcal{B}_2'$ also outputs 1 in $\mathbf{Repro_b}$ exactly when Game b outputs 1, for $b \in \{0,1\}$, and we apply the following two bounds to $\mathcal{B}_2'$. Lemma C.3 with $e = 1$ gives $\Pr[\text{Game } 0 \Rightarrow 1] \le \Pr[\text{Game } 1 \Rightarrow 1] + 4\sqrt{S/|\mathcal{K}|}$, and Corollary C.1 gives $\Pr[\text{Game } 0 \Rightarrow 1] \le 2\Pr[\text{Game } 1 \Rightarrow 1] + 32\,S/|\mathcal{K}|$. In both cases Eq. (13) bounds $\Pr[\text{Game } 1 \Rightarrow 1]$ by $8(q+1)^2\Gamma$; the second step of the proof does not depend on how the first step was bounded. □

G Generic Bounds for Multi-Target Collision Resistance with Random Sampling

The stateful path relies on the multi-target target collision resistance with random sampling of the message digest $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ (Definition 4.9, Theorem 4.2). The proof of the generic bound of [DKKW25, Theorem 5] replaces the challenge oracle by one that samples its answers and reprograms the tweakable hash function, and then bounds the reprogrammed game by an average-case search. The randomizer part of that bound, $\frac{3pK}{2}\sqrt{q'/|\mathcal{R}|}$, is the reprogramming step bounded with Lemma C.1. We repeat the argument here so that the step can be bounded with Lemma C.3 and Corollary C.1, and so that the search step can be counted the way Theorem F.2 counts it[17]. For classical adversaries the reprogramming step can be counted call by call, which gives Corollary G.1 at the end of the section; it is the bound used for the classical SM-rTCR term in Section 6.5.

Throughout, $\mathbf{Th}\colon \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R}) \to \mathcal{N}$ is modeled as a random oracle, $\mathsf{Prop}\colon \mathcal{N} \to \{0,1\}$ is any predicate, $K \in \mathbb{N}$ is the sampling budget, and the adversary $\mathcal{A}$ makes at most $p \le |\mathcal{T}|$ challenge queries and q queries to $\mathbf{Th}$. We assume $|\mathcal{N}| \ge 2$, the bounds being trivial otherwise. Whenever a point is reprogrammed more than once, the latest value counts. An entry $(T_j, \mathsf{m}_j, \perp)$ of the list Q records a challenge query whose K attempts all failed; the winning condition of Definition 4.9 evaluates $\mathbf{Th}$ at $(\mathsf{m}_j \| \rho_j)$ and is read as false for such an entry; likewise the output ρ^* is read as an element of $\mathcal{R}$.

Theorem G.1 (Generic quantum security of SM-rTCR)**.** *Let $\mathcal{A}$ be a quantum adversary with quantum access to* $\mathbf{Th}$ *in the experiment of Definition 4.9 that makes at most p challenge queries*

[17] The theorem also states the bound with the reprogramming step bounded by Lemma C.1, as in [DKKW25, Theorem 5]. It is not used in the concrete bounds; it is the earlier bound with the counts of this section, and it serves as the reference against which the gain from Lemma C.3 is measured (Remarks 6.1 and G.1).

and q queries to $\mathbf{Th}$*, and let* $q' := q + pK$ *and* $S' := pK \cdot q + pK(pK-1)/2 + 1$*. Then*

$$\mathrm{Succ}^{\mathsf{SM\text{-}rTCR},K}_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A}) \le \frac{8(q+1)^2}{|\mathcal{N}|} + \frac{3pK}{2}\sqrt{\frac{q'+1}{|\mathcal{R}|}}, \tag{16}$$

$$\mathrm{Succ}^{\mathsf{SM\text{-}rTCR},K}_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A}) \le \frac{8(q+1)^2}{|\mathcal{N}|} + 4\sqrt{\frac{S'}{|\mathcal{R}|}}, \tag{17}$$

$$\mathrm{Succ}^{\mathsf{SM\text{-}rTCR},K}_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A}) \le \frac{16(q+1)^2}{|\mathcal{N}|} + 32\,\frac{S'}{|\mathcal{R}|}. \tag{18}$$

The three bounds take $\mathcal{N} = \{0,1\}^\nu$ *with the XOR oracle (Remark C.2). The bounds Eqs. (17) and (18) are stated for a halting* $\mathcal{A}$ *(Remark C.1).*

The proof follows [DKKW25, Appendix C]. The first step replaces the challenge oracle by one that samples the seed and the digest of every attempt and reprograms $\mathbf{Th}$ accordingly; the second step rewrites the random oracle around a hidden boolean function whose marked inputs are exactly the inputs that collide with a target, so that a collision is a marked input.

Proof. We consider the following games; $\Pr[\text{Game } i \Rightarrow 1]$ denotes the probability that Game i outputs 1.

Game 0: The experiment $\mathbf{SM\text{-}rTCR}^K_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A})$ of Definition 4.9 with $\mathbf{Th}$ a random oracle.

Game 1: As Game 0, except that every attempt of the challenge oracle samples $\rho \leftarrow_\$ \mathcal{R}$ and $y \leftarrow_\$ \mathcal{N}$, reprograms $\mathbf{Th}(P,T,(\mathsf{m}\|\rho)) := y$, and proceeds with y in place of the evaluation of $\mathbf{Th}$. The winning condition uses the reprogrammed $\mathbf{Th}$.

From Game 0 to Game 1. We construct an algorithm $\mathcal{C}$ for the experiment of Definition C.1 with $X_1 = \mathcal{R}$, $X_2 = \mathcal{P}\times\mathcal{T}\times\mathcal{M}$, and $Y = \mathcal{N}$. $\mathcal{C}$ samples $P \leftarrow_\$ \mathcal{P}$, runs $\mathcal{A}$ on P, and forwards the queries of $\mathcal{A}$ to $\mathbf{Th}$ to its oracle $\mathcal{O}_b$. It answers a challenge query (T,m) by the procedure of Definition 4.9, in which every attempt is replaced by the call $\rho \leftarrow \mathsf{Reprogram}(P,T,\mathsf{m})$ followed by the classical query $y \leftarrow \mathcal{O}_b(\rho,(P,T,\mathsf{m}))$. When $\mathcal{A}$ outputs (j,m^*,ρ^*), $\mathcal{C}$ queries $\mathcal{O}_b$ at $(\rho^*,(P,T_j,\mathsf{m}^*))$ once, compares the answer with the digest y_j recorded for the j-th entry, and outputs 1 if and only if the winning condition holds. With $b=0$ the seed returned by $\mathsf{Reprogram}$ is uniform and $\mathcal{O}_0$ is untouched, so every attempt evaluates the random oracle at a uniform seed, as in Game 0; with $b=1$ the oracle is reprogrammed to a fresh uniform digest at every attempt, as in Game 1. Hence $\mathcal{C}$ outputs 1 in $\mathbf{Repro_b}$ exactly when Game b outputs 1.

$\mathcal{C}$ issues one call to $\mathsf{Reprogram}$ per attempt, hence at most pK calls, and at most $q+pK+1 = q'+1$ queries to $\mathcal{O}_b$: the forwarded queries of $\mathcal{A}$, one read-back per attempt, and the final evaluation. Before its j-th call to $\mathsf{Reprogram}$ it has issued at most q forwarded queries and exactly $j-1$ read-backs (the read-back of an attempt precedes the next attempt, whose seed depends on it), so $\hat{q}_j \le q+j-1$ for the genuine calls, in every execution of either game. The number $a \le pK$ of attempts depends on the execution, also for $K=1$, since $\mathcal{A}$ makes at most p challenge queries. For Lemma C.3 we use the variant $\mathcal{C}'$ of $\mathcal{C}$ padded as in Remark C.4: when $\mathcal{A}$ outputs (j,m^*,ρ^*), $\mathcal{C}'$ issues $pK-a$ further calls to $\mathsf{Reprogram}$ on inputs (P,T,m) that are pairwise distinct, distinct from the inputs of all attempts, and distinct from (P,T_j,m^*), reads nothing after them, and only then issues its final evaluation. By Remark C.4, $\mathcal{C}'$ outputs 1 in $\mathbf{Repro_b}$ exactly when Game b outputs 1, as $\mathcal{C}$ does, it issues exactly pK calls, all before its final evaluation, and $\sum_{j\le pK}\hat{q}_j \le pKq + pK(pK-1)/2 < S'$. Lemma C.1 is applied to $\mathcal{C}$, whose query count it takes in total. Therefore

$$\big|\Pr[\text{Game } 0 \Rightarrow 1] - \Pr[\text{Game } 1 \Rightarrow 1]\big| \;\le\; \min\Big\{\frac{3pK}{2}\sqrt{\frac{q'+1}{|\mathcal{R}|}},\; 4\sqrt{\frac{S'}{|\mathcal{R}|}}\Big\} \tag{19}$$

by Lemmas C.1 and C.3, applied to $\mathcal{C}$ and to $\mathcal{C}'$, respectively, and

$$\Pr[\text{Game } 0 \Rightarrow 1] \;\le\; 2\Pr[\text{Game } 1 \Rightarrow 1] + 32\,\frac{S'}{|\mathcal{R}|} \tag{20}$$

by Corollary C.1, applied to $\mathcal{C}'$.

Bounding Game 1. In Game 1 the seeds and digests of all attempts are independent of $\mathcal{A}$ and of $\mathbf{Th}$, and by the guard of Definition 4.9 each tweak is the subject of at most one challenge query. They can therefore be sampled per tweak before $\mathcal{A}$ is run: for every $T \in \mathcal{T}$, a list L_T of at most K pairs $(\rho_{T,i}, y_{T,i}) \leftarrow_{\$} \mathcal{R} \times \mathcal{N}$, ending with the first pair with $\mathsf{Prop}(y_{T,i}) = 1$, called (ρ^*_T, y^*_T); if there is none, $y^*_T := \bot$. A challenge query (T, m) reprograms $\mathbf{Th}(P, T, (\mathsf{m}\|\rho_{T,i})) := y_{T,i}$ for every pair of L_T and returns (y^*_T, ρ^*_T), respectively $(\bot, \bot)$. Since each list serves at most one challenge query, this is Game 1.

We construct a reduction $\mathcal{B}$ that solves $\mathsf{Avg\text{-}Search}_\lambda$ (Definition C.3) over $\mathcal{S} = \mathcal{T} \times \mathcal{M} \times \mathcal{R}$ with $\lambda = 1/|\mathcal{N}|$ and q queries to $f \leftarrow D_\lambda$. $\mathcal{B}$ samples $P' \leftarrow_{\$} \mathcal{P}$, the lists L_T, a random tweakable hash function $\mathbf{Th}'$ for the parameters other than P', and for every T a random function $g_T \colon \mathcal{M} \times \mathcal{R} \to \mathcal{N} \setminus \{y^*_T\}$ (into $\mathcal{N}$ if $y^*_T = \bot$). Let D be the set of inputs $(T, \mathsf{m}, \rho_{T,i})$ programmed by the challenge queries answered so far, each with its value $y_{T,i}$; if two pairs of L_T share a seed, the later one counts, as the latest reprogramming does in Game 1. $\mathcal{B}$ answers the queries of $\mathcal{A}$ to $\mathbf{Th}$ with

$$\widetilde{\mathbf{Th}}(P, T, (\mathsf{m}\|\rho)) \;=\; \begin{cases} \mathbf{Th}'(P, T, (\mathsf{m}\|\rho)) & \text{if } P \neq P', \\ y_{T,i} & \text{if } P = P' \text{ and } (T, \mathsf{m}, \rho) = (T, \mathsf{m}, \rho_{T,i}) \in D, \\ y^*_T & \text{if } P = P',\ (T, \mathsf{m}, \rho) \notin D,\ y^*_T \neq \bot, \text{ and } f(T, \mathsf{m}, \rho) = 1, \\ g_T(\mathsf{m}, \rho) & \text{otherwise,} \end{cases}$$

evaluating the case distinction, $\mathbf{Th}'$, and g_T itself and querying f once per evaluation; for a quantum $\mathcal{A}$ the evaluation is performed in superposition, with one quantum query to f per query of $\mathcal{A}$ (Remark C.2). A challenge query (T, m) is answered as above, with the guard of Definition 4.9 and the list Q handled as in the experiment and the pairs of L_T inserted into D for m, without querying f. When $\mathcal{A}$ outputs $(j, \mathsf{m}^*, \rho^*)$, $\mathcal{B}$ outputs $(T_j, \mathsf{m}^*, \rho^*)$.

The view of $\mathcal{A}$ is distributed as in Game 1: at a point of D the answer is the digest programmed there, as in Game 1; at a point $(P', T, \mathsf{m}, \rho) \notin D$ with $y^*_T \neq \bot$ it is y^*_T with probability $1/|\mathcal{N}|$ and uniform over $\mathcal{N} \setminus \{y^*_T\}$ otherwise, independently across points, hence uniform; at every other point it is uniform by construction. Suppose $\mathcal{A}$ wins. Then $\rho_j = \rho^*_{T_j} \neq \bot$, the target is $\widetilde{\mathbf{Th}}(P', T_j, (\mathsf{m}_j\|\rho_j)) = y^*_{T_j}$, $(\mathsf{m}^*, \rho^*) \neq (\mathsf{m}_j, \rho_j)$, and $\widetilde{\mathbf{Th}}(P', T_j, (\mathsf{m}^*\|\rho^*)) = y^*_{T_j}$. At tweak T_j the only point of D with value $y^*_{T_j}$ is $(T_j, \mathsf{m}_j, \rho_j)$: by the guard, the points of D at T_j are $(T_j, \mathsf{m}_j, \rho_{T_j,i})$; the accepted pair is the last in L_{T_j}, so a point with a seed other than ρ_j holds the value y of a rejected pair, and $\mathsf{Prop}(y) = 0 \neq \mathsf{Prop}(y^*_{T_j})$. Hence $(T_j, \mathsf{m}^*, \rho^*) \notin D$, and since g_{T_j} never takes the value $y^*_{T_j}$, $f(T_j, \mathsf{m}^*, \rho^*) = 1$. Thus $\mathcal{B}$ succeeds whenever $\mathcal{A}$ wins, and by Lemma C.6,

$$\Pr[\text{Game } 1 \Rightarrow 1] \;\le\; \mathrm{Succ}^{\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}}(\mathcal{B}) \;\le\; 8\,(q+1)^2/|\mathcal{N}|\,. \tag{21}$$

Combining Eqs. (19) and (21) gives Eqs. (16) and (17); combining Eqs. (20) and (21) gives Eq. (18). □

Classical adversaries. For a classical adversary the reprogramming step can be counted call by call. A reprogramming is visible only at a point that was queried before, and the unique-tweak guard of Definition 4.9 separates the challenges from one another, so a query of the adversary can serve against the attempts of one challenge only. What remains of the challenge count is the coincidence of two seeds within one challenge.

Corollary G.1 (Generic classical security of SM-rTCR)**.** *Let $\mathcal{A}$ be a classical adversary in the experiment of Definition 4.9, with* **Th** *modeled as a random oracle, that makes at most p challenge queries and q queries to* **Th**. *Then*

$$\mathrm{Succ}^{\mathsf{SM\text{-}rTCR},K}_{\mathbf{Th},p,\mathsf{Prop}}(\mathcal{A}) \;\le\; \frac{q+1}{|\mathcal{N}|} + \frac{Kq + pK(K-1)/2}{|\mathcal{R}|}\,,$$

and for $K = 1$ the right-hand side is $(q+1)/|\mathcal{N}| + q/|\mathcal{R}|$.

Proof. We use the games and the algorithms $\mathcal{C}$ and $\mathcal{B}$ of the proof of Theorem G.1. For Game 1, the reduction $\mathcal{B}$ is classical when $\mathcal{A}$ is, and the classical case of Lemma C.6 gives $\Pr[\text{Game } 1 \Rightarrow 1] \le (q+1)/|\mathcal{N}|$ in place of Eq. (21). For the step from Game 0 to Game 1 we apply Lemma C.4 to $\mathcal{C}$. Consider the k-th attempt of the challenge query (T_c, m_c), which is a call to Reprogram with input (P, T_c, m_c). The queries that $\mathcal{C}$ has issued before this call at the second input (P, T_c, m_c) are the at most a_c queries of $\mathcal{A}$ at this input and the $k-1$ read-backs of the earlier attempts of the same challenge: the read-backs of other challenges carry other tweaks by the guard of Definition 4.9, and the final evaluation of $\mathcal{C}$ comes after its last call. Hence $n_j \le a_c + (k-1)$ for this call, and with $\sum_c a_c \le q$ over the at most p challenges,

$$\sum_j n_j \;\le\; \sum_c \sum_{k=1}^{K} \big(a_c + (k-1)\big) \;\le\; Kq + pK(K-1)/2\,.$$

For $K = 1$ every call of $\mathcal{C}$ carries a different second input, so $\mathcal{C}$ is an algorithm for the experiment of Definition C.2 whose guard never fires, and the bound is Corollary C.2 with Remark C.6. □

Remark G.1 (Comparison with the bound of [DKKW25])**.** The bound of [DKKW25, Theorem 5] reads $8(q'+1)^2/|\mathcal{N}| + \frac{3pK}{2}\sqrt{q'/|\mathcal{R}|}$ with $q' = q + pK$. Its search prefactor counts the reduction's own read-back queries, which the hard-coded lists L_T make unnecessary, in the same way that the challenge count is removed from the prefactor of Γ in Theorem F.2. Its reprogramming term omits the final evaluation of the reduction, which Lemma C.1 counts; Eq. (16) includes it. For the stateful path of SHRINCS ($K = 1$, $\mathsf{Prop} \equiv 1$, $|\mathcal{N}| = 2^{2n}$) neither difference is visible in Section 6: the hash part is at the 2^{-256} scale, and the reprogramming term is replaced by the smaller forms of Eq. (17) and Eq. (18).

Algorithm 7: Reduction $\mathcal{B}_3$ against SM-UD of $\mathbf{F} \in \mathsf{Th}_\lambda$ for the hybrid transition $\gamma \to \gamma + 1$

Input : Transition index $\gamma \in \{0, \ldots, w-3\}$; oracle $\mathcal{O}(\cdot, b)$ of the SM-UD challenger (Definition 4.6) for $\mathbf{F} \in \mathsf{Th}_\lambda$, and collection oracle Th_λ (Section 4.3); EU-CMA forger $\mathcal{A}$ against FXMSS

Output : A bit b'

Run $\mathcal{A}$ to obtain the structure descriptor; enumerate the signing leaves as $j = 1, \ldots, L$ in a canonical order.

Sample the digests $y_1, \ldots, y_L \leftarrow_\$ \{0,1\}^{2n}$ and assign y_j to the j-th signing leaf `// matches F.2`

for $j \in [L]$ **do**

  Grind (d_j, c_j) as $\mathcal{O}^{\mathsf{Prop}}$ would, evaluating $\mathbf{H}_{\text{grind}}$ through the collection oracle Th_λ: c_j is the smallest counter such that $d_j = \mathbf{H}_{\text{grind}}(P, \mathbf{ADRS}^{\mathsf{g}}_j[0{:}10], y_j, c_j)$ satisfies Prop; parse d_j as $(a_{j,0}, \ldots, a_{j,l_{\mathsf{sf}}-1})$.

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ *with* $\gamma < a_{j,i} - 1$ **do**

  Query $\psi_{j,i} \leftarrow \mathcal{O}(\mathbf{ADRS}^{\gamma}_{j,i}, b)$, where $\mathbf{ADRS}^{\gamma}_{j,i}$ is $\mathbf{ADRS}^{\mathsf{h}}_j$ with chain address i and hash address γ `//` $b = 0$: $\psi_{j,i} = \mathbf{F}(P, \mathbf{ADRS}^{\gamma}_{j,i}, u)$ `with` $u \leftarrow_\$ \{0,1\}^n$`;` $b = 1$: $\psi_{j,i} \leftarrow_\$ \{0,1\}^n$

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ **do**

  if $a_{j,i} = 0$ **then**

    $\sigma_{j,i} \leftarrow_\$ \{0,1\}^n$

  else

    if $\gamma < a_{j,i} - 1$ **then**

      $\phi_{j,i} \leftarrow c^{\gamma+1,\, a_{j,i}-1-(\gamma+1)}(\psi_{j,i},\, i,\, \mathbf{ADRS}^{\mathsf{h}}_j)$, evaluated through Th_λ

    else

      $\phi_{j,i} \leftarrow_\$ \{0,1\}^n$

    end

    $\sigma_{j,i} \leftarrow c^{a_{j,i}-1,\, 1}(\phi_{j,i},\, i,\, \mathbf{ADRS}^{\mathsf{h}}_j)$, evaluated through Th_λ

  end

  $\mathsf{pk}_{j,i} \leftarrow c^{a_{j,i},\, w-1-a_{j,i}}(\sigma_{j,i},\, i,\, \mathbf{ADRS}^{\mathsf{h}}_j)$, evaluated through Th_λ

end

Compress each tuple $(\mathsf{pk}_{j,0}, \ldots, \mathsf{pk}_{j,l_{\mathsf{sf}}-1})$ with $\mathsf{Th}_{l_{\mathsf{sf}}}$ and build the internal nodes with $\mathbf{H}$, evaluating both through Th_λ, to obtain the root $\mathsf{PK.sf}$.

Run Key Generation for the stateless part of the SHRINCS scheme, evaluating every hash call through Th_λ, to obtain $\mathsf{PK.sl}$.

Signal the SM-UD challenger to proceed to the second stage and receive the public parameter P; set $\mathsf{PK.seed} \leftarrow P$.

Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ by simulating the random oracle.

for *each signing query* $(\widetilde{M}_j, (z^\star, j^\star))$*, with* j *the canonical index of the queried leaf* **do**

  Form the tweak $\mathsf{idx}_j = \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(j^\star, 8)$.

  Sample $R_j \leftarrow_\$ \mathcal{R}$ and reprogram $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_j, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M}_j) := y_j$ as in **F.2**.

  Assemble the signature from R_j, the counter c_j, the blocks $(\sigma_{j,0}, \ldots, \sigma_{j,l_{\mathsf{sf}}-1})$, and the authentication path, and reply with it.

end

Obtain the forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ from $\mathcal{A}$.

if σ^* *is valid,* $\widetilde{M}^* \neq \widetilde{M}_j$ *for all* $j \in [1; L]$*, and the abort conditions of* **F.1** *and* **F.3** *do not hold* **then**

  return $b' = 1$

else

  return $b' = 0$

end

Algorithm 8: Reduction $\mathcal{B}_4$ against SM-TCR of $\mathbf{F} \in \mathsf{Th}_\lambda$

Input : Challenge oracle $\mathcal{O}(\cdot,\cdot)$ of the SM-TCR challenger (Definition 4.4) for $\mathbf{F} \in \mathsf{Th}_\lambda$ and collection oracle Th_λ (Section 4.3); EU-CMA forger $\mathcal{A}$ against FXMSS

Output : An SM-TCR collision (t, m^*) or fail

Run $\mathcal{A}$ to obtain the structure descriptor; enumerate the signing leaves as $j = 1, \ldots, L$ in a canonical order.

Sample the digests $y_1, \ldots, y_L \leftarrow_\$ \{0,1\}^{2n}$ and assign y_j to the j-th signing leaf `// matches F.2`

for $j \in [L]$ **do**

  Grind (d_j, c_j) as $\mathcal{O}^{\mathsf{Prop}}$ would, evaluating $\mathbf{H}_{\text{grind}}$ through the collection oracle Th_λ: c_j is the smallest counter such that $d_j = \mathbf{H}_{\text{grind}}(P, \mathbf{ADRS}^{\mathsf{g}}_j[0{:}10], y_j, c_j)$ satisfies Prop; parse d_j as $(a_{j,0}, \ldots, a_{j,l_{\mathsf{sf}}-1})$.

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ **do**

  Set $\beta_{j,i} \leftarrow \max(a_{j,i} - 1, 0)$ and sample $v_{j,i}^{\beta_{j,i}} \leftarrow_\$ \{0,1\}^n$ `// the fresh uniform value of F.4`

  for $\pi = \beta_{j,i}, \ldots, w-2$ **do**

    Query $v_{j,i}^{\pi+1} \leftarrow \mathcal{O}(\mathbf{ADRS}^{\pi}_{j,i}, v^{\pi}_{j,i})$, where $\mathbf{ADRS}^{\pi}_{j,i}$ is $\mathbf{ADRS}^{\mathsf{h}}_j$ with chain address i and hash address π `//` $v_{j,i}^{\pi+1} = \mathbf{F}(P, \mathbf{ADRS}^{\pi}_{j,i}, v^{\pi}_{j,i})$

  end

  Set $\sigma_{j,i} \leftarrow v_{j,i}^{a_{j,i}}$ and $\mathsf{pk}_{j,i} \leftarrow v_{j,i}^{w-1}$.

end

Compress each tuple $(\mathsf{pk}_{j,0}, \ldots, \mathsf{pk}_{j,l_{\mathsf{sf}}-1})$ with $\mathsf{Th}_{l_{\mathsf{sf}}}$ and build the internal nodes with $\mathbf{H}$, evaluating both through the collection oracle Th_λ, to obtain the root $\mathsf{PK.sf}$.

Run Key Generation for the stateless part of the SHRINCS scheme, evaluating every hash call through Th_λ, to obtain $\mathsf{PK.sl}$.

Signal the SM-TCR challenger to proceed to the second stage and receive the public parameter P; set $\mathsf{PK.seed} \leftarrow P$.

Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}$ by simulating the random oracle.

for *each signing query* $(\widetilde{M}_j, (z^\star, j^\star))$*, with* j *the canonical index of the queried leaf* **do**

  Form the tweak $\mathsf{idx}_j = \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(j^\star, 8)$.

  Sample $R_j \leftarrow_\$ \mathcal{R}$ and reprogram $\mathbf{H}^{\mathsf{sf}}_{\mathbf{msg}}(R_j, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M}_j) := y_j$ as in **F.2**.

  Assemble the signature from R_j, the counter c_j, the blocks $(\sigma_{j,0}, \ldots, \sigma_{j,l_{\mathsf{sf}}-1})$, and the authentication path, and reply with it.

end

Obtain the forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ from $\mathcal{A}$ and recompute its verification chains from the blocks of σ^*.

if σ^* *is valid and the recomputation contains an* $\mathbf{F}$*-call at a challenge address* $\mathbf{ADRS}^{\pi}_{j,i}$ *with input* $\mathsf{m}^* \neq v^{\pi}_{j,i}$ *and* $\mathbf{F}(\mathsf{PK.seed}, \mathbf{ADRS}^{\pi}_{j,i}, \mathsf{m}^*) = v_{j,i}^{\pi+1}$ **then**

  return (t, m^*)*, where* t *is the index of the challenge query at* $\mathbf{ADRS}^{\pi}_{j,i}$

else

  return *fail*

end

Algorithm 9: Reduction $\mathcal{B}_5$ against SM-PRE of $\mathbf{F} \in \mathsf{Th}_\lambda$

Input : Challenge oracle $\mathcal{O}(\cdot)$ of the SM-PRE challenger (Definition 4.5) for $\mathbf{F} \in \mathsf{Th}_\lambda$ and collection oracle Th_λ (Section 4.3); EU-CMA forger $\mathcal{A}$ against FXMSS

Output : An SM-PRE preimage (t, m^*) or fail

Run $\mathcal{A}$ to obtain the structure descriptor; enumerate the signing leaves as $j = 1, \ldots, L$ in a canonical order.

Sample the digests $y_1, \ldots, y_L \leftarrow_\$ \{0,1\}^{2n}$ and assign y_j to the j-th signing leaf `// matches F.2`

for $j \in [L]$ **do**

  Grind (d_j, c_j) as $\mathcal{O}^{\mathsf{Prop}}$ would, evaluating $\mathbf{H}_{\text{grind}}$ through the collection oracle Th_λ; parse d_j as $(a_{j,0}, \ldots, a_{j,l_{\mathsf{sf}}-1})$.

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ **do**

  if $a_{j,i} > 0$ **then**

    Query $\sigma_{j,i} \leftarrow \mathcal{O}(\mathbf{ADRS}_{j,i}^{a_{j,i}-1})$, where $\mathbf{ADRS}_{j,i}^{a_{j,i}-1}$ is $\mathbf{ADRS}_j^{\mathsf{h}}$ with chain address i and hash address $a_{j,i} - 1$ `//` $\sigma_{j,i} = \mathbf{F}(P, \mathbf{ADRS}_{j,i}^{a_{j,i}-1}, u)$ `with hidden` $u \leftarrow_\$ \{0,1\}^n$`, the fresh uniform value of` $\mathbf{F}$`.4`

  else

    $\sigma_{j,i} \leftarrow_\$ \{0,1\}^n$

  end

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ **do**

  Complete the public-key element $\mathsf{pk}_{j,i}$ from $\sigma_{j,i}$ as in verification, evaluating each $\mathbf{F}$-call through the collection oracle Th_λ (at the hash addresses $a_{j,i}, \ldots, w-2$, disjoint from the challenge tweaks).

end

Compress each tuple $(\mathsf{pk}_{j,0}, \ldots, \mathsf{pk}_{j,l_{\mathsf{sf}}-1})$ with $\mathsf{Th}_{l_{\mathsf{sf}}}$ and build the internal nodes with $\mathbf{H}$ through Th_λ, to obtain the root $\mathsf{PK.sf}$.

Run Key Generation for the stateless part of the SHRINCS scheme, evaluating every hash call through Th_λ, to obtain $\mathsf{PK.sl}$.

Signal the SM-PRE challenger to proceed to the second stage and receive the public parameter P; set $\mathsf{PK.seed} \leftarrow P$.

Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ by simulating the random oracle.

for *each signing query* $(\widetilde{M}_j, (z^\star, j^\star))$*, with* j *the canonical index of the queried leaf* **do**

  Form the tweak $\mathsf{idx}_j = \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(j^\star, 8)$.

  Sample $R_j \leftarrow_\$ \mathcal{R}$ and reprogram $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}(R_j, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M}_j) := y_j$ as in $\mathbf{F}.\mathbf{2}$.

  Assemble the signature from R_j, the counter c_j, the blocks $(\sigma_{j,0}, \ldots, \sigma_{j,l_{\mathsf{sf}}-1})$, and the authentication path, and reply with it.

end

Obtain the forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ from $\mathcal{A}$ and recompute its verification chains from the blocks of σ^*.

if σ^* *is valid and the recomputation contains an* $\mathbf{F}$*-call at a challenge tweak* $\mathbf{ADRS}_{j,i}^{a_{j,i}-1}$ *with input* m^* *such that* $\mathbf{F}(\mathsf{PK.seed}, \mathbf{ADRS}_{j,i}^{a_{j,i}-1}, \mathsf{m}^*) = \sigma_{j,i}$ **then**

  return (t, m^*)*, where* t *is the index of the challenge query at* $\mathbf{ADRS}_{j,i}^{a_{j,i}-1}$

else

  return *fail*

end

Algorithm 10: Reduction $\mathcal{B}_6$ against SM-TCR of $\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda$

Input : Challenge oracle $\mathcal{O}(\cdot,\cdot)$ of the SM-TCR challenger (Definition 4.4) for $\mathsf{Th}_{l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda$ and collection oracle Th_λ (Section 4.3); EU-CMA forger $\mathcal{A}$ against FXMSS

Output : An SM-TCR collision (t, m^*) or fail

Run $\mathcal{A}$ to obtain the structure descriptor; enumerate the signing leaves as $j = 1, \ldots, L$ in a canonical order.

Sample the digests $y_1, \ldots, y_L \leftarrow_\$ \{0,1\}^{2n}$ and assign y_j to the j-th signing leaf `// matches F.2`

for $j \in [L]$ **do**

  Grind (d_j, c_j) as $\mathcal{O}^{\mathsf{Prop}}$ would, evaluating $\mathbf{H}_{\text{grind}}$ through the collection oracle Th_λ; parse d_j as $(a_{j,0}, \ldots, a_{j,l_{\mathsf{sf}}-1})$.

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ **do**

  Sample the fresh uniform value of **F.4**, $u_{j,i} \leftarrow_\$ \{0,1\}^n$, and compute the block $\sigma_{j,i}$ and the endpoint $\mathsf{pk}_{j,i}$ as **F.4** prescribes, evaluating each **F**-call through the collection oracle Th_λ.

end

for $j \in [L]$ **do**

  Query $\mathsf{pk}_j \leftarrow \mathcal{O}(\mathbf{ADRS}_j^{\mathsf{pk}}, (\mathsf{pk}_{j,0}, \ldots, \mathsf{pk}_{j,l_{\mathsf{sf}}-1}))$, where $\mathbf{ADRS}_j^{\mathsf{pk}}$ is the WOTS+C public-key-compression address of the j-th anchor
  // $\mathsf{pk}_j = \mathsf{Th}_{l_{\mathsf{sf}}}(P, \mathbf{ADRS}_j^{\mathsf{pk}}, \mathsf{pk}_{j,0} \| \ldots \| \mathsf{pk}_{j,l_{\mathsf{sf}}-1})$

end

Build the internal nodes with $\mathbf{H}$ through the collection oracle Th_λ to obtain the root $\mathsf{PK.sf}$.

Run Key Generation for the stateless part of the SHRINCS scheme, evaluating every hash call through Th_λ, to obtain $\mathsf{PK.sl}$.

Signal the SM-TCR challenger to proceed to the second stage and receive the public parameter P; set $\mathsf{PK.seed} \leftarrow P$.

Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ by simulating the random oracle.

for *each signing query* $(\widetilde{M}_j, (z^\star, j^\star))$*, with* j *the canonical index of the queried leaf* **do**

  Form the tweak $\mathsf{idx}_j = \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(j^\star, 8)$.

  Sample $R_j \leftarrow_\$ \mathcal{R}$ and reprogram $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}(R_j, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M}_j) := y_j$ as in **F.2**.

  Assemble the signature from R_j, the counter c_j, the blocks $(\sigma_{j,0}, \ldots, \sigma_{j,l_{\mathsf{sf}}-1})$, and the authentication path, and reply with it.

end

Obtain the forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ from $\mathcal{A}$ and recompute its verification chains and compression call from the blocks of σ^*.

if σ^* *is valid and the recomputation contains a* $\mathsf{Th}_{l_{\mathsf{sf}}}$*-call at a challenge address* $\mathbf{ADRS}_j^{\mathsf{pk}}$ *whose input tuple* m^* *differs from* $(\mathsf{pk}_{j,0}, \ldots, \mathsf{pk}_{j,l_{\mathsf{sf}}-1})$ *while* $\mathsf{Th}_{l_{\mathsf{sf}}}(\mathsf{PK.seed}, \mathbf{ADRS}_j^{\mathsf{pk}}, \mathsf{m}^*) = \mathsf{pk}_j$ **then**

  return (t, m^*)*, where* t *is the index of the challenge query at* $\mathbf{ADRS}_j^{\mathsf{pk}}$

else

  return *fail*

end

Algorithm 11: Reduction $\mathcal{B}_7$ against SM-TCR of $\mathbf{H} \in \mathsf{Th}_\lambda$

Input : Challenge oracle $\mathcal{O}(\cdot,\cdot)$ of the SM-TCR challenger (Definition 4.4) for $\mathbf{H} \in \mathsf{Th}_\lambda$ and collection oracle Th_λ (Section 4.3); EU-CMA forger $\mathcal{A}$ against FXMSS

Output : An SM-TCR collision (t, m^*) or fail

Run $\mathcal{A}$ to obtain the structure descriptor; enumerate the signing leaves as $j = 1, \ldots, L$ in a canonical order.

Sample the digests $y_1, \ldots, y_L \leftarrow_\$ \{0,1\}^{2n}$ and assign y_j to the j-th signing leaf `// matches F.2`

for $j \in [L]$ **do**

  Grind (d_j, c_j) as $\mathcal{O}^{\mathsf{Prop}}$ would, evaluating $\mathbf{H}_{\text{grind}}$ through the collection oracle Th_λ; parse d_j as $(a_{j,0}, \ldots, a_{j,l_{\mathsf{sf}}-1})$.

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ **do**

  Sample the fresh uniform value of **F.4**, $u_{j,i} \leftarrow_\$ \{0,1\}^n$, and compute the block $\sigma_{j,i}$ and the endpoint $\mathsf{pk}_{j,i}$ as **F.4** prescribes, evaluating each **F**-call through the collection oracle Th_λ.

end

Compress each tuple $(\mathsf{pk}_{j,0}, \ldots, \mathsf{pk}_{j,l_{\mathsf{sf}}-1})$ with $\mathsf{Th}_{l_{\mathsf{sf}}}$ through the collection oracle Th_λ to obtain the leaf values.

for *each internal node ν of the structure, in bottom-up order* **do**

  Query $\mathsf{node}_\nu \leftarrow \mathcal{O}(\mathbf{ADRS}_\nu^{\mathsf{tree}}, \mathsf{left}_\nu \,\|\, \mathsf{right}_\nu)$, where $\mathbf{ADRS}_\nu^{\mathsf{tree}}$ is the tree address of ν and left_ν, right_ν are the already-computed children of ν

  `//` $\mathsf{node}_\nu = \mathbf{H}(P, \mathbf{ADRS}_\nu^{\mathsf{tree}}, \mathsf{left}_\nu \| \mathsf{right}_\nu)$

end

Set $\mathsf{PK.sf}$ to the value of the root node.

Run Key Generation for the stateless part of the SHRINCS scheme, evaluating every hash call through Th_λ, to obtain $\mathsf{PK.sl}$.

Signal the SM-TCR challenger to proceed to the second stage and receive the public parameter P; set $\mathsf{PK.seed} \leftarrow P$.

Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ by simulating the random oracle.

for *each signing query $(\widetilde{M}_j, (z^\star, j^\star))$, with j the canonical index of the queried leaf* **do**

  Form the tweak $\mathsf{idx}_j = \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(j^\star, 8)$.

  Sample $R_j \leftarrow_\$ \mathcal{R}$ and reprogram $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}(R_j, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M}_j) := y_j$ as in **F.2**.

  Assemble the signature from R_j, the counter c_j, the blocks $(\sigma_{j,0}, \ldots, \sigma_{j,l_{\mathsf{sf}}-1})$, and the authentication path, and reply with it.

end

Obtain the forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ from $\mathcal{A}$ and recompute its verification path from the blocks of σ^*.

if *σ^* is valid and the recomputation contains an* **H***-call at a challenge address $\mathbf{ADRS}_\nu^{\mathsf{tree}}$ whose input m^* differs from $\mathsf{left}_\nu \| \mathsf{right}_\nu$ while* $\mathbf{H}(\mathsf{PK.seed}, \mathbf{ADRS}_\nu^{\mathsf{tree}}, \mathsf{m}^*) = \mathsf{node}_\nu$ **then**

  return (t, m^*), *where t is the index of the challenge query at $\mathbf{ADRS}_\nu^{\mathsf{tree}}$*

else

  return *fail*

end

Algorithm 12: Reduction $\mathcal{B}_8$ against SM-TCR-EXCL of $\mathsf{Th}_{2\cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda$

Input : Challenge oracle $\mathcal{O}(\cdot,\cdot)$ of the SM-TCR-EXCL challenger (Definition 4.7) for $\mathsf{Th}_{2\cup l_{\mathsf{sf}}} \in \mathsf{Th}_\lambda$ with the tweak partition $(\mathcal{T}_2, \mathcal{T}_3, \omega)$ of Claim 4.9, and collection oracle Th_λ (Section 4.3); EU-CMA forger $\mathcal{A}$ against FXMSS

Output : A SM-TCR-EXCL collision (t, T^*, m^*) or fail

Run $\mathcal{A}$ to obtain the structure descriptor; enumerate the signing leaves as $j = 1, \ldots, L$ in a canonical order.

Sample the digests $y_1, \ldots, y_L \leftarrow_{\$} \{0,1\}^{2n}$ and assign y_j to the j-th signing leaf `// matches F.2`

for $j \in [L]$ **do**

  Grind (d_j, c_j) as $\mathcal{O}^{\mathsf{Prop}}$ would, evaluating $\mathbf{H}_{\text{grind}}$ through the collection oracle Th_λ; parse d_j as $(a_{j,0}, \ldots, a_{j,l_{\mathsf{sf}}-1})$.

end

for $j \in [L]$, $i \in \{0, \ldots, l_{\mathsf{sf}} - 1\}$ **do**

  Sample the fresh uniform value of **F.4**, $u_{j,i} \leftarrow_{\$} \{0,1\}^n$, and compute the block $\sigma_{j,i}$ and the endpoint $\mathsf{pk}_{j,i}$ as **F.4** prescribes, evaluating each **F**-call through the collection oracle Th_λ.

end

for $j \in [L]$ **do**

  Query $\mathsf{pk}_j \leftarrow \mathcal{O}(\mathbf{ADRS}_j^{\mathsf{pk}}, (\mathsf{pk}_{j,0}, \ldots, \mathsf{pk}_{j,l_{\mathsf{sf}}-1}))$, where $\mathbf{ADRS}_j^{\mathsf{pk}} \in \mathcal{T}_3$ is the WOTS+C public-key-compression address of the j-th anchor `// challenge via` $\mathsf{Th}_{l_{\mathsf{sf}}}$

end

for *each internal node ν of the structure, in bottom-up order* **do**

  Query $\mathsf{node}_\nu \leftarrow \mathcal{O}(\mathbf{ADRS}_\nu^{\mathsf{tree}}, \mathsf{left}_\nu \,\|\, \mathsf{right}_\nu)$, where $\mathbf{ADRS}_\nu^{\mathsf{tree}} \in \mathcal{T}_2$ is the tree address of ν and left_ν, right_ν are the already-computed children of ν `// challenge via H`

end

Set $\mathsf{PK.sf}$ to the value of the root node.

Run Key Generation for the stateless part of the SHRINCS scheme, evaluating every hash call through Th_λ, to obtain $\mathsf{PK.sl}$.

Signal the SM-TCR-EXCL challenger to proceed to the second stage and receive the public parameter P; set $\mathsf{PK.seed} \leftarrow P$.

Return $\mathsf{pk} = (\mathsf{PK.seed}, \mathsf{PK.sl}, \mathsf{PK.sf})$ to $\mathcal{A}$; throughout, answer $\mathcal{A}$'s direct queries to $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}$ by simulating the random oracle.

for *each signing query $(\widetilde{M}_j, (z^\star, j^\star))$, with j the canonical index of the queried leaf* **do**

  Form the tweak $\mathsf{idx}_j = \mathsf{toByte}(z^\star, 1) \,\|\, \mathsf{toByte}(j^\star, 8)$.

  Sample $R_j \leftarrow_{\$} \mathcal{R}$ and reprogram $\mathbf{H}_{\mathbf{msg}}^{\mathsf{sf}}(R_j, \mathsf{PK.seed}, \mathsf{PK.sf}, \mathsf{idx}_j, \widetilde{M}_j) := y_j$ as in **F.2**.

  Assemble the signature from R_j, the counter c_j, the blocks $(\sigma_{j,0}, \ldots, \sigma_{j,l_{\mathsf{sf}}-1})$, and the authentication path, and reply with it.

end

Obtain the forgery $(\widetilde{M}^*, (z^\star, i^\star), \sigma^*)$ from $\mathcal{A}$ and recompute its verification path from the blocks of σ^*.

if *σ^* is valid and, at the first position walking towards the root where the recomputed value equals the value of an honest structure node ν, the producing hash call has tweak T^* equal to the challenge tweak T_t at ν or its bijection partner, with input $\mathsf{m}^* \neq \mathsf{m}_t$* **then**

  return (t, T^*, m^*)

else

  return *fail*

end

Algorithm 13: $\mathcal{B}$: reduction from $\mathsf{Avg\text{-}Search}_{1/|\mathcal{N}|}$ to SM-TCR-EXCL

Input : Oracle access to $f \leftarrow D_{1/|\mathcal{N}|}$ on $\{1,2,3\} \times [p] \times \mathcal{M}$, SM-TCR-EXCL adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

Output : $(k, j, \mathsf{m}^*) \in \{1,2,3\} \times [p] \times \mathcal{M}$

1 Sample $P' \leftarrow_{\$} \mathcal{P}$.

2 Sample a random tweakable hash function $\mathbf{Th}$, values $y_i \leftarrow_{\$} \mathcal{N}$ and random functions $g_i \colon \{1,2,3\} \times \mathcal{M} \to \mathcal{N} \setminus \{y_i\}$ for $i \in [p]$, and a random function $g' \colon \mathcal{T} \times \mathcal{M} \to \mathcal{N}$.

3 Run $\mathcal{A}_1$ with oracle access to $\mathbf{Th}(\cdot,\cdot,\cdot)$. Handle its challenge queries with the guards of Definition 4.7, and answer the ith admitted query (T_i, m_i) with y_i.

4 When $\mathcal{A}_1$ signals to continue, define $g \colon \mathcal{T} \times \mathcal{M} \to \mathcal{N}$ as follows, where i ranges over the admitted queries and k is the index with $t \in \mathcal{T}_k$:

$$g(t,\mathsf{m}) : \begin{cases} \text{if } (t = T_i \wedge \mathsf{m} = \mathsf{m}_i) & \text{Return } y_i \\ \text{if } (t = T_i \wedge f(k,i,\mathsf{m}) = 1) & \text{Return } y_i \\ \text{if } (t \in \mathcal{T}_2 \wedge t = \omega^{-1}(T_i) \wedge f(2,i,\mathsf{m}) = 1) & \text{Return } y_i \\ \text{if } (t \in \mathcal{T}_3 \wedge t = \omega(T_i) \wedge f(3,i,\mathsf{m}) = 1) & \text{Return } y_i \\ \text{else if } t \in \Omega(T_i) & \text{Return } g_i(k,\mathsf{m}) \\ \text{else} & \text{Return } g'(t,\mathsf{m}) \end{cases}$$

5 and the tweakable hash function $\mathbf{Th}'$ by $\mathbf{Th}'(P,t,x) : \begin{cases} \text{if } (P = P') : \text{ Return } g(t,x) \\ \text{Return } \mathbf{Th}(P,t,x) \end{cases}$

6 Run $\mathcal{A}_2^{\mathbf{Th}'(\cdot,\cdot,\cdot)}(P')$ and obtain (j, T^*, m^*).

7 Let k be the index with $T^* \in \mathcal{T}_k$, and output (k, j, m^*).