The Exact Bit Security of Hash-Based Signatures

Mikhail Kudinov [1] Benedikt Wagner [2] Emile Wambsgans [2]

[1] Blockstream Research
mishel.kudinov@gmail.com
[2] Ethereum Foundation
{first}@ethereum.org

Abstract

In this work, we analyze the security of hash-based signatures in Lean4. We examine how different ways of accounting for an adversary's work yield different notions of bit security. As our working example, we study two instances of XMSS with 128 bit digests.

Under most of the accounting modes we consider, we get 127 bit security for strong unforgeability. However, under the most conservative convention that counts only the adversary's *direct* random oracle queries, we only get 122 bits of security. We prove these bounds in the random oracle model without computational hardness assumptions and give matching attacks. We also prove a tight bound of 127 bits of strong unforgeability for SLH-DSA-128s.

Our work is primarily motivated by recent autoresearch competitions, and it demonstrates the concrete impact of resource accounting. We hope that it helps designers of future challenges make an informed decision when choosing a notion of bit security that reflects their needs.

Keywords: XMSS, SLH-DSA, hash-based signatures, bit security, Lean4, formal verification

1 Introduction

Hash-based signatures [BDH11, HBG+18, BHH+15, BHK+19, HKRY23] are becoming more and more important. Their foundations date back to the 1970s and 1980s [Lam79, Mer79, Gol87], while optimized constructions such as SPHINCS$^+$ [BHK+19] have been standardized [Nat24] under the name SLH-DSA. Hash-based designs are also under active consideration for use in Bitcoin [KN25, CHK+26] and Ethereum [DKKW25b, DKKW25a].

Bit security. The concrete security level of hash-based constructions is often expressed in *bits*. For instance, one might describe a hash-based signature scheme or succinct argument system as follows:

This construction has 128 *bits of security in the random oracle model.*

Describing the security of constructions by such a single number is often a preferred choice: it enables fair comparisons of different constructions, assuming they all achieve, say, 128 bits of security. Intuitively, in this context we say a construction has λ bits of security, if for any adversary $\mathcal{A}$ running in time at most t, we have

$$\epsilon/t \leq 2^{-\lambda},$$

where ϵ denotes the probability that $\mathcal{A}$ breaks the security notion in consideration. In the random oracle model, the number q of oracle queries provides a lower bound on the adversary's running time. The subtlety we study is how those queries should be counted.

Autoresearch and the need for precise definitions. A particular motivation for us to study the precise definition of bit security comes from the recent *autoresearch* paradigm. Taking the recent competition *sig.golf* [Eth26] as an example, AI agents of participants search for more efficient hash-based signature schemes under rules and optimization criteria set by human designers. The human designers provide a platform in which the agents can read the rules and optimization metrics. These rules are

specified in *Lean4* [dMU21] and the agents are forced to formally prove that their submissions adhere to the rules. For instance, when an agent submits a new hash-based signature scheme, it has to prove in Lean4 that it achieves required bits of security in the random oracle model. The definition of bit security in Lean4 therefore determines which designs the challenge accepts. Our observations show that this choice is critical: a natural parameter set satisfies the definition used in the first version of sig.golf, but fails to meet what we argue is a slightly more natural definition (only counting adversary queries).

Our Observations. To study how the exact definition of bit security affects the security level of a candidate scheme, we look at two instantiations of XMSS signatures [BDH11] as an example. One of them uses the classical checksum encoding due to Winternitz, and one of them uses the more recent target sum encoding [HKRY23]. Both use a lifetime of 2^{32} epochs and hash digests of 128 bit length.

We replace the time t by the number of queries q, where a *counting mode* Z defines *which* queries are counted: the adversary's direct hash queries q_{H} and signing queries q_S, and the hash queries made by the security game q_{Gen}, q_{Sig}, q_{Ver} (key generation, signing, and verification, see Definition 5). All our results, i.e., proving x bits of security and a matching attack at $x+1$ bits, are formally proven in Lean4[1], and we explain the interesting attacks here.

The first observation is that no scheme reaches 128 bits (Table 1): counting all queries gives 127 bits, counting only the adversary's queries gives 122 bits for strong unforgeability. Interestingly, target sum is always at least as secure as checksum.[2] Regarding autoresearch, our work thus shows that the choice of definition decides which submissions an autoresearch challenge accepts. We hope this helps to shape the second version of *sig.golf* and any future autoresearch efforts.

Standard unforgeability (eufcma)

	$Z = \{q_{\mathsf{H}}\}$	$Z = \{q_{\mathsf{H}}, q_S\}$	$Z = \{q_{\mathsf{H}}, q_{\mathsf{Sig}}\}$	$Z = \{q_{\mathsf{H}}, q_S, q_{\mathsf{Sig}}\}$	$Z \cap \{q_{\mathsf{Gen}}, q_{\mathsf{Ver}}\} \neq \varnothing$
SIG_{cs}	122	122	123	124	127
SIG_{ts}	126	127	127	127	127

Strong unforgeability (sufcma)

	$Z = \{q_{\mathsf{H}}\}$	$Z = \{q_{\mathsf{H}}, q_S\}$	$Z = \{q_{\mathsf{H}}, q_{\mathsf{Sig}}\}$	$Z = \{q_{\mathsf{H}}, q_S, q_{\mathsf{Sig}}\}$	$Z \cap \{q_{\mathsf{Gen}}, q_{\mathsf{Ver}}\} \neq \varnothing$
SIG_{cs}	122	122	123	124	127
SIG_{ts}	122	122	127	127	127

Table 1: Summary of our results for standard and strong unforgeability of XMSS [BDH11] with lifetime $L = 2^{32}$ and $n = 128$ bit digests, instantiated either with WOTS$^+$C [HKRY23] (SIG_{ts}: target sum encoding, no checksum chains, $v = 32$ chains of length $2^w = 16$, target sum $T = 330$) or with a classical checksum encoding (SIG_{cs}) (see Section 5.2). We consider all counting modes $Z \subseteq \{q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\}$ with $q_{\mathsf{H}} \in Z$ and compute the bit security according to Definition 5. Intuitively, the counting mode defines which queries we count as the adversary's work. See Definition 3 for the exact definition of the variables $q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}$.

Application to other schemes. In Section 7, we discuss how our findings apply to SLH-DSA [Nat24] and SHRINCS [CHK+26], and explain some takeaways for autoresearch.

2 Background on Lean4

For our proofs, we use Lean4 [dMU21], which is a dependently typed functional programming language and proof assistant. We use version v4.31.0. To model oracles (e.g., signing oracles, random oracles), we use the recent VCVio framework [TH24, TDW+26], commit `cbd4144`. Our code can be found at

[1] https://github.com/b-wagn/hashbased-bitsecurity-paper-lean, commit daba20d.

[2] Intuitively, grinding protects against "cheap" attacks where the adversary makes very few queries.

https://github.com/b-wagn/hashbased-bitsecurity-paper-lean/tree/daba20dce8055c83172206488df574a70e40fb36

The approach that we take when proving statements in Lean4 is the following: we carefully and manually review *what* is proven, which includes the definitions of security, the definition of the schemes, and the main theorem statements (on the order of 1000 lines of Lean4 code (without including VCVio)). We do not manually verify the Lean4 proofs, and instead rely on Lean4's kernel to check them, in combination with manually checking that the results do not contradict any known results from the literature and match our instincts. Provided the relevant modules build successfully and the theorems depend only on the accepted axioms, we treat the formal statements as proved, subject to the correctness of Lean's kernel, the soundness of its logic, and the assumption that VCVio models random oracles correctly. In our case, the accepted axioms are the standard ones (`propext`, `Classical.choice`, `Quot.sound`). We add a `#guard_msgs` command, so that the build fails if a `sorry` or a nonstandard axiom enters. For some of the attacks that we consider interesting, we provide a natural language description here, in addition to the Lean4 proof.

3 Epoch-Based Signatures

We focus on hash-based signatures and will take XMSS [BDH11] as our example. XMSS is an *epoch-based*[3] scheme, which means that signing and verification are done with respect to a given epoch. In this section, we first recall the definition of epoch-based signatures, as well as their security notion, namely, (strong) unforgeability.

Syntax and Correctness. First, we define what an epoch-based scheme is syntactically. As in a regular signature scheme, we have a key generation, a signing, and a verification algorithm. The main difference is that now the scheme has a lifetime L and signing and verification take as an additional input an *epoch* $\mathsf{ep} \in [L]$, where we write $[k] := \{0, \dots, k-1\}$ for an integer k. As our Lean4 proofs will be in the random oracle model, we directly write the syntax for schemes that have access to random oracles.

Definition 1 (Epoch-Based Signature Scheme)**.** Let $L \in \mathbb{N}$ be a natural number and H be a random oracle. An epoch-based signature scheme with lifetime L and with respect to random oracle H is a tuple of (efficient) algorithms $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ with the following syntax:

- $\mathsf{Gen}^{\mathsf{H}}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$
- $\mathsf{Sig}^{\mathsf{H}}(\mathsf{sk}, \mathsf{ep}, \mathsf{m}) \to \sigma / \bot$
- $\mathsf{Ver}^{\mathsf{H}}(\mathsf{pk}, \mathsf{ep}, \mathsf{m}, \sigma) \to b$ is deterministic

We denote the message length by l, i.e., $\mathsf{m} \in \{0,1\}^l$, and will assume throughout that $l = 256$.

For the scheme we consider, correctness holds with overwhelming probability in the random oracle model. To define this properly, we use an experiment in which an adversary can adaptively pick messages to be signed, and it wins if signing fails for any of those. As the keys are intended to be used only once per epoch, we add this constraint to the definition.

Definition 2 (Correctness)**.** Let $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be an epoch-based signature scheme with lifetime L with respect to random oracle H and let $\mathcal{A}$ be any algorithm. Consider the following experiment $\mathbf{CORRECT}_{\mathsf{SIG}}(\mathcal{A})$:

1. Generate keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}^{\mathsf{H}}(\mathsf{par})$ and set $v := 0$.
2. Run $\mathcal{A}$ on input pk, and with access to the following oracle (in addition to random oracle H):
   - $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ for $\mathsf{ep} \in [L], \mathsf{m} \in \{0,1\}^l$:
     (a) If $\mathsf{Signed}[\mathsf{ep}] \neq \bot$, then return $\bot$.
     (b) Otherwise, compute $\sigma \leftarrow \mathsf{Sig}^{\mathsf{H}}(\mathsf{sk}, \mathsf{ep}, \mathsf{m})$, set $\mathsf{Signed}[\mathsf{ep}] := (\mathsf{m}, \sigma)$.

[3] This is often called *stateful* or *synchronized*. We stick to the term *epoch-based*.

(c) If $\sigma = \bot$ or $\mathsf{Ver}^{\mathsf{H}}(\mathsf{pk}, \mathsf{ep}, \mathsf{m}, \sigma) = 0$, set $v := 1$.
(d) Return σ.

3. Output v.

For any algorithm $\mathcal{A}$, we define the following advantage:

$$\mathsf{Adv}^{\mathsf{correct}}_{\mathsf{SIG}}(\mathcal{A}) := \Pr\left[\mathbf{CORRECT}_{\mathsf{SIG}}(\mathcal{A}) = 1\right].$$

Informally, we say that the scheme has correctness error at most δ, if this advantage is at most δq conditioning on $\mathcal{A}$ making at most q queries to SIG.

Security. We now define security for such epoch-based schemes, where we use the standard notion of security for signature schemes, namely, unforgeability. The main difference compared to standard signature schemes is that we make the assumption that the honest signer never signs twice with respect to the same epoch. That is, one can sign multiple times, as long as one never signs for the same epoch twice. We define both a strong and a standard variant, modeling strong and standard unforgeability. These are often abbreviated as SUF-CMA and EUF-CMA, respectively. For our later definitions of bit security, it will be important to count the number of queries, both to the random oracle and to the signing oracle. For convenience, we make them part of the output of the game.

Definition 3 (Unforgeability)**.** Let $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be an epoch-based signature scheme with lifetime L with respect to random oracle H and let $\mathcal{A}$ be any algorithm. Consider the following experiment $\mathbf{SUFCMA}_{\mathsf{SIG}}(\mathcal{A})$:

1. Generate keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}^{\mathsf{H}}(\mathsf{par})$.
2. Run $\mathcal{A}$ on input pk, and with access to the following oracle (in addition to random oracle H):
   - $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ for $\mathsf{ep} \in [L], \mathsf{m} \in \{0,1\}^l$: If $\mathsf{Signed}[\mathsf{ep}] \neq \bot$, then return $\bot$. Otherwise, compute $\sigma \leftarrow \mathsf{Sig}^{\mathsf{H}}(\mathsf{sk}, \mathsf{ep}, \mathsf{m})$, set $\mathsf{Signed}[\mathsf{ep}] := (\mathsf{m}, \sigma)$ and return σ.
3. Obtain from $\mathcal{A}$ a forgery $(\mathsf{ep}^*, \mathsf{m}^*, \sigma^*)$ with $\mathsf{ep}^* \in [L]$ and $\mathsf{m}^* \in \{0,1\}^l$.
4. *Determine Winning.* Set $v := 1$ if it holds that $\mathsf{Ver}^{\mathsf{H}}(\mathsf{pk}, \mathsf{ep}^*, \mathsf{m}^*, \sigma^*) = 1$ and
$$(\mathsf{m}^*, \sigma^*) \neq \mathsf{Signed}[\mathsf{ep}^*]. \qquad (\star)$$
   Otherwise, set $v := 0$.
5. *Count Queries.* Count the number of queries as follows:
   - q_S: number of queries $\mathcal{A}$ made to SIG;
   - q_{H}: number of queries that $\mathcal{A}$ made *directly* to H;
   - q_{Gen}: number of queries that $\mathsf{Gen}^{\mathsf{H}}$ made to H;
   - q_{Sig}: number of queries that $\mathsf{Sig}^{\mathsf{H}}$ made to H, summed over all calls to SIG;
   - q_{Ver}: number of queries that $\mathsf{Ver}^{\mathsf{H}}$ made to H for evaluating the winning condition;
6. Output $(v, q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}})$.

For ease of notation, we write $v(\mathbf{SUFCMA}_{\mathsf{SIG}}(\mathcal{A}))$ for the random variable denoting the first component of the output, and similarly for the other components, e.g., $q_{\mathsf{H}}(\mathbf{SUFCMA}_{\mathsf{SIG}}(\mathcal{A}))$. For any algorithm $\mathcal{A}$, we define the following advantage:

$$\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{SIG}}(\mathcal{A}) := \Pr\left[v(\mathbf{SUFCMA}_{\mathsf{SIG}}(\mathcal{A})) = 1\right].$$

The game $\mathbf{EUFCMA}_{\mathsf{SIG}}(\mathcal{A})$ is the same, but we replace $(\star)$ with $\forall \sigma \in \{0,1\}^* \cup \{\bot\} : (\mathsf{m}^*, \sigma) \neq \mathsf{Signed}[\mathsf{ep}^*]$. We then define

$$\mathsf{Adv}^{\mathsf{eufcma}}_{\mathsf{SIG}}(\mathcal{A}) := \Pr\left[v(\mathbf{EUFCMA}_{\mathsf{SIG}}(\mathcal{A})) = 1\right].$$

4 Definitions of Bit Security

We will now define a set of notions of bit security. Intuitively, we get a different notion of bit security depending on how we count the adversary's work. We discuss other works on defining bit security in Section 8.

Bit Security Informally. Consider the unforgeability notion from Definition 3. Intuitively, λ bits of security means that for every adversary, its advantage-to-work ratio is at most $2^{-\lambda}$. That is, it is accepted if an adversary has high advantage, as long as it needs to spend a lot of time and resources. Making this a bit more precise, recall what we wrote in the introduction: a construction has λ bits of security, if for any adversary $\mathcal{A}$ running in time at most t and with advantage ϵ, we have $\epsilon/t \leq 2^{-\lambda}$. As we work in the random oracle model, we can replace lower bound t by the number of queries q that $\mathcal{A}$ made. Then, the definition becomes: for every adversary making at most q queries and with advantage ϵ, we have $\epsilon \leq q/2^{\lambda}$. This definition may be formal enough for a standalone paper, but when we work with Lean4, we need to be more precise. For instance:

- What does it mean that a given adversary makes at most q queries? The number of queries may depend on the public key and on random coins sampled by the adversary. We will require that it holds *for any choice* of randomness of the experiment.
- How do we count queries? That is, what should be part of q? Is $q = q_{\mathsf{H}}$, or $q = q_{\mathsf{H}} + q_S$, or the sum of any other subset of the counts from Definition 3? This is why we will get a set of definitions.

Bit Security Formally. We now give the formal definition. We first give a definition of what it means to make at most q queries.

Definition 4 (Counting Queries)**.** Let $Z \subseteq \{q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\}$ and $\mathcal{A}$ be any algorithm. Let $q \geq 1$ be an integer. Let SIG be an epoch-based signature scheme with lifetime L with respect to random oracle H. We say that $\mathcal{A}$ makes at most q many Z-queries in $\mathbf{SUFCMA}_{\mathsf{SIG}}$ if

$$\Pr\left[\sum_{\zeta \in Z} \zeta(\mathbf{SUFCMA}_{\mathsf{SIG}}(\mathcal{A})) \leq q\right] = 1.$$

Similarly, we say that $\mathcal{A}$ makes at most q many Z-queries in $\mathbf{EUFCMA}_{\mathsf{SIG}}$ if

$$\Pr\left[\sum_{\zeta \in Z} \zeta(\mathbf{EUFCMA}_{\mathsf{SIG}}(\mathcal{A})) \leq q\right] = 1.$$

The definition deserves some explanation. For that, consider the example $Z = \{q_S, q_{\mathsf{H}}, q_{\mathsf{Ver}}\}$. Then, the definition says that an adversary makes at most q queries (for this Z) in the security game, if for any choice of random coins of the experiment, we have $q_S + q_{\mathsf{H}} + q_{\mathsf{Ver}} \leq q$. That is, we count the number of signing queries, the number of direct hash calls, and the number of hash calls to verify the final forgery into the adversary's work. Another viable option would be to only count $q_S + q_{\mathsf{H}}$, in which case we have $Z = \{q_S, q_{\mathsf{H}}\}$. [4] We are now ready to define notions of bit security. For each Z as above, we get a notion.

Definition 5 (Bit Security)**.** Let $Z \subseteq \{q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\}$. Let SIG be an epoch-based signature scheme with lifetime L with respect to random oracle H. Let $\lambda \in \mathbb{N}$. We say that SIG has λ bits of security in counting mode Z for strong unforgeability, if for any $q \geq 1$ and any classical adversary $\mathcal{A}$ that makes at most q many Z-queries in $\mathbf{SUFCMA}_{\mathsf{SIG}}$, we have

$$\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{SIG}}(\mathcal{A}) \leq q/2^{\lambda}.$$

We say that SIG has λ bits of security in counting mode Z for unforgeability, if for any $q \geq 1$ and any $\mathcal{A}$ that makes at most q many Z-queries in $\mathbf{EUFCMA}_{\mathsf{SIG}}$, we have

$$\mathsf{Adv}^{\mathsf{eufcma}}_{\mathsf{SIG}}(\mathcal{A}) \leq q/2^{\lambda}.$$

[4] In general, any meaningful notion of bit security should at least count the adversary's direct queries q_{H}.

We only consider integer levels λ. Note that Definition 5 only considers classical adversaries, i.e., adversaries that query H on classical inputs. We leave the analogous notions for quantum adversaries in the quantum random oracle model, where counting queries is more subtle, for future work.

5 Background on Hash-Based Signatures

We use XMSS [BDH11] as the working example to show subtleties in defining bit security. For readers not yet familiar with XMSS, we present the construction in this section. For readers familiar with it: we consider XMSS with WOTS$^+$C [HKRY23], i.e., with a target sum encoding with $v = 32$ chains of length $2^w = 16$ and target sum $T = 330$, and XMSS with WOTS-TW [HK22], i.e., with a classical checksum encoding. Both use a lifetime of $L = 2^{32}$, $n = 128$ bit digests, a random public parameter, and standard domain separation for all hash invocations. We denote the two schemes by SIG_{ts} and SIG_{cs}, respectively (Section 5.2).

5.1 Informal XMSS Overview

Let us give an informal overview of XMSS, for readers not yet familiar with it.

One-Time Signature From Hash Chains. The starting point is to build a one-time signature scheme, i.e., a signature scheme that is secure as long as the signer signs at most once. A classical approach to build such a scheme uses *hash chains*. Namely, during key generation, we sample v independent secret values and use each as the starting point of a hash chain. Each chain has 2^w values, indexed from 0 to $2^w - 1$, where each value is obtained by hashing the previous one. The secret key consists of the starting values, while endpoints serve as the public key. To sign a message, the signer first runs it (together with a randomizer) through an encoding function, which outputs a vector $x = (x_0, \dots, x_{v-1}) \in \{0, \dots, 2^w - 1\}^v$. This encoding x intuitively points into the hash chains. The signature contains for each chain i exactly one element, indexed by x_i. Together with the randomizer, these v values form the signature. To verify, the verifier recomputes the encoding and hashes each revealed value another $2^w - 1 - x_j$ times. In this way, it recomputes the chain ends, which it then compares to the public keys. At an intuitive level, a message determines how the work of traversing each chain is split between the signer and the verifier.

Incomparable Encodings. There is a simple attack that the encoding must prevent. Suppose we have seen a signature for a message with encoding x. Since we can always hash forwards, we can turn this into a signature for any message with encoding x' satisfying $x'_j \geq x_j$ for every j. Thus, what we need for one-time signatures based on hash chains is an *incomparable* encoding, where moving to a different valid encoding requires going backwards on at least one chain. Note that this incomparability requirement does not rule out finding a different message with the same encoding. This is why the encoding also needs to satisfy a form of target collision resistance, which intuitively forces $v \cdot w$ to be sufficiently large, and imposes a work-size trade-off. We now describe two encodings achieving incomparability.

Checksum Encoding. The classical approach is to use a *checksum* encoding, which adds $v_2 = \lfloor \log_2(v \cdot (2^w - 1))/w \rfloor + 1$ new chains. The signer samples a randomizer and hashes it alongside the message, giving the encoding $(x_0, \dots, x_{v-1}) \in [2^w]^v$. It reveals the corresponding v chain values as described above. In addition, the signer computes the checksum $v \cdot (2^w - 1) - \sum_{i \in [v]} x_i$, writes it in base 2^w as $(x_v, \dots, x_{v+v_2-1})$, and encodes it via the last v_2 chains. The intuition for security is that every other encoding $x' \neq x$ obtainable by hashing forward the first v chains (i.e., $x'_i \geq x_i$ for all i) leads to a strictly smaller checksum, and so at least one of the checksum digits $x_v, \dots, x_{v+v_2-1}$ has to go backwards.

Target Sum Encoding. The target sum encoding [HKRY23] avoids the additional checksum chains. Instead, it achieves incomparability by requiring all valid encodings to have the same sum $\sum_i x_i$: two distinct encodings with the same sum cannot satisfy $x'_i \geq x_i$ for all i. To obtain such an encoding, the signer hashes the message together with a fresh randomizer, interprets the result as $v = 32$ four-bit chunks, and retries with a new randomizer until their sum is the target 330. The successful randomizer is included in the signature, so the verifier can recompute the chunks and check their sum. Compared to

the checksum encoding, this saves the v_2 checksum chains in the signature and in verification, at the cost of additional work for the signer.

From One-Time to Many-Time. We can use Merkle trees to turn such a one-time secure scheme into an epoch-based many-time scheme. To this end, we generate L one-time key pairs $(\mathsf{pk}_i, \mathsf{sk}_i), i \in [L]$, and commit to the list of public keys using a Merkle tree. The root of this tree is then taken to be the long-term public key pk, and all secret keys serve as the long-term secret key[5]. To sign a message with respect to epoch $\mathsf{ep} \in [L]$, we sign it with $\mathsf{sk}_{\mathsf{ep}}$, and attach a Merkle path authenticating $\mathsf{pk}_{\mathsf{ep}}$ to the one-time signature. The verifier can use the one-time signature to re-derive $\mathsf{pk}_{\mathsf{ep}}$ and then checks the authentication path.

Domain Separation. So far, we have spoken of hashing as if every step used the same function. To enable tighter security proofs and thus better parameters for a given security target, we prefix each hash input with a random public parameter P and a tweak τ. The parameter P is sampled once during key generation, is shared by all hash computations under that key, and is included in the long-term public key alongside the Merkle root. Its purpose is mainly better multi-user security and to get proofs relying on hash function properties in the standard model. The tweak τ identifies where the hash is used: for a chain step, it specifies the epoch, chain index, and position; for a Merkle-tree node, it specifies the level and node index. Message hashing uses a separate type of tweak that includes the epoch.

5.2 XMSS Definition

We now give a more formal definition of the two schemes we use. For that, let $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^n$ be a random oracle. Both schemes are instances of one somewhat generic construction $\mathsf{SIG}[\mathsf{Encode}, v, K]$, parameterized by the number of chains v, the maximum number of signing attempts K, and an encoding function

$$\mathsf{Encode}\colon \{0,1\}^n \to [2^w]^v \cup \{\bot\},$$

that (if it does not abort) maps a digest to a vector of v chain positions. After presenting the generic constructions, we then define the two instantiations SIG_{ts} and SIG_{cs}, using a target sum and checksum encoding, respectively.

Notation and Tweaks. For $D \in \{0,1\}^n$ we write D_b for the b-th least significant bit of D. For bits b_3, b_2, b_1, b_0 we write $(b_3 b_2 b_1 b_0)_2 := 8b_3 + 4b_2 + 2b_1 + b_0$. Every call to H starts with a 9-byte *tweak*

$$\mathsf{enc}(t, p, j) := t \,\|\, p \,\|\, j$$

consisting of a byte t, a 32-bit integer p, and a 32-bit integer j, both encoded little-endian, so that different triples (t, p, j) give different tweaks. The tweak is followed by the public parameter P and the actual input.

Parameters. We focus on the following parameters, which both schemes share:

$$n = 128, \qquad h = 32, \qquad L = 2^h, \qquad w = 4.$$

That is, digests, the public parameter P, and randomizers ρ are $n = 128$ bits long, and we consider a Merkle tree of height $h = 32$, i.e., with 2^{32} leaves, and every hash chain has 16 positions, indexed as $0, \ldots, 15$. Further, messages are assumed to be 256 bits long.

Hash Chains. We define a function that walks a value from position s to position $t \geq s$ of chain i in epoch ep, making $t - s$ queries to H.

- $\mathsf{Chain}^{\mathsf{H}}(P, \mathsf{ep}, i, s, t, x) \to y$:
  1. For $k = s, \ldots, t-1$: set $x := \mathsf{H}(\mathsf{enc}(1, 2^w \cdot i + k, \mathsf{ep}) \,\|\, P \,\|\, x)$.
  2. Return $y := x$.

[5] In practice, the secret keys are all generated using a pseudorandom function from a short seed.

The Generic Construction. We now define the generic scheme $\mathsf{SIG}[\mathsf{Encode}, v, K] = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ with lifetime L. During key generation, we sample a random seed from which everything is derived. Key generation also generates the entire Merkle tree and stores it in the secret key, so that signing only has to hash to find the encoding. Note that for the notions of bit security, it indeed may matter which queries the signer makes: in our description here (and in the Lean code), Sig queries H exactly twice per attempt, namely for ρ and for D.

- $\mathsf{Gen}^{\mathsf{H}}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$:
  1. $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{256}$
  2. $P := \mathsf{H}(\mathsf{enc}(5,0,0) \,\|\, \mathsf{seed})$.
  3. For all $(\mathsf{ep}, i) \in [L] \times [v]$:
     (a) $\mathsf{sk}_{\mathsf{ep},i} := \mathsf{H}(\mathsf{enc}(0,i,\mathsf{ep}) \,\|\, P \,\|\, \mathsf{seed})$.
     (b) $c_{\mathsf{ep},i,0} := \mathsf{sk}_{\mathsf{ep},i}$, $c_{\mathsf{ep},i,d} := \mathsf{Chain}^{\mathsf{H}}(P, \mathsf{ep}, i, d-1, d, c_{\mathsf{ep},i,d-1})$ for $d = 1, \ldots, 2^w - 1$.
     (c) Set $\mathsf{pk}_{\mathsf{ep},i} := c_{\mathsf{ep},i,2^w-1}$.
  4. For all $\mathsf{ep} \in [L]$: $X_{0,\mathsf{ep}} := \mathsf{H}(\mathsf{enc}(2,0,\mathsf{ep}) \,\|\, P \,\|\, \mathsf{pk}_{\mathsf{ep},0} \,\|\, \cdots \,\|\, \mathsf{pk}_{\mathsf{ep},v-1})$.
  5. For all $\ell = 1, \ldots, h$:
     (a) For all $j \in [2^{h-\ell}]$: $X_{\ell,j} := \mathsf{H}(\mathsf{enc}(3,\ell,j) \,\|\, P \,\|\, X_{\ell-1,2j} \,\|\, X_{\ell-1,2j+1})$.
  6. Set $\mathsf{pk} := (\mathsf{root}, P)$ with $\mathsf{root} := X_{h,0}$.
  7. Set $\mathsf{sk} := \big(\mathsf{seed}, P, (c_{\mathsf{ep},i,d})_{\mathsf{ep}\in[L], i\in[v], d\in[2^w]}, (X_{\ell,j})_{\ell\in[h+1], j\in[2^{h-\ell}]}\big)$.
- $\mathsf{Sig}^{\mathsf{H}}(\mathsf{sk}, \mathsf{ep}, \mathsf{m}) \to \sigma/\bot$:
  1. For $a = 0, 1, \ldots, K-1$:
     (a) $\rho := \mathsf{H}(\mathsf{enc}(7,a,\mathsf{ep}) \,\|\, P \,\|\, \mathsf{seed} \,\|\, \mathsf{m})$.
     (b) $D := \mathsf{H}(\mathsf{enc}(4,0,\mathsf{ep}) \,\|\, P \,\|\, \mathsf{m} \,\|\, \rho)$, $x := \mathsf{Encode}(D)$.
     (c) If $x \neq \bot$:
        i. For all $i \in [v]$: $\sigma_i := c_{\mathsf{ep},i,x_i}$.
        ii. For all $\ell \in [h]$: $A_\ell := X_{\ell,\lfloor \mathsf{ep}/2^\ell \rfloor \oplus 1}$.
        iii. Return $\sigma := \big(\rho, (\sigma_i)_{i\in[v]}, (A_\ell)_{\ell\in[h]}\big)$.
  2. Return $\bot$.
- $\mathsf{Ver}^{\mathsf{H}}(\mathsf{pk}, \mathsf{ep}, \mathsf{m}, \sigma) \to b$, where $\mathsf{pk} = (\mathsf{root}, P)$ and $\sigma = (\rho, (\sigma_i)_{i\in[v]}, (A_\ell)_{\ell\in[h]})$:
  1. $D := \mathsf{H}(\mathsf{enc}(4,0,\mathsf{ep}) \,\|\, P \,\|\, \mathsf{m} \,\|\, \rho)$, $x := \mathsf{Encode}(D)$.
  2. If $x = \bot$: Return $b := 0$.
  3. For all $i \in [v]$: $\mathsf{pk}'_i := \mathsf{Chain}^{\mathsf{H}}(P, \mathsf{ep}, i, x_i, 2^w - 1, \sigma_i)$.
  4. $X'_0 := \mathsf{H}(\mathsf{enc}(2,0,\mathsf{ep}) \,\|\, P \,\|\, \mathsf{pk}'_0 \,\|\, \cdots \,\|\, \mathsf{pk}'_{v-1})$.
  5. For $\ell = 0, \ldots, h-1$:
     (a) $j := \lfloor \mathsf{ep}/2^{\ell+1} \rfloor$
     (b) If bit ℓ of ep is 0, set $X'_{\ell+1} := \mathsf{H}(\mathsf{enc}(3,\ell+1,j) \,\|\, P \,\|\, X'_\ell \,\|\, A_\ell)$; otherwise, set $X'_{\ell+1} := \mathsf{H}(\mathsf{enc}(3,\ell+1,j) \,\|\, P \,\|\, A_\ell \,\|\, X'_\ell)$.
  6. Return $b := 1$ if $X'_h = \mathsf{root}$, and $b := 0$ otherwise.

The Two Instantiations. We get the two schemes SIG_{ts} and SIG_{cs} by instantiating Encode, v, and K. One of them uses the target sum encoding, one the classical checksum encoding. More concretely:

- **Target Sum, WOTS$^+$C (SIG_{ts}):**
  - $v = 32$, $K = 2^{32}$, and target sum $T = 330$.
  - Encode_{ts} on input $D \in \{0,1\}^{128}$:

1. Read D as 32 digits of $w = 4$ bits. That is, for $i \in [32]$ let $x_i := (D_{4i+3}D_{4i+2}D_{4i+1}D_{4i})_2$.
2. If $\sum_{i\in[32]} x_i = T$, return $x := (x_0, \ldots, x_{31})$. Otherwise, return $\perp$.

- **Checksum, WOTS-TW (SIG_{cs}):**
  - $v = \mathsf{len}_1 + \mathsf{len}_2 = 35$, where $\mathsf{len}_1 = \lceil n/w \rceil = 32$ is the number of message digits and $\mathsf{len}_2 = \lfloor \log_{2^w}(\mathsf{len}_1(2^w - 1)) \rfloor + 1 = 3$ is the number of checksum digits, $K = 1$.
  - Encode_{cs} on input $D \in \{0,1\}^{128}$:
    1. For $i \in [32]$ let $x_i := (D_{4i+3}D_{4i+2}D_{4i+1}D_{4i})_2$.
    2. Compute the checksum $\mathsf{csum} := \sum_{i\in[32]}(2^w - 1 - x_i) \le 480 < 16^3$, and write it in base $2^w = 16$ with three digits, most significant first: $x_{32+j} := \lfloor \mathsf{csum}/16^{2-j} \rfloor \bmod 16$ for $j \in [3]$.
    3. Return $x := (x_0, \ldots, x_{34})$. In particular, Encode_{cs} never returns $\perp$, which is why a single attempt ($K = 1$) suffices.

Theorem 1 (Correctness). *SIG_{cs} has correctness error 0, and SIG_{ts} has correctness error at most 2^{-256} (cf. Definition 2).*

Proof. The Lean4 code can be found at:

Xmss/Statement/Theorems/Correctness.lean#L16-L27

□

The error is not zero for SIG_{ts} because signing may fail if all 2^{32} randomizers give invalid encodings.

6 Results

We now present our results, both upper and lower bounds, both for standard and strong unforgeability. Importantly, we will see that the results differ depending on which counting mode is used, i.e., which Z in Definition 5. Arguably, any reasonable definition would at least count the adversary's direct queries q_{H} to the random oracle. So we only focus on such Z. We give an overview of our results in Table 1.

6.1 General Upper Bound: ≤ 127 Bits for Every Mode

Before we turn to results that depend on the counting mode, we describe an attack that rules out 128 bits of security for both schemes, for both unforgeability and strong unforgeability, and in every counting mode Z. Thus, it shows that $\lambda \le 127$ in *every* entry of Table 1. Note that natural generic attacks do not suffice for this. For instance, an adversary that searches for a preimage of the Merkle root, or for a second preimage of any honestly computed value, succeeds with probability at most 2^{-n} per query, and thus achieves an advantage of at most $q/2^{128}$ with q queries. Our attack gains a factor of (almost) two over such a search, which is what pushes it below 128 bits.

Notation. We recall some notations: a signature for epoch ep is $\sigma = (\rho, (\sigma_i)_{i\in[v]}, (A_\ell)_{\ell\in[h]})$, where ρ is the randomizer, σ_i is the revealed value of chain i, and $A_0, \ldots, A_{h-1}$ is the authentication path, ordered from the sibling A_0 of the leaf up to the child A_{h-1} of the root. We write $X_\ell := X_{\ell,\lfloor \mathsf{ep}/2^\ell \rfloor}$ for the honest node[6] at level ℓ on the path of epoch ep, so that X_0 is the leaf, $X_h = \mathsf{root}$, and A_ℓ is the sibling of X_ℓ. For SIG_{cs}, the $v = 35$ chain values σ_i include the $\mathsf{len}_2 = 3$ checksum chains (cf. Section 5.2). Further, for each chain k and $z \in \{1, \ldots, 2^w - 1\}$, we write $f_{k,z} := \mathsf{Chain}^{\mathsf{H}}(P, \mathsf{ep}, k, z-1, z, \cdot) \colon \{0,1\}^n \to \{0,1\}^n$ for the hash evaluation that maps the value at position $z-1$ of chain k in epoch ep to the value at position z.

Neighboring Encodings. The idea of the attack is to find a new message whose encoding x' is a *neighbor* of a signed encoding $x = (x_0, \ldots, x_{v-1})$, i.e., a valid encoding that goes one step backwards on some chain i and one step forwards on some chain $j \ne i$: $x'_i = x_i - 1$, $x'_j = x_j + 1$, and $x'_k = x_k$ otherwise. The forward step on chain j only requires one hash evaluation, and the backward step on chain i requires a preimage of σ_i under f_{i,x_i}.

[6] By honest node, we mean the value that the honest signer has computed during key generation.

The Attack. The adversary $\mathcal{A}$ is parameterized by two budgets $B, Q \geq 1$ and proceeds as follows:

1. On input pk, query $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ for an arbitrary epoch ep and message m, and obtain $\sigma = (\rho, (\sigma_i)_{i\in[v]}, (A_\ell)_{\ell\in[h]})$. Make one query to H to compute the encoding x of (m, ρ) for epoch ep.

2. Fix any message $\mathsf{m}' \neq \mathsf{m}$ and let $\rho'_1, \ldots, \rho'_B \in \{0,1\}^n$ be distinct, e.g., the first B strings in lexicographic order. For $b = 1, \ldots, B$, let $\rho' := \rho'_b$ and compute the encoding x' of (m', ρ') for epoch ep using one query to H. If x' is a neighbor of x, with indices i and j as above, go to the next step. If no such b exists, output $\perp$.

3. Let $y_1, \ldots, y_Q \in \{0,1\}^n$ be distinct, e.g., the first Q strings in lexicographic order. For $u = 1, \ldots, Q$, check whether $f_{i,x_i}(y_u) = \sigma_i$. If so, output the forgery $(\mathsf{ep}, \mathsf{m}', \sigma')$ with

$$\sigma' = (\rho', (\sigma'_k)_{k\in[v]}, (A_\ell)_{\ell\in[h]}), \quad \text{where} \quad \sigma'_k = \begin{cases} y_u & \text{if } k = i, \\ f_{j,x_j+1}(\sigma_j) & \text{if } k = j, \\ \sigma_k & \text{otherwise.} \end{cases}$$

   If no such u exists, output $\perp$.

That is, intuitively we use B attempts to find a neighbor encoding, and then Q attempts to invert the single step that needs to be inverted.

Analysis. We now want to lower bound the probability that $\mathcal{A}$ outputs a forgery. For Step 2, let $N(x)$ be the number of neighbors of x. For both schemes, the ρ'_b are distinct and $\mathsf{m}' \neq \mathsf{m}$, so the B digests D are outputs of H on fresh inputs, and thus uniform in $\{0,1\}^{128}$ and independent of each other and of everything else. Further, every fixed valid encoding is the output of Encode for exactly one D. Thus, each iteration finds a neighbor with probability $N(x) \cdot 2^{-n} = N(x) \cdot 2^{-128}$. For SIG_{ts}, every pair (i, j) with $i \neq j$, $x_i \geq 1$, and $x_j \leq 2^w - 2$ yields a neighbor, as it preserves the sum. With our parameters, every encoding with sum $T = 330$ then satisfies $N(x) \geq 220$. For SIG_{cs}, not every such pair yields a valid encoding, as the checksum digits are determined by the message digits.[7] We therefore only count pairs (i, j) among the 32 message digits with $x_i \geq 1$ and $x_j \leq 2^w - 2$. Each of them yields a neighbor, as the sum of the message digits and thus the checksum digits are unchanged. As the signed encoding is uniform, this gives $N(x) \geq 220$ except with probability below 2^{-65}. Choosing $B = 2^{122}$, Step 2 fails with probability at most $(1 - 220 \cdot 2^{-128})^B + 2^{-65} \leq e^{-220/64} + 2^{-65} \leq 2^{-4.9}$ for both schemes, where the term 2^{-65} is only needed for SIG_{cs}.

Now, condition on Step 2 succeeding and consider Step 3. Let s be the honest value at position $x_i - 1$ of chain i, so that $\sigma_i = f_{i,x_i}(s)$. The crucial observation is that σ_i is not a uniformly random target, but the image of s, which is uniformly random in $\{0,1\}^n$ and thus lies in the search space. More precisely, s is an output of H on an input with a chain tweak, and is therefore independent of the encodings x and x', and thus of the choice of i in Step 2, which only depend on outputs of H with the message tweak. Hence, a candidate y_u is a valid preimage either if $y_u = s$, or if $y_u \neq s$ and $f_{i,x_i}(y_u) = \sigma_i$ by chance. The first case happens for some u with probability $Q/2^n$. In the second case, $f_{i,x_i}(y_u)$ is uniform and independent of σ_i, as f_{i,x_i} has only been evaluated on s before Step 3, namely during key generation. Signing does not evaluate any chain, and the only other chain step of $\mathcal{A}$ is on chain $j \neq i$. Thus, Step 3 succeeds with probability

$$p_Q := 1 - \left(1 - \frac{Q}{2^n}\right)\left(1 - \frac{1}{2^n}\right)^Q \approx 1 - \left(1 - \frac{Q}{2^n}\right) e^{-Q/2^n}.$$

For $Q \ll 2^n$, we have $p_Q \approx 2Q/2^n$, i.e., each candidate succeeds with probability roughly 2^{-127} instead of 2^{-128}. We choose $Q = 2^{126}$, which gives $p_Q \approx 0.416$. Overall,

$$\mathsf{Adv}^{\mathsf{eufcma}}_{\mathsf{SIG}}(\mathcal{A}) \geq (1 - 2^{-4.9}) \cdot p_Q \approx 0.402.$$

[7] Pairs that change the checksum digits are valid only in special cases, e.g., moving the least significant checksum digit in the direction opposite to a message digit, if this causes no carry.

Implications. The adversary makes $q_S = 1$ signing query and $q_{\mathsf{H}} \leq B + Q + 2$ direct queries to H, namely one to compute x, B in Step 2, Q in Step 3, and one for the forward step on chain j. Further, q_{Gen}, q_{Sig}, and q_{Ver} do not depend on B and Q: key generation and signing are run once, and the verifier checks a single forgery. Concretely, $q_{\mathsf{Gen}} = L \cdot (v \cdot 2^w + 2) \leq 2^{42}$, and in the worst case, $q_{\mathsf{Sig}} \leq 2K \leq 2^{33}$ and $q_{\mathsf{Ver}} \leq 2 + v(2^w - 1) + h \leq 2^{10}$. Thus, for every counting mode Z, the adversary makes at most $q \leq B + Q + 2^{43} \leq 2^{126} \cdot (1 + 2^{-3.9})$ many Z-queries, and the ratio between advantage and queries is at least

$$\frac{0.402}{2^{126} \cdot (1 + 2^{-3.9})} \approx 2^{-127.4} > 2^{-128}.$$

Hence, neither SIG_{ts} nor SIG_{cs} has 128 bits of security for unforgeability or strong unforgeability, in any counting mode Z.

Intuition. The intuition for the gain of roughly 1 bit is closely related to the analysis of Step 3: each query essentially yields *two* ways of winning: either by guessing the honest preimage, or by finding a target collision.

6.2 Results for Strong Unforgeability

We now consider strong unforgeability. That is, the adversary wins even if it forges a new signature on a previously signed message (cf. Definition 3).

6.2.1 Proven Bounds

Let us first state the bounds that we have proven in Lean4. Existing literature on manually written proofs [HK22, DKKW25b] suggests that we can prove strong unforgeability with security *close to* 128 bits. Our goal was to find the exact bound, for each given Z in the definition of bit security (Definition 5). We get the following result.

Theorem 2 (Bit Security for Strong Unforgeability - Target Sum). *Let SIG_{ts} be the epoch-based signature scheme with target sum encoding from Section 5.2. Let $Z \subseteq \{q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\}$ with $q_{\mathsf{H}} \in Z$. Let λ be the maximum integer so that SIG_{ts} has λ bits of security in counting mode Z for strong unforgeability. Then:*

- *If $Z \subseteq \{q_{\mathsf{H}}, q_S\}$, then $\lambda = 122$.*
- *If $Z \cap \{q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\} \neq \varnothing$, then $\lambda = 127$.*

Proof. The Lean4 code can be found at:

`Xmss/Statement/Theorems/StrongUnforgeability.lean#L21-L23`

The attack showing $\lambda \leq 127$ is the generic one from Section 6.1, and the attack showing $\lambda \leq 122$ is presented in Section 6.2.2. □

Interestingly, counting the queries to H of any of Gen, Sig, or Ver raises the level from 122 to 127 bits. This is because Gen makes $\approx 2^{41}$ queries, Sig up to 2^{33}, and Ver 184 on the forgery, making the attack from Section 6.2.2 too expensive.

Theorem 3 (Bit Security for Strong Unforgeability - Checksum). *Let SIG_{cs} be the epoch-based signature scheme with checksum encoding from Section 5.2. Let $Z \subseteq \{q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\}$ with $q_{\mathsf{H}} \in Z$. Let λ be the maximum integer so that SIG_{cs} has λ bits of security in counting mode Z for strong unforgeability. Then:*

- *If $q_{\mathsf{Gen}} \in Z \vee q_{\mathsf{Ver}} \in Z$, then $\lambda = 127$.*
- *Else if $\{q_{\mathsf{Sig}}, q_S\} \subseteq Z$, then $\lambda = 124$.*
- *Else if $q_{\mathsf{Sig}} \in Z$, then $\lambda = 123$.*
- *Else, $\lambda = 122$.*

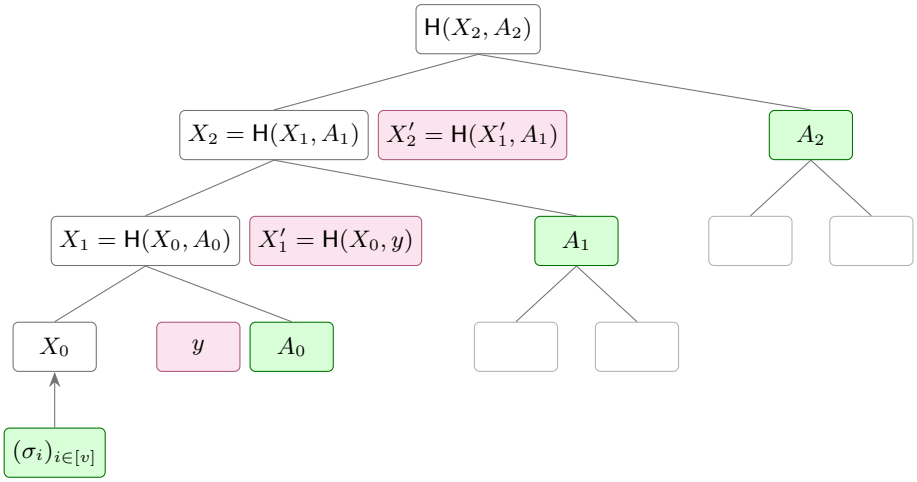


Figure 1: The attack against strong unforgeability for a tree of height $h = 3$. Green values are taken from the honest signature: the one-time signature, from which the verifier recomputes the honest leaf X_0, and the authentication path (A_0, A_1, A_2). The adversary replaces A_0 by a random y, so the verifier computes $X_1' = \mathsf{H}(X_0, y)$ instead of $X_1 = \mathsf{H}(X_0, A_0)$ (pink). The forgery is valid if $X_1' = X_1$, $X_2' = X_2$, or $\mathsf{H}(X_2', A_2) = \mathsf{H}(X_2, A_2)$, i.e., if one of the h evaluations on the path hits the honest node. For readability, we omit the public parameter and the tweaks.

Proof. The Lean4 code can be found at:

`Xmss/Statement/Theorems/StrongUnforgeability.lean#L28-L32`

The generic attack from Section 6.1 shows $\lambda \leq 127$. The remaining bounds follow from Section 6.2.2. The variant replacing a chain value σ_k shows $\lambda \leq 122$, and $\lambda \leq 123$ when counting the signer's 2 queries to H, i.e., q_{Sig}. When counting both q_{Sig} and q_S, already the attack replacing A_0 shows $\lambda \leq 124$. □

6.2.2 Attacks

We describe an attack that makes no direct random oracle queries. Instead, it relies on the hash evaluations that the verifier performs when checking the forgery. Hence, it yields interesting results for counting modes Z with $q_{\mathsf{Ver}} \notin Z$. We use the notation from Section 6.1.

SUF Attack via the Authentication Path. The adversary $\mathcal{A}$ proceeds as follows:

1. On input pk, query $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ for an arbitrary epoch ep and message m, and obtain $\sigma = (\rho, (\sigma_i)_{i\in[v]}, (A_\ell)_{\ell\in[h]})$.
2. Sample $y \xleftarrow{\$} \{0,1\}^n \setminus \{A_0\}$ and output the forgery $(\mathsf{ep}, \mathsf{m}, \sigma^*)$, where

$$\sigma^* = (\rho, (\sigma_i)_{i\in[v]}, (y, A_1, \ldots, A_{h-1})).$$

That is, $\mathcal{A}$ replaces the lowest node of the authentication path by a random string and keeps everything else, including the randomizer ρ and all chain values. We illustrate the attack in Figure 1.

Analysis. As $y \neq A_0$, we have $(\mathsf{m}, \sigma^*) \neq \mathsf{Signed}[\mathsf{ep}]$, so a valid σ^* is a strong forgery. Since ρ and all chain values are unchanged, the verifier recomputes the honest encoding, the honest chain ends, and the honest leaf. In particular, the attack does not depend on the encoding, and it applies to SIG_{ts} and SIG_{cs} alike. Now, consider the h hash evaluations that the verifier performs on the path to the root. The first one compresses the honest leaf together with y, and thus is on an input that has not been queried before. If its output differs from the honest node, then the next evaluation is also on a fresh input, and so on.

The forgery is valid as soon as one of these h evaluations outputs the corresponding honest node, as all nodes above it are then recomputed honestly. Hence,

$$\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{SIG}}(\mathcal{A}) \approx \frac{h}{2^n}.$$

With $h = 32$ and $n = 128$, the advantage is roughly 2^{-123}.

Implications. The adversary makes $q_S = 1$ signing query and $q_{\mathsf{H}} = 0$ direct queries. Thus, in counting modes $Z \subseteq \{q_{\mathsf{H}}, q_S\}$, the adversary makes at most $q = 1$ many Z-queries, and the attack shows that both schemes have at most 123 bits of security for strong unforgeability. On the other hand, as soon as $q_{\mathsf{Ver}} \in Z$, the attack is no longer competitive. For instance, for SIG_{ts}, verifying the forgery costs $1 + (v(2^w - 1) - T) + 1 + h = 184 \approx 2^{7.5}$ queries.

The same idea works for any single value of the signature that enters the verifier's computation before the root. In particular, the adversary gets more chances by replacing a chain value σ_k by a random $y \neq \sigma_k$ instead of A_0: then, the remaining $d_k := 2^w - 1 - x_k$ chain steps on chain k and the leaf compression are evaluated on fresh inputs as well, and each of them may hit the honest value. This gives an advantage of roughly $(d_k + 1 + h)/2^n$. For SIG_{cs}, if $\mathcal{A}$ picks $k = 32$, i.e., the most significant checksum digit, which is 0 or 1, we have $\mathbb{E}[d_{32}] \approx 14.72$, and the advantage is roughly $47.7 \cdot 2^{-128} \approx 2^{-122.4}$. For SIG_{ts}, if $\mathcal{A}$ picks k uniformly at random (and thus makes no query to H), we have $\mathbb{E}[d_k] = (v(2^w - 1) - T)/v = 150/32$, and the advantage is roughly $37.7 \cdot 2^{-128} \approx 2^{-122.8}$. Both are strictly larger than 2^{-123}, and thus show at most 122 bits of security for $Z \subseteq \{q_{\mathsf{H}}, q_S\}$, matching Table 1.

6.3 Results for Unforgeability

We now turn to standard (i.e., non-strong) unforgeability, where the adversary needs to forge on a new message (cf. Definition 3).

6.3.1 Proven Bounds

Again, we first state the result and then explain attacks.

Theorem 4 (Bit Security for Unforgeability - Target Sum). *Let SIG_{ts} be the epoch-based signature scheme with target sum encoding from Section 5.2. Let $Z \subseteq \{q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\}$ with $q_{\mathsf{H}} \in Z$. Let λ be the maximum integer so that SIG_{ts} has λ bits of security in counting mode Z for unforgeability. Then:*

- *If $Z = \{q_{\mathsf{H}}\}$, then $\lambda = 126$.*
- *If $Z \cap \{q_S, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\} \neq \varnothing$, then $\lambda = 127$.*

Proof. The Lean4 code can be found at:

`Xmss/Statement/Theorems/Unforgeability.lean#L21-L23`

The attack showing $\lambda \leq 127$ is the generic one from Section 6.1, and the attack showing $\lambda \leq 126$ is presented in Section 6.3.2. □

Note that compared to strong unforgeability, we get a higher bit security in some cases. It is also clear that the attack showing at most 122 bits of security for strong unforgeability does not translate into an attack forging on a new message.

Theorem 5 (Bit Security for Unforgeability - Checksum). *Let SIG_{cs} be the epoch-based signature scheme with checksum encoding from Section 5.2. Let $Z \subseteq \{q_S, q_{\mathsf{H}}, q_{\mathsf{Gen}}, q_{\mathsf{Sig}}, q_{\mathsf{Ver}}\}$ with $q_{\mathsf{H}} \in Z$. Let λ be the maximum integer so that SIG_{cs} has λ bits of security in counting mode Z for unforgeability. Then:*

- *If $q_{\mathsf{Gen}} \in Z \vee q_{\mathsf{Ver}} \in Z$, then $\lambda = 127$.*
- *Else if $\{q_{\mathsf{Sig}}, q_S\} \subseteq Z$, then $\lambda = 124$.*
- *Else if $q_{\mathsf{Sig}} \in Z$, then $\lambda = 123$.*

- *Else,* $\lambda = 122$.

Proof. The Lean4 code can be found at:

Xmss/Statement/Theorems/Unforgeability.lean#L28-L32

The generic attack from Section 6.1 shows $\lambda \leq 127$. The blind forgery from Section 6.3.2 has advantage roughly $33 \cdot 2^{-128}$ and shows $\lambda \leq 122$. When counting the signer's 2 queries to H, i.e., q_{Sig}, its ratio is roughly $33/2 \cdot 2^{-128} \approx 2^{-123.96}$, which shows $\lambda \leq 123$. When counting both q_{Sig} and q_S, its ratio is roughly $33/3 \cdot 2^{-128} \approx 2^{-124.54}$, which shows $\lambda \leq 124$. □

6.3.2 Attacks

The attack from the previous section does not apply here, as it keeps the signed message. To forge on a new message m', the adversary must use the fresh encoding of (m', ρ'). We first show that for SIG_{cs}, i.e., XMSS with WOTS-TW, the idea of the previous section still works: the adversary outputs random chain values and relies on the verifier's hash evaluations. We then explain why this fails for SIG_{ts}, i.e., XMSS with WOTS$^+$C [HKRY23]. We use the same notation as in the previous section.

Blind Forgeries Against WOTS-TW. The adversary $\mathcal{A}$ proceeds as follows:

1. On input pk, query $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ for an arbitrary epoch ep and message m, and obtain $\sigma = (\rho, (\sigma_i)_{i\in[v]}, (A_\ell)_{\ell\in[h]})$.
2. Fix any message $\mathsf{m}' \neq \mathsf{m}$. Sample $\rho' \xleftarrow{\$} \{0,1\}^n$ and $y_i \xleftarrow{\$} \{0,1\}^n$ for all $i \in [v]$, and output the forgery
$$(\mathsf{ep}, \mathsf{m}', (\rho', (y_i)_{i\in[v]}, (A_\ell)_{\ell\in[h]})).$$

That is, $\mathcal{A}$ keeps only the authentication path of the honest signature and replaces everything else by random strings. We illustrate the attack in Figure 2.

Analysis. As $\mathsf{m}' \neq \mathsf{m}$, a valid forgery is a forgery for unforgeability (and thus also for strong unforgeability). For the checksum encoding, every message hash yields a valid encoding, so the verifier never rejects the encoding of (m', ρ'). It then computes the chain ends starting from the random values y_i. As these are random, the chain ends are wrong with overwhelming probability. However, the verifier still compresses them into a leaf, on an input that has (with overwhelming probability) not been queried before. As in the previous section, the forgery is valid as soon as this leaf compression outputs the honest leaf, or one of the subsequent h evaluations on the path to the root outputs the corresponding honest node. Hence,
$$\mathsf{Adv}^{\mathsf{eufcma}}_{\mathsf{SIG}_{cs}}(\mathcal{A}) \approx \frac{1+h}{2^n} = 33 \cdot 2^{-128} \approx 2^{-122.96}.$$
The adversary makes $q_S = 1$ signing query and no direct query to H. Thus, in counting modes $Z \subseteq \{q_{\mathsf{H}}, q_S\}$, it makes at most $q = 1$ many Z-queries, and SIG_{cs} has at most 122 bits of security for unforgeability, matching Table 1. Note that this is also an attack against strong unforgeability, and it is slightly better than the attack replacing A_0, as the leaf compression gives one more chance.

Why WOTS$^+$C Prevents Blind Forgeries. For SIG_{ts}, the verifier first checks that the encoding of (m', ρ') is valid, i.e., that the digits have the target sum T, and rejects otherwise, before performing any further hash evaluation. For uniformly random ρ', this happens only with probability $p_T \approx 2^{-14.80}$. This leaves the adversary with two options, both of which fail:

- The adversary outputs the blind forgery as above. Then it fails with probability $1 - p_T$ already at the encoding check, and its advantage is roughly $p_T \cdot (1+h) \cdot 2^{-n} \approx 2^{-137.8}$, while making $q = 1$ counted query.
- The adversary first grinds for a valid randomizer, i.e., it evaluates the encoding of (m', ρ') for fresh ρ' until it is valid, and then outputs the blind forgery with that ρ'. Then the forgery succeeds with probability roughly $(1+h) \cdot 2^{-n}$, but the adversary makes $1/p_T \approx 2^{14.80}$ *direct* queries to H in expectation, which are counted in every counting mode. Again, the ratio between advantage and queries is roughly $2^{-137.8}$.

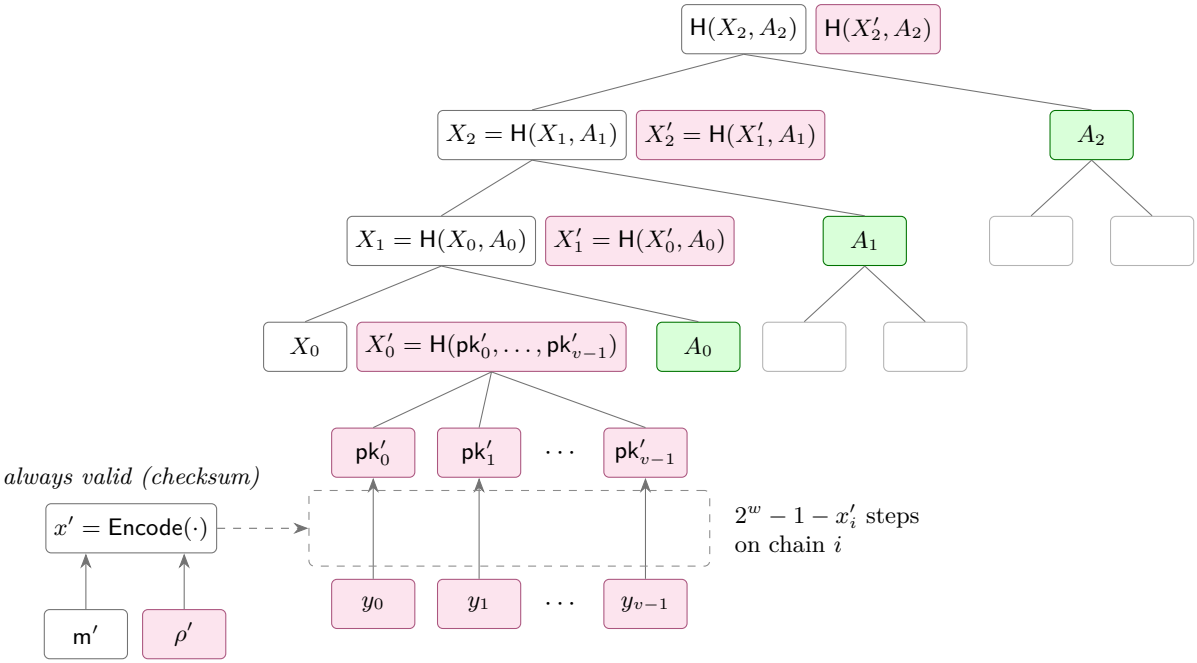


Figure 2: The blind forgery against SIG_{cs} for a tree of height $h = 3$. Green values are taken from the honest signature, i.e., only the authentication path (A_0, A_1, A_2). The adversary picks a new message m', a fresh randomizer ρ', and random chain values $(y_i)_{i\in[v]}$ (pink). As every digest is a valid checksum encoding, the verifier completes the chains to wrong chain ends $(\mathsf{pk}_i')_{i\in[v]}$ and computes the $1 + h$ fresh evaluations X_0', $X_1' = \mathsf{H}(X_0', A_0)$, $X_2' = \mathsf{H}(X_1', A_1)$, and $\mathsf{H}(X_2', A_2)$. The forgery is valid if one of them hits the corresponding honest node. For readability, we omit the public parameter and the tweaks.

In contrast to strong unforgeability, the adversary cannot reuse the encoding of the signed message, as it has to change the message. In other words, the encoding check forces the adversary to pay for the work that the signer spends on finding a valid encoding. Further, even after a valid ρ' is found, the encoding of (m', ρ') typically differs from the signed encoding in many chains, so the honest chain values do not help to make more chain ends correct.

We stress that what prevents the blind forgery is the *grinding*, i.e., that the signer has to search for a randomizer that maps into the set of valid encodings, and not the target sum property, or incomparability, as such. The only thing that matters is the probability that a fresh ρ' leads to a valid encoding, i.e., how *sparse* the valid encodings are among all digests. To see this, consider a variant of Encode_{ts} in which every digest is a valid encoding, e.g., because the digest is mapped onto the set of vectors in $[2^w]^v$ with digit sum T. This variant is still incomparable, as all of its outputs have the same sum. However, the verifier never rejects the encoding, and so the blind forgery succeeds with probability roughly $(1 + h) \cdot 2^{-n}$ using $q = 1$ counted query, exactly as for SIG_{cs}. Conversely, adding grinding on top of the checksum encoding, e.g., by requiring some bits of the digest to be 0, would protect SIG_{cs} against blind forgeries in the same way. Thus, the difference between SIG_{ts} and SIG_{cs} in Table 1 with respect to this attack is not due to the choice of incomparable encoding, but due to the fact that Encode_{ts} is instantiated via rejection sampling.

Replay Forgeries Against SIG_{ts}. For $Z = \{q_{\mathsf{H}}\}$, we sketch an attack showing that SIG_{ts} has at most 126 bits of security for unforgeability. The adversary gets a signature σ with randomizer ρ on m. Given two other messages $\mathsf{m}_1 \neq \mathsf{m}_2$, it then hashes (m_1, ρ). If the digest is a valid encoding, it submits σ for m_1, and otherwise for m_2. Each of the two tries (the adversary's for m_1, and, if needed, the verifier's for m_2) wins at least in the following two disjoint events: the encoding equals the signed encoding (probability 2^{-128}), or it is a different valid encoding (probability about p_T, see above) and the final root comparison succeeds by chance (probability 2^{-128}). Adding the probabilities of the two events, each try wins with probability about $2^{-128} + p_T \cdot 2^{-128} = (1 + p_T) \cdot 2^{-128}$. As the second try only happens if the first encoding

is invalid, i.e., with probability $1 - p_T$, the adversary wins with probability about

$$(1 + p_T) \cdot 2^{-128} + (1 - p_T)(1 + p_T) \cdot 2^{-128} = (2 + p_T - p_T^2) \cdot 2^{-128} > 2^{-127}.$$

7 Conclusion and Discussion

Let us discuss how our results apply to other schemes and what we can learn for autoresearch. In general, we view our work as pointing out a subtlety in the notion of bit security that designers of schemes and of autoresearch challenges should be aware of. We picked the XMSS schemes as our working example because they are easy to understand and practically relevant. For other hash-based signature schemes, e.g., other variants of XMSS, we think that the Lean4 code and the framework of definitions developed in this work can be the starting point for studying their exact bit security.

7.1 Application to SLH-DSA

SLH-DSA [Nat24] is a standardized stateless hash-based signature scheme: a few-time signature FORS [BHK+19] signs the message, and a hypertree of XMSS trees with WOTS-TW (as in SIG_{cs}) signs the FORS public key. Our XMSS analysis can for instance be adapted to SLH-DSA-128s and SLH-DSA-128f, whose parameters we recall in Table 2. Both use $n = 128$ bit digests and the same WOTS-TW instance as SIG_{cs}, i.e., $v = 35$ chains of length $2^w = 16$. The hypertree has total height 63 (resp. 66) and consists of $d = 7$ (resp. 22) layers of XMSS trees, each of height $h = 9$ (resp. 3). The message is signed by FORS with $k = 14$ (resp. 33) Merkle trees, each of height $a = 12$ (resp. 6), i.e., with 2^a leaves; a FORS signature reveals one leaf secret together with its authentication path of length a in each tree.

	SLH-DSA-128s	SLH-DSA-128f	SHRINCS
Digest length n (bits)	128	128	128
WOTS-TW chains v / chain length 2^w	35 / 16	35 / 16	35 / 16
Hypertree height $d \cdot h$	63	66	45
Layers d	7	22	5
XMSS tree height h	9	3	9
FORS trees k	14	33	10
FORS tree height a	12	6	13

Table 2: Parameters of SLH-DSA-128s and SLH-DSA-128f [Nat24], and of the stateless path of SHRINCS [CHK+26], in our notation.

Strong Unforgeability. The attack from Section 6.2.2 translates naturally to SLH-DSA: replacing one of the revealed FORS secret values by a random string makes the verifier evaluate H more than 64 times on the path to the root, where each attempt may hit the honest value, i.e., at most 121 bits of security for $Z \subseteq \{q_{\mathsf{H}}, q_S\}$.

Unforgeability. The generic attack from Section 6.1 carries over, with two changes. First, the bottom-layer WOTS-TW signs a FORS public key rather than a randomized message hash, so the adversary searches for a neighbor among FORS public keys of its own. It generates one FORS key pair, and obtains each new candidate by replacing the right child of the root of one FORS tree by a fresh random string, which costs 2 queries to H (the new root and the new FORS public key). Second, the final message must be signed by the forged WOTS-TW signature, so the adversary grinds the randomizer until the message hash selects that hypertree leaf and, in the modified FORS tree, a leaf below the left child of the root. This takes about 2^{64} (resp. 2^{67}) queries. With the same B and Q, the ratio between advantage and queries is about $2^{-127.5}$, i.e., at most 127 bits of security for unforgeability in every counting mode.

Proven Bound. For SLH-DSA-SHAKE-128s (SHAKE being modeled as a random oracle), we prove in Lean4 127 bits of strong unforgeability for $Z = \{q_{\mathsf{H}}, q_S, q_{\mathsf{Ver}}\}$, assuming a signature budget of 2^{64} (following [Nat24]): `SLH-DSA-SHAKE-128s/Statement/Theorems/StrongUnforgeability.lean#L10`. This result is tight given the generic attack above.

7.2 Application to SHRINCS

SHRINCS [CHK+26] is a candidate for Bitcoin post-quantum migration. It combines a stateful path with a stateless fallback under one public key, and a valid signature on either path is accepted. The stateless path is SLH-DSA with the parameters in Table 2 and a budget of 2^{40} signatures. The stateful path is a flexible XMSS tree (FXMSS) with WOTS$^+$C leaves, i.e., as in SIG_{ts} with $v = 32$ chains of length $2^w = 16$, but with target sum $T = 240$, the most likely sum, so that a fresh randomizer gives a valid encoding with probability $p_T \approx 2^{-6.04}$. Crucially, FXMSS lets the signer place leaves at any depth up to 255, and the recommended shape uses all of them: its q-th signature has an authentication path of length q.

Strong Unforgeability. For the stateless path, replacing a FORS secret value as for SLH-DSA gives $1 + a + 1 + d(1 + h) = 65$ chances, i.e., at most 121 bits of security for $Z \subseteq \{q_{\mathsf{H}}, q_S\}$. For the stateful path, the attack from Section 6.2.2 keeps the encoding and thus is not affected by grinding: replacing a chain value of a depth-255 signature gives about $7.5 + 1 + 255$ chances, i.e., at most 119 bits of security for $Z = \{q_{\mathsf{H}}\}$, where the 255 signing queries are free.

Unforgeability. The generic attack from Section 6.1 carries over to both paths (for the stateless path as for SLH-DSA, with about 2^{46} queries for grinding), i.e., at most 127 bits of security in every counting mode. For $Z = \{q_{\mathsf{H}}\}$, the 2^{40} signatures of the stateless path cover only about 3% of the 2^{45} positions in the hypertree, but every bottom-layer tree contains about 16 signed leaves. Using one query to learn the position of a new message, the adversary reuses the closest signed path with a random FORS signature, which gives about 45 chances, i.e., at most 122 bits of security. For the stateful path, without grinding, a depth-255 leaf gives 256 chances, i.e., at most 120 bits. Grinding reduces this, but only by a factor of about p_T: using one direct query, the adversary checks whether its new message m' with randomizer ρ'_1 gives a valid encoding, and otherwise outputs the forgery with a fresh ρ'_2, so that the verifier performs a second check. The forgery then succeeds with probability about $2p_T \cdot 256 \cdot 2^{-128} \approx 2^{-125.04}$ with $q = 1$, i.e., at most 125 bits for $Z = \{q_{\mathsf{H}}\}$.

7.3 Conclusion for Autoresearch

We were motivated by autoresearch challenges like *sig.golf* [Eth26], where the designers need to explicitly define a security requirement that submissions must satisfy. In the first version of *sig.golf*, the designers picked at least 127 bits of security, for a counting mode that counts all hash queries made by the experiment into the adversary's work. Our work shows that the definition of bit security is indeed very sensitive to the counting mode, which means that some counting modes would have ruled out more submissions than others.

Our suggestion: multiple modes. We recommend using a more fine-grained and complete requirement for assessing the security of a submission in an autoresearch challenge. For instance, instead of only picking one counting mode Z and a minimum security level λ_Z for this mode, one could require different levels for different counting modes. For instance, if Z is the most conservative counting mode $Z = \{q_{\mathsf{H}}\}$, one could require a slightly lower (say 120 bits) level for this mode, but also require 127 bits of security for other counting modes. The requirements should then all hold at the same time, only then would a submission be considered valid. One could even pick different λ_Z for strong and standard unforgeability. In the most "complete" setting, the designers could pick two thresholds λ_Z, λ'_Z for each mode Z, and require at least λ_Z (resp. λ'_Z) bits of security in counting mode Z for strong unforgeability (resp. unforgeability) for every Z. To pick the levels λ_Z, λ'_Z, one approach would be to take a reference scheme, e.g., one standardized by NIST, and compute the security levels for this scheme. Then every accepted submission would be at least as secure as this baseline, in every counting mode.

8 More on Related Work

Let us discuss some related work, in particular other works on hash-based signatures, and works on bit security.

Hash-Based Signatures. For a good introduction to the topic of hash-based signatures, we refer to the book by Boneh and Shoup [BS23]. Our work does not introduce any new construction of hash-based signatures, and we use XMSS as a working example to show subtleties in the notions of bit security. XMSS and the one-time signatures that are used in this context relate back to classical works on hash-based signatures [Mer88, BM94] as well as more recent ones [Hül13, BHK+19, HK22]. The XMSS construction that we look at uses the constant-sum encoding, which dates back to ideas by Bos and Chaum [BC93] and Vaudenay [Vau93]. Such efficient encodings have been studied extensively in the more recent past [CYK16, PZC+21, BHRvV21, HKRY23, ZCY23, KKW25].

Formal Verification of Hash-Based Signatures. We give formally verified security proofs for hash-based signatures in Lean4, using the VCVio framework to model oracle computation [TH24, TDW+26]. Closely related in this regard are the works [BDG+23] and [BDH+24], which verify the security of XMSS and SPHINCS+ using the EasyCrypt proof assistant [BGHZ11]. Both works formalize the modular proof known from the (non machine-checked) literature in EasyCrypt. More concretely, they show that the standard model properties assumed for the hash function imply security of the scheme, without formalizing any random oracle bounds for those notions.

In contrast, our Lean4 analysis is not modular: it does not reduce the scheme to properties of the hash function, but yields a concrete security level in the random oracle model. We believe that *both* approaches for proofs are extremely valuable for a scheme that is relevant for practice.

Concrete Security. Exploring the *concrete* (as opposed to asymptotic) security of constructions has a long history in cryptography, going back (at least) to the important work of Bellare and Rogaway [BR96], which resulted in an entire sub-area of cryptography focusing on tightness of reductions (e.g., see [Cor00, KMP16, BW24, BRTZ24, BHKR25] just to name a few). Bellare and Rogaway do not measure the concrete security in bits. Instead, they define a signature scheme to be (ϵ, t)-secure if any forger running in time t succeeds with probability at most ϵ. To make the random oracle and signing queries explicit, they also define (ϵ, q_S, q_H, t)-security. Interestingly, they make the convention that t includes the time that the signing oracle takes to produce signatures. Compared to our setting, the most important distinction is that we work in the random oracle model alone, without any computational hardness assumptions, while the line of work on tightness mostly considers reductions to computational assumptions. Bernstein and Lange demonstrate that the computational model has a direct impact on the concrete security [BL13], especially with respect to uniform vs. non-uniform adversaries.

Bit Security for Hash-Based Signatures. Within hash-based signatures, bit security claims are typically derived from concrete bounds in the (quantum) random oracle model. For XMSS, such an analysis was given by Hülsing, Rijneveld, and Song [HRS16], who study multi-target attacks and bound the success probability of generic attacks in terms of the number of hash queries. Buchmann, Dahmen, and Hülsing [BDH11] state the bit security of XMSS as $\log(t/\epsilon)$, where ϵ is the adversary's success probability and t its running time. This running time is measured as the number of evaluations of the underlying function families. This matches our definition in spirit, but they give no explicit formal definition, and it is left unspecified which evaluations count towards t. Barak and Mahmoody [BM07] give a lower bound for the number of queries the honest parties need to make in hash-based one-time signatures. Consequently, they need to carefully define what security means. Their definition – similar to ours – uses the ratio of work and success probability to model security, where they only count direct random oracle queries towards the work. In contrast to all of these works, we compare several notions of bit security for signature unforgeability that differ only in how we measure the adversary's resources, all of which simply count queries.

General Work on Bit Security. Measuring the security of a primitive by the ratio between the running time of an adversary and its success probability is a standard approach in cryptography. This

measure goes back to Levin; see, e.g., Goldreich [Gol00, Sec. 2]. More recently, several works have proposed general frameworks for bit security, notably Micciancio and Walter [MW18] and Watanabe and Yasunaga [WY21]. These works also discuss the nuances of defining bit security, in particular for primitives with *decisional* security notions such as pseudorandom generators. Subsequent works compare, refine, and apply these frameworks: Micciancio and Schultz-Wu [MSW24] show that, for decision games, the notion of Watanabe and Yasunaga coincides with that of Micciancio and Walter up to an additive constant, closing a gap in an earlier argument by Watanabe and Yasunaga [WY23]; Lee [Lee24] proposes a notion that treats search and decision primitives in a single framework while retaining a firm operational meaning; and Watanabe and Yasunaga [WY24] prove a variant of Yao's XOR lemma, stated in terms of the Rényi advantage, that avoids the $\log(1/\epsilon)$ bit-security loss incurred by the standard version. For search primitives, which include unforgeability of signatures, the frameworks agree with the traditional definition. Namely, for search games both notions reduce to $\log(T/\epsilon)$, where T is the adversary's work and ϵ its success probability: exactly for Micciancio and Walter, and up to a small additive constant for Watanabe and Yasunaga [WY21, Thms. 1–2]. In other words, these frameworks determine how work and success probability should be combined into a single number, but they take the work T as given. Our definitions are complementary: we fix how work and success probability are combined, and study what should count as work.

Statement on AI Usage

Authors worked extensively on the bit security and on hash-based signatures and are partially involved in designing autoresearch challenges. The authors identified that there is a subtlety in the definition of bit security. Some attacks were found by the authors while the Lean formalization and other attacks were done with significant help of AI. The authors cleaned up the Lean4 statements to make them well organized and readable, and verified that the statements in Lean4 match the statements in the document. The authors wrote the paper on their own, with AI assistance for regular grammar and consistency checks.

References

[BC93] Jurjen N. Bos and David Chaum. Provably unforgeable signatures. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 1–14. Springer, Berlin, Heidelberg, August 1993. doi:10.1007/3-540-48071-4_1. (Cited on Page 18.)

[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, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 421–454. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_14. (Cited on Page 18.)

[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, *PQCrypto 2011*, pages 117–129. Springer, Berlin, Heidelberg, November / December 2011. doi:10.1007/978-3-642-25405-5_8. (Cited on Page 1, 2, 3, 6, 18.)

[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, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 35–67. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0894-2_2. (Cited on Page 18.)

[BGHZ11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, Berlin, Heidelberg, August 2011. doi:10.1007/978-3-642-22792-9_5. (Cited on Page 18.)

[BHH+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 368–397. Springer, Berlin, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_15. (Cited on Page 1.)

[BHK+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2129–2146. ACM Press, November 2019. doi:10.1145/3319535.3363229. (Cited on Page 1, 16, 18.)

[BHKR25] Nicholas Brandt, Dennis Hofheinz, Michael Klooß, and Michael Reichle. Tightly-secure blind signatures in pairing-free groups. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part VI*, volume 16250 of *LNCS*, pages 337–369. Springer, Singapore, December 2025. doi:10.1007/978-981-95-5119-4_11. (Cited on Page 18.)

[BHRvV21] Joppe W. Bos, Andreas Hülsing, Joost Renes, and Christine van Vredendaal. Rapidly verifiable XMSS signatures. *IACR TCHES*, 2021(1):137–168, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8730, doi:10.46586/tches.v2021.i1.137-168. (Cited on Page 18.)

[BL13] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 321–340. Springer, Berlin, Heidelberg, December 2013. doi:10.1007/978-3-642-42045-0_17. (Cited on Page 18.)

[BM94] Daniel Bleichenbacher and Ueli M. Maurer. Directed acyclic graphs, one-way functions and digital signatures. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 75–82. Springer, Berlin, Heidelberg, August 1994. doi:10.1007/3-540-48658-5_9. (Cited on Page 18.)

[BM07] Boaz Barak and Mohammad Mahmoody-Ghidary. Lower bounds on signatures from symmetric primitives. In *48th FOCS*, pages 680–688. IEEE Computer Society Press, October 2007. doi:10.1109/FOCS.2007.71. (Cited on Page 18.)

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, Berlin, Heidelberg, May 1996. doi:10.1007/3-540-68339-9_34. (Cited on Page 18.)

[BRTZ24] Mihir Bellare, Doreen Riepel, Stefano Tessaro, and Yizhao Zhang. Count corruptions, not users: Improved tightness for signatures, encryption and authenticated key exchange. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part II*, volume 15485 of *LNCS*, pages 326–360. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0888-1_11. (Cited on Page 18.)

[BS23] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. Version 0.6, 2023. URL: https://toc.cryptobook.us/. (Cited on Page 18.)

[BW24] Renas Bacho and Benedikt Wagner. Tightly secure non-interactive BLS multi-signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part II*, volume 15485 of *LNCS*, pages 397–422. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0888-1_13. (Cited on Page 18.)

[CHK+26] Conduition, Ethan Heilman, Mikhail Kudinov, Oleksandr Kurbatov, Jonas Nick, and remix7531. SHRINCS: A compact hash-based signature scheme. Draft BIP, https://github.com/SHRINCS/shrincs-bip, 2026. Accessed 2026-10-16. (Cited on Page 1, 2, 16, 17.)

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Berlin, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_14. (Cited on Page 18.)

[CYK16] Jason Paul Cruz, Yoshio Yatani, and Yuichi Kaji. Constant-sum fingerprinting for Winternitz one-time signature. In *2016 International Symposium on Information Theory and Its Applications (ISITA)*, pages 703–707. IEEE, 2016. (Cited on Page 18.)

[DKKW25a] Justin Drake, Dmitry Khovratovich, Mikhail Kudinov, and Benedikt Wagner. Technical note: LeanSig for post-quantum Ethereum. Cryptology ePrint Archive, Paper 2025/1332, 2025. URL: https://eprint.iacr.org/2025/1332. (Cited on Page 1.)

[DKKW25b] Justin Drake, Dmitry Khovratovich, Mikhail A. Kudinov, and Benedikt Wagner. Hash-based multi-signatures for post-quantum Ethereum. *CiC*, 2(1):13, 2025. doi:10.62056/aey7qjp10. (Cited on Page 1, 11.)

[dMU21] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pages 625–635. Springer, 2021. doi:10.1007/978-3-030-79876-5_37. (Cited on Page 2.)

[Eth26] Ethereum Foundation, Yukon, Eigen Labs, Layr Labs. sig.golf: A competition for stateless hash-based signatures, 2026. URL: https://github.com/Layr-Labs/sig.golf-dev. (Cited on Page 1, 17.)

[Gol87] Oded Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 104–110. Springer, Berlin, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_8. (Cited on Page 1.)

[Gol00] Oded Goldreich. On security preserving reductions – revised terminology. Cryptology ePrint Archive, Report 2000/001, 2000. URL: https://eprint.iacr.org/2000/001. (Cited on Page 19.)

[HBG+18] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle signature scheme. Technical Report RFC 8391, RFC Editor, May 2018. URL: https://www.rfc-editor.org/rfc/rfc8391, doi:10.17487/RFC8391. (Cited on Page 1.)

[HK22] Andreas Hülsing and Mikhail A. Kudinov. Recovering the tight security proof of SPHINCS$^+$. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 3–33. Springer, Cham, December 2022. doi:10.1007/978-3-031-22972-5_1. (Cited on Page 6, 11, 18.)

[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. IEEE Computer Society Press, May 2023. doi:10.1109/SP46215.2023.10179381. (Cited on Page 1, 2, 6, 14, 18.)

[HRS16] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 387–416. Springer, Berlin, Heidelberg, March 2016. doi:10.1007/978-3-662-49384-7_15. (Cited on Page 18.)

[Hül13] Andreas Hülsing. W-OTS+ - shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13*, volume 7918 of *LNCS*, pages 173–188. Springer, Berlin, Heidelberg, June 2013. doi:10.1007/978-3-642-38553-7_10. (Cited on Page 18.)

[KKW25] Dmitry Khovratovich, Mikhail A. Kudinov, and Benedikt Wagner. At the top of the hypercube - better size-time tradeoffs for hash-based signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 93–123. Springer, Cham, August 2025. doi:10.1007/978-3-032-01887-8_4. (Cited on Page 18.)

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 33–61. Springer, Berlin, Heidelberg, August 2016. doi:10.1007/978-3-662-53008-5_2. (Cited on Page 18.)

[KN25] Mikhail Kudinov and Jonas Nick. Hash-based signature schemes for Bitcoin. Cryptology ePrint Archive, Paper 2025/2203, 2025. URL: https://eprint.iacr.org/2025/2203. (Cited on Page 1.)

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979. (Cited on Page 1.)

[Lee24] Keewoo Lee. Bit security as cost to demonstrate advantage. *CiC*, 1(1):1, 2024. doi:10.62056/an5txol7. (Cited on Page 19.)

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford University, 1979. (Cited on Page 1.)

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Berlin, Heidelberg, August 1988. doi:10.1007/3-540-48184-2_32. (Cited on Page 18.)

[MSW24] Daniele Micciancio and Mark Schultz-Wu. Bit security: Optimal adversaries, equivalence results, and a toolbox for computational-statistical security analysis. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part II*, volume 15365 of *LNCS*, pages 224–254. Springer, Cham, December 2024. doi:10.1007/978-3-031-78017-2_8. (Cited on Page 19.)

[MW18] Daniele Micciancio and Michael Walter. On the bit security of cryptographic primitives. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 3–28. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78381-9_1. (Cited on Page 19.)

[Nat24] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Technical Report FIPS PUB 205, National Institute of Standards and Technology, August 2024. https://doi.org/10.6028/NIST.FIPS.205. URL: https://doi.org/10.6028/NIST.FIPS.205. (Cited on Page 1, 2, 16, 17.)

[PZC+21] Lucas Pandolfo Perin, Gustavo Zambonin, Ricardo Felipe Custódio, Lucia Moura, and Daniel Panario. Improved constant-sum encodings for hash-based signatures. *Journal of Cryptographic Engineering*, 11(4):329–351, November 2021. doi:10.1007/s13389-021-00264-9. (Cited on Page 18.)

[TDW+26] Devon Tuma, Quang Dao, James Waters, Alexander Hicks, and Nicholas Hopper. VCVio: Verified cryptography in Lean via oracle effects and handlers. Cryptology ePrint Archive, Paper 2026/899, 2026. URL: https://eprint.iacr.org/2026/899. (Cited on Page 2, 18.)

[TH24] Devon Tuma and Nicholas Hopper. VCVio: A formally verified forking lemma and Fiat-Shamir transform, via a flexible and expressive oracle representation. Cryptology ePrint Archive, Report 2024/1819, 2024. URL: https://eprint.iacr.org/2024/1819. (Cited on Page 2, 18.)

[Vau93] Serge Vaudenay. One-time identification with low memory. In *Eurocode'92: International Symposium on Coding Theory and Applications*, pages 217–228. Springer, 1993. (Cited on Page 18.)

[WY21] Shun Watanabe and Kenji Yasunaga. Bit security as computational cost for winning games with high probability. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 161–188. Springer, Cham, December 2021. doi:10.1007/978-3-030-92078-4_6. (Cited on Page 19.)

[WY23] Shun Watanabe and Kenji Yasunaga. Unified view for notions of bit security. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 361–389. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8736-8_12. (Cited on Page 19.)

[WY24] Shun Watanabe and Kenji Yasunaga. Bit-security preserving hardness amplification. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part II*, volume 15365 of *LNCS*, pages 195–223. Springer, Cham, December 2024. doi:10.1007/978-3-031-78017-2_7. (Cited on Page 19.)

[ZCY23] Kaiyi Zhang, Hongrui Cui, and Yu Yu. Revisiting the constant-sum Winternitz one-time signature with applications to SPHINCS+ and XMSS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 455–483. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_15. (Cited on Page 18.)