IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 1, 59 pages.
https://doi.org/10.62056/aey7qjp10
Check for updates

# Hash-Based Multi-Signatures for Post-Quantum Ethereum

Justin Drake[1], Dmitry Khovratovich[1], Mikhail Kudinov[a,2] and Benedikt Wagner[1]

[1] Ethereum Foundation, Switzerland
[2] Eindhoven University of Technology, Netherlands

**Abstract.** With the threat posed by quantum computers on the horizon, systems like Ethereum must transition to cryptographic primitives resistant to quantum attacks. One of the most critical of these primitives is the non-interactive multi-signature scheme used in Ethereum's proof-of-stake consensus, currently implemented with BLS signatures. This primitive enables validators to independently sign blocks, with their signatures then publicly aggregated into a compact aggregate signature.

In this work, we introduce a family of hash-based signature schemes as post-quantum alternatives to BLS. We consider the folklore method of aggregating signatures via (hash-based) succinct arguments, and our work is focused on instantiating the underlying signature scheme. The proposed schemes are variants of the XMSS signature scheme, analyzed within a novel and unified framework. While being generic, this framework is designed to minimize security loss, facilitating efficient parameter selection. A key feature of our work is the avoidance of random oracles in the security proof. Instead, we define explicit standard model requirements for the underlying hash functions. This eliminates the paradox of simultaneously treating hash functions as random oracles and as explicit circuits for aggregation. Furthermore, this provides cryptanalysts with clearly defined targets for evaluating the security of hash functions. Finally, we provide recommendations for practical instantiations of hash functions and concrete parameter settings, supported by known and novel heuristic bounds on the standard model properties.

**Keywords:** Post-Quantum · Ethereum · Multi-Signatures · Hash-Based · Tweakable Hash · Poseidon · Succinct Arguments

## 1 Introduction

Given the looming threat posed by large-scale quantum computers, it is clear that major systems need to transition to post-quantum cryptography. For instance, if Ethereum[1] fails to update its signatures used for proof-of-stake to a post-quantum secure scheme in time, a quantum-capable adversary could exploit vulnerabilities, potentially causing damages worth billions of dollars. Even the perception of such a threat could undermine trust in the system, eroding user confidence and jeopardizing the integrity of their savings.

**Post-Quantum Signatures.** A wide range of cryptographic approaches have been explored to develop post-quantum secure signature schemes. Among these are signatures based on lattices [DLL+17, LDK+20, PFH+20], codes [Ste94, CFS01], isogenies [DKL+20,

E-mail: justin.drake@ethereum.org (Justin Drake), dmitry.khovratovich@ethereum.org (Dmitry Khovratovich), mishel.kudinov@gmail.com (Mikhail Kudinov), benedikt.wagner@ethereum.org (Benedikt Wagner)

[a] Mikhail Kudinov was supported by an NWO VIDI grant (Project No. VI.Vidi.193.066).

[1] see https://ethereum.github.io/yellowpaper/paper.pdf.


Received: 2025-01-14 Accepted: 2025-03-11

DLRW24, SEMR24], multivariate systems of equations [Beu22], or hash functions [BDH11, BHH+15, BHK+19]. Hash-based signatures, in particular, are appealing for multiple reasons: minimal assumptions, ease of implementation, conceptual simplicity[2], and no use of complex algebra. In this work, we focus on hash-based signatures as a promising candidate for Ethereum's proof-of-stake.

**Advanced Signatures.** Despite the advantages mentioned above, hash-based signatures have a significant drawback stemming from their lack of algebraic structure. Namely, they typically are not amenable for turning them into advanced signature variants, such as multi-signatures, threshold signatures, or aggregate signatures. For instance, consider again proof-of-stake in Ethereum, which relies on a non-interactive multi-signature scheme: validators cast votes for blocks by signing them, and these individual signatures are aggregated into a single compact signature stored in the accepted block[3]. Hash-based signatures do not natively support such aggregation features, posing a challenge for their direct application in this context.

**Aggregating with Succinct Arguments.** A potential method for aggregating multiple signatures involves the use of a succinct argument of knowledge – an argument system where the argument is significantly smaller than the underlying witness. To aggregate signatures, one can compute a succinct argument demonstrating knowledge of all individual valid signatures, with the list of signatures serving as the witness. If succinct argument systems based on hash functions are employed, the resulting multi-signature scheme can be plausibly post-quantum secure [CMS19]. Throughout this work, we will call such argument systems *pqSNARKs*. Using pqSNARKs to aggregate hash-based signatures is an elegant and modular approach that can directly take advantage of recent improvements on hash-based succinct arguments, e.g., [HLP24, ZCF24, ACFY24a]. However, as we explain next, this approach also introduces several unique challenges in the design of the signature scheme.

**Random Oracle Paradox.** If the signature scheme's verifier relies on random oracles, a paradox arises when using pqSNARKs for aggregation: in the security proof, the hash function is modeled as a random oracle, yet it is *simultaneously* treated as an *explicit circuit* to be verified within the pqSNARK. Ignoring this discrepancy has unclear security implications, relying on a non-standard heuristic.

**A Cleaner Approach.** To circumvent this paradoxical situation, it is critical to design the scheme so that the verifier's circuit avoids invoking any random oracle. Instead, we aim to prove security of the underlying signature scheme assuming precisely stated *standard model* properties of the hash functions employed, such as variants of preimage resistance or collision resistance. The random oracle model may still be used to build heuristic confidence in the plausibility of these properties in isolation. However, the security of the scheme fundamentally rests on a well-defined set of standard model assumptions about the hash functions. This provides cryptanalysts with concrete targets to analyze. In addition, it should be the goal to provide security proofs that are as tight as possible. Tighter proofs reduce the need to compensate for security losses with overly large parameters, resulting in improved efficiency.

**Efficiency Criteria.** To efficiently utilize pqSNARKs for aggregation, the underlying hash-based signature scheme should satisfy the following properties:

- *Minimal Hashing.* Since the verification process for hash-based signature schemes is typically dominated by hash function evaluations, the efficiency of aggregation is heavily influenced by the amount of data that needs to be hashed. Reducing the

[2] For example, a proof-of-stake setting can benefit from simplicity, as it may enable *formal verification* of the verifier implementation.

[3] To be more precise, Ethereum consensus blocks today contain multiple aggregate signatures, each representing a large number of individual signatures.

amount of hashing required for verification is thus critical for optimizing aggregation performance.

- *Small Signatures.* The scheme should come with concretely small signatures to minimize bandwidth consumption of aggregators. For instance, assuming signatures of size 32 KiB and that Ethereum uses a four-second slot, where one second is allocated for aggregators to receive signatures[4], a committee of, say, $2^{12}$ signers would require $2^{30}$ bits to be received within that second, demanding a bandwidth of at least 1 GiB/s, which is infeasible as a requirement. Signatures of size, say, below 4 KiB would significantly soften this requirement or allow for larger committees.

## 1.1 Our Work

In this work, we present and analyze hash-based non-interactive multi-signature schemes suitable for post-quantum proof-of-stake. To this end, we extensively study hash-based signature schemes meeting the criteria above. Below, we briefly summarize our technical contributions.

**Overall Paradigm.** We consider the classical approach of turning one-time signatures[5], such as Winternitz signatures, into many-time signatures, which originates in Merkle's PhD thesis [Mer79] and is used in XMSS [BDH11]. The idea is as follows: the signer uses a Merkle tree to commit to a long sequence of one-time public keys, and the Merkle root serves as the (many-time) public key. To sign the $i$th message, the signer signs the message with the $i$th one-time secret key and also includes the one-time public key and a Merkle path in the signature. Note that this yields a *synchronized* (sometimes called stateful) signature scheme [GR06], where signing and verification are tied to specific epochs, with at most one signature per epoch. Such schemes are well-suited for applications like proof-of-stake, as noted[6] in prior works [FSZ22, FHSZ23]. Building on this, we transform the synchronized signature scheme into a non-interactive multi-signature scheme by aggregating signatures using succinct arguments. We formally prove the security of this folklore transformation under the assumption of *adaptive* knowledge soundness of the succinct argument system. The main focus of the paper, however, is the underlying signature scheme.

**Unified Analysis Framework.** Although the security of the XMSS paradigm can be generically reduced to the one-time security of the underlying one-time signature, such an analysis tends to be overly loose, leading to inefficient concrete parameters. On the other hand, performing detailed security analyses for each variant of XMSS individually would be too labor-intensive. To address this, we introduce a generalized framework for XMSS based on a novel primitive we term *incomparable encodings*. This abstraction allows us to unify and streamline the analysis of XMSS-like schemes. Subsequently, we instantiate incomparable encodings in multiple ways, yielding a set of schemes. Crucially, we carefully designed this abstraction to enable achieving the most efficient parameters possible. Our framework extends similar existing frameworks [BS20, ZCY23] to enable more instantiations, e.g., by allowing randomized encodings and encoding errors.

**Analysis in the Standard Model.** In our framework, we show that (strong) unforgeability follows from a set of simple standard-model assumptions on the underlying hash function. Notably, previous analyses do not directly apply to our setting. For instance, the tightest security bounds for XMSS rely on modeling message hashing as a reprogrammable random oracle [HK22, HKRY23].

[4] During each slot, a block must be proposed, distributed to validators, signed, signatures have to be aggregated, and the aggregate signature propagated.

[5] A one-time signature remains secure if at most one honestly generated signature is exposed to the adversary.

[6] Note that in proof-of-stake, validators sign one block per epoch, and signing twice per epoch is considered malicious behavior and punished.

**Instantiation, Parameter Requirements, and New Bounds.** To complete the picture and get a concrete proposal for Ethereum, we discuss how to instantiate the hash functions using either SHA-3 (the conservative option) or Poseidon2 (the modern alternative). We also show how to select appropriate output lengths and other parameters to achieve a certain security level. To accomplish this, we leverage existing heuristic bounds and derive new ones, using the (quantum and classical) random oracle model. We conclude with a discussion about the efficiency of our proposed instantiations.

### 1.2 Outline

We structure this paper as follows. In section 2, we summarize the relevant related work and explain how our work compares to it. We also discuss which other approaches may be suitable candidates for post-quantum proof-of-stake. In section 3, we introduce the relevant technical background, including definitions for tweakable hash functions, signatures, and non-interactive multi-signatures. We present and analyze a generalized variant of XMSS signatures and multi-signatures in section 4. For that, we introduce a new abstraction that we call incomparable encodings. We then instantiate these encodings in section 5, leading to several variants of XMSS. In sections 6 and 7, we explain how to set concrete parameters, e.g., output lengths of hash functions for a desired security level, and how to implement tweakable hash functions. We give benchmarks in section 8 and conclude in section 9.

## 2 Related Work and Alternative Approaches

Before going into the technical details of our work, we discuss how our work compares to previous works on hash-based signatures. We also discuss other post-quantum aggregate- and multi-signatures, e.g., from lattices, and assess whether they are suited for a large scale proof-of-stake setting.

### 2.1 Aggregation using Succinct Arguments

The idea of using generic succinct arguments to aggregate signatures is somewhat folklore and not new to our work. For instance, [ACL⁺22] introduces lattice-based succinct arguments and informally mentions the potential application of aggregating GPV signatures [GPV08]. This idea has also received increased formal attention within the context of batch arguments, as we explain next.

**Batch Arguments for NP.** In a (non-interactive) batch argument, we consider a prover and a verifier holding $n$ public statements $\mathsf{stmt}_1, \ldots, \mathsf{stmt}_n$, and the prover additionally holding the respective witnesses $\mathsf{witn}_1, \ldots, \mathsf{witn}_n$, where $(\mathsf{stmt}_i, \mathsf{witn}_i) \in \Gamma$ for some relation $\Gamma$. The goal is for the prover to succinctly convince the verifier of knowledge of all valid witnesses via a publicly verifiable argument string. Importantly, the argument size should be significantly smaller than the combined size of all witnesses. This framework is particularly well-suited for applications like signature aggregation, where the witnesses $\mathsf{witn}_i$ correspond to signatures and the statements $\mathsf{stmt}_i$ to public keys. Batch arguments can also be viewed as specialized succinct non-interactive arguments of knowledge (SNARKs) tailored for highly structured relations derived from $\Gamma$. The key advantage of batch arguments over generic SNARKs lies in avoiding the use of non-falsifiable assumptions, which are typically required for general SNARK constructions [GW11]. Achieving this efficiency from falsifiable assumptions requires a weaker form of the proof of knowledge property, called *somewhere* extraction. Intuitively, it requires that one can set up the common reference string with respect to an index $i^*$ (without revealing $i^*$), such that an extractor can later extract the witness $\mathsf{witn}_{i^*}$.

Waters and Wu [WW22], followed by the work of Devadas et al. [DGKV22], have constructed batch arguments for NP, and aggregate signatures in the standard model as an application. Their results established that somewhere extraction suffices to construct aggregate signatures. Notably, [WW22] relies on pairing-friendly groups and is therefore not post-quantum secure. Conversely, [DGKV22] is based on lattices and allows multi-hop aggregation (i.e., aggregating aggregates). Unfortunately, it makes use of heavy cryptographic machinery and is therefore far from being a candidate for practical deployment. Turning it into a concretely efficient and practical scheme is an interesting direction for future work. Recent advancements include more expressive policies. For instance, in monotone-policy batch arguments [BBK+23], the prover can show that *enough* of the statements hold. A promising application of this are monotone-policy aggregate signatures [BCJP24]. Here, one can publicly derive a succinct verification key that combines all individual public keys and we can still verify that a large enough subset signed a message, which is what we ultimately want in the proof-of-stake setting. In contrast, using non-interactive multi-signatures we also need to publish a bit vector that indicates who signed along with the aggregate signature. While this is a great improvement in terms of (asymptotic) efficiency, it comes at the cost of losing accountability. It is also not clear how these relatively novel constructions perform in practice.

**Aggregating Hash-Based Signatures with SNARKs.** The work most closely related to ours is the recent study by Khaburzaniya et al. [KCLM22], which employs pqSNARKs to construct hash-based aggregate and threshold signatures. At a high level, their goals align closely with ours, as both approaches non-interactively aggregate hash-based signatures using pqSNARKs. Despite these similarities, we view our contributions as complementary rather than overlapping. Khaburzaniya et al. focus primarily on optimizing the arithmetization (specifically, the Algebraic Intermediate Representation, AIR) of the verifier's circuit for use in a pqSNARK. In contrast, our focus are the underlying signature schemes themselves. We assume a generic pqSNARK framework and delve into concrete security and rigorous security proofs, explicitly stating standard model assumptions that the hash functions need to satisfy, as well as the exact security properties that are required for the pqSNARK. Additionally, our work explores trade-offs between hashing operations and signature size, offering a broader analysis of the design space. A significant difference between our approaches is the type of signatures being aggregated. Khaburzaniya et al. aggregate only one-time signatures, whereas our work covers aggregating synchronized many-time signatures. Furthermore, their underlying one-time signature scheme is Winternitz with one-bit chunks. This choice minimizes the number of hash invocations, but it results in a substantial individual signature size of approximately 8 KiB. The authors argue that individual signature size is less critical than the computational cost of hash operations. In contrast, our analysis simultaneously considers a variety of trade-offs between hashing and signature size. We emphasize that individual signature size plays a crucial role in settings with a lot of signers, especially in reducing bandwidth requirements for the aggregating party. Another key distinction lies in the level of rigor. Khaburzaniya et al. provide convincing proof sketches and intuitive arguments but do not present formal security definitions or analyses. In contrast, we prioritize concrete security and robust formal definitions, ensuring our scheme meets strong security guarantees and parameters can be set in a theoretically sound way. This level of rigor is essential for schemes intended for deployment in major blockchains like Ethereum. Clearly stating assumptions about the underlying hash functions is particularly important when relying on newer hash functions such as Poseidon. Finally, combining their advancements in verifier circuit optimization with our in-depth study of signature schemes may be a promising direction for future research.

## 2.2 Hash-Based Signatures

In this section, we discuss related hash-based constructions, highlight the challenges of reusing parts of their analysis, and explain how our results and analysis differ.

**SPHINCS, XMSS, and One-Time Signatures.** The schemes most relevant and closely related to our work are SPHINCS$^+$ [HBD$^+$22], SPHINCS+C [HKRY23], XMSS [BDH11, HBG$^+$18], and rapidly verifiable XMSS [BHRvV21]. One of our key observations is that, in the proof-of-stake setting where validators sign only once per slot[7], a *synchronized* scheme suffices. This eliminates the need for the additional complexity inherent in SPHINCS$^+$ and SPHINCS+C. Instead, we can adopt a much simpler XMSS-like structure, which enables significantly more efficient aggregation using succinct arguments.

**Hash-Efficient Variants.** As outlined in the introduction, one of our primary goals is to design a scheme with a minimal number of hashes required for verification. Both SPHINCS+C and rapidly verifiable XMSS address reduced verification time by focusing on lowering the verifier's hash complexity. Specifically, the SPHINCS+C paper introduces a variant of Winternitz one-time signatures that eliminates the checksum, also discussed and applied to XMSS in [ZCY23]. We adopt this idea in our target sum Winternitz instantiation within our generalized XMSS framework. In contrast, rapidly verifiable XMSS retains the checksum but probabilistically reduces the number of verification hashes for honest signers. However, this reduction is not mandatory, as signatures remain verifiable even if the signer uses plain Winternitz, unless additional complex checks are imposed. Given this, we favor the simpler approach inspired by SPHINCS+C.

In [ZCY23], the authors have analyzed the constant-sum encoding approach and showed that it achieves the optimal encoding rate when the chain elements sum to half of the maximally allowable value. While this configuration offers the best encoding rate, we also focus on reducing verification time, particularly by minimizing the number of required hash computations, even at the cost of slightly increased signing time. To this end, we allow for the flexibility to select a target value larger than half of the maximally allowable value. As discussed earlier, our work provides a security proof for a generalized XMSS framework that works with any incomparable encoding scheme.

**Generic Yet Tight Analysis without Random Oracles.** Integrating the SPHINCS+C methodology with existing XMSS analyses presents a key challenge to us: achieving the tightest possible security reduction without relying on random oracles. Addressing this requires adapting existing security proofs and combining techniques from earlier works. Although neither SPHINCS$^+$ nor SPHINCS+C rely on random oracles to prove security [HK22, HKRY23], we cannot directly reuse their proofs. In both schemes, the integrated XMSS structure is proven secure only under the weaker notion of *known message attacks*. While this suffices when XMSS is used within SPHINCS$^+$, it falls short when XMSS operates independently. In contrast, the security proofs for XMSS [BDH11, BHRvV21] rely on modeling message hashing as a reprogrammable random oracle. This effectively reduces security against chosen message attacks to security against known message attacks. To address this, we combine elements of earlier proofs (e.g., [Hül13, KKF21]) with modern techniques to achieve a tighter security reduction without relying on random oracles. Furthermore, we ensure our analysis is sufficiently general to support multiple instantiations. This is accomplished by introducing a generalized XMSS framework. We compare this with similar existing concepts in remark 4.

**Strong Unforgeability.** An additional novelty of our work is that we analyze *strong* unforgeability security of our generalized XMSS scheme. In contrast, all other proofs (to our knowledge) only focus on existential unforgeability. Proving the strongest possible security notion is important when a scheme is meant to be used in a complex system like Ethereum. We have found a work [BDE$^+$11] that analyzes Winternitz one-time signatures

[7] Throughout the paper, we will use the terms *slot* and *epoch* interchangeably.

with regards to strong unforgeability. However, the result is given for a less efficient variant of Winternitz scheme and can not be transferred directly to modern versions of Winternitz. So we could not use that in our proofs.

**Other Hash-Based Constructions.** A recent paper by Atapoor et al. [AdSGK24] briefly mentions aggregating hash-based signatures using succinct arguments. The main contribution of their work is to propose a hash-based signature scheme where the public key is derived from the secret key via a one-way function, and the signature consists of a succinct *zero-knowledge* proof of knowledge of the secret key, tagged with the message. This approach employs succinct arguments for individual signatures, requiring argument recursion and thus proofs about random oracle relations for aggregation. By contrast, our approach is significantly simpler, avoiding such recursive arguments.

## 2.3 Other Post-Quantum Aggregate and Multi-Signatures

While hash-based signatures are appealing, as already explained in the introduction, we still want to discuss multi-signatures based on other post-quantum assumptions such as lattices or isogenies. We identify a few examples of lattice-based constructions that warrant further investigation as alternatives to our hash-based proposal. However, parameter selection for deploying lattice-based constructions is notably more error-prone compared to purely hash-based approaches.

**Fiat-Shamir and Friends.** Using any signature scheme based on the Fiat-Shamir heuristic [FS87] in combination with a succinct argument would cause the paradoxical situation mentioned in the introduction. Namely, one would, at the same time, treat hash functions as random oracles and as explicit circuits. In particular, this applies to Dilithium [DLL+17, LDK+20] and to MPC-in-the-head and VOLE-in-the-head signatures such as FAEST [BBd+23] or Biscuit [BKPV23]. It also holds for Falcon signatures [PFH+20] if random seeds are used. A variant of Falcon without random seeds in combination with a pqSNARK would be a reasonable route for further exploration, because in this case the random oracle can be evaluated on the message outside of the circuit that is proven.

**Non-Interactive Constructions.** Boneh and Kim [BK20] have proposed two lattice-based constructions: one enables non-interactive aggregation of Lyubashevsky and Micciancio's one-time signatures [LM08], while the other supports many-time signatures but requires interaction. The MMSAT scheme [DHSS20] achieves asymptotically linear-size aggregate signatures, with size $O(\log k) + 2nk$ for $k$ signers and security parameter $n$. For moderate values of $k$ (e.g., $k = 1000$), these signatures can be significantly smaller than pqSNARKs. The scheme is based on a somewhat exotic lattice assumption called *Vandermonde-SIS*.

Another line of work constructs lattice-based non-interactive multi-signatures in a synchronized setting [FSZ22, FHSZ23]. The authors explain that the synchronized setting is well-suited for a proof-of-stake application and we follow this observation. Their approach, akin to lattice-based XMSS, uses a homomorphic Merkle tree for aggregation. However, individual signatures exceed 32 KiB, making them impractical for our setting, as noted in the introduction.

A promising approach involves aggregating Falcon signatures [PFH+20] using the lattice-based proof system LaBRADOR [BS23], as explored in recent works [TS23, AAB+24]. This method achieves compact individual and aggregate signatures. However, its security proofs rely on rewinding, which has unclear implications in the post-quantum setting [LMQW22].

We suspect that recent lattice-based folding schemes [BC24, FKNP24] are a good starting point for further research on constructing lattice-based non-interactive signature aggregation.

**Interactive Constructions.** In addition to the non-interactive constructions mentioned above, a number of post-quantum multi-signature schemes employ interactive signing protocols [DOTT21, Che23, LLL+24, ADP24, DFMS24]. These protocols require multiple rounds of interaction, introducing additional latency as well as operational overhead related to maintenance and scheduling, particularly in asynchronous networks. By contrast, non-interactive protocols that support public aggregation of individual signatures mitigate these challenges. They enable multiple redundant aggregators to independently collect and merge signatures within predefined time frames, simplifying coordination and reducing complexity.

A middle ground between non-interactive and interactive signing is achieved when a single message-independent preprocessing round is required per signature. Once this preprocessing is completed, the message can be signed in a non-interactive, publicly aggregatable manner. In such schemes, the preprocessing phase can occur well in advance of the time-critical path, allowing the scheme to function like a non-interactive protocol once the message to be signed becomes available. We are aware of one example of such a scheme in the lattice setting [BTT22]. However, it suffers from significant communication complexity, especially in the preprocessing round, requiring a per-signer outgoing broadcast size of 3500 KiB for 1024 signers, as reported in [Che23].

# 3 Preliminaries

As common, we use $\mathbb{N}, \mathbb{R}$ to denote the natural and real numbers, respectively. We use the notation $[L] := \{1, \ldots, L\} \subseteq \mathbb{N}$ to denote the first $L$ natural numbers. We use the notation $s \xleftarrow{\$} S$ to state that $s$ is sampled uniformly at random from $S$, where $S$ is a finite set. For a distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ means that $x$ is sampled from $\mathcal{D}$. Let $\mathcal{D}_1, \mathcal{D}_2$ be distributions on the same support $\mathcal{X}$. Then, their statistical distance is defined as $\frac{1}{2}\sum_{x\in\mathcal{X}} |\Pr[\mathcal{D}_1 = x] - \Pr[\mathcal{D}_2 = x]|$. We often write $\Pr_{\mathbf{G}}[E]$ or $\Pr[E \mid \mathbf{G}]$ to denote the probability that some event $E$ occurs in the experiment $\mathbf{G}$. We denote the event that an experiment $\mathbf{G}$ outputs a bit $b$ by $\mathbf{G} \Rightarrow b$. For a probabilistic algorithm $\mathcal{A}$, we write $y := \mathcal{A}(x; \rho)$ to denote that $\mathcal{A}$ outputs $y$ on input $x$ with random coins $\rho$, and $y \leftarrow \mathcal{A}(x)$ if $\rho$ is sampled uniformly at random from the algorithms randomness space. We use the notation $y \in \mathcal{A}(x)$ to denote that there are random coins $\rho$ such that $\mathcal{A}$ outputs $y$ on input $x$ with these coins $\rho$. We denote the running time of an algorithm $\mathcal{A}$ by $\mathbf{T}(\mathcal{A})$. We often require algorithms to be *efficient*, which is not a formally well-specified term, as we are not working in the realm of asymptotic security. However, we assume the reader to have an intuitive understanding of what it means, and it means at least that the running time is a polynomial in its input size. We assume that all algorithms and adversaries have (implicit) access to a set of public system parameters $\mathsf{par}$. Unless specified otherwise, all oracles that algorithms obtain should be understood as classical oracles, i.e., algorithms have classical access to these oracles. In all experiments and security games, we implicitly initialize numerical variables with 0, and lists, maps, and sets as empty. We say that a function $F$ is *efficiently computable* if there is an efficient algorithm that computes $F$.

## 3.1 Tweakable Hash Functions

In [BHK+19], the notion of tweakable hash functions has been introduced. The idea is to unify the description of the way hashing is done in different hash-based signatures. This abstraction is similar to the definition of keyed hash functions, although tweakable hash functions have three inputs instead of two. The first is called a public parameter $P \in \mathcal{P}$ and is usually meant to be random, and the same for all the hash function calls related to a user in a hash-based signature. The second input is a tweak $T \in \mathcal{T}$. A tweak is a deterministic value for domain separation that distinguishes different hash function calls in the scheme. One way to think of it is as a unique identifier or an address of a hash function call. The last input is the message that we want to hash.

**Definition 1** (Tweakable Hash Function)**.** Consider sets $\mathcal{H}$ (the hash space), $\mathcal{P}$ (the public parameters space), $\mathcal{T}$ (the tweak space), and let $\mathcal{M}$ (the message space). A tweakable hash function is an efficiently computable function

$$\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}.$$

The first property we define for tweakable hash functions, called *$\epsilon$-uniformity*, is statistical in nature and serves to bound the efficiency of the signature construction. Specifically, we require that the outputs of tweakable hash functions are distributed close to uniformly, assuming parts of the input message are sampled at random. We state this property as a worst-case condition over all parameters (and tweaks and messages). However, for practical hash functions, this property might only hold for most parameters, with exceptions being rare and hard to find. It is important to note that we utilize this property solely to establish a theoretical bound on correctness in section 5.2. This, in turn, impacts the number of retries a signer might need, making it a factor of efficiency. The number of retries will in practice be chosen based on experiments, and can even differ from signer to signer. We could have opted for a cleaner approach involving an adversarial correctness notion throughout the paper, but this would result in a significantly less readable presentation.

**Definition 2** (Uniformity)**.** Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ be a tweakable hash function, where $\mathcal{M} = \mathcal{M}_0 \times \mathcal{R}$. We say that $\mathsf{Th}$ is $\epsilon$-uniform for seed space $\mathcal{R}$ if for all $P \in \mathcal{P}$, all $T \in \mathcal{T}$, and all $\mathsf{m} \in \mathcal{M}_0$, the following two distributions have statistical distance at most $\epsilon$:

$$\{x \mid x \xleftarrow{\$} \mathcal{H}\} \text{ and } \{\mathsf{Th}(P, T, (\mathsf{m}, \rho)) \mid \rho \xleftarrow{\$} \mathcal{R}\}.$$

To prove the security of hash-based signatures we will rely on certain security properties of tweakable hash functions. Concretely, we will use established properties that have also been used to prove the security of SPHINCS$^+$ in [HK22]:

- *single-function, multi-target collision resistance for distinct tweaks*;
- *single-function, multi-target preimage resistance for distinct tweaks*;
- *single-function, multi-target undetectability for distinct tweaks*.

All three notions allow the adversary to specify the tweaks used in challenges, but the adversary must not reuse a tweak. The first notion we define is single-function, multi-target collision resistance (for distinct tweaks). Here, the adversary first gets access to an oracle that evaluates the tweakable hash function for a random public parameter $P$ not known to the adversary. In the second stage, the adversary learns this parameter $P$ and is supposed to find a collision to one of the images that it obtained before.

**Definition 3** (Multi-Target Collision Resistance)**.** Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ be a tweakable hash function as defined in definition 1. Let $\mathcal{A}$ be a (stateful) algorithm, and $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}TCR}_{\mathsf{Th},p}(\mathcal{A})$:

1. Generate a random public parameter $P \xleftarrow{\$} \mathcal{P}$.
2. Run $\mathcal{A}$ with (classical) access to an oracle that takes $T \in \mathcal{T}$ and $M \in \mathcal{M}$ and works as follows:
    - If $|Q| \geq p$ or there is an $M' \in \mathcal{M}$ with $(T, M') \in Q$, return $\perp$.

- Otherwise, insert $(T, M)$ into the list $Q$ and output $\mathsf{Th}(P, T, M)$.

3. When $\mathcal{A}$ signals to continue, then continue running $\mathcal{A}$ with input $P$, but without the oracle access.

4. Obtain from $\mathcal{A}$ an output $(j, M)$ with $M \in \mathcal{M}, j \in [|Q|]$. Denote the $j$th entry in $Q$ by $(T_j, M_j)$.

5. Output 1 if $\mathsf{Th}(P, T_j, M_j) = \mathsf{Th}(P, T_j, M)$ and $M \neq M_j$. Otherwise, output 0.

For any such algorithm $\mathcal{A}$, we define the following success probability:

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

The second security notion that we need for is a form of preimage resistance, namely, single-function, multi-target preimage resistance. We give a general definition here but we will use it only for a single target. In this notion, the adversary again gets an oracle, but this oracle now chooses the message randomly, and the goal of the adversary in the second stage is to find any preimage.

**Definition 4** (Multi-Target Preimage Resistance)**.** Let $\mathsf{Th}\colon \mathcal{P}\times\mathcal{T}\times\mathcal{M} \to \mathcal{H}$ be a tweakable hash function as defined in definition 1. Let $\mathcal{A}$ be a (stateful) algorithm, $\mathcal{M}' \subseteq \mathcal{M}$, and $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}PRE}_{\mathsf{Th},\mathcal{M}',p,}(\mathcal{A})$:

1. Generate a random public parameter $P \xleftarrow{\$} \mathcal{P}$.

2. Run $\mathcal{A}$ with (classical) access to an oracle that takes $T \in \mathcal{T}$ and works as follows:

   - If $|Q| \geq p$ or there is an $x' \in \mathcal{M}'$ with $(T, x') \in Q$, return $\perp$.
   - Otherwise, sample $x \xleftarrow{\$} \mathcal{M}'$, insert $(T, x)$ into the list $Q$ and output $\mathsf{Th}(P, T, x)$.

3. When $\mathcal{A}$ signals to continue, then continue running $\mathcal{A}$ with input $P$, but without the oracle access.

4. Obtain from $\mathcal{A}$ an output $(j, M)$ with $M \in \mathcal{M}', j \in [|Q|]$. Denote the $j$th entry in $Q$ by $(T_j, x_j)$.

5. Output 1 if $\mathsf{Th}(P, T_j, M) = \mathsf{Th}(P, T_j, x_j)$. Otherwise, output 0.

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

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

The third notion we consider is undetectability. Intuitively, undetectability states that the hash function output is indistinguishable from random. As with preimage resistance we will only utilize a single-target version of the following notion.

**Definition 5** (Multi-Target Undetectability)**.** Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ be a tweakable hash function as defined in definition 1. Let $\mathcal{A}$ be a (stateful) algorithm, $\mathcal{M}' \subseteq \mathcal{M}$, and $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}UD}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A})$:

1. Sample $b \xleftarrow{\$} \{0, 1\}$ and $P \xleftarrow{\$} \mathcal{P}$.

2. Run $\mathcal{A}$ with (classical) access to an oracle that takes $T \in \mathcal{T}$ and works as follows:

   - If $|Q| \geq p$ or $T \in Q$, return $\perp$. Otherwise, insert $T$ into the list $Q$.
   - If $b = 0$, sample $x \xleftarrow{\$} \mathcal{M}'$ and return $y := \mathsf{Th}(P, T, x)$.
   - If $b = 1$, return $y \xleftarrow{\$} \mathcal{H}$.

3. When $\mathcal{A}$ signals to continue, then continue running $\mathcal{A}$ with input $P$, but without the oracle access.

4. Obtain from $\mathcal{A}$ a bit $b' \in \{0,1\}$ and output $b'$.

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

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

We also introduce a new notion, which we will use it in section 5. Looking ahead, we will use a tweakable hash function in encodings that we use during signing, and the new notion will be essential to bound the probability that an adversary finds two messages with the same encoding. To capture multiple instantiations, some inspired by [HKRY23, BHRvV21, GHHM21], we want to allow the encoding to be randomized and fail with a certain probability. If the encoding fails, the signer will re-hash the message in combination with a new randomness. To model this, our new definition is parameterized by a predicate $\mathsf{Prop}$ that tells the hash oracle in the game when to return a digest and randomness. In our application, this predicate will tell if the encoding has succeeded. It is worth mentioning that if we set $K = 1$ and $\mathsf{Prop}$ to be constantly 1, then our new notion matches *multi-target extended target-collision resistance with nonce* (nM-eTCR) [GHHM21].

**Definition 6** (Multi-Target Collision Resistance with Random Sampling)**.** Let $K \in \mathbb{N}$ be an integer. Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R}) \to \mathcal{H}$ be a tweakable hash function, where the input is split into a message part $M \in \mathcal{M}$ and a randomness part $\rho \in \mathcal{R}$. Let $\mathsf{Prop}\colon \mathcal{H} \to \{0,1\}$ be a function that represents some property on the output space. Let $\mathcal{A}$ be a (stateful) algorithm, and $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}rTCR}^{K}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A})$:

1. Generate a random public parameter $P \xleftarrow{\$} \mathcal{P}$.

2. Run $\mathcal{A}$ with an input $P$ and with (classical) access to an oracle that takes $T \in \mathcal{T}$ and $M \in \mathcal{M}$ and works as follows:

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

3. Obtain from $\mathcal{A}$ an output $(j, M^*, \rho^*)$ with $M \in \mathcal{M}$, $j \in [|Q|]$. Denote the $j$th entry in $Q$ by $(M_j, T_j, \rho_j)$.

4. Output 1 if $\mathsf{Th}(P, T_j, M_j, \rho_j) = \mathsf{Th}(P, T_j, M^*, \rho^*)$ and $(M^*, \rho^*) \neq (M_j, \rho_j)$. Otherwise, output 0.

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

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

Table 1: Upper bounds on the success probability of an adversary against properties of tweakable hash functions $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$, when the hash function is modeled as a (classical or quantum) random oracle. For $\mathsf{SM\text{-}rTCR}$, we assume $\mathcal{M} = \mathcal{M}_0 \times \mathcal{R}$. Here, $q$ is the number of (quantum or classical) queries to the hash function and $p$ is the number of classical queries to the challenge oracle, $K$ denotes the number of queries the challenge oracle in $\mathsf{SM\text{-}rTCR}$ makes to the hash function. We set $q' := q + pK$. We will only apply undetectability and preimage resistance for the case $|\mathcal{M}'| = |\mathcal{H}|$.

| | Classical Bound | Reference | Quantum Bound | Reference |
|---|---|---|---|---|
| $\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},p}(\mathcal{A})$ | $\frac{2q+1}{\lvert\mathcal{H}\rvert} + \frac{2q}{\lvert\mathcal{P}\rvert}$ | Section E | $\frac{32(q+1)^2}{\lvert\mathcal{H}\rvert} + \frac{32q^2}{\lvert\mathcal{P}\rvert}$ | [HK22], Section E |
| $\mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{M}',p=1}(\mathcal{A})$ | $\frac{q+1}{\lvert\mathcal{H}\rvert} + \frac{q+1}{\lvert\mathcal{M}'\rvert}$ | [HRS16, BHK+19] | $\frac{8(q+1)^2}{\lvert\mathcal{H}\rvert} + \frac{12(q+1)}{\sqrt{\lvert\mathcal{M}'\rvert}}$ | Section F |
| $\mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathsf{Th},\mathcal{M}',p=1}(\mathcal{A})$ | $\frac{q}{\lvert\mathcal{M}'\rvert}$ | [Hül13, DSS05] | $\frac{12q}{\sqrt{\lvert\mathcal{M}'\rvert}}$ | [HK22], Section C |
| $\mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A})$ | $\frac{(q'+1)}{\lvert\mathcal{H}\rvert} + \frac{q' \cdot pK}{\lvert\mathcal{R}\rvert}$ | Section D | $\frac{8(q'+1)^2}{\lvert\mathcal{H}\rvert} + \frac{3pK}{2} \cdot \sqrt{\frac{q'}{\lvert\mathcal{R}\rvert}}$ | Section D |

**Heuristic Analysis.** In our work, we will reduce the security of hash-based signature schemes to the security of the presented properties of tweakable hash functions. However, to give concrete security levels and deduce parameters one needs to estimate the complexity of breaking these properties. To this end, we present bounds in table 1, assuming the tweakable hash is heuristically modeled as a classical or quantum random oracle [BDF+11]. Some of these bounds are novel and proven in the Supplementary Material. For others we had to revisit the proofs to ensure they work for general input and output spaces. One example is for preimage resistance. The security analysis of this notion in the quantum random oracle was previously based on a conjecture (see [BH19, BHK+19, HK22]). We provide a security analysis without any conjecture. Another example is for target collision resistance. Here, a security bound against a quantum adversary was given in [HK22]. We give a bound against classical adversary in section E and update the quantum bound to work for sets $\mathcal{P}$ of arbitrary size. We also revisit the security bound for undetectability in section C, to show that the proof from [HK22] still applies for arbitrary tweakable hash functions, without restrictions on input and output domains.

## 3.2 Signatures and Multi-Signatures

We now turn to defining signatures and the object we ultimately aim to construct, namely, non-interactive multi-signatures. As already explained in previous works [FSZ22, FHSZ23], in the proof-of-stake setting it is sufficient to consider signatures and multi-signatures in the *synchronized* setting [GR06, AGH10, HW18, DGNW20]. In these schemes, signatures are computed and verified with respect to an epoch $\mathsf{ep} \in [L]$, where $L$ denotes the lifetime of a key, and we assume that every signer only signs one message per epoch, and that we only aggregate signatures on the same message (as usual in multi-signatures) and for the same epoch.

**Definition 7** (Synchronized Signature Scheme)**.** Let $L \in \mathbb{N}$ be a natural number. A synchronized signature scheme with lifetime $L$ is a tuple of efficient algorithms $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ with the following syntax:

- $\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$ takes as input system parameters $\mathsf{par}$ and outputs a public key $\mathsf{pk}$ and a secret key $\mathsf{sk}$.
- $\mathsf{Sig}(\mathsf{sk}, \mathsf{ep}, \mathsf{m}) \to \sigma$ takes as input a secret key $\mathsf{sk}$, an epoch $\mathsf{ep} \in [L]$, and a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$ and outputs a signature $\sigma$.
- $\mathsf{Ver}(\mathsf{pk}, \mathsf{ep}, \mathsf{m}, \sigma) \to b$ is deterministic, takes as input a public key $\mathsf{pk}$, an epoch $\mathsf{ep} \in [L]$, a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$, and a signature $\sigma$, and outputs a bit $b \in \{0,1\}$.

Further, we say that $\mathsf{SIG}$ has correctness error at most $\delta$, if for all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(\mathsf{par})$, all epochs $\mathsf{ep} \in [L]$, and all messages $\mathsf{m} \in \{0,1\}^{l_{msg}}$ we have

$$\Pr[\mathsf{Ver}(\mathsf{pk}, \mathsf{ep}, \mathsf{m}, \sigma) = 0 \mid \sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, \mathsf{ep}, \mathsf{m})] \leq \delta.$$

*Remark* 1 (Message Length). Note that throughout the paper, we consider signatures with respect to messages of a fixed length $l_{msg}$. This is without loss of generality, as arbitrarily long messages can first be compressed using any collision resistant hash function. Clearly, this compression can be done outside of any pqSNARK circuit, and the compressed message is an input to the circuit.

**Definition 8** (Synchronized Security). Let $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a synchronized signature scheme with lifetime $L$, let $\mathcal{A}$ be any algorithm. Consider the following experiment $\mathbf{SY\text{-}UF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{A})$:

1. Generate keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par})$.
2. Run $\mathcal{A}$ on input $\mathsf{par}$ and $\mathsf{pk}$, and with (classical) access to the following oracle:
   - $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ for $\mathsf{ep} \in [L], \mathsf{m} \in \{0,1\}^{l_{msg}}$: If $\mathsf{Signed}[\mathsf{ep}] \neq \bot$, then return $\bot$. Otherwise, compute $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, \mathsf{ep}, \mathsf{m})$, set $\mathsf{Signed}[\mathsf{ep}] := (\mathsf{m}, \sigma)$ and return $\sigma$.
3. Obtain from $\mathcal{A}$ a forgery $(\mathsf{ep}^*, \mathsf{m}^*, \sigma^*)$ with $\mathsf{ep}^* \in [L]$ and $\mathsf{m}^* \in \{0,1\}^{l_{msg}}$. Output 1 if it holds that $\mathsf{Ver}(\mathsf{pk}, \mathsf{ep}^*, \mathsf{m}^*, \sigma^*) = 1$ and $(\mathsf{m}^*, \sigma^*) \neq \mathsf{Signed}[\mathsf{ep}^*]$. Otherwise, output 0.

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

$$\mathsf{Adv}_{\mathsf{SIG}}^{\mathsf{SY\text{-}UF\text{-}CMA}}(\mathcal{A}) := \Pr[\mathbf{SY\text{-}UF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{A}) \Rightarrow 1].$$

*Remark* 2 (Strong Unforgeability). Our definition models *strong* unforgeability, i.e., a forgery is even considered valid if it is for a message that has been queried before, but with a new signature.

In a non-interactive multi-signature, we require that individual signatures on the same message can be publicly aggregated into an (ideally, short) aggregate signature. The aggregate signature can then be verified with respect to the list of public keys. Again, we consider the synchronized setting. We denote the aggregation algorithm by $\mathsf{Aggregate}$ and postpone the formal definition of synchronized (non-interactive) multi-signatures to section A.1.

## 3.3 Merkle Trees

We recall Merkle trees, implemented using tweakable hash functions. Abstractly, a Merkle tree represents a vector commitment, namely, a succinct commitment to a sequence of values, for which any value can later be opened using a short opening. In the case of a Merkle tree, this opening is called a Merkle path.

**Construction 1** (Merkle Tree). *Let $\mathcal{L}$ be a set and $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ be a tweakable hash function with $\mathcal{L} \subseteq \mathcal{M}$ and $\mathcal{H}^2 \subseteq \mathcal{M}$. The Merkle tree using $\mathsf{Th}$ with $2^h$ leafs in leaf space $\mathcal{L}$ is defined by the following set of algorithms:*

- $\mathsf{BuildTree}(P, x_1, \ldots, x_{2^h}) \to ((X_{l,i-1})_{i \in [2^{h-l}]})_{l \in [h]}$, *where $P \in \mathcal{P}$ and $x_j \in \mathcal{L}$ for all $j \in [2^h]$:*
  1. $s := 2^h$

2. *For* $i \in [2^h]$*:* $X_{0,i-1} := \mathsf{Th}(P, \mathsf{tweakmt}(0, i-1), x_i)$
3. *For* $l \in [h]$*:*
   (a) $s := s/2$
   (b) *For* $i \in \{0, \ldots, s-1\}$*:* $X_{l,i} := \mathsf{Th}(P, \mathsf{tweakmt}(l, i), (X_{l-1,2i}, X_{l-1,2i+1}))$

- $\mathsf{Root}(P, x_1, \ldots, x_{2^h}) \to \mathsf{root}$*, where* $P \in \mathcal{P}$ *and* $x_j \in \mathcal{L}$ *for all* $j \in [2^h]$*:*
  1. $((X_{l,i-1})_{i\in[2^{h-l}]})_{l\in[h]} := \mathsf{BuildTree}(P, x_1, \ldots, x_{2^h})$
  2. $\mathsf{root} := X_{h,0}$
- $\mathsf{Path}(P, x_1, \ldots, x_{2^h}, i) \to \mathsf{path}$*, where* $P \in \mathcal{P}$*,* $x_j \in \mathcal{L}$ *for all* $j \in [2^h]$ *and* $i \in [2^h]$*:*
  1. $((X_{l,i-1})_{i\in[2^{h-l}]})_{l\in[h]} := \mathsf{BuildTree}(P, x_1, \ldots, x_{2^h})$
  2. $\hat{i} := i - 1$
  3. *For* $l \in \{0, \ldots, h-1\}$*:*
     (a) $\mathsf{sibl}[l] := \hat{i} \oplus$ *0x01*
     (b) $\hat{i} := \lfloor \hat{i}/2 \rfloor$
  4. $\mathsf{path} := (X_{l,\mathsf{sibl}[l]})_{0 \le l < h}$
- $\mathsf{VerPath}(P, \mathsf{root}, i, x, \mathsf{path}) \to b$*, where* $P \in \mathcal{P}$*,* $x \in \mathcal{L}$ *and* $i \in [2^h]$*:*
  1. *Write* $\mathsf{path} := (\hat{X}_l)_{0 \le l < h}$
  2. $X := \mathsf{Th}(P, \mathsf{tweakmt}(0, i-1), x)$*,* $\hat{i} := i - 1$
  3. *For* $l \in \{0, \ldots, h-1\}$*:*
     (a) *If* $\hat{i}$ *mod* $2 = 0$*:* $X := \mathsf{Th}(P, \mathsf{tweakmt}(l+1, \lfloor \hat{i}/2 \rfloor), (X, \hat{X}_l))$
     (b) *If* $\hat{i}$ *mod* $2 = 1$*:* $X := \mathsf{Th}(P, \mathsf{tweakmt}(l+1, \lfloor \hat{i}/2 \rfloor), (\hat{X}_l, X))$
     (c) $\hat{i} := \lfloor \hat{i}/2 \rfloor$
  4. $b := 0$*, if* $X = \mathsf{root}$*:* $b := 1$

*Here,* $\mathsf{tweakmt}\colon \{0, \ldots, h\} \times \{0, \ldots, 2^h - 1\} \to \mathcal{T}$ *denotes a fixed publicly known injective mapping.*

*Remark* 3 (Time-Space Trade-Offs). A signer could decide to store some (or all) of the inner nodes of the Merkle tree, to avoid recomputing the entire tree in algorithm Path.

**Lemma 1** (Correctness of Merkle Trees)**.** *Consider a Merkle tree with* $2^h$ *leafs in leaf space* $\mathcal{L}$ *as defined in Construction 1. Then, for every* $x_1, \ldots, x_{2^h} \in \mathcal{L}$ *and every* $i \in [2^h]$*, we have*

$$\mathsf{VerPath}(\mathsf{Root}(x_1, \ldots, x_{2^h}), i, x_i, \mathsf{Path}(x_1, \ldots, x_{2^h}, i)) = 1.$$

*Proof.* This follows by inspection. □

### 3.4 Non-Interactive Argument Systems

We will make black-box use of non-interactive argument systems for our multi-signature construction, while the focus of our work is to explore the security and efficiency of hash-based candidates for the underlying signature scheme. Nonetheless, finding a secure, efficient, and conceptually simple instantiation of such argument systems will be a necessary next step on the road to post-quantum proof-of-stake. The formal definition of non-interactive argument systems and their security (specifically, adaptive knowledge-soundness) is postponed to section A.2. Intuitively, such a system allows a prover holding a statement-witness pair $(\mathsf{stmt}, \mathsf{witn}) \in \Gamma$ in some relation $\Gamma$ to produce an argument string $\pi$ via an algorithm ArgProve. The verifier, which only holds the statement stmt can then check this argument string via an algorithm ArgVer. If it outputs 1, the verifier is convinced that the prover indeed knew a correct witness.

# 4 Generalized XMSS Multi-Signature

In this section, we introduce and analyze a generalized variant of XMSS [BDH11] signatures, and show how to transform it into a multi-signature scheme. Our generalization enables the simultaneous analysis of multiple variants of XMSS. At the same time, it achieves comparable security to directly analyzing individual variants, with no additional security loss.

## 4.1 Incomparable Encoding Schemes

To capture multiple variants of XMSS in a single abstract construction, we introduce the notion of *incomparable encoding schemes*. To understand the definition, it is instructive to recall the basic structure of XMSS signatures. The public key of a XMSS signature is a Merkle root committing to a list of one-time public keys. In the case of XMSS, these are keys for the Winternitz one-time signature scheme. A signature for a message $\mathsf{m}$ and an epoch $\mathsf{ep}$ then contains two components: (1) a one-time signature computed using $\mathsf{sk}_{\mathsf{ep}}$ on the message $\mathsf{m}$, from which a one-time public key $\mathsf{pk}_{\mathsf{ep}}$ can be computed, (2) a Merkle path linking $\mathsf{pk}_{\mathsf{ep}}$ to the Merkle root. The Winternitz one-time signature and the variants we consider here use an internal signing mechanism that abstractly has the following properties:

- It signs digests $x \in (\{0,1\}^w)^v$, i.e., strings of length $vw$ split into $w$-bit chunks. To sign a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$, $\mathsf{m}$ is first mapped to $x$.
- This mapping must be *incomparable*: roughly, there are no two digests $x$, $x'$ obtained from distinct messages such that each chunk of $x'$ is larger than the respective chunk of $x$.

For instance, the Winternitz scheme first hashes $\mathsf{m}$ into a digest of length $\kappa < vw$ bits and then augments the digest with a short checksum of length $vw - \kappa$ to obtain $x$. The checksum ensures incomparability. We now make this abstraction formal by giving the definition of incomparable encoding schemes and a security notion for it. Such a scheme maps a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$ to a codeword $x \in \mathcal{C}$. Crucially, the code $\mathcal{C}$ has the incomparability property sketched above. Namely, two distinct codewords are incomparable. It may still be possible that two messages map to the same codeword, but it should be computationally hard to find such messages. To model this, we introduce a target collision resistance notion.

**Definition 9** (Incomparable Encoding Scheme)**.** An incomparable encoding (IE) with public parameter space $\mathcal{P}$, randomness space $\mathcal{R}$, lifetime $L$, chunk size $w$, code length $v$, and code $\mathcal{C} \subseteq \{0,\ldots,2^w-1\}^v$ is an efficiently computable function

$$\mathsf{IncEnc}\colon \mathcal{P} \times \{0,1\}^{l_{msg}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\perp\},$$

such that for every *distinct* codewords $x = (x_1,\ldots,x_v) \in \mathcal{C}$ and $x' = (x'_1,\ldots,x'_v) \in \mathcal{C}$, we have

$$(\exists i \in [v] : x_i < x'_i) \wedge (\exists i' \in [v] : x'_{i'} < x_{i'}).$$

**Definition 10** (Error of IE)**.** Let $\mathsf{IncEnc}\colon \mathcal{P} \times \{0,1\}^{l_{msg}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\perp\}$ be an incomparable encoding scheme. We say that $\mathsf{IncEnc}$ has error at most $\delta$, if for every $P \in \mathcal{P}$, $\mathsf{m} \in \{0,1\}^{l_{msg}}$, and every $\mathsf{ep} \in [L]$, we have

$$\Pr_{\rho \xleftarrow{\$} \mathcal{R}}[\mathsf{IncEnc}(P,\mathsf{m},\rho,\mathsf{ep}) = \perp] \leq \delta.$$

**Definition 11** (Target Collision Resistance for IE)**.** Let $\mathsf{IncEnc}\colon \mathcal{P} \times \{0,1\}^{l_{msg}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\perp\}$ be an incomparable encoding scheme with code $\mathcal{C} \subseteq \{0, \ldots, 2^w - 1\}^v$. Let $K \in \mathbb{N}$ be an integer. For any algorithm $\mathcal{A}$, consider the following experiment $\textbf{T-COLL-RES}^K_{\mathsf{IncEnc},p}(\mathcal{A})$:

1. Sample parameters $P \xleftarrow{\$} \mathcal{P}$.
2. Run $\mathcal{A}$ with input $P$ and (classical) access to an oracle that takes as input $\mathsf{m} \in \{0,1\}^{l_{msg}}$, $\mathsf{ep} \in [L]$ and is defined as follows:
   (a) If there exists an entry $(\mathsf{m}', \rho, \mathsf{ep}, x) \in \mathcal{L}$ (with the same $\mathsf{ep}$) or $|\mathcal{L}| \geq p$ , then return $\perp$.
   (b) Set $\mathsf{ctr} := 0$ and $x := \perp$. While $\mathsf{ctr} < K$ and $x = \perp$:
      i. Sample $\rho \xleftarrow{\$} \mathcal{R}$.
      ii. Set $x := \mathsf{IncEnc}(P, \mathsf{m}, \rho, \mathsf{ep})$.
      iii. Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
   (c) If $x = \perp$, insert $(\mathsf{m}, \perp, \mathsf{ep}, \perp)$ into $\mathcal{L}$ and return $\perp$.
   (d) Else (note: $x \in \mathcal{C}$), insert $(\mathsf{m}, \rho, \mathsf{ep}, x)$ into $\mathcal{L}$ and return $(x, \rho)$.
3. Get from $\mathcal{A}$ a triple $(\mathsf{m}^*, \rho^*, \mathsf{ep}^*) \in \{0,1\}^{l_{msg}} \times \mathcal{R} \times [L]$. Then compute the encoding $x^* := \mathsf{IncEnc}(P, \mathsf{m}^*, \rho^*, \mathsf{ep}^*)$.
4. Output 1 if and only if there is a pair $(\mathsf{m}, \rho) \neq (\mathsf{m}^*, \rho^*)$ with $(\mathsf{m}, \rho, \mathsf{ep}^*, x^*) \in \mathcal{L}$.

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

$$\mathsf{Adv}^{\mathsf{T\text{-}COLL\text{-}RES},K}_{\mathsf{IncEnc},p}(\mathcal{A}) := \Pr\left[\textbf{T-COLL-RES}^K_{\mathsf{IncEnc},p}(\mathcal{A}) \Rightarrow 1\right].$$

*Remark* 4 (Related Notions). Our definition of incomparable encodings is somewhat inspired by [ZCY23]. In contrast to them, we allow for randomized encoding functions via an explicit randomness space, we make the epoch and public parameters an input of the encoding, and we define a computational security notion resembling target collision resistance for it. We will make use of this notion in the security analysis of our generalized XMSS signature. Also, our encodings can fail (output $\perp$), which allows us to capture more instantiations. The incomparability notion is also similar to the notion of domination free functions presented in [BS20]. In contrast to their abstraction, again ours is randomized and takes more inputs. Also, we apply our abstraction directly to XMSS, whereas they define a generalized variant of Winternitz one-time signatures. We found that considering XMSS directly yields a tighter analysis.

## 4.2 Generalized XMSS Signature

With the definition of incomparable encoding schemes at hand, we can now define an abstract version of the XMSS signature scheme. For that, we make use of hash chains, as defined next.

**Construction 2** (Hash Chains)**.** *Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ be a tweakable hash function such that $\mathcal{H} \subseteq \mathcal{M}$. Let $L, v, w \in \mathbb{N}$ and $P \in \mathcal{P}$. For a start index $k \in \{0, \ldots, 2^w - 1\}$, a number of steps $s \in \{0, \ldots, 2^w - 1 - k\}$, an element $x \in \mathcal{H}$, a chain index $i \in [v]$, and epoch $\mathsf{ep} \in [L]$, we denote the evaluation of the ith hash chain in epoch $\mathsf{ep}$ for $s$ steps starting from $x$ as $\mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, k, s, x) \in \mathcal{H}$. Formally:*

- $\mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, k, s, x) \to y$*:*
  1. $y := x$

2. *If* $s = 0$*, return* $y$*.*
3. *For* $j \in [s]$*:* $y := \mathsf{Th}(P, \mathsf{tweak}(\mathsf{ep}, i, k + j), y)$

*Here, we assume* $\mathsf{tweak}\colon [L] \times [v] \times [2^w - 1] \to \mathcal{T}$ *is a fixed publicly known injective mapping. Importantly, we assume that the image of this mapping is disjoint from the image of* $\mathsf{tweakmt}$ *in Construction 1.*

The following lemma is essential for correctness of the generalized XMSS construction. It states that first walking $s$ steps, and then walking the remaining $2^w - 1 - s$ steps results in the same as walking $2^w - 1$ steps in one go.

**Lemma 2** (Associativity of Hash Chains)**.** *Let* $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ *be a tweakable hash function such that* $\mathcal{H} \subseteq \mathcal{M}$*. Let* $L, v, w \in \mathbb{N}$ *and* $P \in \mathcal{P}$*. Fix any* $i \in [v]$*,* $\mathsf{ep} \in [L]$*,* $x \in \mathcal{H}$*, and* $s \in \{0, \ldots, 2^w - 1 - k\}$*. Then, we have*

$$\mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, 0, 2^w - 1, x) = \mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, s, 2^w - 1 - s, \mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, 0, s, x))$$

*Proof.* This follows from the simple observation that the tweaks that are used are the same on both sides. The tweaks used on the left hand side are $\mathsf{tweak}(\mathsf{ep}, i, 1), \ldots, \mathsf{tweak}(\mathsf{ep}, i, 2^w - 1)$. This is the same as on the right hand side: namely, the tweaks on the right hand side are $\mathsf{tweak}(\mathsf{ep}, i, 1), \ldots, \mathsf{tweak}(\mathsf{ep}, i, s)$ and then $\mathsf{tweak}(\mathsf{ep}, i, s + 1), \ldots, \mathsf{tweak}(\mathsf{ep}, i, 2^w - 1)$. ☐

**Construction 3** (Generalized XMSS)**.** *Let* $L = 2^h$ *be a power of two. Let* $\mathsf{IncEnc}\colon \mathcal{P} \times \{0,1\}^{l_{msg}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\perp\}$ *be an incomparable encoding with public parameter space* $\mathcal{P}$*, randomness space* $\mathcal{R}$*, lifetime* $L$*, chunk size* $w$*, code length* $v$*, and code* $\mathcal{C} \subseteq \{0, \ldots, 2^w - 1\}^v$*. Let* $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ *be a tweakable hash function, such that* $\mathcal{H} \subseteq \mathcal{M}$*,* $\mathcal{H}^2 \subseteq \mathcal{M}$*, and* $\mathcal{H}^v \subseteq \mathcal{M}$*. Let* $K \in \mathbb{N}$ *be an integer. Consider the Merkle tree using* $\mathsf{Th}$ *with* $2^h$ *leafs in leaf space* $\mathcal{L} = \mathcal{H}^v$*, as defined in Construction 1. We construct a synchronized signature scheme* $\mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K]$ *using hash chains (cf. Construction 2) with lifetime* $L$ *as follows:*

- $\mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K].\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$*:*
  1. $P \xleftarrow{\$} \mathcal{P}$
  2. *For* $\mathsf{ep} \in [L]$*:*
     (a) *For* $i \in [v]$*:* $\mathsf{sk}_{\mathsf{ep},i} \xleftarrow{\$} \mathcal{H}$
     (b) *For* $i \in [v]$*:* $\mathsf{pk}_{\mathsf{ep},i} := \mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, 0, 2^w - 1, \mathsf{sk}_{\mathsf{ep},i})$
     (c) $\mathsf{sk}_{\mathsf{ep}} := (\mathsf{sk}_{\mathsf{ep},1}, \ldots, \mathsf{sk}_{\mathsf{ep},v})$
     (d) $\mathsf{pk}_{\mathsf{ep}} := (\mathsf{pk}_{\mathsf{ep},1}, \ldots, \mathsf{pk}_{\mathsf{ep},v})$
  3. $\mathsf{root} := \mathsf{Root}(P, \mathsf{pk}_1, \ldots, \mathsf{pk}_L)$
  4. $\mathsf{pk} := (\mathsf{root}, P)$
  5. $\mathsf{sk} := (P, (\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_L, \mathsf{sk}_L))$
- $\mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K].\mathsf{Sig}(\mathsf{sk}, \mathsf{ep}, \mathsf{m}) \to \sigma$*:*
  1. *Write* $\mathsf{sk} = (P, (\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_L, \mathsf{sk}_L))$
  2. $\mathsf{path}_{\mathsf{ep}} := \mathsf{Path}(P, \mathsf{pk}_1, \ldots, \mathsf{pk}_L, \mathsf{ep})$
  3. *Set* $\mathsf{ctr} := 0$ *and* $x := \perp$*. While* $\mathsf{ctr} < K$ *and* $x = \perp$*:*
     (a) *Sample* $\rho \xleftarrow{\$} \mathcal{R}$ *and set* $x := \mathsf{IncEnc}(P, \mathsf{m}, \rho, \mathsf{ep})$
     (b) *Set* $\mathsf{ctr} := \mathsf{ctr} + 1$
  4. *If* $x = \perp$*, return* $\perp$

5. *Compute* $\sigma_{\mathsf{OTS}}$ *using* $\mathsf{sk}_{\mathsf{ep}}$*:*
   (a) *Write* $x = (x_1, \ldots, x_v) \in \{0, \ldots, 2^w - 1\}^v$
   (b) *Write* $\mathsf{sk}_{\mathsf{ep}} = (\mathsf{sk}_{\mathsf{ep},1}, \ldots, \mathsf{sk}_{\mathsf{ep},v})$
   (c) *For* $i \in [v]$*:* $\sigma_{\mathsf{OTS},i} := \mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, 0, x_i, \mathsf{sk}_{\mathsf{ep},i})$
   (d) $\sigma_{\mathsf{OTS}} := (\sigma_{\mathsf{OTS},1}, \ldots, \sigma_{\mathsf{OTS},v})$
6. $\sigma := (\rho, \sigma_{\mathsf{OTS}}, \mathsf{path}_{\mathsf{ep}})$

- $\mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K].\mathsf{Ver}(\mathsf{pk}, \mathsf{ep}, \mathsf{m}, \sigma) \to b$*:*

1. *Write* $\sigma = (\rho, \sigma_{\mathsf{OTS}}, \mathsf{path}_{\mathsf{ep}})$ *and* $\mathsf{pk} = (\mathsf{root}, P)$
2. $x := \mathsf{IncEnc}(P, \mathsf{m}, \rho, \mathsf{ep})$
3. *If* $x \notin \mathcal{C}$*: return* $0$
4. *Write* $x = (x_1, \ldots, x_v) \in \{0, \ldots, 2^w - 1\}^v$
5. *Write* $\sigma_{\mathsf{OTS}} = (y_1, \ldots, y_v) \in \mathcal{H}^v$
6. *For each* $i \in [v]$ *compute:* $\mathsf{pk}_{\mathsf{ep},i} = \mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}}(P, x_i, 2^w - 1 - x_i, y_i)$
7. $\mathsf{pk}_{\mathsf{ep}} := (\mathsf{pk}_{\mathsf{ep},1}, \ldots, \mathsf{pk}_{\mathsf{ep},v})$
8. $b := \mathsf{VerPath}(P, \mathsf{root}, \mathsf{ep}, \mathsf{pk}_{\mathsf{ep}}, \mathsf{path}_{\mathsf{ep}})$

*Remark* 5 (Generating Keys using PRFs). In practice, the secret key would be generated using a pseudorandom function to save memory. We omit this optimization here and note that this only changes the security bound by an additional additive term for the security of the pseudorandom function.

*Remark* 6 (Verifier Hashing). As explained earlier, the amount of hashing in the verification algorithm directly influences the computational cost of generating a succinct argument to aggregate signatures. In the generalized XMSS construction, the verifier performs hashing operations for two primary purposes: (1) to verify the Merkle path and (2) to traverse the chains. Additionally, there may be further hashing required to evaluate $\mathsf{IncEnc}$. For (2), the worst case hashing is given by the expression

$$\max_{x \in \mathcal{C}} \sum_{i \in [v]} 2^w - 1 - x_i = v(2^w - 1) - \min_{x \in \mathcal{C}} \sum_{i \in [v]} x_i.$$

Therefore, we want to use encodings with codewords $x$ for which $\sum_{i \in [v]} x_i$ is as big as possible.

*Remark* 7 (Number of Repetitions). In practice, different signers are free to choose different values for $K$, or even to loop for an unbounded number of times until they find a valid codeword.

**Lemma 3** (Correctness of Generalized XMSS)**.** *Assuming* $\mathsf{IncEnc}$ *has error at most* $\delta$*. Then, the scheme* $\mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K]$*, as defined in Construction 3 has correctness error at most* $\delta^K$*. In other words, the correctness error is at most* $2^{-\lambda}$ *if we set*

$$K := \begin{cases} 1, & \text{if } \delta = 0 \\ \lambda / \log(1/\delta), & \text{if } \delta > 0 \end{cases}.$$

*Proof.* Correctness of the construction follows by the correctness of Merkle trees (lemma 1) and from lemma 2, assuming a suitable $\rho \in \mathcal{R}$ is found. Thus, it remains to upper bound the probability that during the signing procedure, for all of the $K$ independently sampled $\rho \in \mathcal{R}$ we have $\mathsf{IncEnc}(\mathsf{m}, \rho, \mathsf{ep}) = \perp$. This probability is given by $\delta^K$. $\square$

We now show that the security of $\mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K]$ follows from the security of the incomparable encoding scheme $\mathsf{IncEnc}$ and the tweakable hash function $\mathsf{Th}$. Our proof also makes use of techniques from the latest proofs [Hül13, KKF21, HK22, HKRY23]. One aspect in which it differs significantly from those is that we consider a form of strong unforgeability.

**Theorem 1** (Security of Generalized XMSS)**.** *Consider the scheme* $\mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K]$ *with parameters* $n, v, w, L, K \in \mathbb{N}$ *and* $\mathsf{Th}, \mathsf{IncEnc}$ *as in Construction 3. Then, for every algorithm* $\mathcal{A}$*, that makes no more than* $q_s$ *signature queries, there are algorithms* $\mathcal{B}_i$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_i)$ *for all* $i$ *and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SY\text{-}UF\text{-}CMA}}_{\mathsf{SIG}[\mathsf{IncEnc},\mathsf{Th}]}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},2\cdot L\cdot v\cdot 2^w}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{T\text{-}COLL\text{-}RES},K}_{\mathsf{IncEnc},q_s}(\mathcal{B}_2) + 2\cdot \mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},L\cdot v\cdot 2^w}(\mathcal{B}_3)\\ + L\cdot v\cdot 2^w\left(2^w\cdot \mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{B}_5) + \mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{B}_6)\right).\end{aligned}$$

*Proof.* Write $\mathsf{SIG} := \mathsf{SIG}[\mathsf{IncEnc}, \mathsf{Th}, K]$. We prove the statement by giving a sequence of games. For the $i$th game, denoted by **Game.**$i$, we let $\mathsf{Adv}^{\mathbf{Game.}i}_{\mathsf{SIG}}(\mathcal{A})$ be the probability that the game outputs 1.

**Game.0:** Our starting point is the original synchronized security game for adversary $\mathcal{A}$ and scheme $\mathsf{SIG}$, see definition 8. To recall, the game first generates a pair $(\mathsf{pk}, \mathsf{sk})$ as in the scheme and then gives the public key $\mathsf{pk} = (\mathsf{root}, P)$ to $\mathcal{A}$. The adversary then gets access to an oracle $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ to obtain signatures for messages $\mathsf{m}$ and epochs $\mathsf{ep}$. The oracle can only be called once per epoch, and stores the resulting message signature pair $(\mathsf{m}, \sigma)$ as $\mathsf{Signed}[\mathsf{ep}] := (\mathsf{m}, \sigma)$. Finally, the adversary outputs a forgery $(\mathsf{ep}^*, \mathsf{m}^*, \sigma^*)$ and wins if $\mathsf{Ver}(\mathsf{pk}, \mathsf{ep}^*, \mathsf{m}^*, \sigma^*) = 1$ and $(\mathsf{m}^*, \sigma^*) \neq \mathsf{Signed}[\mathsf{ep}^*]$. In the scheme we consider, signatures have the form $\sigma = (\rho, \sigma_{\mathsf{OTS}}, \mathsf{path}_{\mathsf{ep}})$. That is, the second part of the winning condition states that $(\mathsf{m}^*, (\rho^*, \sigma^*_{\mathsf{OTS}}, \mathsf{path}^*_{\mathsf{ep}^*})) \neq \mathsf{Signed}[\mathsf{ep}^*]$, where $\sigma^* = (\rho^*, \sigma^*_{\mathsf{OTS}}, \mathsf{path}^*_{\mathsf{ep}^*})$. We also make the following assumption, which is without loss of generality: we have $\mathsf{Signed}[\mathsf{ep}^*] \neq \bot$ at the end of the game, i.e., $\mathcal{A}$ queried the signing oracle for the forgery epoch[8]. By definition, we have

$$\mathsf{Adv}^{\mathsf{SY\text{-}UF\text{-}CMA}}_{\mathsf{SIG}[\mathsf{IncEnc},\mathsf{Th}]}(\mathcal{A}) = \mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{SIG}}(\mathcal{A}).$$

**Game.1:** We now change the winning condition. Informally, we rule out that the adversary forges by breaking the security of the Merkle tree. More precisely, denote the list of one-time public keys that the game generated during key generation by $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$, i.e., $\mathsf{root} = \mathsf{Root}(P, \mathsf{pk}_1, \ldots, \mathsf{pk}_L)$. Further, let $\sigma^* = (\rho^*, \sigma^*_{\mathsf{OTS}}, \mathsf{path}^*_{\mathsf{ep}})$ be $\mathcal{A}$'s forgery, and let $\mathsf{pk}^*_{\mathsf{ep}^*}$ denote the one-time public key for this epoch that the verification algorithm recomputes from $\sigma^*_{\mathsf{OTS}}$ and $x^* = \mathsf{IncEnc}(P, \mathsf{m}^*, \rho^*, \mathsf{ep}^*)$. Let $\mathsf{path}_{\mathsf{ep}^*} := \mathsf{Path}(P, \mathsf{pk}_1, \ldots, \mathsf{pk}_L, \mathsf{ep}^*)$ be the Merkle path that the signing oracle $\textsc{Sig}(\mathsf{ep}^*, \cdot)$ would include in signatures. Now, the game **Game.1** would output 0 if

$$(\mathsf{pk}_{\mathsf{ep}^*}, \mathsf{path}_{\mathsf{ep}^*}) \neq (\mathsf{pk}^*_{\mathsf{ep}^*}, \mathsf{path}^*_{\mathsf{ep}^*}). \tag{1}$$

Here, the left hand side is what honest signing would compute, and the right hand side is derived from the forgery. Otherwise, the game checks the winning condition as before. The games only differ if eq. (1) holds. We will now argue that this event can be bounded by a reduction $\mathcal{B}_1$ breaking target collision resistance, i.e.,

$$\left|\mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{SIG}}(\mathcal{A}) - \mathsf{Adv}^{\mathbf{Game.1}}_{\mathsf{SIG}}(\mathcal{A})\right| \leq \mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},2\cdot L\cdot v\cdot 2^w}(\mathcal{B}_1).$$

[8]Otherwise, just build a wrapper adversary around $\mathcal{A}$ that queries the signing oracle on a different message after receiving the forgery from $\mathcal{A}$.

To understand how such a reduction $\mathcal{B}_1$ works, consider the part of the Merkle tree that is revealed when opening the leaf at position $\mathsf{ep}^*$. Assume $(\mathsf{pk}_{\mathsf{ep}^*}, \mathsf{path}_{\mathsf{ep}^*}) \neq (\mathsf{pk}^*_{\mathsf{ep}^*}, \mathsf{path}^*_{\mathsf{ep}^*})$. Both pairs reveal nodes at the same positions in the Merkle tree, but because the pairs are not the same, at least one of the nodes differ. That is, the part of the Merkle tree that is recomputed from the forgery differs from the one that would be recomputed from an honest signature for that epoch, i.e., from the Merkle tree that the game originally created during key generation. The nodes in which the two differ can be the leaf in the Merkle tree (in case $\mathsf{pk}_{\mathsf{ep}^*} \neq \mathsf{pk}^*_{\mathsf{ep}^*}$), or some internal node on the authentication path. Still, because the forgery is accepted, the root of the Merkle tree computed from the forgery must match the root of the original Merkle tree. Due to a pigeon hole argument there must be a collision somewhere in the Merkle tree. We now sketch how to use it to break target collision resistance.

In a reduction, we would first get a $\mathsf{Th}(P, \cdot, \cdot)$ oracle. We would use it to simulate the key generation process in **Game.0**. To do so we first generate secret elements $\mathsf{sk}_{i,j},\ i \in [L],\ j \in [v]$ uniformly at random. Next, we query $\mathsf{Th}(P, \cdot, \cdot)$ with the corresponding secret values and tweaks to build all chains and compute $\mathsf{pk}_{\mathsf{ep}},\ \mathsf{ep} \in [L]$. Now, we build the Merkle tree. For that, we again use the oracle $\mathsf{Th}(P, \cdot, \cdot)$. In this way, we can set up all chains and the Merkle tree without explicit access to $P$. Then, we would signal that the first stage of the target collision resistance game is completed, and get $P$ from the game. In combination with the Merkle root, this serves as the public key, which we then give to $\mathcal{A}$. As we know all secret keys, we can perfectly simulate the rest of **Game.0** for $\mathcal{A}$. If for the forgery it holds that $(\mathsf{pk}_{\mathsf{ep}^*}, \mathsf{path}_{\mathsf{ep}^*}) \neq (\mathsf{pk}^*_{\mathsf{ep}^*}, \mathsf{path}^*_{\mathsf{ep}^*})$, there must be a collision as explained above. By recomputing the root of the Merkle tree from the forged signature we find this collision efficiently and can break target collision resistance.

**Game.2:** This is the same as **Game.1** but we let the game output 0 if we can extract a collision for the incomparable encoding scheme. More precisely, recall that we consider only the case in which $\textsc{Sig}(\mathsf{ep}^*, \cdot)$ has been queried, and let $(\mathsf{m}, \sigma) = \mathsf{Signed}[\mathsf{ep}]$, where $\sigma = (\rho, \sigma_{\mathsf{OTS}}, \mathsf{path}_{\mathsf{ep}^*})$. We let the game output 0 if we have

$$(\mathsf{m}, \rho) \neq (\mathsf{m}^*, \rho^*) \wedge \mathsf{IncEnc}(P, \mathsf{m}, \rho, \mathsf{ep}^*) = \mathsf{IncEnc}(P, \mathsf{m}^*, \rho^*, \mathsf{ep}^*). \tag{2}$$

Otherwise, the game outputs what **Game.1** would output. We can easily bound the difference between these two games using a reduction $\mathcal{B}_2$ against target collision resistance of $\mathsf{IncEnc}$ (definition 11). We sketch it:

1. Get $P$ as input from the target collision resistance game, and access to an oracle, denoted by O. Use $P$ to generate $(\mathsf{pk}, \mathsf{sk})$ as in **Game.1** honestly.

2. Run $\mathcal{A}$ as in **Game.1** on input $\mathsf{pk}$, and implement $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ as in **Game.1**, but to compute $x$ use oracle O on input $\mathsf{m}, \mathsf{ep}$.

3. Get from $\mathcal{A}$ a forgery, and output $(\mathsf{m}^*, \rho^*, \mathsf{ep}^*)$ if eq. (2) holds.

First, the simulation of the game provided by the reduction is perfect, and its running time is about that of $\mathcal{A}$. Second, note that if eq. (2) holds, then $\mathcal{B}_2$ breaks the target collision resistance of $\mathsf{IncEnc}$. We get

$$\left|\mathsf{Adv}^{\mathbf{Game.1}}_{\mathsf{SIG}}(\mathcal{A}) - \mathsf{Adv}^{\mathbf{Game.2}}_{\mathsf{SIG}}(\mathcal{A})\right| \leq \mathsf{Adv}^{\mathsf{T\text{-}COLL\text{-}RES},K}_{\mathsf{IncEnc},q_s}(\mathcal{B}_2).$$

Let us summarize what we have now, using the same notation as above: if **Game.2** outputs 1, then $(\mathsf{m}^*, (\rho^*, \sigma^*_{\mathsf{OTS}}, \mathsf{path}^*_{\mathsf{ep}^*})) \neq (\mathsf{m}, (\rho, \sigma_{\mathsf{OTS}}, \mathsf{path}_{\mathsf{ep}^*}))$. Due to the changes we have introduced, this means one of the following must hold:

1. $(\mathsf{m}, \rho) \neq (\mathsf{m}^*, \rho^*)$ and $x \neq x^*$ for $x = \mathsf{IncEnc}(P, \mathsf{m}, \rho, \mathsf{ep}^*)$ and $x^* = \mathsf{IncEnc}(P, \mathsf{m}^*, \rho^*, \mathsf{ep}^*)$, or

2. $(\mathsf{m}, \rho) = (\mathsf{m}^*, \rho^*)$ (consequently: $x = x^*$), but $\sigma_{\mathsf{OTS}} \neq \sigma^*_{\mathsf{OTS}}$.

In both cases, we have $(\mathsf{pk}_{\mathsf{ep}^*}, \mathsf{path}_{\mathsf{ep}^*}) = (\mathsf{pk}^*_{\mathsf{ep}^*}, \mathsf{path}^*_{\mathsf{ep}^*})$ with the notation of **Game.1**. In the following, we will first eliminate the second case, and then focus on the first one.

**Game.3:** As already mentioned, we now deal with the second case. Namely, we define **Game.3** to be exactly as **Game.2**, but it outputs 0 if **Game.2** would output 1 and we are in the second case, i.e., $(\mathsf{m}, \rho) = (\mathsf{m}^*, \rho^*)$, but $\sigma_{\mathsf{OTS}} \neq \sigma^*_{\mathsf{OTS}}$ and $(\mathsf{pk}_{\mathsf{ep}^*}, \mathsf{path}_{\mathsf{ep}^*}) = (\mathsf{pk}^*_{\mathsf{ep}^*}, \mathsf{path}^*_{\mathsf{ep}^*})$. To bound the difference between **Game.2** and **Game.3**, denote $\sigma_{\mathsf{OTS}} = (y_1, \ldots, y_v)$ and $\sigma^*_{\mathsf{OTS}} = (y^*_1, \ldots, y^*_v)$ and assume we are in this second case. Then, there must be at least one chain $i \in [v]$ such that $y_i \neq y^*_i$. At the same time, because $\mathsf{pk}_{\mathsf{ep}^*} = \mathsf{pk}^*_{\mathsf{ep}^*}$, we have

$$\begin{aligned}\mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}^*}(P, x_i, 2^w - 1 - x_i, y_i) = \mathsf{pk}_{\mathsf{ep}^*,i} = \mathsf{pk}^*_{\mathsf{ep}^*,i} &= \mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}^*}(P, x^*_i, 2^w - 1 - x^*_i, y^*_i) \\ &= \mathsf{Chain}_{\mathsf{Th},i,\mathsf{ep}^*}(P, x_i, 2^w - 1 - x_i, y_i),\end{aligned}$$

where we have used that $x = x^*$. This constitutes a collision somewhere in the chain, on the way from $y_i$ (resp. $y^*_i$) to $\mathsf{pk}_{\mathsf{ep}^*,i} = \mathsf{pk}^*_{\mathsf{ep}^*,i}$. More formally, we can build a reduction $\mathcal{B}_3$ that breaks target collision resistance of $\mathsf{Th}$ if we are in this case. It requires $L \cdot v \cdot 2^w$ many targets (one per step in each chain in each epoch). We leave the reduction as a simple exercise to the reader, and get

$$\left|\mathsf{Adv}^{\mathbf{Game.2}}_{\mathsf{SIG}}(\mathcal{A}) - \mathsf{Adv}^{\mathbf{Game.3}}_{\mathsf{SIG}}(\mathcal{A})\right| \leq \mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},L\cdot v\cdot 2^w}(\mathcal{B}_3).$$

Now that we have ruled out the second case, we can focus on the first case, i.e., the case in which $x \neq x^*$. Note that the verification algorithm checks that $x^* \in \mathcal{C}$ and $x \in \mathcal{C}$ by construction. Therefore, the definition of the incomparable encoding scheme (definition 9) ensures that there exists some $i \in [v]$ such that $x^*_i < x_i$. From now on, $i^*$ denotes the minimum $i^*$. The focus of the following games will be to consider what happens in this chain $i^*$. There are two options: either, the value at position $x_{i^*}$ in the chain that we recompute from the adversary's signature $\sigma^*_{\mathsf{OTS}}$ is different from $y_i$. In this case, we have another collision (as the ends of the chain are the same) and we can again reduce to target collision resistance. Or, it is the same, in which case our goal will be to reduce to preimage resistance. Subsequent games implement this intuition.

**Game.4:** The game is exactly as **Game.3**, but it additionally outputs 0 if

$$\mathsf{Chain}_{\mathsf{Th},i^*,\mathsf{ep}^*}(P, x^*_{i^*}, x_{i^*} - x^*_{i^*}, y^*_{i^*}) \neq y_{i^*}, \tag{3}$$

where we use the notation from **Game.3**. Denote the left hand side by $\hat{y}$. Clearly, **Game.3** and **Game.4** only differ if eq. (3) holds. We claim that this again constitutes a collision. To see this, note that

$$\begin{aligned}\mathsf{Chain}_{\mathsf{Th},i^*,\mathsf{ep}^*}(P, x_{i^*}, 2^w - 1 - x_{i^*}, y_{i^*}) = \mathsf{pk}_{\mathsf{ep}^*,i^*} &= \mathsf{pk}^*_{\mathsf{ep}^*,i^*} \\ &= \mathsf{Chain}_{\mathsf{Th},i^*,\mathsf{ep}^*}(P, x^*_{i^*}, 2^w - 1 - x^*_{i^*}, y^*_{i^*}) \\ &= \mathsf{Chain}_{\mathsf{Th},i^*,\mathsf{ep}^*}(P, x_{i^*}, 2^w - 1 - x_{i^*}, \hat{y}),\end{aligned}$$

where we have again used that $\mathsf{pk}_{\mathsf{ep}^*} = \mathsf{pk}^*_{\mathsf{ep}^*}$, that the forgery contains a valid signature, and a straight-forward generalization of lemma 2. Assuming eq. (3) holds, there must be some collision in that chain. Again, we can formally obtain an efficient reduction $\mathcal{B}_4$ that breaks target collision resistance, and

$$\left|\mathsf{Adv}^{\mathbf{Game.3}}_{\mathsf{SIG}}(\mathcal{A}) - \mathsf{Adv}^{\mathbf{Game.4}}_{\mathsf{SIG}}(\mathcal{A})\right| \leq \mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},L\cdot v\cdot 2^w}(\mathcal{B}_4).$$

From now on, we can hence assume that

$$\mathsf{Chain}_{\mathsf{Th},i^*,\mathsf{ep}^*}(P, x^*_{i^*}, x_{i^*} - x^*_{i^*}, y^*_{i^*}) = y_{i^*}, \tag{4}$$

and the idea is to use preimage resistance to bound the probability of that. Intuitively, $y_{i^*}$ is a hash for which the adversary never learned a preimage, and we can compute a preimage by following the chain from $y^*_{i^*}$. However, note that the preimage of $y_{i^*}$ is not necessarily uniform, as it is also a hash. To deal with that, we apply undetectability. To apply undetectability, we will first make sure we know the epoch $\mathsf{ep}^*$, the chain $i^*$, and the position in the chain $x_{i^*}$ in advance, using a guessing argument.

**Game.5:** We let the game sample $(\mathsf{ep}^*, i^*, x_{i^*}) \xleftarrow{\$} [L] \times [v] \times [2^w - 1]$ in the beginning, then run **Game.4**, but abort as soon as it is clear that these guesses are not correct. This is a standard guessing argument. As the view of $\mathcal{A}$ does not depend on this guess and the game does not change assuming the guess is correct, we get

$$\mathsf{Adv}_{\mathsf{SIG}}^{\mathbf{Game.4}}(\mathcal{A}) \leq L \cdot v \cdot (2^w - 1) \cdot \mathsf{Adv}_{\mathsf{SIG}}^{\mathbf{Game.5}}(\mathcal{A}) \leq L \cdot v \cdot 2^w \cdot \mathsf{Adv}_{\mathsf{SIG}}^{\mathbf{Game.5}}(\mathcal{A}).$$

**Game.6:** We change how key generation works. Namely, after sampling $(\mathsf{ep}^*, i^*, x_{i^*})$ as in **Game.5**, the game sets up $(\mathsf{pk}, \mathsf{sk})$ as in **Game.5**, with the following exception: the position[9] $x_{i^*} - 1$ in the $i^*$th chain for epoch $\mathsf{ep}^*$ is sampled uniformly at random (call its value $z^*$) instead of being a hash of the previous position. Everything else stays the same. Note that assuming the guess of $x_{i^*}$ was correct, the reduction can still simulate the game, e.g., it defines $y_{i^*}$ to be the hash of $z^*$. We can easily apply undetectability (with one target) to get

$$\left|\mathsf{Adv}_{\mathsf{SIG}}^{\mathbf{Game.5}}(\mathcal{A}) - \mathsf{Adv}_{\mathsf{SIG}}^{\mathbf{Game.6}}(\mathcal{A})\right| \leq 2^w \cdot \mathsf{Adv}_{\mathsf{Th},\mathcal{H},1}^{\mathsf{SM\text{-}UD}}(\mathcal{B}_5),$$

for a reduction $\mathcal{B}_5$. The $2^w$ factor comes from a hybrid argument. To understand that, note that the undetectability notion challenges the adversary to distinguish from a hash of a random input and a random string. In our case, we are substituting an intermediate block in one of the chains, which was generated as a result of several consecutive hashes. To cover this difference, a standard hybrid argument can be applied which results in the $2^w$ factor. That is, in the $j$th hybrid, we would replace the $j$th element in the chain with a random value. For a more detailed presentation, we refer the reader to [HK22].

**Final Reduction:** In the final step, we bound the advantage in **Game.6** using an efficient reduction $\mathcal{B}_6$ that breaks preimage resistance of $\mathsf{Th}$ (for a single target). It works as follows:

1. In the first stage, the reduction has access to an oracle O that takes tweaks as input and returns images of random messages.
2. The reduction samples $(\mathsf{ep}^*, i^*, x_{i^*})$ as explained in **Game.5**. It then calls $\mathrm{O}(T)$ with $T = \mathsf{tweak}(\mathsf{ep}^*, i^*, x_{i^*})$ to get $y_{i^*}$.
3. The reduction signals that it completed the first stage, which means it obtains $P$ from the game.
4. The reduction uses $P$ to complete setting up the public key and all information needed to simulate **Game.6** to $\mathcal{A}$. Then, the reduction simulates **Game.6** to $\mathcal{A}$.
5. Once $\mathcal{A}$ outputs its forgery, the reduction uses eq. (4) to compute a preimage of $y_{i^*}$ (by walking the chain) and returns it to the game.

It is clear that the reduction perfectly simulates **Game.6** and that its running time is dominated by that of $\mathcal{A}$. It is also clear that the reduction finds a preimage if **Game.6** outputs 1, and so we can conclude with

$$\mathsf{Adv}_{\mathsf{SIG}}^{\mathbf{Game.6}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{Th},\mathcal{H},1}^{\mathsf{SM\text{-}PRE}}(\mathcal{B}_6).$$

□

[9] Note that $x_{i^*} \geq 1$ and so the position $x_{i^*} - 1$ is well defined.

## 4.3 Multi-Signature Construction

We now show how to aggregate individual signatures of our generalized XMSS signature scheme. Formally, we show how to turn any synchronized signature scheme into a synchronized non-interactive multi-signature scheme. To this end, we follow a well-known approach, e.g., [KCLM22, ACL+22, WW22, DGKV22]: we use any succinct argument system, and the aggregate signature is the succinct argument string. We show that security of the resulting multi-signature reduces *tightly* to knowledge soundness of the argument system and the synchronized signature. It is important to note that our proof relies on *adaptive* knowledge soundness, see section 3.4.

**Construction 4** (Argument-Based Multi-Signature). *Let* $\mathsf{SIG}$ *be a synchronized signature scheme with lifetime* $L$*. Consider the relation*

$$\Gamma := \left\{ \left( \underbrace{(k, \mathsf{ep}, \mathsf{m}, (\mathsf{pk}_i)_{i=1}^k)}_{\mathsf{stmt}}, \underbrace{(\sigma_i)_{i=1}^k}_{\mathsf{witn}} \right) \middle| \; \forall i \in [k] : \mathsf{SIG.Ver}(\mathsf{pk}_i, \mathsf{ep}, \mathsf{m}, \sigma_i) = 1 \right\}.$$

*Let* $\mathsf{AS} = (\mathsf{ArgProve}, \mathsf{ArgVer})$ *be a non-interactive argument system for* $\Gamma$ *with respect to a random oracle* $\mathsf{H}$*. We construct a synchronized multi-signature scheme with lifetime* $L$*, denoted by* $\mathsf{MS}[\mathsf{SIG}, \mathsf{AS}]$*, as follows:*

- $\mathsf{MS}[\mathsf{SIG}, \mathsf{AS}].\mathsf{Gen} = \mathsf{SIG.Gen}$ *and* $\mathsf{MS}[\mathsf{SIG}, \mathsf{AS}].\mathsf{Sig} = \mathsf{SIG.Sig}$
- $\mathsf{MS}[\mathsf{SIG}, \mathsf{AS}].\mathsf{Aggregate}(\mathsf{ep}, \mathsf{m}, ((\mathsf{pk}_i, \sigma_i))_{i=1}^k) \to \bar{\sigma}$*:*
  1. $\mathsf{stmt} := (k, \mathsf{ep}, \mathsf{m}, (\mathsf{pk}_i)_{i=1}^k), \quad \mathsf{witn} := (\sigma_i)_{i=1}^k$
  2. $\bar{\sigma} := \mathsf{ArgProve}^{\mathsf{H}}(\mathsf{stmt}, \mathsf{witn})$
- $\mathsf{MS}[\mathsf{SIG}, \mathsf{AS}].\mathsf{Ver}((\mathsf{pk}_i)_{i=1}^k, \mathsf{ep}, \mathsf{m}, \bar{\sigma}) \to b$*:*
  1. $\mathsf{stmt} := (k, \mathsf{ep}, \mathsf{m}, (\mathsf{pk}_i)_{i=1}^k)$
  2. $b := \mathsf{ArgVer}^{\mathsf{H}}(\mathsf{stmt}, \bar{\sigma})$

**Lemma 4** (Correctness of Argument-Based Multi-Signature). *If* $\mathsf{AS}$ *has correctness error at most* $\delta_{\mathsf{AS}}$ *and* $\mathsf{SIG}$ *has correctness error at most* $\delta_{\mathsf{SIG}}$*, then the scheme* $\mathsf{MS}[\mathsf{SIG}, \mathsf{AS}]$*, as defined in Construction 4, has correctness error at most* $\delta \colon \mathbb{N} \to \mathbb{R}$ *with* $\delta(k) = \delta_{\mathsf{AS}} + k\delta_{\mathsf{SIG}}$ *for all* $k \in \mathbb{N}$*.*

*Proof.* We consider an epoch $\mathsf{ep}$ and a message $\mathsf{m}$, and $k$ signatures $\sigma_i \leftarrow \mathsf{Sig}(\mathsf{sk}_i, \mathsf{ep}, \mathsf{m})$ generated honestly that are aggregated. By a union bound, the probability that we have $((k, \mathsf{ep}, \mathsf{m}, (\mathsf{pk}_i)_{i=1}^k), (\sigma_i)_{i=1}^k) \notin \Gamma$ is at most $k\delta_{\mathsf{SIG}}$. Under the assumption that we have $((k, \mathsf{ep}, \mathsf{m}, (\mathsf{pk}_i)_{i=1}^k), (\sigma_i)_{i=1}^k) \in \Gamma$, the probability that the argument does not verify is at most $\delta_{\mathsf{AS}}$. □

**Theorem 2** (Security of Argument-Based Multi-Signature). *Consider* $\mathsf{MS}[\mathsf{SIG}, \mathsf{AS}]$*, as defined in Construction 4. Assume that* $\mathsf{AS}$ *is an argument of knowledge with extractor* $\mathsf{Ext}$*, loss* $\mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}}$*, and extraction time* $\theta$*. Then, for any algorithm* $\mathcal{A}$ *that makes at most* $t$ *quantum queries to* $\mathsf{H}$*, there is an algorithm* $\mathcal{B}$ *with*

$$\mathbf{T}(\mathcal{B}) \leq \theta(t) + \mathbf{T}(\mathcal{A}) \quad \textit{and} \quad \mathsf{Adv}^{\mathsf{MS\text{-}SY\text{-}UF\text{-}CMA}}_{\mathsf{MS}[\mathsf{SIG},\mathsf{AS}]}(\mathcal{A}) \leq \mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}}\left(\mathsf{Adv}^{\mathsf{SY\text{-}UF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{B})\right).$$

*Proof.* Write $\mathsf{MS} := \mathsf{MS}[\mathsf{SIG}, \mathsf{AS}]$ for short. Our proof will use a sequence of two games, **Game.0** and **Game.1**, and a final reduction. We denote the probability that **Game.**$i$ outputs 1 by $\mathsf{Adv}^{\mathbf{Game}.i}_{\mathsf{MS}}(\mathcal{A})$.

**Game.0:** This is the original synchronized multi-signature game as defined in definition 13. That is, the game first samples a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par})$ and gives $\mathsf{pk}$ to $\mathcal{A}$. Then, it runs $\mathcal{A}$ with classical oracle access to a signing oracle. Notably, $\mathcal{A}$ also gets quantum access to the random oracle $\mathsf{H}$ used in the argument system $\mathsf{AS}$. Finally, the adversary outputs a forgery $(k^*, (\mathsf{pk}_i^*)_{i=1}^{k^*}, \mathsf{ep}^*, \mathsf{m}^*, \bar{\sigma}^*)$ with $\mathsf{ep}^* \in [L]$ and $\mathsf{m}^* \in \{0,1\}^{l_{msg}}$. The game outputs 1 if and only if $\mathsf{Ver}((\mathsf{pk}_i^*)_{i=1}^{k^*}, \mathsf{ep}^*, \mathsf{m}^*, \bar{\sigma}^*) = 1$, i.e., $\mathsf{ArgVer}^{\mathsf{H}}(\mathsf{stmt}, \bar{\sigma}^*) = 1$ for $\mathsf{stmt} := (k, \mathsf{ep}, \mathsf{m}, (\mathsf{pk}_i)_{i=1}^{k})$, and the signing oracle did not sign $\mathsf{m}^*$ in epoch $\mathsf{ep}^*$. In this case, we say that $\mathsf{m}^*$ is fresh. It is also required that there is an $i_0$ such that $\mathsf{pk}_{i_0}^* = \mathsf{pk}$. By definition:

$$\mathsf{Adv}_{\mathsf{MS[SIG,AS]}}^{\mathsf{MS\text{-}SY\text{-}UF\text{-}CMA}}(\mathcal{A}) = \mathsf{Adv}_{\mathsf{MS}}^{\mathbf{Game.0}}(\mathcal{A}).$$

**Game.1:** This game is the same, with two changes: first, the random oracle $\mathsf{H}$ is now provided to $\mathcal{A}$ by the extractor $\mathsf{Ext}$. Second, once the game has checked that $\mathsf{ArgVer}^{\mathsf{H}}(\mathsf{stmt}, \bar{\sigma}^*) = 1$ and that $\mathsf{m}^*$ is fresh, it gives $(\mathsf{stmt}, \pi)$ for $\pi := \bar{\sigma}^*$ to $\mathsf{Ext}$ and get $\mathsf{witn}$ back. It only outputs 1 if $(\mathsf{stmt}, \mathsf{witn}) \in \Gamma$. We claim that

$$\mathsf{Adv}_{\mathsf{MS}}^{\mathbf{Game.0}}(\mathcal{A}) \leq \mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}}\left(\mathsf{Adv}_{\mathsf{MS}}^{\mathbf{Game.1}}(\mathcal{A})\right). \tag{5}$$

To see this, we construct an algorithm $\hat{\mathcal{B}}$ that we use in definition 15. It runs in $\mathbf{KN\text{-}REAL}_{\mathsf{AS}}(\mathcal{A})$ (resp. $\mathbf{KN\text{-}IDEAL}_{\mathsf{AS},\mathsf{Ext}}(\mathcal{A})$) and gets oracle access to $\mathsf{H}$ (resp. $\mathsf{Ext}$). It internally simulates **Game.0** to $\mathcal{A}$ by forwarding $\mathcal{A}$'s oracle queries to its own oracle. Notably, if the winning condition of **Game.0** were to output 0, $\hat{\mathcal{B}}$ outputs $\perp$. Otherwise, $\hat{\mathcal{B}}$ outputs $\mathsf{stmt} := (k, \mathsf{ep}, \mathsf{m}, (\mathsf{pk}_i)_{i=1}^{k})$ and $\pi := \bar{\sigma}$. By definition of $\hat{\mathcal{B}}$, we have

$$\mathsf{Adv}_{\mathsf{MS}}^{\mathbf{Game.0}}(\mathcal{A}) = \Pr\left[\mathbf{KN\text{-}REAL}_{\mathsf{AS}}(\hat{\mathcal{B}}) \Rightarrow 1\right].$$

By the knowledge soundness of $\mathsf{AS}$, we get

$$\Pr\left[\mathbf{KN\text{-}REAL}_{\mathsf{AS}}(\hat{\mathcal{B}}) \Rightarrow 1\right] \leq \mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}}\left(\Pr\left[\mathbf{KN\text{-}IDEAL}_{\mathsf{AS},\mathsf{Ext}}(\hat{\mathcal{B}}) \Rightarrow 1\right]\right).$$

Note that $\mathbf{KN\text{-}IDEAL}_{\mathsf{AS},\mathsf{Ext}}(\hat{\mathcal{B}})$ is the same as **Game.1**, and so

$$\Pr\left[\mathbf{KN\text{-}IDEAL}_{\mathsf{AS},\mathsf{Ext}}(\hat{\mathcal{B}}) \Rightarrow 1\right] = \mathsf{Adv}_{\mathsf{MS}}^{\mathbf{Game.1}}(\mathcal{A}).$$

This shows eq. (5). Also, note that $\hat{\mathcal{B}}$ makes as many queries to $\mathsf{H}$ as $\mathcal{A}$ makes, and therefore **Game.1** runs in time at most $\theta(t) + \mathbf{T}(\mathcal{A})$.

**Final Reduction:** We can easily bound the probability that **Game.1** outputs 1 using a reduction $\mathcal{B}$ that breaks synchronized security of $\mathsf{SIG}$. The reduction gets as input a public key and access to a signing oracle. It forwards the key to $\mathcal{A}$ and simulates **Game.1** by relaying signing queries between the adversary and its own signing oracle. It uses $\mathsf{Ext}$ to provide the random oracle to $\mathcal{A}$, as specified in **Game.1**. If **Game.1** outputs 1, then the extracted witness satisfies $\mathsf{witn} = (\sigma_i)_{i=1}^{k}$, where $\mathsf{SIG.Ver}(\mathsf{pk}, \mathsf{ep}^*, \mathsf{m}^*, \sigma_{i_0}) = 1$. Also, $\mathsf{m}^*$ has never been signed in epoch $\mathsf{ep}^*$ by the signing oracle. Therefore, $\mathcal{B}$ can output $(\mathsf{ep}^*, \mathsf{m}^*, \sigma^*)$ with $\sigma^* := \sigma_{i_0}$ as its forgery. We get

$$\mathsf{Adv}_{\mathsf{MS}}^{\mathbf{Game.1}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{SIG}}^{\mathsf{SY\text{-}UF\text{-}CMA}}(\mathcal{B}).$$

As $\mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}}$ is a non-decreasing function, we get the result. □

# 5 Instantiations of Incomparable Encodings

We now give several instantiations of the abstract construction presented in section 4. To this end, we specify incomparable encoding schemes and show their security. As corollaries, we obtain concrete security bounds for our variants of XMSS.

## 5.1 Classical Winternitz

The first instantiation that we give is essentially the classical Winternitz construction, using tweakable hashes[10]. That is, if we plug it into our generalized XMSS construction, we essentially obtain XMSS (instantiated with tweakable hash functions).

**Construction 5** (IE for Winternitz)**.** *Let $w, L \in \mathbb{N}$ be integers. Let $\mathsf{Th}^{msg}\colon \mathcal{P} \times \mathcal{T} \times (\{0,1\}^{l_{msg}} \times \mathcal{R}) \to \{0,\dots,2^w-1\}^{n_0}$ be a tweakable hash function. Set $n_1 := \lfloor \log_{2^w}(n_0(2^w-1)) \rfloor + 1$. Set $v := n_0 + n_1$. With this, we define the encoding function*

$$\mathsf{IncEnc}_{\mathsf{W}}[\mathsf{Th}^{msg}]\colon \mathcal{P} \times \{0,1\}^{l_{msg}} \times \mathcal{R} \times [L] \to \mathcal{C} \cup \{\bot\},$$

*where $\mathcal{C} \subseteq \{0,\dots,2^w-1\}^v$ is defined as the image of this function. It is given by the following instructions on input $P \in \mathcal{P}, \mathsf{m} \in \{0,1\}^{l_{msg}}, \rho \in \mathcal{R}, \mathsf{ep} \in [L]$:*

1. $(x_1,\dots,x_{n_0}) := \mathsf{Th}^{msg}(P, \mathsf{tweakm}(\mathsf{ep}), (\mathsf{m}, \rho))$ *for* $x_i \in \{0,\dots,2^w-1\}$
2. $c := n_0(2^w-1) - \sum_{i=1}^{n_0} x_i$. *Note:* $0 \le c \le n_0(2^w-1)$
3. *Write* $c = \sum_{i=1}^{n_1} c_i 2^{w(i-1)}$ *for* $c_i \in \{0,\dots,2^w-1\}$
4. *Return* $(x_1,\dots,x_{n_0},c_1,\dots,c_{n_1}) \in \{0,\dots,2^w-1\}^v$

*Here, we assume* $\mathsf{tweakm}\colon [L] \to \mathcal{T}$ *is a fixed publicly known injective mapping.*

**Lemma 5** (Correctness and Error of Winternitz)**.** *The function* $\mathsf{IncEnc}_{\mathsf{W}}[\mathsf{Th}^{msg}]$ *as defined in Construction 5 is an incomparable encoding scheme and has error $\delta = 0$.*

*Proof.* All that we have to prove is that $\mathsf{IncEnc}_{\mathsf{W}}[\mathsf{Th}^{msg}]$ is an incomparable encoding, as $\mathsf{IncEnc}_{\mathsf{W}}[\mathsf{Th}^{msg}]$ never outputs $\bot$. This is (implicitly) in [BS20], but we recall a proof for completeness. Consider two distinct codewords $(x_1,\dots,x_{n_0},c_1,\dots,c_{n_1}) \in \{0,\dots,2^w-1\}^v$ and $(x'_1,\dots,x'_{n_0},c'_1,\dots,c'_{n_1}) \in \{0,\dots,2^w-1\}^v$, i.e., outputs of $\mathsf{IncEnc}_{\mathsf{W}}$. That is, we know that $c = \sum_{i=1}^{n_1} c_i 2^{w(i-1)}$ is a correct checksum for $(x_1,\dots,x_{n_0})$ and $c' = \sum_{i=1}^{n_1} c'_i 2^{w(i-1)}$ is a correct checksum for $(x'_1,\dots,x'_{n_0})$. Assume towards contradiction that $x_i \le x'_i$ and $c_j \le c'_j$ for all $i \in [n_0]$ and all $j \in [n_1]$. Define $\bar{x} = \sum_{i=1}^{n_0} x_i$ and $\bar{x}' = \sum_{i=1}^{n_0} x'_i$. With that, we have $c = n_0(2^w-1) - \bar{x}$ and $c' = n_0(2^w-1) - \bar{x}'$. Due to the inequalities, we also know that $c \le c'$ and $\bar{x} \le \bar{x}'$. As the two codewords are distinct, at least one of the inequalities $x_i \le x'_i$ and $c_j \le c'_j$ has to be strict. In the first case, at least one of the $x_i \le x'_i$ is strict. In particular, we have $\bar{x} < \bar{x}'$ and therefore

$$c = n_0(2^w-1) - \bar{x} > n_0(2^w-1) - \bar{x}' = c'.$$

But this contradicts $c \le c'$. In the second case, at least one of the $c_i \le c'_i$ is strict, i.e., $c < c'$. But again,

$$c = n_0(2^w-1) - \bar{x} \ge n_0(2^w-1) - \bar{x}' = c',$$

a contradiction. □

**Lemma 6** (Target Collision-Resistance of Winternitz)**.** *Consider the function* $\mathsf{IncEnc}_{\mathsf{W}}[\mathsf{Th}^{msg}]$ *as defined in Construction 5, and any $K, p \in \mathbb{N}$. Then, for every algorithm $\mathcal{A}$, there is an algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ and*

$$\mathsf{Adv}^{\mathsf{T\text{-}COLL\text{-}RES},K}_{\mathsf{IncEnc}_{\mathsf{W}}[\mathsf{Th}^{msg}],p}(\mathcal{A}) \le \mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th}^{msg},p,\mathsf{Prop}}(\mathcal{B}),$$

*where* $\mathsf{Prop}\colon \{0,1\}^* \to \{0,1\}$ *always outputs* 1.

[10] More precisely, Winternitz' scheme is a one-time signature scheme, whereas we specify an incomparable encoding. But plugging our incomparable encoding into the generalized XMSS construction, we obtain (almost) the same as if we implement a Merkle tree on top of Winternitz. Of course, we use tweakable hashes and the classical Winternitz scheme does not.

*Proof.* The reduction $\mathcal{B}$ runs in the game for target collision resistance with random sampling for the tweakable hash function $\mathsf{Th}^{msg}$, see definition 6. It is as follows: $\mathcal{B}$ gets as input $P \in \mathcal{P}$ and it gets access to an oracle, which we denote by O. $\mathcal{B}$ runs $\mathcal{A}$ in the target collision resistance game for $\mathsf{IncEnc_W}[\mathsf{Th}^{msg}]$, by giving $P$ as an input and providing the following oracle to $\mathcal{A}$: On input a message $\mathsf{m}$ and an epoch $\mathsf{ep}$, the oracle (simulated by $\mathcal{B}$) first checks if there exists an entry of the form $(\mathsf{m}', \rho, \mathsf{ep}, x) \in \mathcal{L}$ or $|\mathcal{L}| \geq p$. If so, it returns $\perp$. Otherwise, it calls O on input $\mathsf{tweakm(ep)}$ and $\mathsf{m}$. The oracle forwards the response $(x, \rho)$ of O to $\mathcal{A}$ and inserts $(\mathsf{m}, \rho, \mathsf{ep}, x)$ into $\mathcal{L}$ (or $(\mathsf{m}, \perp, \mathsf{ep}, \perp)$ if the response was $\perp$). Finally, when the adversary outputs a triple $(\mathsf{m}^*, \rho^*, \mathsf{ep}^*)$, the reduction first checks if $\mathcal{A}$ wins the game, i.e., if there is a pair $(\mathsf{m}, \rho) \neq (\mathsf{m}^*, \rho^*)$ with $(\mathsf{m}, \rho, \mathsf{ep}^*, x^*) \in \mathcal{L}$. If so, say this is the $j^*$th entry in $\mathcal{L}$. Then, the reduction forwards $(j^*, \mathsf{m}^*, \rho^*)$ to its game.

Clearly, the running time of $\mathcal{B}$ is dominated by running $\mathcal{A}$. As different epochs yield different tweaks, it can be seen that the oracle is simulated perfectly to $\mathcal{A}$. Also, if we assume that $\mathcal{A}$ wins the target collision resistance game of $\mathsf{IncEnc_W}[\mathsf{Th}^{msg}]$, then $\mathcal{B}$ wins its game as well. $\square$

**Corollary 1** (Winternitz Instantiation)**.** *Let* $\mathsf{Th}^{msg}\colon \mathcal{P}\times\mathcal{T}\times(\{0,1\}^{l_{msg}}\times\mathcal{R}) \to \{0,\ldots,2^w-1\}^{n_0}$ *be a tweakable hash function. Let* $\mathsf{Th}\colon \mathcal{P}\times\mathcal{T}\times\mathcal{M}\to\mathcal{H}$ *be a tweakable hash function, such that* $\mathcal{H}\subseteq\mathcal{M}$*,* $\mathcal{H}^2\subseteq\mathcal{M}$*, and* $\mathcal{H}^v\subseteq\mathcal{M}$*. Set* $K := 1$ *and* $\mathsf{Prop}\colon \{0,1\}^* \to \{0,1\}$ *always outputs* $1$*. Consider the scheme* $\mathsf{SIG} := \mathsf{SIG}[\mathsf{IncEnc_W}[\mathsf{Th}^{msg}], \mathsf{Th}, K]$ *obtained from combining Constructions 3 and 5.*

*Then, this scheme has correctness error* $0$*. Furthermore, for every algorithm* $\mathcal{A}$*, there are algorithms* $\mathcal{B}_i$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_i)$ *for all* $i$ *and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SY\text{-}UF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},2\cdot L\cdot v\cdot 2^w}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th}^{msg},q_s,\mathsf{Prop}}(\mathcal{B}_2) + 2\cdot\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},L\cdot v\cdot 2^w}(\mathcal{B}_3)\\ + L\cdot v\cdot 2^w\left(2^w\cdot\mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{B}_5) + \mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{B}_6)\right),\end{aligned}$$

*where* $q_s$ *is the number of signing queries that* $\mathcal{A}$ *makes.*

## 5.2 Target Sum Winternitz

A subtle problem of the Winternitz construction before is that an attacker may compute a signature with a specifically crafted randomness $\rho$ such that the number of verification hashes is high, which has a negative impact on aggregation efficiency. To do so, the attacker just has to try to minimize the sum $\sum_i x_i$. One approach to get a more explicit control on the number of hashes that the verifier makes (see remark 6) is to enforce that the sum $\sum_i x_i$ of chunks is always equal to a constant $T$. One would regenerate $x$ using a counter or fresh randomness if until it satisfies this constraint. In this case, it is known that the checksum can be omitted [HKRY23, ZCY23], which intuitively shrinks the signature size compared to classical Winternitz. We now give an incomparable encoding scheme that uses this technique.

**Construction 6** (IE for Target Sum Winternitz)**.** *Let* $v, w, T \in \mathbb{N}$ *be integers. Let* $\mathsf{Th}^{msg}\colon \mathcal{P}\times\mathcal{T}\times(\{0,1\}^{l_{msg}}\times\mathcal{R})\to\{0,\ldots,2^w-1\}^v$ *be a tweakable hash function. Define the code*

$$\mathcal{C} := \left\{(x_1,\ldots,x_v)\in\{0,\ldots,2^w-1\}^v \;\middle|\; \sum_{i=1}^{v} x_i = T\right\} \subseteq \{0,\ldots,2^w-1\}^v.$$

*With this, we define the encoding function*

$$\mathsf{IncEnc_{TSW}}[\mathsf{Th}^{msg}, T]\colon \mathcal{P}\times\{0,1\}^{l_{msg}}\times\mathcal{R}\times[L]\to\mathcal{C}\cup\{\perp\}.$$

*It is given by the following instructions on input* $P\in\mathcal{P}, \mathsf{m}\in\{0,1\}^{l_{msg}}, \rho\in\mathcal{R}, \mathsf{ep}\in[L]$*:*

*1.* $x := \mathsf{Th}^{msg}(P, \mathsf{tweakm}(\mathsf{ep}), (\mathsf{m}, \rho))$

*2. If* $x \notin \mathcal{C}$*, return* $\perp$*. Else, return* $x \in \{0, \ldots, 2^w - 1\}^v$

*Here, we assume* $\mathsf{tweakm}\colon [L] \to \mathcal{T}$ *is a fixed publicly known injective mapping.*

**Lemma 7** (Correctness and Error of Target Sum Winternitz)**.** *Consider the function* $\mathsf{IncEnc}_{\mathsf{TSW}}[\mathsf{Th}^{msg}, T]$ *as defined in Construction 6, and assume that* $\mathsf{Th}^{msg}$ *is* $\epsilon$*-uniform for seed space* $\mathcal{R}$ *(see definition 2). Then,* $\mathsf{IncEnc}_{\mathsf{TSW}}[\mathsf{Th}^{msg}, T]$ *is an incomparable encoding scheme and has error*

$$\delta = \epsilon + (1 - \eta_T/2^{vw}), \quad \textit{where} \quad (1 + x + \cdots x^{2^w-1})^v = \sum_{i=0}^{(2^w-1)v} \eta_i x^i \in \mathbb{R}[x].$$

*Proof.* We first show that $\mathsf{IncEnc}_{\mathsf{TSW}}[\mathsf{Th}^{msg}, T]$ is an incomparable encoding scheme. To this end, let $x, x' \in \mathcal{C}$ be distinct with $x = (x_1, \ldots, x_v)$ and $x' = (x'_1, \ldots, x'_v)$. Now, assume towards contradiction that every coordinate of $x$ is larger or equal than the respective coordinate of $x'$. We know that at least one of these inequalities has to be strict as $x \neq x'$. Then, we have

$$T = \sum_{i=1}^{v} x_i > \sum_{i=1}^{v} x'_i = T,$$

a contradiction. We now focus on the error of the scheme. For that, we need to fix $P \in \mathcal{P}$, a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$, and an epoch $\mathsf{ep} \in [L]$. We consider the experiment of sampling $\rho \xleftarrow{\$} \mathcal{R}$ and want to get an upper bound on

$$\Pr_{\rho}\left[\mathsf{IncEnc}_{\mathsf{TSW}}[\mathsf{Th}^{msg}, T](P, \mathsf{m}, \rho, \mathsf{ep}) = \perp\right] = \Pr_{\rho}\left[x \notin \mathcal{C}\right] \leq \Pr_{\bar{x}}\left[\bar{x} \notin \mathcal{C}\right] + \epsilon,$$

where we have used $\epsilon$-uniformity of $\mathsf{Th}^{msg}$ and $\bar{x} \xleftarrow{\$} \{0,1\}^{vw}$ in the last step. Therefore, we want to find the probability that the sum of $v$ uniform independent values $0 \leq \bar{x}_i < 2^w$ is not equal to $T$. The total number of ways to pick $v$ such values is of course $2^{vw}$. The number of ways that sum to $T$ is exactly the coefficient of $x^T$ in the expression

$$(1 + x + \cdots x^{2^w-1})^v.$$

A closed expression could be found using the theory of generating functions, using the identity

$$(1 + x + \cdots x^{2^w-1})(1 - x) = 1 - x^{2^w-1}.$$

□

**Lemma 8** (Target Collision-Resistance of Target Sum Winternitz)**.** *Consider the function* $\mathsf{IncEnc}_{\mathsf{TSW}}[\mathsf{Th}^{msg}, T]$ *as defined in Construction 5, and any* $K, p \in \mathbb{N}$*. Let* $\mathsf{Prop}\colon \{0,1\}^* \to \{0,1\}$ *be the predicate that outputs* 1 *if and only if its input is in* $\mathcal{C}$*. Then, for every algorithm* $\mathcal{A}$*, there is an algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *and*

$$\mathsf{Adv}^{\mathsf{T\text{-}COLL\text{-}RES},K}_{\mathsf{IncEnc_W}[\mathsf{Th}^{msg}],p}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th}^{msg},p,\mathsf{Prop}}(\mathcal{B}).$$

*Proof.* The reduction works exactly as in the proof of lemma 6, noting that $\mathsf{Prop}$ outputs 1 exactly if the target collision resistance game for $\mathsf{IncEnc}_{\mathsf{TSW}}[\mathsf{Th}^{msg}, T]$ (see definition 11) finds a valid $x \neq \perp$. □

**Corollary 2** (Target Sum Winternitz Instantiation)**.** *Let* $\mathsf{Th}^{msg}\colon \mathcal{P}\times\mathcal{T}\times(\{0,1\}^{l_{msg}}\times\mathcal{R}) \to \{0,\dots,2^w-1\}^v$ *be a tweakable hash function. Let* $\mathsf{Th}\colon \mathcal{P}\times\mathcal{T}\times\mathcal{M}\to\mathcal{H}$ *be a tweakable hash function, such that* $\mathcal{H}\subseteq\mathcal{M}$*,* $\mathcal{H}^2\subseteq\mathcal{M}$*, and* $\mathcal{H}^v\subseteq\mathcal{M}$*. Fix integers* $T\in\mathbb{N}$ *and* $K\in\mathbb{N}$*. Let* $\mathsf{Prop}\colon\{0,1\}^*\to\{0,1\}$ *be the predicate as in lemma 8. Consider the scheme* $\mathsf{SIG} := \mathsf{SIG}[\mathsf{IncEnc}_{\mathsf{TSW}}[\mathsf{Th}^{msg},T],\mathsf{Th},K]$ *obtained from combining Constructions 3 and 6.*

*Then, this scheme has correctness error at most*

$$\left(\epsilon + (1-\eta_T/2^{vw})\right)^K, \quad \text{where} \quad (1+x+\cdots x^{2^w-1})^v = \sum_{i=0}^{(2^w-1)v} \eta_i x^i \in \mathbb{R}[x].$$

*Furthermore, for every algorithm* $\mathcal{A}$*, there are algorithms* $\mathcal{B}_i$ *with* $\mathbf{T}(\mathcal{A})\approx\mathbf{T}(\mathcal{B}_i)$ *for all* $i$ *and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SY\text{-}UF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},2\cdot L\cdot v\cdot 2^w}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th}^{msg},q_s,\mathsf{Prop}}(\mathcal{B}_2) + 2\cdot\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},L\cdot v\cdot 2^w}(\mathcal{B}_3)\\ + L\cdot v\cdot 2^w\left(2^w\cdot\mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{B}_5) + \mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{B}_6)\right),\end{aligned}$$

*where* $q_s$ *is the number of signing queries that* $\mathcal{A}$ *makes.*

# 6 Parameter Requirements

In this section, we discuss how to set parameters of the schemes. For example, we describe how large the set of parameters $\mathcal{P}$ or the output length of the tweakable hash function has to be, assuming a desired security level is given. To this end, we proceed in two conceptual steps. First, we use the security bounds that we get from theorem 1 and corollaries 1 and 2, which gives us security levels we need for the security properties of hash functions. In a second step, to get concrete parameters for (approximately) $k_C$ bits of classical security and $k_Q$ bits of quantum security, we then use the heuristic bounds from table 1. Again, we note that these are only heuristics and cryptanalysis should focus on the security properties of hash functions with the desired security levels from the first step. We will split our discussion into the parameters related to to the encoding $\mathsf{IncEnc}$ and $\mathsf{Th}^{msg}$ (i.e., $w, v, |\mathcal{R}|$), and to the parameters related to $\mathsf{Th}$ (i.e., $|\mathcal{P}|$ and $|\mathcal{H}|$). In general, we assume that $w$, $L$, $l_{msg}$, $k_C$, and $k_Q$ are given.

**Security Levels for Hash Function Properties.** Our goal is that for any adversary $\mathcal{A}$ running in time $\mathbf{T}(\mathcal{A})$, the fraction $\mathsf{Adv}^{\mathsf{SY\text{-}UF\text{-}CMA}}_{\mathsf{SIG}}(\mathcal{A})/\mathbf{T}(\mathcal{A})$ is at most $2^{-k}$, where $k = k_C$ or $k = k_Q$ depending on whether $\mathcal{A}$ is quantum. Looking at theorem 1 and corollaries 1 and 2, we see that the advantage is the sum of five terms. Consequently, we want that each of these terms, divided by the running time, is at most $2^{-k-\log 5}$. This means we need to ensure the following hardness bounds, for any algorithm $\mathcal{A}$:

$$\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},2\cdot L\cdot v\cdot 2^w}(\mathcal{A})/\mathbf{T}(\mathcal{A}) \leq 2^{-(k+\log 5)}. \tag{6}$$

$$\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},L\cdot v\cdot 2^w}(\mathcal{A})/\mathbf{T}(\mathcal{A}) \leq 2^{-(k+\log 5+1)}. \tag{7}$$

$$\mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{A})/\mathbf{T}(\mathcal{A}) \leq 2^{-(k+\log 5+2w+\log L+\log v)}. \tag{8}$$

$$\mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{H},1}(\mathcal{A})/\mathbf{T}(\mathcal{A}) \leq 2^{-(k+\log 5+w+\log L+\log v)}. \tag{9}$$

$$\mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th}^{msg},q_s,\mathsf{Prop}}(\mathcal{A})/\mathbf{T}(\mathcal{A}) \leq 2^{-(k+\log 5)}. \tag{10}$$

Note that the last requirement depends on the instantiation of the incomparable encoding, in particular on $\mathsf{Th}^{msg}$. In the following, we use the heuristics from table 1 to suggest how to set parameters satisfying these requirements.

**Message Hash and Randomness - Winternitz.** We start with the parameters for the instantiations, focusing first on the Winternitz instantiation (Construction 5). Specifically, we assume that $w$ and $q_s$ are given[11] and we want to determine requirements on $|\mathcal{R}|$ and $n_0$, which also dictates how to set $v$. What we need to satisfy is eq. (10). We use table 1, and note that $p = q_s, K = 1$ and therefore $q' = q + q_s$. We also note that $\{0, \ldots, 2^w - 1\}^{n_0}$ takes the role of $\mathcal{H}$, i.e., we want a lower bound on $|\{0, \ldots, 2^w - 1\}^{n_0}| = 2^{wn_0}$. We will use that the running time of the adversary must be at least $q' + 1$. Now, start with the classical setting. The bound consists of two terms, and we want each of these terms is at most $2^{-(k_C + \log 5 + 1)}$. From the first term, we get the requirement that $wn_0 \geq k_C + \log 5 + 1$. Looking at the second term, we get the requirement that $\log |\mathcal{R}| \geq k_C + \log 5 + \log q_s + 1$. Now, we turn to the quantum setting. Here, again the bound consists of two terms, and we want each of these terms is at most $2^{-(k_Q + \log 5 + 1)}$. The first term is $8(q' + 1)^2/2^{wn_0}$, which when divided by the running time (at least $q' + 1$) becomes $8(q' + 1)/2^{wn_0}$. Now, the first case is that $q' + 1 \geq 2^{k_Q + \log 5 + 1}$. In this case we are done trivially. In the other case, our requirement becomes

$$\frac{8 \cdot 2^{k_Q + \log 5 + 1}}{2^{wn_0}} \leq 2^{-(k_Q + \log 5 + 1)}.$$

Isolating $wn_0$ this becomes $wn_0 \geq 2(k_Q + \log 5 + 1) + 3$. Looking at the second term, we divide by the running time, lower bounded by $q'$, and get

$$\frac{3}{2} q_s K \cdot \sqrt{\frac{q'}{|\mathcal{R}|}} \cdot q'^{-1} = \frac{3}{2} K \cdot \sqrt{\frac{q_s^2 q'}{|\mathcal{R}| q'^2}} = \frac{3}{2} K \cdot \sqrt{\frac{q_s^2}{|\mathcal{R}| q'}} \leq \frac{3}{2} K \cdot \sqrt{\frac{q_s}{|\mathcal{R}|}},$$

where we have used $q_s \leq q'$. If we want that this is at most $2^{-(k_Q + \log 5 + 1)}$, then we get a lower bound $\log |\mathcal{R}| \geq 2(k_Q + \log 5 + \log 3 + \log K) + \log q_s$, and we can use $K = 1$. With that, we get the following list of requirements.

**Parameter Requirement 1** (Parameters for Winternitz)**.** *Let $w$, $q_s$ be given, and assume we use Construction 5. Then, if we want (approximately) $k_C$ bits of classical security and $k_Q$ bits of quantum security, we need to satisfy the following:*

$$n_0 w \geq \max\{k_C + \log 5 + 1, 2(k_Q + \log 5 + 1) + 3\}, \tag{11}$$

$$\log |\mathcal{R}| \geq \max\{k_C + \log 5 + \log q_s + 1, 2(k_Q + \log 5 + \log 3) + \log q_s\}. \tag{12}$$

*Once $n_0$ is set, $v$ can be set as described in Construction 5.*

**Message Hash and Randomness - Target Sum Winternitz.** Turning to the instantiation based on target sum Winternitz (Construction 6), we see that the only difference to Winternitz in terms of setting parameters is that we no longer assume $K = 1$, and that $n_0 w$ is replaced with $vw$.

**Parameter Requirement 2** (Parameters for Target Sum Winternitz)**.** *Let $w$, $q_s$, $K$ be given, and assume we use Construction 6. Then, if we want (approximately) $k_C$ bits of classical security and $k_Q$ bits of quantum security, we need to satisfy the following:*

$$vw \geq \max\{k_C + \log 5 + 1, 2(k_Q + \log 5 + 1) + 3\}, \tag{13}$$

$$\log |\mathcal{R}| \geq \max\{k_C + \log 5 + \log q_s + \log K + 1, 2(k_Q + \log 5 + \log 3 + \log K) + \log q_s\}. \tag{14}$$

**Hash and Parameter Length.** Now that we know how to set $v$, we turn to the parameters related to $\mathsf{Th}$, e.g., $|\mathcal{H}|$ or $|\mathcal{P}|$, which are dictated by eqs. (6) to (9). These only depend on the underlying incomparable encoding via the parameters $w$ and $v$, which we assume as given for this paragraph. We start with the classical setting. Focus

[11] One can always upper bound $q_s$ with $q_s \leq L$.

on eq. (8) first. Looking at table 1, we know that $\mathcal{H}$ takes the role of $\mathcal{M}'$, and we know that the running time of an adversary is at least the number of oracle queries $q$. Therefore, we need to satisfy that $(q/|\mathcal{H}|)/q \leq 2^{-(k_C+\log 5+2w+\log L+\log v)}$, or equivalently that $\log|\mathcal{H}| \geq k_C + \log 5 + 2w + \log L + \log v$. Now, continue with eq. (9). The bound consists of two terms, the first one being $(q+1)/|\mathcal{H}|$. and the second one being (almost) equal to the term in the undetectability bound. We lower bound the running time with $q+1$, and we want each of the terms to be at most $2^{-(k_C+\log 5+w+\log L+\log v+1)}$. As $w \geq 1$, this follows already from the lower bound on $\log|\mathcal{H}|$ we have derived from eq. (8). Next, focus on eqs. (6) and (7). From table 1, we get that it is sufficient to ensure that

$$\left(\frac{2q+1}{|\mathcal{H}|} + \frac{2q}{|\mathcal{P}|}\right) \Big/ q \leq 2^{-(k_C+\log 5+1)}.$$

If we upper bound $2q+1$ with $2 \cdot \mathbf{T}(\mathcal{A})$, then we see that the requirement on $|\mathcal{H}|$ we have so far already ensures that the first term $2/|\mathcal{H}|$ is at most $2^{-(k_C+\log 5+2)}$ for $L \geq 2, v \geq 1$. We also want to make the second term $2/|\mathcal{P}|$ to be at most $2^{-(k_C+\log 5+2)}$, so we require $\log|\mathcal{P}| \geq k_C + \log 5 + 3$. We proceed in a similar way for the quantum setting, using the appropriate bounds from table 1. We summarize the requirements in the following.

**Parameter Requirement 3** (Hash and Parameter Length)**.** *Let $L$, $w$, $v$ be already given, and assume that $w \geq 1$, $L, v \geq 2$. Then, if we want (approximately) $k_C$ bits of classical security and $k_Q$ bits of quantum security, we need to satisfy the following:*

$$\log|\mathcal{H}| \geq \max\{k_C + \log 5 + 2w + \log L + \log v, 2(k_Q + \log 5 + 2w + \log L + \log v + \log 12)\}, \tag{15}$$

$$\log|\mathcal{P}| \geq \max\{k_C + \log 5 + 3, 2(k_Q + \log 5 + 2) + 5\}. \tag{16}$$

# 7 Instantiations of Tweakable Hash Functions

Our constructions require two tweakable hash functions $\mathsf{Th}$ and $\mathsf{Th}^{msg}$. To recall, $\mathsf{Th}$ takes three inputs $P \in \mathcal{P}$, $T \in \mathcal{T}$ and $M \in \mathcal{M}$, and outputs a hash in a space $\mathcal{H}$, where we need that $\mathcal{H}, \mathcal{H}^2, \mathcal{H}^v \subseteq \mathcal{M}$. The function $\mathsf{Th}^{msg}$ takes four inputs $P \in \mathcal{P}$, $T \in \mathcal{T}$, $M \in \{0,1\}^{l_{msg}}$, and[12] $R \in \mathcal{R}$. It outputs a list of integers in $\{0, \ldots, 2^w - 1\}$, of length $n_0$ for Construction 5 and length $v$ for Construction 6. We describe two possible instantiations. One uses the classical hash function SHA-3 which operates on bit strings. The other instance is optimized for modern non-interactive argument systems and uses the recent hash function Poseidon2 [KBM23], which operates on elements of a finite field $\mathbb{F}_p$. Throughout this section, $||$ denotes concatenation.

## 7.1 Tweak Functions

We start by giving a possible instantiation of the tweak functions (see Constructions 1 and 2). The first function $\mathsf{tweak}\colon [L] \times [v] \times [2^w - 1] \to \mathcal{T}$ is defined as

$$\mathsf{tweak}(\mathsf{ep}, i, k) = (\underbrace{0}_{8\text{ bits}} \,||\, \underbrace{\mathsf{ep}}_{\lceil \log L \rceil \text{ bits}} \,||\, \underbrace{i}_{\lceil \log v \rceil \text{ bits}} \,||\, \underbrace{k}_{w \text{ bits}}). \tag{17}$$

The second function $\mathsf{tweakmt}\colon [\log L] \times [L] \to \mathcal{T}$ is

$$\mathsf{tweakmt}(l, i) = (\underbrace{1}_{8\text{ bits}} \,||\, \underbrace{l}_{\lceil \log(\lceil \log L \rceil) \rceil \text{ bits}} \,||\, \underbrace{i}_{\lceil \log L \rceil \text{ bits}}). \tag{18}$$

[12]For simplicity, we write $\mathsf{Th}^{msg}(P, T, M, R)$ instead of $\mathsf{Th}^{msg}(P, T, (M, R))$ in this section.

The third function $\mathsf{tweakm}\colon [L] \to \mathcal{T}$ for message hashing (Constructions 5 and 6) is

$$\mathsf{tweakm}(\mathsf{ep}) = (\underbrace{2}_{8 \text{ bits}} \,||\, \underbrace{\mathsf{ep}}_{\lceil \log L \rceil \text{ bits}}) \tag{19}$$

It is clear that the ranges of all three functions are disjoint. One may use larger lengths if this is more convenient, e.g., encoding $\mathsf{ep}$ as one 64-bit integer if $L < 2^{64}$, as long as this is done consistently.

## 7.2 Tweakable Hash From SHA-3

SHA-3-256 [Nat15] is a hash function designed in 2007 and later standardized by NIST within the SHA-3 family. It maps an arbitrarily long bit string to a 256-bit output. We simply write SHA-3 for short. For this instantiation, we use $\mathcal{P} = \{0,1\}^{l_p}$, $\mathcal{T} = \{0,1\}^{l_t}$, $\mathcal{R} = \{0,1\}^{l_{rnd}}$ and $\mathcal{H} = \{0,1\}^n$. The message input $M$ in both $\mathsf{Th}$ and $\mathsf{Th}^{msg}$ is a bit string of some length $l_m$, where $l_m$ can take one of the following values depending on where the hash function is used:

- $l_m = vn$ to hash the leaf in Construction 1 with $\mathsf{Th} = \mathsf{Th}_{\text{SHA-3}}$ (to be used in Construction 3).
- $l_m = 2n$ to hash pairs of nodes in Construction 1 $\mathsf{Th} = \mathsf{Th}_{\text{SHA-3}}$ (to be used in Construction 3).
- $l_m = n$ in hash chains in Construction 2 $\mathsf{Th} = \mathsf{Th}_{\text{SHA-3}}$ (to be used in Construction 3).
- $l_m = l_{msg}$ for message hashing with $\mathsf{Th}^{msg} = \mathsf{Th}^{msg}_{\text{SHA-3}}$ in Constructions 5 and 6.

Below, we explain how $\mathsf{Th}^{msg}_{\text{SHA-3}}$ and $\mathsf{Th}_{\text{SHA-3}}$ are constructed.

### 7.2.1 Message Hashing

For message hashing, we define

$$\mathsf{Th}^{msg}_{\text{SHA-3}}(P, T, M, R) = \mathsf{Truncate}_{\ell w \text{ bits}} \left(\text{SHA-3}(R||P||T||M)\right) \in \{0, \ldots, 2^w - 1\}^\ell,$$

where $\ell = n_0$ for Construction 5 and $\ell = v$ for Construction 6, assuming $\ell w \leq 256$. Here, $\mathsf{Truncate}_{\nu \text{ bits}}$ takes first $\nu$ bits of the resulting bit string. Note that we do not add any domain separation for different spaces of keys or parameters. If this is needed one should prefix the hash input with some encoding of input spaces.

### 7.2.2 Chain, Leaf, and Tree Hashing

We then set

$$\mathsf{Th}_{\text{SHA-3}}(P, T, M) = \mathsf{Truncate}_{n \text{ bits}} \left(\text{SHA-3}(P||T||M)\right).$$

Note that the tweak value differs for all invocations of our tweakable hash functions so all SHA-3 calls actually get a different input. The input length is $l_p + l_t + l_m$ bits.

### 7.2.3 Resistance to Attacks

SHA-3-256 and its round-reduced versions have been targets of cryptanalytic attacks in the last decade. At the time of writing, no collision or preimage attack faster than exhaustive search is known for the full SHA-3-256. In particular, we are not aware of any attacks against the security notions (definitions 3 to 6) that we require.

## 7.3 Tweakable Hash From Poseidon2

Poseidon2 [GKS23] is a family of hash functions which are defined on various prime field domains. For each prime $p$ and integer $t \in \{4, 8, 12, 16, 20, 24\}$, Poseidon2 defines a bijective function (i.e., a permutation) $\mathsf{PoseidonPerm}_{p,t}$ on $\mathbb{F}_p^t$. A hash function is obtained via one of two modes:

- *Compression Mode.* We have

$$\mathsf{PoseidonCompress}_{p,t,u}(\mathbf{x}) = \mathsf{Truncate}_u(\mathsf{PoseidonPerm}_{p,t}(\mathbf{x}) + \mathbf{x}) \in \mathbb{F}_p^u,$$

  where $\mathsf{Truncate}_u$ takes first $u$ elements of the output, $\mathbf{x} \in \mathbb{F}^t$ and $+$ is elementwise addition in $\mathbb{F}_p^t$. This mode limits the input length to $t$ field elements but is the more efficient one.

- *Sponge Mode.* Here, $\mathsf{PoseidonPerm}$ is iteratively applied to a state, which is an element in $\mathbb{F}^t$, while simultaneously absorbing parts of the input. This mode is the most flexible at the expense of some computational overhead.

For the rest of this section, we assume that $\mathbb{F}_p$ is the prime field on which the circuits are constructed for aggregation proofs.

**Padding.** As $t \in \{4, 8, 12, 16, 20, 24\}$, we will need to pad some of our inputs with a vector $\mathbf{0}$ of zero field elements to increase its length to the next multiple of 4. Note that this only works if the input length is at most 24 field elements. In other cases, the sponge mode has to be used. Our description assumes parameter settings for which only leaf hashing requires the sponge mode.

**Classical Security.** To apply the heuristic bounds and use section 6 to set candidate parameters, we need to assume, as done in the design paper of Poseidon2, that $\mathsf{PoseidonCompress}_{p,t,u}$ behaves like a random oracle of the form $\mathbb{F}^t \to \mathbb{F}^u$ for all practical purposes, and up to $p^u$ permutation queries. Similarly, the Sponge mode with capacity $\mathsf{c}$ and rate $\mathsf{r}$, which outputs $u$ field elements needs to securely instantiate a random oracle mapping into $\mathbb{F}^u$ up to $\min(p^u, p^{\mathsf{c}/2})$ permutation queries [KBM23]. We emphasize again that this is only for getting heuristic candidate parameters, and security of the scheme ultimately relies on standard model assumptions about Poseidon2 with these parameters. We encourage any cryptanalytic effort to study Poseidon2 with regards to these standard model assumptions.

**Quantum Security.** Similarly, using our heuristics means that we need to assume that $\mathsf{PoseidonCompress}_{p,t,u}$ behaves sufficiently like a quantum random oracle of the form $\mathbb{F}^t \to \mathbb{F}^u$, for up to $p^{u/2}$ quantum queries. The security of the Sponge mode also degrades: we are only able to claim security up to $\min(p^{u/3}, p^{\mathsf{c}/3})$ permutation queries [Unr21].

**Additional Bounds.** To summarize, in addition to the bounds from section 6, we have to additionally satisfy the following bounds for the Poseidon2 parameters:

$$\textit{Compression Mode.} \quad u \log p \geq \max(k_C, 2k_Q), \tag{20}$$

$$\textit{Sponge Mode.} \quad u \log p \geq \max(k_C, 3k_Q), \quad \mathsf{c} \log p \geq \max(2k_C, 3k_Q). \tag{21}$$

Here, $k_C$ and $k_Q$ denote classical and quantum security levels as in section 6.

**Input Spaces.** The public parameter space $\mathcal{P}$ is defined as $\mathcal{P} = \mathbb{F}_p^{l_p}$, where $l_p$ is taken such that (16) is satisfied. The tweak space $\mathcal{T}$ is defined as $\mathcal{T} = \{0,1\}^{l_t}$ where $l_t$ is selected to accommodate the tweak values defined in eqs. (17) to (19). The seed space $\mathcal{R}$ is defined as $\mathcal{R} = \mathbb{F}_p^{l_p}$, where $l_p$ is taken such that eq. (12) and eq. (14), respectively, are satisfied, depending on whether Construction 5 or Construction 6 is used. The message space $\mathcal{M}$ is $\{0,1\}^{l_{msg}}$ for the message hash $\mathsf{Th}^{msg} = \mathsf{Th}^{msg}_{\text{Poseidon2}}$, and it is $\mathbb{F}_p^{l_m}$ for various $l_m$ for $\mathsf{Th} = \mathsf{Th}_{\text{Poseidon2}}$.

**Tweak Encoding.** So far, we have described tweaks as being bit strings. To use tweaks in Poseidon2, we need to encode them as vectors of field elements. This is done as follows, for a tweak $T \in \{0,1\}^{l_t}$:

1. Let $\xi$ be the minimum number such that $p^{\xi} > 2^{l_t}$.
2. Interpret $T$ as base-$p$ integer $A_T$.
3. Then $\mathsf{EncT}(T)$ is the equivalent representation of $A_T$ as a vector of $\xi$ elements of $\mathbb{F}_p$.

#### 7.3.1 Message Hashing

We now give more details on how to implement the message hash $\mathsf{Th}^{msg} = \mathsf{Th}^{msg}_{\text{Poseidon2}}$.

**Outputs.** For both instantiations (Constructions 5 and 6), we introduce an additional parameter $\eta'$, which models the number of field elements that the hash function outputs, before injectively mapping them to an output in $\{0,\ldots,2^w-1\}^{\ell}$, with $\ell = n_0$ for Construction 5 and $\ell = v$ for Construction 6. More precisely, focus on the instantiation based on Winternitz first (Construction 5). We set $\eta'$ to be the minimum such that $\eta' \log p$ exceeds the right hand side of (11). Note that this implies that it also exceeds the right hand side of (20). Then, we find the minimum $n_0$ such that $n_0 w \geq \eta' \log p$. Finally, we use an injective function $\mathsf{Decode}_{p,\eta',w}$ that interprets its input $\mathbb{F}_p^{\eta'}$ in as an integer in $\mathbb{Z}_{p^{\eta'}}$, and represents it in base $2^w$ to get a vector in $\{0,\ldots,2^w-1\}^{n_0}$. We proceed in a similar way for the target sum Winternitz instantiation (Construction 6), replacing $n_0$ with $v$ and (11) with (13).

**Message Encoding.** For message hashing, the input is a bit string in $\{0,1\}^{l_{msg}}$. As for tweaks, we need to encode this bit string as a vector of field elements first, which is done as follows:

1. Let $\chi$ be the minimum number such that $p^{\chi} > 2^{l_{msg}}$.
2. Interpret $M$ as base-$p$ integer $A_M$.
3. Then $\mathsf{EncM}(M)$ is the equivalent representation of $A_M$ as a vector of $\chi$ elements of $\mathbb{F}_p$.

**Hash Function.** The total input length is $l_{\textit{th-msg-in}} = l_p + \xi + \chi + l_{rnd}$. Let $t_{\textit{th-msg}}$ be minimal multiple of 4 that is not smaller than $l_{\textit{th-msg-in}}$. Then, we define the tweakable hash function for message hashing as

$$\begin{aligned}&\mathsf{Th}^{msg}_{\text{Poseidon2}}(P,T,M,R)\\&\qquad= \mathsf{Decode}_{p,\eta',w}(\mathsf{PoseidonCompress}_{p,t_{\textit{th-msg}},\eta'}(R||P||\mathsf{EncT}(T)||\mathsf{EncM}(M)||\mathbf{0})),\end{aligned}$$

where $\mathbf{0}$ represents a padding of $t_{\textit{th-msg}} - l_{\textit{th-msg-in}}$ zero field elements and can be empty, if we have $t_{\textit{th-msg}} = l_{\textit{th-msg-in}}$.

**On Uniformity.** Note that the mapping that we define in this way does not have a uniform output distribution. One may be concerned that this causes security issues. However, note that message hashing needs to satisfy only one security property, namely, multi-target collision resistance with random sampling (definition 6). If we set parameters as above, then our heuristic bounds apply to the output of $\mathsf{PoseidonCompress}$. As decoding is injective, this property is preserved.

This non-uniformity also has an impact on correctness of the signature scheme. Namely, formally applying $\epsilon$-uniformity (see definition 2) via lemma 7 would not yield a sufficient correctness bound. We do not claim any formal correctness guarantees when using the Poseidon2-based instantiation, but we note that in our experiments, the correctness error still seemed to be sufficiently small when setting the target sum as if the message hash were uniform.

### 7.3.2 Chain, Tree, and Leaf Hashing

For the tweakable hash function $\mathsf{Th} = \mathsf{Th}_{\text{Poseidon2}}$, we need to hash three types of inputs: (1) values within chains, i.e., values in $\mathcal{H}$, (2) pairs of nodes in the Merkle tree, i.e., values in $\mathcal{H}^2$, and (3) leafs, i.e., values in $\mathcal{H}^v$. We define $\mathcal{H}$ to be $\mathcal{H} := \mathbb{F}_p^\eta$, where $\eta$ is chosen large enough so that eq. (15) is satisfied. This also implies that $\eta \log p$ exceeds the right hand side of eq. (20). As we also use the sponge mode for $\mathsf{Th}_{\text{Poseidon2}}$, we need to respect eq. (21) as well, which means $\eta \log p$ must also exceed the right hand side of the first inequality in (21).

**Chain Hashing.** For (1), we use the compression mode, since in this case all inputs fit into 24 field elements if a 31-bit prime field is used, which is a convenient setting for $\mathsf{PoseidonPerm}$ within hash-based succinct arguments. Let $t_{\textit{th-ch}}$ be minimal multiple of 4 that is not smaller than $l_{\textit{th-ch-in}} = l_p + \xi + \eta$. We set

$$\mathsf{Th}_{\text{Poseidon2}}(P, T, M) = \mathsf{PoseidonCompress}_{p,t_{\textit{th-ch}},\eta}(P||\mathsf{EncT}(T)||M||\mathbf{0}) \text{ for } M \in \mathcal{H} = \mathbb{F}_p^\eta,$$

where $\mathbf{0}$ contains of $t_{\textit{th-ch}} - l_{\textit{th-ch-in}}$ zero field elements as before.

**Tree Hashing.** We now continue with (2), i.e., with hashing pairs of nodes within the Merkle tree. Each such node is the output of a previous hashing invocation, i.e., we now hash inputs in $\mathcal{H}^2$. Let $t_{\textit{th-tr}}$ be minimal multiple of 4 that is not smaller than $l_{\textit{th-tr-in}} = l_p + \xi + 2\eta$. We set

$$\mathsf{Th}_{\text{Poseidon2}}(P, T, M) = \mathsf{PoseidonCompress}_{p,l_p+\xi+2\eta,\eta}(P||\mathsf{EncT}(T)||M||\mathbf{0}) \text{ for } M \in \mathcal{H}^2 = \mathbb{F}_p^{2\eta},$$

where $\mathbf{0}$ contains of $t_{\textit{th-tr}} - l_{\textit{th-tr-in}}$ zero field elements and can be empty.

**Leaf Hashing.** We now turn to (3), where we need to hash long inputs in $\mathcal{H}^v$ as well, namely, when we hash the leafs in the Merkle tree, which correspond to $v$ ends of hash chains. To do that, we employ the sponge mode with the SAFE API [KBM23]. For the sponge mode, we first define the state size and the capacity $\mathsf{c}$ and rate $\mathsf{r}$, measured in the number of state elements and satisfying eq. (21). As we have already mentioned above, the output length $\eta$ is selected respecting eq. (21). We then take a reasonable value for $\mathsf{r}$; for a 31-bit field we set $\mathsf{r} = 24 - \mathsf{c}$. Then, we define $\mathsf{Th}_{\text{Poseidon2}}(P, T, M)$ as follows, for input $M \in \mathcal{H}^v = \mathbb{F}_p^{v\eta}$:

1. Produce the capacity value $V_c := \mathsf{PoseidonCompress}_{p,24,\mathsf{c}}(l_p||l_t||v||\eta) \in \mathbb{F}_p^{\mathsf{c}}$, where $l_p, l_t, v, \eta$ are interpreted as 32-bit values. Their 128-bit concatenation $l_p||l_t||v||\eta$ is interpreted as an element of $\mathbb{F}_p^{24}$ using the base-$p$ representation.
2. Pad $P||\mathsf{EncT}(T)||M$ with (possibly zero) field elements $0 \in \mathbb{F}_p$ so that the resulting vector $V$ has $\mathsf{r} \cdot s$ elements for some $s$, i.e., $V = (v_0, v_1, \dots, v_{\mathsf{r}\cdot s-1})$.
3. Set $S := (\underbrace{0, 0, \dots, 0}_{\mathsf{r}\text{ elements}}, V_c)$.
4. For $i$ from 1 to $s$:
   (a) $S := S + (v_{\mathsf{r}\cdot i}, v_{\mathsf{r}\cdot i+1}, \dots, v_{\mathsf{r}\cdot i+\mathsf{r}-1}, \underbrace{0, 0, \dots, 0}_{\mathsf{c}\text{ elements}})$ where addition is componentwise.
   (b) $S := \mathsf{PoseidonPerm}_{p,24}(S)$
5. Output $\mathsf{Truncate}_\eta(S)$.

Note that the parameter $M$ is always much bigger than its analogue in chain and tree hashing, which makes all three functions distinct.

#### 7.3.3 Resistance to Attacks

Poseidon2 (from 2023) is a relatively recent design. Together with Poseidon [GKR+21] (from 2019) it has been the subject of active cryptanalysis, but no attack has been published on any full version of Poseidon or Poseidon2. We thus expect that the security notions (definitions 3 to 6) that we require hold for Poseidon2. A recent initiative aims to further asses the security of Poseidon2[13].

# 8 Efficiency

In this section, we compare the schemes we have analyzed in terms of efficiency. We consider the schemes obtained from instantiating the generalized XMSS framework (Construction 3) with Construction 5 and Construction 6, for different parameters satisfying the requirements in section 6, and for the instantiations of hash functions as in section 7.

*Remark* 8. We only present a preliminary set of benchmarks. For example, for now we do not benchmark aggregation times using state-of-the-art pqSNARK implementations. Such benchmarks will be important before our proposed schemes can be used in Ethereum.

## 8.1 Setup

We first describe which schemes we compare, which metrics we consider, and how we obtain our results. We set all parameters following section 6 using security levels $k_C = 128$ (classical) and $k_Q = 64$ (quantum). This corresponds to NIST's Level 1 requirements [Nat16]. A justification for this is that attacking the scheme with Grover's algorithm [Gro96] requires about $2^{k_Q}$ *sequential time* (as opposed to work), as Grover's algorithm does not parallelize well [Zal99, Flu17].

**Constructions.** The constructions we compare use chunk sizes $w \in \{1, 2, 4, 8\}$. For Construction 6, we set the target sum to $T = \lceil \delta E \rceil$, where $E = v(2^w - 1)/2$ would be the expected sum if the message hash was uniform[14]. We consider cases $\delta = 1$ and $\delta = 1.1$. We consider key lifetimes $L = 2^{18}$ and $L = 2^{20}$. Note that longer lifetimes (e.g., $L = 2^{32}$) are desirable, but benchmarking those requires more engineering effort, in particular as the secret key and Merkle tree would no longer fit into main memory[15]. We leave benchmarking such longer lifetimes for future work. For the target sum encoding, we have assumed $K \leq 4096$ to set parameters. For instantiations based on Poseidon2, we assume a 31-bit field. For all constructions, we determine the remaining parameters following sections 6 and 7 using a Python script. The script can be found in the following repository:

https://github.com/b-wagn/hashsig-parameters

The Python script also determines the signature size as well as the worst-case and average-case hash complexity of verification, which impacts aggregation time.

**Implementation and Running Times.** To evaluate the computational efficiency, we have created a prototype Rust implementation of the signature schemes analyzed in this work. In particular, our implementation follows the abstractions used in this work and instantiations use the parameters determined using the Python script. It can be found in the following repository:

https://github.com/b-wagn/hash-sig

[13] See https://www.poseidon-initiative.info/.

[14] Of course, the message hash is not uniform.

[15] One can deal with this in many ways, e.g., by first computing half of the Merkle tree, saving it to disk, then computing the other half, and so on. Another approach, which requires further investigation, is to use a multi-tree version of our variants of XMSS, similar to [HRB13].

We benchmark this implementation with Criterion[16] on a MacBook Pro with Apple M3 Pro chip, 18 GB memory. We have not implemented any parallelization.

Table 2: Comparison of instantiations of our generalized XMSS with different incomparable encoding schemes, all using SHA-3-256. We compare instantiations based on classical Winternitz (Construction 5, denoted by W) and Target Sum Winternitz (Construction 6, denoted by TSW), with different parameters. We compare running times, signature size, and verification hash complexity (worst-case: WC, average-case: AC). Average-case hashing has been determined via simulation. Signature size is given in KiB (1 KiB = 1024 Bytes), hashing is given in words (1 word = 32 Bytes). For TSW, we set the target sum to $T = \lceil \delta v(2^w - 1)/2 \rceil$.

| | **Encoding** | **Parameters** | **Gen** [s] | **Sign** [$\mu$s] | **Ver** [$\mu$s] | **Sig** [KiB] | **Hash AC** [w] | **Hash WC** [w] |
|---|---|---|---|---|---|---|---|---|
| **Lifetime** $L = 2^{18}$ | W | $w = 1$ | 17.27 | 44.93 | 25.27 | 4.17 | 288.47 | 407.28 |
| | W | $w = 2$ | 17.27 | 38.91 | 30.01 | 2.31 | 288.95 | 464.91 |
| | W | $w = 4$ | 33.54 | 65.90 | 65.78 | 1.47 | 576.54 | 1021.66 |
| | W | $w = 8$ | 273.44 | 493.35 | 542.74 | 1.06 | 4644.38 | 8393 |
| | TSW | $w = 1, \delta = 1$ | 16.44 | 48.51 | 24.04 | 3.98 | 274.38 | 274.38 |
| | TSW | $w = 1, \delta = 1.1$ | 16.50 | 59.35 | 22.19 | 3.98 | 261.38 | 261.38 |
| | TSW | $w = 2, \delta = 1$ | 16.35 | 44.84 | 28.72 | 2.22 | 276.62 | 276.62 |
| | TSW | $w = 2, \delta = 1.1$ | 16.39 | 54.79 | 26.37 | 2.22 | 258.75 | 258.75 |
| | TSW | $w = 4, \delta = 1$ | 31.16 | 83.08 | 59.64 | 1.39 | 522.44 | 522.44 |
| | TSW | $w = 4, \delta = 1.1$ | 31.17 | 100.85 | 54.25 | 1.39 | 477.72 | 477.72 |
| | TSW | $w = 8, \delta = 1$ | 244.68 | 675.19 | 464.75 | 1.01 | 4008.53 | 4008.53 |
| | TSW | $w = 8, \delta = 1.1$ | 244.82 | 784.85 | 419.14 | 1.01 | 3613.22 | 3613.22 |
| **Lifetime** $L = 2^{20}$ | W | $w = 1$ | 69.37 | 44.91 | 25.68 | 4.22 | 293.04 | 411.91 |
| | W | $w = 2$ | 68.64 | 39.17 | 30.41 | 2.46 | 301.39 | 480.09 |
| | W | $w = 4$ | 134.28 | 65.94 | 66.48 | 1.52 | 583.14 | 1026.41 |
| | W | $w = 8$ | 1091.25 | 491.08 | 540.62 | 1.11 | 4655.74 | 8398 |
| | TSW | $w = 1, \delta = 1$ | 65.70 | 48.91 | 24.31 | 4.03 | 279 | 279 |
| | TSW | $w = 1, \delta = 1.1$ | 65.79 | 59.41 | 22.63 | 4.03 | 266 | 266 |
| | TSW | $w = 2, \delta = 1$ | 65.18 | 44.89 | 29.15 | 2.36 | 288.12 | 288.12 |
| | TSW | $w = 2, \delta = 1.1$ | 65.06 | 54.77 | 26.82 | 2.36 | 269.91 | 269.91 |
| | TSW | $w = 4, \delta = 1$ | 124.52 | 82.89 | 59.69 | 1.44 | 527.19 | 527.19 |
| | TSW | $w = 4, \delta = 1.1$ | 124.54 | 100.69 | 54.62 | 1.44 | 482.47 | 482.47 |
| | TSW | $w = 8, \delta = 1$ | 978.97 | 673.45 | 465.53 | 1.06 | 4013.53 | 4013.53 |
| | TSW | $w = 8, \delta = 1.1$ | 979.15 | 792.64 | 420.59 | 1.06 | 3618.22 | 3618.22 |

## 8.2 Results

Now, we discuss the results, which we present in tables 2 and 3. In particular, we discuss several trade-offs, and how various parameters impact the efficiency of the schemes.

**Impacts of Lifetime.** The lifetime $L$ has a linear impact on the running time of key generation, while the time required for signing and verification is almost unaffected. On the other hand, its impact on signature size and hashing is minimal, as only the Merkle path changes slightly, along with minor parameter adjustments (see eq. (15)). We note again that supporting large $L$ results in challenges when it comes to memory management, as the Merkle tree would not fit in memory.

**Impacts of Chunk Sizes.** The chain length increases exponentially with the chunk size $w$, while the number of chains $v$ only decreases linearly. Thus, increasing $w$ reduces the signature size linearly as fewer chains are needed. However, verifier hashing and running times are determined by chain length. This highlights a trade-off between signature size, computational efficiency, and verifier hashing. The values $w = 2$ and $w = 4$ offer the best balance. In contrast, $w = 1$ results in large signatures, while $w = 8$ is computationally inefficient and hash-inefficient due to very long chains.

[16] See https://docs.rs/criterion/latest/criterion/.

Table 3: Comparison of instantiations of our generalized XMSS with different incomparable encoding schemes, all using Poseidon2. We compare instantiations based on Winternitz (Construction 5, denoted by W) and Target Sum Winternitz (section 5.2, denoted by TSW), with different parameters. We compare running times, signature size, and verification hash complexity (worst-case: WC, average-case: AC). For hashing, we count how often the Poseidon permutation has to be called, and denote the permutation of width $t$ field elements by $\pi_t$. Average-case hashing has been determined via simulation. Signature size is given in KiB (1 KiB = 1024 Bytes). For TSW, we set the target sum to $T = \lceil \delta v(2^w - 1)/2 \rceil$.

| | Encoding | Parameters | **Gen** [s] | **Sign** [$\mu$s] | **Ver** [$\mu$s] | **Sig** [KiB] | $\pi_{16}$ **AC** | $\pi_{24}$ **AC** | $\pi_{16}$ **WC** | $\pi_{24}$ **WC** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Lifetime $L = 2^{18}$** | W | $w = 1$ | 179.01 | 362.59 | 416.54 | 4.97 | 81 | 97 | 158 | 97 |
| | W | $w = 2$ | 168.19 | 350.04 | 408.67 | 2.75 | 122 | 59 | 237 | 59 |
| | W | $w = 4$ | 330.52 | 638.08 | 769.41 | 1.66 | 325 | 41 | 615 | 41 |
| | W | $w = 8$ | 2717.28 | 4820 | 5820 | 1.11 | 2917 | 31 | 5355 | 31 |
| | TSW | $w = 1, \delta = 1$ | 172.67 | 541.45 | 396.56 | 4.75 | 77 | 93 | 77 | 93 |
| | TSW | $w = 1, \delta = 1.1$ | 172.29 | 898.22 | 376.62 | 4.75 | 69 | 93 | 69 | 93 |
| | TSW | $w = 2, \delta = 1$ | 166.51 | 530.83 | 372.93 | 2.65 | 117 | 57 | 117 | 57 |
| | TSW | $w = 2, \delta = 1.1$ | 166.22 | 888.55 | 351.37 | 2.65 | 105 | 57 | 105 | 57 |
| | TSW | $w = 4, \delta = 1$ | 312.49 | 1090.00 | 650.82 | 1.58 | 292 | 39 | 292 | 39 |
| | TSW | $w = 4, \delta = 1.1$ | 312.64 | 1670.00 | 602.75 | 1.58 | 263 | 39 | 263 | 39 |
| | TSW | $w = 8, \delta = 1$ | 2501.01 | 9760.00 | 4900.00 | 1.06 | 2550 | 30 | 2550 | 30 |
| | TSW | $w = 8, \delta = 1.1$ | 2499.97 | 14570.00 | 4320.00 | 1.06 | 2295 | 30 | 2295 | 30 |
| **Lifetime $L = 2^{20}$** | W | $w = 1$ | 780.89 | 362.44 | 418.31 | 5.03 | 82 | 99 | 158 | 99 |
| | W | $w = 2$ | 705.42 | 336.30 | 400.60 | 2.81 | 122 | 61 | 237 | 61 |
| | W | $w = 4$ | 1353.18 | 617.48 | 746.28 | 1.72 | 326 | 43 | 615 | 43 |
| | W | $w = 8$ | 11122.95 | 4981.20 | 6039.40 | 1.34 | 2917 | 35 | 5355 | 35 |
| | TSW | $w = 1, \delta = 1$ | 752.57 | 520.42 | 401.32 | 4.81 | 77 | 95 | 77 | 95 |
| | TSW | $w = 1, \delta = 1.1$ | 731.79 | 844.01 | 381.23 | 4.81 | 69 | 95 | 69 | 95 |
| | TSW | $w = 2, \delta = 1$ | 667.76 | 527.17 | 379.56 | 2.7 | 117 | 59 | 117 | 59 |
| | TSW | $w = 2, \delta = 1.1$ | 668.14 | 853.66 | 354.09 | 2.7 | 105 | 59 | 105 | 59 |
| | TSW | $w = 4, \delta = 1$ | 1249.52 | 1057.40 | 661.61 | 1.64 | 292 | 41 | 292 | 41 |
| | TSW | $w = 4, \delta = 1.1$ | 1248.35 | 1600.00 | 603.65 | 1.64 | 263 | 41 | 263 | 41 |
| | TSW | $w = 8, \delta = 1$ | 9972.32 | 9509.50 | 4870.60 | 1.27 | 2550 | 34 | 2550 | 34 |
| | TSW | $w = 8, \delta = 1.1$ | 9927.97 | 14271.00 | 4358.60 | 1.27 | 2295 | 34 | 2295 | 34 |

**Winternitz vs. Target Sum.** Let us now compare the classical Winternitz instantiation (W in table 2) and the target sum instantiation (TSW in table 2). When it comes to key generation time, the classical Winternitz instantiation is slower due to the additional chains required for the checksum. In terms of signing time, the target sum instantiation is slower because retries are necessary, until the sum matches the target sum. For verification, the classical Winternitz instantiation is again slower, and we see that it has larger signatures and hashing complexities. Both is mostly due to the larger number of chains. Moreover, it is evident that for the target sum instantiation, the average-case and worst-case hashing complexities are identical. This highlights that we have an explicit control over hashing complexity in this variant. Therefore, if one can afford a slight increase in signing time, the target sum instantiation is clearly preferable.

**Signing Time vs. Verifier Hashing.** For the target sum instantiation (TSW in table 2), we see that signing time can be traded off against verifier hashing by increasing the target sum. Concretely, compare any two consecutive lines in table 2 with the same chunk size $w$ and $\delta = 1$ versus $\delta = 1.1$. We can observe that signing time increases for $\delta = 1.1$ as more retries are needed, while verification time and (verification) hashing complexity decrease.

**Impacts of Hash Functions.** When comparing tables 2 and 3, we observe that Poseidon2-based instantiations are significantly slower than their SHA-3-based counterparts (concretely, a factor of about 10). Additionally, signature sizes are generally slightly larger for Poseidon2-based instantiations. This is primarily because the hash function outputs are vectors of field elements (31 bits) rather than vectors of bytes (8 bits), resulting in less fine-grained control over their length.

### 8.3 On Aggregation via Succinct Arguments

While we do not provide concrete benchmarks for signature aggregation using pqSNARKs, we give a high-level discussion on the topic. We emphasize that the following estimates are preliminary.

**Candidates for pqSNARKs.** There are two main approaches to implementing a pqSNARK for signature aggregation:

1. *Custom Circuit Approach.* One could design and optimize a dedicated circuit and employ a hash-based argument, e.g., via post-quantum instances of the *Plonky3 framework*[17] or *stwo*[18]. Ideally, these circuits are formally verified before use in Ethereum.

2. *zkVM-Based Approach.* Alternatively, one could utilize zkVMs, which can generically prove the verifier's code. This approach simplifies the process of writing an Ethereum specification and is less error-prone. However, it comes at the cost of reduced efficiency.

Regardless of the choice, any selected pqSNARK must be adaptively knowledge-sound (see section 4.3).

**Aggregate Signature Size.** Our preliminary estimates suggest that the aggregate signature size using Plonky3 (which employs FRI [BBHR18]) ranges from 2 MB to 3 MB. This means as soon as we have more than 1000 signatures, aggregation saves space. These estimates do not leverage algebraic conjectures commonly used to improve efficiency. Incorporating such conjectures could further reduce the aggregate signature size. Moreover, pqSNARKs continue to evolve, promising additional improvements. For example, replacing FRI with STIR [ACFY24a] or WHIR [ACFY24b] could significantly shrink argument size. The STIR paper suggests potential reductions by a factor of 2.5, indicating that, when combined with algebraic conjectures, aggregate signature sizes below 1 MB are achievable.

**Aggregation Times.** Assuming that verifying a single signature requires approximately 160 hash operations (or Poseidon2 permutations), which corresponds to $\mathsf{TSW}$ with parameters $w = 2, \delta = 1.1$, we estimate that aggregating up to $10,000$ signatures within one second is feasible. This estimation is based on the requirement to prove approximately $1.75 \cdot 10^6$ hashes per second, a performance goal that appears attainable given current advancements in pqSNARKs.

## 9 Conclusion

In this work, we have presented and analyzed variants of XMSS signatures. We have taken care to obtain a security analysis leading to efficient and theoretically sound parameters, and relying on explicitly stated standard model properties of the underlying hash functions. In combination with a pqSNARK, we view our schemes as a family of proposals for use in post-quantum Ethereum. The defining features of our schemes are their conceptual simplicity, reliance solely on hash functions, and the rigorous theoretical analysis supporting them. Although we have not discussed specific instantiations of the pqSNARK, our work complements the broader industry efforts to develop efficient and secure pqSNARKs. That said, we emphasize that reasonable alternatives exist and merit further investigation, and we refer back to section 2 for a comprehensive discussion.

One key takeaway from our study is that the pqSNARK used to aggregate signatures must be *adaptively* knowledge-sound. Another important contribution is our characterization of the security properties and levels required of hash functions (e.g., Poseidon2) for

[17] See `https://github.com/plonky3/plonky3`.
[18] See `https://github.com/starkware-libs/stwo`.

our proposed schemes. These properties provide concrete targets for cryptanalysis and further research. In particular, we encourage cryptanalysts to study hash functions like Poseidon2 with regards to the notions we use.

## References


[AAB+24] Marius A. Aardal, Diego F. Aranha, Katharina Boudgoust, Sebastian Kolby, and Akira Takahashi. Aggregating falcon signatures with LaBRADOR. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 71–106. Springer, Cham, August 2024. doi:10.1007/978-3-031-68376-3_3.

[ACFY24a] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: Reed-solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 380–413. Springer, Cham, August 2024. doi:10.1007/978-3-031-68403-6_12.

[ACFY24b] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed–solomon proximity testing with super-fast verification. Cryptology ePrint Archive, Paper 2024/1586, 2024. URL: https://eprint.iacr.org/2024/1586.

[ACL+22] Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable - (extended abstract). In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 102–132. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_4.

[ADP24] Nabil Alkeilani Alkadri, Nico Döttling, and Sihang Pu. Practical lattice-based distributed signatures for a small number of signers. In Christina Pöpper and Lejla Batina, editors, *ACNS 24International Conference on Applied Cryptography and Network Security, Part I*, volume 14583 of *LNCS*, pages 376–402. Springer, Cham, March 2024. doi:10.1007/978-3-031-54770-6_15.

[AdSGK24] Shahla Atapoor, Cyprien Delpech de Saint Guilhem, and Al Kindi. STARK-based signatures from the RPO permutation. Cryptology ePrint Archive, Paper 2024/1553, 2024. URL: https://eprint.iacr.org/2024/1553.

[AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 473–484. ACM Press, October 2010. doi:10.1145/1866307.1866360.

[BBd+23] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP*

*2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018. doi:10.4230/LIPIcs.ICALP.2018.14.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 252–283. Springer, Cham, August 2023. doi:10.1007/978-3-031-38545-2_9.

[BC24] Dan Boneh and Binyi Chen. LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Paper 2024/257, 2024. URL: https://eprint.iacr.org/2024/257.

[BCJP24] Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part IV*, volume 14654 of *LNCS*, pages 168–195. Springer, Cham, May 2024. doi:10.1007/978-3-031-58737-5_7.

[BDE+11] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the Winternitz one-time signature scheme. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11*, volume 6737 of *LNCS*, pages 363–378. Springer, Berlin, Heidelberg, July 2011. doi:10.1007/978-3-642-21969-6_23.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Berlin, Heidelberg, December 2011. doi:10.1007/978-3-642-25385-0_3.

[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. Springer, Berlin, Heidelberg, November / December 2011. doi:10.1007/978-3-642-25405-5_8.

[Beu22] Ward Beullens. MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 355–376. Springer, Cham, September / October 2022. doi:10.1007/978-3-030-99277-4_17.

[BH19] Daniel J. Bernstein and Andreas Hülsing. Decisional second-preimage resistance: When does SPR imply PRE? In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 33–62. Springer, Cham, December 2019. doi:10.1007/978-3-030-34618-8_2.

[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.

[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.

[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.

[BK20] Dan Boneh and Sam Kim. One-time and interactive aggregate signatures from lattices, 2020. URL: https://crypto.stanford.edu/~skim13/agg_ots.pdf.

[BKPV23] Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier Verbel. Biscuit. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[BS20] Dan Boneh and Victor Shoup. *A graduate course in applied cryptography*. 2020. URL: https://toc.cryptobook.us/.

[BS23] Ward Beullens and Gregor Seiler. LaBRADOR: Compact proofs for R1CS from module-SIS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 518–548. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_17.

[BTT22] Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. MuSig-L: Lattice-based multi-signature with single-round online phase. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 276–305. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_10.

[CF24] Alessandro Chiesa and Giacomo Fenzi. zkSNARKs in the ROM with unconditional UC-security. In *Theory of Cryptography Conference*, pages 67–89. Springer, 2024. doi:10.1007/978-3-031-78011-0_3.

[CFS01] Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a McEliece-based digital signature scheme. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 157–174. Springer, Berlin, Heidelberg, December 2001. doi:10.1007/3-540-45682-1_10.

[Che23] Yanbo Chen. DualMS: Efficient lattice-based two-round multi-signature with trapdoor-free simulation. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 716–747. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_23.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 1–29. Springer, Cham, December 2019. doi:10.1007/978-3-030-36033-7_1.

[DFMS24] Giuseppe D'Alconzo, Andrea Flamini, Alessio Meneghetti, and Edoardo Signorini. A framework for group action-based multi-signatures and applications to LESS, MEDS, and ALTEQ. Cryptology ePrint Archive, Paper 2024/1691, 2024. URL: https://eprint.iacr.org/2024/1691.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd FOCS*, pages 1057–1068. IEEE Computer Society Press, October / November 2022. doi:10.1109/FOCS54457.2022.00103.

[DGNW20] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2093–2110. USENIX Association, August 2020.

[DHSS20] Yarkın Doröz, Jeffrey Hoffstein, Joseph H. Silverman, and Berk Sunar. MMSAT: A scheme for multimessage multiuser signature aggregation. Cryptology ePrint Archive, Report 2020/520, 2020. URL: https://eprint.iacr.org/2020/520.

[DKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 64–93. Springer, Cham, December 2020. doi:10.1007/978-3-030-64837-4_3.

[DLL+17] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633, 2017. URL: https://eprint.iacr.org/2017/633.

[DLRW24] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. SQIsignHD: New dimensions in cryptography. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 3–32. Springer, Cham, May 2024. doi:10.1007/978-3-031-58716-0_1.

[DOTT21] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 99–130. Springer, Cham, May 2021. doi:10.1007/978-3-030-75245-3_5.

[DSS05] C. Dods, Nigel P. Smart, and Martijn Stam. Hash based digital signature schemes. In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 96–115. Springer, Berlin, Heidelberg, December 2005. doi:10.1007/11586821_8.

[FHSZ23] Nils Fleischhacker, Gottfried Herold, Mark Simkin, and Zhenfei Zhang. Chipmunk: Better synchronized multi-signatures from lattices. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 386–400. ACM Press, November 2023. doi:10.1145/3576915.3623219.

[FKNP24] Giacomo Fenzi, Christian Knabenhans, Ngoc Khanh Nguyen, and Duc Tu Pham. Lova: Lattice-based folding scheme from unstructured lattices. Cryptology ePrint Archive, Paper 2024/1964, 2024. URL: https://eprint.iacr.org/2024/1964.

[Flu17] Scott Fluhrer. Reassessing grover's algorithm. Cryptology ePrint Archive, Report 2017/811, 2017. URL: https://eprint.iacr.org/2017/811.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[FSZ22] Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. Squirrel: Efficient synchronized multi-signatures from lattices. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1109–1123. ACM Press, November 2022. doi:10.1145/3548606.3560655.

[GHHM21] Alex B. Grilo, Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Tight adaptive reprogramming in the QROM. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 637–667. Springer, Cham, December 2021. doi:10.1007/978-3-030-92062-3_22.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the poseidon hash function. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 177–203. Springer, Cham, July 2023. doi:10.1007/978-3-031-37679-5_8.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. doi:10.1145/1374376.1374407.

[GR06] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 257–273. Springer, Berlin, Heidelberg, April 2006. doi:10.1007/11745853_17.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC*, pages 212–219. ACM Press, May 1996. doi:10.1145/237814.237866.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. doi:10.1145/1993636.1993651.

[HBD+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[HBG+18] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018. URL: https://www.rfc-editor.org/info/rfc8391, doi:10.17487/RFC8391.

[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.

[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.

[HLP24] Ulrich Haböck, David Levit, and Shahar Papini. Circle STARKs. Cryptology ePrint Archive, Report 2024/278, 2024. URL: https://eprint.iacr.org/2024/278.

[HRB13] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal parameters for XMSS MT. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar R. Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*, volume 8128 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2013. doi:10.1007/978-3-642-40588-4_14.

[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.

[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.

[HW18] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 197–229. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_7.

[KBM23] Dmitry Khovratovich, Mario Marhuenda Beltrán, and Bart Mennink. Generic security of the SAFE API and its applications. In *ASIACRYPT (8)*, volume 14445 of *Lecture Notes in Computer Science*, pages 301–327. Springer, 2023. doi:10.1007/978-981-99-8742-9_10.

[KCLM22] Irakliy Khaburzaniya, Konstantinos Chalkias, Kevin Lewi, and Harjasleen Malvai. Aggregating and thresholdizing hash-based signatures using STARKs. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 393–407. ACM Press, May / June 2022. doi:10.1145/3488932.3524128.

[KKF21] Mikhail Aleksandrovich Kudinov, Evgeniy Olegovich Kiktenko, and Aleksey Konstantinovich Fedorov. Security analysis of the w-ots$^+$ signature scheme: Updating security bounds. *Matematicheskie Voprosy Kriptografii [Mathematical Aspects of Cryptography]*, 12(2):129–145, June 2021. URL: http://dx.doi.org/10.4213/mvk362, doi:10.4213/mvk362.

[KLM06] Phillip Kaye, Raymond Laflamme, and Michele Mosca. An introduction to quantum computing. Oxford University Press, 2006.

[LDK+20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[LLL+24] Qiqi Lai, Feng-Hao Liu, Yang Lu, Haiyang Xue, and Yong Yu. Scalable two-round $n$-out-of-$n$ and multi-signatures from lattices in the quantum random oracle model. Cryptology ePrint Archive, Paper 2024/1574, 2024. URL: https://eprint.iacr.org/2024/1574.

[LM08] Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 37–54. Springer, Berlin, Heidelberg, March 2008. doi:10.1007/978-3-540-78524-8_3.

[LMQW22] Alex Lombardi, Ethan Mook, Willy Quach, and Daniel Wichs. Post-quantum insecurity from LWE. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 3–32. Springer, Cham, November 2022. doi:10.1007/978-3-031-22318-1_1.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. Stanford university, 1979.

[Nat15] National Institute of Standards and Technology. SHA-3 Standard: Permutation-based hash and extendable-output functions. *Federal Information Processing Standards Publication (FIPS)*, 2015.

[Nat16] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. *National Institute of Standards and Technology Reports*, 2016. available at https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. URL: https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[PFH+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[SEMR24] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. AprèsSQI: Extra fast verification for SQIsign using extension-field signing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 63–93. Springer, Cham, May 2024. doi:10.1007/978-3-031-58716-0_3.

[Ste94] Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 13–21. Springer, Berlin, Heidelberg, August 1994. doi:10.1007/3-540-48329-2_2.

[TS23] Toi Tomita and Junji Shikata. Compact aggregate signature from module-lattices. Cryptology ePrint Archive, Report 2023/471, 2023. URL: https://eprint.iacr.org/2023/471.

[Unr17] Dominique Unruh. Post-quantum security of Fiat-Shamir. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 65–95. Springer, Cham, December 2017. doi:10.1007/978-3-319-70694-8_3.

[Unr21] Dominique Unruh. Compressed permutation oracles (and the collision-resistance of sponge/SHA3). Cryptology ePrint Archive, Paper 2021/062, 2021. URL: https://eprint.iacr.org/2021/062.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_15.

[Zal99] Christof Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, Oct 1999. doi:10.1103/PhysRevA.60.2746.

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 138–169. Springer, Cham, August 2024. doi:10.1007/978-3-031-68403-6_5.

[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.

[Zha12] Mark Zhandry. How to construct quantum random functions. In *53rd FOCS*, pages 679–687. IEEE Computer Society Press, October 2012. doi:10.1109/FOCS.2012.37.


# Supplementary Material

# A Postponed Definitions

## A.1 Multi-Signatures

**Definition 12** (Synchronized Multi-Signature Scheme)**.** Let $L \in \mathbb{N}$ be a natural number. A synchronized (non-interactive) multi-signature scheme with lifetime $L$ is a tuple of efficient algorithms $\mathsf{MS} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Aggregate}, \mathsf{Ver})$ with the following syntax:

- $\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$ takes as input system parameters $\mathsf{par}$ and outputs a public key $\mathsf{pk}$ and a secret key $\mathsf{sk}$.
- $\mathsf{Sig}(\mathsf{sk}, \mathsf{ep}, \mathsf{m}) \to \sigma$ takes as input a secret key $\mathsf{sk}$, an epoch $\mathsf{ep} \in [L]$, and a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$ and outputs a signature $\sigma$.

- $\mathsf{Aggregate}(\mathsf{ep}, \mathsf{m}, ((\mathsf{pk}_i, \sigma_i))_{i=1}^k) \to \bar{\sigma}$ is deterministic, takes as input an epoch $\mathsf{ep} \in [L]$, a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$, and a list of public keys and signatures $(\mathsf{pk}_i, \sigma_i)$, and outputs an aggregate signature $\bar{\sigma}$.
- $\mathsf{Ver}((\mathsf{pk}_i)_{i=1}^k, \mathsf{ep}, \mathsf{m}, \bar{\sigma}) \to b$ is deterministic, takes as input a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$, an epoch $\mathsf{ep} \in [L]$, a message $\mathsf{m} \in \{0,1\}^{l_{msg}}$, and an aggregate signature $\bar{\sigma}$, and outputs a bit $b \in \{0,1\}$.

Further, we say that $\mathsf{MS}$ has correctness error at most $\delta \colon \mathbb{N} \to \mathbb{R}$, if for all $k \in \mathbb{N}$, all $(\mathsf{pk}_i, \mathsf{sk}_i) \in \mathsf{Gen}(\mathsf{par})$ for $i \in [k]$, all epochs $\mathsf{ep} \in [L]$, and all messages $\mathsf{m} \in \{0,1\}^{l_{msg}}$, we have

$$\Pr\left[\mathsf{Ver}((\mathsf{pk}_i)_{i=1}^k, \mathsf{ep}, \mathsf{m}, \bar{\sigma}) = 0 \;\middle|\; \begin{array}{c} \forall i \in [k]: \; \sigma_i \leftarrow \mathsf{Sig}(\mathsf{sk}_i, \mathsf{ep}, \mathsf{m}), \\ \bar{\sigma} \leftarrow \mathsf{Aggregate}(\mathsf{ep}, \mathsf{m}, ((\mathsf{pk}_i, \sigma_i))_{i=1}^k) \end{array}\right] \leq \delta(k).$$

**Definition 13** (Synchronized Multi-Signature Security)**.** Let $\mathsf{MS} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Aggregate}, \mathsf{Ver})$ be a synchronized signature scheme with lifetime $L$, let $\mathcal{A}$ be any algorithm. Consider the following experiment $\mathbf{MS\text{-}SY\text{-}UF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{A})$:

1. Generate keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par})$.
2. Run $\mathcal{A}$ on input $\mathsf{par}$ and $\mathsf{pk}$, and with (classical) access to the following oracle:
   - $\textsc{Sig}(\mathsf{ep}, \mathsf{m})$ for $\mathsf{ep} \in [L], \mathsf{m} \in \{0,1\}^{l_{msg}}$: If $\mathsf{Signed}[\mathsf{ep}] \neq \bot$, then return $\bot$. Otherwise, compute $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, \mathsf{ep}, \mathsf{m})$, set $\mathsf{Signed}[\mathsf{ep}] := \mathsf{m}$, and return $\sigma$.
3. Obtain from $\mathcal{A}$ a forgery $(k^*, (\mathsf{pk}_i^*)_{i=1}^{k^*}, \mathsf{ep}^*, \mathsf{m}^*, \bar{\sigma}^*)$ with $\mathsf{ep}^* \in [L]$ and $\mathsf{m}^* \in \{0,1\}^{l_{msg}}$. Output 1 if it holds that $\mathsf{Ver}((\mathsf{pk}_i^*)_{i=1}^{k^*}, \mathsf{ep}^*, \mathsf{m}^*, \bar{\sigma}^*) = 1$, $\mathsf{m}^* \neq \mathsf{Signed}[\mathsf{ep}^*]$, and there is an $i$ such that $\mathsf{pk}_i^* = \mathsf{pk}$. Otherwise, output 0.

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

$$\mathsf{Adv}_{\mathsf{MS}}^{\mathsf{MS\text{-}SY\text{-}UF\text{-}CMA}}(\mathcal{A}) := \Pr\left[\mathbf{MS\text{-}SY\text{-}UF\text{-}CMA}_{\mathsf{MS}}(\mathcal{A}) \Rightarrow 1\right].$$

## A.2 Non-Interactive Argument Systems

In the following, we define non-interactive argument systems. We will make black-box use of these systems for our multi-signature construction, as the focus of our work is to explore the security and efficiency of hash-based candidates for the underlying signature scheme. Nonetheless, finding a secure, efficient, and conceptually simple instantiation of such argument systems will be a necessary next step on the road to post-quantum proof-of-stake.

**Definition 14** (Non-Interactive Proof System)**.** Let $\Gamma \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation, where for a pair $(\mathsf{stmt}, \mathsf{witn}) \in \Gamma$, we refer to $\mathsf{stmt}$ as the statement, and $\mathsf{witn}$ as the witness. Let $\mathsf{H}$ be a random oracle. A non-interactive argument system for $\Gamma$ with respect to $\mathsf{H}$ is defined to be a pair of efficient algorithms $\mathsf{AS} = (\mathsf{ArgProve}, \mathsf{ArgVer})$ with (classical) oracle access to $\mathsf{H}$ and the following syntax:

- $\mathsf{ArgProve}^{\mathsf{H}}(\mathsf{stmt}, \mathsf{witn}) \to \pi$ is deterministic[19], takes as input a statement $\mathsf{stmt}$ and a witness $\mathsf{witn}$, and outputs an argument string $\pi$.
- $\mathsf{ArgVer}^{\mathsf{H}}(\mathsf{stmt}, \pi) \to b$ is deterministic, takes as input a statement $\mathsf{stmt}$ and an argument string $\pi$, and outputs a bit $b \in \{0,1\}$.

[19] As we do not require any zero-knowledge property, we can assume that proving is deterministic.

Further, we say that $\mathsf{AS}$ has correctness error at most $\delta$, if for all pairs $(\mathsf{stmt}, \mathsf{witn}) \in \Gamma$, we have

$$\Pr\left[\mathsf{ArgVer}^{\mathsf{H}}(\mathsf{stmt}, \pi) = 0 \mid \pi := \mathsf{ArgProve}^{\mathsf{H}}(\mathsf{stmt}, \mathsf{witn})\right] \leq \delta,$$

with probability taken over the randomness of $\mathsf{H}$.

*Remark* 9 (Succinctness). To obtain non-trivial aggregation of signatures, we need that the argument system is *succinct*, meaning that the size of $\pi$ is significantly smaller than the size of the witness.

*Remark* 10 (Random Oracles). We highlight again that the verifier of the signature scheme should not make random oracle calls. More precisely, what we need to avoid is that the relation $\Gamma$ that we prove is defined with respect to a random oracle. On the other hand, the succinct argument itself can use random oracles, and it has been shown that for succinctness, non-falsifiable assumptions are necessary [GW11].

The security property of interest is *knowledge soundness*, which intuitively guarantees that any efficient prover capable of producing a valid (i.e., verifying) argument string must also know a valid witness. This is typically formalized by requiring the existence of an efficient extractor that can derive the witness from the argument string. Knowledge soundness is particularly useful in our setting where we want to prove that aggregating signatures with a succinct argument yields a secure multi-signature. In the security proof, we first extract all individual signatures from the aggregate signature and then reduce to the security of the underlying signature scheme.

The formal definition of knowledge soundness involves significant subtleties, as extensively discussed by Unruh [Unr17]. These challenges become even more pronounced when considering quantum adversaries that can query the random oracle in superposition. This is relevant for analyzing argument systems in the quantum random oracle model (QROM) [BDF+11]. Two concrete examples highlight these subtleties: First, Chiesa et al. [CMS19] demonstrate that modern pqSNARKs based on hash-functions are knowledge-sound in the QROM. Unfortunately, their definition is *non-adaptive*, meaning that the statement cannot depend on the results of random oracle queries. In the context of aggregating signatures, the statement corresponds to the list of public keys and the message, which can indeed be chosen adversarially after querying the random oracle. Despite this limitation, the results of Chiesa et al. remain an important indication that pqSNARKs are a post-quantum secure method for aggregation. Second, Unruh's final definition [Unr17] allows the extractor arbitrary black-box access to the adversary, including the ability to rewind it[20]. However, in applications like signature aggregation, we need to argue that the extracted individual signature is fresh (i.e., not derived from the signing oracle). Running the adversary multiple times to extract signatures makes this argument unclear.

To address this, we use a definition of knowledge soundness that is both adaptive and straight-line: (1) the (quantum) extractor provides the random oracle to the adversary. (2) once the adversary terminates, the extractor must extract a valid witness. We conjecture that state-of-the-art pqSNARK constructions satisfy this adaptive straight-line definition[21]. Verifying this conjecture in the quantum setting is an important avenue for future work.

**Definition 15** (Knowledge Soundness)**.** Let $\Gamma \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation. Let $\mathsf{H}$ be random oracle. Let $\mathsf{AS} = (\mathsf{ArgProve}, \mathsf{ArgVer})$ be a non-interactive argument system for $\Gamma$ with respect to $\mathsf{H}$. Let $\mathcal{A}$ be an algorithm. Consider the following experiment, $\mathbf{KN\text{-}REAL}_{\mathsf{AS}}(\mathcal{A})$:

1. Run $\mathcal{A}$ with quantum access to $\mathsf{H}$ and obtain an output $(\mathsf{stmt}, \pi)$.

[20] Rewinding is particularly problematic in quantum settings, but readers unfamiliar with this issue may disregard it for now.

[21] In the classical random oracle model, hash-based pqSNARKs are already known to satisfy strong notions of adaptive straight-line extractability [CF24].

2. Output $\mathsf{ArgVer}^{\mathsf{H}}(\mathsf{stmt}, \pi)$.

Let $\mathsf{Ext}$ be another algorithm, and consider the experiment $\mathbf{KN\text{-}IDEAL}_{\mathsf{AS},\mathsf{Ext}}(\mathcal{A})$:

1. Run $\mathcal{A}$ with quantum access to an oracle $\mathsf{H}$ *provided by* $\mathsf{Ext}$ and obtain an output $(\mathsf{stmt}, \pi)$.

2. Run $\mathsf{Ext}$ on input $(\mathsf{stmt}, \pi)$ and obtain $\mathsf{witn}$ from $\mathsf{Ext}$.

3. Output $\mathsf{ArgVer}^{\mathsf{H}}(\mathsf{stmt}, \pi) \wedge (\mathsf{stmt}, \mathsf{witn}) \in \Gamma$.

Then, we say that $\mathsf{AS}$ is an argument of knowledge with extractor $\mathsf{Ext}$, loss $\mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}} \colon \mathbb{R} \to \mathbb{R}$, where $\mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}}$ is a non-decreasing function, and extraction time $\theta$, if for every quantum algorithm $\mathcal{A}$ that makes at most $t$ quantum queries to $\mathsf{H}$ in $\mathbf{KN\text{-}REAL}_{\mathsf{AS}}(\mathcal{A})$, we have that $\mathbf{KN\text{-}IDEAL}_{\mathsf{AS},\mathsf{Ext}}(\mathcal{A})$ runs in time $\theta(t)$ and

$$\Pr\left[\mathbf{KN\text{-}REAL}_{\mathsf{AS}}(\mathcal{A}) \Rightarrow 1\right] \leq \mathsf{Loss}_{\mathsf{AS},\mathsf{Ext}}\left(\Pr\left[\mathbf{KN\text{-}IDEAL}_{\mathsf{AS},\mathsf{Ext}}(\mathcal{A}) \Rightarrow 1\right]\right).$$

# B (Quantum) Random Oracle Tools

To get an impression for how to set parameters, we use heuristic bounds for the security notions we have defined for tweakable hash functions. To derive these bounds, we use the (classical and quantum) random oracle model, relying on several tools that we present in this section.

**Adaptive Reprogramming.** The first tool that we need is adaptive reprogramming, as introduced and analyzed in [GHHM21]. We first define the experiment, and the recall a bound in the quantum random oracle model. For convenience, we also state a simple bound in the classical random oracle.

**Definition 16** (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 $\mathsf{Repro}_b$:

- Sample a random oracle $\mathrm{O}_0 \xleftarrow{\$} Y^{X_1 \times X_2}$, i.e., $\mathrm{O}_0 \colon X_1 \times X_2 \to Y$.

- Define a copy of $\mathrm{O}_0$ as $\mathrm{O}_1 := \mathrm{O}_0$.

- Run $\mathcal{A}$ with (classical or quantum) access to $\mathrm{O}_b$ 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 $\mathrm{O}_b$ 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 $(x_1, y) \xleftarrow{\$} X_1 \times Y$.

2. $\mathrm{O}_1 := \mathrm{O}_1^{(x_1,x_2)\mapsto y}$, i.e., $\mathrm{O}_1$ is reprogrammed such that $\mathrm{O}_1(x_1, x_2) = y$.

3. Return $x_1$.

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

$$\mathsf{Adv}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) = \left|\Pr\left[\mathsf{Repro}_0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{Repro}_1(\mathcal{A}) \Rightarrow 1\right]\right|.$$

**Lemma 9** ([GHHM21])**.** *Let $X_1$, $X_2$ and $Y$ be finite sets, and let $\mathcal{A}$ be any algorithm in the game in definition 16. Assume that $\mathcal{A}$ issues $R$ many classical calls to* Reprogram *and $q$ many quantum queries to $O_b$. Then, we have*

$$\mathsf{Adv}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) \leq \frac{3R}{2} \cdot \sqrt{\frac{q}{|X_1|}}.$$

**Lemma 10.** *Let $X_1$, $X_2$ and $Y$ be finite sets, and let $\mathcal{A}$ be any classical algorithm in the game in definition 16. Assume that $\mathcal{A}$ issues $R$ many classical calls to* Reprogram *and $q$ many classical queries to $O_b$. Then, we have*

$$\mathsf{Adv}^{\mathsf{Repro}}_{R,q}(\mathcal{A}) \leq \frac{R \cdot q}{|X_1|}.$$

*Proof.* To detect the reprogramming the adversary must query the random oracle on at least one of the reprogrammed seeds $x_1$ *before* the reprogramming query. Since these are chosen uniformly at random the probability that a $x_1$ collides with one of the $q$ queries that were done before is $q/|X_1|$. With a union bound, we get the claimed bound. □

**HRS-Framework for Sets.** The second tool that we need is the HRS-framework from [HRS16]. The idea is that an adversary that is given oracle access to a boolean function should have a hard time to find an input which evaluates to 1, assuming the boolean function has only a few such inputs. Although the authors of [HRS16] used functions over a boolean input domain $\{0,1\}^c$, the results naturally generalize to functions that map an arbitrary set to $\{0,1\}$. Here, we present this adaptation of the HRS-Framework [HRS16] to arbitrary sets.

**Definition 17** (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 $D_\lambda$ on $\mathcal{F}$ such that a function $f \leftarrow D_\lambda$ drawn from $D_\lambda$ 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

$$\mathsf{Adv}_{\mathsf{Avg\text{-}Search}_\lambda}(\mathcal{A}) := \Pr\left[ f(x) = 1 \;\middle|\; f \leftarrow D_\lambda, \quad x \leftarrow \mathcal{A}^f(\cdot) \;\right].$$

**Lemma 11** ([HRS16], [BHRvV21])**.** *For any algorithm $\mathcal{A}$ that makes at most $q$ queries to $f$, it holds that*

$$\mathsf{Adv}_{\mathsf{Avg\text{-}Search}_\lambda}(\mathcal{A}) \leq \begin{cases} \lambda(q+1), & \textit{if } \mathcal{A} \textit{ is a classical algorithm with classical access to } f \\ 8\lambda(q+1)^2, & \textit{if } \mathcal{A} \textit{ is a quantum algorithm with quantum access to } f \end{cases}.$$

To give a proof for lemma 11 we follow the same steps as in [HRS16], but we do not make a restriction to bit strings. We present this proof here for completeness, relying on Theorem 7.2 from [Zha12], as recalled next.

**Theorem 3** ([Zha12])**.** *Fix an integer $q$, and let $D_\lambda$ be a family of distributions on $\{f : \mathcal{X} \to \mathcal{Y}\}$ indexed by $\lambda \in [0,1]$. Suppose there is an integer $d$ such that for every $2q$ pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, the function (in $\lambda$)*

$$p_\lambda := \Pr_{f \leftarrow D_\lambda} [\forall i \in \{1, \ldots, 2q\} \colon f(x_i) = y_i]$$

*is a polynomial of degree at most $d$ in $\lambda$. Then, any quantum algorithm $\mathcal{A}$ making $q$ queries can only distinguish $D_\lambda$ from $D_0$ with probability at most $2\lambda d^2$, i.e.,*

$$\left| \Pr_{f \xleftarrow{\$} D_0} \left[\mathcal{A}^f() = 1\right] - \Pr_{f \xleftarrow{\$} D_\lambda} \left[\mathcal{A}^f() = 1\right] \right| \leq 2\lambda d^2.$$

*Proof of lemma 11.* To translate the result from theorem 3 to our needs, we set $\mathcal{X} = \mathcal{S}$ and $\mathcal{Y} = \{0,1\}$. Let $k$ be the number of $y_i = 1$ in an arbitrary collection of $2q$ pairs $\{(x_i, y_i)\}_{i=1}^{2q}$. Then, by the definition of $p_\lambda$ we have

$$p_\lambda := \Pr_{f \leftarrow D_\lambda} [\forall i \in \{1, \ldots, 2q\} \colon f(x_i) = y_i] = \lambda^k (1-\lambda)^{2q-k}.$$

Hence, $p_\lambda$ is a polynomial in $\lambda$ with degree at most $2q$. Hence, the advantage in distinguishing $f$ from $D_\lambda$ and $f$ from $D_0$ is bounded by $8\lambda q^2$. Since the distribution $D_0$ always outputs the constant 0 function, obtaining a marked item (i.e., $x \in \mathcal{S}$ with $f(x) = 1$) for $f \leftarrow D_\lambda$ immediately distinguishes $D_\lambda$ from $D_0$. Given an algorithm $\mathcal{A}$ that queries $f$ and outputs $x$ after $q$ queries, it is sufficient to do one more query to check if $f(x) = 1$ and distinguish $D_\lambda$ from $D_0$. Thus, we obtain that

$$\mathsf{Adv}_{\mathsf{Avg\text{-}Search}_\lambda}(\mathcal{A}) \leq 8\lambda(q+1)^2.$$

The classical bound just follows from the fact that each query can be successful with probability $\lambda$ and if the adversary has not found a solution through the first $q$ queries it may output a random guess.

□

# C Multi-Target Undetectability

In this section, we revisit the analysis of undetectability. In [HK22], the analysis was given for a tweakable hash function of the form $\mathcal{P} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$. We show that the proof also works for the case of a tweakable hash function $\mathsf{Th} \colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$, i.e., with general input and output domains. We emphasize that this does not introduce a new proof, and we simply follow the proof from [HK22] while removing the unnecessary restriction on the function's input and output spaces.

**Definition 18** (Distinguishing Weights)**.** Let $\mathcal{F}$ be the set of all functions of the form $\mathcal{M} \to \{0,1\}$, and define the sets $S_i = \{f \in \mathcal{F} \mid wt(f) = i\}$ where $wt(f) = |\{x \mid f(x) = 1\}|$. Let $\mathcal{A}$ be a (stateful) algorithm. Consider the following experiment $\mathbf{Dist}^{S_i, S_j}(\mathcal{A})$:

1. Sample $b \xleftarrow{\$} \{0,1\}$.
2. Run $\mathcal{A}$ with (quantum) access to an oracle $f$:
   - If $b = 0$, set $f \xleftarrow{\$} S_i$.
   - If $b = 1$, set $f \xleftarrow{\$} S_j$.
3. After no more than $q$ queries to $f$ from $\mathcal{A}$ obtain a bit $b' \in \{0,1\}$ and output $b'$.

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

$$\mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_i,S_j)}(\mathcal{A}) = \left|\Pr\left[\mathbf{Dist}^{S_i,S_j}(\mathcal{A}) \Rightarrow 1 \mid b=0\right] - \Pr\left[\mathbf{Dist}^{S_i,S_j}(\mathcal{A}) \Rightarrow 1 \mid b=1\right]\right|.$$

One can derive the following lemma from Theorem 9.3.2 and Lemma 9.3.6 in [KLM06].

**Lemma 12** ([KLM06])**.** *Let $S_i$ be as defined above. The advantage of any $q$ query quantum algorithm in distinguishing $S_0$ from $S_1$ is $\mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_0,S_1)}(\mathcal{A}) \leq 6q/\sqrt{|\mathcal{M}|}$.*

In our reduction we need sets $S_0^l$ and $S_1^l$. We say $f\colon [l] \times \mathcal{M} \to \{0,1\}^n$ is in $S_i^l$, if $f(j,\cdot) \in S_i$ for every $j \in [l]$. We now show that distinguishing $f \xleftarrow{\$} S_1^l$ from $f \xleftarrow{\$} S_0^l$ is as hard as distinguishing $f \xleftarrow{\$} S_1$ from $f \xleftarrow{\$} S_0$.

**Lemma 13.** *Consider sets $S_0$, $S_1$, $S_0^l, S_1^l$ as defined above. Then, $\mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_0,S_1)}(\mathcal{A}) = \mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_0^l,S_1^l)}(\mathcal{A})$.*

*Proof.* Assume an algorithm $\mathcal{A}$ can distinguish $f \xleftarrow{\$} S_1$ from $f \xleftarrow{\$} S_0$. Then, to distinguish $f \xleftarrow{\$} S_1^l$ from $f \xleftarrow{\$} S_0^l$, we run $\mathcal{A}$ on $f(1,\cdot)$. Hence, $\mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_0,S_1)}(\mathcal{A}) \leq \mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_0^l,S_1^l)}(\mathcal{A})$.

To show equality we now give the reduction in the opposite direction. Without loss of generality we view the elements of $\mathcal{M}$ as integers $\{0,\ldots,|\mathcal{M}|-1\}$ or as values in $\mathbb{Z}_{|\mathcal{M}|}$. Assume we have an algorithm that distinguishes $f \xleftarrow{\$} S_1^l$ from $f \xleftarrow{\$} S_0^l$. Our task is to distinguish $f' \xleftarrow{\$} S_1$ from $f' \xleftarrow{\$} S_0$. To build $f$ from $f'$ we sample a random value from $\mathcal{M}$ using a random function $e\colon [l] \to \mathcal{M}$, and set $f(i,x) := f'(x + e(i) \bmod |\mathcal{M}|)$. One can see that if $f'$ was a constant zero function then $f$ is a collection of constant zero functions, so $f \in S_0^l$. On the other hand, if $f' \in S_1$ then for each $i$, the function $f(i,\cdot)$ outputs 1 for exactly one random value, since $e(i)$ were chosen uniformly at random, so $f \in S_1^l$. Also, as all the $e(i)$ are uniform and independent, $f$ is distributed uniformly in $S_1^l$. Hence, $\mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_0,S_1)}(\mathcal{A}) \geq \mathsf{Adv}_{\mathcal{F},q}^{\mathsf{Dist}(S_0^l,S_1^l)}(\mathcal{A})$. □

**Theorem 4.** *Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ a tweakable hash function modeled as quantum random oracle. Consider any quantum adversary $\mathcal{A}$ against undetectability for a given $\mathcal{M}' \subseteq \mathcal{M}$, for $p$ targets making $q$ queries to $\mathsf{Th}$. Then, there is a quantum adversary $\mathcal{B}$ that makes $2q$ queries to its oracle and distinguishes $S_0^p$ from $S_1^p$ with*

$$\mathsf{Adv}_{\mathsf{Th},\mathcal{M}',p}^{\mathsf{SM\text{-}UD}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathcal{F},2q}^{\mathsf{Dist}(S_0^p,S_1^p)}(\mathcal{B}) \leq \frac{12q}{\sqrt{|\mathcal{M}'|}}.$$

*Proof.* The first inequality is show exactly as in [HK22]. Using lemmas 12 and 13, we get the second inequality and complete the proof. □

# D Multi-Target Collision Resistance with Random Sampling

In definition 6, we have introduced the notion of multi-target collision resistance with random sampling. We will now show that this notion is indeed plausible by giving an analysis in the (quantum) random oracle model. As a result, we obtain an upper bound on the success probability of any adversary in breaking the notion. Naturally, this bound depends on the number of random oracle queries. We prove the following theorem.

**Theorem 5.** *Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times (\mathcal{M} \times \mathcal{R}) \to \mathcal{H}$ be a tweakable hash function modeled as a (classical or quantum) random oracle, that takes a public parameter $P \in \mathcal{P}$, a tweak $T \in \mathcal{T}$ and an input that consists of two parts: a message $M \in \mathcal{M}$ and a seed $\rho \in \mathcal{R}$. Let $\mathsf{Prop}\colon \mathcal{H} \to \{0,1\}$ be any property. Let $\mathcal{A}$ be any (classical or quantum) adversary* *against multi-target collision resistance with random sampling (definition 6) that makes at most $q$ (classical or quantum) queries to the random oracle* $\mathsf{Th}$ *and $p$ classical queries to its challenge oracle. Then, there exists a (classical or quantum) adversary $\mathcal{B}$ against* $\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}$ *that makes no more than $q' = q + pK$ queries to its oracle and a (classical or quantum) adversary $\mathcal{C}$ in the game* $\mathsf{Repro}$ *as in definition 16 that makes no more than $q' = q+pK$ queries to its random oracle and no more than $pK$ queries to its reprogramming oracle such that:*

$$\mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{Repro}}_{pK,q+pK}(\mathcal{C}).$$

*Consequently, from lemmas 9 to 11, we obtain the following bounds:*

$$\mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}) \leq \frac{(q'+1)}{|\mathcal{H}|} + \frac{q'pK}{|\mathcal{R}|} \qquad \textit{for a classical adversary.}$$

$$\mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}) \leq \frac{8(q'+1)^2}{|\mathcal{H}|} + \frac{3 \cdot pK}{2} \cdot \sqrt{\frac{q'}{|\mathcal{R}|}} \qquad \textit{for a quantum adversary.}$$

*Proof.* We give a sequence of games **Game.**$i$ to prove the claim, and denote the probability that the $i$th game outputs 1 by $\mathsf{Adv}^{\mathbf{Game}.i}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A})$.

**Game.0:** Our initial game is the original game for multi-target collision resistance with random sampling, see definition 6. To recall, the adversary gets as input a parameter $P$ and it gets classical access to a challenge oracle that takes as input some message $M \in \mathcal{M}$ and a tweak $T \in \mathcal{T}$. The tweak must be fresh (not used in the previous queries). The oracle then randomly samples a seed $\rho \xleftarrow{\$} \mathcal{R}$ until the digest $x := \mathsf{Th}(P, T, M, \rho)$ satisfies property $\mathsf{Prop}$. If the oracle finds such a seed then it returns $(x, \rho)$. If the oracle does not manage to find a seed after $K$ tries, it returns $\perp$. The task of the adversary is to find a message and a seed that collides with one of the returned digests under the same tweak. By definition, we have

$$\mathsf{Adv}^{\mathsf{SM\text{-}rTCR},K}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}) = \mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}).$$

**Game.1:** Now consider **Game.1** in which we sample the seed and the output uniformly at random to answer the challenge queries. To align our responses with the $\mathsf{Th}$ we reprogram the hash function. Intuitively, since $\mathsf{Th}$ is modeled as a random oracle, we can bound the probability that the adversary notices this reprogramming using the **Repro** game. The formal representation of **Game.1** is the following, where $\mathcal{A}$ gets (classical or quantum) access to $\mathsf{Th}$ throughout the game:

1. Generate a random public parameter $P \xleftarrow{\$} \mathcal{P}$.
2. Run $\mathcal{A}$ on input $P$ with classical access to an oracle that takes $T \in \mathcal{T}$ and $M \in \mathcal{M}$ and works as follows:
   - If $|Q| \geq p$ or there is a tuple $(T, M', \rho') \in Q$, for some $M', \rho'$ return $\perp$.
   - Otherwise Set $\mathsf{ctr} = 0$ and $x = \perp$. While $\mathsf{ctr} < K$ and $x = \perp$:
     (a) Sample $\rho \xleftarrow{\$} \mathcal{R}$.
     (b) Sample $y \xleftarrow{\$} \mathcal{H}$.
     (c) Program $\mathsf{Th}(P, T, M, \rho) := y$.
     (d) If $\mathsf{Prop}(y) = 1$: Insert $(T, M, \rho)$ into $Q$ and set $x := y$.
     (e) Else: Set $x := \perp$, $\rho := \perp$.
     (f) Set $\mathsf{ctr} := \mathsf{ctr} + 1$.
   - If $x = \perp$: Insert $(T, M, \perp)$ into $Q$.

- Output $(x, \rho)$.

3. Obtain from $\mathcal{A}$ an output $(j, M^*, \rho^*)$ with $M \in \mathcal{M}$, $j \in [|Q|]$. Denote the $j$th entry in $Q$ by $(M_j, T_j, \rho_j)$.

4. Output 1 if $\mathsf{Th}(P, T_j, M_j, \rho_j) = \mathsf{Th}(P, T_j, M^*, \rho^*)$ and $(M^*, \rho^*) \neq (M_j, \rho_j)$. Otherwise, output 0.

Here the difference from **Game.0** is in Lines (b) and (c) of the challenge oracle. Instead of querying $\mathsf{Th}$ we generate the output uniformly at random and reprogram $\mathsf{Th}$ to match the generated value. Note, that the call to the challenge oracle in **Game.1** can be represented as two calls in the **Repro** game: a call to $\mathsf{Reprogram}$ and an $\mathrm{O}_1$ call afterwards. Here, we consider $(P, T, M) \in X_2$ and $\rho \in X_1 = \mathcal{R}$. In this reduction, we make at most $p \cdot K$ reprogramming calls and $p \cdot K + q$ calls to $\mathsf{Th}$. As a result we obtain

$$|\mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}) - \mathsf{Adv}^{\mathbf{Game.1}}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A})| \leq \mathsf{Adv}^{\mathsf{Repro}}_{pK,q+pK}(\mathcal{C}).$$

**Game.2:** We now change how the tweakable hash function (currently modeled as a random oracle) is defined. Concretely, we define it based on a boolean function $f\colon \mathcal{T} \times \mathcal{M} \times \mathcal{R} \to \{0,1\}$ with $f \leftarrow D_\lambda$, for $\lambda = 1/|\mathcal{H}|$. Looking ahead, in this way we will later be able to use the HRS framework (see definition 17). In addition, we change the reprogramming routine. We will see that these changes are purely conceptual and do not change the view of the adversary. So we claim that the success probability in **Game.2** will stay the same as in **Game.1**. First we show how to construct a tweakable hash function from the boolean function $f$.

1. Generate a random tweakable hash function $\mathsf{Th}'$ and public parameter $P$.

2. For each $t \in \mathcal{T}$, sample an ordered set $S_t$ and a pair $(\rho_t^*, x_t)$ as follows: Set $\mathsf{ctr} = 0$ and $x = \perp$. While $\mathsf{ctr} < K$ and $x = \perp$:
   (a) Sample $\rho \xleftarrow{\$} \mathcal{R}$.
   (b) Sample $y \xleftarrow{\$} \mathcal{H}$.
   (c) If $\mathsf{Prop}(y) = 1$: Append $(\rho, y)$ to $S_t$, set $x := y$, $\rho^* := \rho$, and define $(\rho_t^*, x_t) := (\rho^*, x)$.
   (d) Else: Append $(\rho, y)$ to $S_t$ and set $x = \perp$.

3. Sample random functions $g_t\colon \mathcal{M} \times \mathcal{R} \to \mathcal{H} \setminus \{x_t\}$ for each $t \in \mathcal{T}$.

4. Construct a function $g\colon \mathcal{T} \times \mathcal{M} \times \mathcal{R} \to \mathcal{H}$ the following way: On input $(t, m, \rho)$ check
   - If $f(T_i, m, \rho) = 1$ : Return $x_t$.
   - If $f(T_i, m, \rho) \neq 1$ : Return $g_t(m, \rho)$

5. Define $\mathsf{Th}$ as $\mathsf{Th}(p, t, (m, \rho)) := \begin{cases} g(t, m, \rho), & \text{if } p = P. \\ \mathsf{Th}'(p, t, m, \rho), & \text{otherwise.} \end{cases}$

Note that the constructed $\mathsf{Th}$ is still a uniformly random function. Next we update our reprogramming techniques by using the values from our construction of $\mathsf{Th}$. This is a purely conceptual change. The new game is as follows (with the winning condition as before):

1. Use the parameter $P$ generated in the construction of $\mathsf{Th}$.

2. Run $\mathcal{A}$ with an input $P$ and with (classical) access to an oracle that takes $T \in \mathcal{T}$ and $M \in \mathcal{M}$ and works as follows:

- If $|Q| \geq p$ or there is a tuple $(T, M', \rho') \in Q$, for some $M', \rho'$ return $\perp$.
- For each input $(\rho_j, y_j) \in S_T$:
  - Program $\mathsf{Th}(P, T, M, \rho_j) := y_j$.
- Insert $(T, M, \rho_T^*)$ into $Q$, where $(\rho_T^*, x_T) \in D_T$.
- Output $(x_T, \rho_T^*)$.

Through these two changes we managed to incorporate the boolean function into the game while keeping all the distributions the same. As we have argued, we get

$$\mathsf{Adv}^{\mathbf{Game.}1}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}) = \mathsf{Adv}^{\mathbf{Game.}2}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}).$$

**Final reduction:** In the previous game we managed to incorporate the boolean function into the construction of $\mathsf{Th}$ and updated the reprogramming routine. As a result, a successful forgery should satisfy $\mathsf{Th}(P, T_j, M_j, \rho_j) = \mathsf{Th}(P, T_j, M^*, \rho^*) = x_{T_j}$ and $(M^*, \rho^*) \neq (M_j, \rho_j)$. This means that $(T_j, M^*, \rho^*)$ must satisfy the boolean function $f$ by construction. Hence, we can use the forgery in a reduction to break the $\mathsf{Adv}_{\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}}$ property. We get

$$\mathsf{Adv}^{\mathbf{Game.}2}_{\mathsf{Th},p,\mathsf{Prop}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}}(\mathcal{B}).$$

This concludes the proof. □

# E Multi-Target Collision Resistance

In this section, we give a bound on success probability against multi-target collision resistance (see definition 3) in the random oracle model, assuming a classical adversary. In [HK22], an analysis was given in the quantum random oracle model. We reuse their proof ideas to derive a bound in the classical setting. In addition, we give an updated bound for quantum adversary against tweakable hash function, where $|\mathcal{P}| \neq 2^k$ for some $k$.

**Definition 19** (Distinguishing from Constant Zero)**.** Let $f_P\colon \mathcal{P} \to \{0,1\}$ be the boolean function, with $f_P(pp) = 1$ if and only if $pp = P$. Let $f_0\colon \mathcal{P} \to \{0,1\}$ be the boolean function for which $f_0(x) = 0$ for all $pp \in \mathcal{P}$. Let $\mathcal{A}$ be a (stateful) algorithm, and consider the following experiment $\mathbf{ZeroDist}(\mathcal{A})$:

1. Generate a random public parameter $P \xleftarrow{\$} \mathcal{P}$.
2. Flip a random coin $b \xleftarrow{\$} \{0,1\}$.
3. If $b = 1$ give $\mathcal{A}$ oracle access to $f_P$. If $b = 0$ give $\mathcal{A}$ oracle access to $f_0$.
4. When $\mathcal{A}$ signals to continue, then remove the access to the given oracle and continue running $\mathcal{A}$ with input $P$.
5. Obtain from $\mathcal{A}$ a bit $b' \in \{0,1\}$ and output $b'$.

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

$$\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{A}) = \left|\Pr\left[\mathbf{ZeroDist}(\mathcal{A}) \Rightarrow 1 \mid b = 0\right] - \Pr\left[\mathbf{ZeroDist}(\mathcal{A}) \Rightarrow 1 \mid b = 1\right]\right|.$$

**Lemma 14.** *Let $\mathcal{A}$ be a classical algorithm that makes no more then $q$ classical queries to its oracle. Then, we have*

$$\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{A}) \leq \frac{q}{|\mathcal{P}|}.$$

*Proof.* It is straightforward to see that if the oracle has not been queried during the first stage on the selected public parameter $P$ then indistinguishability holds. The probability that the public parameter will match one of the $q$ queries is bounded by $q/|\mathcal{P}|$. □

To give a more general bound (i.e., for $|\mathcal{P}| \neq 2^k$) for target collision resistance in the quantum random oracle model, first recall that in [HK22], the authors rely on the (**) bound from [HRS16], which states that $\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{A}) \leq 4q^2/2^k$, if $\mathcal{P} = \{0,1\}^k$. If $2^k < |\mathcal{P}| < 2^{k+1}$, without loss of generality we can view $\mathcal{P}$ as being represented by $(k+1)$-bit integers in $0, \ldots, |\mathcal{P}|-1$. In the following lemma, we consider this setting, and the experiment **ZeroDist**$(\mathcal{A})$ where the adversary $\mathcal{A}$ is quantum and it has quantum access to the boolean function.

**Lemma 15.** *Let $\mathcal{A}$ be a quantum algorithm that makes no more then $q$ quantum queries to its oracle. Then, we have*

$$\mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{A}) \leq \frac{8q^2}{|\mathcal{P}|}.$$

*Proof.* Assume there is a quantum adversary $\mathcal{A}$ that breaks ZeroDist of a boolean function that operates on $\mathcal{P}$ with probability at least $\epsilon$. Then, it is straightforward to show that it can be utilized to break ZeroDist for $\hat{K} \xleftarrow{\$} \{0,1\}^{k+1}$ with probability at least $1/2\epsilon$, since the probability that $P \in \mathcal{P}$ for a random $P \xleftarrow{\$} \{0,1\}^{k+1}$ is at least $1/2$. Hence, we can conclude that $\epsilon \leq 8q^2/2^{k+1} \leq 8q^2/|\mathcal{P}|$. □

**Lemma 16** ([HK22])**.** *Let* Th *be a tweakable hash function modeled as a random oracle. For any quantum algorithm $\mathcal{A}$ against multi-target collision resistance (see definition 3) that makes at most $q$ quantum queries to its random oracle* Th*, there are quantum adversaries $\mathcal{B}$ (making $2q$ queries) and $\mathcal{C}$ (making $2q$ queries) such that*

$$\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},p}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{ZeroDist}}(\mathcal{C}).$$

We can reuse this result from [HK22] since the reductions work regardless whether the adversary is quantum or classical, and if the adversary is classical, then so are the reductions. By using the classical bounds from lemmas 11 and 14, we get the classical counterpart. We also state the updated bound in the quantum random oracle model.

**Lemma 17.** *Let* Th *be a tweakable hash function modeled as a random oracle. For any classical algorithm $\mathcal{A}$ against multi-target collision resistance (see definition 3) that makes at most $q$ classical queries to its random oracle* Th*, we have*

$$\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},p}(\mathcal{A}) \leq \frac{2q+1}{|\mathcal{H}|} + \frac{2q}{|\mathcal{P}|}.$$

**Lemma 18.** *Let* Th *be a tweakable hash function modeled as a random oracle. For any quantum algorithm $\mathcal{A}$ against multi-target collision resistance (see definition 3) that makes at most $q$ quantum queries to its random oracle* Th*, we have*

$$\mathsf{Adv}^{\mathsf{SM\text{-}TCR}}_{\mathsf{Th},p}(\mathcal{A}) \leq \frac{32(q+1)^2}{|\mathcal{H}|} + \frac{32q^2}{|\mathcal{P}|}.$$

# F Multi-Target Preimage Resistance

In this section we derive a new bound for multi-target preimage resistance (see definition 4) in the quantum random oracle model. Although there was a security analysis of multi-target preimage resistance based on some conjecture (see [BH19, BHK+19, HK22]), we want to give a bound that is not relying on any conjecture. To this end, we first introduce the related notion of *single-function, multi-target one-wayness for distinct tweaks* and analyze its security, and then reduce multi-target preimage resistance to it.

## F.1 Multi-Target One-Wayness

We start with a definition of single-function, multi-target one-wayness for distinct tweaks. This notion is different from multi-target preimage resistance in that the challenges are generated uniformly at random from the output space, rather then by computing a hash of a random input.

**Definition 20** (Multi-Target One-Wayness)**.** Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ be a tweakable hash function as defined in definition 1. Let $\mathcal{A}$ be a (stateful) algorithm, $\mathcal{M}' \subseteq \mathcal{M}$, and $p \in [|\mathcal{T}|]$. Consider the following experiment $\mathbf{SM\text{-}OW}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A})$:

1. Generate a random public parameter $P \xleftarrow{\$} \mathcal{P}$.
2. Run $\mathcal{A}$ with (classical) access to an oracle that takes $T \in \mathcal{T}$ and works as follows:
   - If $|Q| \geq p$ or there is an $y' \in \mathcal{H}$ with $(T, y') \in Q$, return $\perp$.
   - Otherwise, sample $y \xleftarrow{\$} \mathcal{H}$, insert $(T, y)$ into the list $Q$ and output $y$.
3. When $\mathcal{A}$ signals to continue, then continue running $\mathcal{A}$ with input $P$, but without the oracle access.
4. Obtain from $\mathcal{A}$ an output $(j, M)$ with $M \in \mathcal{M}'$, $j \in [|Q|]$. Denote the $j$th entry in $Q$ by $(T_j, y_j)$.
5. Output 1 if $\mathsf{Th}(P, T_j, M) = y_j$. Otherwise, output 0.

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

$$\mathsf{Adv}^{\mathsf{SM\text{-}OW}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) := \Pr[\mathbf{SM\text{-}OW}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) \Rightarrow 1].$$

**Theorem 6.** *Let $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ be a tweakable hash function modeled as a random oracle. Let $\mathcal{A}$ be any quantum adversary against multi-target one-wayness (definition 20) on subspace $\mathcal{M}' \subseteq \mathcal{M}$, that makes at most $q$ quantum queries to $\mathsf{Th}$ and $p$ classical query to its challenge oracle. Then, there is a quantum adversary $\mathcal{B}$ against $\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}$ that makes $q$ queries to its oracle, such that*

$$\mathsf{Adv}^{\mathsf{SM\text{-}OW}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}}(\mathcal{B}) \leq \frac{8(q+1)^2}{|\mathcal{H}|}.$$

*Proof.* We prove the statement via a sequence of games, where the probability that the $i$th game outputs 1 is denoted by $\mathsf{Adv}^{\mathbf{Game.}i}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A})$. Without loss of generality we view the set $\mathcal{H}$ as $n$ bit representation of integers $\{0, \ldots, |\mathcal{H}| - 1\}$.

**Game.0:** Our initial game is the original game for multi-target one-wayness, see definition 20. By definition, we have

$$\mathsf{Adv}^{\mathsf{SM\text{-}OW}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) = \mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}).$$

**Game.1:** This game is different from **Game.0** in the way we construct the hash function $\mathsf{Th}$. For the construction we will need several random functions:

- Function $g\colon \mathcal{T} \to \mathcal{H}$;
- Function $\mathsf{Th}'\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$;
- Function $h'\colon \mathcal{T} \times \mathcal{M}' \to \mathcal{H} \setminus \{0\}^n$;
- Boolean function $f\colon \mathcal{T} \times \mathcal{M}' \to \{0, 1\}$ sampled from the distribution $D_{1/|\mathcal{H}|}$, see definition 17.

Using $h'$, we construct a random function $h\colon \mathcal{T} \times \mathcal{M}' \to \mathcal{H}$, but with the constraint that $h(t,x)$ never evaluates to $g(t)$. The construction of $h$ is as follows:

1. On input $t, x$ compute $h'(t,x) = y' \in \mathcal{H} \setminus \{0\}^n$;
2. If $y' \leq g(t)$: Return $y' - 1$;
3. If $y' > g(t)$: Return $y'$.

With $g$, $h$, $\mathsf{Th}'$, and $f$, we now explain how $\mathsf{Th}$ is implemented in this game. First a random $P \xleftarrow{\$} \mathcal{P}$ is sampled. Then, $\mathsf{Th}$ works as follows on input $pp \in \mathcal{P}, t \in \mathcal{T}, x \in \mathcal{M}$:

- If $pp = P \ \wedge x \in \mathcal{M}' \ \wedge f(t,x) = 1$: Return $g(t)$.
- If $pp = P \ \wedge x \in \mathcal{M}' \ \wedge f(t,x) \neq 1$: Return $h(t,x)$.
- If $pp \neq P \vee \ x \notin \mathcal{M}'$: Return $\mathsf{Th}'(pp,t,x)$.

One can see that the distribution of $\mathsf{Th}$ is still uniform. We will use this very $P$ instead of sampling a new one and use function $g$ to respond to the challenge queries. Concretely, **Game.1** is as follows, where $\mathcal{A}$ obtains quantum random oracle access to $\mathsf{Th}$ throughout the game:

1. Sample $P$ and use it for $\mathsf{Th}$ as explained above.
2. Run $\mathcal{A}$ with (classical) access to an oracle that takes $T \in \mathcal{T}$ and works as follows:
   - If $|Q| \geq p$ or there is an $y' \in \mathcal{H}$ with $(T, y') \in Q$, return $\perp$.
   - Otherwise, compute $y = g(T)$, insert $(T, y)$ into the list $Q$ and output $y$.
3. When $\mathcal{A}$ signals to continue, then continue running $\mathcal{A}$ with input $P$, but without the oracle access.
4. Obtain from $\mathcal{A}$ an output $(j, M)$ with $M \in \mathcal{M}', j \in [|Q|]$. Denote the $j$th entry in $Q$ by $(T_j, y_j)$.
5. Output 1 if $\mathsf{Th}(P, T_j, M) = y_j$. Otherwise, output 0.

Since all the distributions remained the same the success probability of $\mathcal{A}$ also remains the same.

$$\mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) = \mathsf{Adv}^{\mathbf{Game.1}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}).$$

**Final reduction:** The last step is to bound the success probability of the adversary in **Game.1**. One can see that any solution for **Game.1** corresponds to the solution for the $\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}$ problem. Namely, the adversary must output a solution with the selected public parameter $P$ and in the subspace $\mathcal{M}'$. For such a solution we have two options: either $f(T_j, M) = 1$, or $f(T_j, M) \neq 1$. If $f(T_j, M) = 1$ then $(T_j, M)$ constitutes a solution for $\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}$. If $f(T_j, M) \neq 1$, then $\mathsf{Th}(P, T_j, M)$ would evaluate to $h(T_j, M)$, which was constructed to be never equal to $g(T_j) = y_j$ and hence this can not be a solution. Note that during our reduction we do not have to query $f$ during the challenge queries and only need $f$ for queries to $\mathsf{Th}$. We denote the number of queries to $\mathsf{Th}$ as $q$, so our reduction does no more then $q$ queries to $f$. Hence,

$$\mathsf{Adv}^{\mathbf{Game.1}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{Avg\text{-}Search}_{1/|\mathcal{H}|}}(\mathcal{B}).$$

This concludes the proof. $\square$

## F.2 Multi-Target Preimage Resistance

Now that we have a bound on multi-target one-wayness, we can prove a bound on multi-target preimage resistance.

**Theorem 7.** *Let* $\mathsf{Th}\colon \mathcal{P} \times \mathcal{T} \times \mathcal{M} \to \mathcal{H}$ *be a tweakable hash function modeled as a quantum random oracle. Let* $\mathcal{A}$ *be any quantum adversary against multi-target preimage resistance (definition 4) on subspace* $\mathcal{M}' \subseteq \mathcal{M}$*, that makes at most* $q$ *quantum queries to* $\mathsf{Th}$ *and* $p$ *classical query to its challenge oracle. Then, there are quantum algorithms* $\mathcal{B}$ *making at most* $q$ *quantum queries to* $\mathsf{Th}$ *and* $\mathcal{C}$ *making at most* $q+1$ *quantum queries such that*

$$\mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SM\text{-}OW}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{C}).$$

*Consequently, we have*

$$\mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) \leq \frac{8(q+1)^2}{|\mathcal{H}|} + \frac{12(q+1)}{\sqrt{|\mathcal{M}'|}}.$$

*using the bounds we already know.*

*Proof.* We prove the statement via a sequence of games, where the probability that the $i$th game outputs 1 is denoted by $\mathsf{Adv}^{\mathbf{Game}.i}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A})$.

**Game.0:** Our initial game is the original game for multi-target preimage resistance, see definition 4. By definition, we have

$$\mathsf{Adv}^{\mathsf{SM\text{-}PRE}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) = \mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}).$$

**Game.1:** This game is different from **Game.0** in the way we respond to the challenges. Instead of sampling a random input and returning a hash of the inputs we return a random value from $\mathcal{H}$. Note that the difference in these two games matches exactly the two cases in the undetectability game (see definition 5). A precise representation of **Game.1** is the following, where $\mathcal{A}$ gets quantum random oracle access to $\mathsf{Th}$ throughout the game:

1. Generate a random public parameter $P \xleftarrow{\$} \mathcal{P}$.
2. Run $\mathcal{A}$ with (classical) access to an oracle that takes $T \in \mathcal{T}$ and works as follows:
   - If $|Q| \geq p$ or there is an $y' \in \mathcal{H}$ with $(T, y') \in Q$, return $\perp$.
   - Otherwise, sample $y \xleftarrow{\$} \mathcal{H}$, insert $(T, y)$ into the list $Q$ and output $y$.
3. When $\mathcal{A}$ signals to continue, then continue running $\mathcal{A}$ with input $P$, but without the oracle access.
4. Obtain from $\mathcal{A}$ an output $(j, M)$ with $M \in \mathcal{M}'$, $j \in [|Q|]$. Denote the $j$th entry in $Q$ by $(T_j, y_j)$.
5. Output 1 if $\mathsf{Th}(P, T_j, M) = y_j$. Otherwise, output 0.

It is clear that there is a trivial reduction $\mathcal{C}$ such that

$$|\mathsf{Adv}^{\mathbf{Game.0}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) - \mathsf{Adv}^{\mathbf{Game.1}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A})| \leq \mathsf{Adv}^{\mathsf{SM\text{-}UD}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{C}).$$

**Final reduction:** The final step is to bound the success probability of the adversary in **Game.1**. One can see that the description of **Game.1** exactly matches the description of the multi-target one-wayness experiment for subspace $\mathcal{M}'$, see definition 20. Hence,

$$\mathsf{Adv}^{\mathbf{Game.1}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SM\text{-}OW}}_{\mathsf{Th},\mathcal{M}',p}(\mathcal{B}).$$

This concludes the proof. $\square$