Mitigating Multi-Target Attacks in Hash-based Signatures

Preliminary Version

Andreas Hülsing[1], Joost Rijneveld[2], and Fang Song[3]

[1] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
andreas@huelsing.net

[2] Radboud University, Digital Security Group,
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
joost@joostrijneveld.nl

[3] Department of Combinatorics & Optimization,
and Institute for Quantum Computing
University of Waterloo
fang.song@uwaterloo.ca

Abstract. This work introduces XMSS-T, a new hash-based signature scheme with tight security. Previous hash-based signature schemes are facing a loss of security, linear in performance parameters like the total tree height. Our new scheme can use hash functions with a smaller output length at the same security level, immediately leading to a smaller signature size. XMSS-T is stateful, however, the same techniques also apply directly to the recent stateless hash-based signature scheme SPHINCS (Eurocrypt 2015), and the signature size is improved as a result.

Being a little more specific and technical, the tight security stems from new multi-target notions of hash-function properties which we define and analyze. We give precise complexity for breaking these security properties under both classical and quantum generic attacks, thus establishing a reliable estimate for the quantum security of XMSS-T. Especially, we prove quantum upper and lower bounds for the query complexity tailored for cryptographic applications, whereas standard techniques in quantum query complexity have limitations such as they usually only consider worst-case complexity. Our proof techniques may be useful elsewhere.

We also implement XMSS-T and compare its performance to that of the most recent stateful hash-based signature scheme XMSS (PQCrypto 2011).

Keywords: post-quantum cryptography, hash-based signatures, hash function security, multi-target attacks, quantum algorithms

This work was supported by European Commission through the ICT program under contract ICT-645622 (PQCRYPTO). Part of this work was done while the first author was visiting IQC. F.S. acknowledges support by Cryptoworks21, Canada's NSERC and ORF.

1 Introduction

Hash-based signatures are considered the most promising post-quantum alternative to signature schemes like RSA and ECDSA, today. This is especially due to the good understanding of hash function security and exact reductionist proofs. The latter show the exact relation between the hardness of breaking the schemes and the hardness of breaking certain properties of the used hash function. This allows to give good estimates on the security of specific parameter sets.

Traditionally, the security of hash-based signature schemes was related to collision-resistance of the used hash function. In recent years several works focused on basing security on milder assumptions [16,12,13,22,24,6], such as second-preimage resistance and one-wayness. There are two fundamental reasons driving this trend. On the one hand, the attacks against the collision-resistance of SHA1 and MD5 motivated researchers to develop collision-resilient signature schemes [20,28]. On the other hand, collision resistance is subject to birthday attacks while (second-)preimage resistance is not. Hence, to reach a security level of λ bits, a hash function with $n = 2\lambda$ bit digests is needed if collision resistance is required whereas for (second-)preimage resistance only $n = \lambda$ bit digests are needed. Halving the output size of the used hash function immediately halves the signature and key sizes of hash-based signatures.

Multi-target attacks. The above statement is only half the truth because it bears on the implicit assumption that a hash function is used only once. Clearly, for many cryptographic constructions this is not the case. Consider for example preimage resistance (aka. one-wayness). For many cryptographic constructions, an adversary will be able to learn a magnitude of function values and security breach may occur once he finds a preimage for just one of them. More specifically, suppose that a hash function with n bit outputs is used d times in a cryptographic construction. If it suffices to invert the hash function on any one out of the d outputs to break the security of the scheme, then the attack complexity is downgraded to $\mathcal{O}(2^n/d)$ instead of $\mathcal{O}(2^n)$. Intuitively this is because every input value that an adversary tries has probability $d/2^n$ of being a solution instead of $1/2^n$, if we treat the hash function as a random function. For theoretical (asymptotic) security this worries nobody as d is normally at most polynomial in n. However, when choosing parameters in practice this can easily cause serious consequences.

This issue is indeed very pertinent to hash-based signatures. Consider for example the hash-based signature scheme XMSS [13] and its multi-tree version XMSS^{MT} [24] (see Section 4) with parameters that allow to use a keypair for a virtually unlimited amount of signatures (e.g. a total tree height of $h = 60$). In this case, an attacker can learn about 2^{66} images under the same hash function and will succeed in forging a signature if he finds a single preimage for any one of the 2^{66} values. Consequently, to achieve for example security of 256 bits one cannot use a 256 bit hash function but has to use one with output length 322. This does not only imply the use of a hash function with a bigger output length (and hence a slowdown), it also increases the signature size by roughly 25%.

This work. In this work we introduce a new hash-based signature scheme XMSS-T that is not vulnerable to multi-target attacks. Towards this end, we propose two new *multi-target* notions for preimage and second-preimage resistance. We then analyze the generic security of hash functions with regard to these new properties against classical and quantum adversaries, proving upper and lower bounds on the query complexity of generic attacks.

More specifically, the first type of notions (single-function multi-target) models a notion that is implicitly used by recent collision-resilient hash-based signature schemes like XMSS, XMSS^{MT} and SPHINCS [13,24,6]. In these notions, an adversary $\mathcal{A}$ receives p target values and a random function from the hash function family. Then, $\mathcal{A}$ is asked to find a preimage (or second-preimage, respectively) for one of the target values under the given function. We prove that compared to standard (second-)preimage resistance, the query complexity of generic attacks drops by a factor p for classical and $\sqrt{p}$ for quantum adversaries. Then we introduce multi-function multi-target notions of preimage and second-preimage resistance. For these notions, $\mathcal{A}$ is given multiple pairs of function and target value, drawn independently at random. It is now $\mathcal{A}$'s goal to find a preimage (or second-preimage, respectively) for one of the target values under the associated function. We prove that in this case the query complexity of generic attacks is exactly the same as for the standard (single-function, single-target) notions.

Given that multi-function multi-target notions are as hard as the standard notions of preimage and second-preimage resistance we construct a new hash-based signature scheme with security based on these new notions. As the basic construction follows that of XMSS, we call the new scheme XMSS-T, indicating XMSS with tightened security. While XMSS looses in the bit security an amount linear in several parameters including the total tree height, XMSS-T looses only two bits, independent of any parameters. The differences between XMSS^{MT} and XMSS-T are a different hash tree and one-time signature scheme construction such that the security can be based on the multi-target multi-function properties. The basic change is that for every hash function call within a hash tree or a hash chain, a different hash function key and different bitmasks are used. Note that XMSS-T is stateful and it may be not suitable in some practical use cases. The good news is that we can make similar changes to the stateless hash-based signature scheme SPHINCS easily. Roughly speaking, it amounts to replacing the used hash trees and one-time signatures by the ones described in this work.

Finally, we present an implementation of XMSS-T and compare it to XMSS and XMSS^{MT}. We show that the applied changes only have marginal performance implications (a factor 3 loss in speed for all algorithms). Our code is available at `http://www.huelsing.net/code/xmss_ref.tar.gz`.

A remark on proving quantum generic security. At first sight the tasks of breaking the old and the newly introduced security notions for hash function families seem quite similar to some standard problems studied in quantum query complexity. However due to some limitations, existing results such as techniques for proving quantum query lower bounds [4,2] cannot be applied directly. First of all, quantum query complexity usually considers *worst-case* complexity only, whereas in cryptographic settings we care about average-case complexity. For example, there are famous works showing upper and lower bounds on finding collisions in r-to-1 functions [10,1]. Nonetheless, random functions, whose properties our work studies, are very unlikely to be r-to-1. Another issue is that, as observed by Zhandry [32], quantum query lower bounds often just have implications about quantum algorithms with *high* success probability. For cryptographic applications however, even an adversary with small but noticeable chance of breaking a scheme is still relevant. Therefore, a complete lower bound would be bounding the success probability of any algorithm making a specified number of queries. It might be possible to find fixes by digging into existing works, the situation is yet unclear. Therefore techniques developed in this work are interesting in their own right and may be useful in other cryptographic settings as well.

Organization. We introduce and discuss the new security notions for hash function families in Section 2, where detailed analysis for quantum generic security is presented in Secton 3. In Section 4 we present XMSS-T and discuss its security in Section 5. Finally, we present our implementation results in Section 6.

Notation. We write $x \xleftarrow{\$} \mathcal{X}$ if x is randomly chosen from the set $\mathcal{X}$ using the uniform distribution. We further write log for $\log_2$. We denote the uniform distribution over bit strings of length n by $\mathcal{U}_n$. We write $m = \text{poly}(n)$ to denote that m is a function, polynomial in n. We call a function $\epsilon(n) : \mathbb{N} \to [0,1]$ negligible and write $\epsilon(n) = \text{negl}(n)$ if for any $c \in \mathbb{N}, c > 0$ there exists a $n_c \in \mathbb{N}$ s.th. $\epsilon(n) < n^{-c}$ for all $n > n_c$.

2 New security notions for hash function families

In this section, we recall some known and define several new security notions for (hash) function families and discuss their security against both classical and quantum generic attacks. In the following we restrict ourselves to function families that operate on bit strings and have a fixed input size, as this is the case in our constructions. However, the definitions are the same for the more general case. In the following let $n \in \mathbb{N}$ be the security parameter, $m = \text{poly}(n), k = \text{poly}(n)$, and $\mathcal{H}_n = \{\text{H}_K : \{0,1\}^m \to \{0,1\}^n\}_{K \in \{0,1\}^k}$ be a family of functions. We say a function family $\mathcal{H}_n$ is efficient if there exists a probabilistic polynomial time (PPT) algorithm that evaluates $\text{H}_K(M)$ for any $M \in \{0,1\}^m$ and $K \in \{0,1\}^k$. We require all used functions to be efficient, unless we state otherwise. For hash-based signatures we are mainly interested in functions with $m, k \geq n$. However, we try to keep our results as general as possible and make it explicit whenever we are relying on $m, k \geq n$.

2.1 Defining the security notions

Preimage-resistance (OW). Let's revisit the standard notion of preimage resistance (a.k.a. one-wayness). We define the success probability of an adversary

$\mathcal{A}$ against the preimage resistance of a hash function family $\mathcal{H}_n$ as

$$\begin{aligned}\mathrm{Succ}^{\mathrm{OW}}_{\mathcal{H}_n}(\mathcal{A}) = \mathsf{Pr}[\, & K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m, Y \longleftarrow \mathrm{H}_K(M);\\ & M' \xleftarrow{\$} \mathcal{A}(K,Y) : Y = \mathrm{H}_K(M')] \ . \end{aligned} \tag{1}$$

Single-function, multi-target preimage resistance (SM-OW). We now define the success probability of an adversary against SM-OW. This is the basic multi-target notion of preimage resistance implicitly used by previous collision resilient hash-based signature schemes like XMSS. We show in Section 3 that this notion is significantly easier to attack than standard preimage resistance. The definition takes another parameter p defining the number of targets.

$$\begin{aligned}\mathrm{Succ}^{\mathrm{SM\text{-}OW}}_{\mathcal{H}_n,p}(\mathcal{A}) = \mathsf{Pr}[\, & K \xleftarrow{\$} \{0,1\}^k; M_i \xleftarrow{\$} \{0,1\}^m, Y_i \longleftarrow \mathrm{H}_K(M_i), 0 < i \leq p;\\ & M' \xleftarrow{\$} \mathcal{A}(K,(Y_1,\dots,Y_p)) : \exists 0 < i \leq p, Y_i = \mathrm{H}_K(M')] \ . \end{aligned} \tag{2}$$

Multi-function, multi-target preimage resistance (MM-OW). Next we define the success probability of an adversary $\mathcal{A}$ against MM-OW. This is the notion we are aiming for with XMSS-T as it is as hard to break as standard preimage resistance, as we will show below. Again the definition is parameterized by the number of targets:

$$\begin{aligned}\mathrm{Succ}^{\mathrm{MM\text{-}OW}}_{\mathcal{H}_n,p}(\mathcal{A}) = \mathsf{Pr}[\, & K_i \xleftarrow{\$} \{0,1\}^k, M_i \xleftarrow{\$} \{0,1\}^m, Y_i \longleftarrow \mathrm{H}_{K_i}(M_i), 0 < i \leq p;\\ & (j, M') \xleftarrow{\$} \mathcal{A}((K_1,Y_1),\dots,(K_p,Y_p)) : Y_j = \mathrm{H}_{K_j}(M')] \ . \end{aligned} \tag{3}$$

The difference between these two new definitions is that for SM-OW all targets are for the same function while for MM-OW each target has an associated random function from the family. We decided that $\mathcal{A}$ has to output the associated index i in case of MM-OW as otherwise any reduction would have to search for i and $\mathcal{A}$ knows i for any attack that does better than guessing.

Second-preimage resistance (SPR). After presenting the multi-target notions for one-wayness, we now turn to second-preimage resistance. We start revisiting the standard notion of second-preimage resistance. We define the success probability of an adversary $\mathcal{A}$ against the second-preimage resistance (SPR) of a hash function family $\mathcal{H}_n$ as

$$\begin{aligned}\mathrm{Succ}^{\mathrm{SPR}}_{\mathcal{H}_n}(\mathcal{A}) = \mathsf{Pr}[\, & K \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m;\\ & M' \xleftarrow{\$} \mathcal{A}(K,M) : M' \neq M \wedge \mathrm{H}_K(M) = \mathrm{H}_K(M')] \ . \end{aligned} \tag{4}$$

Note that in this definition the adversary is not promised to receive an M that actually has a second-preimage. Hence, especially for families $\mathcal{H}_n$ with $m = n$, i.e. same size of domain and co-domain, the adversaries success probability is largely influenced by the probability that a random M actually has a second-preimage.

Single-function, multi-target second-preimage resistance (SM-SPR). As for one-wayness, we define two multi-target notions: single-function multi-target second-preimage resistance (SM-SPR) and multi-function multi-target second-preimage resistance (MM-SPR). The first one (SM-SPR) is the notion implicitly used in XMSS. The latter is the notion we aim for with XMSS-T that is as hard to break as standard second-preimage resistance, as we will prove below. We start defining the success probability of an adversary against SM-SPR. The definition again takes another parameter p defining the number of targets:

$$\begin{aligned}\mathrm{Succ}^{\text{SM-SPR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \mathsf{Pr}\,[&K \xleftarrow{\$} \{0,1\}^k; M_i \xleftarrow{\$} \{0,1\}^m, 0 < i \le p;\\ &M' \xleftarrow{\$} \mathcal{A}(K, (M_1, \ldots, M_p)) :\\ &\exists 0 < i \le p : M' \ne M_i \wedge \mathrm{H}_K(M_i) = \mathrm{H}_K(M')]\ . \end{aligned} \tag{5}$$

Multi-function, multi-target second-preimage resistance (MM-SPR). Next we define the success probability of an adversary $\mathcal{A}$ against MM-SPR. Again the definition is parameterized by the number of targets:

$$\begin{aligned}\mathrm{Succ}^{\text{MM-SPR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \mathsf{Pr}\,[&K_i \xleftarrow{\$} \{0,1\}^k, M_i \xleftarrow{\$} \{0,1\}^m, 0 < i \le p;\\ &(j, M') \xleftarrow{\$} \mathcal{A}((K_1, M_1), \ldots, (K_p, M_p)) :\\ &M' \ne M_j \wedge \mathrm{H}_{K_j}(M_j) = \mathrm{H}_{K_j}(M')]\ . \end{aligned} \tag{6}$$

Extended target collision resistance (eTCR) In [21] Halevi and Krawczyk introduced extended target collision resistance (eTCR) as a hash function property that is close to target collision resistance. In the classical target-collision resistance game, the adversary is allowed to choose a target message M. Afterwards he learns a function (by learning a key K) and has to find a collision for the M under this function H_K. While the setup of the eTCR game is exactly the same, the adversary wins if he can present a new message M' and a (possibly new) key K' such that $\mathrm{H}_K(M) = \mathrm{H}_{K'}(M')$. Formally, the success probability of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ have shared memory, against eTCR is defined as follows:

$$\begin{aligned}\mathrm{Succ}^{\text{eTCR}}_{\mathcal{H}_n}(\mathcal{A}) = \mathsf{Pr}\,[&M \xleftarrow{\$} \mathcal{A}_1(1^n); K \xleftarrow{\$} \{0,1\}^k; (M', K') \xleftarrow{\$} \mathcal{A}_2(K, M) :\\ &M' \ne M \wedge \mathrm{H}_K(M) = \mathrm{H}_{K'}(M')]\ . \end{aligned} \tag{7}$$

Multi-target extended target collision resistance (M-eTCR) We can also define a multi target version (eTCR is inherently multi function anyway). To keep the definition readable we use a challenge oracle $\mathsf{Box}(\cdot)$ that on input of a message outputs a uniformly random function key. This oracle models the ability of $\mathcal{A}$ to adaptively obtain p eTCR challenges for the same function family. We denote by (M_i, K_i) the ith query-answer pair of $\mathsf{Box}(\cdot)$. The success probability of an adversary $\mathcal{A}$ against m-eTCR that makes no more than p queries to $\mathsf{Box}(\cdot)$ is defined as:

$$\begin{aligned}\mathrm{Succ}^{\text{M-ETCR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \mathsf{Pr}\,[\,(M',K',i) \xleftarrow{\$} \mathcal{A}^{\mathsf{Box}(\cdot)}(1^n) :\\ M' \neq M_i \wedge \mathrm{H}_{K_i}(M_i) = \mathrm{H}_{K'}(M')]\ .\end{aligned} \tag{8}$$

2.2 Generic security

To determine secure parameters for hash function families or constructions based on them, their security against generic attacks is analyzed. Generic attacks show which security level is achievable at all for a given property as they do not take any possibly existing function specific weaknesses into account. A hash function family is considered broken if the security level for one property is (significantly) lower than the generic security.

Classical generic security. The standard way to analyze the complexity of generic attacks against a security property of hash function families is analyzing the success probability of an adversary $\mathcal{A}$ against a random function family to which it is given black box access. The classical security is well understood in the literature. The security of the new notions we defined can be easily established as well. For completeness, we give brief justifications in Appendix A. Table 1 summarizes the classical and quantum generic security.

Quantum generic security. When we analyze the properties of hash functions under generic quantum attacks, we treat any hash function as a random function and the adversary can issue quantum superposition queries to the function. Namely, we are essentially working under the quantum random-oracle model [7]. When there are multiple functions, we assume they are independent random functions and the adversary can query them jointly in superposition. Namely, queries in the form of

$$\sum_{K,M,z} \alpha_{K,M,z} |K, M, z\rangle \mapsto \sum_{K,M,z} \alpha_{K,M,z} |K, M, z + H_K(M)\rangle\ ,$$

are permitted[4]. This choice is meant to capture the fact that in reality all hash functions are public, and a quantum adversary can certainly evaluate them jointly in superposition. This is in contrast to the classical setting, where each query must specify an index, and hence the adversary only gets one value of *one* function per query. One can define a similar model in the quantum setting (i.e., each query must specify one and only one function index K) and study all the security properties therein. We stress that this model seems weaker than the

[4] Alternatively, one can think of it as a global random function $(K, M) \mapsto O(K, M)$.

one we choose, and in particular our lower bounds results are hence stronger. Namely, they hold against stronger quantum attacks. It is an interesting theoretical question as to determining whether the two models are indeed different.

We prove our results regarding quantum generic security in Section 3. Our findings are summarized in Table 1. Please note that the constant hidden in the Θ is small, i.e. 16 for the lower bounds.

	OW, MM-OW, SPR, MM-SPR	SM-OW, SM-SPR	eTCR	M-eTCR
Classical	$\frac{q+1}{2^n}$	$\frac{(q+1)p}{2^n}$	$\frac{(q+1)}{2^n} + \frac{q}{2^k}$	$\frac{(q+1)p}{2^n} + \frac{qp}{2^k}$
Quantum	$\Theta(\frac{(q+1)^2}{2^n})$	$\Theta(\frac{(q+1)^2 p}{2^n})$	$\Theta(\frac{(q+1)^2}{2^n})$	$\Theta(\frac{(q+1)^2 p}{2^n})$

Table 1. Security against generic classical and quantum attacks. Entries represent the success probability of a q-query adversary.

3 Analyzing Quantum Generic Security

In the following we establish the generic security of hash function families against quantum attacks on the defined properties. For each security property, we give attacks and analyze their success probabilities. All attacks are based on Grover's quantum search algorithm, but we will need to analyze the complexity for random problem instances. More importantly, we establish matching lower bounds for all cases. The proofs of lower bounds follow a unified structure. Specifically, we first define a family of distributional search problems and bound the success probability of quantum algorithms against these problems. Then, we reduce various instances of the search problem to the task of breaking each of the security properties we care about. The hardness of the distributional search problems hence implies the generic security of hash functions for these security properties.

3.1 Toolbox

(Generalized) Grover's quantum search algorithm. One of the most useful algorithmic tools in quantum computing is Grover's quantum search algorithm and its many generalizations (e.g., [19,8,11,9] to name a few). Here we just need a simple version for searching a universe with multiple marked items. We state it in the following Lemma.

Lemma 1. *Let $f : \mathcal{X} \to \{0,1\}$ be an oracle function and let $\mathcal{X}_f = \{x \in \mathcal{X} : f(x) = 1\}$. Then there is a quantum algorithm* QSearch *with q queries that finds an $x \in \mathcal{X}_f$ with success probability $\Omega(q^2 \frac{|\mathcal{X}_f|}{|X|})$.*

Most of the attacks we describe later will apply QSearch in a straightforward way. However, since our problem instances are generated randomly, we will need to give a new analysis of the average-case performance.

A hard average-case search problem. It is well known that Grover's search algorithm is also optimal [5]. Namely, adopting notations from Lemma 1, any q-query algorithm can find a marked item with probability at most $O(q^2 \frac{|X_f|}{|X|})$. However since the security notions we defined all refer to average-case problems, the worst-case lower bound of Grover's search is not very useful. Here we introduce a distributional search problem, and prove a stringent hardness result.

Definition 1. *Let $\mathcal{F} := \{f : \{0,1\}^m \to \{0,1\}\}$ be the collection of all boolean functions on $\{0,1\}^m$. Let $\lambda \in [0,1]$ and $\varepsilon > 0$. Define a family of distributions D_λ on $\mathcal{F}$ such that $f \leftarrow_R D_\lambda$ satisfies*

$$f : x \mapsto \begin{cases} 1 & \text{with prob. } \lambda, \\ 0 & \text{with prob. } 1 - \lambda \end{cases}$$

for any $x \in \{0,1\}^m$.

We define $\mathsf{Avg\text{-}Search}_\lambda$ to be the problem that given oracle access to $f \leftarrow D_\lambda$, finds an x such that $f(x) = 1$. For any quantum algorithm $\mathcal{A}$ that makes q queries, we define

$$\mathsf{Succ}^q_\lambda(\mathcal{A}) := \Pr_{f \leftarrow D_\lambda}\left[f(x) = 1 : x \leftarrow \mathcal{A}^f(\cdot)\right].$$

Theorem 1. $\mathsf{Succ}^q_\lambda(\mathcal{A}) \leq 8\lambda(q+1)^2$ *holds for any quantum algorithm $\mathcal{A}$ with q queries.*

Note that this theorem matches the intuitive argument that for $f \leftarrow D_\lambda$, there are $2^m\lambda$ marked items on average and hence any quantum algorithm needs $\Theta(\sqrt{2^m/(2^m\lambda)}) = \Theta(1/\sqrt{\lambda})$ queries. A proof for a similar claim exists in [3, Lemma 37], but it is rather tedious. For completeness, we give a clean proof using a tool developed by Zhandry [31]. Here we present a simplified version of the tool, taken from [30].

Lemma 2. *[30, Theorem 7.2] Fix q, and let D_λ 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 $p_\lambda := \Pr_{f \leftarrow D_\lambda}[f(x_i) = y_i \forall i \in \{1, \ldots, 2q\}]$ is a polynomial of degree at most d in λ. Then any quantum algorithm $\mathcal{A}$ making q queries can only distinguish D_λ from D_0 with probability at most $2\lambda d^2$.*

Apply this lemma to the D_λ we defined earlier, we get

Lemma 3. *Let D_λ be defined as in Definition 1, and $\mathcal{A}$ be any quantum algorithm making at most q quantum queries. Then*

$$\mathsf{ADV}^q_{\mathcal{A}}(D_0, D_\lambda) \overset{def}{=} \left| \Pr_{f \leftarrow D_0}[\mathcal{A}^f(\cdot) = 1] - \Pr_{f \leftarrow D_\lambda}[\mathcal{A}^f(\cdot) = 1] \right| \leq 8\lambda q^2 .$$

Proof. Let $\{(x_i, y_i) \in \{0,1\}^m \times \{0,1\}\}_{i=1}^{2q}$ be an arbitrary collection of $2q$ pairs. Let k be the number of i with $y_i = 1$. By the definition of D_λ, we have that

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

It is easy to see that p_λ is a polynomial in λ with degree at most $2q$. Therefore we can apply Lemma 2 with $d = 2q$. We obtain that $\mathsf{ADV}^q_{\mathcal{A}}(D_0, D_\lambda) \leq 8\lambda q^2$.

Proof (Proof of Theorem 1). Observe that D_0 is the trivial function that maps every input to 0. Clearly if one can find a marked item (i.e. $x \in \{0,1\}^m$ with $f(x) = 1$) in $f \leftarrow D_\lambda$ it immediately distinguishes D_λ from D_0. Specifically assume there is an algorithm $\mathcal{A}$ that queries f and outputs x after q queries. With one extra query, one can check if $f(x) = 1$ and tell apart D_λ from D_0. Thus we obtain that $\mathsf{Succ}^q_{\mathcal{A}}(\lambda) \leq \mathsf{ADV}^{q+1}_{\mathcal{A}}(D_0, D_\lambda) \leq 8\lambda(q+1)^2$.

Simulating random functions In our reductions to show lower bounds, we usually assume we have access to some random function $f : \mathcal{X} \to \mathcal{Y}$. Ultimately, we will need to simulate f efficiently so that any algorithm with q queries cannot notice a difference. Fortunately, the following claim allows us to do so by sampling uniformly from a $2q$*-wise independent hash function family* $\mathcal{H}$.

Lemma 4. *[31, Theorem 6.1] For any quantum adversary that makes no more than q queries to either a truly random function or a function drawn uniformly from $\mathcal{H}$, the final states are identical.*

There exists a vast literature on efficient constructions of t-wise independent hash functions. Interested readers are referred to, e.g., [25,15,26]. There is a technical subtlety though. Most constructions of $\mathcal{H}$ consider output space $\mathcal{Y}$ with size being a prime or a prime power. We need one with $\mathcal{Y} = [N], N = 2^n - 1$. A natural approach is to pick a prime $M >> N$ and construct a $2q$-wise independent family $\mathcal{H}_0 : \mathcal{X} \to [M]$. Then we would expect that $\mathcal{H} : x \mapsto \mathcal{H}_0(x) \bmod N$ will suffice for our purpose, modulo a tiny error. However we were unable to identify a rigorous proof in the literature for the correctness of this "mod" construction, especially with respect to quantum attacks. We show such a proof formally in Appendix B.

Sometimes we need a random function f that excludes some output $y \in \mathcal{Y}$. This is easy to realize as follows. We take a random function $g : \mathcal{X} \to [k]$ where $k = |\mathcal{Y}| - 1$. Then $f(x)$ will be obtained by applying g on x and then mapping the outcome to $\mathcal{Y} \backslash y$ according to some canonical isomorphism (e.g., any thing smaller than y remains unchanged, and anything else is incremented by 1.).

3.2 Hardness of breaking the security: lower bounds

We analyze in this section the hardness of generic quantum attacks on the various notions of hash functions. Basically, we reduce $\mathsf{Avg\text{-}Search}_\lambda$ with various λ to the task of breaking the security notion generically. The hardness of $\mathsf{Avg\text{-}Search}$

then implies the security against generic quantum attacks. The bounds for OW, SPR and their variants are given in Propositions 1 and 2. While the proofs are quite similar we have to deal with a restriction for the OW notions that we did not figure out how to circumvent. Namely, we require that $2^m \gg 2^n$ (e.g. $m = 2n$) and $p \ll 2^n$, which is the case for most relevant hash function families. The complexity for eTCR and M-eTCR involves additional technical difficulty concerning programming a random oracle, and we analyze them in Proposition 3.

Proposition 1. *Let $m = cn$ for a positive real constant $c \in \mathbb{R}^+$ and $p = o(n)$. For any quantum adversary with q queries, it holds that*

$$\mathrm{Succ}^{\text{OW}}_{\mathcal{H}_n}(\mathcal{A}) = O((q+1)^2/2^n), \mathrm{Succ}^{\text{SM-OW}}_{\mathcal{H}_n}(\mathcal{A}) = O((q+1)^2 p/2^n),$$
$$\mathrm{Succ}^{\text{MM-OW}}_{\mathcal{H}_n}(\mathcal{A}) = O((q+1)^2/2^n)\,.$$

Proposition 2. *For any quantum adversary with q queries, it holds that*

$$\mathrm{Succ}^{\text{SPR}}_{\mathcal{H}}(\mathcal{A}) = O((q+1)^2/2^n), \mathrm{Succ}^{\text{SM-SPR}}_{\mathcal{H}_n}(\mathcal{A}) = O((q+1)^2 p/2^n),$$
$$\mathrm{Succ}^{\text{MM-SPR}}_{\mathcal{H}_n}(\mathcal{A}) = O((q+1)^2/2^n)\,.$$

We give the proof for MM-OW. The others can be proven analogously and we describe the reductions in Appendix C.

Proof (Hardness of MM-OW*).* Given an Avg-Search instance, we construct an instance of MM-OW in Figure 1:

Given: $f \leftarrow D_\lambda : [p] \times \{0,1\}^m \to \{0,1\}^n$, $\lambda = 1/2^n$.

1. For $i = 1, \ldots, p$, sample $y_i \leftarrow \{0,1\}^n$ independently and uniformly at random. Denote $S = \{y_i\}_{i=1}^p$.
2. For $i = 1, \ldots, p$, let $g_i : \{0,1\}^m \to \{0,1\}^n \backslash y_i$ be independent random functions. Construct $H_i : \{0,1\}^m \to \{0,1\}^n$ as follows: for any $x \in \{0,1\}^m$

$$x \mapsto \begin{cases} y_i, & \text{if } f(i\|x) = 1, \\ g_i(x) & \text{otherwise.} \end{cases}$$

Output: MM-OW instance $(S, \{H_i\})$. Namely an adversary is given $\{y_i\}$ and oracle access to $\{H_i\}$, and the goal is to find (i^*, x^*) such that $H_{i^*}(x^*) = y_{i^*}$.

Fig. 1. Reducing Avg-Search to MM-OW

Note that the way that f is generated ensures that the constructed H_i is distributed identically to a uniformly random function $H : \{0,1\}^m \to \{0,1\}^n$. However the joint distribution $(\{y_i\}, \{H_i\})$ has a slight discrepancy from that in the definition Eq. 3. This is because y_i may have no preimages but in the definition existence of preimages is guaranteed. Nonetheless, we can argue that (see Appendix D) in the regime we are interested in, i.e. $2^m \gg 2^n$ (e.g. $m = 2n$) and $p \ll 2^n$, this only incurs a negligible error. This implies that any q-query attacker solving MM-OW will give rise to a $2q$-query algorithm for $\mathsf{Avg\text{-}Search}_\lambda$. As a consequence

$$\mathrm{Succ}^{\text{MM-OW}}_{\mathcal{H}_n}(\mathcal{A}) \le \mathrm{ADV}^{2q}_{\mathcal{A}}(\lambda) \le 16(q+1)^2/2^n\,,$$

according to Theorem 1. We remark that, as mentioned in Sect. 3.1, $\tilde{H}_i$ can be implemented efficiently.

Proposition 3. *Let $\varepsilon = 8q^2/2^n$ and $\delta = 4q^2/2^k$. For any quantum adversary with q queries, it holds that*

$$\mathrm{Succ}^{\text{eTCR}}_{\mathcal{H}_n}(\mathcal{A}) \le \varepsilon + 2\delta, \quad \mathrm{Succ}^{\text{M-eTCR}}_{\mathcal{H}_n}(\mathcal{A}) \le p(\varepsilon + 2\delta)\,.$$

To prove the proposition, we need a lemma that allows us to adaptively program a quantum random oracle. The proof follows standard techniques (see similar analyses for different scenarios in [29,17]). Let $\mathcal{A}$ be an arbitrary quantum algorithm and let $H : \{0,1\}^m \times \{0,1\}^k \to \{0,1\}^n$ be a random function. Consider two games as follows:

- Game G_0: $\mathcal{A}$ gets access to H. In phase 1, after making at most q_1 queries to H, $\mathcal{A}$ outputs a message $M \in \{0,1\}^m$. Then a random $\hat{K} \in_R \{0,1\}^k$ is sampled and $(\hat{K}, H_{\hat{K}}(M))$ is handed to $\mathcal{A}$. $\mathcal{A}$ continues to the second phase and makes at most q_2 queries. $\mathcal{A}$ outputs $b \in \{0,1\}$ at the end.
- Game G_1: $\mathcal{A}$ gets access to H. After making at most q_1 queries to H, $\mathcal{A}$ outputs a message $M \in \{0,1\}^m$. Then a random $\hat{K} \in_R \{0,1\}^k$ is sampled as well as a random range element $y \in_R \{0,1\}^n$. Program $H_{\hat{K}}(M) = y$ and call the new oracle H'. $\mathcal{A}$ receives $(\hat{K}, y = H'_{\hat{K}}(M))$ and proceeds to the second phase. After making at most q_2 queries, $\mathcal{A}$ outputs $b \in \{0,1\}$ at the end.

Lemma 5. $|\Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_1) = 1]| \le 2\delta$*, with* $\delta = 4q^2/2^k$.

Proof. We use a hybrid argument. Consider the following (hybrid) games:

- Hyb_0: it is identical to G_0 except that $\hat{K} \leftarrow \{0,1\}^k$ is sampled at the very beginning. It does not make any difference and $\Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1] = \Pr[\mathcal{A}(G_0) = 1]$. (In short: H/H)
- Hyb_1: During phase 1 we define $\bar{H}_K(x) = H_{\hat{K}}(x)$ if $K \neq \hat{K}$ and $\bar{H}_{\hat{K}}(*) = 0$. The remaining of the game follows exactly as Hyb_0. (In short: $\bar{H}/H$)
- Hyb_2: At then end of Phase 1, we program H to H' so that $(\hat{K}, M) \mapsto y$ and proceeds to Phase 2 using H'. (In short: $\bar{H}/H'$)
- Hyb_3: the only difference from Hyb_2 is that in Phase 1, we use H instead of $\bar{H}$. Note that Hyb_3 is exactly G_1, and hence $\Pr[\mathcal{A}(\mathsf{Hyb}_3) = 1] = \Pr[\mathcal{A}(G_1) = 1]$. (In short: H/H')

It is easy to observe that $\Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1] = \Pr[\mathcal{A}(\mathsf{Hyb}_2) = 1]$, because an adversary cannot tell any difference between H and $H' : (\hat{K}, M) \mapsto y$ if he never queried any $(\hat{K}, *)$ before. Next we show that

$$(*) \quad |\Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1]| \leq \delta\,.$$

The same argument will show that $|\Pr[\mathcal{A}(\mathsf{Hyb}_2) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_3) = 1]| \leq \delta$. Therefore we can prove Lemma 5.

The proof of $(*)$ is a generalization of [5, Theorem 3.3] to an average-case, a computational version of which was given in [17, Lemma 6]. Intuitively, since $\hat{K} \in_R K$ is sampled uniformly at random, the overall amplitudes of $\mathcal{A}$'s queries in Phase 1 of the form $|\hat{K}, *\rangle$ will be tiny. Therefore, one can change the values of H on these inputs without causing noticeable effect. To make it formal, we first claim that two oracles $f_{\hat{K}} : \{0,1\}^k \to \{0,1\}$ where $f_{\hat{K}}(K) = 1$ iff. $K = \hat{K}$ and the all-zero function $f_0(K) = 0, \forall K \in \{0,1\}^k$ are hard to distinguish for a randomly chosen $\hat{K}$ even if $\hat{K}$ is released after the adversary makes the last query. Namely we have:

$$\begin{aligned}(**) \quad &\Big|\Pr[b = 1 : \hat{K} \leftarrow \{0,1\}^k, \mathcal{A}_1^{f_{\hat{K}}}(\cdot), b \leftarrow \mathcal{A}_2(\hat{K})] \\ &- \Pr[b = 1 : \hat{K} \leftarrow \{0,1\}^k, \mathcal{A}_1^{f_0}(\cdot), b \leftarrow \mathcal{A}_2(\hat{K})]\Big| \\ &\leq 4q^2/2^k =: \delta\end{aligned}$$

holds where $\mathcal{A}_1$ is any q-query algorithm and $\mathcal{A}_2$ receives the final state of $\mathcal{A}_1$ and $\hat{K}$. This follows by a simple adaption of the standard proof of the lower bound for Grover's search problem.

Now assume there is $\mathcal{A}$ such that

$$|\Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1]| \geq \delta\,.$$

We construct $(\mathcal{A}_1, \mathcal{A}_2)$ that violates (**). Given f, which is either $f_{\hat{K}}$ with uniformly random $\hat{K}$ or f_0, we construct $H : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$ such that: $H(K, M) = g(K, M)$ if $f(K) = 0$ and $H(K, M) = 0$ otherwise. Here g is a random function chosen from $\{\{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n\}$. $\mathcal{A}_1$ then plays the games we defined above by simulating $\mathcal{A}$ and using H in Phase 1. At the end of Phase 1 $\hat{K}$ is released, then $\mathcal{A}_2$ uses $\hat{K}$ and continues simulating $\mathcal{A}$ in Phase 2. Finally $\mathcal{A}_2$ outputs whatever $\mathcal{A}$ outputs. Observe that we get Hyb_0 if $f = f_0$ and get Hyb_1 if $f = f_{\hat{K}}$. Therefore

$$\begin{aligned}&\Big|\Pr[b = 1 : \hat{K} \leftarrow \{0,1\}^k, \mathcal{A}_1^{f_{\hat{K}}}(\cdot), b \leftarrow \mathcal{A}_2(\hat{K})] \\ &- \Pr[b = 1 : \hat{K} \leftarrow \{0,1\}^k, \mathcal{A}_1^{f_0}(\cdot), b \leftarrow \mathcal{A}_2(\hat{K})]\Big| \\ = &|\Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1]| \geq \delta\,,\end{aligned}$$

which gives a contradiction.

Proof (Proof of Proposition 3). We give a reduction from Avg-Search to breaking eTCR. Assume that there is $\mathcal{A}$ that breaks eTCR with probability η. We construct an adversary $\mathcal{A}'$ that solves Avg-Search with probability $\eta - 2\delta$. Note that as long as $\mathcal{A}$ does not notice that we reprogrammed H, its view would be identical to that of the standard eTCR game, and by assumption $\mathcal{A}$ wins with probability at least η. By Lemma 5, reprogramming only incurs an additive error $2\delta = 4q^2/2^k$. We claim that $\Pr[f(K^*, M^*) = 1] \geq \eta - 2\delta$. But we know that the success probability of Avg-Search is at most $\varepsilon := 8q^2/2^n$ by Lemma 3. Therefore $\eta \leq \varepsilon + 2\delta$ and this proves Proposition 3. We can generalize the arguments above to the multi-target case easily.

Given: $f \leftarrow D_\lambda : \mathcal{X} := \{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^n$, $\lambda = 1/N$.

1. Sample $y \leftarrow \{0,1\}^n$ uniformly at random.
2. Let $g : \mathcal{X} \rightarrow \{0,1\}^n \setminus \{y\}$ be a random function. Construct $H : \mathcal{X} \rightarrow \{0,1\}^n$ as follows: for any $x \in \mathcal{X}$
$$x \mapsto \begin{cases} y & \text{if } f(x) = 1, \\ g(x) & \text{otherwise.} \end{cases}$$
3. $\mathcal{A}$ accesses H and issues q_1 queries. $\mathcal{A}$ outputs M at end of Phase 1.
4. Sample $K \leftarrow \{0,1\}^k$. Program $H_K(M) = y$. Denote the new oracle H'. Send (K, y) to $\mathcal{A}$.
5. $\mathcal{A}$ makes q_2 queries to H'. Outputs (K^*, M^*).

Output: (K^*, M^*).

Fig. 2. Reducing Avg-Search to eTCR

3.3 Quantum attacks: upper bounds

In this section, we apply quantum search algorithm QSearch to attack the various notions generically. In most cases, we get matching upper bounds.

Proposition 4. *There exist quantum adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_8$ all with $O(q)$ queries, such that*

$$\begin{gathered} \mathrm{Succ}^{\text{OW}}_{\mathcal{H}_n}(\mathcal{A}_1) = \Omega(q^2/2^n)\,, \mathrm{Succ}^{\text{SM-OW}}_{\mathcal{H}_n}(\mathcal{A}_2) = \Omega(q^2p/2^n)\,, \\ \mathrm{Succ}^{\text{MM-OW}}_{\mathcal{H}_n}(\mathcal{A}_3) = \Omega(q^2/2^n)\,; \mathrm{Succ}^{\text{SPR}}_{\mathcal{H}_n}(\mathcal{A}_4) = \Omega(q^2/2^n)\,, \\ \mathrm{Succ}^{\text{SM-SPR}}_{\mathcal{H}_n}(\mathcal{A}_5) = \Omega(q^2p/2^n)\,, \mathrm{Succ}^{\text{MM-SPR}}_{\mathcal{H}_n}(\mathcal{A}_6) = \Omega(q^2/2^n)\,; \\ \mathrm{Succ}^{\text{eTCR}}_{\mathcal{H}_n}(\mathcal{A}_7) = \Omega(q^2/2^n)\,, \mathrm{Succ}^{\text{M-eTCR}}_{\mathcal{H}_n}(\mathcal{A}_8) = \Omega(q^2p/2^n)\,. \end{gathered}$$

For ease of presentation, we give proofs for preimage-resistance and SM-OW. This should demonstrate the main idea, and the rest can be proved similarly.

Proof (Quantum attack on OW*).* We describe a $O(q)$-query attacker $\mathcal{A}_1$ as follows. Given y and oracle access to H, $\mathcal{A}_1$ will apply QSearch to search for x such that $H(x) = y$. More specifically, $\mathcal{A}_1$ constructs $g_H : \{0,1\}^m \to \{0,1\}$ such that $g_H(x) = 1$ iff. $H(x) = y$. Each evaluation on g_H can be realized efficiently by two queries to h. For any $h \in \mathcal{H}$, let $p_h := \Pr_{H \leftarrow \mathcal{H}}[H = h]$ and let $X_H = |H^{-1}(y)|$ be the random variable representing the preimage size of y. Then by Lemma 1 we can see that

$$\begin{aligned}\mathrm{Succ}^{\mathrm{OW}}_{\mathcal{H}_n}(\mathcal{A}_1) &= \sum_h p_h \cdot \Omega(q^2 \frac{X_h}{2^m}) = \Omega(\frac{q^2}{2^m} \sum_h p_h X_h) \\ &= \Omega(\frac{q^2}{2^m} \cdot \mathbb{E}(X_H)) = \Omega(\frac{q^2}{2^n}).\end{aligned}$$

In the last step we observe, by linearity of expectation, that

$$\mathbb{E}(X_H) = \sum_{x \in X} \Pr_{H \leftarrow \mathcal{H}}[H(x) = y] = 2^m/2^n\,.$$

Proof (Quantum attack on SM-OW*).* We describe a $O(q)$-query attacker $\mathcal{A}_2$ as follows. Given $y_1, \ldots, y_p$ and oracle access to H, $\mathcal{A}_2$ will apply QSearch to search for x such that $H(x) \in \{y_i\}_{i=1}^p$. Let $X_H = |\bigcup_{i=1}^p \{H^{-1}(y_i)\}|$ be the random variable representing the preimage size of all y_i. Similar calculation as above tells us that

$$\mathrm{Succ}^{\mathrm{SM\text{-}OW}}_{\mathcal{H}_n}(\mathcal{A}_2) = \Omega(\frac{q^2}{2^m}) \cdot \mathbb{E}(X_H)\,.$$

Finding $\mathbb{E}(X_H)$ is a bit more complicated than above. Consider the probabilistic space induced by random choices of $x_i \leftarrow \{0,1\}^m$ and $H \leftarrow \mathcal{H}_n$. For each $j = 1, \ldots, p$, let E_j be the event that $T := \{y_i = H(x_i)\}_{i=1}^p$ contains j distinct values in $\{0,1\}^n$. Then for any $x \in \{0,1\}^m$, define $\mathbb{I}_x := 1$ if $H(x) \in T$ and $\mathbb{I}_x := 0$ otherwise. We get that

$$\mathbb{E}(\mathbb{I}_x) = \Pr[H(x) \in T] = \sum_j \Pr[H(x) \in T | E_j] \cdot \Pr[E_j]\,,$$

Observe that $\Pr[H(x) \in T | E_j] = j/2^n$ and $Pr[E_j] = \binom{2^n}{j} / \sum_{k=1}^p \binom{2^n}{k}$. Therefore $\mathbb{E}(\mathbb{I}_x) = \alpha_p/2^n$ where $\alpha_p := \frac{\sum_{j=1}^p j \cdot \binom{2^n}{j}}{\sum_{k=1}^p \binom{2^n}{k}}$.

Then we obtain that $\mathbb{E}(X_H) = \sum_{x \in \{0,1\}^m} \mathbb{E}(\mathbb{I}_x) = \alpha_p 2^m/2^n$ by linearity of expectation. We are only interested when $p \ll 2^n$, in which case $\alpha_p \approx p$, and hence we have that

$$\mathrm{Succ}^{\mathrm{SM\text{-}OW}}_{\mathcal{H}_n}(\mathcal{A}_2) = \Omega(\frac{q^2}{2^m}) \cdot \mathbb{E}(X_H) = \Omega(\frac{pq^2}{2^n})\,.$$

Proof (Quantum attacks on eTCR *and* m-eTCR*).* Consider an adversary $\mathcal{A}_7$ who simply picks an arbitrary message M in Phase 1, and after receiving K applies QSearch to find (K', M') such that $H(K', M') = H(K, M)$ with $M \neq M'$. Following a similar calculation as the proof of Proposition 4, the success probability of $\mathcal{A}$ will be

$$\mathrm{Succ}^{\mathrm{eTCR}}_{\mathcal{H}_n}(\mathcal{A}_7) = \Omega((1 - 1/2^m)\frac{q^2}{2^n}) .$$

This will match the bound in Proposition 3 except with error $O(1/N)$ whenever $M, K \gtrsim N$ are not significantly smaller than N. As to the multi-round version (m-eTCR), consider a similar attacker $\mathcal{A}_8$ who picks $M_1, \ldots, M_p$ arbitrarily (assuming they are distinct) and receiving random $K_i, i = 1, \ldots, p$. At the end $\mathcal{A}_8$ invokes QSearch to find (K', M') such that $\exists i \in [p]$ with $M \neq M_i$ and $H(K', M') = H(K_i, M_i)$. By a similar derivation, we get that

$$\mathrm{Succ}^{\mathrm{m\text{-}eTCR}}_{\mathcal{H}_n}(\mathcal{A}_8) = \Omega(p(1 - p/2^m)\frac{q^2}{2^n}) .$$

Again, this matches the lower bound in the regime where $m, k \gtrsim n$ and $p \ll 2^n$.

4 XMSS-T

The eXtended Merkle Signature Scheme (XMSS) was proposed by Buchmann, Dahmen, and Hülsing in [13]. The original proposal for XMSS essentially combines a collision-resilient version of the Winternitz one-time signature scheme (WOTS) from [12] with the collision-resilient hash tree construction from [16] and adds two different kinds of pseudorandom key generation, one leading an EU-CMA-secure and one a forward-secure signature scheme. Under the name XMSS^{MT} Hülsing, Rausch, and Buchmann [24] later proposed a multi-tree version of XMSS.

In this work we introduce XMSS-T, XMSS with tightened security. In contrast to XMSS, XMSS-T avoids multi-target attacks. To this end, XMSS-T uses a new hash tree construction and a new WOTS variant WOTS-T. XMSS-T is based on XMSS^{MT}. The main difference in the construction of XMSS^{MT} and XMSS-T is the use of independent function keys and bitmasks for *every* call to a hash function inside of the hash trees or WOTS-T. XMSS^{MT} used a single fixed key per function family and the same bitmask per internal tree level or chain position. The function keys and bitmasks used by XMSS-T are needed for verification. To keep the public key small these values are generated pseudorandomly, using a hash-based pseudorandom function family and a seed value that becomes part of the public key. In the following we describe XMSS-T.

Parameters. XMSS-T uses several parameters and several functions. The main security parameter is $n \in \mathbb{N}$, the message digest length $m \in \mathrm{poly}(n)$. The functions include two keyed, short-input cryptographic hash functions $\mathrm{F} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ and $\mathrm{H} : \{0,1\}^n \times \{0,1\}^{2n} \rightarrow \{0,1\}^n$; one arbitrary-input randomized hash function $\mathcal{H} : \{0,1\}^m \times \{0,1\}^* \rightarrow \{0,1\}^m$; and two ensembles of pseudorandom function families $\mathcal{F}_n : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^n$, $\mathcal{F}_m : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^m$, where we denote by $\{0,1\}^*$ the ability to handle arbitrary input lengths up to some practical limit (e.g. 2^{64} bits as in the case of the SHA family). Of course, these functions can all be built from a single cryptographic hash function, but the security analysis gets easier separating the functions according to the required properties.

XMSS-T uses a hyper-tree (a tree of trees) of total height $h \in \mathbb{N}$, where h is a multiple of d and the hyper-tree consists of d layers of trees, each having height h/d. WOTS allows for a space-time trade-off using the Winternitz parameter $w \in \mathbb{N}, w > 1$. The Winternitz parameter w and the length of the bit string that is signed λ determine ℓ the number of function chains for WOTS:

$$\ell_{1,\lambda} = \left\lceil \frac{\lambda}{\log(w)} \right\rceil, \quad \ell_{2,\lambda} = \left\lfloor \frac{\log(\ell_1(w-1))}{\log(w)} \right\rfloor + 1, \quad \ell_\lambda = \ell_1 + \ell_2.$$

The bit strings signed using WOTS are the m-bit message digests on the lowest layer and the n-bit root nodes of the layer below on all other layers.

As a running example we present concrete numbers for XMSS-T-256; the choices are explained in Section 6. For XMSS-T-256 we use $n = 256, m = 316, h = 60, d = 3, w = 16$ which leads to $\ell_n = 67$ and $\ell_m = 82$.

Addressing scheme. XMSS-T requires an addressing scheme for hash function calls. Every addressing scheme that assigns to every call to either F or H within the virtual structure of a XMSS-T hyper-tree a unique address can be used (e.g. numbering all the calls in some order). We suggest to use a recursive addressing scheme that numbers sub-structures (e.g. a OTS key pair) inside a structure (e.g. a tree). The addressing scheme generates an address for a substructure, taking the address of the structure and appending the index of the substructure. For trees, which contain three different kinds of substructures (OTS key pairs, L-trees, and nodes), an additional identifier for the type of substructure is added. Below we assume that a function $\textsc{GenAddr}(\mathbf{a}_s, \mathit{index})$ exists that takes the address of the structure and the index of the substructure and outputs a unique address for this substructure within an XMSS-T key pair. The advantage of this addressing scheme is that it only uses information that is available when the hash call is executed.

The addressing scheme is publicly known and the same addresses can be used for all XMSS-T key pairs. The resulting addresses are used as inputs to PRF $\mathcal{F}_n$ to pseudorandomly generate function keys and bitmasks.

WOTS-T. We now describe the new WOTS version. The construction differs from [22] in that it uses fresh keys and bitmasks for each hash function call. We denote the message length by $\lambda \in \{n, m\}$ and to improve readability we write ℓ, ℓ_1, and ℓ_2 instead of $\ell_\lambda, \ell_{1,\lambda}$, and $\ell_{2,\lambda}$. We include pseudorandom key generation, meaning that a seed value takes the place of a secret key in our description. We describe the algorithms as used by XMSS-T, hence, they take global secret and public information. For a standalone version, this information would have to be generated during key generation.

The difference between all WOTS variants is in the way the so called chaining function is constructed. WOTS-T uses the function F to construct the following chaining function:

Chaining function $c^{i,j}(x, \mathbf{a}_{\mathcal{C}}, \textsc{Seed})$*:* On input of value $x \in \{0,1\}^n$, iteration counter $i \in \mathbb{N}$, start index $j \in \mathbb{N}$, chain address $\mathbf{a}_{\mathcal{C}}$, and (public) seed $\textsc{Seed}$, the chaining function works the following way. In case $i = 0$, c returns x, i.e., $c^{0,j}(x, \mathbf{a}_{\mathcal{C}}, \textsc{Seed}) = x$. For $i > 0$ we define c recursively as

$$c^{i,j}(x, \mathbf{a}_{\mathcal{C}}, \textsc{Seed}) = \mathrm{F}(k_{i,j}, c^{i-1,j}(x, \mathbf{a}_{\mathcal{C}}, \textsc{Seed}) \oplus r_{i,j}),$$

where key $k_{i,j} = \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\mathcal{C}}, 2\cdot(j+i)))$ and bitmask $r_{i,j} = \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\mathcal{C}}, 2\cdot(j+i)+1))$. I.e. in every round, the function first takes the bitwise xor of the previous value $c^{i-1,j}(x, \mathbf{a}_{\mathcal{C}}, \textsc{Seed})$ and bitmask $r_{i,j}$ and evaluates F with key $k_{i,j}$ on the result.

Now we describe the three algorithms of WOTS-T.

Key Generation Algorithm $((\mathsf{sk}, \mathsf{pk}) \longleftarrow \mathrm{WOTS}.\mathsf{kg}(\mathcal{S}, \mathbf{a}_{\mathrm{OTS}}, \textsc{Seed}))$*:* On input of a global secret key seed $\mathcal{S} \in \{0,1\}^n$ (used for every WOTS-T keypair within a XMSS-T keypair), the address of the WOTS-T keypair within a tree $\mathbf{a}_{\mathrm{OTS}}$, and public seed $\textsc{Seed}$, the key generation algorithm computes the internal secret key $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$ as $\mathsf{sk}_i \longleftarrow \mathcal{F}_n(\mathcal{S}, \textsc{GenAddr}(\mathbf{a}_{\mathrm{OTS}}, i))$, i.e., the ℓ n bit secret key elements are derived form the secret key seed using the address of the chain they are contained in. The public key pk is computed as

$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) = (c^{w-1,0}(\mathsf{sk}_1, \mathbf{a}_{\mathcal{C}_1}, \textsc{Seed}), \ldots, c^{w-1,0}(\mathsf{sk}_\ell, \mathbf{a}_{\mathcal{C}_\ell}, \textsc{Seed})),$$

where $\mathbf{a}_{\mathcal{C}_i} = \textsc{GenAddr}(\mathbf{a}_{\mathrm{OTS}}, i)$. Note that $\mathcal{S}$ requires less storage than sk; thus we generate sk and pk on the fly when necessary.

Signature Algorithm $(\sigma \longleftarrow \mathrm{WOTS}.\mathsf{sign}(M, \mathcal{S}, \mathbf{a}_{\mathrm{OTS}}, \textsc{Seed}))$*:* On input of a λ-bit message M, the global secret key seed $\mathcal{S} \in \{0,1\}^n$, the address of the WOTS-T keypair within a tree $\mathbf{a}_{\mathrm{OTS}}$, and public seed $\textsc{Seed}$, the signature algorithm first computes a base-w representation of M: $M = (M_1 \ldots M_{\ell_1})$, $M_i \in \{0, \ldots, w-1\}$. That is, M is treated as the binary representation of a natural number x and then the w-ary representation of x is computed. Next it computes the checksum $C = \sum_{i=1}^{\ell_1}(w - 1 - M_i)$ and its base w representation $C = (C_1, \ldots, C_{\ell_2})$. The length of the base w representation of C is at most ℓ_2 since $C \leq \ell_1(w-1)$. We set $B = (b_1, \ldots, b_\ell) = M \parallel C$, the concatenation of the base w representations of M and C. Then the internal secret key is generated using $\mathsf{sk}_i \longleftarrow \mathcal{F}_n(\mathcal{S}, \textsc{GenAddr}(\mathbf{a}_{\mathrm{OTS}}, i))$ the same way as during key generation. The signature is computed as

$$\sigma = (\sigma_1, \ldots, \sigma_\ell) = (c^{b_1,0}(\mathsf{sk}_1, \mathbf{a}_{\mathcal{C}_1}, \textsc{Seed}), \ldots, c^{b_\ell,0}(\mathsf{sk}_\ell, \mathbf{a}_{\mathcal{C}_\ell}, \textsc{Seed})),$$

where $\mathbf{a}_{\mathcal{C}_i} = \textsc{GenAddr}(\mathbf{a}_{\mathrm{OTS}}, i)$ as above.

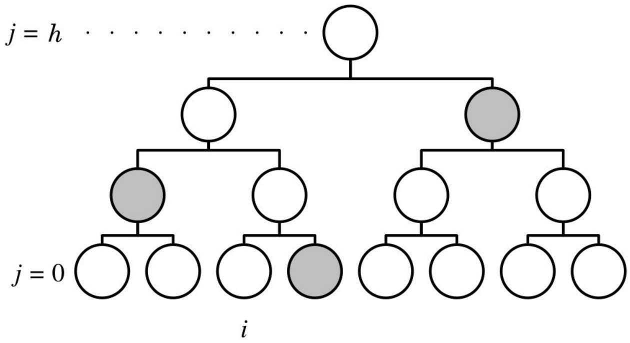

Fig. 3. The authentication path for leaf i.

Verification Algorithm ($\mathsf{pk}' \longleftarrow \mathrm{WOTS}.\mathsf{vf}(M, \sigma, \mathbf{a}_{\mathrm{OTS}}, \textsc{Seed})$)*:* On input of a λ-bit message M, a signature σ, the address of the WOTS-T keypair within a tree $\mathbf{a}_{\mathrm{OTS}}$, and public seed $\textsc{Seed}$, the verification algorithm first computes the b_i, $1 \le i \le \ell$ as described above. Then it returns:

$$\mathsf{pk}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell) = (c^{w-1-b_1,b_1}(\sigma_1, \mathbf{a}_{\mathcal{C}_\ell}, \textsc{Seed}), \ldots, c^{w-1-b_\ell,b_\ell}(\sigma_\ell, \mathbf{a}_{\mathcal{C}_\ell}, \textsc{Seed})).$$

A formally correct verification algorithm would compare pk' to a given public key and output true on equality and false otherwise. In XMSS-T this comparison is delegated to the overall verification algorithm.

Binary Hash Trees. The central elements of a Merkle tree signature scheme are full binary hash trees. We use a new construction that allows multi-target-attack resilience. In XMSS-T, a binary hash tree of height h always has 2^h leaves which are n bit strings L_i, $i \in [2^h - 1]$. Each node $N_{i,j}$, for $0 < j \le h$, $0 \le i < 2^{h-j}$, of the tree stores an n-bit string. For the leaf nodes define $N_{i,0} = L_i$. The values of the internal nodes $N_{i,j}$ are computed as

$$N_{i,j} = \mathrm{H}_{k_{i,j}}((N_{2i,j-1} \| N_{2i+1,j-1}) \oplus (r_{i,j})),$$

where key $k_{i,j} = \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\textsc{Tree}}, 4 \cdot (j+i)))$ and bitmask $r_{i,j} = (\mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\mathcal{C}}, 4 \cdot (j+i)+1)) \| \mathcal{F}_n(\textsc{Seed}, \textsc{GenAddr}(\mathbf{a}_{\mathcal{C}}, 4 \cdot (j+i)+2)))$. We also denote the root as $\textsc{Root} = N_{0,h}$.

An important notion is the authentication path $\mathsf{Auth}_i = (\mathsf{A}_0, \ldots, \mathsf{A}_{h-1})$ of a leaf L_i shown in Figure 3. Auth_i consists of all the sibling nodes of the nodes contained in the path from L_i to the root. For a discussion on how to compute authentication paths, see Section 6. Given a leaf L_i together with its authentication path Auth_i, the root of the tree can be computed using Algorithm 1.

L-Tree. In addition to the full binary trees above, we also use unbalanced binary trees called L-Trees as in [16]. These are exclusively used to hash WOTS-T public keys. The ℓ_λ leaves of an L-Tree are the elements of a WOTS-T public key and

```
Input: Leaf index $i$, leaf $L_i$, authentication path $\mathsf{Auth}_i = (\mathsf{A}_0, \dots, \mathsf{A}_{h-1})$ for $L_i$.
Output: Root node ROOT of the tree that contains $L_i$.
Set $P_0 \leftarrow L_i$;
for $j \leftarrow 1$ up to $h$ do
    Set $i' = \lfloor i/2^j \rfloor$;
    $P_j = \begin{cases} \mathrm{H}_{k_{i',j}}((P_{j-1} || \mathsf{A}_{j-1}) \oplus r_{i',j}), & \text{if } \lfloor i/2^{j-1} \rfloor \equiv 0 \mod 2; \\ \mathrm{H}_{k_{i',j}}((\mathsf{A}_{j-1} || P_{j-1}) \oplus r_{i',j}), & \text{if } \lfloor i/2^{j-1} \rfloor \equiv 1 \mod 2; \end{cases}$
end
return $P_h$
```

Algorithm 1: Root Computation

the tree is constructed as described above but with one difference: A left node that has no right sibling is lifted to a higher level of the L-Tree until it becomes the right sibling of another node. Apart from this the computations work the same as for binary trees. The L-Trees have height $\lceil \log \ell_\lambda \rceil$.

4.1 XMSS-T

Given all of the above we can finally describe the algorithms of the XMSS-T construction. An XMSS-T keypair completely defines a hyper-tree of height h that consists of d layers of trees of height h/d. Each of these trees looks as follows. The leaves of a tree are $2^{h/d}$ L-Tree root nodes that each compress the public key of a WOTS-T key pair. Hence, a tree can be viewed as a key pair that can be used to sign $2^{h/d}$ messages. The hyper-tree is structured into d layers. On layer $d-1$ it has a single tree. On layer $d-2$ it has $2^{h/d}$ trees. The roots of these trees are signed using the WOTS-T key pairs of the tree on layer $d-1$. In general, layer i consists of $2^{(d-1-i)(h/d)}$ trees and the roots of these trees are signed using the WOTS-T key pairs of the trees on layer $i+1$. Finally, on layer 0 the WOTS-T key pairs are used to sign the message digests.

To improve readability, we only give a functional description of the algorithms of XMSS-T. To obtain a practical scheme, this has to be combined with the distributed signature generation method from [24] which in turn makes use of the BDS algorithm [14] for efficient tree traversal.

Key Generation Algorithm $((\mathsf{SK}, \mathsf{PK}) \longleftarrow \mathsf{kg}(1^n))$*:* The key generation algorithm first samples two secret values $(\mathsf{SK}_1, \mathsf{SK}_2) \in \{0,1\}^n \times \{0,1\}^n$. The value $\mathsf{SK}_1 = \mathcal{S}$ is the seed used for pseudorandom key generation in WOTS-T. The value SK_2 is used to generate pseudorandom values to randomize the message hash in sign. Also, the public seed $\textsc{Seed} \xleftarrow{\$} \{0,1\}^n$ is sampled as a uniform random value.

The remaining part of kg consists of generating the root node of the tree on layer $d-1$. Towards this end the WOTS-T key pairs for the single tree on layer $d-1$ are generated using SK_1 as $\mathcal{S}$. The ith leaf L_i of the tree is the root of an L-Tree that compresses pk_i. Finally, a binary hash tree is built using the constructed leaves and its root node becomes PK_1.

Besides the secret values and $\textsc{Seed}$, the secret key also contains the index i of the next WOTS-T key pair to use for message signing. The index takes h bits and is initialized with the all 0 bit string. The XMSS-T secret key is $\mathsf{SK} = (i = 0^h, \mathsf{SK}_1, \mathsf{SK}_2, \textsc{Seed})$, the public key is $\mathsf{PK} = (\mathsf{PK}_1, \textsc{Seed})$. kg returns the key pair $((\mathsf{SK}_1, \mathsf{SK}_2, \textsc{Seed}), (\mathsf{PK}_1, \textsc{Seed}))$.

Signature Algorithm $((\Sigma, \mathsf{SK}) \longleftarrow \mathsf{sign}(M, \mathsf{SK}))$*:* On input of a message $M \in \{0,1\}^*$ and secret key $\mathsf{SK} = (i, \mathsf{SK}_1, \mathsf{SK}_2, \textsc{Seed})$, sign computes a randomized message digest $D \in \{0,1\}^m$: First, a pseudorandom $R \in \{0,1\}^m$ is computed as $R \longleftarrow \mathcal{F}_m(\mathsf{SK}_2, M)$. Then, $D \longleftarrow \mathcal{H}(R, M)$ is computed as the randomized hash of M using R as randomness. Note that signing is deterministic, i.e., we need no real randomness as all required 'randomness' is pseudorandomly generated using PRF $\mathcal{F}_m$.

Given index i, the ith WOTS-T key pair on layer $d = 0$ is used to sign D. More specifically, this is the i_0th WOTS-T keypair in the i'_0th tree on layer 0, where i_0 is given by the last h/d bits of i and i'_0 by the remaining $(d-1)h/d$ bits of i. Next, the authentication path Auth_{i_0} for the i_0th leaf of the i'_0th tree is computed as well as the root of that tree. Now, for every layer $1 \leq \delta \leq d-1$ the same procedure is repeated with the difference that $i = i'_{\delta-1}$ and the root computed on layer $\delta - 1$ is signed. So, to sign the root from layer $\delta - 1$, the i_δth WOTS-T keypair in the i'_δth tree on layer δ is used, where i_δ is given by the last h/d bits of $i'_{\delta-1}$ and i'_δ by the remaining $(d-1)h/d$ bits of $i'_{\delta-1}$. Then the authentication path Auth_{i_δ} for the i_δth leaf of the i'_δth tree is computed as well as the root of that tree. The XMSS-T signature $\Sigma = (i, R, \sigma_{\mathrm{W},0}, \mathsf{Auth}_{i_0}, \ldots, \sigma_{\mathrm{W},d-1}, \mathsf{Auth}_{i_{d-1}})$ contains the used index i, randomness R and one WOTS-T signature – authentication path pair $\sigma_{\mathrm{W},j}, \mathsf{Auth}_{i_j}$, $j \in [d-1]$ per layer.

Finally, sign updates the secret key SK setting $i = i + 1$ and outputs the pair(Σ, SK).

Verification Algorithm $(b \longleftarrow \mathsf{vf}(M, \Sigma, \mathsf{PK}))$*:* On input of a message $M \in \{0,1\}^*$, a signature Σ, and a public key PK, the algorithm computes the message digest $D \longleftarrow \mathcal{H}(R, M)$ using the randomness R contained in the signature. Using i, the indices i_δ, i'_δ are computed for $0 \leq \delta \leq d-1$. The message digest D and the $\textsc{Seed}$ from PK are used to compute the first WOTS-T public key $\mathsf{pk}_{\mathrm{W},0} \longleftarrow \mathrm{WOTS}.\mathsf{vf}(D, \sigma_{\mathrm{W},0}, \mathbf{a}_{\mathrm{OTS}_0}, \textsc{Seed})$, where $\mathbf{a}_{\mathrm{OTS}_0}$ is the address of the i_0th WOTS-T keypair in the i'_0th tree on layer 0. An L-Tree is used to compute L_{i_0}, the leaf corresponding to $\mathsf{pk}_{\mathrm{W},0}$. Then, the root $\textsc{Root}_0$ of the respective tree is computed using Algorithm 1 with index i_0, leaf L_{i_0} and authentication path Auth_{i_0}.

Then, this procedure gets repeated for layers 1 to $d-1$ with the following two differences. First, on layer $1 \leq \delta \leq d-1$ the root of the previously processed tree $\textsc{Root}_{\delta-1}$ is used to compute the WOTS-T public key $\mathsf{pk}_{\mathrm{W},\delta}$. Second, the leaf computed from $\mathsf{pk}_{\mathrm{W},\delta}$ using an L-Tree is L_{i_δ}. The result of the final repetition on layer $d-1$ is a value $\textsc{Root}_{d-1}$ for the root node of the single tree on the top layer. This value is compared to the first element of the public key, i.e.,

$\mathsf{PK}_1 \stackrel{?}{=} \textsc{Root}_{d-1}$. If the comparison holds, vf returns true, otherwise it returns false.

5 Security

In the following we give a security reduction for XMSS-T. First, we review the required security definitions. Afterwards we give a security reduction for XMSS-T.

Existential Unforgeability under Adaptive Chosen Message Attacks. The standard security notion for digital signature schemes is existential unforgeability under adaptive chosen message attacks (EU-CMA) [18] which is defined using the following experiment. By $\textsc{Dss}(1^n)$ we denote a signature scheme with security parameter n.

Experiment $\mathsf{Exp}^{\mathsf{EU\text{-}CMA}}_{\textsc{Dss}(1^n)}(\mathcal{A})$
$(\mathsf{sk}, \mathsf{pk}) \longleftarrow \ \mathsf{kg}(1^n)$
$(\mathrm{Msg}^\star, \sigma^\star) \longleftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$
Let $\{(\mathrm{Msg}_i, \sigma_i)\}_1^q$ be the query-answer pairs of $\mathsf{sign}(\mathsf{sk}, \cdot)$.
Return 1 iff $\mathsf{vf}(\mathsf{pk}, \mathrm{Msg}^\star, \sigma^\star) = 1$ and $\mathrm{Msg}^\star \not\in \{\mathrm{Msg}_i\}_1^q$.

For the success probability of an adversary $\mathcal{A}$ in the above experiment we write

$$\mathrm{Succ}^{\text{EU-CMA}}_{\textsc{Dss}(1^n)}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\text{EU-CMA}}_{\textsc{Dss}(1^n)}(\mathcal{A}) = 1\right].$$

A signature scheme is called EU-CMA-secure if any PPT adversary has only negligible success probability:

Definition 2 (EU-CMA). *Let $n \in \mathbb{N}$, $\textsc{Dss}$ a digital signature scheme as defined above. We call $\textsc{Dss}$ EU-CMA-secure if for all $q, t = \mathrm{poly}(n)$ the maximum success probability $\mathrm{InSec}^{\text{EU-CMA}}(\textsc{Dss}(1^n); t, q)$ of all possibly probabilistic adversaries $\mathcal{A}$ running in time $\leq t$, making at most q queries to Sign in the above experiment, is negligible in n:*

$$\mathrm{InSec}^{\text{EU-CMA}}(\textsc{Dss}(1^n); t, q) \stackrel{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\text{EU-CMA}}_{\textsc{Dss}(1^n)}(\mathcal{A})\} = \mathrm{negl}(n).$$

To be precise, XMSS-T is a so-called key-evolving signature scheme which automatically updates the secret key after each signature. We capture this, assuming that the oracle $\mathsf{sign}(\mathsf{sk}, \cdot)$ in the above experiment replaces the secret key sk after each signature with the one returned by XMSS-T.sign and that it returns the empty string when $i \geq 2^h$, i.e. when the maximum number of signatures were done.

Pseudorandom Function Families. In the following we give the missing definition for the properties of (hash) function families that we use, namely pseudorandomness. In our definition we use the definition of (hash) function families from Section 2. In the definition of the success probability of an adversary against pseudorandomness ($\textsc{prf}$) the adversary gets black-box access to an oracle Box.

Box is either initialized with a function from $\mathcal{H}_n$ or a function from the set $\mathcal{G}(m, n)$ of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$. The goal of the adversary is to distinguish both cases:

$$\begin{aligned}\mathrm{Succ}^{\text{PRF}}_{\mathcal{H}_n}(\mathcal{A}) = \Big| &\mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{H}_n : \mathcal{A}^{\mathsf{Box}(\cdot)} = 1] \\ &- \mathsf{Pr}[\mathsf{Box} \xleftarrow{\$} \mathcal{G}(m,n) : \mathcal{A}^{\mathsf{Box}(\cdot)} = 1]\Big| . \end{aligned} \qquad (9)$$

Using this success probability, we define a pseudorandom function family the following way.

Definition 3 (PRF). *Let $\mathcal{H}_n$ be defined as above. We call $\mathcal{H}_n$ a pseudorandom function family, if it is efficient and for all $t = \mathrm{poly}(n)$ the maximum success probability* $\mathrm{InSec}^{\text{PRF}}(\mathcal{H}_n; t)$ *of all possibly probabilistic adversaries $\mathcal{A}$, running in time $\leq t$, is negligible in n:*

$$\mathrm{InSec}^{\text{PRF}}(\mathcal{H}_n; t) \overset{def}{=} \max_{\mathcal{A}}\{\mathrm{Succ}^{\text{PRF}}_{\mathcal{H}_n}(\mathcal{A})\} = \mathrm{negl}(n) .$$

5.1 Security reduction

We now proof the security of XMSS-T. We will base the security of the core scheme on the multi-function multi-target second-preimage resistance of F, H, the pseudorandomness of $\mathcal{F}_n$, the multi-target extended target collision resistance of $\mathcal{H}$ and a functional requirement on F defined below in the random oracle model. Please note that the random oracle model is only required to show that we can hand out the seed SEED used to generate the public function keys and bitmasks. Towards this end, we have to split the use of $\mathcal{F}_n$ into two parts. Assume two functions $\mathcal{F}_n^1$ and $\mathcal{F}_n^2$. We assume $\mathcal{F}_n^1$ is used in place of $\mathcal{F}_n$ for pseudorandom (secret) key generation and generation of the message hash randomness. For $\mathcal{F}_n^1$ we require standard model pseudorandomness. On the other hand, $\mathcal{F}_n^2$ is used to replace $\mathcal{F}_n$ when generating the hash keys $k_{i,j}$ and bitmasks $r_{i,j}$. In the proof, *only* $\mathcal{F}_n^2$ is modeled as random oracle (using the concatenation of key and input as input to the RO).

As mentioned above we need an additional requirement on F. Informally we require that every element in the image of F has at least two preimages, i.e.,

$$(\forall k \in \{0,1\}^n)(\forall y \in \mathrm{IMG}(\mathrm{F}_k))(\exists x, x' \in \{0,1\}^n) : x \neq x' \wedge \mathrm{F}_k(x) = f_k(x'). \quad (10)$$

Please note that this requirement meets the expectation for a random function. This additional requirement is needed to not having to use the one-wayness of F. If we had to use the one-wayness of F, we still would have to guess the messages an adversary sends to the oracle. The reason is that plugging a challenge image into a chain means not knowing any previous value of the chain. Hence, we could not answer a query where the signature contains such a previous value of a chain. This would imply a security loss of roughly h bits. Given the above property we can instead extract a second preimage if $\mathcal{A}$ inverts F with probability $1/2$, loosing only 1 bit in the security level.

Now we got everything needed for the security reduction. We proof the following theorem:

Theorem 2. *XMSS-T is existentially unforgeable under adaptive chosen message attacks with respect to the random oracle model if*

- F *and* H *are multi-function multi-target second-preimage resistant function families,*
- F *fulfills the requirement of Eqn. 10,*
- $\mathcal{F}_n^1, \mathcal{F}_m$ *are pseudorandom function families,*
- $\mathcal{F}_n^2$ *is modeled as a random oracle, and*
- $\mathcal{H}$ *is an multi -target extend target collision resistant hash function family.*

More specifically, the insecurity function $\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{XMSS-T}; \xi, 2^h\right)$ *describing the maximum success probability over all adversaries running in time* $\leq \xi$ *against the* EU-CMA *security of XMSS-T is bounded by*

$$\begin{aligned}&\mathrm{InSec}^{\text{EU-CMA}}\left(\textit{XMSS-T}; \xi\right)\\&\quad\leq \mathrm{InSec}^{\text{PRF}}\left(\mathcal{F}_n^1; \xi\right) + \mathrm{InSec}^{\text{PRF}}\left(\mathcal{F}_m; \xi\right)\\&\quad+ \max\{\mathrm{InSec}^{\text{M-eTCR}}\left(\mathcal{H}; \xi\right), 2\mathrm{InSec}^{\text{MM-SPR}}\left(\mathrm{F}; \xi\right), \mathrm{InSec}^{\text{MM-SPR}}\left(\mathrm{H}; \xi\right)\}\end{aligned}$$

The general idea of the proof follows that of previous hash-based schemes. There are a few mutually exclusive cases what could have happened if an adversary succeeded. First, the attacker could have broken the m-eTCR property of $\mathcal{H}$. This case is easily detected and can be handled in a straight-forward manner. Otherwise, the message digests have to differ. In this case the adversary has found a second preimage or a preimage for F or H with high probability. To extract a second preimage in this case, the reduction takes one MM-SPR challenge (M, K) per hash function call (H and F). Then, for this call the function is keyed with K and the bitmask is selected such that the input to the hash function is M. This means, if the input before the XOR with the bitmask is X, we use $X \oplus M$ as bitmask. Then the RO is programmed such that it generates this bitmask and key for this hash function call. This programming is done adaptively, only when the adversary queries the RO for a value. Now, any second preimage in the scheme will be a valid solution for MM-SPR of either H or F. In the proof below we omit indices for the MM-SPR challenge pairs to preserve readability.

Proof. Assume that there exists an adversary $\mathcal{A}$ running in time ξ that breaks the EU-CMA security of XMSS with probability $\epsilon_{\mathcal{A}}$. In the following we will prove that $\epsilon_{\mathcal{A}} \leq \mathrm{InSec}^{\text{EU-CMA}}\left(\text{XMSS-T}; \xi, 2^h\right)$. First, consider the following two games:

Game 1. This is the original game.

Game 2. This is the same as Game 1 but instead of using random elements from $\mathcal{F}_n^1$ and $\mathcal{F}_m$ (by sampling $\mathcal{S}$ and SK_2 from the key space), two truly random functions $\mathrm{G}_n : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^n$ and $\mathrm{G}_m : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^m$ are used.

The difference in the success probability of $\mathcal{A}$ playing one of these games must be bound by $\text{InSec}^{\text{PRF}}\left(\mathcal{F}_n^1;\xi\right)+\text{InSec}^{\text{PRF}}\left(\mathcal{F}_m;\xi\right)$, otherwise we could use $\mathcal{A}$ to distinguish $\mathcal{F}_n^1$ or $\mathcal{F}_m$ from a truly random function, breaking the pseudorandomness which would contradict the assumption. Hence, it suffices to analyze the success probability of $\mathcal{A}$ in Game 2. Towards this end, we construct an oracle machine $\mathcal{M}^{\mathcal{A}}$ that breaks either the multi-function multi-target second-preimage resistance of F or of H, or the extended target collision resistance of $\mathcal{H}$. $\mathcal{M}^{\mathcal{A}}$ takes q_1 challenge pairs $\{(K_i,M_i)\}_1^{q_1}$ for F and q_2 challenge pairs $\{(K_i',M_i')\}_1^{q_2}$ for H where q_1 (q_2) is the number of calls to F (H) for the XMSS-T key pair. Each of these challenge pairs gets associated with one specific call to F or H within the XMSS-T key pair.

$\mathcal{M}^{\mathcal{A}}$ first samples a random seed $\text{Seed} \xleftarrow{\$} \{0,1\}^n$. The XMSS-T public key PKbecomes $(\mathsf{PK}_1 = \text{H}_{K_j'}(M_j'), \text{Seed})$ for (K_j', M_j') – the pair associated with the call to H that computes the root. Now $\mathcal{A}$ is run on this PK. When $\mathcal{A}$ makes his ith query using some message Msg_i, $\mathcal{M}^{\mathcal{A}}$ first sends Msg_i to the M-ETCR challenger, receiving back a function key R_i. Then it computes the message digest as $D_i = \mathcal{H}(R_i, \text{Msg}_i)$ and sets the signature index to i. The next steps are the same for each tree involved in the signature. First, $\mathcal{M}^{\mathcal{A}}$ computes the chain indices b. The WOTS$^+$ signature is collected by selecting the challenge (K,M) for the b_jth call to F in the jth chain and computing the jth signature element as $\text{F}_K(M)$. Similarly, the authentication path for the WOTS$^+$ key pair is generated by figuring out the nodes that are required. Then, these nodes are calculated as $\text{H}_K(M)$ where (K,M) is the challenge pair associated with the call to H that computes this authentication path node. The same is done for the root node. Afterwards, the whole procedure is repeated for the parent tree, until the top tree is done. Then the XMSS-T signature Σ_i is sent back to $\mathcal{A}$.

If $\mathcal{A}$ queries the RO, the RO is adaptively programmed such that it outputs the right bitmasks and keys. W.l.o.g., we assume that $\mathcal{A}$ makes each query only once and stores the result. For any query $(X_1||X_2) \in \{0,1\}^n \times \{0,1\}^*$ where $X_1 \neq \text{Seed}$ or X_2 is no valid address of a bitmask or a key for a hash function call, the RO simply outputs a random value. Otherwise, assume for simplicity (again w.l.o.g.) that the adversary always queries the RO for the bitmask(s) and the key of a hash function call at once. Then the RO is programmed as follows: Assume (K,M) is the challenge pair for the associated hash call. If the address refers to the first hash call in a WOTS$^+$ function chain, the start value of that chain is generated as $X = \text{G}(\mathsf{SK}_1, \mathbf{a})$ where $\mathbf{a}$ is the address of that function chain. Next, the RO is programmed to return bitmask $r = X \oplus M$ and key K. If the address refers to a hash call in a WOTS$^+$ function chain that is not the first in that chain, let (K',M') be the challenge template associated with the previous hash call in that chain. Then the RO is programmed to return bitmask $r = \text{F}_{K'}(M') \oplus M$ and key K. Finally, if the address refers to a hash call to H, let (K_1,M_1), (K_2,M_2) be the challenge templates associated with the hash calls computing its two child nodes. Then the RO is programmed to return bitmask $r = (\text{H}_{K_1}(M_1)||\text{H}_{K_2}(M_2)) \oplus M$ and key K (Note that the challenge templates might be associated with calls to F if the address is associate with the computation of a node on level 1 in an L-tree. In this case H is replaced by F in the computation of r).

When $\mathcal{A}$ outputs a forgery (Msg, Σ) with $\Sigma = (i, R, \sigma_{\mathrm{W},0}, \mathsf{Auth}_{i_0}, \ldots, \sigma_{\mathrm{W},d-1}, \mathsf{Auth}_{i_{d-1}})$, $\mathcal{M}^{\mathcal{A}}$ runs the verification algorithm on (Msg, Σ) and (Σ_i, Msg$_i$). If the forgery is invalid, $\mathcal{M}^{\mathcal{A}}$ returns $\perp$. Otherwise, three mutually exclusive cases can occur. $\mathcal{M}^{\mathcal{A}}$ compares the values computed during the two verification runs in order of computation.

Case 1: If $D = \mathcal{H}(R, \mathrm{Msg}) = \mathcal{H}(R_i, \mathrm{Msg}_i) = D_i$, i.e., if the digests of the ith query is the same as that of the forgery, $\mathcal{M}^{\mathcal{A}}$ broke m-eTCR and returns (i, R, Msg). Hence, the probability that $\mathcal{A}$ outputs a case 1 forgery must be upper bounded by $\mathrm{InSec}^{\text{M-eTCR}}(\mathcal{H}; \xi)$ per assumption.

If the digests are different, the corresponding b_i are also different and hence, parts of the data computed by the two verification runs must also differ. Now, $\mathcal{M}^{\mathcal{A}}$ only compares the computed WOTS$^+$ public keys and the computed root values. By the pigeonhole principle, the signatures have to agree on one of these for the first time as they lead to the same root of the top tree.

Case 2: If the data generated verifying the two signatures first agrees on a WOTS$^+$ public key, the message digests or the root nodes signed with this WOTS$^+$ keypair where different. Hence, we got a WOTS$^+$ forgery. In this case, by the construction of the checksum there must be one chain j in this WOTS$^+$ keypair such that $b_j < (b_j)_i$, i.e. the jth signature value of the forgery belongs to an *earlier* hash call than the one of the answer to the ith query. As both chains end in the same public key value, they most collide at the output of some call to F. If this point is not the $(b_j)_i$th call it has to be a later one. In this case, the input to the colliding call to F computed from the forgery is a second preimage for the challenge template associated with that call to F and $\mathcal{M}^{\mathcal{A}}$ outputs it, breaking MM-SPR of F. Otherwise, the two chains collide on the output of the $(b_j)_i$th call to F, i.e., on $(\sigma_j)_i$, the jth value of the original signature. Let (K, M) be the challenge pair associated with the call to F that produced $(\sigma_j)_i$. According to Eqn. 10, $(\sigma_j)_i$ has at least two preimages under F_k. As $\mathcal{A}$ has no information about the preimage, the value X that can be computed from the forgery and that leads $f_K(X) = (\sigma_j)_i$ is unequal to M with at least probability 1/2. In that case, $\mathcal{M}^{\mathcal{A}}$ found a second preimage of M under F_K and outputs it. Otherwise it returns $\perp$. Consequently, the probability that $\mathcal{A}$ outputs a case 2 forgery must be upper bounded by $2\mathrm{InSec}^{\text{MM-SPR}}(\mathrm{F}; \xi)$ per assumption.

Case 3: If the data generated verifying the two signatures first agrees on a root node, the WOTS$^+$ public keys that are used to compute this root node have to differ. A third time by the pigeonhole principle, there must be one call to H between the WOTS$^+$ public key and the root node where the output for the forgery and the correct signature agree for the first time. As the input data depends on previously computed outputs of H (or F), it must differ. Hence, for challenge pair (K, M), the input to this call to H_K is a second preimage for M, that $\mathcal{M}^{\mathcal{A}}$ returns breaking MM-SPR of H. Hence, the probability that $\mathcal{A}$ outputs a case 3 forgery must be upper bounded by $\mathrm{InSec}^{\text{MM-SPR}}(\mathrm{H};\xi)$.

Combining the upper bounds from the three cases shows that the success probability $\epsilon_{\mathcal{A}}$ of $\mathcal{A}$ winning in Game 2 must be upper bounded by

$$\epsilon_{\mathcal{A}} \leq \max\{\mathrm{InSec}^{\text{M-eTCR}}(\mathcal{H};\xi), 2\mathrm{InSec}^{\text{MM-SPR}}(\mathrm{F};\xi), \mathrm{InSec}^{\text{MM-SPR}}(\mathrm{H};\xi)\}.$$

Combining this with the result that the difference in $\mathcal{A}$'s success probability between playing in Game 1 and playing in Game 2 must be upper bounded by $\mathrm{InSec}^{\text{PRF}}\left(\mathcal{F}_n^2;\xi\right)$, we get the claimed bound on the success probability of any adversary $\mathcal{A}$ running in time ξ:

$$\begin{aligned}\mathrm{Succ}^{\text{EU-CMA}}_{\text{XMSS-T}}(\mathcal{A}) \leq{}& \mathrm{InSec}^{\text{PRF}}\left(\mathcal{F}_n^1;\xi\right) + \mathrm{InSec}^{\text{PRF}}(\mathcal{F}_m;\xi)\\ &+ \max\{\mathrm{InSec}^{\text{M-eTCR}}(\mathcal{H};\xi), 2\mathrm{InSec}^{\text{MM-SPR}}(\mathrm{F};\xi), \mathrm{InSec}^{\text{MM-SPR}}(\mathrm{H};\xi)\}\end{aligned}$$

□

6 Implementation

In Section 4, we described XMSS-T, which builds on XMSS^{MT}, altering the functions that are used to construct WOTS chains and hash trees. We now examine the cost of this change in terms of computation time. In order to measure the cost of the additional bitmasks and keys that are required for each application of the functions F and H, we have implemented and benchmarked XMSS-T and XMSS^{MT}. We use the BDS tree traversal algorithm [14] to speed up the authentication path computation, making the scheme practical.

We examine the scheme for two separate parameter configurations, obtaining measurements for both a single-tree and a multi-tree set-up. For both settings, we use $w = 16$ and $n = 256$. For the first benchmark, we set $h = 20, d = 1$, and $m = 276$. We use the same subtree height for the second configuration, setting $h = 60, d = 3$, and $m = 316$ to construct three layers of subtrees with a height of twenty nodes each. We set $k = 2$ as the BDS parameter for both parameter sets, we rely on the SHA 256 function to construct F and H, and use ChaCha20 as the pseudorandom generator. These choices are in accordance with the parameters defined in the XMSS^{MT} Internet-Draft [23]. Following the security analysis in the last section and the lower bounds in Section 2, these parameters have a security level of more than 256 bits classical, and 128 bits quantum (assuming that each hash query requires more than 4 bit operations) for XMSS-T. For XMSS we get approximately 190 bits classical and 95 bits quantum for $h = 60$ (230 and 115 for $h = 20$). To carry out these benchmarks, we have used a single core of an Intel Core i7-4770K CPU, running at 3.5 GHz, although the implementation was not optimized specifically for this platform.

In the first setting, with $h = 20$, we measure an average signing time of 12 488 458 clock cycles per signature for XMSS, and 34 862 033 clock cycles for XMSS-T. For the multi-tree version ($h = 60$), the scheme takes 13 014 401 clock cycles per signature for XMSS, and 37 025 552 cycles for XMSS-T.

This difference is quite significant. However, this was to be expected as the running time of the scheme is largely dominated by applications of F and H – precisely the functions that are changed for XMSS-T. For plain XMSS with the aforementioned parameters, these functions merely consist of calls to SHA-256 with inputs of 256 and 512 bits, respectively. Each of these inputs fits within the internal block size of SHA-256 (512 bits). When considering the Merkle-Damgård construction [27] that defines the structure of SHA-256, this implies a single application of the internal compression function. When transforming F and H into keyed hash functions, the input length increases. To ensure that the key and the input are in separate blocks, the key is prefixed with 256 zero-bits. This results in inputs of 768 and 1024 bits, respectively, implying the need for two blocks, as well as two applications of the compression function. The straightforward calls to SHA-256 for F and H run in 1 072 and 1 924 cycles, while the keyed variants take 1 932 and 2 812 cycles, respectively.

A bigger factor weighing down F and H is the time needed to generate the keys and bitmasks pseudorandomly. Both these values require calls to the pseudorandom generator. For F, we require two output blocks of 256 bits each; H requires three. At an expense of 560 cycles per output block, generating randomness for the masks and keys carries a significant cost.

Altogether, the runtime is increased by a factor of 3 while the security level is increased by up to 66 bits classical and 33 bits quantum.

References

1. Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004. 4
2. Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. 3
3. Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. Quantum attacks on classical proof systems (the hardness of quantum rewinding). In *FOCS 2014*, pages 474–483. IEEE, October 2014. Preprint on IACR ePrint 2014/296. 9
4. Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. 3
5. Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997. 9, 13
6. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In Marc Fischlin and Elisabeth Oswald, editors, *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 368–397. Springer Berlin Heidelberg, 2015. 2, 3
7. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In DongHoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer Berlin Heidelberg, 2011. 7
8. Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *arXiv preprint quant-ph/9605034*, 1996. 8

9. Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002. 8
10. Gilles Brassard, Peter Hoyer, and Alain Tapp. Quantum algorithm for the collision problem. *arXiv preprint quant-ph/9705002*, 1997. 4
11. Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In *Automata, Languages and Programming*, pages 820–831. Springer, 1998. 8
12. Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the Winternitz one-time signature scheme. In A. Nitaj and D. Pointcheval, editors, *Africacrypt 2011*, volume 6737 of *LNCS*, pages 363–378. Springer Berlin / Heidelberg, 2011. 2, 16
13. Johannes 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 2011*, volume 7071 of *LNCS*, pages 117–129. Springer Berlin / Heidelberg, 2011. 2, 3, 16
14. Johannes Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, volume 5299 of *LNCS*, pages 63–78. Springer Berlin / Heidelberg, 2008. 20, 27
15. J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977. 10
16. Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography 2008*, volume 5299 of *LNCS*, pages 109–123. Springer, 2008. 2, 16, 19
17. Edward Eaton and Fang Song. Making existential-unforgeable signatures strongly unforgeable in the quantum random-oracle model. In *10th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2015, May 20-22, 2015, Brussels, Belgium*, pages 147–162, 2015. 12, 13
18. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. 22
19. Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996. 8
20. Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 41–59. Springer Berlin / Heidelberg, 2006. 2
21. Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 41–59. Springer Berlin / Heidelberg, 2006. 6
22. Andreas Hülsing. W-OTS+ shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and AboulElla Hassanien, editors, *Africacrypt 2013*, volume 7918 of *LNCS*, pages 173–188. Springer, 2013. 2, 17
23. Andreas Hülsing, D. Butin, S. Gazdag, and A. Mohaisen. Xmss: Extended hash-based signatures draft-irtf-cfrg-xmss-hash-based-signatures-01. Crypto Forum Research Group Internet-Draft, 2015. `https://tools.ietf.org/html/draft-irtf-cfrg-xmss-hash-based-signatures-01`. 27
24. Andreas Hülsing, Lea Rausch, and Johannes Buchmann. Optimal parameters for XMSS^{MT}. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl,

and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, volume 8128 of *LNCS*, pages 194–208. Springer Berlin Heidelberg, 2013. 2, 3, 16, 20

25. A Joffe et al. On a set of almost deterministic k-independent random variables. *The Annals of Probability*, 2(1):161–162, 1974. 10
26. Howard Karloff and Yishay Mansour. On construction of k-wise independent random variables. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 564–573. ACM, 1994. 10
27. Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University, 1979. 28
28. Ilya Mironov. Collision-resistant no more: Hash-and-sign paradigm revisited. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *LNCS*, pages 140–156. Springer Berlin / Heidelberg, 2006. 2
29. Dominique Unruh. Quantum position verification in the random oracle model. In *CRYPTO 2014*, volume 8617 of *LNCS*, pages 1–18. Springer, August 2014. Preprint on IACR ePrint 2014/118. 12
30. Mark Zhandry. How to construct quantum random functions. In *FOCS 2012*, pages 679–687. IEEE, 2012. Full version available at `http://eprint.iacr.org/2012/182`. 9, 33, 34
31. Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 758–775. Springer Berlin Heidelberg, 2012. 9, 10
32. Mark Zhandry. A note on the quantum collision and set equality problems. *Quantum Information and Computation*, 15(7&8), 2015. 4

A Classical generic security

Preimage resistance. For preimage resistance, analysis shows that the success probability of any classical $\mathcal{A}$ that makes q queries to its oracle is

$$\mathrm{Succ}^{\text{OW}}_{\mathcal{H}_n}(\mathcal{A}) = \left(\frac{q+1}{2^n}\right), \tag{11}$$

where the probability is taken over the internal coins of the oracle and the random choices of K and M. An attacker that makes no query but simply outputs a random domain element has success probability 2^{-n} of hitting the target Y. An attacker that makes one query can verify the first guess. If that one did not hit Y he can make another guess which he now can not verify anymore. Together this gives a success probability of $2/2^n$. Iterating this gives the above bound. Consequently, an attacker needs $\mathcal{O}(2^n)$ queries to reach a success probability of at least 0.5.

Single-function multi-target preimage resistance. For SM-OW a similar analysis shows a bound of

$$\mathrm{Succ}^{\text{SM-OW}}_{\mathcal{H}_n,p}(\mathcal{A}) = \left(\frac{(q+1)p}{2^n}\right), \tag{12}$$

The reason is that the success probability of a single guess is now $p/2^n$. Otherwise, the argument follows along the lines of the above argument. Consequently, the query complexity of a successful attack is $\mathcal{O}(2^n/p)$. Please note, we also can get this result using a reduction from OW. In this case, we replace a random Y_i by the given Y from the OW game. The reduction looses a factor $1/p$.

Multi-function multi-target preimage resistance. While the previous cases are more or less known results, for MM-OW we are not aware of any such results. The difference to SM-OW is that the adversary now basically plays p independent OW games at once. In contrast to the OW game $\mathcal{A}$ can not use a query he made to attack Y_i for any other Y_j for $j \neq i$. The reason is that different functions are associated to the different Y_i. So, in the classical case we get a query bound of

$$\mathrm{Succ}^{\text{MM-OW}}_{\mathcal{H}_n,p}(\mathcal{A}) = \left(\frac{q+1}{2^n}\right), \tag{13}$$

The reason is that a guess has success probability $1/2^n$. As one also has to guess (K_i, Y_i), every verification query can only check if a given M fulfills $Y_i = \mathrm{H}_{K_i}(M)$ for a single i. Viewed differently, each query has to fix K_i in advance and outputs the associated Y_i only with probability 2^{-n}. Consequently, we get the same query bound $\mathcal{O}(2^n)$ as for OW.

Second-preimage resistance. In the case of second-preimage resistance, the success probability of any $\mathcal{A}$ that makes q queries to its oracle is

$$\mathrm{Succ}^{\text{SPR}}_{\mathcal{H}_n}(\mathcal{A}) = \left(\frac{q+1}{2^n}\right), \tag{14}$$

where the probability is taken over the internal coins of the oracle and the random choices of K and M. The bound can easily be derived following the analysis for one-wayness. Consequently, an attacker needs $\mathcal{O}(2^n)$ queries to reach a success probability of at least 0.5.

Single-function multi-target second-preimage resistance. For SM-SPR a similar analysis shows a bound of

$$\mathrm{Succ}^{\text{SM-SPR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \left(\frac{(q+1)p}{2^n}\right). \tag{15}$$

Again, the analysis follows along the lines of the respective analysis for SM-OW. Consequently, the query complexity of a successful attack is $\mathcal{O}(2^n/p)$.

Multi-function multi-target preimage resistance. While the previous cases are more or less known results, for MM-SPR we are not aware of any such results. The difference to SM-SPR is that the adversary now basically plays p independent SPR games at once. As for MM-OW, in the classical case, we get a query bound of

$$\mathrm{Succ}^{\text{MM-SPR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \left(\frac{q+1}{2^n}\right). \tag{16}$$

Again, the analysis follows along the lines of the respective analysis for MM-OW. Consequently, the query complexity of a successful attack is $\mathcal{O}(2^n)$.

Extended target collision resistance. For eTCR, analysis shows that the success probability of any adversary $\mathcal{A}$ that makes no more than q queries to its oracle is

$$\mathrm{Succ}^{\text{eTCR}}_{\mathcal{H}_n}(\mathcal{A}) \leq \left(\frac{q+1}{2^n} + \frac{q}{2^k}\right), \tag{17}$$

where the probability is taken over the internal coins of the oracle and the random choice of K.

Consider an arbitrary adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ attacking eTCR. A_1 makes q_1 queries and outputs a message M. Afterwards $\mathcal{A}_2$ obtains K and makes q_2 queries, with $q_1 + q_2 = q$. Without loss of generality, we assume that $\mathcal{A}_1$ stores all his query results in the shared memory. When $\mathcal{A}_2$ receives K, we can distinguish two mutually exclusive cases:

Case 1: $\mathcal{A}_1$ already queried the oracle for $\mathrm{H}_K(M)$. To simplify analysis, we consider this a success for $\mathcal{A}$. As K is a random key and $\mathcal{A}_1$ made queries for no more than q_1 different keys, this case occurs with probability

$$\epsilon_1 \leq \frac{q_1}{2^k}.$$

Case 2: $\mathcal{A}_1$ did not query $\mathrm{H}_K(M)$ before. In this case, every query made by $\mathcal{A}_1$ and every query made by $\mathcal{A}_2$ has probability 2^{-n} to hit $\mathrm{H}_K(M)$ and hence to be a solution. If $\mathcal{A}_2$ does not find a solution using all query results, he can make another guess that has the same success probability as the queries before. Hence, the success probability in this case is exactly

$$\epsilon_2 = \frac{q_1 + q_2}{2^n}.$$

The sum of the two bounds $\epsilon_1 + \epsilon_2$ takes its maximum for $q_1 = q$. This gives the claimed bound. Note that the analysis for case 1 above is very rough and could be tightened (This is only a success if $\mathcal{A}_1$ already found a pseudo-collision for (K, M)). However, in all relevant cases we know $k \gtrsim n$ and hence tightening is of little use.

Multi-target-eTCR. Now, switching to M-eTCR the complexities for the two cases change as follows: In both cases a factor of q is lost. In Case 1 this is caused by the fact that there are now q keys returned (over the game) that might hit a previously queried one. In Case 2 this is caused by the fact that each query works for all q targets (as in the case of MM-SPR). This leads to the bound

$$\mathrm{Succ}^{\text{M-eTCR}}_{\mathcal{H}_n,p}(\mathcal{A}) = \frac{(q+1)p}{2^n} + \frac{qp}{2^k}.$$

B t-wise independence with arbitrary range

As noted earlier, most constructions of t-wise independent hash functions consider output space $\mathcal{Y}$ of size a prime or a prime power. We need one with $\mathcal{Y} = [N], N = 2^n - 1$. A natural approach is to pick a prime $M >> N$ and construct a t-wise independent family $\mathcal{H}_M : \mathcal{X} \to [M]$. Then $\mathcal{H} : x \mapsto \mathcal{H}_0(x) \bmod N$ will be good for our purpose, modulo a tiny error.

We show below that the "mod" construction above works. First of all, let F_k be the collection of all functions from $\mathcal{X}$ to $[k]$. We know that for any q-query quantum adversary $\mathcal{A}$

$$\Pr_{H \leftarrow \mathcal{H}_M}[\mathcal{A}^H(\cdot) = 1] = \Pr_{f \leftarrow F_M}[\mathcal{A}^f(\cdot) = 1].$$

Note that this still holds with the extra mod operation. Namely,

$$\Pr_{h \leftarrow \mathcal{H}_M}[\mathcal{A}^{H_N}(\cdot) = 1] = \Pr_{f \leftarrow F_M}[\mathcal{A}^{f_N}(\cdot) = 1],$$

where $g_N = \mathrm{mod}_N \circ g$ denote the composition of an arbitrary function g and mod_N, i.e., $g_N(x) = g(x) \bmod N$.

Therefore if we can show that

$$\left| \Pr_{f \leftarrow F_M}[\mathcal{A}^{f_N}(\cdot) = 1] - \Pr_{f \leftarrow F_N}[\mathcal{A}^f(\cdot) = 1] \right| = \mathrm{negl}(n) \quad (*),$$

then $H_N = \mathrm{mod}_N \circ H$ with $H \leftarrow \mathcal{H}_M$ will behave as a random function from X to $[N]$, except with negligible error.

We are left to prove (*). We need a tool by Zhandry [30].

Lemma 6. *[30, Theorem 7.3] There is a universal constant C such that the following holds. Let D_r be a family of distributions on functions from $\mathcal{X}$ to $\mathcal{Y}$ indexed by $r \in \mathbb{Z} \cup \infty$. Suppose that, for every integer k and every k pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, the function $p(r) = \Pr_{f \leftarrow D_r}[f(x_i) = y_i \forall i \in \{1, \dots, k\}]$ is a polynomial of degree at most k in $1/r$. Then any quantum algorithm making q quantum queries can only distinguish the distribution D_r from D_∞ with probability at most Cq^3/r.*

We define a family of distributions D_r on functions from $\mathcal{X}$ to $[N]$. $f \leftarrow D_r$ is generated as follows: first sample $f_r \leftarrow F_r$ a random function from $\mathcal{X}$ to $[r]$ and then set $f = \mathrm{mod}_N \circ f_r$. Observe that $f \leftarrow D_\infty$ is identical to $f \leftarrow F_N$, a truly random function from $\mathcal{X}$ to $[N]$. Now consider arbitrary k pairs $(x, y) \in \mathcal{X} \times [N]$, it is easy to verify that

$$\Pr_{f \leftarrow D_r}[\forall i \in [k], f(x_i) = y_i]$$

is a polynomial in $1/r$ with degree at most k.

Therefore, by Lemma 6, we can pick $M = \Omega(q^3 N)$ and obtain that D_M and D_∞ are indistinguishable by any q-query adversaries. Namely (*) holds.

C Reducing Avg-Search to OW, SM-OW, SPR, SM-SPR, and MM-SPR

Here we describe the reductions from Avg-Search to other security notions. They can be analyzed similarly to the case of MM-OW, as we discussed in Sect. 3.2, by which we prove the remaining of Proposition 1.

Given: $f \leftarrow D_\lambda : \{0,1\}^m \to \{0,1\}^n$, $\lambda = 1/2^n$.

1. Sample $y \leftarrow \{0,1\}^n$ uniformly at random.
2. Let $g : \{0,1\}^m \to \{0,1\}^n \setminus \{y\}$ be a random function. Construct $\tilde{H} : \{0,1\}^m \to \{0,1\}^n$ as follows: for any $x \in \{0,1\}^m$

$$x \mapsto \begin{cases} y & \text{if } f(x) = 1, \\ g(x) & \text{otherwise.} \end{cases}$$

Output: OW instance $(y, \tilde{H})$. Namely an adversary is given y and oracle access to $\tilde{H}$, and the goal is to find x^* such that $\tilde{H}(x^*) = y$.

Fig. 4. Reducing Avg-Search to OW.

Given: $f \leftarrow D_\lambda : \{0,1\}^m \to \{0,1\}^n$, $\lambda = p/2^n$.

1. For $i = 1, \ldots, p$, sample $y_i \leftarrow \{0,1\}^n$ independently and uniformly at random. Denote $S = \{y_i\}_{i=1}^p$.
2. Let $I : \{0,1\}^m \to [p]$ be a random function and $g : \{0,1\}^m \to \{0,1\}^n \backslash S$ be another random function. Construct $\tilde{H} : \{0,1\}^m \to \{0,1\}^n$ as follows: for any $x \in \{0,1\}^m$

$$x \mapsto \begin{cases} y_i, i = I(x) & \text{if } f(x) = 1, \\ g(x) & \text{otherwise.} \end{cases}$$

Output: SM-OW instance $(S, \tilde{H})$. Namely an adversary is given $\{y_i\}$ and oracle access to $\tilde{H}$, and the goal is to find x^* such that there exists $i \in [p]$ with $\tilde{H}(x^*) = y_i$.

Fig. 5. Reducing Avg-Search to SM-OW

More specifically, the quantum oracle $\sum_{x,z} \alpha_{x,z} |x\rangle|z\rangle \overset{\tilde{H}}{\mapsto} \sum_{x,z} \alpha_{x,z} |x, z + \tilde{H}(x)\rangle$ is implemented as follows:

$$\begin{aligned} & \sum \alpha_{x,z} |x, z\rangle \\ \mapsto & \sum \alpha_{x,z} |x, z\rangle |f(x)\rangle \quad \text{evaluate } f \\ \mapsto & \sum \alpha_{x,z} |x\rangle |z + (f(x) \cdot y_{I(x)} + \overline{f(x)} \cdot g(x))\rangle |f(x)\rangle \quad \text{set } \tilde{H}(x) \text{ conditioned on } f \\ \mapsto & \sum \alpha_{x,z} |x\rangle |z + (f(x) \cdot y_{I(x)} + \overline{f(x)} \cdot g(x))\rangle \quad \text{uncompute } f \\ =: & \sum \alpha_{x,z} |x\rangle |z + \tilde{H}(x)\rangle \end{aligned}$$

Given: $f \leftarrow D_\lambda : \{0,1\}^m \to \{0,1\}^n$, $\lambda = 1/2^n$.

1. Sample $\hat{x} \leftarrow \{0,1\}^m$ and $y \leftarrow \{0,1\}^n$ uniformly at random.
2. Let $g : \{0,1\}^m \to \{0,1\}^n \setminus \{y\}$ be a random function. Construct $\tilde{H} : \{0,1\}^m \to \{0,1\}^n$ as follows: for any $x \in \{0,1\}^m$

$$x \mapsto \begin{cases} y & \text{if } x = \hat{x} \\ y & \text{if } x \neq \hat{x} \& f(x) = 1, \\ g(x) & \text{otherwise.} \end{cases}$$

Output: SPR instance $(\hat{x}, \tilde{H})$. Namely an adversary is given y and oracle access to $\tilde{H}$, and the goal is to find x^* such that $x^* \neq \hat{x}$ and $\tilde{H}(x^*) = y$.

Fig. 6. Reducing Avg-Search to SPR.

Given: $f \leftarrow D_\lambda : \{0,1\}^m \to \{0,1\}^n$, $\lambda = \mathbf{p}/2^n$.

1. For $i = 1, \ldots, p$, sample $m_i \leftarrow \{0,1\}^m$ and $y_i \leftarrow \{0,1\}^n$ independently and uniformly at random. Denote $S = \{m_i\}$ and $T = \{y_i\}$.
2. Let $I : \{0,1\}^m \to [p]$ and $g : \{0,1\}^m \to \{0,1\}^n \backslash T$ be random functions. Construct $\tilde{H} : \{0,1\}^m \to \{0,1\}^n$ as follows: for any $x \in \{0,1\}^m$

$$x \mapsto \begin{cases} y_i & \text{if } x = m_i \\ y_i, i = I(x) & \text{if } x \notin S \wedge f(x) = 1, \\ g(x) & \text{otherwise.} \end{cases}$$

Output: SPR instance $(\hat{x}, \tilde{H})$. Namely an adversary is given y and oracle access to $\tilde{H}$, and the goal is to find x^* such that $x^* \neq \hat{x}$ and $\tilde{H}(x^*) = y$.

Fig. 7. Reducing Avg-Search to SM-SPR.

Given: $f \leftarrow D_\lambda : \{0,1\}^m \to \{0,1\}^n$, $\lambda = \mathbf{p}/2^n$.

1. For $i = 1, \ldots, p$, sample $m_i \leftarrow \{0,1\}^m$ and $y_i \leftarrow \{0,1\}^n$ independently and uniformly at random. Denote $S = \{m_i\}$ and $T = \{y_i\}$.
2. For $i = 1, \ldots, p$, let $g_i : \{0,1\}^m \to \{0,1\}^n \backslash T$ be random function. We construct $\tilde{H}_i : \{0,1\}^m \to \{0,1\}^n$ as follows: for any $x \in \{0,1\}^m$

$$x \mapsto \begin{cases} y_i & \text{if } x = m_i \\ y_i & \text{if } x \neq m_i \wedge f(i \| x) = 1 \\ g_i(x) & \text{otherwise.} \end{cases}$$

Output: MM-SPR instance $(S, \{\tilde{H}_i\}_{i=1}^p)$. Namely an adversary is given m_i and oracle access to $\tilde{H}_i$, and the goal is to find (i^*, x^*) such that $x^* \neq m_{i^*}$ and $\tilde{H}_{i^*}(x^*) = y_{i^*}$.

Fig. 8. Reducing Avg-Search to MM-SPR.

D Probability distribution of OW-challenge

The only difference to the MM-OW-setting is that a y_i might not have a preimage. This issue occurs in all proofs for *OW properties. The reason is that an Avg-Search instance is not guaranteed to contain a marked item. Especially, in the given MM-OW case, we are not guaranteed that for all $i \in [p]$ there exists an x such that $f(i\|x) = 1$. This is in contrast to the *OW definitions where the adversary is promised that all y_i have a preimage. However, this case only occurs with very low probability. Indeed, we could even show that these cases are not detectable by any polynomial time quantum adversary.

However, as we are concerned with proving lower bounds also for exponential time attacks, we take a different approach and just limit the error probability. For each i, the probability that y_i has no preimage is

$$\mathsf{Pr}\left[y_i \text{ has no preimage}\right] = \mathsf{Pr}\left[\forall x \in \{0,1\}^m : f(i\|x) = 0\right] = (1-2^{-n})^{2^m} < e^{-2^{m-n}}$$

where we use $e^x > 1 + x \Leftrightarrow e^{-2^{-n}} > 1 - 2^{-n}$. Therefore all y_i have a preimage with probability $> (1 - e^{-2^{m-n}})^p = \delta$. Note that δ is constant when the relation between m and n is fixed. Hence, the reduction produces a valid MM-OW instance with probability $> \delta$, and each query to $\{\tilde{H}_i\}$ can be efficiently answered by two queries to f.

Consider the reduction from Avg-Search to OW. Clearly the output (y, H) is distributed identically as we sample a random hash function H and a random element from the codomain $\mathcal{Y} = \{0,1\}^n$. We denote this distribution as $D_1 = (H, y) : H \leftarrow \mathcal{H}, y \leftarrow \mathcal{Y}$. Then we show that D_1 is close to another distribution $D_0 = (H, H(x)) : H \leftarrow \mathcal{H}, x \leftarrow \mathcal{X}$ which is the one in the definition for $\mathsf{Succ}^{\mathrm{OW}}$ in Eqn. 1.

$$\begin{aligned}
SD(D_0, D_1) &= \sum_{H,y} \frac{1}{2} \left| Pr_{H,x}(H, H(x) = y) - \Pr_{H,y}(H, y) \right| \\
&= \frac{1}{2} \sum_{H,y} \left| (\Pr_x(H(x) = y|H) - \Pr_y(y|H)) \cdot Pr_H(H) \right| \\
&= \frac{1}{2} \sum_{H,y} \frac{1}{|\mathcal{H}|} \left| \Pr_x(H(x) = y|H) - \frac{1}{|\mathcal{Y}|} \right| \\
&= \frac{1}{2} \sum_y \sum_H \frac{1}{|\mathcal{H}|} \left| \frac{|H^{-1}(y)|}{|\mathcal{X}|} - \frac{1}{|\mathcal{Y}|} \right| \\
&= \frac{1}{2|\mathcal{X}|} \sum_y \mathbb{E}_H(|Z_{y,H} - |\mathcal{X}|/|\mathcal{Y}||)
\end{aligned}$$

where $Z_{y,H} := |\{x \in \mathcal{X} : H(x) = y\}|$. Observe that $\mathbb{E}_H(Z_{y,H}) = |\mathcal{X}|/|\mathcal{Y}|$, so by Jensen's inequality we have that $\mathbb{E}_H(|Z_{y,H} - |\mathcal{X}|/|\mathcal{Y}||) \leq \sqrt{\mathbb{E}_H((Z_{y,H} - |\mathcal{X}|/|\mathcal{Y}|)^2)} = \sqrt{Var(Z_{y,H})}$. By a simple calculation we can see that $Var(Z_{y,H}) = \frac{|\mathcal{X}|(|\mathcal{Y}|-1)}{|\mathcal{Y}|^2}$.

Therefore

$$SD(D_0, D_1) \leq \frac{1}{2|\mathcal{X}|}|\mathcal{Y}|\sqrt{\frac{|\mathcal{X}|}{|\mathcal{Y}|}} = \frac{1}{2}\sqrt{\frac{|\mathcal{Y}|}{|\mathcal{X}|}}\,.$$

This is negligibly small whenever $|\mathcal{X}| >> |\mathcal{Y}|$ (e.g., $m \geq 2n$).