An Intermediate Secret-Guessing Attack on Hash-Based Signatures

Roland Booth[1], Yanhong Xu[1], Sabyasachi Karati[2], and Reihaneh Safavi-Naini[1]

[1] Department of Computer Science, University of Calgary, Canada
[2] Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata, India

Abstract. Digital signature schemes form the basis of trust in Internet communication. Shor (FOCS 1994) proposed quantum algorithms that can be used by a quantum computer to break the security of today's widely used digital signature schemes, and this has fuelled intensive research on the design and implementation of post-quantum digital signatures. Hash-based digital signatures base their security on one-way functions that in practice are instantiated by hash functions. Hash-based signatures are widely studied and are part of NIST's post-quantum standardization effort.

In this paper we present a multi-target attack that we call Intermediate Secret-Guessing attack on two hash-based signatures: XMSS$^{\text{MT}}$ (Draft SP 800-208 that was considered by NIST for standardization), and K2SN-MSS (AsiaCCS 2019). The attack allows an adversary to forge a signature on an arbitrary message. We describe the intuition behind the attack and give details of its application on the attacked schemes together with corresponding theoretical analysis. The attack implies that the effective security levels of XMSS (a special case of XMSS$^{\text{MT}}$), XMSS$^{\text{MT}}$, and K2SN-MSS are 10, 39 and 12 bits lower than their designed security levels given access to 2^{20}, 2^{60}, and 2^{20} signatures, respectively.

We implement the attack for each scheme, and give our results for reduced security parameters that validate our theoretical analysis. We also show that the attack can be avoided by modifying the application of a pseudorandom function for key generation. Our work shows the subtleties of replacing randomness with pseudo-randomness in the key generation of hash-based signatures, and the need for careful analysis of such designs.

Keywords: Post-quantum cryptography, hash-based signatures, multi-target attacks, XMSS$^{\text{MT}}$, K2SN-MSS, implementation

1 Introduction

Hash-Based Signatures. Digital signature schemes [21] are used to authenticate the origin of a message and form the basis of trust establishment for interactions on the Internet. The security of today's digital signature schemes relies on the hardness of mathematical problems that have efficient solutions if a quantum computer exists [41] and so post-quantum digital signatures must

use new computational problems that stay hard when a quantum computer of sufficient scale is built.

The idea of hash-based signatures (HBS) dates back to the pioneering work of Lamport [32]. A number of improvements have been proposed by Diffie, Merkle, and Winternitz [38,37]. All these schemes are one-time, and become insecure if two messages are signed. To use the signature scheme many-times (2^h), a direct approach is to generate 2^h one-time signature (OTS) schemes. However, this would require 2^h public key and secret key pairs, which will be highly impractical for large h. To construct a many-time signature scheme with short public key, Merkle [37] proposed what we know as Merkle signature scheme (MSS). The MSS uses 2^h instances of OTS, each with a public and secret key pair $(\mathsf{opk}_i, \mathsf{osk}_i)$, and builds a Merkle (binary) tree whose leaves are the hashes of the public keys of the OTS instances, and each internal node is computed as the hash of its two child nodes. The public key consists of the root of the Merkle tree, and the secret key contains the secret keys of all the 2^h OTS instances. The signature of a message M consists of an index i that specifies an OTS instance $(\mathsf{opk}_i, \mathsf{osk}_i)$, the one-time signature σ_{OTS} on M under the key opk_i, the key opk_i, and an authentication path AUTH_i that is used to verify the validity of opk_i. Since the pioneering work of Merkle [37,38], a large number of works, e.g. [13,11,12,19,17,10], have been proposed to improve various aspects of MSS.

In 2011, Buchmann et al. [10] proposed an extended MSS (XMSS) together with a forward-secure [2,6] variant. To reduce the size of the secret key that consists of the secret keys of 2^h OTS instances, a pseudorandom function (PRF) is used with an n-bit master seed to generate an n-bit OTS seed for each OTS instance, which is in turn used to generate the secret key of that instance. There have been a number of variants of XMSS [24,7,26,36,23,16,25,30,8] that provide higher security and efficiency. Prominently, Hülsing et al. [24] proposed a multi-tree variant of XMSS, known as XMSS^{MT}, which greatly improves the key generation time and can sign virtually unlimited number of messages. This variant was later selected by NIST as the standard algorithm for stateful HBS [16]. Karati and Safavi-Naini [30] proposed K2SN-MSS scheme, that extends KSN-OTS [28] to sign multiple messages, and proved its security in the same security model used by XMSS. The authors gave an implementation of the scheme that has comparable, and in some cases superior, performance compared to XMSS^{MT}. KSN-OTS and K2SN-MSS both use SWIFFT [34] as the hash function.

The security of modern digital signature schemes is proved against Existential Unforgeability against Chosen Message Attack (EU-CMA) where the attacker must generate a valid signature on some message of their choice, after *querying* a signing oracle to obtain q message-signature pairs. The security proof of HBS assumes truly random keys for OTS, and then shows that replacing truly random keys with pseudorandom keys, which are obtained by using a PRF with a random seed, will only reduce the security by a negligible amount. The actual generation of the pseudorandom keys, however, is considered an implementation detail and is not part of the security model and proof. The goal of this paper is to show that

an improper application of PRF can significantly reduce the designed security of the scheme.

Our Contributions and Techniques. We propose an Intermediate Secret-Guessing (ISG) attack, which is a multi-target attack, on two many-time signature schemes: an earlier version of XMSS$^{\mathrm{MT}}$ and K2SN-MSS. The attack on XMSS$^{\mathrm{MT}}$ applies to SP 800-208 draft [23,15] but not the final version [16].

The attack breaks the EU-CMA security of the two schemes by outputting a forgery on an arbitrary message. It exploits a weakness in the way the secret key of an OTS is generated from its associated seed, without using any other information unique to this instance such as its index. Concretely, the OTS secret keys in [23,15,30] are generated as follows. Let SEED be a master seed and $\mathcal{F}$ be a secure PRF. The key generation algorithm first generates seeds for OTS instances as $\mathrm{SEED}_{\mathrm{OTS},i} = \mathcal{F}(\mathrm{SEED}, i)$ for $i \in [0, 2^h - 1]$, and then generates OTS secret keys as $\mathsf{osk}_i = (\mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, 1), \ldots, \mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, \ell))$. Note that both SEED and $\mathrm{SEED}_{\mathrm{OTS},i}$ are n bits, with n being the designed security level of the scheme. For concreteness, we outline the attack below.

Attack in a nutshell. The attack has two phases. In the first *Query* Phase, the attacker collects $q \in [1, 2^h]$ signatures by querying the 2^h-time XMSS$^{\mathrm{MT}}$/K2SN-MSS as a signing oracle[3]. Next, in the *Secret-Guessing* Phase, the attacker repeatedly guesses the value of an n-bit OTS seed.

For each guess the attacker evaluates the PRF and detects if the guessed value is the seed used for generating one of the OTS signatures. If, for example, the guess is the seed of the i-th OTS signature, then the secret key of the i-th OTS instance is revealed. The seed, together with the i-th queried signature, enables a forgery on an arbitrary message of the attacker's choice. Since there are in total q OTS signatures and the probability that the guess will match one of the q targets is $q/(2^n)$, we expect to recover one of the OTS seeds after $2^n/q$ guesses.

How a guessed seed is matched for XMSS$^{\mathbf{MT}}$. A crucial detail missing from the aforementioned outline is how to match a guessed seed and the seed used in the i-th OTS signature. We note that evaluating the PRF on an OTS seed produces the OTS secret key, which consists of ℓ n-bit strings. In addition, a signature of the Winternitz OTS scheme (WOTS+) [22] that is employed in the XMSS$^{\mathrm{MT}}$ reveals some of these ℓ strings directly. A straightforward method is then to compare the j-th n-bit string generated from the guessed seed, with the one generated from the authentic one. If they are equal, then one considers the guess to be correct. This seed guess-verification strategy was also proposed by ETSI CyberSupport[4] in the public comment [1] on SP 800-208 draft. They further estimated that on average there are q/w WOTS+ signatures that will reveal the j-th n-bit string. Here, w is the Winternitz parameter. Therefore,

[3] Note that XMSS$^{\mathrm{MT}}$ is slightly more complicated since we have more than q OTS signatures from q queried signatures. See Section 3 for more detail.

[4] ETSI CyberSupport only outlined the idea of matching a guessed seed with the real seed but did not develop the idea into a full attack.

the attack can only have q/w targets to compare with, and will be expected to succeed after $(2^n w)/q$ guesses.

Our attack, that is independently discovered, starts with the same guessing strategy. However we observe that when the j-th strings derived from the two seeds match, with a probability around $1/2$, the seed may not be the real seed. This reduces the success chance of the attack by a factor of $1/2$. To improve the success probability of guessing the correct seed, we compare two strings computed from the guessed seed (instead of one) with those computed from the real one. We then show that with this tweak, that is when two strings match, the guess is correct with overwhelming probability. We further show that at least 91% WOTS+ signatures will reveal at least two strings of its secret key. This improves the expected number of guesses to $2^n/(0.91q)$ by increasing the number of targets to $0.91q$.

How a guessed seed is matched for K2SN-MSS. Verifying a guessed seed in K2SN-MSS is not as straightforward as in XMSS$^{\mathrm{MT}}$. This is because a KSN-OTS signature does not directly reveal the strings of the secret key. Rather, the signature is the sum of a subset of strings in the secret key that is determined by the message. We therefore go a step further and evaluate the PRF on a guessed seed, compute a KSN-OTS signature and then compare the computed signature with q extracted KSN-OTS signatures. If the q messages are distinct, then one computed KSN-OTS signature can be matched against only 1 target, rendering the success probability of the ISG attack almost the same as that of a brute-force attack. However, the success probability increases significantly if the same message is used for all queries.

Analysis, implementation and experiments. We analyze our attack theoretically. We derive the success probability and estimate the runtime of the attack when the number of queries and guesses are q and g, respectively, and then provide an estimation of the effective security levels[5] of the two attacked schemes. The analysis shows that the ISG attack implies the effective security levels of XMSS, XMSS$^{\mathrm{MT}}$, and K2SN-MSS are 10, 39, and 12 bits lower than their designed security levels given access to $q = 2^{20}$, $q = 2^{60}$, and $q = 2^{20}$ signatures, respectively.

To verify our analytical results, we implement the ISG attack on XMSS$^{\mathrm{MT}}$and K2SN-MSS, that have bit security of 256 and 512 bits, respectively. Even though the attack diminishes their security levels, the experiment is still infeasible in practice. We thus perform our experiments on reduced security parameter of 16 bits for both schemes that result in feasible computation, allowing us to verify our theoretical results.

Discussion. The security implications of bad randomness in cryptosystems is widely recognized. Numerous cryptographic algorithms that use the output of a PRF that expands a truly random seed have been broken by adversarial control of the random seed [43,33,42]. There have also been reported weaknesses in the design of PRF algorithms that have led to predictable outputs [20,40]. Our work

[5] Security level is calculated as $\log_2(\tau/\epsilon)$, where τ is the runtime and ϵ is the success probability of ISG attack.

indicates that bad application of a PRF for generating structured randomness can compromise the security of schemes with proven properties. Modeling and proving the full security of hash based signatures, including generating randomness using a PRF, is an interesting direction for future research.

Related work. Shor's algorithm [41] and the prospect of building quantum computers at scale have fueled research on cryptographic schemes with post-quantum security. HBS is an attractive approach to construct digital signature schemes with post-quantum security because OWFs can be instantiated with hash functions that have been intensively studied in recent years, and avoid using new and less studied hardness assumptions. The security of HBS schemes was initially reduced to the collision resistance of the hash functions, and later to the second preimage resistance using a Merkle-like tree structure that used random bitmasks for intermediate tree nodes. The schemes, however, become vulnerable to a new type of attack called a multi-target attack that has been more recently proposed by Hülsing et al. [26]. To protect against this attack, Hülsing et al. proposed a new HBS scheme, called XMSS-T (XMSS with tight security), in which each hash function call is keyed with a different key and uses a different bitmask.

Leighton-Micali signature (LMS) and its hierarchical system (HSS) for multiple messages [36] use WOTS [37,38,32] as the underlying OTS scheme. Both LMS and HSS were also selected as the standard algorithms for stateful HBS [16]. Katz [31] showed that earlier versions [35] of LMS and HSS can be subjected to a multi-target attack. To strengthen the security of the schemes, any hash computation within LMS and HSS prepends a different prefix to the value that will be hashed. These prefixes can be seen to have the same role as using different keys and bitmasks in [26].

All above schemes are stateful and require the signer to maintain a state and update it after each signature. The security of stateful schemes critically depends on the correctness of the state update. Bernstein et al. proposed the first practical stateless HBS scheme SPHINCS [7], which was later improved in followup works [25,3,8,5,4]. All versions submitted to the NIST post-quantum competition[6] employ the same addressing scheme as in [26] and are immune to multi-target attacks.

Stateless signatures have also been constructed based on symmetric key primitives. Picnic [14] is an example of such a scheme and uses efficient zero-knowledge protocols based on the "MPC-in-the-head" paradigm [27]. Picnic 1.0 was shown [18] to be vulnerable to multi-target attacks. The more recent version of Picnic, however, is secure against these attacks.

Organization. The rest of the paper is organized as follows. In Section 2 we briefly describe XMSS$^{\mathrm{MT}}$ and K2SN-MSS. Section 3 presents our ISG attack on XMSS$^{\mathrm{MT}}$ and shows its impact on the security level of XMSS$^{\mathrm{MT}}$ and Section 4 outlines the attack and its impact on K2SN-MSS. We then present our implementation results in Section 5. Finally, we propose countermeasures for the attack and conclude the paper in Section 6 and Section 7, respectively.

[6] https://csrc.nist.gov/projects/post-quantum-cryptography

2 Preliminaries

In this section, we briefly describe XMSS$^{\mathrm{MT}}$and K2SN-MSS.

2.1 Description of XMSS$^{\mathrm{MT}}$

XMSS$^{\mathrm{MT}}$ uses a variant of WOTS+ [22] as the underlying OTS scheme. Let w be the Winternitz parameter, n be the security parameter. The secret key is $\mathsf{osk} = (\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$ that contains ℓ strings of bit size n. The public key is $\mathsf{opk} = (\mathbf{y}_1, \ldots, \mathbf{y}_\ell)$ where each $\mathbf{y}_i$ is computed from $\mathbf{x}_i$ by applying a PRF $w-1$ times. To sign a message $M \in \{0,1\}^n$, one first computes its base-w representation $B_M = (b_1, \ldots, b_\ell)$ and output a signature $\sigma = (\mathbf{z}_1, \ldots, \mathbf{z}_\ell)$ where $\mathbf{z}_i$ is computed from $\mathbf{x}_i$ by applying the PRF b_i times. Specifically, if $b_i = 0$, then $\mathbf{z}_i = \mathbf{x}_i$. The signature is considered valid if one is able to obtain opk by applying the PRF $w-1-b_i$ times on $\mathbf{z}_i$ for all $i \in [1, \ell]$. More details can be found in Appendix A or [26,23].

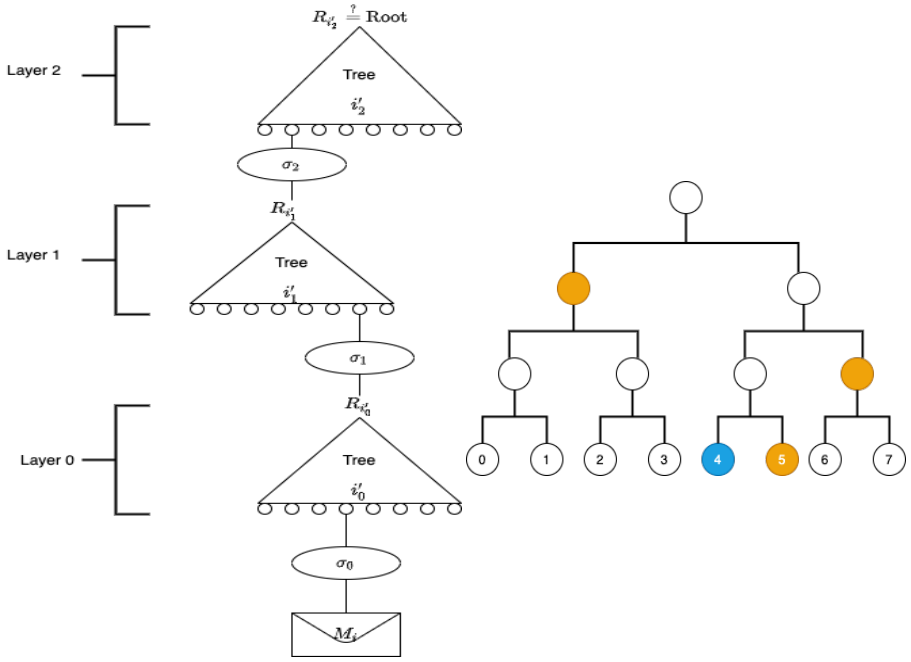

Fig. 1: A schematic representation of an XMSS$^{\mathrm{MT}}$ instance with $d = 3$ layers (Left) and the authentication path (yellow nodes) for the leaf 4 in a Merkle Tree of height 3 (Right).

An XMSS instance is a Merkle tree whose leaves are the WOTS+ instances. An XMSS$^{\mathrm{MT}}$ instance is essentially a tree of XMSS instances, called a *hyper tree*, where the XMSS trees on the lowest layer sign the actual messages, and the

XMSS trees on the upper layers sign the roots of the XMSS trees below them. In Figure 1, we give a representation of an XMSS$^{\mathrm{MT}}$ instance with 3 layers. Consider an XMSS$^{\mathrm{MT}}$ tree of the total height h that has d layers of XMSS trees of height $h' = h/d$. Let the root value of the sole XMSS tree on layer $d-1$ be ROOT. The public key consists of ROOT and some other public information, while the secret key contains $\mathrm{SEED}_{\mathrm{RAND}}$ and a master $\mathrm{SEED} \in \{0,1\}^n$. The former is used to compute a randomness R whenever we need to sign a message while the latter is to compute all the secret keys of the WOTS+ instances. Concretely, to compute the secret key of an OTS instance, the algorithm first generates seeds of all XMSS trees as $\mathrm{SEED}_{\mathrm{XMSS}} = \mathcal{F}(\mathrm{SEED}, s\|t)$, where s is the layer of the XMSS tree and t is the index of the tree within that layer. Next, it generates seeds for all OTS instances within a specific XMSS tree as $\mathrm{SEED}_{\mathrm{OTS},i} = \mathcal{F}(\mathrm{SEED}_{\mathrm{XMSS}}, i)$ for $i \in [0, 2^{h'}-1]$. Finally, the secret strings of the i-th OTS instance computed as $\mathbf{x}_{i,j} = \mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, j)$ for $j \in [1,\ell]$. We note that the input to the generation of $\mathbf{x}_{i,j}$ does not depend on the index i and this fact is exploited in our ISG attack.

An XMSS$^{\mathrm{MT}}$ signature of a message M is of the form

$$\Sigma = (i, R, \sigma_0, \mathrm{AUTH}_0, \ldots, \sigma_{d-1}, \mathrm{AUTH}_{d-1}),$$

where $i \in [0, 2^h - 1]$ is the index that specifies the i_j-th WOTS+ instance within the i'_j-th XMSS tree on the layer j for all $j \in [0, d-1]$ (see Figure 1), $R = \mathcal{F}(\mathrm{SEED}_{\mathrm{RAND}}, i)$, σ_0 is the one-time signature on the message digest $D = \mathcal{H}_{\mathrm{msg}}(R\|\mathrm{ROOT}\|i, M) \in \{0,1\}^n$ with $\mathcal{H}_{\mathrm{msg}}$ being a hash function, and σ_j is the one-time signature on the root value $R_{i'_j}$ for $j \in [1, d-1]$. To verify a signature Σ on M, one first computes D as described above, and proceeds as follows. It computes the i_0-th WOTS+ public key opk_{i_0} from the message-signature pair (D, σ_0). Then root $R_{i'_0}$ of the i'_0-th XMSS tree on the layer 0 is computed from opk_{i_0} and AUTH_0. This procedure is then repeated for layers 1 to $d-1$ until root $R_{i'_{d-1}}$ of the i'_{d-1}-th XMSS tree on the layer $d-1$ is obtained. The signature Σ is valid if $R_{i'_{d-1}} = \mathrm{ROOT}$.

2.2 Description of K2SN-MSS

We now give a very brief overview of the K2SN-MSS protocol [30]. It is a single-tree MSS where the underlying OTS is the KSN-OTS scheme [28]. The latter is an OTS scheme that employs an additive homomorphic hash function family SWIFFT [34]. The secret key is $\mathsf{osk} = (\mathbf{x}_1, \ldots, \mathbf{x}_t)$ that consists of t binary strings of size $\hat{n}\hat{m}$ while the public key is $\mathsf{opk} = (\mathbf{y}_1, \ldots, \mathbf{y}_t)$ where $\mathbf{y}_i = \mathrm{SWIFFT}_{\mathbf{k}}(\mathbf{x}_i)$ for some key $\mathbf{k}$ specifying the SWIFFT function. To sign a message $M \in \{0,1\}^m$, one first derives a subset B_M of $\{1, 2, \ldots, t\}$ from M and then computes the signature as $\sigma = \sum_{j \in B_M}(\mathbf{x}_j)$. Here $|B_M| = t/2$. The signature is considered valid if $\mathrm{SWIFFT}_{\mathbf{k}}(\sigma) = \sum_{j \in B_M}(\mathbf{y}_j) \bmod p$ and σ has small entries. More details can be found in [28].

In order to sign 2^h messages, K2SN-MSS builds a Merkle tree on top of 2^h KSN-OTS instances. The public key consists of the root of the tree and some other public information while secret key is a master $\mathrm{SEED} \in \{0,1\}^n$. The seed

is used to generate secret keys of those 2^h KSN-OTS instances as in XMSS$^{\mathrm{MT}}$. Specifically, it first generates seeds for all OTS instances as $\mathrm{SEED}_{\mathrm{OTS},i} = \mathcal{F}(\mathrm{SEED}, i)$ for all $i \in [0, 2^h - 1]$. Next, it computes the secret strings of the i-th OTS instance as $\mathbf{x}_{i,j} = \mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, j)$ for all $j \in [1, t]$. The fact that $\mathbf{x}_{i,j}$ does not depend on the index i is exploited in our ISG attack.

A K2SN-MSS signature of a message M is of the form $\Sigma = (i, \sigma_i, \mathsf{opk}_i, \mathrm{AUTH}_i)$, where i is the index of the used OTS instance, σ_i is the one-time signature on M under the public key opk_i of the i-th OTS instance, and AUTH_i is the authentication path. The signature is valid if σ_i is a valid signature on M and that opk_i is authenticated against AUTH_i. We observe that the signing algorithm here is not randomized as in XMSS$^{\mathrm{MT}}$, which makes forging a signature quite straightforward once we guess correctly the seed of a KSN-OTS instance.

3 ISG Attack on XMSS$^{\mathrm{MT}}$

We first give an overview of our ISG attack on XMSS$^{\mathrm{MT}}$. We assume that the attacker has access to $q \in [1, 2^h]$ signatures on the same message M_Q[7] and repeatedly guess WOTS+ seeds for at most $g \in [1, 2^n]$ times. The goal of the attack is to output a forgery Σ_F on a message M_F of the attacker's choice with the condition that $M_F \neq M_Q$. Note that in Step 1, $q' > q$ since there are WOTS+ instances on higher layers other than layer 0. In Step 2, we only store pairs that reveal at least two strings (out of ℓ) of their secret keys. In Step 3, we simply guess the seed as an n-bit representation of 0 up to $g - 1$.

1. From the q queried signatures, extract q' WOTS+ message-signature pairs.
2. Out of q' pairs, filter out those that contain less data (about their underlying secret keys) than some threshold. For the remaining pairs, store the data in some tables for efficient match in the next step.
3. For each guess $\mathrm{SEED}' \in [0, g - 1]$, derive a corresponding PRF output and compare with the stored data.
4. If a match is found for SEED', output a forgery Σ_F on M_F using SEED'.

In the following, we show in Section 3.1 how to verify the legitimacy of a guessed seed. Section 3.2 describes how to forge a signature if the guessed seed is legitimate and Section 3.3 gives the detailed description of the attack. Lastly, we analyze the runtime and success probability of the attack in Section 3.4.

3.1 Verifying a WOTS+ Seed Guess for XMSS$^{\mathrm{MT}}$

Consider an XMSS tree within an XMSS$^{\mathrm{MT}}$ hyper tree. Let $\mathrm{SEED}_{\mathrm{OTS},i}$ and osk_i be the seed and the secret key of the i-th WOTS+ instance in this XMSS tree, and let σ_i be a signature on a message M computed from osk_i. Given only the pair (M, σ_i), we want to determine if a guessed SEED' is equal to the legitimate seed

[7] This is not compulsory in our attack on XMSS$^{\mathrm{MT}}$, which randomizes the message before signing it.

$\text{SEED}_{\text{OTS},i}$. Recall that the secret key of the i-th WOTS+ instance is computed as

$$\mathsf{osk}_i = (\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,\ell}) = (\mathcal{F}(\text{SEED}_{\text{OTS},i}, 1), \ldots, \mathcal{F}(\text{SEED}_{\text{OTS},i}, \ell)),$$

To sign a message M, one first computes $B_M = (b_1, \ldots, b_\ell)$ and outputs the signature $\sigma_i = (\mathbf{z}_1, \ldots, \mathbf{z}_\ell)$ such that $\mathbf{z}_j = \mathbf{x}_{i,j}$ if $b_j = 0$ for some $j \in [1, \ell]$.

Verifying a guess against one WOTS+ signature. Given a signature $\sigma_i = (\mathbf{z}_1, \ldots, \mathbf{z}_\ell)$ and its corresponding message M, find two indices k_1 and k_2 such that b_{k_1} and b_{k_2} are zero. Then $\mathbf{z}_{k_1}$ and $\mathbf{z}_{k_2}$ are the k_1-th and the k_2-th elements of osk_i. If there are no such indices, then this test is inconclusive. Next, we compute $\mathsf{osk}' = (\mathbf{x}'_1, \ldots, \mathbf{x}'_\ell)$ from SEED' by evaluating PRF $\mathcal{F}$, and check if $\mathbf{x}'_{k_1} = \mathbf{z}_{k_1}$ and $\mathbf{x}'_{k_2} = \mathbf{z}_{k_2}$. If they are equal, we claim that $\text{SEED}' = \text{SEED}_{\text{OTS},i}$ with all but negligible probability. Fix a WOTS+ signature as above and let $\text{SEED}_{\text{OTS},i}$ be the real seed. Then

$$\begin{aligned} & \Pr[\text{SEED}' = \text{SEED}_{\text{OTS},i} | (\mathbf{x}'_{k_1} = \mathbf{x}_{k_1}) \wedge (\mathbf{x}'_{k_2} = \mathbf{x}_{k_2})] \\ = & \frac{\Pr[(\text{SEED}' = \text{SEED}_{\text{OTS},i}) \wedge (\mathbf{x}'_{k_1} = \mathbf{x}_{k_1}) \wedge (\mathbf{x}'_{k_2} = \mathbf{x}_{k_2})]}{\Pr[(\mathbf{x}'_{k_1} = \mathbf{x}_{k_1}) \wedge (\mathbf{x}'_{k_2} = \mathbf{x}_{k_2})]} \\ = & \frac{E_1}{E_1 + E_2}, \end{aligned}$$

where $E_1 = \Pr[(\text{SEED}' = \text{SEED}_{\text{OTS},i}) \wedge (\mathbf{x}'_{k_1} = \mathbf{x}_{k_1}) \wedge (\mathbf{x}'_{k_2} = \mathbf{x}_{k_2})] = \frac{1}{2^n}$, $E_2 = \Pr[(\text{SEED}' \neq \text{SEED}_{\text{OTS},i}) \wedge (\mathbf{x}'_{k_1} = \mathbf{x}_{k_1}) \wedge (\mathbf{x}'_{k_2} = \mathbf{x}_{k_2})] = (1 - \frac{1}{2^n})\frac{1}{2^{2n}}$, and the probability is taken over $\text{SEED}' \in \{0,1\}^n$. Note that conditioned on $\text{SEED}' \neq \text{SEED}_{\text{OTS},i}$, distributions of $\mathbf{x}'_{k_1}, \mathbf{x}'_{k_2}$ are indistinguishable from random distribution over $\{0,1\}^n$ due to the security of $\mathcal{F}$. One then sees that $E_1/(E_1 + E_2)$ is all but negligible.

The reason to compare two elements instead of just one is because a similar argument shows that $\Pr[\text{SEED}' = \text{SEED}_{\text{OTS},i} | \mathbf{x}'_{k_1} = \mathbf{x}_{k_1}] \approx \frac{1}{2}$. In other words, SEED' is not $\text{SEED}_{\text{OTS},i}$ with probability around $1/2$ if the k_1-th strings derived from the guessed seed and the real seed match only.

Verifying a guess against multiple WOTS+ signatures. Given q message-signature pairs $(M_0, \sigma_0), \ldots, (M_{q-1}, \sigma_{q-1})$ from q WOTS+ instances, the goal is to determine efficiently if a guess SEED' is the seed of one of these instances. For each pair (M_i, σ_i), we discard those whose signatures do not reveal at least two strings of their secret keys. For the remaining ones, we extract exactly two strings and then construct a tuple that contains these strings and the index i so we know which pair these strings are extracted from. The tuples will be sorted into tables that can be efficiently searched.

To this end, we build $\ell - 1$ tables $\text{T}_1, \ldots, \text{T}_{\ell-1}$. For $k_1 \in [1, \ell-1]$, T_{k_1} contains tuples of the form $(\mathbf{x}_{k_1}, k_2, \mathbf{x}_{k_2}, i)$ indexed by $\mathbf{x}_{k_1}$. Here $\mathbf{x}_{k_1}, \mathbf{x}_{k_2}$ are two strings of the secret key revealed in σ_i.

Next, compute $\mathsf{osk}' = (\mathbf{x}'_1, \ \ldots, \ \mathbf{x}'_\ell)$ from a guessed SEED' as before. Then for every $k_1 \in [1, \ell - 1]$, use a binary search algorithm to search T_{k_1}, checking whether it contains a tuple indexed by $\mathbf{x}'_{k_1}$. Suppose that, for some index k_1, we find a tuple $(\mathbf{x}'_{k_1}, \ k_2, \ \mathbf{x}_{k_2}, \ i)$ in T_{k_1}. Using the index k_2, we further compare $\mathbf{x}'_{k_2}$

with $\mathbf{x}_{k_2}$. If equal, we conclude that SEED′ is the underlying seed for computing σ_i, as shown in Section 3.1.

Note that to uniquely identify the location of a WOTS+ signature σ in an XMSS$^{\mathrm{MT}}$ tree, one must know the index i of the XMSS$^{\mathrm{MT}}$ signature that σ is extracted from and the hyper tree layer j that σ belongs to. To this end, we store tuples of the form $(\mathbf{x}_{k_1},\ k_2,\ \mathbf{x}_{k_2},\ i,\ j)$ instead.

3.2 Using a WOTS+ Seed to Forge a Signature

Let us now describe how to forge an XMSS$^{\mathrm{MT}}$ signature once we guess correctly the underlying seed of a WOTS+ instance. Let an XMSS$^{\mathrm{MT}}$ signature on message M_i be

$$\Sigma_i = (i, R, \sigma_0, \mathrm{AUTH}_0, \ldots, \sigma_{d-1}, \mathrm{AUTH}_{d-1}).$$

Suppose we have guessed the seed SEED′ of σ_j from Σ_i, and now want to forge an XMSS$^{\mathrm{MT}}$ signature on an arbitrary message M_F. We proceed as follows. Recall that the index i specifies that σ_j is from the i_j-th WOTS+ instance in the i'_j-th XMSS tree on layer j of the hyper tree.

Case 1: $j = 0$. It implies that SEED′ is the seed of the i_0-th WOTS+ instance which is used to sign $D_i = \mathcal{H}_{\mathrm{msg}}(R\|\mathrm{ROOT}\|i, M_i)$. To compute a forged signature Σ_F on M_F, we first compute a WOTS+ signature on the digest of M_F and then replace σ_0 in Σ_i with the new signature. (Recall that σ_0 is a signature on D_i.) Concretely, compute $\widehat{D}_i = \mathcal{H}_{\mathrm{msg}}(R\|\mathrm{ROOT}\|i, M_F)$ and a WOTS+ signature $\sigma_{0,F}$ of $\widehat{D}_i$ using the SEED′. Let $\Sigma_F = (i, R, \sigma_{0,F}, \mathrm{AUTH}_0, \ldots, \sigma_{d-1}, \mathrm{AUTH}_{d-1})$ be obtained by substituting σ_0 with $\sigma_{0,F}$. It is straightforward to verify the validity of Σ_F.

Case 2: $0 < j \leq d-1$. It implies that SEED′ is the seed of the i_j-th WOTS+ instance that is used to sign the root $R_{i'_{j-1}}$ of the i'_{j-1}-th XMSS tree on layer $j-1$. Recall that during the verification process of the pair (M_i, Σ_i), one computes WOTS+ public keys and XMSS roots $\mathsf{opk}_{i_0}, R_{i'_0}, \mathsf{opk}_{i_1}, R_{i'_1}, \ldots, \mathsf{opk}_{i_{d-1}}, R_{i'_{d-1}}$ sequentially and then compares $R_{i'_{d-1}}$ with ROOT. Since M_i is legitimately signed, the computed values are the real ones and in particular $R_{i'_{d-1}} = \mathrm{ROOT}$.

To compute a forged signature Σ_F, our strategy is to run the verification algorithm on (M_F, Σ_i) up to the point that the (fake) root value of i'_{j-1}-th XMSS tree is computed. Then we compute a WOTS+ signature on this (fake) root value using the SEED′, and replace σ_j in Σ_i with the new signature. Since the new WOTS+ signature is legitimately signed, one is able to compute the real WOTS+ public key opk_{i_j}. In fact, from this point on, all the computed WOTS+ public keys and XMSS roots are the real ones and thus the signature is considered valid. Concretely, we perform the following steps.

To begin with, compute $\widehat{D}_i = \mathcal{H}_{\mathrm{msg}}(R\|\mathrm{ROOT}\|i, M_F)$. Next, compute a fake i_0-th WOTS+ public key $\widehat{\mathsf{opk}}_{i_0}$ from the pair $(\widehat{D}_i, \sigma_0)$, and a fake root $\widehat{R}_{i'_0}$ from $\widehat{\mathsf{opk}}_{i_0}$ and AUTH_0 as in the verification process of XMSS$^{\mathrm{MT}}$ described in Section 2.1. This procedure is repeated for layers 1 to $j-1$ until a fake root

$\widehat{R}_{i'_{j-1}}$ is obtained. Then compute the signature $\sigma_{j,F}$ of the root $\widehat{R}_{i'_{j-1}}$ using SEED′. Finally, replace σ_j in Σ_i with $\sigma_{j,F}$, we obtain

$$\Sigma_F = (i, R, \sigma_0, \text{AUTH}_0, \ldots, \sigma_{j,F}, \text{AUTH}_j, \ldots, \sigma_{d-1}, \text{AUTH}_{d-1}).$$

3.3 ISG Attack on XMSS$^{\text{MT}}$

Putting everything together, we are ready to describe our ISG attack on XMSS$^{\text{MT}}$. The inputs of the attack are the number of signature queries $q \in [1, 2^h]$ and the number of seed guesses $g \in [1, 2^n]$, and the output is a forgery (M_F, Σ_F) if successful, or empty otherwise.

The attack initializes a set S_{MSS} to store the response from the signing oracle, and $\ell - 1$ tables $\text{T}_1, \ldots, \text{T}_{\ell-1}$ as described in Section 3.1 to store the data extracted from WOTS+ signatures. It operates in two phases.

In the Query Phase, the attacker queries the signing oracle with an arbitrarily chosen M_Q for q times. On the i-th query, an XMSS$^{\text{MT}}$ signature $\Sigma_i = (i, R, \sigma_0, \text{AUTH}_0, \ldots, \sigma_{d-1}, \text{AUTH}_{d-1})$ is obtained and then stored in S_{MSS}. From Σ_i, d WOTS+ message-signature pairs

$$(D_i, \sigma_0), (R_{i'_0}, \sigma_1), \ldots, (R_{i'_{d-2}}, \sigma_{d-1})$$

are computed. This can be done by running the XMSS$^{\text{MT}}$ verification algorithm. For each $\sigma_j = (\mathbf{z}_{j,1}, \ldots, \mathbf{z}_{j,\ell})$ with $j \in [0, d-1]$, let $\mathbf{z}_{j,k_1}, \mathbf{z}_{j,k_2}$ be two strings of secret key revealed and then insert $(\mathbf{z}_{j,k_1}, k_2, \mathbf{z}_{j,k_2}, i, j)$ to table T_{k_1}. If no two such strings exists, discard σ_j. Note that the WOTS+ signatures on layers 1 to $d-1$ may be repeated and thus are ignored once they appear again.

In the Secret-Guessing Phase, the attacker repeatedly guesses WOTS+ seeds until it succeeds, or runs out of the g guesses. Let SEED′ be the j-th guess. It first computes $\mathsf{osk}' = (\mathbf{x}'_1, \ldots, \mathbf{x}'_\ell)$ from SEED′ and then searches tables $\text{T}_1, \ldots, \text{T}_{\ell-1}$. If there exists a tuple $(\mathbf{z}_{k_1}, k_2, \mathbf{z}_{k_2}, i, j)$ such that $\mathbf{z}_{k_1} = \mathbf{x}'_{k_1}, \mathbf{z}_{k_2} = \mathbf{x}'_{k_2}$, we know that SEED′ is the underlying seed of the j-th WOTS+ instance from the i-th queried signature with all but negligible probability. Thus, a forged signature Σ_F on the message M_F of the attacker's choice can be computed (as long as $M_F \neq M_Q$) as described in Section 3.2. Otherwise, we move to the next guess. If no forgery is computed after g guesses, return $\perp$.

3.4 Analysis of ISG Attack on XMSS$^{\text{MT}}$

Number of targets. To calculate the success probability of our attack, it is crucial to find out the number of targets $\text{N}_{\text{TARGETS}}$. Recall that we have q' WOTS+ message-signature pairs. However, not all of them are valid targets to be matched against. Let P be the probability that a WOTS+ signature on a random message reveals at least two strings of its secret key. Then $\text{N}_{\text{TARGETS}} = q' \cdot P$. It is not hard to verify that $q' = \sum_{i=0}^{d-1} \lceil \frac{q}{2^{h' \cdot i}} \rceil$, where $h' = \frac{h}{d}$. Furthermore, P is lower bounded by $1 - (1 - \frac{1}{w})^{\ell_1} - \frac{\ell_1}{w}(1 - \frac{1}{w})^{\ell_1 - 1}$. (See Appendix B for details.) Given parameters $w = 16$, $n = 256$, $\ell_1 = 64$ for WOTS+, $P \geq 0.9153$.

Success probability of the ISG attack. The success probability of the ISG attack on XMSS$^{\mathrm{MT}}$ and inputs $q \in [1,\ 2^h]$, $g \in [1,\ 2^n]$ is:

$$\mathrm{Succ}^{\text{EU-CMA}}_{\mathrm{XMSS}^{\mathrm{MT}},\mathcal{A}(q,\ g)}(1^n) = 1 - \left(\frac{2^n - g}{2^n}\right)^{\mathrm{N}_{\mathrm{TARGETS}}}.$$

The attack outputs a forgery if and only if a guessed SEED$'$ equals one of the $\mathrm{N}_{\mathrm{TARGETS}}$ seeds, or equivalently, at least one of the $\mathrm{N}_{\mathrm{TARGETS}}$ seeds is in the set $[0, g-1]$. Note that these seeds are the outputs of a pseudorandom function whose output distribution over $\{0,1\}^n$ is indistinguishable from random. Therefore, the probability that none of these seeds is in the set $[0, g-1]$ can be approximated as $(\frac{2^n-g}{2^n})^{\mathrm{N}_{\mathrm{TARGETS}}}$, and the success probability is thus $1 - (\frac{2^n-g}{2^n})^{\mathrm{N}_{\mathrm{TARGETS}}}$.

How the runtime is measured. We measure the algorithmic time as the number of hash function evaluations, PRF evaluations, and the comparisons of $\mathcal{O}(n)$-bit strings. Denote these atomic operations as $\mathrm{C}_{\mathrm{HASH}}, \mathrm{C}_{\mathrm{PRF}}$ and $\mathrm{C}_{\mathrm{COMP}}$.

Runtime of the ISG attack. The runtime of the ISG attack on an instance of XMSS$^{\mathrm{MT}}$ with the inputs $q \in [1,\ 2^h]$ and $g \in [1,\ 2^n]$ is

$$\tau_{\mathrm{XMSS}^{\mathrm{MT}}}(q,\ g) \leq q \cdot \tau_{\mathrm{MSGDIGEST}} + (q' - q) \cdot \tau_{\mathrm{XMSSROOTAVG}} \tag{1}$$

$$+ \mathrm{N}_{\mathrm{TARGETS}} \cdot \log\left(\frac{\mathrm{N}_{\mathrm{TARGETS}}}{\ell - 1}\right) \cdot \mathrm{C}_{\mathrm{COMP}} \tag{2}$$

$$+ g \cdot \left(\ell \cdot \mathrm{C}_{\mathrm{PRF}} + (\ell - 1) \cdot \log\left(\frac{\mathrm{N}_{\mathrm{TARGETS}}}{\ell - 1}\right) \cdot \mathrm{C}_{\mathrm{COMP}}\right) \tag{3}$$

$$+ \tau_{\mathrm{COMPUTEFORGERYIS}}. \tag{4}$$

The time complexity of the attack is dominated by, (1) the time to compute the q' WOTS+ message-signature pairs, (2) the time to sort $\mathrm{N}_{\mathrm{TARGETS}}$ targets, (3) the time to search against the $\ell - 1$ tables, and (4) the time to compute the forgery. Details are in the Appendix B.

Effective security level of XMSS$^{\mathrm{MT}}$. Following [17], the bit security of a digital signature scheme (DSS) is estimated as $\log_2\left(\tau_{\mathrm{DSS}}(q,g)/\mathrm{Succ}^{\text{EU-CMA}}_{\mathrm{DSS},\mathcal{A}(q,\ g)}(1^n)\right)$. Using the above formulas, we evaluate the effective security levels of XMSS and XMSS$^{\mathrm{MT}}$ in Table 1. In the calculations, we assume $\mathrm{C}_{\mathrm{HASH}} = \mathrm{C}_{\mathrm{PRF}} = \mathrm{C}_{\mathrm{COMP}} = 1$.

Table 1: Effective security level of XMSS and XMSS$^{\mathrm{MT}}$ on concrete parameter sets.

Scheme	**Designed security level**	**Scheme parameters**	**Attack parameters**	**Effective security level**
XMSS	$n = 256$	$w = 16$, $\ell = 67$, $h = 20$	$q = 2^{20}$, $g = 2^{205}$	246.06
XMSS$^{\mathrm{MT}}$	$n = 256$	$w = 16$, $\ell = 67$, $h = 60$ $d = 12$	$q = 2^{60}$, $g = 2^{205}$	216.84

From the table, we see that the effective security levels of XMSS and XMSS$^{\mathrm{MT}}$ are 10 and 39 bits lower than their designed security levels. These results demon-

strate the significant effect of our attack. It is worth noting that our attack is more effective on XMSS$^{\mathrm{MT}}$due to significantly more target values.

4 ISG Attack on K2SN-MSS

We now give an outline of the ISG attack on K2SN-MSS. Assume that the attacker has access to $q \in [1, 2^h]$ signatures on the same message M_Q, and guesses KSN-OTS seeds for at most $g \in [1, 2^n]$ times. The aim of the attack is to output a forged signature Σ_F on an arbitrary message M_F where $M_F \neq M_Q$. It is crucial that all queries use the same message M_Q. This is because, unlike WOTS+, a KSN-OTS signature does not reveal strings of the secret key directly. Instead, a KSN-OTS signature reveals a sum of $t/2$ strings of its secret keys where the choice of the strings used for the sum depends on the message being signed.

1. From the q queried signatures, simply extract q KSN-OTS signatures.
2. Sort the q KSN-OTS signatures by interpreting the signatures as bit strings.
3. For each guess $\mathrm{SEED}' \in [0, g-1]$, evaluate the PRF on $t/2$ inputs determined by B_{M_Q} and sums the $t/2$ outputs. (This is equivalent to signing M_Q using the secret key derived from SEED'.) Compare the sum with the stored signatures using a binary search algorithm.
4. If a match is found for SEED', output a forgery Σ_F on M_F using SEED'.

We show how to verify the correctness of a guessed seed in Section 4.1, and how to forge a signature if we guess the seed correctly in Section 4.2. Section 4.3 describes our attack on K2SN-MSS and its runtime and success probability.

4.1 Verifying a KSN-OTS Seed Guess for K2SN-MSS

Let σ_i be a signature of the message M_Q derived from $\mathrm{SEED}_{\mathrm{OTS},i}$ for some i. To test if SEED' is $\mathrm{SEED}_{\mathrm{OTS},i}$, it is tempting to simply evaluate PRF on two inputs using SEED' and then compare with the extracted data as the attack on XMSS$^{\mathrm{MT}}$. As we observe, however, this is impossible since KSN-OTS signature does not reveal strings of its secret key directly. To solve this issue, we compute a KSN-OTS signature on M_Q as $\sigma' = \sum_{j \in B_{M_Q}} \mathcal{F}(\mathrm{SEED}', j)$ and then compare it with σ_i. If $\sigma' = \sigma_i$, we claim that $\mathrm{SEED}' = \mathrm{SEED}_{\mathrm{OTS},i}$ with overwhelming probability. Let

$$\begin{aligned}\Pr[\mathrm{SEED}' = \mathrm{SEED}_{\mathrm{OTS},i} | \sigma' = \sigma_i] &= \frac{\Pr[(\mathrm{SEED}' = \mathrm{SEED}_{\mathrm{OTS},i}) \wedge (\sigma' = \sigma_i)]}{\Pr[\sigma' = \sigma_i]} \\ &= \frac{E_1}{E_1 + E_2},\end{aligned}$$

where $E_1 = \Pr[(\sigma' = \sigma_i) \wedge (\mathrm{SEED}' = \mathrm{SEED}_{\mathrm{OTS},i})] = \frac{1}{2^n}$, $E_2 = \Pr[(\sigma' = \sigma_i) \wedge (\mathrm{SEED}' \neq \mathrm{SEED}_{\mathrm{OTS},i})] \leq (1 - \frac{1}{2^n})\frac{1}{2^{\hat{n}\hat{m}}}$, and the probability is taken over $\mathrm{SEED}' \in \{0,1\}^n$. Note that conditioned on $\mathrm{SEED}' \neq \mathrm{SEED}_{\mathrm{OTS},i}$, σ' is the addition of $t/2$ pseudorandom elements over $\{0,1\}^n$. Therefore, $\Pr[\sigma' = \sigma_i | \mathrm{SEED}' \neq \mathrm{SEED}_{\mathrm{OTS},i}] \leq \left(\max_j \binom{t/2}{j} \frac{1}{2^{t/2}}\right)^{\hat{n}\hat{m}} \leq \frac{1}{2^{\hat{n}\hat{m}}}$. Since $t = 262$, $\hat{n}\hat{m} = 2n$ in [30], the probability $E_1/(E_1 + E_2) = 1 - E_2/(E_1 + E_2)$ is all but negligible. This proves our claim.

4.2 Using a KSN-OTS seed to forge a signature

It is quite easy to forge a signature on M_F once we guess $\text{SEED}_{\text{OTS},i}$. Let the i-th queried signature be $\Sigma_i = (i, \sigma_i, \mathsf{opk}_i, \text{AUTH}_i)$. We simply compute the KSN-OTS signature σ_F on M_F using $\text{SEED}_{\text{OTS},i}$, and output $\Sigma_F = (i, \sigma_F, \mathsf{opk}_i, \text{AUTH}_i)$. It is straightforward to verify the validity of Σ_F on M_F.

4.3 ISG Attack on K2SN-MSS and Its Analysis

ISG attack on K2SN-MSS. The inputs of the attack are $q \in [1, 2^h]$ and $g \in [1, 2^n]$ as in Section 3.3, and the goal is to output a forgery (M_F, Σ_F). The attack initializes a set S_{MSS} to store the received signatures from the signing oracle, and a table T_{OTS} to store the extracted KSN-OTS signatures. It operates in two phases.

In the Query Phase, the attacker queries the signing oracle with an arbitrary message M_Q for q times. On the i-th query, a signature $\Sigma_i = (i, \sigma_i, \mathsf{opk}_i, \text{AUTH}_i)$ is received and stored in S_{MSS}. From Σ_i, we extract σ_i and insert (σ_i, i) in T_{OTS} that is indexed by σ_i. Note that for all i, σ_i is a signature on M_Q.

In the Secret-Guessing Phase, the attacker guesses the KSN-OTS seeds until it succeeds, or runs out of the g guesses. Let SEED' be the j-th guess. The attacker first computes $\sigma' = \sum_{j \in B_{M_Q}} \mathcal{F}(\text{SEED}', j)$, and then searches in T_{OTS}. If there is a tuple (σ_i, i) such that $\sigma_i = \sigma'$, then the attacker knows SEED' is the underlying seed with overwhelming probability as shown in Section 4.1. Thus, a forged signature Σ_F on M_F can be computed as described in Section 4.2. If it did not return any forgery after g guesses, abort.

Success probability of the ISG attack. Note that unlike XMSS^{MT}, the attack on K2SN-MSS has exactly q valid targets from the q queried signatures. Thus, following Section 3.4, the success probability of the ISG attack on K2SN-MSS using the inputs $q \in [1,\ 2^h]$, $g \in [1,\ 2^n]$ is:

$$\text{Succ}^{\text{EU-CMA}}_{\text{K2SN-MSS},\mathcal{A}(q,\ g)}(1^n) = 1 - \left(\frac{2^n - g}{2^n}\right)^q.$$

Runtime of the ISG attack. The runtime of the ISG attack on an instance of K2SN-MSS with the inputs $q \in [1,\ 2^h]$ and $g \in [1,\ 2^n]$ is given as:

$$\tau_{\text{K2SN}}(q,\ g) \leq\ q \cdot \log q \cdot \text{C}_{\text{COMP}} + g \cdot (\frac{t}{2} \cdot \text{C}_{\text{PRF}} + \log q \cdot \text{C}_{\text{COMP}}) + \frac{t}{2} \cdot \text{C}_{\text{PRF}}.$$

The runtime is dominated by (1) the time to sort the q KSN-OTS signatures, (2) the time to compute a KSN-OTS signature from a guessed seed and compare it with the sorted signatures, and (3) the time to compute a forgery if the attack succeeds. In the worst case, we have to run the guesses g times.

Effective security level of K2SN-MSS. We estimate the new security level of K2SN-MSS as $\log_2 \left(\tau_{\text{K2SN-MSS}}(q, g)/\text{Succ}^{\text{EU-CMA}}_{\text{K2SN-MSS},\mathcal{A}(q,\ g)}(1^n)\right)$. As in Section 3.4, we choose $\text{C}_{\text{HASH}} = \text{C}_{\text{PRF}} = \text{C}_{\text{COMP}} = 1$. For parameters $n = 512$, $\hat{n} = 64, \hat{m} = 16$, $t = 262$, $h = 20$, the effective security level of K2SN-MSS is 500.15 for $q = 2^{20}$ and $g = 2^{250}$. This is 12 bits lower than the designed security level 512.

5 Implementation and Experiments

We implemented the ISG attacks on XMSS^{MT} and K2SN-MSS utilizing implementations from [39][8] and [29] as the signing oracles. Our implementation can be found in [9]. In order to make the attack feasible and obtain a meaningful performance estimate, we reduce the search space of the attack by fixing all but the least significant $n' = 16$ bits of OTS seeds. No other changes are made to the attacked schemes.

Description of the experiments. Our two experiments are performed on **Skylake** Intel®Core^{TM}i7-6700 4-core CPU @ 3.40GHz running. The system has 8GB RAM and the timing experiments are performed on a single core. The OS is 64-bit Ubuntu-18.04 LTS and C codes are compiled by GCC version 7.5.0. During the experiments, the turbo boost and hyper-threading are turned off.

For each experiment, we performed 1000 trials on each of the possible input pairs (q, g) where $q \in \{1, 2^2, 2^4, 2^6, 2^8\}$ and $g \in \{1, 2^2, 2^4, \ldots, 2^{14}, 2^{16}\}$. From these trials we obtain an average runtime and success probability of the attack.

Results of the experiments. Figure 2 and Figure 3 show some of our experimental results. Figure 2 shows that the theoretical and actual success probabilities of the ISG attack on XMSS^{MT} and K2SN-MSS are well matched.

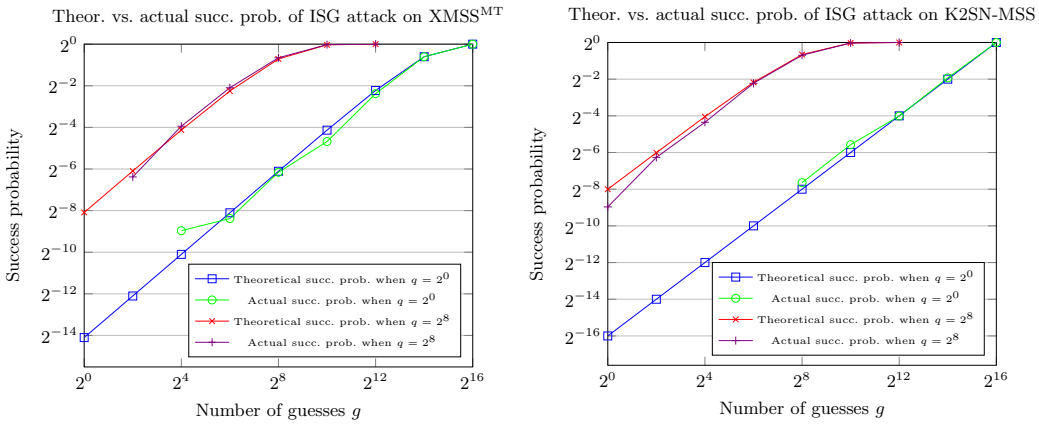

Fig. 2: Theoretical and actual success probability of ISG attack on XMSS^{MT}(Left) and K2SN-MSS (Right).

We note that the theoretical runtime is a count of the atomic operations while the actual runtime is in milliseconds. Figure 3 shows that both the theoretical and the actual runtimes increase at a similar rate as the number of guesses increase. Also note that the actual runtime begins to grow more slowly as g

[8] For XMSS^{MT}, we use the commit "fb7e3f8edce8d412a707f522d597ab3546863202" that is published on Apr 24, 2019 as the weakness was fixed in later commits.

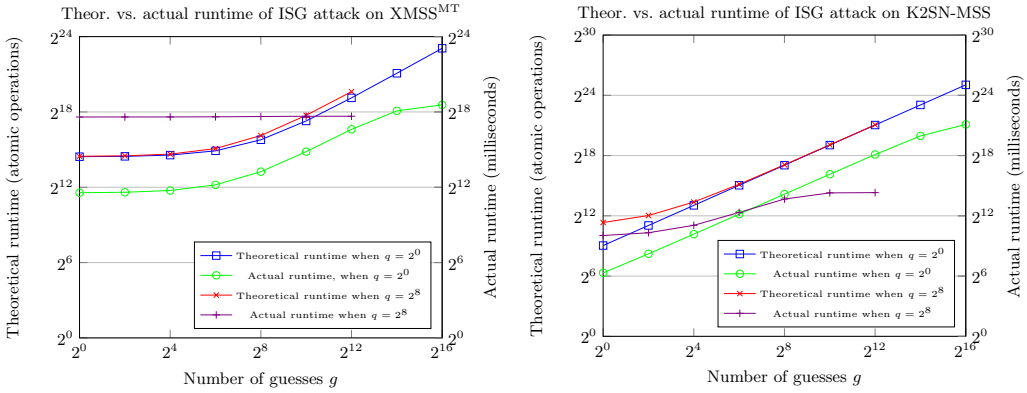

Fig. 3: Theoretical and actual runtime of ISG attack on XMSS$^{\mathrm{MT}}$(Left) and K2SN-MSS (Right).

gets close to its maximum value due to that our actual attack terminates before making all the g guesses.

6 Mitigations Against the ISG Attack

To protect against the ISG attack, the generation of the pseudorandom keys for OTSs must be revised. Recall that in XMSS$^{\mathrm{MT}}$, osk_i is generated as

$$\mathsf{osk}_i = (\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,\ell}) = (\mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, 1), \ldots, \mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, \ell)).$$

To prevent the attack, it suffices to generate osk_i by having the input to $\mathcal{F}$ dependent on the position of the OTS instance. Specifically, one computes the secret key of the i-th WOTS+ instance within an XMSS tree as

$$\mathsf{osk}_i = \big(\mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, s\|t\|i\|1), \ldots, \mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, s\|t\|i\|\ell)\big),$$

where s is the layer of the XMSS tree and t is the index of that tree within layer s.

The same strategy can also be used for K2SN-MSS. Specifically, one can generate osk_i in the following manner:

$$\mathsf{osk}_i = (\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,t}) = (\mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, i\|1), \ldots, \mathcal{F}(\mathrm{SEED}_{\mathrm{OTS},i}, i\|t)).$$

ETSI CyberSupport [1] also proposed a fix to prevent the attack by generating each secret string as

$$\mathbf{x}_{i,j} = \mathcal{F}(\mathrm{SEED}_{\mathrm{XMSS}}, \mathrm{ADDRES}),$$

where $\mathrm{SEED}_{\mathrm{XMSS}}$ is the seed of an XMSS tree and ADDRES is the unique address of $\mathbf{x}_{i,j}$ within the hyper tree.

7 Concluding Remarks

We proposed a multi-target attack called the ISG attack on XMSS$^{\mathrm{MT}}$ and K2SN-MSS, two hash-based signature schemes with provable security. The attacks, however, do not contradict the security proofs of the two schemes because the pseudorandom generation of secret keys is outside the security model and proofs of these schemes, and is considered an implementation detail of the algorithms. Thus our attack can be seen as an attack on the implementation. As discussed above, preventing the attack is straightforward. However, proving the soundness of using a secure PRF in an MSS structure remains a non-trivial open question. Our results show once again the importance of detailed specifications of cryptographic systems, and not leaving out important details that are needed in practice.

Acknowledgment

The works of Roland Booth, Yanhong Xu and Reihaneh Safavi-Naini were supported in part by Alberta Innovates Strategic Chair in Information Security Grant and Natural Sciences and Engineering Research Council of Canada Discovery Grant. Roland Booth was also supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number 551629 - 2020]. (Roland Booth a été financé par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéro de référence 551629 - 2020].)

A Description of WOTS+

We now describe the WOTS+ used in [26,23]. Let w be the Winternitz parameter, n be the security parameter, and $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a secure hash function. Define $\ell_1 = \left\lceil \frac{n}{\log_2(w)} \right\rceil$ and $\ell_2 = \left\lfloor \frac{\log_2(l_1(w-1))}{\log_2(w)} \right\rfloor + 1$, and $\ell = \ell_1 + \ell_2$. The secret key of WOTS+ is $\mathsf{osk} = (\mathbf{x}_1, \ldots, \mathbf{x}_\ell) \in (\{0,1\}^n)^\ell$ and the public key is $\mathsf{opk} = (\mathbf{y}_1, \ldots, \mathbf{y}_\ell)$ where $\mathbf{y}_i = c^{w-1,0}(\mathbf{x}_i, \mathbf{a}_{c_i}, \textsc{PubSeed})$. Here $\mathbf{a}_{c_i}$ is the address of the i-th chain within the OTS instance, PubSeed is a public seed, and $c^{i,j}(\mathbf{x}, \mathbf{a}_c, \textsc{PubSeed}) = F(k_{i,j}, c^{i-1,j}(\mathbf{x}, \mathbf{a}_c, \textsc{PubSeed}) \oplus r_{i,j})$ and $c^{0,j}(\mathbf{x}, \mathbf{a}_c, \textsc{PubSeed}) = \mathbf{x}$ for all $j \in \mathbb{Z}^+$, where $k_{i,j}, r_{i,j}$ are pseudorandomly computed. To sign a message M, one first computes a base-w representation $M = (M_1, \ldots, M_{\ell_1})$, then computes the checksum $C = \sum_{j=1}^{\ell_1}(w - 1 - M_j)$ and its base-w representation $C = (C_1, \ldots, C_{\ell_2})$. Set $B = (b_1, \ldots, b_\ell) = M \| C$. The signature of M is

$$\sigma = (\mathbf{z}_1, \ldots, \mathbf{z}_\ell) = (c^{b_1,0}(\mathbf{x}_1, \mathbf{a}_{c_1}, \textsc{PubSeed}), \ldots, c^{b_\ell,0}(\mathbf{x}_\ell, \mathbf{a}_{c_\ell}, \textsc{PubSeed})).$$

The signature $\sigma = (\mathbf{z}_1, \ldots, \mathbf{z}_\ell)$ is considered valid if for all $j \in [1, \ell]$: $\mathbf{y}_j = c^{w-1-b_j,b_j}(\mathbf{z}_j, \mathbf{a}_{c_j}, \textsc{PubSeed})$.

B Deferred Details of the ISG Attack on XMSS$^{\mathbf{MT}}$

Lower bound on P. Consider a WOTS+ signature σ on a random message M, let $B = (b_1, \ldots, b_\ell)$ be its base-w representation. The number of secret strings revealed in σ is the same as the number of b_i such that $b_i = 0$. Given a random message M, the probability that $b_i = 0$ for $i \in [1, \ell_1]$ is $\frac{1}{w}$. Unfortunately, there is no easy way to calculate the probability that $b_i = 0$ for $i \in [\ell_1 + 1, \ell]$. To this end, we provide a lower bound for P. Denote E as the number of b_i such that $b_i = 0$ for $i \in [1, \ell]$ and F as the number of b_i such that $b_i = 0$ for $i \in [1, \ell_1]$, then we obtain the following:

$$\begin{aligned} P = \Pr[E \geq 2] \geq \Pr[F \geq 2] &= 1 - \Pr[F = 0] - \Pr[F = 1] \\ &= 1 - (1 - \frac{1}{w})^{\ell_1} - \frac{\ell_1}{w}(1 - \frac{1}{w})^{\ell_1 - 1}. \end{aligned}$$

Runtime of ISG attack on XMSS$^{\mathbf{MT}}$. In Table 2, we give the details of the runtime of our ISG attack on XMSS$^{\mathrm{MT}}$.

Subroutine	**Runtime**
Compute WOTS+ chain node	$\tau_{\text{WOTS+ChainNode}} = 2 \cdot \text{C}_{\text{PRF}} + \text{C}_{\text{Hash}}$
Compute WOTS+ signature (on average)*	$\tau_{\text{WOTS+SignAvg}} \approx \ell \cdot \text{C}_{\text{PRF}} + \ell \cdot \frac{(w-1)}{2} \cdot \tau_{\text{WOTS+ChainNode}}$
Compute WOTS+ public key from a message-signature pair (on average)	$\tau_{\text{WOTS+PKAvg}} \approx \ell \cdot \frac{(w-1)}{2} \cdot \tau_{\text{WOTS+ChainNode}}$
Compute XMSS$^{\mathrm{MT}}$ tree node	$\tau_{\text{TreeNode}} = 3 \cdot \text{C}_{\text{PRF}} + \text{C}_{\text{Hash}}$
Compute XMSS$^{\mathrm{MT}}$message digest	$\tau_{\text{MsgDigest}} = \text{C}_{\text{PRF}} + \text{C}_{\text{Hash}}$
Compute XMSS tree root from a WOTS+ message-signature pair and authentication path	$\tau_{\text{XMSSRootAvg}} = \tau_{\text{WOTS+PKAvg}} + (\ell - 1 + h') \cdot \tau_{\text{TreeNode}}$
Compute XMSS$^{\mathrm{MT}}$ signature forgery from a WOTS+ seed (in the worst case)⋆	$\tau_{\text{ComputeForgeryIS}} = \tau_{\text{MsgDigest}} + (d - 1) \cdot \tau_{\text{XMSSRootAvg}} + \tau_{\text{WOTS+SignAvg}}$

*: Computation of a WOTS+ signature includes $\sum_{j=1}^{\ell} b_j$ computation of the chain node, where $(b_1, \ldots, b_\ell)$ is the base-w representation of a message M and its checksum C. On average, $\sum_{j=1}^{\ell} b_j = \ell \cdot \frac{w-1}{2}$. Similarly, computation of a WOTS+ public key includes $\sum_{j=1}^{\ell}(w - 1 - b_j)$ computation of the chain node, which on average is $\sum_{j=1}^{\ell} b_j = \ell \cdot \frac{w-1}{2}$.

⋆: In the worst case, the guessed seed is on layer $d-1$. This implies that computation of a forgery contains computation of the message digest, $d - 1$ XMSS tree roots and a WOTS+ signature.

Table 2: Runtimes of subroutines of ISG attack on XMSS$^{\mathrm{MT}}$.

References

1. Public comments on draft sp 800-208. https://csrc.nist.gov/CSRC/media/Publications/sp/800-208/draft/documents/sp800-208-draft-comments-received.pdf, accessed: 12/10/2020
2. Anderson, R.: Two remarks on public key cryptology. Unpublished. Available from http://www. cl. cam. ac. uk/users/rja14 (1997)
3. Aumasson, J.P., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., et al.: Sphincs (2020), round 3 Submisstion to NIST Post Quantum Project
4. Aumasson, J., Endignoux, G.: Clarifying the subset-resilience problem. IACR Cryptol. ePrint Arch. **2017**, 909 (2017)
5. Aumasson, J., Endignoux, G.: Improving stateless hash-based signatures. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 219–242. Springer (2018)
6. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 431–448. Springer (1999)
7. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: practical stateless hash-based signatures. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 368–397. Springer (2015)
8. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The sphincs$^+$ signature framework. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) CCS 2019. pp. 2129–2146. ACM (2019)
9. Booth, R., Karati, S.: Isg attack. https://github.com/rmbooth2/isg-attack (Dec 2020), accessed: 2021-16-16
10. Buchmann, J., Dahmen, E., Hülsing, A.: XMSS - A practical forward secure signature scheme based on minimal security assumptions. In: Yang, B. (ed.) PQCrypto 2011,. LNCS, vol. 7071, pp. 117–129. Springer (2011)
11. Buchmann, J., Dahmen, E., Klintsevich, E., Okeya, K., Vuillaume, C.: Merkle signatures with virtually unlimited signature capacity. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 31–45. Springer (2007)
12. Buchmann, J., Dahmen, E., Schneider, M.: Merkle tree traversal revisited. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 63–78. Springer (2008)
13. Buchmann, J., García, L.C.C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS - an improved merkle signature scheme. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 349–363. Springer (2006)
14. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Picnic: A family of post-quantum secure digital signature algorithms. https://microsoft.github.io/Picnic/
15. Cooper, D.A., Apon, D.C., Dang, Q.H., Davidson, M.S., Dworkin, M.J., Miller, C.A.: Recommendation for stateful hash-based signature schemes. NIST Special Publication (SP) 800-208 draft (2019), https://doi.org/10.6028/NIST.SP.800-208-draft
16. Cooper, D.A., Apon, D.C., Dang, Q.H., Davidson, M.S., Dworkin, M.J., Miller, C.A.: Recommendation for stateful hash-based signature schemes. NIST Special Publication (SP) 800-208 (2020), https://doi.org/10.6028/NIST.SP.800-208
17. Dahmen, E., Okeya, K., Takagi, T., Vuillaume, C.: Digital signatures out of second-preimage resistant hash functions. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 109–123. Springer (2008)

18. Dinur, I., Nadler, N.: Multi-target attacks on the picnic signature scheme and related protocols. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 699–727. Springer (2019)
19. Dods, C., Smart, N.P., Stam, M.: Hash based digital signature schemes. In: Smart, N.P. (ed.) IMACC 2005. LNCS, vol. 3796, pp. 96–115. Springer (2005)
20. Gjøsteen, K.: Comments on dual-ec-drbg/nist sp 800-90, draft december 2005 (04 2006)
21. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)
22. Hülsing, A.: W-OTS+ - shorter signatures for hash-based signature schemes. In: Youssef, A.M., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 173–188. Springer (2013)
23. Hülsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: Xmss: extended merkle signature scheme. Tech. rep., RFC 8391 (2018)
24. Hülsing, A., Rausch, L., Buchmann, J.: Optimal parameters for XMSS MT. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E.R., Xu, L. (eds.) CD-ARES 2013. LNCS, vol. 8128, pp. 194–208. Springer (2013)
25. Hülsing, A., Rijneveld, J., Schwabe, P.: Armed SPHINCS - computing a 41 KB signature in 16 KB of RAM. In: Cheng, C., Chung, K., Persiano, G., Yang, B. (eds.) PKC 2016. LNCS, vol. 9614, pp. 446–470. Springer (2016)
26. Hülsing, A., Rijneveld, J., Song, F.: Mitigating multi-target attacks in hash-based signatures. In: Cheng, C., Chung, K., Persiano, G., Yang, B. (eds.) PKC 2016. LNCS, vol. 9614, pp. 387–416. Springer (2016)
27. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge proofs from secure multiparty computation. SIAM J. Comput. **39**(3), 1121–1152 (2009)
28. Kalach, K., Safavi-Naini, R.: An efficient post-quantum one-time signature scheme. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 331–351. Springer (2015)
29. Karati, S.: K2sn-mss. `https://github.com/skarati/K2SN-MSS` (Jun 2019), accessed: 2020-01-21
30. Karati, S., Safavi-Naini, R.: K2SN-MSS: an efficient post-quantum signature. In: Galbraith, S.D., Russello, G., Susilo, W., Gollmann, D., Kirda, E., Liang, Z. (eds.) AsiaCCS 2019. pp. 501–514. ACM (2019)
31. Katz, J.: Analysis of a proposed hash-based signature standard. In: Chen, L., McGrew, D.A., Mitchell, C.J. (eds.) SSR 2016. LNCS, vol. 10074, pp. 261–273. Springer (2016)
32. Lamport, L.: Constructing digital signatures from a one way function. Tech. Rep. CSL-98 (October 1979), this paper was published by IEEE in the Proceedings of HICSS-43 in January, 2010.
33. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Ron was wrong, whit is right. IACR Cryptol. ePrint Arch. **2012**, 64 (2012)
34. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: A modest proposal for FFT hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer (2008)
35. McGrew, D., Curcio, M.: Hash-based signatures. Internet-Draft draft-mcgrew-hash-sigs-02 (2014), `https://datatracker.ietf.org/doc/html/draft-mcgrew-hash-sigs-02`

36. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-Micali hash-based signatures. Tech. rep., RFC 8554 (2019), https://doi.org/10.17487/RFC8554
37. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer (1989)
38. Merkle, R.C.: Secrecy, authentication, and public key systems. Ph.D. Thesis, Stanford University (1979)
39. Rijneveld, J., Hülsing, A., Cooper, D., Westerbaan, B.: xmss-reference. https://github.com/XMSS/xmss-reference/commit/fb7e3f8edce8d412a707f522d597ab3546863202 (Apr 2019)
40. Schoenmakers, B., Sidorenko, A.: Cryptanalysis of the dual elliptic curve pseudorandom generator. IACR Cryptol. ePrint Arch. **2006**, 190 (2006)
41. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: FOCS 1994. pp. 124–134. IEEE Computer Society (1994)
42. Strenzke, F.: An analysis of openssl's random number generator. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016. LNCS, vol. 9665, pp. 644–669. Springer (2016)
43. Yang, G., Duan, S., Wong, D.S., Tan, C.H., Wang, H.: Authenticated key exchange under bad randomness. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 113–126. Springer (2011)