Best known attacks against XMSS

Work In Progress

1 Introduction

This document attempts to analyze the best known attacks against XMSS (see [1]) in the Random Oracle Model (ROM).

2 Notations

- v: number of WOTS hash chains
- w: length of each hash chain (signer hashes between 0 and $w-1$ times for each hash chain)
- d: number of hashes per WOTS (difference between $v \cdot (w-1)$ and the target sum)
- $\mathcal{L}_d$: layer of the hypercube with distance d (cf. definition 3 of [2]), of cardinal ℓ_d.
- $p = 2^{31} - 2^{24} + 1$ is the KoalaBear prime.
- $\mathbb{F}$: the finite field of size p.
- $h_{i,o} : \mathbb{F}^i \to \mathbb{F}^o$ represent a family of hash functions (i (resp. o) field elements as input (resp. output)), modeled as random oracles.
- M: number of field elements representing the message to sign (before encoding).
- R: number of field elements contained in the randomness sampled by the signer, before computing the encoding.
- slot: Ethereum slot (every 12s currently). Each WOTS is associated to one slot. (typically, the first WOTS is for slot n, the next one $n+1$, etc). A slot requires 2 field elements to be stored.
- $tweak_{\text{pk}}$: The "tweak" part of the XMSS public key (the other part being a Merkle root). We use a different naming than [1], where this notion is captured by a "public parameter".
- T: number of field elements contained in $tweak_{\text{pk}}$
- $Enc : \mathbb{F}^{M+R+T+2} \to \mathcal{L}_d \cup \{\perp\}$ is the encoding function (taking as input: message, randomness, $tweak_{\text{pk}}$ and slot).
- D_{wots} (resp. D_{merkle}): number of field elements per hash digest in WOTS (resp. in the Merkle tree).
- L: lifetime of the XMSS public key ($L = 2^l$ where l is the depth of the Merkle tree)
- $Rate$: number of digests (of size D_{wots}) ingested per hashing step, when compressing the WOTS public key into a digest of size D_{merkle}.
- S: number of signers in the attack scenario
- q: number of queries of the adversary to the random oracle modeling the hash function

3 XMSS construction

We use a construction similar to [1], with the following differences:

- **Tweaks**: In [1], every hash (even within the same XMSS) has a different tweak.

  In our construction:

  - The tweak for WOTS depends on $\textit{tweak}_{\text{pk}}$ and the slot (i.e we remove the dependency in the position of the hash among $v \cdot (w-1)$ possibilities).
  - The tweak for the Merkle tree depends on $\textit{tweak}_{\text{pk}}$ (i.e we remove the dependency in the position within the tree).

- **Message encoding**. We don't specify a precise encoding, but instead use a general framework where only one value is relevant: $\mathbb{P}_{enc}$ (cf. next section).

3.1 Encoding

The encoding is built by hashing $M + R + T + 2$ field elements (message + randomness + $\textit{tweak}_{\text{pk}}$ + slot) into a sufficiently large number of field elements: E, and then by mapping them to $\mathcal{L}_d \cup \{\perp\}$ with $f : \mathbb{F}^E \to \mathcal{L}_d \cup \{\perp\}$:

$$Enc : \mathbb{F}^{M+R+T+2} \to \mathcal{L}_d \cup \{\perp\} := f \circ h_{M+R+T+2,E}$$

The way f is constructed doesn't matter (we can for instance extract 24 bits of each field element, as suggested here). The only relevant value is the probability of the most likely vertex of the hypercube:

$$\mathbb{P}_{enc} = \max_{l \in \mathcal{L}_d} \mathbb{P}[f(x) = l | x \overset{\$}{\leftarrow} \mathbb{F}^E]$$

3.2 Winternitz One Time Signature (WOTS)

In WOTS every digest is composed by D_{wots} field elements.

- secret key = $(sk^1_{wots}, \ldots, sk^v_{wots}) \in (D_{wots})^v$
- public key = $(pk^1_{wots}, \ldots, pk^v_{wots}) \in (D_{wots})^v$
- The hash chains are built by iterating:

$$x \in F^{D_{wots}} \to h_{D_{wots}+T+2, D_{wots}}(x | \textit{tweak}_{\text{pk}} | \text{slot}) \in F^{D_{wots}}$$

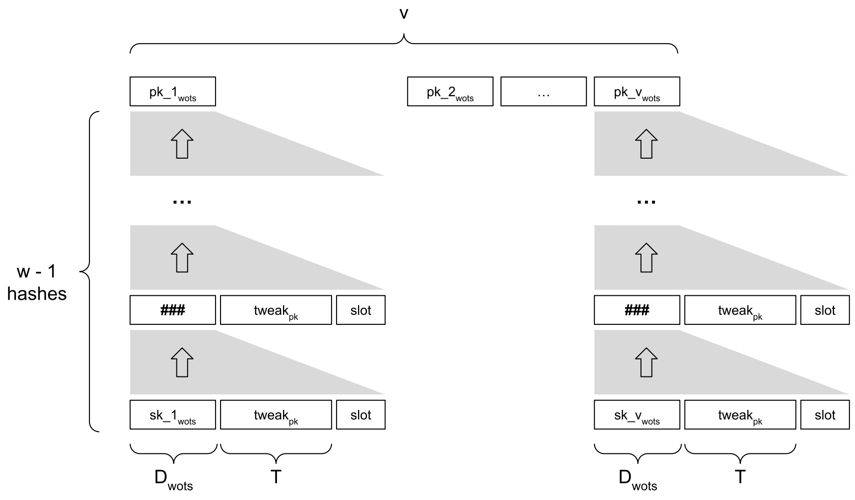
3.3 Compressing the WOTS public key

Let's call $Rate$ the number of digests (of size D_{wots}) ingested per hashing step. (capacity = D_{merkle} field elements).

To compress the WOTS public key, we compute the following:

- $c_1 = h_{T+Rate \cdot D_{wots}, D_{merkle}}(\textit{tweak}_{\text{pk}}, pk^1_{wots}, \ldots, pk^R_{wots} ate) \in \mathbb{F}^{D_{merkle}}$
- $c_2 = h_{D_{merkle}+Rate \cdot D_{wots}, D_{merkle}}(c_1, pk^{Rate+1}_{wots}, \ldots, pk^{2 \cdot Rate}_{wots}) \in \mathbb{F}^{D_{merkle}}$
- $\ldots$
- $c_{v/Rate-1} = \ldots$
- WotsPKCompr = $h_{D_{merkle}+Rate \cdot D_{wots}, D_{merkle}}(c_{v/Rate-1}, \ldots) \in \mathbb{F}^{D_{merkle}}$

Figure 1: WOTS visualized



3.4 Merkle tree

The lifetime of the XMSS is part of the public key (meaning: the verifier will only accept Merkle authentication paths with the correct depth). The Merkle tree has height l, so the XMSS has lifetime $L = 2^l$. Each digest in the Merkle tree contains D_{merkle} field elements.

Each step of the Merkle tree is built as follows:

$$Output = h_{T+2 \cdot D_{merkle}, D_{merkle}}(tweak_{\mathrm{pk}}, left, right)$$

4 Attacking XMSS

4.1 Scenario

The scenario of the attack is the following:

- There are S signers: the S XMSS public keys are sent to the attacker. The public key are valid for slots $[a : a + L]$ (L: lifetime).
- We repeat the following L times, for each slot:
  - The attacker sends a chosen message.
  - Every signer responds with a valid signature, at the corresponding slot.
- Overall, the attack will access $S \cdot L$ valid signatures.
- In the end, the attacker outputs a signer index, a slot, a message and a signature. The attack succeeds if the signature is well-formed, and if the message is different than the one originally handled by the signer at the corresponding slot.

Our goal is to conjecture an upper bound on the probability of success of such an attack, given how many times the attacker calls the oracle representing the hash function.

4.2 Reducing the attack to one signer

Let's denote the following events:

- SA : Successful Attack

- CAT : Collision Among Tweaks (2 signers share the same $\mathit{tweak}_{\mathrm{pk}}$).

$$\mathbb{P}(SA) = \mathbb{P}(SA|CAT) \cdot \mathbb{P}(CAT) + \mathbb{P}(SA|\overline{CAT}) \cdot \mathbb{P}(\overline{CAT})$$
$$\leq \mathbb{P}(CAT) + \mathbb{P}(SA|\overline{CAT}) \leq S^2/p^T + \mathbb{P}(SA|\overline{CAT})$$

We now focus on $\mathbb{P}(SA|\overline{CAT})$: the probability of success of the attack in the event of no collision among the tweaks:

Definition: We consider a message m and an associated signature, containing randomness r. Let's call $enc := Enc(m, r, \mathit{tweak}_{\mathrm{pk}}, slot) \in \mathcal{L}_d$ the corresponding encoding. If $\mathbb{P}[f(x) = enc | x \xleftarrow{\$} \mathbb{F}^E] = P_{max}$, we say the signature has the *Peak-collision* property. Intuitively, such signatures are the easiest ones to attack.

Conjecture 1. *As long as there are no collisions among the signer tweaks, the probability of success of the attack is bounded by the probability of success of the best attack when there is only 1 signer ($S = 1$), assuming at least one signature emitted by this unique signer has the Peak-collision property.*

As a result, we now assume there is only one signer ($S = 1$), and one of the signature has the **Peak-collision** property.

4.3 No preprocessing advantage

We denote by q the total number of queries of the adversary to the random oracle modeling the hash function.

Conjecture 2. *Assuming the randomness space is large enough: $p^R \geq q$, the adversary gains no advantage in choosing the messages. One of best strategy is to chose them at random.*

4.4 Attack strategies

We describe 4 high level attack strategies:

- **Attack the encoding**: finding a new pair (message, randomness) whose encoding collides with a previously signed message.
- **Attack the WOTS hash chains**: Finding at least one preimage of any of the $d + v$ digests present in a WOTS signature (it can be the same (secret) preimage used by the signer, or a new one).
- **Attack the WOTS public key compression**: Constructing a WOTS, with a new public key, and for which the compression (digest of size D_{merkle}) collides with an already existing one.
- **Attack the Merkle tree**: Crafting a valid Merkle authentication path for a new XMSS key (controlled by the attacker).

Conjecture 3. *The best attack involves at least one the high level strategies above.*

We need to find an upper bound on the probability of success of an attack against each the 4 scenarios.

4.4.1 Attack the encoding

Conjecture 4. *One of the best attack is to loop among all the possible triplets of (message, randomness, slot), excluding those already signed, and to compute their encoding, hoping to collide with a previously signed message.*

At each iteration of this loop, the attack succeeds with probability $\leq \mathbb{P}_{enc}$.

Overall, with q iterations (so q queries to the oracle modeling the hash function), the probability of success of this attack is:

$$\mathbb{P}_{enc}^{attack}(q) \leq q \cdot \mathbb{P}_{enc}$$

4.4.2 Attack the WOTS hash chains

Let $n := p^{D_{wots}}$ be the size of our WOTS digests.

Let $k := d + v$ the maximum number of target digests, per WOTS (we assume they are distinct wlog).

Conjecture 5. *One of the best attack is to first fix an arbitrary slot, and then to loop among the possible preimages (n possibilities), and for each hash obtained, comparing it against all the target digests (hoping for collision).*

The attack succeeds in the i^{th} iteration of this loop if:

- it finds a preimage known by the signer, but kept secret. This happens with probability $\frac{k}{n-i}$
- it finds an original preimage. This happens with probability $\frac{k}{n}$

Overall, with q iterations (so q queries to the oracle modeling the hash function), the probability of success of this attack is:

$$\mathbb{P}_{wots}^{attack}(q) \leq q \cdot \frac{k}{n} + \sum_{i=0}^{q-1} \frac{k}{n-i} \leq \frac{2 \cdot q \cdot k}{n-q} = \frac{2 \cdot q \cdot (d+v)}{p^{D_{wots}} - q}$$

4.4.3 Attack the WOTS public key compression

Conjecture 6. *One of the best attack is to first fix an arbitrary slot, and then to repeatedly replace the first hash chain by a crafted one, and to run the compression of the WOTS public key until the end, hoping for a collision with one of the intermediary $v/Rate$ digests (containing D_{merkle} field elements).*

Such an attack succeeds with probability:

$$\mathbb{P}_{pkCompress}^{attack}(q) \leq \frac{q}{p^{D_{merkle}}}$$

4.4.4 Attack the Merkle tree

We describe an attack against the Merkle tree, parameterized by $n \in [1..2N]$, in 2 steps:

1. **Construction of n colliding sub-Merkle-trees**: Finding n new, distinct, subtrees with a *common merkle root*. Let's call $root_{attack}$ this Merkle root, shared by the n subtrees. Let's denote by l_i the height of the i^{th} subtree. For simplicity, let's assume wlog that the first slot of our XMSS Merkle tree (the real one) starts at 0. The i^{th} subtree corresponds to slots $[k_i \cdot 2^{l_i}..(k_i+1) \cdot 2^{l_i}]$ for some offset k_i. All the tuples (l_i, k_i) should be distinct. For each subtree, the attacker should know at least one valid Merkle authentication path for a WOTS, and the corresponding secret key. For each subtree, the location of the root, when (virtually) integrated into the main Merkle tree, should always be on the same side (left / right), before compression. In other words, either all the (k_i) are even, or they are all odd. Wlog, we assume they are on the left ((k_i) are even).

2. **Insertion of a subtree**: Looping over $neighbor \in \mathbb{F}^{D_{merkle}}$, hoping to find a lucky neighbor allowing to insert one of the n subtrees into the real Merkle tree. At each iteration, the adversary computes

$$h_{T+2 \cdot D_{merkle}, D_{merkle}}(tweak_{\text{pk}}, root_{attack}, neighbor)$$

... and hopes that it collides with one of the n possible targets in the main Merkle tree (the target corresponding to the i^{th} subtree is the hash digest, in the real Merkle tree, located at height $l_i + 1$ (starting from bottom), and column k_i (starting from left)).

Conjecture 7. *No attack against the Merkle tree is more efficient than the protocol described above (for some value of n).*

Step 1 can be reduced to a generalized Birthday problem: Finding n distinct inputs that, when hashed, collides to the same value.

A union bound over all the possible tuples of n values created among the q queries gives an upper bound on the probability of success of step 1:

$$\mathbb{P}_{step1}(q) \leq \frac{\binom{q}{n}}{(p^{D_{merkle}})^{n-1}} \leq \frac{q^n}{(p^{D_{merkle}})^{n-1}}$$

The bound for step 2 is straightforward: at each iteration, there are n targets, in a space of size $p^{D_{merkle}}$

$$\mathbb{P}_{step2}(q) \leq \frac{n \cdot q}{p^{D_{merkle}}}$$

Overall, such an attack succeeds with probability:

$$\mathbb{P}^{attack}_{merkle}(q) \leq \max_{1 \leq n \leq 2 \cdot L} \left[\min(1, \frac{q^n}{(p^{D_{merkle}})^{n-1}}) \cdot \min(1, \frac{n \cdot q}{p^{D_{merkle}}}) \right]$$

4.5 Final bound

The probability of success of the attack described in Section 4.1, with q queries, is bounded by:

$$\mathbb{P}(SA) \leq \frac{S^2}{p^T} + q \cdot \mathbb{P}_{enc} + \frac{2 \cdot q \cdot (d+v)}{p^{D_{wots}} - q} + \frac{q}{p^{D_{merkle}}} + \mathbb{P}^{attack}_{merkle}(q)$$

Note: the term $\frac{S^2}{p^T}$ is not tight at all (which is fine as long as there are not too many signers at the same time).

5 Instantiations

We suggest:

- $T = 8$
- $R = 5$
- $D_{merkle} = D_{wots} = 5$

Assuming $S < 2^{50}$ signers/slot, and $d + v < 256$, we get, for $q < 2^{140}$:

$$\mathbb{P}(SA) \leq 2^{-147} + q \cdot \mathbb{P}_{enc} + \frac{q}{2^{140}}$$

As a result, we get ≥ 140 bits of *security-against-best-known-attacks*, if we omit the encoding.

If we use the trivial encoding (cf here), we can for instance obtain:

- $v = 68$, $w = 4$, $d = 85$: 135 bits of *security-against-best-known-attacks*, < 7 bits of grinding at signing, 151 hashes at verification, signature size of 1.91 KiB.
- $v = 45$, $w = 8$, $d = 130$: 134 bits of *security-against-best-known-attacks*, < 8 bits of grinding at signing, 184 hashes at verification, signature size of 1.48 KiB.

References

[1] J. Drake, D. Khovratovich, M. Kudinov, and B. Wagner, “Hash-based multi-signatures for post-quantum ethereum,” Cryptology ePrint Archive, Paper 2025/055, 2025. [Online]. Available: https://eprint.iacr.org/2025/055

[2] ——, “Technical note: LeanSig for post-quantum ethereum,” Cryptology ePrint Archive, Paper 2025/1332, 2025. [Online]. Available: https://eprint.iacr.org/2025/1332