Hash Based Signatures

DSC 4110/6111: Modern Cryptography and AI/ML Security

Dr. Laltu Sardar

School of Data Science,
Indian Institute of Science Education and Research Thiruvananthapuram (IISER TVM)

August 19, 2026

Table of Content

Hash Based Signatures

From Mathematical Hardness to Hash-Based Signatures

We have seen digital signature schemes based on mathematical hardness assumptions:

RSA Factoring	DSA / ElGamal DLP	ECDSA ECDLP

Their security relies on problems that are believed to be computationally hard for classical computers.

But a sufficiently powerful quantum computer can use **Shor's algorithm** to solve

Factoring and DLP / ECDLP.

Therefore,

RSA / DSA / ECDSA $\longrightarrow$ Quantum Vulnerability

A Different Foundation for Digital Signatures

Can we construct digital signatures without relying on

Factoring, DLP, ECDLP?

A natural candidate is the cryptographic hash function.
Cryptographic hash functions provide properties such as

Preimage Resistance

Second-Preimage Resistance

Collision Resistance

These properties can be used to construct digital signatures.

Hash-Based Signatures

Hash-based signatures build digital signatures primarily from **cryptographic hash functions**, rather than number-theoretic hardness assumptions.

Hash Function $\longrightarrow$ Digital Signature

Lamport-Diffie One-Time Signature (LD-OTS)

Lamport-Diffie One-Time Signature (LD-OTS)

The **Lamport-Diffie One-Time Signature (LD-OTS)** is a hash-based digital signature scheme.
Let

$$H : \{0,1\}^* \to \{0,1\}^n$$

be a cryptographic hash function.
The basic idea is simple:

$$\boxed{\text{Secret random values} \xrightarrow{H} \text{Public values}}$$

For each bit of the message, the signer prepares two secret values:

$$\boxed{(x_{i,0}, x_{i,1})}$$

and publishes

$$\boxed{(y_{i,0}, y_{i,1}) = (H(x_{i,0}), H(x_{i,1})).}$$

A signature reveals exactly one of the two secret values for each message bit.

One-Time

The same LD-OTS key pair should be used to sign **only one message**.

LD-OTS: Key Generation

Let the hash output be n bits:

$$H : \{0,1\}^* \to \{0,1\}^n.$$

For a message digest

$$m = b_1 b_2 \cdots b_n,$$

generate two random secret values for every bit:

$$x_{i,0}, x_{i,1} \xleftarrow{\$} \{0,1\}^n.$$

Thus the secret key is

$$\boxed{SK = \{x_{i,0}, x_{i,1}\}_{i=1}^{n}.}$$

Compute

$$y_{i,0} = H(x_{i,0}), \qquad y_{i,1} = H(x_{i,1}).$$

The public key is

$$\boxed{PK = \{y_{i,0}, y_{i,1}\}_{i=1}^{n}.}$$

LD-OTS: Signing

Let the message digest be

$$m = b_1 b_2 \cdots b_n.$$

For each bit b_i:

$$\boxed{\sigma_i = x_{i,b_i}.}$$

That is,

$$b_i = 0 \quad \Rightarrow \quad \sigma_i = x_{i,0},$$

$$b_i = 1 \quad \Rightarrow \quad \sigma_i = x_{i,1}.$$

The complete signature is

$$\boxed{\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n).}$$

The signer therefore reveals one secret value from each pair.

Important

The other value in every pair remains secret.

LD-OTS: Verification

Given message digest

$$m = b_1 b_2 \cdots b_n$$

and signature

$$\sigma = (\sigma_1, \ldots, \sigma_n),$$

the verifier computes

$$H(\sigma_i)$$

for every i.
The verifier accepts if

$$\boxed{H(\sigma_i) = y_{i,b_i} \qquad \forall i \in \{1, \ldots, n\}.}$$

Equivalently,

$$\boxed{\forall i: \quad H(\sigma_i) = \begin{cases} y_{i,0}, & b_i = 0, \\ y_{i,1}, & b_i = 1. \end{cases}}$$

The verifier therefore checks that every revealed secret value corresponds to the correct public hash value.

LD-OTS: Correctness

During signing, for each message bit b_i, the signer chooses

$$\sigma_i = x_{i,b_i}.$$

The corresponding public value was generated as

$$y_{i,b_i} = H(x_{i,b_i}).$$

Therefore,

$$\boxed{H(\sigma_i) = H(x_{i,b_i}) = y_{i,b_i}.}$$

Hence every verification equation holds:

$$\boxed{H(\sigma_i) = y_{i,b_i} \qquad \forall i.}$$

Therefore, a correctly generated signature always verifies.

Why Is LD-OTS a One-Time Signature?

Why Is LD-OTS a One-Time Signature?

For every message bit b_i, LD-OTS contains two secret values:

$$x_{i,0}, \qquad x_{i,1}.$$

To sign $M = b_1 b_2 \cdots b_n$, the signer reveals only

$$\boxed{\sigma_i = x_{i,b_i}}.$$

Thus, after one signature, the verifier knows one secret from each pair, while the other remains hidden:

$$\boxed{\{x_{i,0}, x_{i,1}\} \quad \longrightarrow \quad \text{one revealed, one hidden}}$$

For a new message $M' = b'_1 b'_2 \cdots b'_n$, a valid signature would require revealing

$$x_{i,b'_i}.$$

If $b'_i \neq b_i$, this requires the **previously hidden** secret value.

Security Idea

The attacker can compute $H(x_{i,b_i})$ but cannot feasibly recover $x_{i,1-b_i}$ from its hash, assuming preimage resistance of H.

What Goes Wrong If the Key Is Reused?

Suppose the same LD-OTS key is used to sign $M = 1010, \quad M' = 0110.$
The two signatures reveal:

Position	1	2	3	4
M	1	0	1	0
M'	0	1	1	0

Hence, at positions 1 and 2, both secrets are revealed: $\boxed{x_{1,0}, x_{1,1}, x_{2,0}, x_{2,1}}$ while positions 3 and 4 reveal only one secret each.

The attacker can now choose either bit at positions 1 and 2.
For example, construct

$$\boxed{M'' = 0010.}$$

Its required signature components are

$$\boxed{(x_{1,0}, x_{2,0}, x_{3,1}, x_{4,0}),}$$

all of which are already available from the two signatures.
Therefore the attacker can construct a valid signature for M''.

Conclusion

Reusing an LD-OTS key can reveal enough secret values to forge a signature on a new message.

Why Is LD-OTS a One-Time Signature?

From the previous example, whenever two signed messages differ at position i,

$$b_i \neq b'_i,$$

the two signatures reveal both secret values:

$$\boxed{x_{i,0},\ x_{i,1}.}$$

If the messages agree,

$$b_i = b'_i,$$

the same secret is revealed again.

Therefore, after signing multiple messages, the attacker may learn both secrets at several positions:

$$\boxed{\text{Different bits} \quad \Longrightarrow \quad \text{both secrets revealed.}}$$

The attacker can then choose either secret at those positions and construct signatures for other messages.

Verdict

$$\boxed{\text{One LD-OTS key pair} \quad \longrightarrow \quad \text{one signature only}}$$

Security is guaranteed only when the OTS key is never reused.

Message Digest vs. Message

In the previous example, the attacker constructed a valid signature for $\boxed{m'' = 0010}$ but m'' is a **message digest**, not necessarily an actual message.
For a real message $M'' \in \{0,1\}^*$, we need

$$\boxed{H(M'') = m''.}$$

Thus, obtaining a valid signature for a chosen digest does **not** automatically give a valid forgery for a meaningful message. However, key reuse can reduce the attacker's search space.
For example, if two signed digests are

$$m = 0101, \qquad m' = 1010,$$

then every position differs, so both secrets are revealed:

$$\boxed{x_{i,0}, x_{i,1} \quad \forall i.}$$

The attacker can therefore generate a valid OTS signature for **any chosen digest** m''.
The remaining challenge is finding a message M'' such that

$$H(M'') = m''.$$

Key Point: OTS key reuse does not directly give a message forgery, but it can make the digest-forgery problem much easier.

Limitations of LD-OTS

Limitations of LD-OTS

LD-OTS is simple and secure, but it has an important efficiency problem.
For every message bit b_i, we need two independent n-bit secret values:

$$(x_{i,0}, x_{i,1}).$$

For an n-bit message digest:

$$\text{Secret values} = 2n, \qquad \text{Signature values} = n.$$

Thus, the key and signature contain a large number of n-bit values , i.e., $2n^2$ and n^2 bits resp.
The reason is that LD-OTS treats every bit independently:

$$1 \text{ bit} \quad \longrightarrow \quad 2 \text{ secret values.}$$

For a long message digest, this increases

$$\text{key size}, \qquad \text{signature size}, \qquad \text{hash computations.}$$

Question

Can we use one n-bit secret value to represent multiple message bits and thereby reduce the size?

Yes.

Winternitz-OTS (W-OTS)

Winternitz-OTS (W-OTS)

Winternitz-OTS: Design Philosophy of Basic Construction

LD-OTS uses two n-bit secrets for every message bit. Can we use **one n-bit secret for several bits**?
Divide the message digest into t blocks:

$$m = m_1 \| m_2 \| \cdots \| m_t, \qquad |m_i| = w, \qquad t = \frac{n}{w}.$$

Each block represents

$$m_i \in \{0, \ldots, 2^w - 1\}.$$

For every block, generate only one n-bit secret:

$$\boxed{x_i \in \{0,1\}^n}.$$

The value m_i determines how many times H is applied to x_i.

$$\boxed{\text{One } n\text{-bit secret} \quad \longrightarrow \quad \text{one } w\text{-bit block}}$$

Goal

Reduce key and signature size while retaining the **one-time** property.

W-OTS: Key Generation

Let

$$H : \{0,1\}^n \to \{0,1\}^n, \qquad L = 2^w - 1.$$

Generate

$$x_1, \ldots, x_t \xleftarrow{\$} \{0,1\}^n.$$

$$\boxed{SK = (x_1, \ldots, x_t)}$$

For each x_i, construct a hash chain:

$$x_i \to H(x_i) \to \cdots \to H^L(x_i).$$

Set

$$\boxed{y_i = H^L(x_i)}$$

and therefore

$$\boxed{PK = (y_1, \ldots, y_t)}.$$

Thus the number of secret/public values is

$$\boxed{t = \frac{n}{w}}.$$

W-OTS: Signing

Write the message digest as

$$m = m_1 \| m_2 \| \cdots \| m_t, \qquad m_i \in \{0, \ldots, 2^w - 1\}.$$

For each block, reveal the corresponding point in its hash chain:

$$\boxed{\sigma_i = H^{m_i}(x_i).}$$

Hence

$$\boxed{\sigma = (\sigma_1, \ldots, \sigma_t).}$$

For example, if $m_i = 3$:

$$x_i \to H(x_i) \to H^2(x_i) \to \boxed{H^3(x_i) = \sigma_i}.$$

The signature reveals one point from each hash chain.

W-OTS: Verification

Given

$$m = m_1 \| \cdots \| m_t$$

and

$$\sigma = (\sigma_1, \ldots, \sigma_t),$$

the verifier computes, for every i,

$$H^{L-m_i}(\sigma_i).$$

Since

$$\sigma_i = H^{m_i}(x_i),$$

we have

$$\begin{aligned} H^{L-m_i}(\sigma_i) &= H^{L-m_i}(H^{m_i}(x_i)) \\ &= H^{L}(x_i) = y_i. \end{aligned}$$

Therefore accept iff

$$\boxed{H^{L-M_i}(\sigma_i) = y_i \quad \forall i.}$$

A Problem: The Message Value Can Be Increased

Consider a hash chain: $x_i \to H(x_i) \to \cdots \to H^L(x_i)$.

Suppose the signed block is

$$m_i = 3.$$

The signature reveals

$$\sigma_i = H^3(x_i).$$

An attacker can easily move **forward**:

$$H(\sigma_i) = H^4(x_i), \qquad H^2(\sigma_i) = H^5(x_i), \ldots$$

Hence the attacker can potentially produce a valid signature for

$$\boxed{m_i' > m_i}.$$

Why?

$$H^{L-m_i'}\left(H^{m_i'}(x_i)\right) = H^L(x_i) = y_i.$$

Problem

The basic construction prevents decreasing a block value, but does not prevent increasing it.

Next: Add a checksum

W-OTS: Adding a Checksum

The basic W-OTS construction has a weakness:
$m_i \to m_i' > m_i$ can be achieved by hashing forward along the chain.
We therefore add a checksum value C with the property

If the message values increase, the checksum must decrease.

For

$$m = m_1 \| \cdots \| m_t, \qquad m_i \in \{0, \ldots, 2^w - 1\},$$

define

$$c = \sum_{i=1}^{t} (2^w - m_i).$$

Hence,

$$m_i \uparrow \implies c \downarrow .$$

The checksum is then appended to the message:

$$m' = (m_1, \ldots, m_t,\ c_1, \ldots, c_{t_c})$$

where $c_1, \ldots, c_{t_c}$ are the base-w digits of c. i.e., they are w-bit block each.

Why Does the Checksum Prevent Forgery?

Suppose an attacker changes a message block: $m_i \to m_i' > m_i$.
From

$$c = \sum_{i=1}^{t} (2^w - m_i),$$

we get

$$c' = \sum_{i=1}^{t} (2^w - m_i') < c.$$

Thus,

Increasing message blocks $\implies$ decreasing checksum.

But the attacker can only move **forward** in a hash chain:

$$H^a(x_i) \to H^{a+1}(x_i).$$

To create the smaller checksum value, the attacker would need to move **backward**:

$$H^{a+1}(x_i) \to H^a(x_i),$$

which requires inverting H.
Idea: The checksum links the message blocks together: increasing the message requires decreasing some checksum block, which cannot be obtained by forward hashing.

Checksum: Example 1

Let

$$w = 4, \qquad t = 4,$$

so each message block is in

$$\{0, \ldots, 2^4 - 1\} = \{0, \ldots, 15\}.$$

Suppose

$$(m_1, m_2, m_3, m_4) = (3, 7, 4, 10).$$

The checksum is

$$\begin{aligned} c &= \sum_{i=1}^{4} (16 - m_i) \\ &= (16 - 3) + (16 - 7) + (16 - 4) + (16 - 10) \\ &= 13 + 9 + 12 + 6 \\ &= \boxed{40}. \end{aligned}$$

Representing c in base 16:

$$40 = 2 \times 16 + 8, \qquad \boxed{(c_1, c_2) = (2, 8)}.$$

Thus the complete block sequence is

$$\boxed{(3, 7, 4, 10, 2, 8)}.$$

Checksum: Example 2

Let

$$w = 4, \qquad t = 4,$$

and consider

$$(m_1, m_2, m_3, m_4) = (4, 6, 8, 2).$$

Then

$$\begin{aligned} c &= (16 - 4) + (16 - 6) + (16 - 8) + (16 - 2) \\ &= 12 + 10 + 8 + 14 \\ &= \boxed{44}. \end{aligned}$$

In base 16:

$$44 = 2 \times 16 + 12, \qquad \boxed{(c_1, c_2) = (2, 12)}.$$

Now increase one message block:

$$(4, 6, 8, 2) \to (4, 7, 8, 2).$$

The checksum becomes

$$c' = 44 - 1 = 43.$$

Thus

$$\boxed{m_i \uparrow \implies c \downarrow}$$

which is exactly the property required for preventing forward-hashing forgery.

W-OTS: Key and Signature Size

Let the message digest contain n bits and let each block contain w bits.

$$t = \frac{n}{w}, \qquad L = 2^w - 1.$$

With checksum, let t_c be the number of checksum blocks. Then

$$\ell = t + t_c.$$

Each chain contributes one n-bit value.

$$|SK| = \ell n, \qquad |PK| = \ell n, \qquad |\sigma| = \ell n.$$

For LD-OTS:

$$|SK| = 2n, \qquad |PK| = 2n, \qquad |\sigma| = n.$$

Thus, for suitable w, W-OTS uses fewer chain values than LD-OTS and can significantly reduce key/signature size.

But

W-OTS is still a **one-time signature**: one W-OTS key pair should be used for only one message.

Why Is W-OTS Still One-Time?

Each W-OTS key contains hash chains

$$x_i \to H(x_i) \to \cdots \to H^L(x_i).$$

For a message block m_i, the signature reveals

$$\sigma_i = H^{m_i}(x_i).$$

After signing, the attacker can always move **forward**:

$$H^{m_i}(x_i) \to H^{m_i+1}(x_i) \to \cdots$$

The checksum prevents increasing the message blocks, but reusing the same chains for another signature can reveal additional chain positions.
Therefore,

W-OTS key pair $\to$ one signature

Next Question

Can we construct a scheme that uses the efficiency of W-OTS but allows the signer to sign **multiple messages**?

One Solution $\to$ Merkle Signature Scheme

Merkle Signature Scheme (MSS)

Merkle Signature Scheme: Design Philosophy

W-OTS is a one-time signature. To sign multiple messages, generate many independent W-OTS key pairs:

$$(SK_1, PK_1), \ldots, (SK_N, PK_N), \qquad N = 2^h.$$

Instead of publishing all PK_i, authenticate them using a Merkle tree.
For each W-OTS public key, compute

$$z_i = H(PK_i).$$

Use these as the leaves of a binary hash tree. The root is

$$\boxed{R = \text{MSS public key.}}$$

Thus,

$$\boxed{2^h \text{ W-OTS keys} \quad \longrightarrow \quad 1 \text{ Merkle root}}$$

Each W-OTS key is used only once.

MSS: Key Generation

Generate $N = 2^h$ independent W-OTS key pairs:

$$(SK_i, PK_i), \qquad i = 0, \ldots, N-1.$$

Compute the leaf values

$$z_i = H(PK_i).$$

Build the binary Merkle tree:

$$z_i^{(0)} = z_i,$$

$$z_i^{(j+1)} = H\left(z_{2i}^{(j)} \| z_{2i+1}^{(j)}\right).$$

After h levels, obtain the root

$$\boxed{R = z_0^{(h)}}.$$

The keys are

$$\boxed{PK_{\mathrm{MSS}} = R}$$

and the signer stores the W-OTS secret keys and the tree state.

Add picture here

MSS: Signing

To sign the next message M, choose an unused OTS key SK_i.
First generate the OTS signature:

$$\sigma_{\mathrm{OTS}} = \mathrm{Sign}_{SK_i}(M).$$

Let

$$z_i = H(PK_i).$$

Provide the Merkle authentication path

$$A_i = (a_0, a_1, \ldots, a_{h-1}),$$

containing one sibling node from each tree level.
The MSS signature is

$$\boxed{\sigma = (i, \sigma_{\mathrm{OTS}}, PK_i, A_i).}$$

After signing, mark SK_i as used.

$$i \leftarrow i + 1.$$

MSS: Verification

Given

$$(M, \sigma) = (M, i, \sigma_{\mathrm{OTS}}, PK_i, A_i),$$

first verify the OTS signature:

$$\mathrm{Verify}_{PK_i}(M, \sigma_{\mathrm{OTS}}) = 1.$$

Compute the leaf:

$$z_i = H(PK_i).$$

Using the authentication path, reconstruct the Merkle root:

$$z_i \to z^{(1)} \to \cdots \to z^{(h)}.$$

At each level,

$$z^{(j+1)} = H(z^{(j)} \| a_j)$$

or

$$H(a_j \| z^{(j)}),$$

according to the position of the node.
Accept iff

$$\boxed{z^{(h)} = R}.$$

MSS: Why Is It Secure?

An MSS forgery requires breaking at least one of two components.

1. OTS forgery

The attacker must produce a valid OTS signature for a new message without knowing the corresponding secret key.

This relies on the one-wayness of H.

2. Merkle-tree forgery

The attacker must produce a different PK_i that still leads to the known root R.

This requires finding a hash collision or a suitable second-preimage.

Thus,

$$\boxed{\text{MSS security} \leftarrow \text{OTS security} + \text{Hash-function security}}$$

The crucial requirement is that every OTS key is used only once.

Example

XMSS uses W-OTS+ as its underlying OTS scheme.

Secret key of MSS

Secret Key Management in MSS

Recall that an MSS tree contains 2^h one-time signature keys:

$$(SK_0, PK_0), (SK_1, PK_1), \ldots, (SK_{2^h-1}, PK_{2^h-1})$$

- To construct the Merkle tree, the public key of every leaf is required:

$$L_i = H(PK_i)$$

- Therefore, conceptually, we need access to all **one-time secret keys** SK_i during key generation.
- Storing all SK_i explicitly is inefficient.
- We therefore need a mechanism to **generate/recover** each SK_i when required.

Key observation

The individual SK_i values do not need to be stored permanently. They can be deterministically generated from a single secret seed.

Seed-Based Generation of MSS Secret Keys

Let SK_{seed} be a master secret seed.

A pseudorandom function (PRF) or secure pseudorandom generator is used to derive each one-time secret key:

$$\boxed{SK_i = \text{PRF}(SK_{\text{seed}}, i)}$$

where i is the leaf index.

- The signer stores only the master seed SK_{seed}.
- When SK_i is required, it is regenerated using its index i.
- Hence, the large collection of secret keys need not be stored.

Important

The **master seed** is the actual long-term secret. The individual SK_i values are derived secrets.

Limitations of MSS

Need of a few time signature scheme

Limitations of Merkle Signature Scheme (MSS)

Although MSS enables multiple signatures using hash functions, it has some important limitations:

- **Stateful signing**
  - Each leaf corresponds to a one-time signature key.
  - The signer must maintain the index of the next unused leaf.
  - Reusing a leaf can compromise security.
- **Limited number of signatures**
  - A tree with 2^h leaves supports only 2^h signatures.
  - A new tree/key pair is required after all leaves are exhausted.
- **Key/state management**
  - The signing state must be stored reliably.
  - Rollback, duplication, or loss of state can cause key reuse.

Key limitation

MSS provides many signatures by carefully managing a collection of **one-time signatures**, but this comes at the cost of **state management**.

From One-Time to Few-Time Signatures

A natural question arises:

Can we safely use a hash-based signing key for a small number of signatures without maintaining a strict state?

- **One-Time Signature (OTS)**
  - A secret key is used to sign only one message.
  - Example: WOTS+.
- **Few-Time Signature (FTS)**
  - A signing key can safely sign a **small number of messages**.
  - Security gradually degrades as more signatures are issued.
  - The construction can remain **stateless**.
- **Key idea**
  - Generate a large collection of secret values.
  - Each message selects a small subset of these values.
  - Reveal only the selected values in the signature.

Example: FORS

HORS is a few-time signature component used in SPHINCS+ / SLH-DSA.

HORS: Hash to Obtain Random Subsets:

a few time Signature Scheme

From MSS to HORS

MSS achieves multiple signatures by maintaining a collection of one-time signature keys.

- Can we construct a signature scheme that is:
  - based only on hash functions,
  - **stateless**, and
  - allows a **small number of signatures**?
- Instead of using one secret key for one message, generate a large collection of independent secret values:

$$x_0, x_1, \ldots, x_{t-1}.$$

- A message determines a small subset of these values.
- The signer reveals only the selected secret values.

Key Idea

Use a large pool of secret values and let each message select a small random subset.

HORS = Hash to Obtain Random Subsets

HORS: Basic Construction

Let:

$$t = \text{number of secret values}, \qquad k = \text{number of values selected per signature.}$$

Key Generation:

$$SK = (x_0, x_1, \ldots, x_{t-1})$$

$$PK = (y_0, y_1, \ldots, y_{t-1}), where y_i = H(x_i)$$

Signing:

$$H(M) \rightarrow (i_1, i_2, \ldots, i_k)$$

and reveal:

$$\boxed{\sigma = (x_{i_1}, x_{i_2}, \ldots, x_{i_k})}$$

Verification:

$$H(x_{i_j}) \stackrel{?}{=} y_{i_j} \qquad j = 1, \ldots, k.$$

Observation: The signature reveals only k out of the t secret values.

HORS: Simple Example

Suppose:

$$t = 16, \qquad k = 4.$$

The secret key contains:

$$x_0, x_1, \ldots, x_{15}.$$

The public key contains:

$$H(x_0), H(x_1), \ldots, H(x_{15}).$$

Suppose:

$$H(M) \to (3, 7, 11, 14).$$

The signer reveals:

$$\boxed{\sigma = (x_3, x_7, x_{11}, x_{14})}$$

The verifier checks:

$$H(x_3) = y_3, \quad H(x_7) = y_7, \quad H(x_{11}) = y_{11}, \quad H(x_{14}) = y_{14}.$$

Core intuition The message determines **which secrets can be revealed**. The signature proves that the signer knows those particular secrets.

Why is HORS a Few-Time Signature?

Each signature reveals k secret values.

After several signatures, the adversary gradually learns a set of secret values:

$$S = \{x_i : \text{revealed in previous signatures}\}.$$

A forgery is possible if a new message selects only already-revealed indices.

If r distinct secret values have been exposed, then approximately:

$$\Pr[\text{successful forgery}] \approx \left(\frac{r}{t}\right)^k.$$

Therefore:

More signatures $\Rightarrow$ more exposed secrets $\Rightarrow$ higher forgery probability

Few-Time Property

HORS does not have a fixed "number of uses". Its security gradually decreases as more signatures are generated.

HORST: Hash-based One-time Randomized Signature Technique:

a few time Signature Scheme

Limitations of HORS

HORS reduces the signature size by revealing only k secret values from a large set of t secret values. However, it has an important drawback.
For

$$x_1, \ldots, x_t \xleftarrow{\$} \{0,1\}^n,$$

the public key contains

$$y_i = H(x_i), \qquad PK = (y_1, \ldots, y_t).$$

Thus,

$$\boxed{|PK| = tn}$$

which can be very large when t is large.
The question is:

Can we compress the HORS public key?

We need a single compact value that authenticates all $y_1, \ldots, y_t$.

Idea

Use a Merkle tree to replace the large HORS public key by its **Merkle root**.

HORST: Design Philosophy

HORS has

$$t$$

public values:

$$y_i = H(x_i).$$

Build a binary Merkle tree whose leaves are these values:

$$y_1, \ldots, y_t \quad \longrightarrow \quad \text{Merkle Tree} \quad \longrightarrow \quad R.$$

The root becomes the HORST public key:

$$\boxed{PK_{\text{HORST}} = R}.$$

For a message, HORS selects k indices

$$i_1, \ldots, i_k.$$

The signature reveals the corresponding secret values

$$x_{i_1}, \ldots, x_{i_k}$$

together with their Merkle authentication paths.
Thus,

$$\boxed{\text{HORS} + \text{Merkle Tree} \quad \longrightarrow \quad \text{HORST}}$$

HORST: Key Generation

Let

$$t = 2^{\tau}$$

and

$$H : \{0,1\}^* \to \{0,1\}^n.$$

Generate

$$x_1, \ldots, x_t \xleftarrow{\$} \{0,1\}^n.$$

The secret key is

$$\boxed{SK = (x_1, \ldots, x_t)}.$$

Compute the HORS public values:

$$y_i = H(x_i).$$

Use

$$y_1, \ldots, y_t$$

as the leaves of a binary Merkle tree.
Let R be its root.
The public key is simply

$$\boxed{PK = R}.$$

Hence the public key size is reduced from tn bits to n bits.

HORST: Signing

Hash the message and derive k indices:

$$H(M) \longrightarrow (i_1, i_2, \ldots, i_k), \qquad i_j \in \{1, \ldots, t\}.$$

For each selected index, reveal the corresponding secret:

$$\boxed{\sigma_j = x_{i_j}.}$$

For every selected leaf, also provide its Merkle authentication path:

$$A_{i_j} = (a_{j,0}, \ldots, a_{j,\tau-1}).$$

The HORST signature is therefore

$$\boxed{\sigma = (x_{i_1}, \ldots, x_{i_k}, A_{i_1}, \ldots, A_{i_k})\,.}$$

Only k of the t secret values are revealed.

HORST: Verification

The verifier obtains

$$(i_1, \ldots, i_k)$$

from $H(M)$.
For each revealed secret x_{i_j}, compute

$$y_{i_j} = H(x_{i_j}).$$

Using its authentication path, reconstruct the Merkle root:

$$y_{i_j} \to \cdots \to R_j.$$

Accept iff every reconstructed root equals the public root:

$$\boxed{R_1 = R_2 = \cdots = R_k = PK.}$$

Thus the verifier checks both:

$$\text{HORS validity} \quad + \quad \text{membership of each leaf in the Merkle tree.}$$

HORST: Security and Limitation

The Merkle tree hides the large HORS public key, but it does not remove the fundamental HORS reuse problem. Each signature reveals k secret values:

$$x_{i_1}, \ldots, x_{i_k}.$$

If the same HORST key is reused many times, more secret values are revealed.
Eventually, an attacker may know enough secret values to construct a valid signature for a new set of indices.
Therefore,

HORST is a few-time signature scheme.

Its security depends on:

one-wayness of H + difficulty of obtaining enough unrevealed secrets.

Key Improvement

HORS has a large public key; HORST replaces it with one Merkle root, while retaining the HORS-based few-time signing idea.

FORS: Forest of Random Subsets:

a few time Signature Scheme

FORS: Design Philosophy

HORS selects several secret values from one large set.
HORST reduces the large HORS public key using a Merkle tree.
FORS takes a different approach:

$$\boxed{k \text{ independent trees}}$$

Each tree contains

$$2^a$$

secret values.
For every tree, the message determines exactly one leaf to reveal.
Thus, for a message digest of ka bits,

$$m = m_1 \| m_2 \| \cdots \| m_k, \qquad |m_i| = a,$$

each m_i selects one leaf in tree i.

$$\boxed{\text{one message} \quad \longrightarrow \quad \text{one leaf from each of } k \text{ trees}}$$

Main Idea

FORS provides a compact few-time signature component using only hash functions and Merkle authentication paths.

FORS: Key Generation

Let

$$H : \{0,1\}^* \to \{0,1\}^n.$$

Construct k independent binary trees, each of height a.
Tree i contains

$$2^a$$

random secret values:

$$x_{i,0}, x_{i,1}, \ldots, x_{i,2^a-1}.$$

The leaf values are

$$y_{i,j} = H(x_{i,j}).$$

Build a Merkle tree for each set of leaves and let its root be

$$R_i.$$

The FORS public key is derived from all k roots:

$$\boxed{PK_{\text{FORS}} = H(R_1 \| R_2 \| \cdots \| R_k).}$$

The secret key consists of the secret leaf values, or a seed from which they are deterministically generated.

FORS: Signing

Hash the message and divide the resulting ka bits into k parts:

$$H(M) = m_1 \| m_2 \| \cdots \| m_k, \qquad |m_i| = a.$$

Each m_i selects one leaf in tree i:

$$j_i = \text{bin}(m_i).$$

The signer reveals the selected secret value

$$x_{i,j_i}$$

and its Merkle authentication path

$$A_{i,j_i}.$$

Hence the FORS signature is

$$\boxed{\sigma_{\text{FORS}} = (x_{1,j_1}, \ldots, x_{k,j_k}, A_{1,j_1}, \ldots, A_{k,j_k})}.$$

One leaf is revealed from each tree.

FORS: Verification

The verifier obtains

$$m_1, \ldots, m_k$$

from $H(M)$ and therefore knows the selected leaves

$$j_1, \ldots, j_k.$$

For each tree i, compute

$$y_{i,j_i} = H(x_{i,j_i}).$$

Using the authentication path, reconstruct the corresponding root:

$$y_{i,j_i} \to \cdots \to R'_i.$$

The verifier then computes

$$PK' = H(R'_1 \| R'_2 \| \cdots \| R'_k).$$

Accept iff

$$\boxed{PK' = PK_{\mathrm{FORS}}.}$$

Thus, the verifier checks that every revealed secret belongs to the correct FORS tree.

FORS: Why Does It Work?

For each selected leaf,

$$x_{i,j_i}$$

is revealed.
But its sibling nodes on the path remain hidden.
The authentication path proves that

$$H(x_{i,j_i})$$

is a leaf of the tree with root R_i.
Therefore, a valid signature establishes

$$\text{revealed secret} \rightarrow \text{leaf} \rightarrow \text{tree root} \rightarrow PK_{\text{FORS}}.$$

An attacker who does not know another leaf secret cannot simply replace a selected leaf with a new one without breaking the hash function.

Security

FORS security relies on the one-wayness and collision resistance of the underlying hash function.

FORS: Why Is It Few-Time?

A FORS signature reveals one secret value from each of the k trees.
After one signature:

$$k \text{ secret values are revealed.}$$

After multiple signatures, more leaves become known.
If the same leaf is selected again, no new secret is revealed. But different signatures may gradually expose many different leaves.
Thus,

$$\boxed{\text{More signatures} \longrightarrow \text{more revealed secrets.}}$$

If enough secret values become known, the attacker may construct a signature for another set of indices.
Therefore, FORS is designed as a

$$\boxed{\text{few-time signature component}}$$

rather than an unlimited-use signature scheme.

FORS: HORS, HORST and FORS

The constructions can now be viewed as successive improvements:

Scheme	Main Idea
HORS	Select several secrets from one large set
HORST	Use a Merkle tree to compress the HORS public key
FORS	Use k independent small Merkle trees

For FORS:

$$\boxed{k \text{ trees} \times 2^a \text{ leaves/tree}}$$

and one message selects

$$\boxed{1 \text{ leaf from each tree.}}$$

This structure makes FORS particularly suitable as a component inside a larger stateless hash-based signature construction.

Question: How can FORS be combined with a larger hash-based signature structure to obtain a practical post-quantum signature scheme? Answer–

SPHINCS+

SPHINCS+:

a Stateless hash based signature scheme

In the next day's slide

From FORS to SPHINCS+

We have seen several hash-based constructions:

LD-OTS	One-time signature
W-OTS	More compact OTS
MSS	Many OTS keys using a Merkle tree
HORS	Few-time signature
HORST	HORS with Merkle-tree compression
FORS	Few-time signature using multiple trees

We now want a practical signature scheme that is:

$$\text{hash-based} \quad + \quad \text{post-quantum secure} \quad + \quad \text{stateless.}$$

The challenge is to combine these ideas without requiring the signer to maintain a state indicating which one-time key has been used.

Goal

Construct a practical **stateless hash-based signature scheme**.

Why Do We Need a Stateless Construction?

In MSS, every W-OTS key must be used only once.
Therefore, the signer maintains a state:

$$i = 0, 1, \ldots, 2^h - 1.$$

After signing:

$$i \leftarrow i + 1.$$

If the state is lost, rolled back, or incorrectly synchronized, an OTS key may be reused.
This creates an important practical problem:

$$\boxed{\text{State management} \quad \longrightarrow \quad \text{key-reuse risk}}$$

Can we construct a hash-based signature scheme where the signer does not need to remember which OTS key was previously used?

$$\boxed{\text{Stateless Hash-Based Signature}}$$

SPHINCS+: Basic Construction Idea

SPHINCS+ combines the ideas we have already studied.
At a high level:

$$\boxed{\text{FORS} + \text{W-OTS} + \text{Hypertree}}$$

The message is first processed by a FORS instance:

$$M \longrightarrow \text{FORS signature.}$$

The FORS public key is then authenticated through a hierarchy of W-OTS and Merkle trees:

$$\text{FORS} \to \text{W-OTS} \to \text{Merkle tree} \to \cdots \to \text{root.}$$

The final public key is the root of the top-level tree.

Key Idea

SPHINCS+ replaces explicit signer state with a large, deterministically addressable collection of one-time keys.

SPHINCS+ Structure

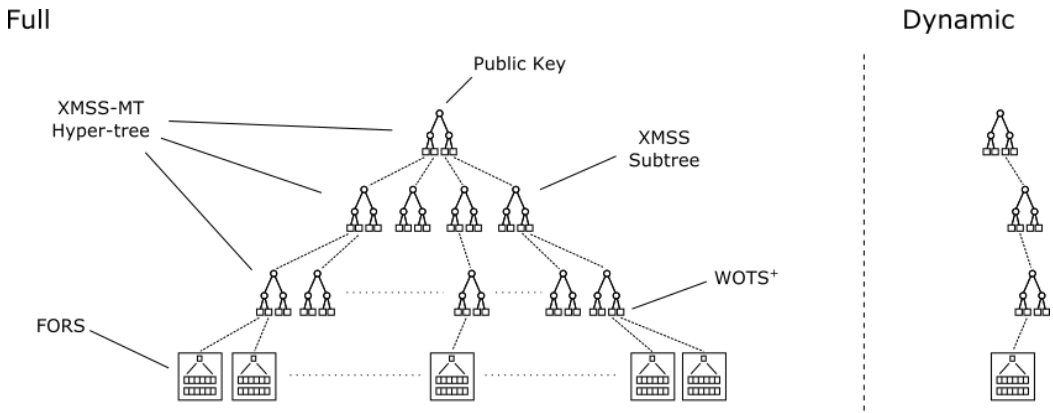

Figure: Overview of the SPHINCS+ signature scheme [1]

[1] Source: https://ar5iv.labs.arxiv.org/html/1904.06525

References

Textbook:

1. Douglas R. Stinson and Maura Paterson. *Cryptography: Theory and Practice*. 4th Edition. Chapman & Hall/CRC Press, 2019. ISBN: 9780367148591.

Acknowledgment:

- The figures in this work were generated with the assistance of ChatGPT (OpenAI).

Dr. Laltu Sardar, Assistant Professor, IISER Thiruvananthapuram
laltu.sardar@iisertvm.ac.in, laltu.sardar.crypto@gmail.com
Course webpage: https://laltu-sardar.github.io/courses/26_crypto.html.