Basics of MD Hashes and Hash-Based Signatures

John Kelsey, NIST and KU Leuven
Overview of Talk

*SPHINCS*⁺ attack depends on a lot of obscure details
In this talk, I’m going to cover some basics to make the attack easier to understand.

1. Some internal details of SPHINCS⁺
2. Some internal details of SHA256
Part 1: Hash based signatures

4 bits (0..15) encoded in each hash chain

private key

public key

checksum
Preliminaries: Hash Functions

What do we need from a hash function?

- Collision resistance (Important generally, not for our attack)
- Preimage resistance
- Second preimage resistance
- Many other properties may be important for other applications

Note: cryptographic hash functions are designed to behave randomly.
For any hash function, we have these generic attacks:

- **Preimages** (Given $H$, find $X$ so $\text{HASH}(X) = H$)
  Just try $2^n$ values for $X$ until $\text{HASH}(X) = H$.

- **Second Preimages** (Given $X$, find $Y$ so $\text{HASH}(X) = \text{HASH}(Y)$)
  Just try $2^n$ values for $Y$ until $\text{HASH}(X) = \text{HASH}(Y)$.

If hash function behaves randomly, these are the best we can do.
Multitarget attacks

- Suppose I have $N$ different target hashes, $H_1, \ldots, H_N$
- Multitarget preimage: find $X$ such that $\text{HASH}(X) \in \{H_1, \ldots, H_N\}$
- This is $N$ times faster than normal preimage attack
- SPHINCS+ has a huge number of target hash values—need to prevent this attack!
Defense: Prefixes

- To prevent multitarget attacks, SPHINCS+ employs a unique prefix
  - Every single hash call in SPHINCS+ has a unique prefix
  - prefix = $PK\cdot$seed $\parallel$ ADRS
  - Result: Multitarget attacks blocked

$$H_1 \leftarrow \text{HASH}(P_1 \parallel M_1)$$

$$\ldots$$

$$H_N \leftarrow \text{HASH}(P_N \parallel M_N)$$

- ...because hash $H_i$ always has only one valid prefix, $P_i$
What’s a Signature

1. Public and Private keys:
   - $PK, SK \leftarrow \text{Generate}()$
   - Private key: Only I know this
   - Public key: I want everyone to know this

2. Signing:
   - Sign with private key $SK$
   - $\sigma \leftarrow \text{Sign}(SK, M)$

3. Verification:
   - $\text{Verify}(\sigma, M, PK) \rightarrow \text{“good” or “bad”}$

Normally, we can sign many messages with one key.
What’s a **One-Time** Signature? A **Few-Times** Signature?

- One-time signature: I can only sign one message per keypair
  - Signing two different messages lets an attacker forge signatures!
  - Note: Signing same message twice is fine.
  - **WOTS**+ used for one-time signatures in SPHINCS+

- Few-times signature: I can sign up to $N$ distinct messages safely
  - Sign too many—leak too much information—attacker can forge signatures
  - $N$ is usually not super large—like 10 or 20.
  - **FORS** used for few-times signatures in SPHINCS+

*SPHINCS*+ uses both of these
Winternitz/WOTS+ Signatures

One-time signature scheme
- Based on hash chains
- Requires a checksum
- Used in SPHINCS+

WOTS+ is specific variant of Winternitz used in SPHINCS+
Digression: Hash Chain

Compute each element in chain by hashing previous element.

\[ X_i = \text{HASH}(X_{i-1}) \]

Only need to know starting value—can compute all other values from there.

Can’t go backward in chain because of preimage resistance.
Winternitz: Signing 2 bits with one hash chain

- Compute hash chain $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow y$
- $x_0$ is private signing key; $y$ is public key
- To sign value 01, we reveal $x_1$.
- To verify, walk rest of chain: $y = \text{HASH(HASH(HASH}(x_1)))$

Works with chains of length $2^w$, for any $w \geq 1$!
Hash chain of $2^w$ entries encodes $w$ bits

So we can encode 256-bit hash with $\left\lceil \frac{256}{w} \right\rceil$ hash chains.
Problem: Attacker can increment values

Arrrows represent hash operations!

- Let: $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow y$
- $y$ is public key

- To sign value 01, we reveal $x_1$.
- Anyone can walk rest of chain: $y = \text{HASH}(\text{HASH}(\text{HASH}(x_1)))$
  - But anyone can change a signature on 01 to a signature on 10 or 11...
  - ...just keep computing the hash!

This is why we need a checksum
The Winternitz checksum

- Write \( \text{HASH}(\text{message}) \) as a sequence of \( a \) 4-bit digits, \( t_0, t_1, \ldots, t_{a-1} \)
- \( \text{max} = a(2^w - 1) \)
- Checksum \( \leftarrow \text{max} - \sum_{i=0}^{a} t_i \)
- Now, walking forward on any chain requires walking backward on checksum!
- Checksum is written as a base-16 number and encoded in three more hash chains

\textit{Checksum ensures any change requires going backward on some hash chain}
Winternitz/WOTS+: Encoding the checksum

- Need 64 hash chains of length $2^4$ to encode hash
  - One for each 4 bits chunk of hash being signed.
- Maximum possible sum of values in those chains is:

\[ 64 \times 15 = 960 \]

- $0 \leq \text{Checksum} \leq 960$
  - Need $\lg(\text{max} + 1)$ bits!
- Since each chain encodes 4 bits, we need three more chains to encode checksum.

*SPHINCS+ category 5, Winternitz signature is 67 hash values!*
Making a WOTS+ Public Key in SPHINCS+ (1)

- Private key = $X_{0...66,0}$, generated pseudorandomly.
- prefix[i,j] = the unique prefix for this one time key, this chain (i), and this step (j)
- For each $i = 0...66$:
  - For $j = 1...15$:
    - $X_{i,j} = \text{HASH}(\text{prefix}[i,j] \ || \ X_{i,j-1})$
- Note: Each hash operation incorporates a unique prefix.
Making a WOTS+ Public Key in SPHINCS+ (2)

- Given final values in all 67 hash chains, $X_{0...66,15}$
- Public key also includes a prefix for this particular one-time key ID
- Public key preimage =

$$\text{prefix} \parallel X_{0,15} \parallel X_{1,15} \parallel \ldots \parallel X_{66,15}$$

- Public key = hash of public key preimage
How it’s done in SPHINCS+

Hash-Based Signatures: 19 / 40
How do we use one-time keys?

- One-time signatures aren’t very useful—you want to sign many times
- SPHINCS+ can sign up to $2^{64}$ times
- First tool we need for this: A Merkle tree
Merkle Trees

Hash-Based Signatures:
A binary tree made by hashing things together!

- Make a list of $2^n$ one-time signing keys, $PK_{0,1,2,...,2^n-1}$
- Hash each pair together to make input to next hash.
- Keep going until we reach the root.

The root contains the hash of all the leaves.
I have a list of $2^n$ items.

Compute Merkle tree and give you root.

Later, I can prove $PK_i$ is member of list with $n$ hashes.
A hypertree is a "tree of trees"

Each tree is a Merkle tree full of one-time keys

Each tree after the first is generated on the fly as needed

Each tree has its root signed by a one-time key from the previous tree
Big Idea

1. Generate Merkle Tree of $2^k$ keys.
2. Use each key to sign a Merkle Tree of $2^k$ trees.
3. Result: We have $2^{k^2}$ keys.

And we can iterate this process as many times as we like.
Tree of Trees...of Keys

Getting $2^{40}$ keys with $2^{20}$-element Merkle trees:
Tree of Trees...of Keys

Getting $2^{40}$ keys with $2^{20}$-element Merkle trees:

- Generate a list of $2^{20}$ one-time signing keys.
- For each of those keys, we have a tree of $2^{20}$ one-time signing keys we can generate.
  - Using PRF, we can ensure we always generate same tree*.
- Produce paths through both trees + both signatures!

* This is critical–otherwise we might sign different things with same key!
We can have many levels of trees

In SPHINCS+, we have huge numbers of trees

- Always around $2^{64}$ leaves in the hypertree
- Leaves are used to sign few times signature keys
- SPHINCS+256s (slower/smaller version): 8 layers of tree, each tree of depth 8
- SPHINCS+256f (faster/larger version): 17 layers of tree, each of depth 4
SPHINCS+ Structure

Hypertrees–Trees of Trees:
Structure of SPHINCS+ signatures

- Top level Merkle tree
  - Root = (some of) master public key
  - Leaves = Winternitz one-time keys

- Hypertree of $2^{64}$ or $2^{68}$ one-time keys on bottom layer
  - 8 layers of depth 8, or 17 layers of depth 4
  - A Winternitz one-time key signs root of next Merkle tree
  - Leaf of this tree = next WOTS key used.

- Messages are signed with FORS (few-time signature) keys
  - The final one-time key in the hypertree always signs a FORS key
  - Each FORS key can sign a small number of times before losing security
  - This allows us to have smaller hypertree without losing security
SPHINCS+ Structure
Part 2: Hash functions and Merkle-Damgård Hashes
Merkle-Damgård hashes: How SHA256 is Made

Our result only applies to SPHINCS+ when it is using SHA256 to get 256-bit security

Understanding it requires looking ”under the hood” of SHA256
Merkle-Damgård Hashes (1)

Big idea: Make a good fixed-length hash function, then build a variable-length hash from it.

We need a fixed-length compression function, $F(h, m)$

- $h_{in} =$ hash chaining value, $n$ bits. (Example $n = 256$)
- $h_{out} =$ hash chaining value, $n$ bits.
- $m =$ message block, $w$ bits. (Example $w = 512$)

- Pad the message, break into $w$-bit chunks, and process sequentially.
Merkle-Damgård Hashes: How SHA2 Works

1. Pad message to integer multiple of 512 bits:
   ▶ 10* padding
   ▶ ...plus length of unpadded message (Merkle-Damgård strengthening)

2. Break padded message into 512-bit blocks $m_0, m_1, m_2, \ldots, m_{k-1}$.

3. $h_{-1} =$ fixed initial value, $iv$.

4. $h_i \leftarrow F(h_{i-1}, m_i)$ for $i = 0, 1, 2, \ldots, k - 1$.

5. Final $h_i$ is $\text{HASH}(M)$

*Note: Only impact of $m_0 \ldots i$ is on $h_i$*
Herding Hash Functions

Hypertrees–Trees of Trees:
A problem

- I want to carry out a multitarget preimage attack
- My messages all start with different prefixes
- What can I do?
The Diamond Structure: A Merkle-Tree Computed by Finding Collisions.

- Starting from $2^k$ different prefixes
- Find pairwise collisions to map these down to a single intermediate hash value
- Result: A diamond structure that routes $2^k$ input hash chaining values into hash value

*Note: Edges have multiple message blocks; nodes are hash chaining values.*
How this is used in our attack

Hypertrees–Trees of Trees:

- P1 || free1
- P2 || free2
- P3 || free3
- P4 || free4
- P5 || free5
- P6 || free6
- P7 || free7
- P8 || free8

- h0
- h1
- h2
- h3
- h4
- h5
- h6
- h7
- h10
- h11
- h12
- h21

- t0 → Original P1 hash
- t1 → Original P2 hash
- t2 → Original P3 hash
- t3 → Original P4 hash
- t4 → Original P5 hash
- t5 → Original P6 hash
- t6 → Original P7 hash
- t7 → Original P8 hash
Wrapup

- We’ve discussed internals of SPHINCS+
  - WOTS+ signatures
  - Merkle trees
  - Hypertrees
  - How SPHINCS+ works
- ...and internals of SHA256
  - Merkle-Damgård hashes
  - Multitarget attacks
  - The diamond structure