Basics of MD Hashes and Hash-Based Signatures

John Kelsey, NIST and KU Leuven

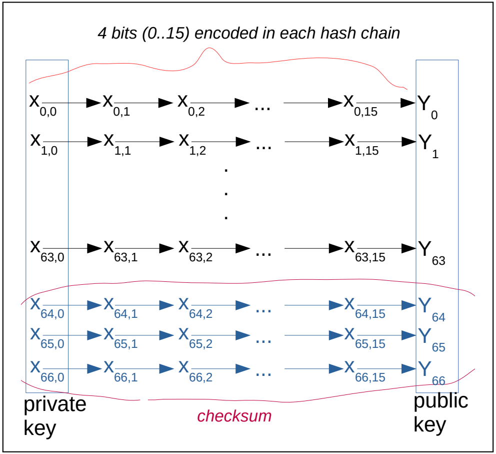
Overview of Talk

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



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.

Generic Attacks

For **any** *hash function, we have these generic attacks:*

- Preimages (Given H, find X so $\texttt{HASH}(X) = H$)

  Just try 2^n values for X until $\texttt{HASH}(X) = H$.

- Second Preimages (Given X, find Y so $\texttt{HASH}(X) = \texttt{HASH}(Y)$)

  Just try 2^n values for Y until $\texttt{HASH}(X) = \texttt{HASH}(Y)$.

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

Multitarget attacks

- Suppose I have N different target hashes, $H_{1,\ldots,N}$
- Multitarget preimage: find X such that $\mathtt{HASH}(X) \in \{H_1, \ldots, H_N\}$
- 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.seed $\|$ ADRS
  - Result: Multitarget attacks blocked

$$H_1 \leftarrow \texttt{HASH}(P_1 \parallel M_1)$$
$$\dots$$
$$H_N \leftarrow \texttt{HASH}(P_N \parallel M_N)$$

  - ...because hash H_i always has only one valid prefix, P_i

What's a Signature

1. Public and Private keys:
   - $PK, SK \leftarrow$Generate()
   - Private key: Only I know this
   - Public key: I want everyone to know this
2. Signing:
   - Sign with private key SK
   - $\sigma \leftarrow$Sign(SK, M)
3. Verification:
   - Verify$(\sigma, M, PK) \rightarrow$ "good" or "bad"

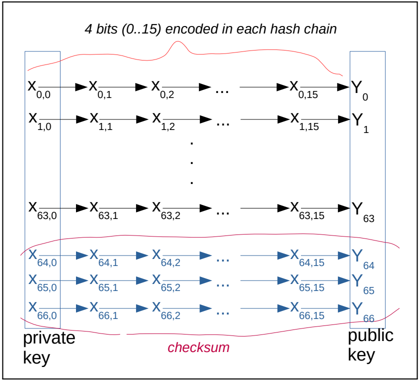
Normally, we can sign many messages with one key.

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

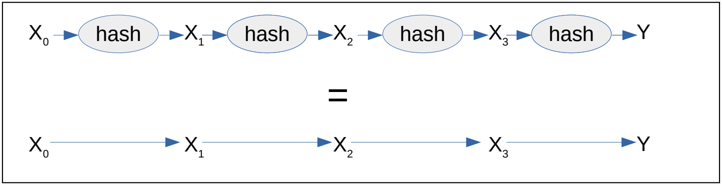
Winternitz/WOTS+ Signatures



- 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

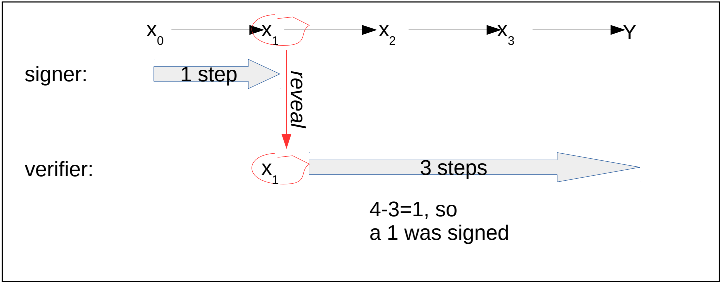


- Compute each element in chain by hashing previous element.

$$X_i = \texttt{HASH}(X_{i-1})$$

- Only need to know starting value–can compute all other values from there.
- Can't go backward in chain because of preimage resistance.

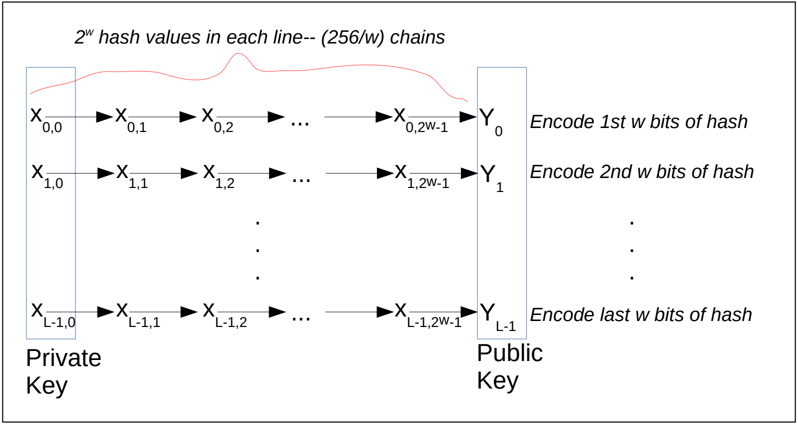
Winternitz: Signing 2 bits with one hash chain



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

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 $\lceil \frac{256}{w} \rceil$ *hash chains.*

Problem: Attacker can increment values

Arrrows represent hash operations!

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

- To sign value 01, we reveal x_1.
- Anyone can walk rest of chain: $y = \texttt{HASH}(\texttt{HASH}(\texttt{HASH}(x_1)))$
  - 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 HASH(message) as a sequence of a 4-bit digits, $t_{0,1,\ldots,a-1}$
- $\text{max} = a(2^w - 1)$
- $\text{checksum} \leftarrow \text{max} - \sum_{i=0}^{a} t_i$
- 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

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:

$$64 \times 15 = 960$$

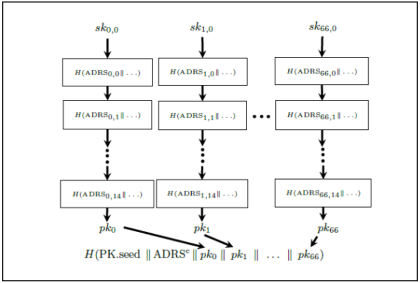
  - $0 \leq \text{Checksum} \leq 960$
    Need $\lg(\max +1)$ *bits!*
- Since each chain encodes 4 bits, we need three more chains to encode checksum.

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 $i = 0 \ldots 66$:
  - For $j = 1 \ldots 15$:
    - $X_{i,j} = \texttt{HASH}(\text{prefix}[i,j] \parallel X_{i,j-1})$
- Note: Each hash operation incorporates a unique prefix.

Making a WOTS+ Public Key in SPHINCS+ (2)

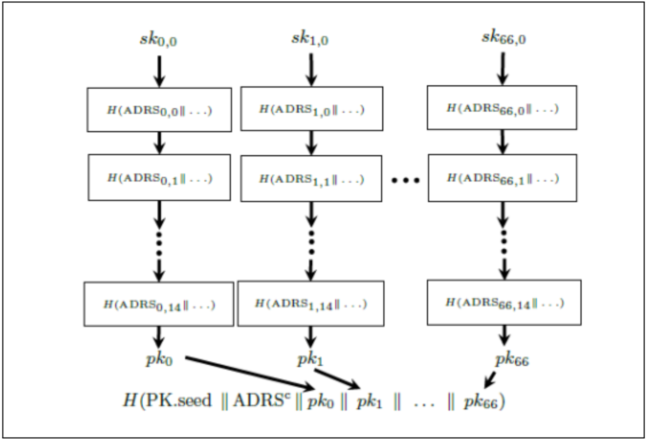


- Given final values in all 67 hash chains, $X_{0\ldots66,15}$
- Public key also includes a prefix for this particular one-time key ID
- Public key preimage =

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

- Public key = hash of public key preimage

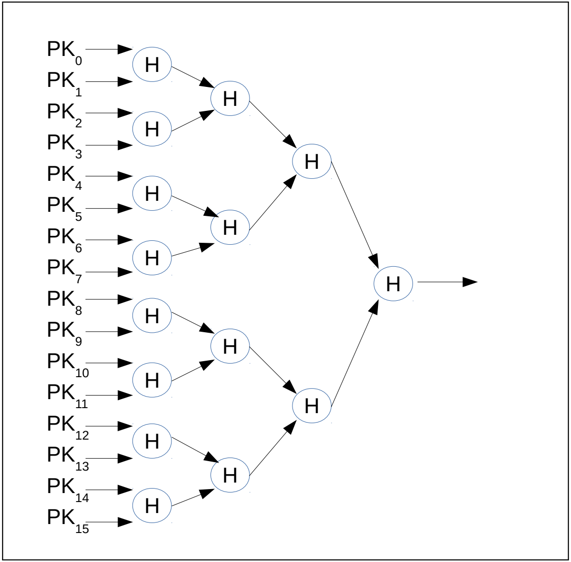
How it's done in SPHINCS+



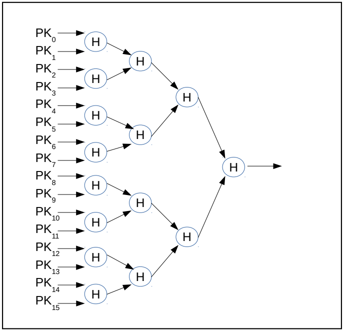
How do we use one-time keys?

- 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



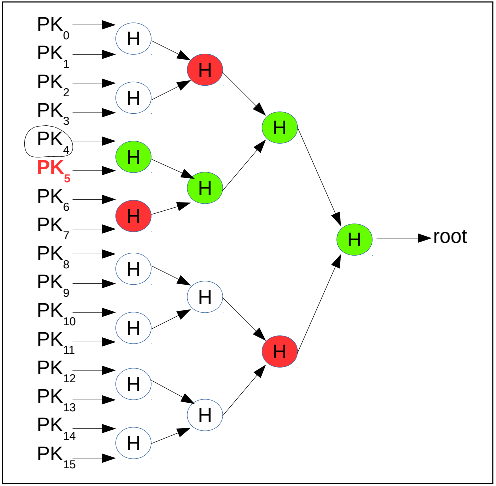
A binary tree made by hashing things together!



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

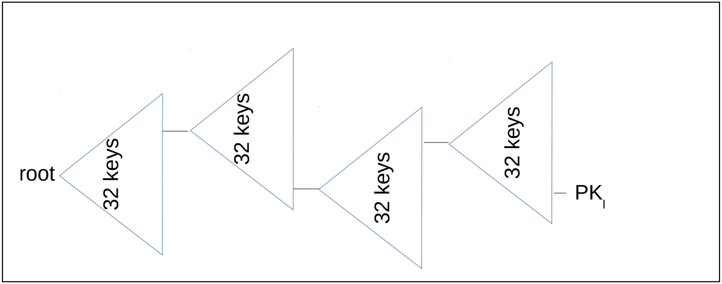
The root contains the hash of all the leaves.

Merkle Tree Path



- 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.

Hypertrees



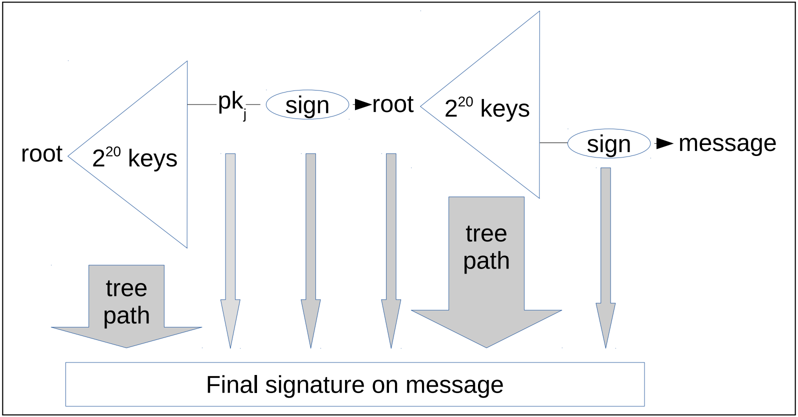
- A hypertree is a "tree of trees"
- Each tree is a Merkle tree full of one-time keys
- Each tree after the first is **generated on the fly as needed**
- Each tree has its root signed by a one-time key from the previous tree

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:

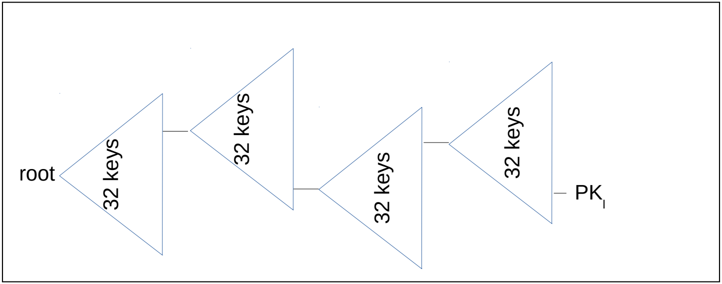
Tree of Trees...of Keys

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!*

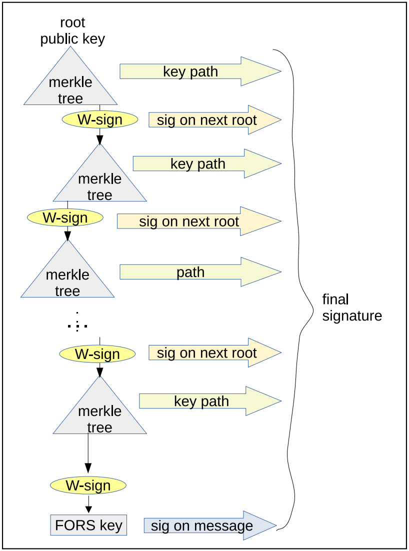
We can have *many* levels of trees



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

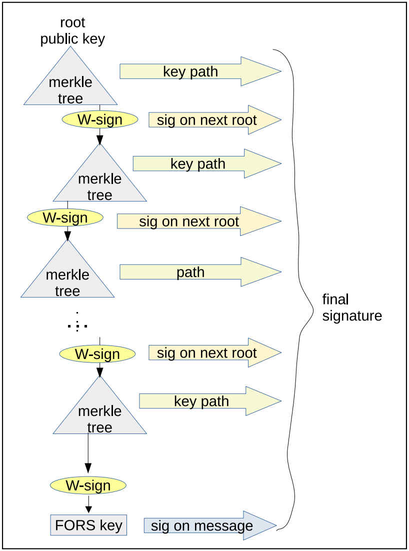
SPHINCS+ Structure



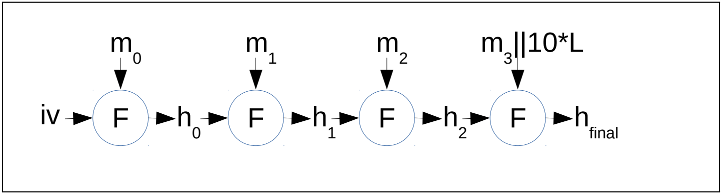
Structure of SPHINCS+ signatures

- 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

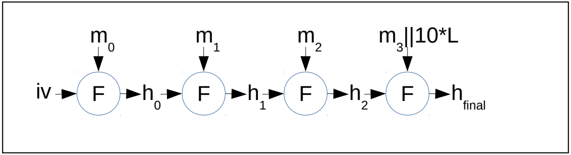
SPHINCS+ Structure

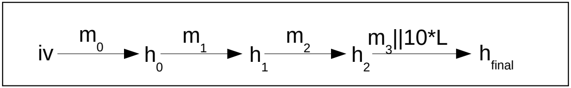


Part 2: Hash functions and Merkle-Damgård Hashes



Merkle-Damgård hashes: How SHA256 is Made

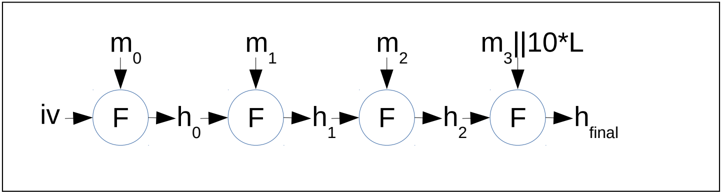




- Our result only applies to SPHINCS+ when it is using SHA256 to get 256-bit security
- Understanding it requires looking "under the hood" of SHA256

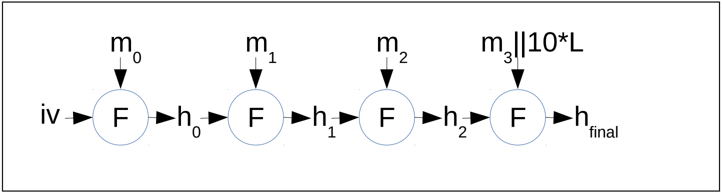
Merkle-Damgård Hashes (1)

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.

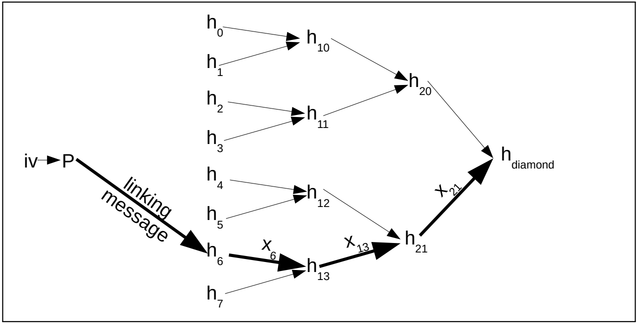
Merkle-Damgård Hashes: How SHA2 Works



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 $m_{0,1,2,\ldots,k-1}$.
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 $\mathtt{HASH}(M)$

Note: Only impact of $m_{0\ldots i}$ is on h_i

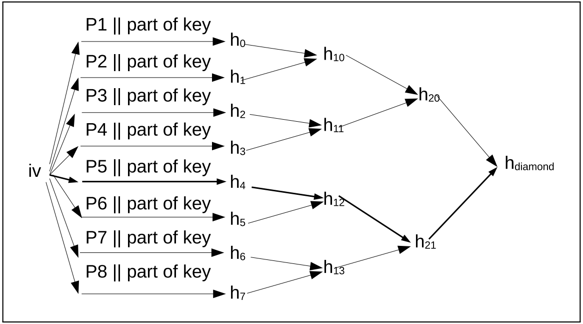
Herding Hash Functions



A problem

- I want to carry out a multitarget preimage attack
- My messages all start with different prefixes
- What can I do?

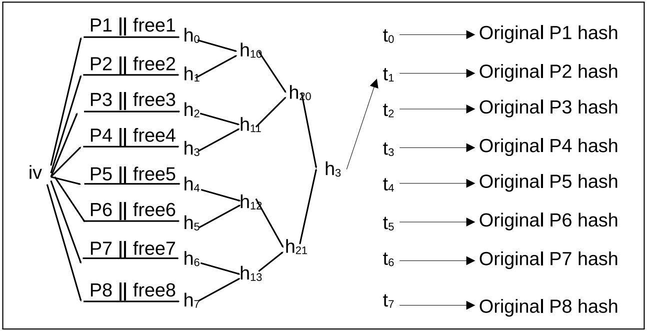
The Diamond Structure: A Merkle-Tree Computed by Finding Collisions.



- 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



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