Hash-based signatures II

Stateful and stateless signatures

Daniel J. Bernstein[1,2] and Tanja Lange[3]

[1]University of Illinois at Chicago

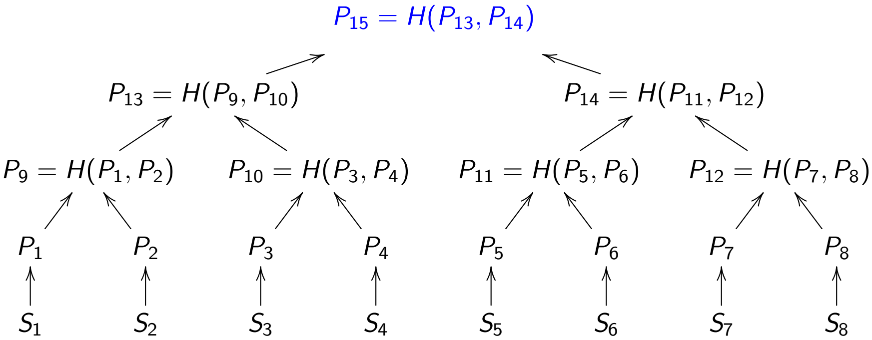
[2]Ruhr University Bochum

[3]Eindhoven University of Technology

14 May 2021

Merkle's (e.g.) 8-time signature system

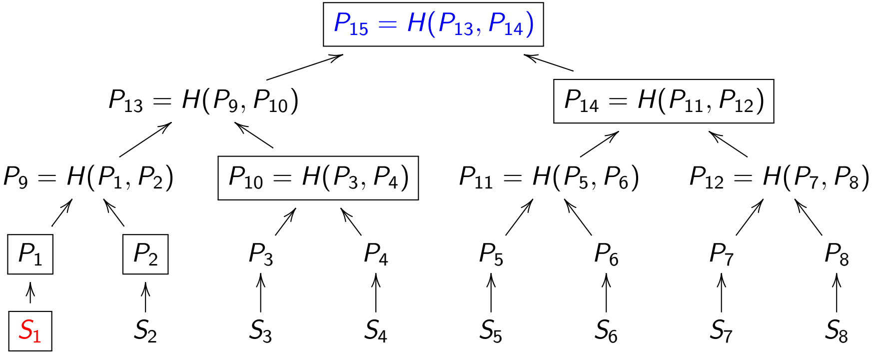
Hash 8 one-time public keys into a single Merkle public key P_{15}.



$S_i \to P_i$ can be Lamport or Winternitz one-time signature system.
Each such pair (S_i, P_i) may be used only once.

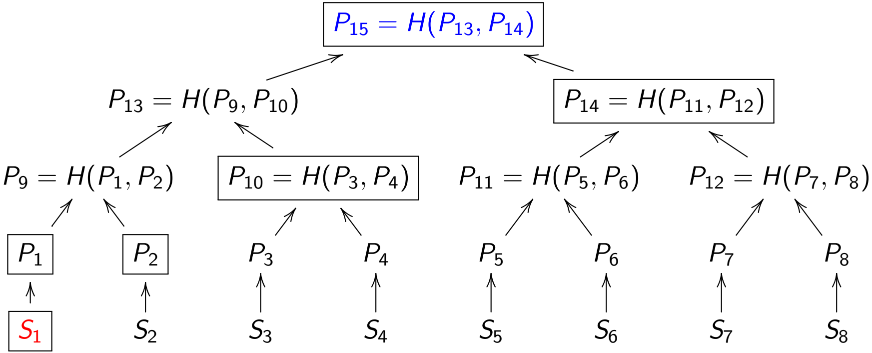
Signature in 8-time Merkle hash tree

Signature of first message: $(\texttt{sign}(m, S_1), P_1, P_2, P_{10}, P_{14})$.



Signature in 8-time Merkle hash tree

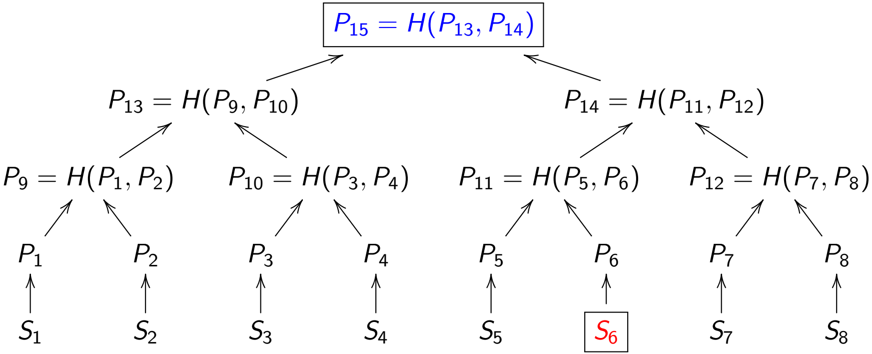
Signature of first message: $(\texttt{sign}(m, S_1), P_1, P_2, P_{10}, P_{14})$.



Verify signature $\texttt{sign}(m, S_1)$ with public key P_1 (provided in signature).
Link P_1 against public key P_{15} by computing $P'_9 = H(P_1, P_2)$, $P'_{13} = H(P'_9, P_{10})$, and comparing $H(P'_{13}, P_{14})$ with P_{15}.
Reject if $H(P'_{13}, P_{14}) \neq P_{15}$ of if the signature verification failed.

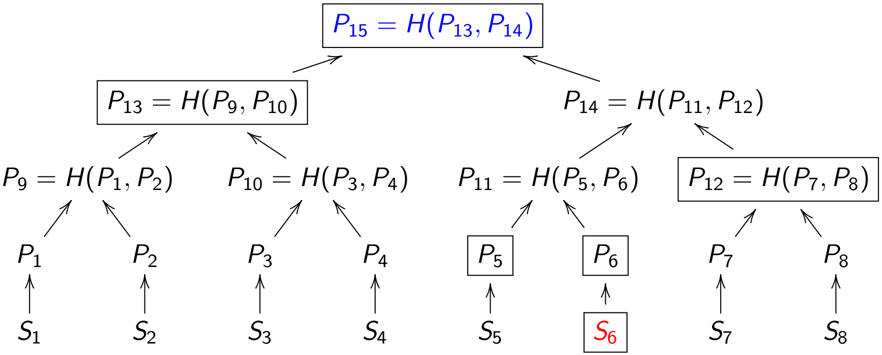
Signature in 8-time Merkle hash tree

Signature of sixth message:



Signature in 8-time Merkle hash tree

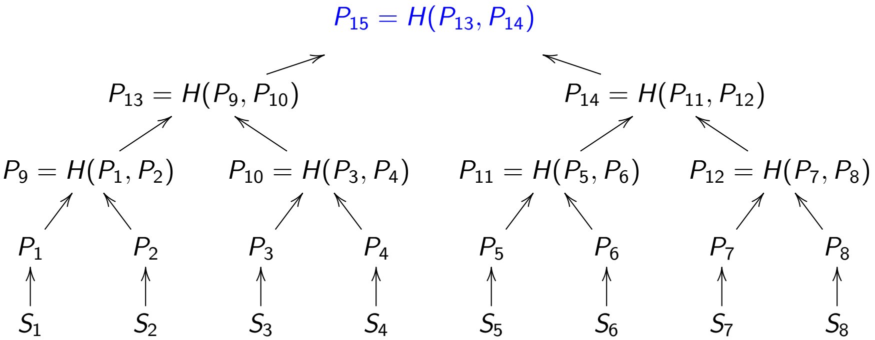
Signature of sixth message: $(\texttt{sign}(m', S_6), P_6, P_5, P_{12}, P_{13})$.



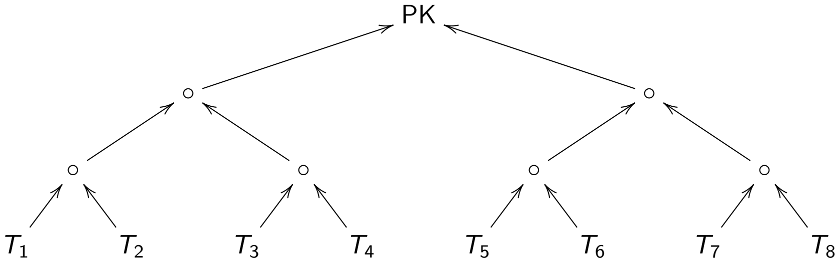
Improvements to Merkle's scheme

- Each public key (root of the tree) is good only for fixed number of messages, typically 2^n.
- The public key is very short: just one hash output.
  But each signature contains n public keys along with the one-time signature.
- Computing the public key requires computing and storing 2^n one-time signature keys.

Trees of Merkle trees

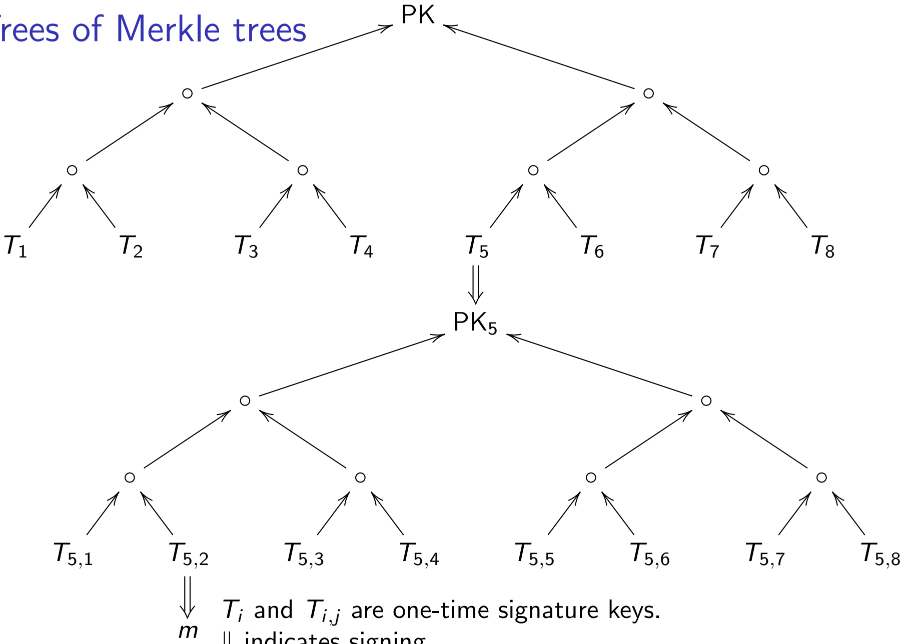


Trees of Merkle trees



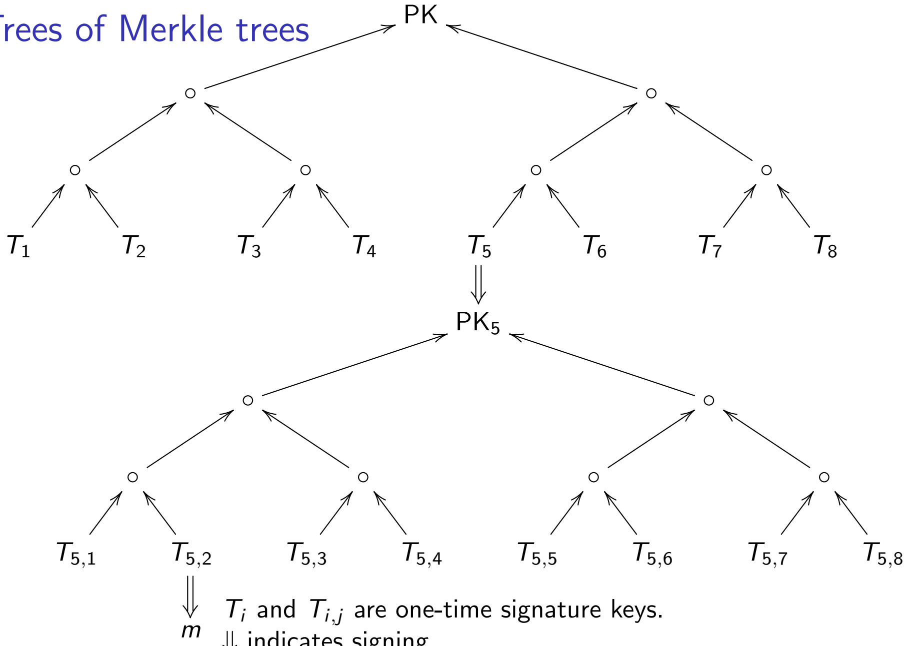
T_i are one-time signature keys.
$\uparrow$ indicates input to hash function.

Trees of Merkle trees



T_i and $T_{i,j}$ are one-time signature keys.
$\Downarrow$ indicates signing.

Trees of Merkle trees



T_i and $T_{i,j}$ are one-time signature keys.

$\Downarrow$ indicates signing.

No need to know PK_5 when generating the top tree.

Improvements to Merkle's scheme

- Each public key (root of the tree) is good only for fixed number of messages, typically 2^n.
- The public key is very short: just one hash output.
  But each signature contains n public keys along with the one-time signature.
- Computing the public key requires computing and storing 2^n one-time signature keys.
- Can trade time for space by computing the secret keys S_{ij} deterministically from a short secret seed and location (i, j).
  Very little storage for the seed but more time in signature generation.

Improvements to Merkle's scheme

- Each public key (root of the tree) is good only for fixed number of messages, typically 2^n.
- The public key is very short: just one hash output.
  But each signature contains n public keys along with the one-time signature.
- Computing the public key requires computing and storing 2^n one-time signature keys.
- Can trade time for space by computing the secret keys S_{ij} deterministically from a short secret seed and location (i, j).
  Very little storage for the seed but more time in signature generation.
- Building trees of trees increases the signature length (one extra one-time signature per tree) and signing time. See PhD thesis of Andreas Hülsing for an optimized schedule of what to store and when to precompute.
  Only the top tree is needed to generate the public key.

Stateful hash-based signatures

- Only one prerequisite: a good hash function, e.g. SHA3-512.
  Hash functions map long strings to fixed-length strings.
  Signature schemes use hash functions in handling plaintext.
- Old idea: 1979 Lamport one-time signatures.
- 1979 Merkle extends to more signatures.

Pros:

- Post quantum
- Only need secure hash function
- Security well understood
- Fast

Cons:

- Biggish signature
  though some tradeoffs possible
- Stateful, i.e., ever reusing a subkey breaks security.
  Adam Langley "for most environments it's a huge foot-cannon."

Stateful hash-based signatures

- Only one prerequisite: a good hash function, e.g. SHA3-512.
  Hash functions map long strings to fixed-length strings.
  Signature schemes use hash functions in handling plaintext.
- Old idea: 1979 Lamport one-time signatures.
- 1979 Merkle extends to more signatures.

Pros:

- Post quantum
- Only need secure hash function
- Security well understood
- Fast
- We can count: OS update, code signing, ...naturally keep state.

Cons:

- Biggish signature
  though some tradeoffs possible
- Stateful, i.e., ever reusing a subkey breaks security.
  Adam Langley "for most environments it's a huge foot-cannon."

Standardization progress

- CFRG has published 2 RFCs: RFC 8391 and RFC 8554

Datatracker Groups Documents Meetings Other User

```
Internet Research Task Force (IRTF)                          A. Huelsing
Request for Comments: 8391                                  TU Eindhoven
Category: Informational                                         D. Butin
ISSN: 2070-1721                                             TU Darmstadt
                                                               S. Gazdag
                                                              genua GmbH
                                                            J. Rijneveld
                                                     Radboud University
                                                             A. Mohaisen
                                        University of Central Florida
                                                                May 2018

              XMSS: eXtended Merkle Signature Scheme
```

Datatracker Groups Documents Meetings Other User

```
Internet Research Task Force (IRTF)                            D. McGrew
Request for Comments: 8554                                     M. Curcio
Category: Informational                                       S. Fluhrer
ISSN: 2070-1721                                            Cisco Systems
                                                              April 2019

                Leighton-Micali Hash-Based Signatures
```

Standardization progress

- CFRG has published 2 RFCs: RFC 8391 and RFC 8554
- NIST has standardized XMSS and LMS.
  Only concern is about statefulness in general.

Standardization progress

- CFRG has published 2 RFCs: RFC 8391 and RFC 8554
- NIST has standardized XMSS and LMS.
  Only concern is about statefulness in general.

- ISO SC27 JTC1 WG2 has started a study period on stateful hash-based signatures.

Towards stateless signatures

- These signatures are stateful, need to remember which leaf signature was used.

Towards stateless signatures

- These signatures are stateful, need to remember which leaf signature was used.
- Can we build trees so large that this is not a problem?

Towards stateless signatures

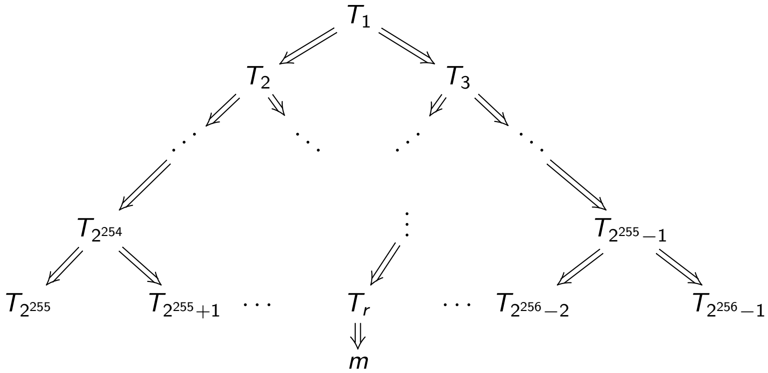
- These signatures are stateful, need to remember which leaf signature was used.
- Can we build trees so large that this is not a problem?
- By the birthday paradox we need 2^{256} leaves!

Towards stateless signatures

- These signatures are stateful, need to remember which leaf signature was used.
- Can we build trees so large that this is not a problem?
- By the birthday paradox we need 2^{256} leaves!
- Cannot precompute this tree . . .

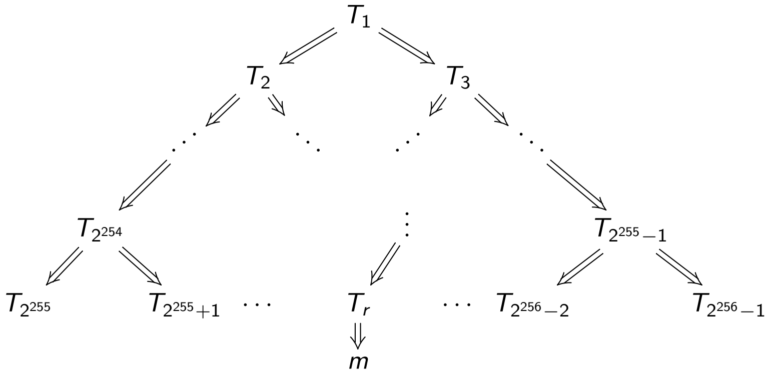
Huge trees (1987 Goldreich), keys on demand (Levin)

Signer chooses random $r \in \{2^{255}, 2^{255}+1, \ldots, 2^{256}-1\}$,
uses one-time public key T_r to sign message;
uses one-time public key T_i to sign (T_{2i}, T_{2i+1}) for $i < 2^{255}$.
Generates ith secret key deterministically as $H_k(i)$ where k is master secret. Important for efficiency



Huge trees (1987 Goldreich), keys on demand (Levin)

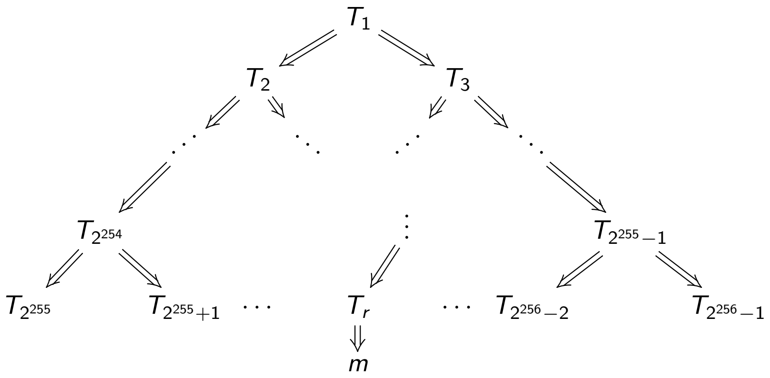
Signer chooses random $r \in \{2^{255}, 2^{255}+1, \ldots, 2^{256}-1\}$,
uses one-time public key T_r to sign message;
uses one-time public key T_i to sign (T_{2i}, T_{2i+1}) for $i < 2^{255}$.
Generates ith secret key deterministically as $H_k(i)$ where k is master secret. Important for efficiency



T_i for small i gets used repeatedly (each time an m falls in that sub-tree)

Huge trees (1987 Goldreich), keys on demand (Levin)

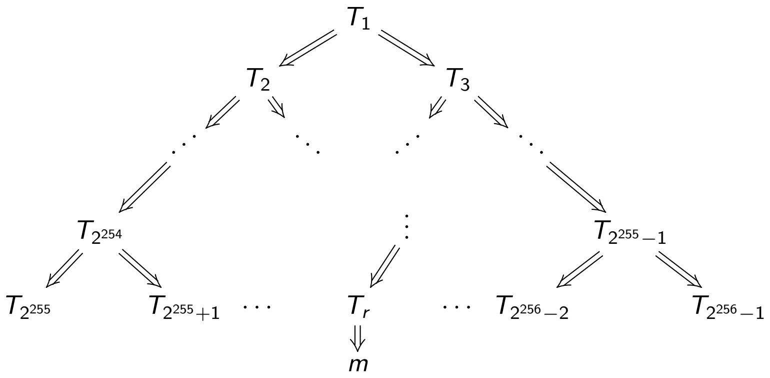
Signer chooses random $r \in \{2^{255}, 2^{255}+1, \ldots, 2^{256}-1\}$,
uses one-time public key T_r to sign message;
uses one-time public key T_i to sign (T_{2i}, T_{2i+1}) for $i < 2^{255}$.
Generates ith secret key deterministically as $H_k(i)$ where k is master secret. Important for efficiency



T_i for small i gets used repeatedly (each time an m falls in that sub-tree) but $H_k(i)$ being deterministic means it signs the same value, so no break.

Huge trees (1987 Goldreich), keys on demand (Levin)

Signer chooses random $r \in \{2^{255}, 2^{255}+1, \ldots, 2^{256}-1\}$,
uses one-time public key T_r to sign message;
uses one-time public key T_i to sign (T_{2i}, T_{2i+1}) for $i < 2^{255}$.
Generates ith secret key deterministically as $H_k(i)$ where k is master secret. Important for efficiency and security.



T_i for small i gets used repeatedly (each time an m falls in that sub-tree) but $H_k(i)$ being deterministic means it signs the same value, so no break.

Use Goldreich to create stateless hash-based signatures

0.6 MB for hash-based Goldreich signature using
short-public-key Winternitz-16 one-time signatures.

Would dominate traffic in typical applications,
and add user-visible latency on typical network connections.

Use Goldreich to create stateless hash-based signatures

0.6 MB for hash-based Goldreich signature using
short-public-key Winternitz-16 one-time signatures.

Would dominate traffic in typical applications,
and add user-visible latency on typical network connections.

Example:
Debian operating system is designed for frequent upgrades.
At least one new signature for each upgrade.
Typical upgrade: one package or just a few packages.
1.2 MB average package size.
0.08 MB median package size.

Use Goldreich to create stateless hash-based signatures

0.6 MB for hash-based Goldreich signature using
short-public-key Winternitz-16 one-time signatures.

Would dominate traffic in typical applications,
and add user-visible latency on typical network connections.

Example:
Debian operating system is designed for frequent upgrades.
At least one new signature for each upgrade.
Typical upgrade: one package or just a few packages.
1.2 MB average package size.
0.08 MB median package size.

Example:
HTTPS typically sends multiple signatures per page.
1.8 MB average web page in Alexa Top 1000000.

Can we do with fewer leaves?

Reason for the large number: must never hit same leaf twice.

Can we do with fewer leaves?

Reason for the large number: must never hit same leaf twice.

Leaf is chosen by hash function $H(m)$.

Can we do with fewer leaves?

Reason for the large number: must never hit same leaf twice.

Leaf is chosen by hash function $H(m)$.

Change definition of H to have many components

$$H(m) = (h_0, h_1, \ldots, h_{k-1}),$$

where each $h_i \in \{0, 1, 2, \ldots, t-1\}$ for some t.
Collisions mean that all h_i match.

Can we do with fewer leaves?

Reason for the large number: must never hit same leaf twice.

Leaf is chosen by hash function $H(m)$.

Change definition of H to have many components

$$H(m) = (h_0, h_1, \ldots, h_{k-1}),$$

where each $h_i \in \{0, 1, 2, \ldots, t-1\}$ for some t.
Collisions mean that all h_i match.

r-subset resilience

Let $H(m_j) = (h_{j,0}, h_{j,1}, \ldots, h_{j,k-1})$.
H is r-subset-resilient if given $H(m_1), H(m_2), \ldots, H(m_r)$ the probability of finding m' with $H(m') = (h'_0, h'_1, \ldots, h'_{k-1})$ and $h_f \in \{h_{j,i} | 0 \leq i < k, 1 \leq j \leq r\}$ for $0 \leq f < k$ is negligible.

Can we do with fewer leaves?

Reason for the large number: must never hit same leaf twice.

Leaf is chosen by hash function $H(m)$.

Change definition of H to have many components

$$H(m) = (h_0, h_1, \ldots, h_{k-1}),$$

where each $h_i \in \{0, 1, 2, \ldots, t-1\}$ for some t.
Collisions mean that all h_i match.

r-subset resilience

Let $H(m_j) = (h_{j,0}, h_{j,1}, \ldots, h_{j,k-1})$.
H is r-subset-resilient if given $H(m_1), H(m_2), \ldots, H(m_r)$
the probability of finding m' with $H(m') = (h'_0, h'_1, \ldots, h'_{k-1})$ and
$h_f \in \{h_{j,i} | 0 \leq i < k, 1 \leq j \leq r\}$ for $0 \leq f < k$ is negligible.

The same leaf public key can be used for $r + 1$ signatures
if H if r-subset-resilient.

Few-times signature HORS

(Hash to Obtain Random Subset)

General parameters:

- Integer parameters k, t, ℓ.
- Hash function $H : \{0,1\}^* \to \{0,1\}^{k \cdot \log_2 t}$.
- One-way function $f : \{0,1\}^\ell \to \{0,1\}^\ell$.

KeyGen:

- Picks t strings $s_i \in \{0,1\}^\ell$, compute $v_i = f(s_i)$ for $0 \le i < t$.
- Public key $P = (v_0, v_1, \ldots, v_{t-1})$; secret key $S = (s_0, s_1, \ldots, s_{t-1})$.

Sign $m \in \{0,1\}^*$:

- Compute $H(m) = (h_0, h_1, \ldots, h_{k-1})$, where each $h_i \in \{0, 1, 2, \ldots, t-1\}$.
- Signature on m is $\sigma = (s_{h_0}, s_{h_1}, s_{h_2}, \ldots, s_{h_{k-1}})$.

Verify:

- Compute $H(m) = (h_0, h_1, \ldots, h_{k-1})$ and $(f(s_{h_0}), f(s_{h_1}), f(s_{h_2}), \ldots, f(s_{h_{k-1}}))$.
- Verify that $f(s_{h_i}) = v_{h_i}$ for $0 \le i < t$.

HORS exercises, assume H is surjective

1. Let $\ell = 80$, $t = 2^5$, and $k = 3$. How large (in bits) are the public and secret keys? How large is a signature? How many different signatures can the signer generate for a fixed key pair as $H(m)$ varies? Ignore that s-values could collide.
2. The same public key can be used for $r + 1$ signatures if H is r-subset-resilient.
   Even for $r = 1$, i.e. after seeing just one typical signature, an attacker has an advantage at creating a fake signature. What are the options beyond exact collisions in H?
3. Let $\ell = 80$, $t = 2^5$, and $k = 3$. Let m be a message so that $H(m) = (h_0, h_1, h_2)$ satisfies that $h_i \neq h_j$ for $i \neq j$. You get to specify messages that Alice signs. You may not ask Alice to sign m. State the smallest number of HORS signatures you need to request from Alice in order to construct a signature on m? How many calls to H does this require on average? You should assume that H and f do not have additional weaknesses beyond having too small parameters. Explain how you could use under 1000 evaluations of H if you are allowed to ask for two signatures.

Ingredients of SPHINCS (and SPHINCS-256)

Drastically reduce tree height (to 60).

Replace one-time leaves with few-time leaves.

Optimize few-time signature size *plus* key size.
New few-time HORST (HORS with trees),
improving upon HORS.

Use hyper-trees (12 layers), as in GMSS.

Use masks, as in XMSS and XMSS^{MT},
for standard-model security proofs.

Optimize short-input (256-bit) hashing speed.
Use sponge hash (with ChaCha12 permutation).

Use fast stream cipher (again ChaCha12).

Vectorize hash software and cipher software.

See paper for details: `sphincs.cr.yp.to`

Updated version is NIST submission SPHINCS+
`https://sphincs.org/`.