Selected Areas in Cryptology
Cryptanalysis
Week 6

Marc Stevens

stevens@cwi.nl

https://homepages.cwi.nl/~stevens/mastermath/

Asymmetric from symmetric cryptography

- Can we build asymmetric cryptography from symmetric cryptography?

- Benefits:
  - Symmetric cryptography seems generally to resist quantum cryptanalysis
  - No number-theoretic assumptions needed

- This week:
  - Hash-based signatures (continued): making schemes more practical
  - MPC-in-the-head on symmetric cryptography

Summary last week

- Lamport 1-bit and k-bit message OTS
  - $2k$ pre-images as private key, reveal k pre-images selectively based on message
  - Hash function needs to be Pre-secure and UD-secure
- Winternitz(+) OTS
  - Use hashchains to trade-off speed for size by signing multiple bits at once
  - Use extra checksum hashchains to prevent trivial manipulation
  - WOTS+: Hash function needs to be Pre-, Sec-, and UD-secure
- MerkleTree
  - Compact composite public key for many OTS public keys
  - Each signature includes membership proof for used OTS public key
- Trees of Trees
  - Tree of MerkleTrees, Parent Tree sign public key of Child Trees
- These are all Stateful: need to keep track of state or break security!

Real World Schemes

- Stateful HBS: need to be really careful maintaining state!
  - XMSS: Based on MerkleTree using WOTS+ (NIST standard)
  - XMSS-MT: Based on Tree of XMSS (NIST standard)
  - LMS: Based on MerkleTree using WOTS (NIST standard)
  - HSS: Based on Tree of LMS (NIST standard)
  - Note these have various tweaks including:
    - Extra prefix/suffix/tweak per hash call to avoid various attacks (tweak=alter function instead of more input)
    - [BDS08] algorithm to maintain internal state of current path to prevent signature calls with a lot of update work
  - Let's cover important improvements!

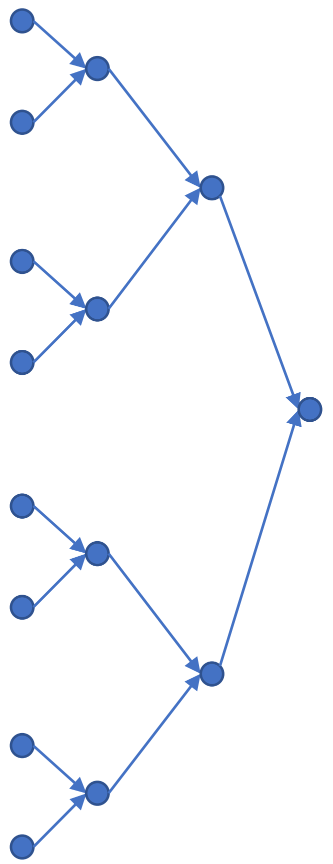
Merkle Tree Signature Time

- Key generation: can compute all leaf pk_i from 1 $seed$
- Hence, private key is simply $seed, counter$
- To generate i-th signature we need all nodes for i-th path
  - No nodes stored ⇒ need to compute *all* leaf pk_i again
  - Note that node on height v is needed for 2^v consecutive sigs

- Trick 1: store authentication path (h nodes) & reuse
  - On average h leaf pk_i need to be computed
  - But worst case is switching from left-half to right-half:
    2^h leaf pk_i need to be computed!

- Trick 2: store 2^k nodes for top k levels
  - Worst case is now only 2^{h-k} leaf pk_i to be computed

- Trick 3: [BDS08] distribute computation of future needed nodes
  - Extra storage: $\sim (3.5\ h + 2^k)$ hashes
  - Per signature: $\leq \big((h-k)/2 + 1\big)$ leaf pk_i to be computed



Hash-then-sign

- Transform signature scheme S for k-bit messages into scheme S' for arbitrary length messages
  - $S'.\mathrm{sign}(sk, m) = S.\mathrm{sign}\big(sk, f(m)\big)$

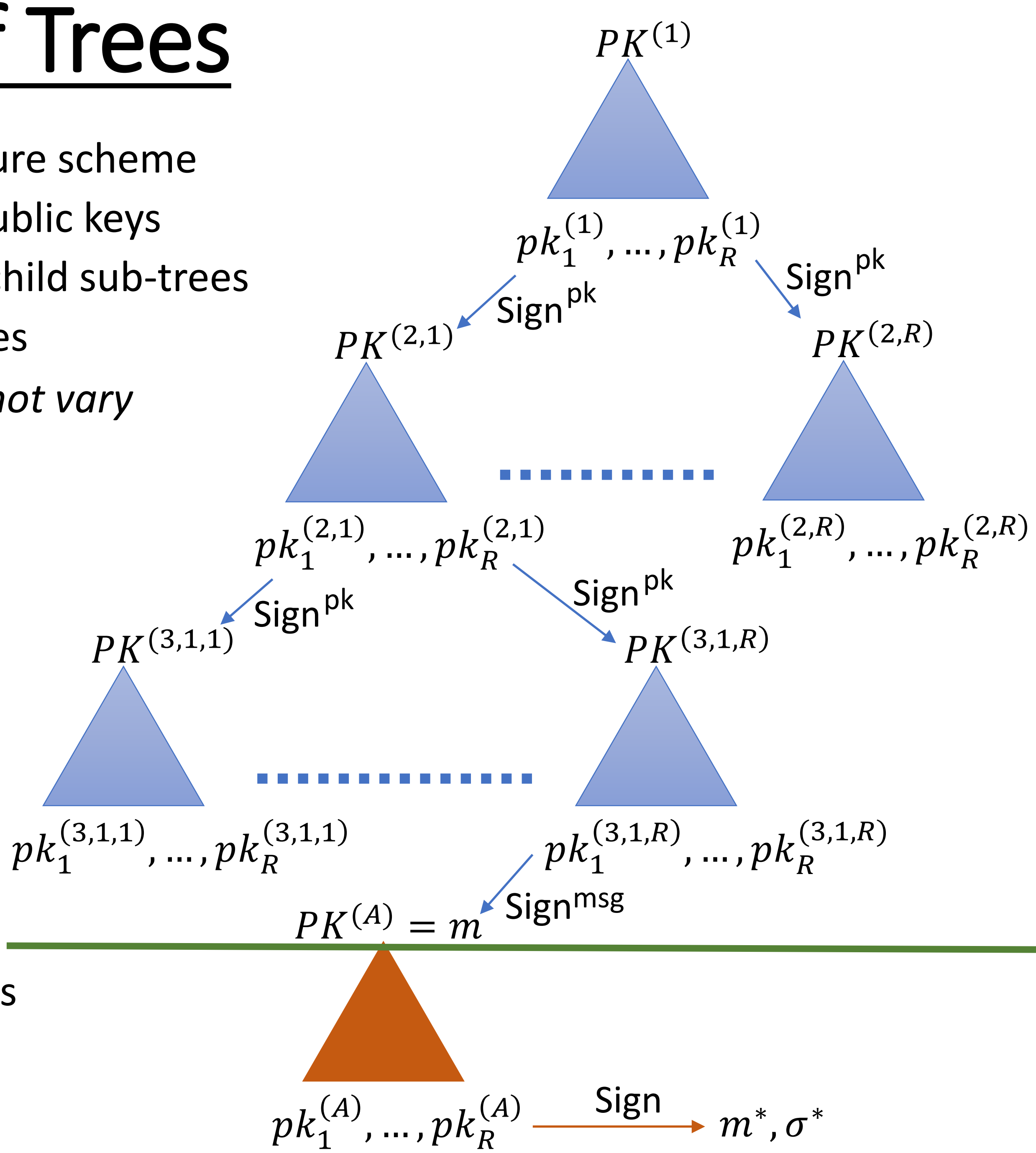
- A signature forgery (m^*, σ^*) for S' implies
  - Either hash collision $f(m^*) = f(m),\ \sigma^* = \sigma_m$ for a message $m \neq m^*$ that has been signed
  - Otherwise, if $f(m^*)$ hasn't been signed by S before then this must be a forgery for S

- And indeed, an attacker finding a hash collision $f(m^*) = f(m)$ directly results in a forgery
  - Requesting a signature σ_m for $m \Rightarrow (m^*, \sigma_m)$ is a valid forgery
  - Actually used in real world: Rogue Certificate Authority [<u>S</u>SA+09], [F<u>S</u>15]

- A better way:
  - $S'.\mathrm{sign}(sk, m) = r|\sigma$, where $r \leftarrow \{0,1\}^n$, $\sigma \leftarrow S.\mathrm{sign}(sk, f(pk|r|m)$
  - To use a hash collision, the attacker first needs to guess r correctly
  - Also prevents brute-force multi-user attacks:
a second pre-image guess $f(pk'|r'|m')$ needs to match $pk' = pk$ for a specific user

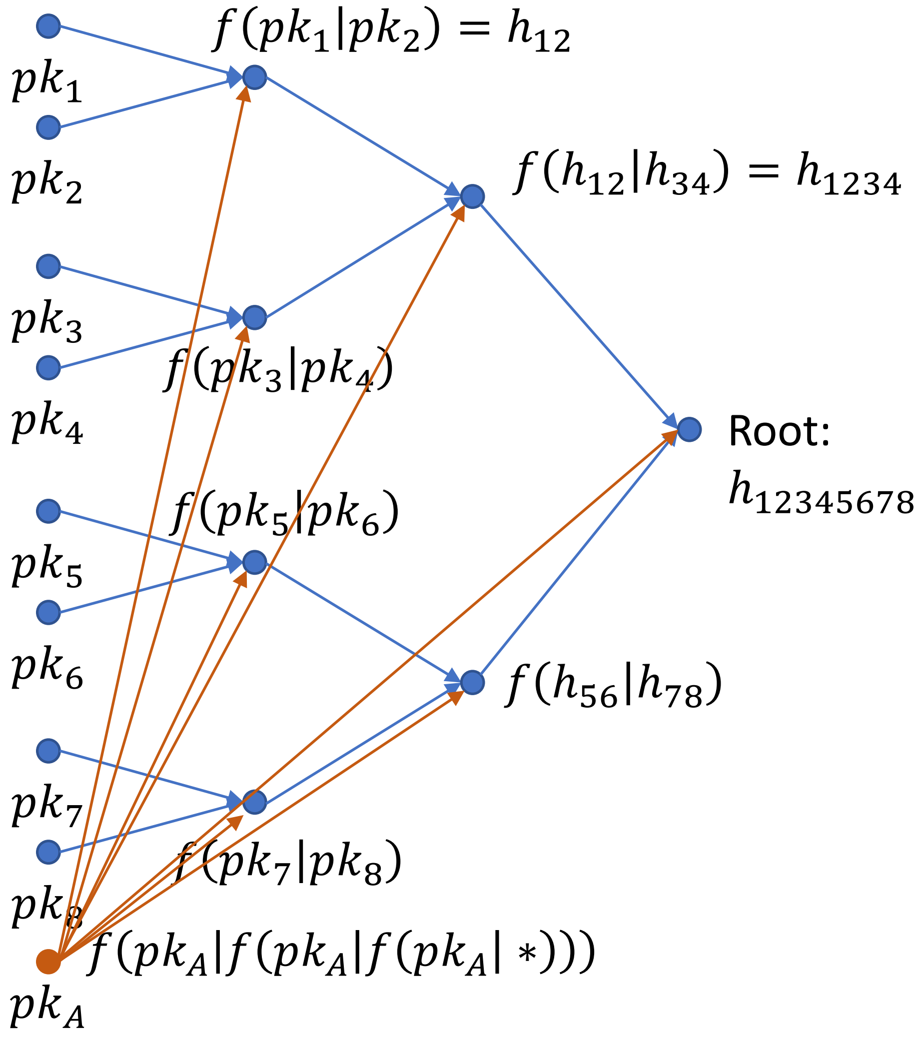
Trees of Trees

- Composite signature scheme of composite signature scheme
  - Each sub-tree is 1 Merkle Tree of R WOTS+ public keys
  - Sub-trees are used to sign root public key of child sub-trees
  - Tree Depth $D \Rightarrow R^D$ total amount of signatures
  - *Deterministic sub-trees to ensure $PK^{(i,j)}$ cannot vary*
- An attack on this composite scheme implies
  - either a WOTS+ signature forgery (incl hash-then-sign)
  - and/or a MerkleTree membership forgery
  - Or does it?...
- Attack can confuse verifier by extending tree by having his own WOTS+ public key signed
- $\Rightarrow$ strengthen scheme by using hash-then-sign with different prefixes for signing keys vs messages
- Very similar to Certificate signing



Merkle Tree

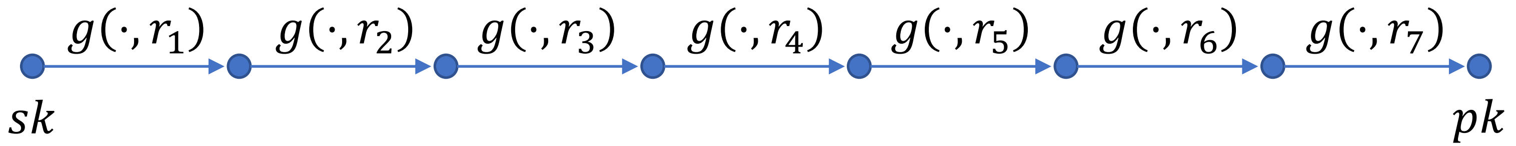
- *Composite* signature scheme using e.g. WOTS+-256-2^8
- An attack on this composite scheme implies
  - either a WOTS+ signature forgery (including hash-then-sign)
  - and/or a second pre-image (using WOTS+ trick)
- However, an attacker has many Sec/Pre targets
  - With carefully crafted chain
  - Can target any hash value in MerkleTree: # targets $T = R - 1$
  - $\Rightarrow$ Attack cost $\sim 2^n/T$
  - Number of targets T for Trees of Trees even larger!
  - Multi-user: obtain even more targets
- Reduce multi-user/multi-target attacks
  - Use different prefix/suffix/tweak for each:
    - MerkleTree node
    - Subtree index in Trees-of-Trees
    - User (add chosen random value to top level public key)



WOTS+ Random Bitstrings

- WOTS+ requires $w - 1$ random bitstrings in public key

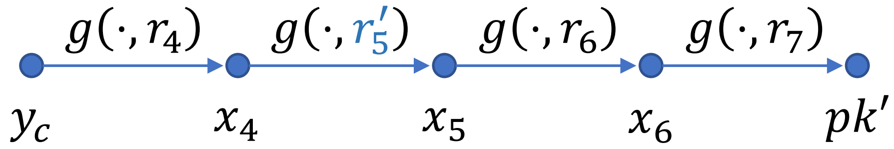
$$g(x, r) = f(x \oplus r)$$



- Can we use less random bitstrings? Say only 1?
- No!
  - If $r_1 = \cdots = r_7 = r$ then changing r_5 implies changing $r_1, \ldots, r_4$ and thus x_4
  - We cannot efficiently embed second pre-image challenge x_c anymore

Embedding second pre-image challenge x_c

$$r_5' = x_4 \oplus x_c \;\Rightarrow\; x_5 = f(x_4 \oplus r_5') = f(x_c)$$



- But! We can reuse random bitstrings for all WOTS+ instances in composite scheme
  - Reduction proof now requires guessing if there's a forgery for which WOTS+ instance it will be
  - (But number of WOTS+ instances is polynomial in λ, so still only polynomial factor loss extra)
  - $\Rightarrow$ Only need to give 1 sequence of random bitstrings in top composite scheme's public key
  - $\Rightarrow$ Reduces signature size
  - Note: random bitstrings can also be reused for MerkleTree to get Coll→Sec

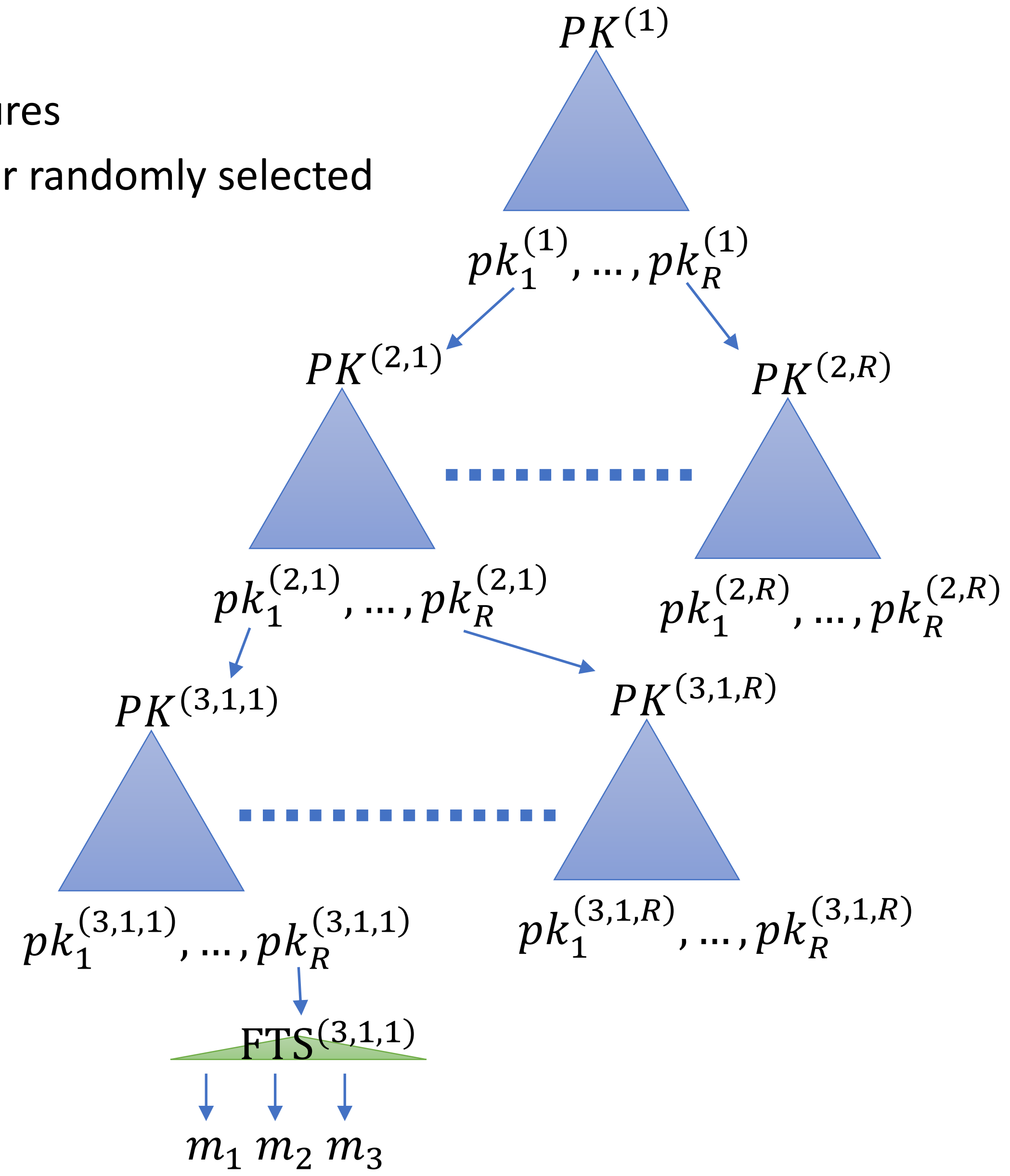
State Footcannon!

- Stateful HBS: need to be really careful maintaining state!
- What can go wrong?
  - Programming errors
  - Hardware failures (crash / write error) causing fail to record that a key is used
  - Virtual Machine cloning:
    - Now 2 VM's are set to sign using the same key
    - But possibly different messages!
  - Active attacks changing state, e.g. computer hack, or physical attack against smartcard
- For federal use, NIST has strict rules to prevent any procedural fault that leads to reusing same leaf key

- Can we also build stateless HBS?

Goldreich's stateless HBS

- Goldreich's stateless HBS:
  - HBS scheme with very large $2^{2\lambda}$ number of possible signatures
  - For each signature, index $i \leftarrow \{1, \dots, 2^{2\lambda}\}$ is message hash or randomly selected
  - Expected amount of signatures before a collision occurs:
    - $\frac{\sqrt{\pi}}{2} 2^{\lambda}$ signatures $\Rightarrow$ λ-bit security against key reuse
  - Original construction is binary tree of OTS
    - $\Rightarrow$ signature size > 1MiB
- SPHINCS:
  - Use deterministic virtual Tree of Trees with WOTS+
  - Leaf HBS are instead *few-time HBS*: HORST
  - $\Rightarrow$ Only need OTS T-o-T for 2^{60} signatures instead of 2^{256}
  - Sizes: PK / SK / SIG: ~ 1KiB / 1KiB / 40KiB
- SPHINCS+:
  - Each hash function call has different tweak & bitmask
  - Replaced HORST → FORS
  - SPHINCS+-128s-robust (NIST level 1)
    - Sizes: PK / SK / SIG : ~ 64B / 32B / 7.7KiB



HORS

- HORS is a few-time HBS
  - Secret key: set of 2^a secret values $\{sk_1, \dots, sk_{2^a}\}$
  - Public key: hash outputs of secret key $\{f(sk_1), \dots, f(sk_a)\}$
  - Signing:
    - Split n-bit hash $f(r|m)$ into coefficients $c_1, \dots, c_t$ of a bits, where $a \cdot t = n$
    - Reveal indexed secret values: $\sigma_m = (r, \sigma_1, \dots, \sigma_t) = \left(r, sk_{c_1}, \dots, sk_{c_t}\right)$
    - Note that indices might not be different: just reveal the same value again
  - Verifier:
    - Split k-bit message into coefficients $c_1, \dots, c_t$ of a bits
    - Verify $f(\sigma_i) =? \, pk_{c_i}$ for all $i = 1, \dots, t$
  - Security reduces to
    - Sec + Pre + UD: can program $pk_j = y_c$ or $sk_j = x_c$ (and abort if $j \in \{c_1, \dots, c_t\}$)
    - Finding a m^* for which the signature components have all been revealed by queries
      - i.e.: $\{c_1^*, \dots, c_t^*\} \subset \bigcup_{m \text{ queried}} \{c_i \,|(c_1, \dots, c_t) \leftarrow \text{split}(m)\}$

HORS

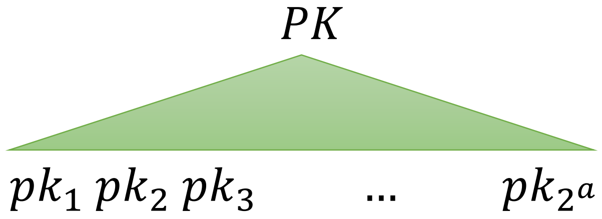
- For example:
  - Parameters: $n = 256, a = 16, t = 16$
    - Secret key: $2^a = 65536$ values of n bits (can all be generated from 1 seed)
    - Public key: $2^a = 65536$ hash values of n bits (in total: 16 MiB !)
    - Signature: t values of n bits
  - Consider that the adversary has queried 4 signatures
    - $\Rightarrow$ a fraction $\frac{4 \cdot t}{65536} = \frac{1}{1024} = 2^{-10}$ of secret values are public
    - Assuming outputs of f behave as random bitstrings
      - $\Rightarrow \Pr[sk_{c_1^*}, \dots, sk_{c_t^*} \text{ are public}] \leq \left(2^{-10}\right)^{16} = 2^{-160}$, for $(c_1^*, \dots, c_t^*) \leftarrow \text{split}(m^*)$
  - Security decreases with # signatures:
  - Note that due to r, adversary cannot predict which sk_i are revealed each query

# Signatures	Probability of all $sk\_(c_i^*)$ being public $\leq$
1	$\left(16/2^{16}\right)^{16} = 2^{-192}$
2	$\left(32/2^{16}\right)^{16} = 2^{-176}$
4	$\left(64/2^{16}\right)^{16} = 2^{-144}$
8	$\left(128/2^{16}\right)^{16} = 2^{-128}$
16	$\left(256/2^{16}\right)^{16} = 2^{-112}$

HORST

- HORS public key is 16 MiB for $a = t = 16$ and $n = 256$
- Can we do better?

- HORST = HORS with Trees
  - Use MerkleTree for public key values
  - HORST public key is MerkleTree root hash value: 256 bits



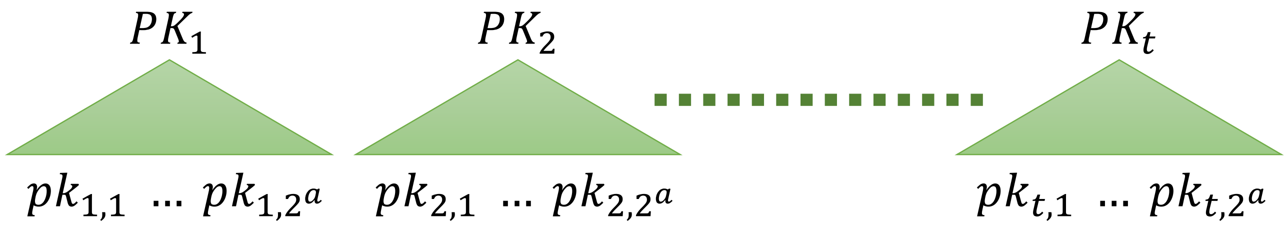
  - Signature increases with membership proofs of revealed values
    - $t \cdot n$ bits for revealing secret values
    - ~ $t \cdot a \cdot n$ bits for membership proofs
    - Example: ~ 8 KiB (can be made smaller with more optimizations)

  - Verify signature:
    - Verify pre-images
    - Verify membership proofs to root hash
    - Verify indices c_i with position of pk_{c_i} in tree!

FORS

- SPHINCS+ is improvement of SPHINCS that replaces HORST by FORS
- Variant on HORST with added security
  - Avoid that coefficients with same value $c_i = c_j$ reveal the same secret value
  - Idea: use HORST scheme for each coefficient independently

$$PK = f(PK_1|PK_2| \dots |PK_t)$$



  - New public key is still single hash value: hash of concatenation of the t root hashes
  - Membership proof variant:
    - MerkleTree membership proof contains index & the values to reveal to be able to compute root
    - Instead of verifying individual roots $PK_i' =? PK_i$,
    - FORS verifies all recomputed roots together:
      - $f(PK_1'|PK_2'| \dots PK_t') =? PK$
    - $\Rightarrow$ no extra overhead in publishing PK_i in public key or signature

Real World Schemes

- Stateful HBS: need to be really careful maintaining state!
  - XMSS: Based on MerkleTree using WOTS+ (NIST standard)
  - XMSS-MT: Based on Tree of XMSS (NIST standard)
  - LMS: Based on MerkleTree using WOTS (NIST standard)
  - HSS: Based on Tree of LMS (NIST standard)
  - Note these have various tweaks including:
    - Extra prefix/suffix/tweak per hash call to avoid various attacks (tweak=alter function instead of more input)
    - Optimized TreeHash algorithm to maintain internal state of current path to prevent signature calls with a lot of update work

- Stateless HBS: avoid keeping track of state by enabling random paths!
  - SPHINCS+: Based on Trees of MerkleTrees of WOTS+ (NIST standard)
    - Uses FORS instead of WOTS+ at leaf MerkleTree
    - FORS is a few-time signature scheme (FTS)
    - Number of potential signatures is so large, one can randomly choose path to a FTS instance
    - Even with many signatures, the probability a FTS instance is used too often is negligible

Summary

- MerkleTree signature time improvements
  - Storing extra nodes & distribute computation of future needed nodes
- Security improvements
  - Hash-then-sign: unpredictable message hash with signer's randomness
  - Trees-of-trees: separation between signing subtree vs message
  - Multi-target/user attacks: specialize every hash function call
  - WOTS+: can reuse randomness in MerkleTree/Trees-of-trees
- Stateless HBS
  - Goldreich: HBS with $\geq 2^{2\lambda}$ signatures $\Rightarrow 2^{\lambda}$ signatures at λ-bit security
  - Few-time HBS schemes: HORS, HORST, FORS
  - SPHINCS: Trees-of-trees with WOTS+, and HORST as leaf FTS
  - SPHINCS+: improved SPHINCS with FORS, NIST standard