SYNOPSYS®

Stateful hash-based signatures

From theory to practice

Dr. Nimisha Limaye, Staff Engineer

September 4, 2024

Outline

- Basic of digital signatures, effect of quantum computers, next steps
- Stateful Hash-based signatures
- Leighton Micali Signature (LMS) scheme
- eXtended Merkle Signature Scheme (XMSS)
- Summary
- Synopsys products

Hash function

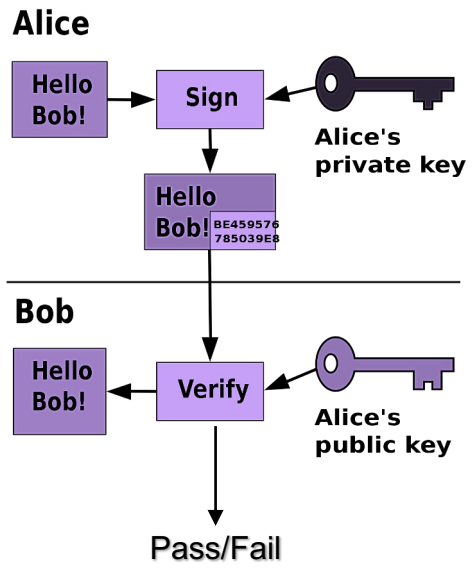
Preliminaries

- Converts long binary strings (message) to fixed-length output strings (digest)
  - $\mathrm{H}: \{0,1\}^* \rightarrow \{0,1\}^n$
- Cryptographic hash function ($x \rightarrow H(x) = y$):
  - Preimage resistance: hash function $H()$ is one-way; cannot deduce x given $H(x)$
  - 2nd preimage resistance: cannot find $x' \neq x$ such that $H(x) = H(x')$, given x and $H(x)$
  - Collision resistance: cannot find $\mathrm{x}_1 \neq x_2$ such that $H(x_1) = H(x_2)$
  - Examples:
    - SHA-2, SHA-3 and its XOF variants
- Applications:
  - Digital signatures: a cryptographic hash function increases the security and efficiency of a digital signature scheme when the digest is digitally signed instead of the message itself
  - Pseudorandom bit generation, message authentication codes, and key derivation functions

Digital Signatures

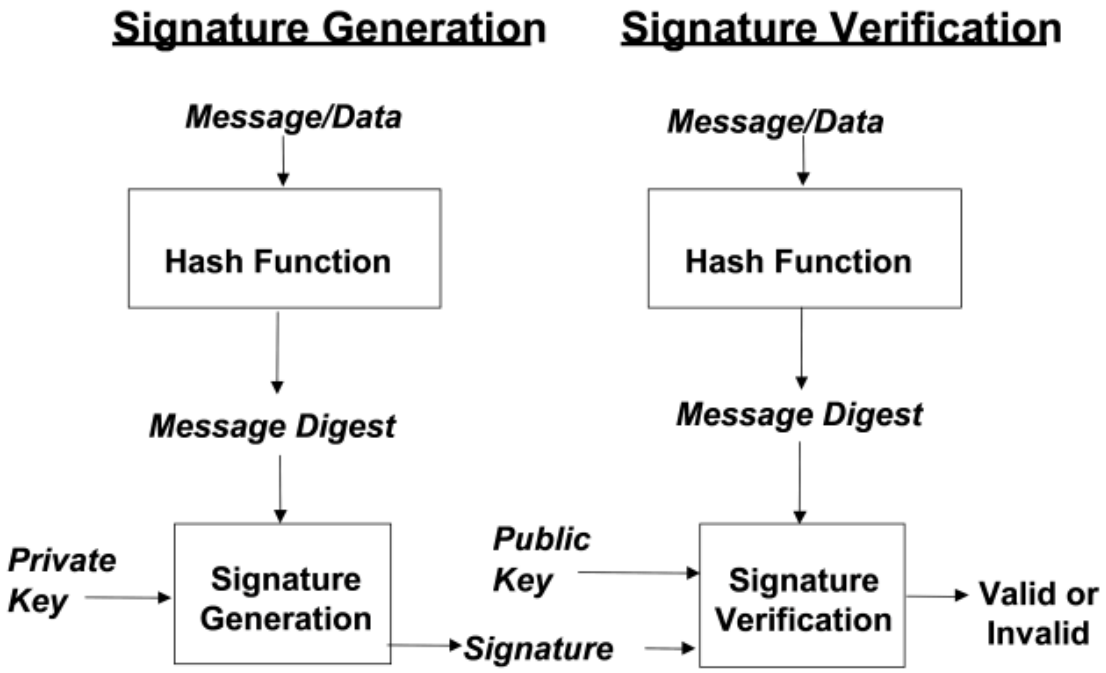
Basics

- Messages are signed by the private key and verified by anyone using the public key
  - Bob can verify a message signed by Alice using Alice's public key
  - Guarantees that the message was not modified and was signed by the holder of the corresponding private key (Alice)
- Messages to be signed must be small
  - Message could be a FW image for secure boot (100s of MB); Alice does not sign the image, but a hash (cryptographic digest) of the message
    - Preimage resistance: hash function $H()$ is one-way; cannot deduce x given $H(x)$
    - 2nd preimage resistance: cannot find $x' \neq x$ such that $H(x) = H(x')$, given x and $H(x)$
    - Collision resistance: cannot find $x_1 \neq x_2$ such that $H(x_1) = H(x_2)$
    - e.g. Using SHA2 or SHA3
  - Bob also computes the digest of the image and verifies Alice's signature against the digest



Digital signatures

Commercial National Security Algorithm Suite 1.0 (before Quantum Computers)



Algorithm	Function	Specification	Parameters
Advanced Encryption Standard (AES)	Symmetric block cipher for information protection	FIPS PUB 197	Use 256-bit keys for all classification levels.
Elliptic Curve Diffie-Hellman (ECDH) Key Exchange	Asymmetric algorithm for key establishment	NIST SP 800-56A	Use Curve P-384 for all classification levels.
Elliptic Curve Digital Signature Algorithm (ECDSA)	Asymmetric algorithm for digital signatures	FIPS PUB 186-4	Use Curve P-384 for all classification levels.
Secure Hash Algorithm (SHA)	Algorithm for computing a condensed representation of information	FIPS PUB 180-4	Use SHA-384 for all classification levels.
Diffie-Hellman (DH) Key Exchange	Asymmetric algorithm for key establishment	IETF RFC 3526	Minimum 3072-bit modulus for all classification levels
RSA	Asymmetric algorithm for key establishment	FIPS SP 800-56B	Minimum 3072-bit modulus for all classification levels
RSA	Asymmetric algorithm for digital signatures	FIPS PUB 186-4	Minimum 3072-bit modulus for all classification levels.

https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF

Quantum computers versus Cryptography

Shor's algorithm and Grover's algorithm

Shor's algorithm	**Grover's algorithm**
Exponential speedup	Quadratic speedup; find preimages in $O(2^{n/2})$
Broke prime factorization and discrete logarithm	Still not practical to solve searching, collision finding algorithms
RSA, ECDSA, ECDH, DSA no longer secure	Larger key sizes/digest needed for AES and Hashes

Quantum computers versus Cryptography

Commercial National Security Algorithm Suite 1.0 to 2.0

Algorithm	Function	Specification	Parameters
Advanced Encryption Standard (AES)	Symmetric block cipher for information protection	FIPS PUB 197	Use 256-bit keys for all classification levels.
Elliptic Curve Diffie-Hellman (ECDH) Key Exchange	Asymmetric algorithm for key establishment	NIST SP 800-56A	Use Curve P-384 for all classification levels.
Elliptic Curve Digital Signature Algorithm (ECDSA)	Asymmetric algorithm for digital signatures	FIPS PUB 186-4	Use Curve P-384 for all classification levels.
Secure Hash Algorithm (SHA)	Algorithm for computing a condensed representation of information	FIPS PUB 180-4	Use SHA-384 for all classification levels.
Diffie-Hellman (DH) Key Exchange	Asymmetric algorithm for key establishment	IETF RFC 3526	Minimum 3072-bit modulus for all classification levels
RSA	Asymmetric algorithm for key establishment	FIPS SP 800-56B	Minimum 3072-bit modulus for all classification levels
RSA	Asymmetric algorithm for digital signatures	FIPS PUB 186-4	Minimum 3072-bit modulus for all classification levels.

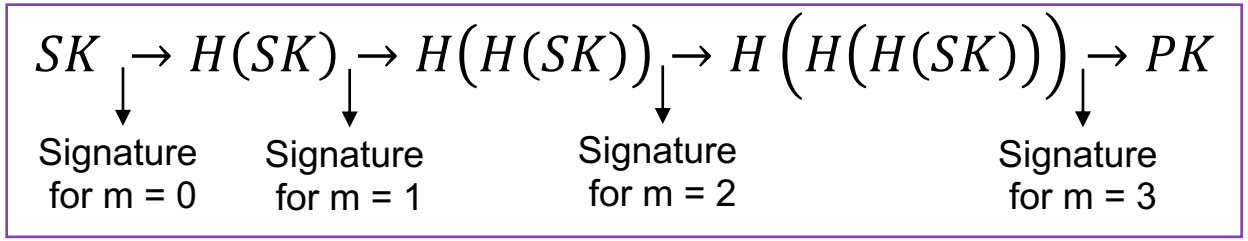
Algorithm	Function	Specification	Parameters
Advanced Encryption Standard (AES)	Symmetric block cipher for information protection	FIPS PUB 197	Use 256-bit keys for all classification levels.
ML-KEM (aka CRYSTALS-Kyber)	Asymmetric algorithm for key establishment	FIPS PUB 203	Use Category 5 parameter, ML-KEM-1024, for all classification levels.
ML-DSA (aka CRYSTALS-Dilithium)	Asymmetric algorithm for digital signatures in any use case, including signing firmware and software	FIPS PUB 204	Use Category 5 parameter, ML-DSA-87, for all classification levels.
Secure Hash Algorithm (SHA)	Algorithm for computing a condensed representation of information	FIPS PUB 180-4	Use SHA-384 or SHA-512 for all classification levels.
Leighton-Micali Signature (LMS)	Asymmetric algorithm for digitally signing firmware and software	NIST SP 800-208	All parameters approved for all classification levels. LMS SHA-256/192 is recommended.
Xtended Merkle Signature Scheme (XMSS)	Asymmetric algorithm for digitally signing firmware and software	NIST SP 800-208	All parameters approved for all classification levels.

https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF

Hash-based Signatures

Basic building block

- Comprises two components:
  - One-time signature (OTS) scheme
  - Single, long-term public key



- Winternitz OTS: Building block
  - Message is hashed → digest is encoded as a base b number → each digit is signed using a hash chain
- Hash chain:
  - sequential hashing of digit (b-1) times to obtain WOTS public key and sequential hashing m times to sign the digit value “m”
  - Consider a 2-bit digit m (b = 4) to be signed with secret key SK
    - signature for m = 0 is SK and signature for m = 3 is PK
  - PKs associated with all the digits are compressed into a WOTS PK
    - Compression step is different in XMSS and in LMS

Hash-based Signatures

Key generation

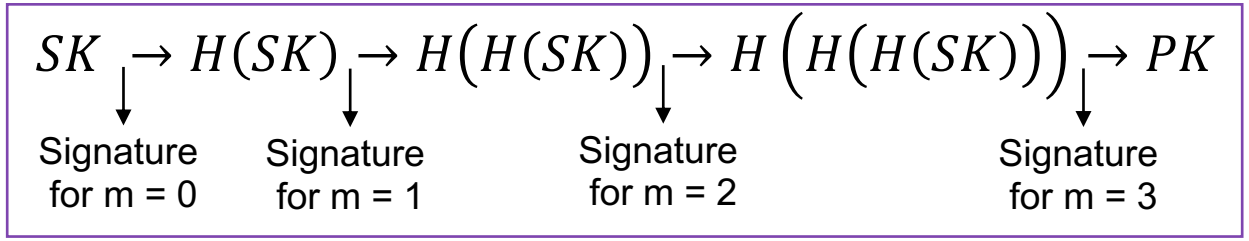
- Private key (sk): A random value of length 192 or 256 depending on the approved parameter set
  - Note: Hash function (H) can be used to obtain sk from some random seed ($SEED$) and chain number (i)
- Hash chain: Input = private key, Output = public key
- Hash chain length ($\mathrm{w}len$): Depending on the Winternitz parameter, $wlen = b - 1$
  - Winternitz parameter (w): $b = w$ for XMSS, $b = 2^{\mathrm{w}}$ for LMS
  - In Practice: Winternitz parameter (w) specified for XMSS is 16 and for LMS is {1,2,4,8} as per RFC 8391 and RFC 8554
- Public key (pk_i): Digest obtained after hashing $sk_i, wlen$ times.

$$SEED, i \rightarrow H \rightarrow sk_i \rightarrow H \rightarrow H(sk_i) \rightarrow H \rightarrow H^2(sk_i) \rightarrow \cdots \rightarrow H^{wlen}(sk_i) = pk_i$$

- All PKs compressed together forms the WOTS PK
  - $SEED$ and OTS PK forms the key-pair to sign one message

Hash-based Signatures

Signature generation

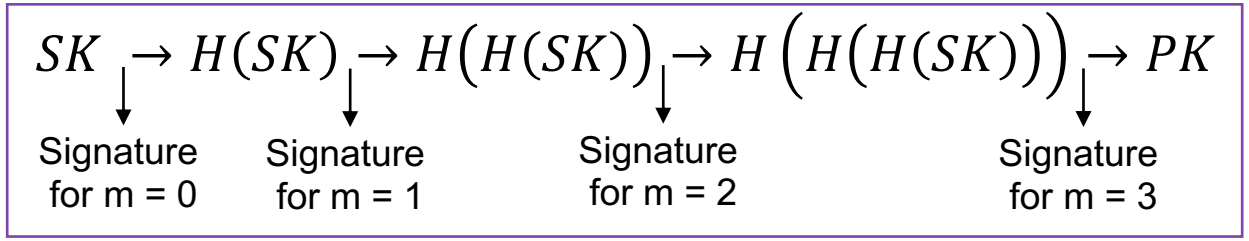


- Digest is encoded as base b number
  - for b = 16, digest = 0x65FA is converted into 4 hexadecimal digits “6”, “5”, “F” and “A”
  - for b = 4, digest = 0x65FA is converted into 8 digits “1”, “2”, “1”, “1” ,“3”, “3”, “2” and “2”
- Signing each digit of the digest (x_i) with sk_i to obtain WOTS public key $\mathrm{p}k_i$
  - $wlen$ depends on the digit value of x_i; x_i = 6 makes wlen = 6

	Digest			
x	6	5	F	A
sk	sk_0	sk_1	sk_2	sk_3
sig	$H^6(sk_0)$	$H^5(sk_1)$	$H^{15}(sk_2)$	$H^{10}(sk_3)$
pk	$H^{15}(sk_0)$	$H^{15}(sk_1)$	$H^{15}(sk_2)$	$H^{15}(sk_3)$

Hash-based Signatures

Signature verification



- Digest is encoded as base b number
  - for b = 16, digest = 0x65FA is converted into 4 hexadecimal digits “6”, “5”, “F” and “A”
  - for b = 4, digest = 0x65FA is converted into 8 digits “1”, “2”, “1”, “1” ,“3”, “3”, “2” and “2”
- Signing each digit of the digest (x_i) with sk_i to obtain WOTS public key $\mathrm{p}k_i$
  - $wlen$ depends on the digit value of x_i; x_i = 6 makes wlen = 6
- Hashing each signed digit (sig_i), $w - 1 - wlen_i$ times to verify pk_i
  - To verify if digit = 6 was signed using sk_0, hash the signature $H^6(sk_0)$ 9 times and compare with pk_0

	Digest			
x	6	5	F	A
sk	sk_0	sk_1	sk_2	sk_3
sig	$H^6(sk_0)$	$H^5(sk_1)$	$H^{15}(sk_2)$	$H^{10}(sk_3)$
pk	$H^{15}(sk_0)$	$H^{15}(sk_1)$	$H^{15}(sk_2)$	$H^{15}(sk_3)$

Hash-based Signatures

Possible attack

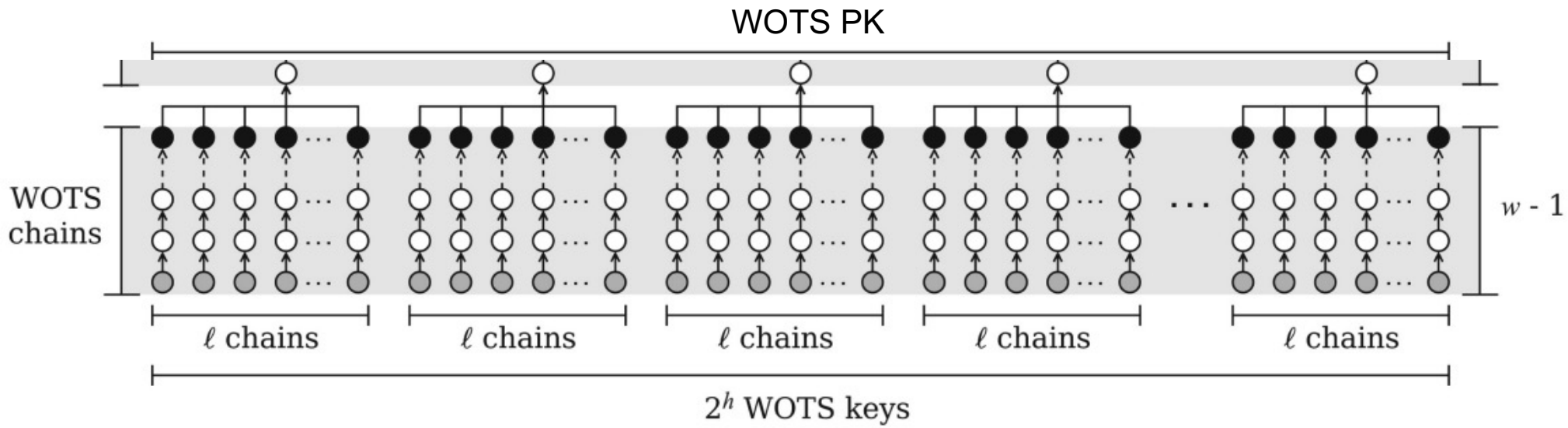
- An attacker can generate valid signatures for other messages/digests
  - Attacker acquires a signature for digit of 1
  - Attacker can generate valid signatures for any message > 1 by further hashing the signature
  - Attacker cannot generate valid signature for digit of 0 provided hash function is pre-image resistant
- Checksum added to resist an attacker from generating valid signatures
  - All digits are added, and the sum is encoded as base b
  - Checksum digits are signed with sk_i to obtain $\mathrm{p}k_i$
  - Attacker cannot increase message digits and decrease checksum digits to generate valid signatures

	Digest				Checksum	
len	6	5	F	A	2	4
sk	sk_0	sk_1	sk_2	sk_3	$s4$	sk_5
sig	$H^6(sk_0)$	$H^5(sk_1)$	$H^{15}(sk_2)$	$H^{10}(sk_3)$	$H^2(sk_4)$	$H^4(sk_5)$
pk	$H^{15}(sk_0)$	$H^{15}(sk_1)$	$H^{15}(sk_2)$	$H^{15}(sk_3)$	$H^{15}(sk_4)$	$H^{15}(sk_5)$

Hash-based Signatures

WOTS chains

- In practice:
  - For SHA-256/256, XMSS: w = 16, l = 67
  - l = 67: 256-bit digest encoded as base 4 digits, total digits: 64, total number of checksum digits: 3



Campos F, Kohlstadt T, Reith S, Stöttinger M. LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4. Progress in Cryptology - AFRICACRYPT 2020.

Hash-based Signatures

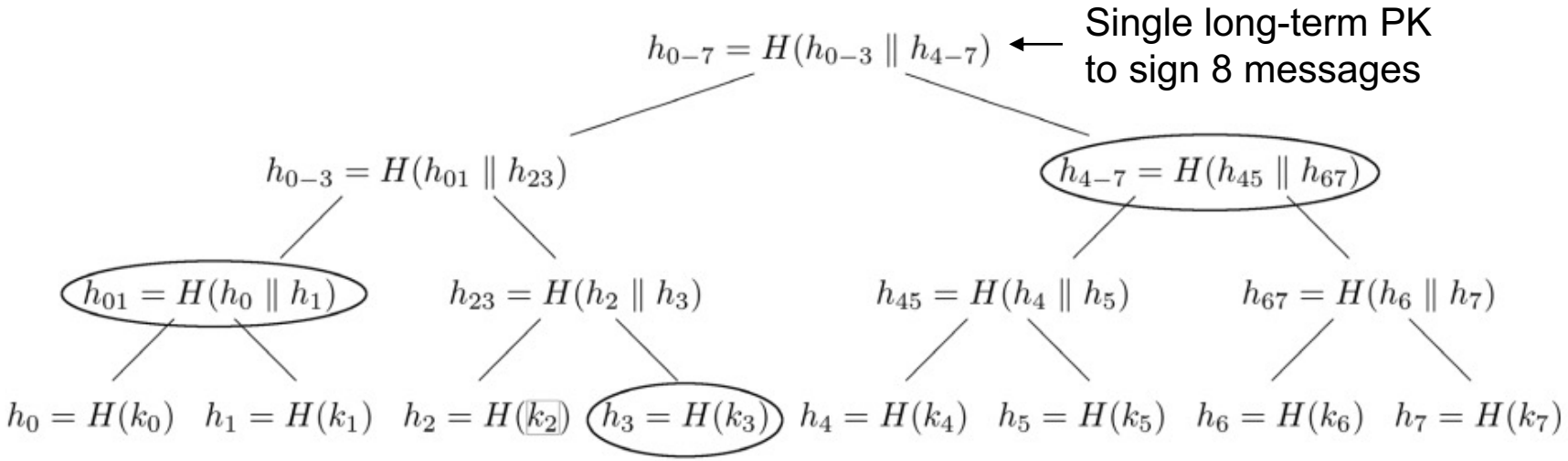
Merkle Tree

- One WOTS PK can only sign one message
- Concatenating many WOTS PKs can give us a single long-term PK
  - Pro: Can sign multiple messages
  - Con: Unacceptably large public key
- Use Merkle trees: balanced binary tree
  - Hash WOTS PKs to form the leaves of the tree
  - Hash the leaves in pair to form the next level up until all WOTS PKs are used to generate a single hash value (root node) → very short single long-term PK
  - Each key-pair can be used only once and must be kept track off → stateful
  - Pro: single long-term public key will be very short (192/256-bits long)
  - Con: additional information required to be provided along with the signature

Hash-based Signatures

Merkle Tree

- Each h_i (leaf) of the tree is the output of $\mathrm{H}(k_i)$; k_i are WOTS PKs
  - 8 messages can be signed with a tree of height 3
- If message is signed with k_2,
  - signer includes authentication path (h_3, h_{01}, h_{4-7}) and WOTS PK (k_2)
  - verifier computes h'_2, h'_{23}, h'_{0-3}, h'_{0-7}
  - if h'_{0-7} == h_{0-7} then k_2 may be used to verify the signed message

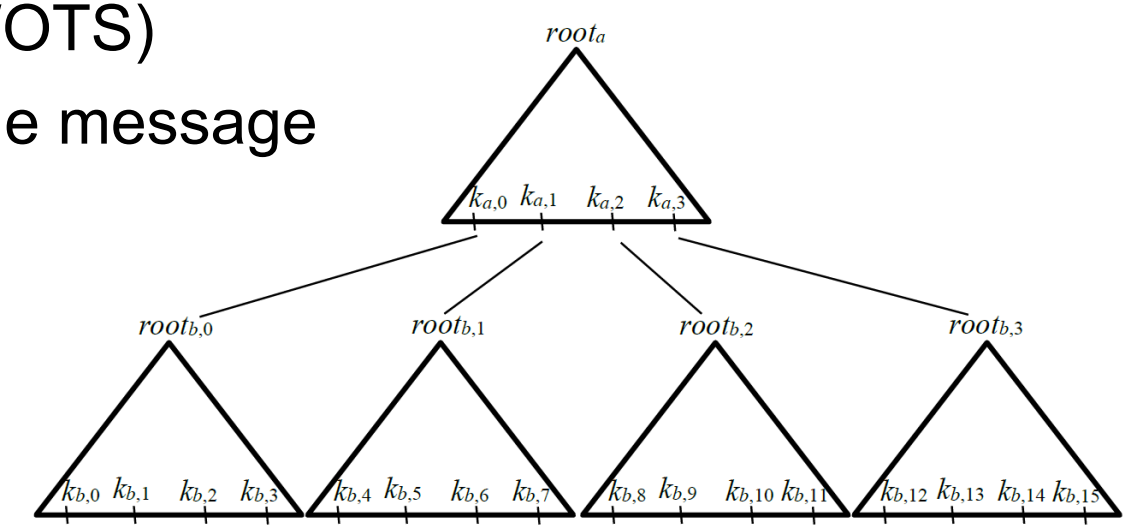


https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf

Hash-based Signatures

HyperTrees

- Computing the public key root value is prohibitively expensive as the tree gets large
- Solution: HyperTree
  - Tree of Trees
- Each layer signs the Tree below (using WOTS)
- Bottom Layer performs the WOTS over the message
- Advantages
  - Faster key generation
  - More one-time signatures
- Disadvantages
  - Larger signature values
  - Increased signing/verification latency



https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf

LMS

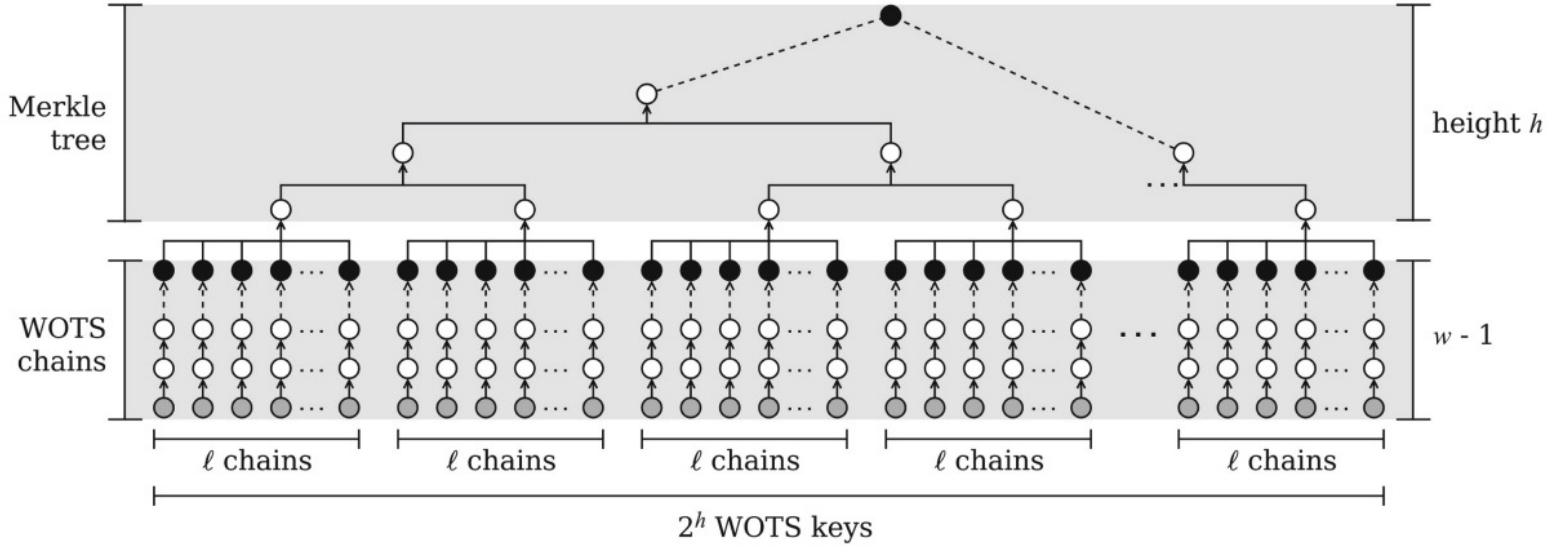
Prefixes

- Strengthen security by prepending prefix during hashing
- $\mathrm{pk} = H\left(p_3 \parallel H\left(p_2 \parallel H(p_1 \parallel sk)\right)\right)$
- $p_1, p_2, \text{and } p_3$ prefixes: include unique identifier for the long-term PK, indicator whether the hash is part of Winternitz chain or Merkle tree
- If Winternitz chain, then prefix includes number of the WOTS+ key, digit of the digest or the checksum being signed, location of hash in the chain
  - Ensures that each prefix in each hash function call is different
  - Resists collision attack
- In practice: H(I || u32str(q) || u16str(i) || u8str(j) || sk)
  - simple hash formatting (concatenation) but complex input formatting (non-uniform bit-sizes for a given data-width)
  - j denotes chain index (location of hash in the chain)
  - i denotes index of the secret key (digit index of digest or checksum)
  - q denotes LMS leaf identifier (tracks used private keys)
  - I denotes the LMS key identifier (128-bit identifier of the LMS public/private key pair)

LMS

PK leaf compression: WOTS+ PK

- All individual public keys for each digit are concatenated
- Single hash function to obtain single leaf PK of the Merkle tree



- In Practice: K = H(I || u32str(q) || u16str(D_PBLC) || y[0] || ... || y[p-1])
  - D_PBLC denotes fixed 2-byte value 0x8080
  - y[0] .. y[p-1] denotes the WOTS+ public keys (pk) derived from p secret keys (sk)

Campos F, Kohlstadt T, Reith S, Stöttinger M. LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4. Progress in Cryptology - AFRICACRYPT 2020.

LMS

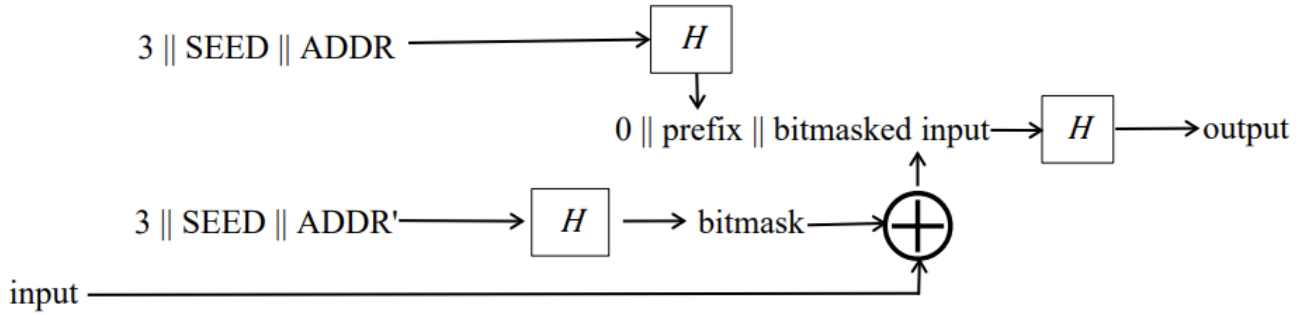
Parameter set approved by NIST

- NIST approved SHA-256 and SHAKE256 hash function
- Output/digest sizes supported are 256-bits and 192-bits
- Winternitz parameter set approved is {1,2,4,8}
- Tree height (h): 5,10,15,20,25
- Number of hash chains (l) depends on digest length and Winternitz parameter
  - For w=2, 256-bit digest is divided into 2-bit digits → 128 digits
  - Max. value of 2-bit digit=3; 3*128=384 → log(384) ≈ 9; to store 9-bit binary value using 2-bit digits, additional 5 indices → checksum = 5
  - Total number of indices to store digest and checksum = 133

XMSS

Prefixes and bitmasks

- Strengthen security by prepending prefix during hashing
- Also uses bitmasks which are exclusive-ORed with the input
- $\mathrm{pk} = H\left(p_{15} \ldots \| \left(H\left(p_2 \| \left(H\left(p_1 \| (sk \oplus bm_1)\right) \oplus bm_2\right)\right) \oplus bm_3\right) \ldots\right)$
- $p_1, p_2, \ldots$ and p_{15} prefixes: include unique identifier for the long-term PK, indicator whether the hash is part of Winternitz chain or Merkle tree
- If Winternitz chain, then prefix includes number of the WOTS+ key, digit of the digest or the checksum being signed, location of hash in the chain
  - Ensures that each prefix in each hash function call is different
  - Resists collision attack
- In practice:
  - Complex hash formatting (prefixes, XOR with bitmasks)
  - Simple input formatting

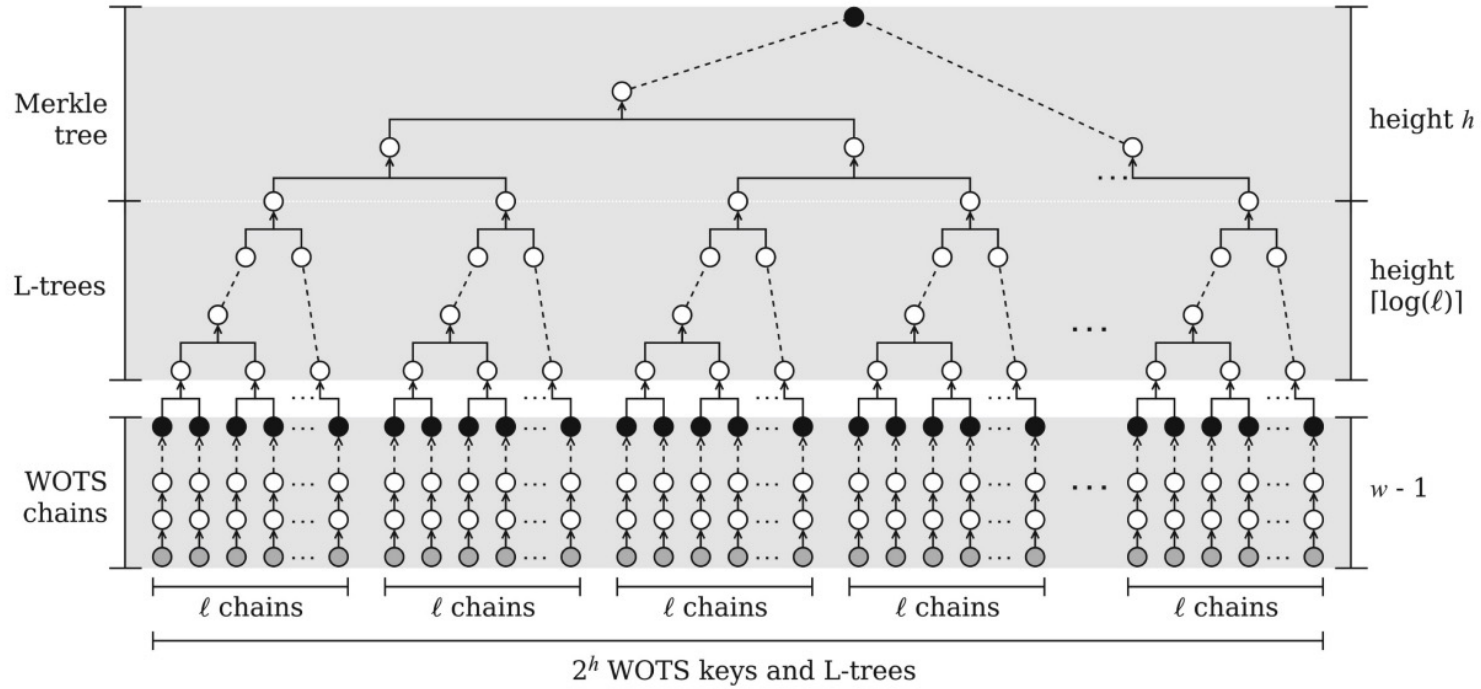


https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf

XMSS

PK leaf compression

- Using L-Tree: Unbalanced binary tree
- W=16, l = 67, #hashes called in L-Tree = 70




```
+-------------------------+
| layer address  (32 bits)|
+-------------------------+
| tree address   (64 bits)|
+-------------------------+
| type = 1       (32 bits)|
+-------------------------+
| L-tree address (32 bits)|
+-------------------------+
| tree height    (32 bits)|
+-------------------------+
| tree index     (32 bits)|
+-------------------------+
| keyAndMask     (32 bits)|
+-------------------------+
```

An L-tree Address

Campos F, Kohlstadt T, Reith S, Stöttinger M. LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4. Progress in Cryptology - AFRICACRYPT 2020
Request For Comments: RFC8391, https://www.rfc-editor.org/rfc/rfc8391

XMSS

Parameter set approved by NIST

- NIST approved SHA-256 and SHAKE256 hash function
- Output/digest sizes supported are 256-bits and 192-bits
- Winternitz parameter recommended is 16
- Tree height (h): 10,16,20
- Number of private keys per SEED becomes 67 for 256-bit and 51 for 192-bit
  - For w=16, 256-bit digest is divided into 4-bit digits → 64 digits
  - Max. value of 4-bit digit=15; 15*64=960 → log(960) ≈ 10; to store 10-bit binary value using 4-bit digits, additional 3 indices → checksum = 3
  - Total number of indices to store digest and checksum = 67

Summary

- Hash-based signatures replaced classical asymmetric algorithms to resist quantum computers
  - Security of HBS only depends on hash functions → infeasibility of finding pre-image and second pre-image
- XMSS and LMS standardized by NIST: SP 800-208
  - Winternitz One-time signature
  - Single long-term public key
  - Merkle Tree
- XMSS has more complex hashing scheme and simple input formatting
- LMS has simpler hashing scheme but more complex input formatting
- Stateful schemes
  - Signatory must keep track of which WOTS+ leaves have been previously used
  - Every signature must use a different leaf → Prefixes and bitmasks makes sure of this!
- Typical use case is for FW image signing
  - Finite number of FW images to be signed
  - Signatory is a usually a single entity

References

- Recommendation document for Stateful HBS: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf
- RFC for LMS: https://www.rfc-editor.org/rfc/pdfrfc/rfc8554.txt.pdf
- RFC for XMSS: https://www.rfc-editor.org/rfc/pdfrfc/rfc8391.txt.pdf
- Digital Signature Standard: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf
- SHA-3 standard: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
- CNSA 2.0: https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF
- Report on PQC: https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf

Moving Towards a Post Quantum Future @ Synopsys

Quantum Safety with Synopsys PQC Solutions

Public Key Accelerators

Target I/F | ALU | Operand Registers | Control & Status Registers | Instruction Unit / Memory

New IP

Agile Public Key Accelerators
- PQC algorithms
- ECC & RSA
- Hybrid crypto
- Crypto agility
- High performance
- Scalable
- Simple integration
- Full offload from processor
- Physical security

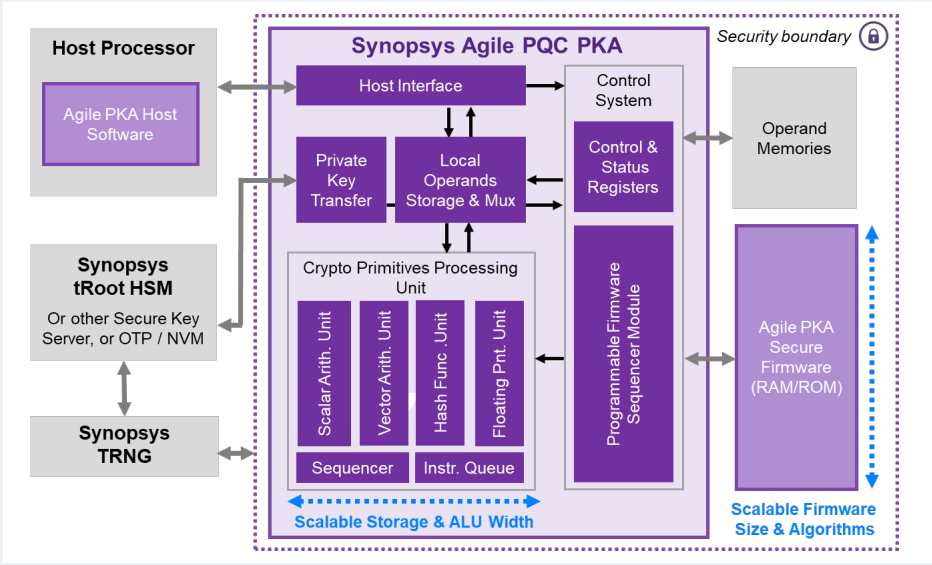
Cipher & Hash Accelerators

Target I/F | Initiator I/F | Sequencer | SDMA | Crypto Cores Symmetric Hash/MAC | keys

TRNGs

Host I/F | Serial I/F | Registers | Random Noise Seed Generator | PRNG Noise Whitening Circuit | Random Number I/F

Existing symmetric/hash engines and TRNG remain effective solutions
- Quantum safe with larger keys
- Support hash-based PQC and key generation entropy for binomial distributions, etc.



Agile PKA complies with the latest quantum resistant standards
- Agile and highly configurable, adaptable to future PQC algorithm updates via firmware
- NIST standards-compliant: ML-KEM (FIPS 203), ML-DSA (FIPS 204), SLH-DSA (FIPS 205), XMSS and LMS (SP 800-208)
- Supports full PQC digital signatures, key encapsulation, key exchange, and encrypt/decrypt functions
- Support for traditional ECC/RSA algorithms
- Ready for FIPS 140-3 security certification
- Protects against side channel and fault injection attacks

BACKUP

Stateless Hash-based Signature (SLH-DSA)

(SPHINCS+)

- NIST approved stateless hash-based signature scheme SLH-DSA based on Sphincs+ algorithm (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf)
  - Removes the stateful requirement by using few-time signature scheme instead of one-time signature scheme as in XMSS and LMS
  - Suitable for distributed signatories
- Combines a many time signature scheme (FORS) with an XMSS^{MT} like signature
  - FORS: Forest of Random Subsets
  - Message is signed with FORS
  - FORS signature is signed with XMSS^{MT}
- The specific leaf in the hyper-tree is selected randomly on every signature
  - There are so many leaves (>= 2^{63}) chance of collision is infinitesimally small
  - Use of FORS mitigates collisions

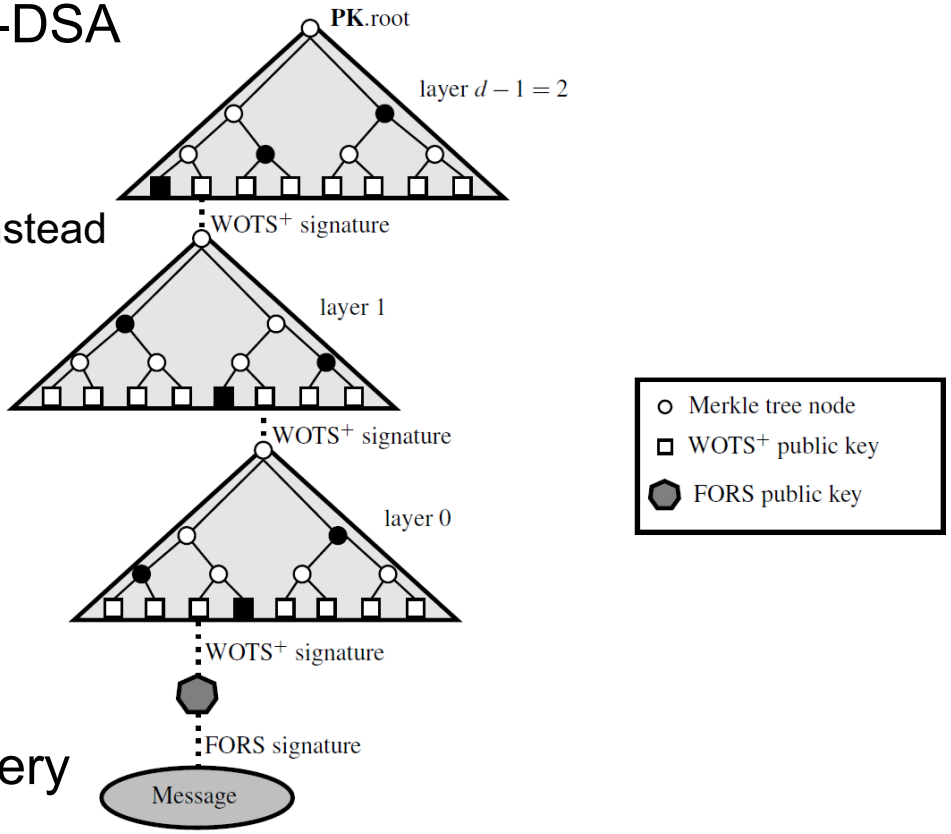


Figure 1. An SLH-DSA signature

XMSS

PK leaf compression: In practice

- Using L-Tree: Unbalanced binary tree
- W=16, l = 67, #hashes called in L-Tree = 70

```
Algorithm 8: ltree

  Input: WOTS+ public key pk, address ADRS, seed SEED
  Output: n-byte compressed public key value pk[0]

  unsigned int len' = len;
  ADRS.setTreeHeight(0);
  while ( len' > 1 ) {
    for ( i = 0; i < floor(len' / 2); i++ ) {
      ADRS.setTreeIndex(i);
      pk[i] = RAND_HASH(pk[2i], pk[2i + 1], SEED, ADRS);
    }
    if ( len' % 2 == 1 ) {
      pk[floor(len' / 2)] = pk[len' - 1];
    }
    len' = ceil(len' / 2);
    ADRS.setTreeHeight(ADRS.getTreeHeight() + 1);
  }
  return pk[0];
```

```
Algorithm 7: RAND_HASH

  Input:  n-byte value LEFT, n-byte value RIGHT, seed SEED,
          address ADRS
  Output: n-byte randomized hash

  ADRS.setKeyAndMask(0);
  KEY = PRF(SEED, ADRS);
  ADRS.setKeyAndMask(1);
  BM_0 = PRF(SEED, ADRS);
  ADRS.setKeyAndMask(2);
  BM_1 = PRF(SEED, ADRS);

  return H(KEY, (LEFT XOR BM_0) || (RIGHT XOR BM_1));
```

Hash-based Signatures

eXtended Merkle Signature Scheme (XMSS) versus Leighton-Micali Signature (LMS)

XMSS	LMS
Uses Merkle tree to sign multiple messages	Uses Merkle tree to sign multiple messages
Stateful; index I needs to be tracked	Stateful; index I needs to be tracked
Hash function used in WOTS comprises XOR with bitmask; $H(prefix, bm \oplus m)$	Hash function used in WOTS comprises prefixes; $H(prefix, m)$
Public keys of individual message chunks are combined using an L-tree to form the leaf public key PK	Public keys of individual message chunks are concatenated and hashed to form the leaf public key PK
# hashes to obtain leaf PK = 70 for w = 16	# hashes to obtain leaf PK = 1
XMSS^{MT} allows up to 12 levels of trees	HSS allows up to 8 levels of trees