Hash-based signature schemes

Martin Stanek

Department of Computer Science
Comenius University
stanek@dcs.fmph.uniba.sk

Cryptology 1 (2023/24)

Content

Introduction

- signature scheme resistant to quantum computers
  - security is not based on hardness of factorization, discrete log etc.
  - security based on properties of hash functions, such as preimage resistance, 2nd preimage resistance, collision resistance
- we discuss some schemes and their limitations
  - number of signing operations
  - state
  - key/signature length

Lamport scheme

- Lamport (1979)
- $f : X \longrightarrow Y$ (for example a hash function)
- message/hash $m = m_1, m_2, \ldots, m_n \in \{0, 1\}^n$
- private key: $x_{i,j} \overset{\$}{\longleftarrow} X$ for $i = 1, \ldots, n$, $j \in \{0, 1\}$
- public key: $y_{i,j} = f(x_{i,j})$
- signature $\sigma = (x_{1,m_1}, x_{2,m_2}, \ldots, x_{n,m_n})$
- verification of $\sigma = (\sigma_1, \ldots, \sigma_n)$ given m:

$$f(\sigma_i) \overset{?}{=} y_{i,m_i} \qquad i = 1, \ldots, n.$$

Lamportova scheme – remarks

- **one-time** signature scheme
  - signing two messages (if they differ in more than one bit) $\Rightarrow$ combine their signatures to forge signature for a new message
  - signing a hash does not help – signatures of $O(\lg n)$ messages are enough to cover 0 and 1 on almost all hash positions
- key length for 256-bit hash function f and for $n = 256$:
  - public key: $2 \cdot 256 \cdot 256 = 16$ KiB
  - private key: 16 KiB (for 256-bit values $x_{i,j}$)
- signature length: 8 KiB
- speed is not a problem

Lamport scheme – improvements (1)

- Merkle (1979)
- short private key
  - generate the values with PRNG or PRF
- short public key
  - $y = H(y_{1,0}, y_{1,1}, \dots, y_{n,0}, y_{n,1})$
  - values $y_{i,1-m_i}$, i.e., those not used in signing, are explicitly added to the signature (making it 2 times longer)
  - verification: computation of y_{i,m_i} and verification of y

Lamportova scheme – improvements (2)

- reducing the length of keys and signatures by half (approx.):
  - add $\lfloor \lg n \rfloor + 1$ bits to the message (or its hash), counting 0 bits: $m' = m \,||\, (\#_0 m)_2$
  - let $n' = n + \lfloor \lg n \rfloor + 1$
  - private key: $x_i \overset{\$}{\leftarrow} X$ for $i = 1, \dots, n'$
  - public key: $y_i = f(x_i)$
  - signature is a sequence:

$$(x_i)_{i \in I} \qquad \text{for } I = \{1 \le i \le n' \mid m'_i = 1\}$$

  - m' contains at least one bit 1
  - change from 0 to 1 in m – corresponding x_i is not in the signature
  - change from 1 to 0 in m – more zeroes, hence at least one 0 changes to 1 in the counter and corresponding x_i is not in the signature

WOTS

- WOTS – Winternitz one time signature
- reducing signature length and increasing time complexity (TMTO)
- key-dependent function $f : X \times X \longrightarrow X$ (like MAC)
  - notation: $f(k, x) = f_k(x)$
- iterating f:
  - $f_k^0(x) = k, f_k^1(x) = f_k(x), f_k^2(x) = f_{f_k(x)}(x), \dots$
  - $f_k^r(x) = f_{f_k^{r-1}(x)}(x)$, for $r \geq 1$
  - other WOTS variants, such as direct iteration of (hash) function, require stronger security assumptions (collision resistance)
- parameter $w > 1$, for example $w = 16$ or $w = 32$
- w-ary representation of m
  - $m = (m_1, \dots, m_{l_1})$, where $0 \leq m_i < w$ for $i = 1, \dots, l_1$
- checksum: $C = \sum_{i=1}^{l_1}(w - 1 - m_i)$
- let $l = l_1 + l_2$, where l_2 is a maximal length of C (w-ary representation)

WOTS (2)

- private key: $k_1, \dots, k_l \overset{\$}{\leftarrow} X$
- public key: $(x, y_1, \dots, y_l)$, where
  - $x \overset{\$}{\leftarrow} X$
  - $y_i = f_{k_i}^{w-1}(x)$ for $i = 1, \dots, l$
- signing:
  1. split message and its checksum into blocks: $m \,||\, C \mapsto m_1, \dots, m_l$
  2. signature $\sigma = (\sigma_1, \dots, \sigma_l) = (f_{k_1}^{m_1}(x), \dots, f_{k_l}^{m_l}(x))$
- signature verification:
  1. given m; compute its checksum and split both into blocks
  2. verify

$$f_{\sigma_i}^{w-1-m_i}(x) \overset{?}{=} y_i \qquad i = 1, \dots, l$$

WOTS (3)

- correctness is trivial
- the scheme is insecure without the checksum:
  - the attacker takes σ_i for m_i and for any $m'_i > m_i$ he can compute

$$\sigma'_i = f_{k_i}^{m'_i}(x) = f_{\sigma_i}^{(m'_i - m_i)}(x)$$

  - σ'_i is a correct signature for i-th block m'_i
  - the checksum prevents these shifts
- still one-time scheme

WOTS – a sample parameters instantiation

- key length (let f be a 256-bit hash function, and $|m| = 256$):
  - for $w = 16$: $l_1 = 256/4 = 64$, $l_2 = \lfloor \lg(64 \cdot 15)/4 \rfloor + 1 = 2$
  - we get $l = 66$
  - public key: $(l + 1) \cdot 256 \approx 2.1$ KiB
  - private key: $l \cdot 256 \approx 2.1$ KiB (for 256-bit values)
- signature length: $l \cdot 256 \approx 2.1$ KiB
  - approx. $\lg w = 4$ time shorter than in the original Lamport scheme
- speed – comparison with the original Lamport scheme
  - signing: $\approx l \cdot w/2$ calls of f vs. 0
  - verification: $\approx l \cdot w/2$ calls of f vs. $|m|$ calls (WOTS is approx. $w/(2 \cdot \lg w)$ times slower)

WOTS$^+$

- Hülsing (2013)
- similar to WOTS, different iteration of f $\Rightarrow$ tighter security proof
  - weaker or more standard security assumptions (f properties)
  - WOTS$^+$ as a replacement for WOTS in other schemes
- iteration of $f : K \times X \longrightarrow X$
  - input: key $k \in K$, $x \in X$, counter $i \in \mathbb{N}$, randomizing values $\mathbf{r} = (r_1, \ldots, r_j)$ for $j \geq i$
  - computation:

$$\begin{aligned} c_k^0(x, \mathbf{r}) &= x \\ c_k^1(x, \mathbf{r}) &= f_k(c_k^0(x, \mathbf{r}) \oplus r_1) \\ &\ldots \\ c_k^i(x, \mathbf{r}) &= f_k(c_k^{i-1}(x, \mathbf{r}) \oplus r_i) \end{aligned}$$

WOTS$^+$ (2)

- parameters w, $l = l_1 + l_2$ just like in WOTS
- private key: $x_1, \dots, x_l \overset{\$}{\leftarrow} X$
- public key: $((\mathbf{r}, k), y_1, \dots, y_l)$
  - $k \overset{\$}{\leftarrow} K$
  - $\mathbf{r} = (r_1, \dots, r_{w-1})$, where $r_i \overset{\$}{\leftarrow} X$
  - $y_i = c_k^{w-1}(x_i, \mathbf{r})$, for $i = 1, \dots, l$
- signing:
  1. (just like WOTS) split message and its checksum into blocks: $m \,||\, C \mapsto m_1, \dots, m_l$
  2. signature $\sigma = (\sigma_1, \dots, \sigma_l) = (c_k^{m_1}(x_1, \mathbf{r}), \dots, c_k^{m_l}(x_l, \mathbf{r}))$

Remark: The checksum ensures that for given $m_1, \dots, m_l$ any other message contains at least one $m'_j < m_j$.

WOTS$^+$ (3)

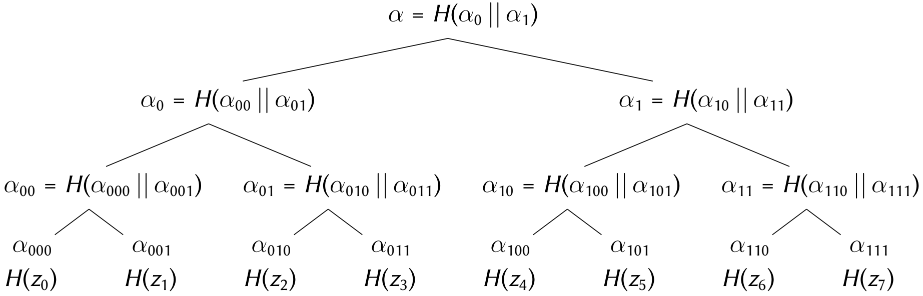
- verification:
  1. split m and its checksum into blocks
  2. verify
  $$c_k^{w-1-m_i}(\sigma_i, \mathbf{r}_{m_i+1,w-1}) \overset{?}{=} y_i \qquad i = 1, \dots, l$$
  where $\mathbf{r}_{m_i+1,w-1} = (r_{m_i+1}, \dots, r_{w-1})$

- reducing the key length:
  - values y_i in WOTS a WOTS$^+$ are computed in verification, the public key can be replaced by their hash
  - values in $\mathbf{r}$ can be generated from a seed (PRNG)
  - similarly for values in the private key

Merkle hash tree

- Merkle, 1979
- signature scheme for a single message is impractical
- multiple one-time signature schemes
  - combined together into one tree-like structure
- Merkle hash tree – various applications, e.g.,
  - file systems (ZFS), BitTorrent, Bitcoin, Git, …
- binary tree:
  - input: data $z_0, \dots, z_{2^h-1}$, for $h \geq 1$
  - assume 2^h input data for simplicity
  - H – hash function
  - values in the leaf nodes: $H(z_i)$, for $i = 0, \dots, 2^h - 1$
  - value in the node v: $H(a \,||\, b)$, where a (or b) is the value of the left (or right) child of v

Example



$\alpha = H(\alpha_0 \,||\, \alpha_1)$
$\alpha_0 = H(\alpha_{00} \,||\, \alpha_{01})$
$\alpha_1 = H(\alpha_{10} \,||\, \alpha_{11})$
$\alpha_{00} = H(\alpha_{000} \,||\, \alpha_{001})$
$\alpha_{01} = H(\alpha_{010} \,||\, \alpha_{011})$
$\alpha_{10} = H(\alpha_{100} \,||\, \alpha_{101})$
$\alpha_{11} = H(\alpha_{110} \,||\, \alpha_{111})$
α_{000}
$H(z_0)$
α_{001}
$H(z_1)$
α_{010}
$H(z_2)$
α_{011}
$H(z_3)$
α_{100}
$H(z_4)$
α_{101}
$H(z_5)$
α_{110}
$H(z_6)$
α_{111}
$H(z_7)$

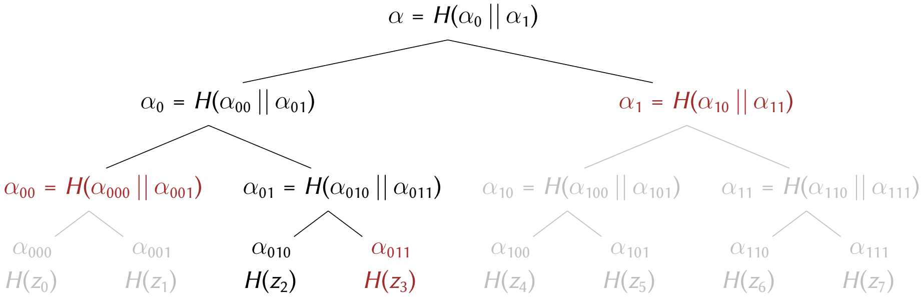
Merkle hash tree for $h = 3$

Merkle Signature Scheme (MSS)

- leaf data – public keys of OTS schemes
- public key: root value of the Merkle hash tree
- private key:
  - key for some (suitable) algorithm to generate OTS schemes
  - alternatively a sequence of OTS schemes private keys
- we are able to sign up to 2^h messages:
  - use OTS schemes sequentially one by one
  - saved state – counter of already used OTS schemes
- a signature contains:
  - signature using next OTS scheme
  - public key of this OTS scheme
  - authentication path

Authentication path

- authentication path for a given leaf:
  - a sequence of sibling nodes values on the path to the root
  - it allows to compute the root value from leaf and additional h values



$\alpha = H(\alpha_0 \,||\, \alpha_1)$
$\alpha_0 = H(\alpha_{00} \,||\, \alpha_{01})$
$\alpha_1 = H(\alpha_{10} \,||\, \alpha_{11})$
$\alpha_{00} = H(\alpha_{000} \,||\, \alpha_{001})$
$\alpha_{01} = H(\alpha_{010} \,||\, \alpha_{011})$
$\alpha_{10} = H(\alpha_{100} \,||\, \alpha_{101})$
$\alpha_{11} = H(\alpha_{110} \,||\, \alpha_{111})$
α_{000} $H(z_0)$
α_{001} $H(z_1)$
α_{010} $H(z_2)$
α_{011} $H(z_3)$
α_{100} $H(z_4)$
α_{101} $H(z_5)$
α_{110} $H(z_6)$
α_{111} $H(z_7)$

authentication path for α_{010}: $\text{auth}(\alpha_{010}) = (\alpha_{011}, \alpha_{00}, \alpha_1)$

MSS – verification

- input:
  - MSS public key (root value)
  - message
  - signature in OTS scheme and its public key
  - authentication path
- verification steps:
  1. verify signature in OTS scheme
  2. compute a root value (from public key and authentication path)
  3. compare the root value with the public key of MSS
- security depends on properties of these elements:
  - H used in Merkle hash tree
  - used OTS scheme
  - cryptographic constructions used in generation of OTS schemes

MSS – remarks

- all OTS schemes and entire tree must be generated for MSS instantiation
  - limit on h (big trees are impractical), and therefore on the number of signatures
- authentication path computation:
  - Merkle tree traversal – sequential computation of authentication paths
  - possible in linear time and memory $O(h)$ (per authentication path)
- What if 2^h OTS schemes were used?
  - generate a new instance (tree) and distribute public key
  - use the last OTS scheme to sign a new tree
  - create a sufficiently large instance such so this does not happen
- signatures are longer (authentication path) – h hash values
  - size of the tree impact the length of the signature

MSS – remarks (2)

- if WOTS (WOTS$^+$) is used in MSS
  - public key of OTS scheme is computed from signature (beside x for WOTS or $(\mathbf{r}, k)$ for WOTS$^+$)
  - signature is shorter, this idea used, for example, in XMSS
- problem in MSS: no longer one-time scheme but now it is **stateful**
  - important for security: avoid using a OTS scheme multiple times
  - important for efficiency: authentication path computation
- state might be acceptable for some use cases
  - for example CA signs certificates in HSM
- in general having state is undesirable and potential problem
  - load-balancing, system recovery from backups (old state), etc.

XMSS (eXtended Merkle Signature Scheme)

- Buchmann, Dahmen, Hülsing (2011)
- modification of Merkle hash tree
  - xor *masks* when aggregating
  - L-trees hanging in leaves – storing public keys for WOTS schemes (hashing public key – useful in security proofs)
  - assumption of second preimage resistance is sufficient, instead of collision resistance
- using WOTS – public key not needed in signature
  - it can be computed from signature and verified when verifying XMSS tree
- masks (random string with suitable length)
  - left and right masks chosen for each non-leaf height

XMSS tree

$$\begin{tikzcd}[column sep=tiny]
 & & & & \alpha = H(\alpha_0 \oplus b_0^l \,||\, \alpha_1 \oplus b_0^r) \arrow[dll, dash] \arrow[drr, dash] & & & & \\
 & & \alpha_0 = H(\alpha_{00} \oplus b_1^l \,||\, \alpha_{01} \oplus b_1^r) \arrow[dl, dash] \arrow[dr, dash] & & & & \alpha_1 = H(\alpha_{10} \oplus b_1^l \,||\, \alpha_{11} \oplus b_1^r) \arrow[dl, dash] \arrow[dr, dash] & & \\
 & \ldots & & \alpha_{01} = H(\alpha_{010} \oplus b_2^l \,||\, \alpha_{011} \oplus b_2^r) \arrow[dl, dash] \arrow[dr, dash] & & \ldots & & \ldots & \\
 & & \alpha_{010} & & \alpha_{011} & & & &
\end{tikzcd}$$

root of L-tree (for public key in WOTS scheme)
(new set of masks, shared among all L-trees)

XMSS – remarks

- L-tree
  - construction like XMSS tree (new independent masks, but same for each L-tree)
  - leaves – public keys for WOTS scheme $(x, y_1, \dots, y_l)$
  - not necessary a power of 2, but still binary tree
- still stateful
- example parameters (proposed in 2011):
  - $h = 20$, capacity for 2^{20} signatures
  - SHA-256, public/private key length: 13 568 / 280 bits
  - $w = 16 \Rightarrow$ signature 22 296 bits; 196-bit security
  - $w = 64 \Rightarrow$ signature 16 664 bits; 146-bit security
- a more recent proposal: RFC 8391 (2018)

Stateless schemes

- remove state from the scheme
- basic idea by Goldreich: a huge binary tree, for example $h = 256$
  - every node represents an OTS scheme
  - every scheme can be generated using private key and position of the node (PRF/PRNG)
- private key: random bit string
- public key: public key of OTS scheme in the root of the tree (Y)

Signature

- signing message m:
  1. $H(m)$ denotes a concrete leaf β in the tree
  2. compute keys for OTS schemes for all nodes (and their siblings) on path from β to the root (let public keys $Y_h^l, Y_h^r, \dots, Y_1^l, Y_1^r$)
  3. sign $H(m)$ using OTS scheme for $\beta \mapsto \sigma_m$)
  4. every siblingly pair of public keys is signed by OTS scheme for their parent node, obtaining signatures $(\sigma_0, \sigma_1, \dots, \sigma_{h-1})$, where σ_0 is the signature in the root
  5. signature: $\sigma = (\sigma_m, \sigma_0, \dots, \sigma_{h-1}, Y_1^l, Y_1^r, \dots, Y_h^l, Y_h^r)$

Verification

- input:
  - message m (or $H(m)$)
  - public key for root OTS scheme Y
  - signature $\sigma = (\sigma_m, \sigma_0, \ldots, \sigma_{h-1}, Y_1^l, Y_1^r, \ldots, Y_h^l, Y_h^r)$
- verification steps:
  1. determine a leaf β from $H(m)$
  2. verify signature σ_m (using Y_h^l or Y_h^r)
  3. verify *certification path*, i.e. signatures $(\sigma_0, \ldots, \sigma_{h-1})$, using public keys from σ and using public key Y for σ_0 verification
- state not needed, probability of collision is negligible
- signature in any node is always the same
  - public keys of children are certified
  - OTS scheme is sufficient
- problem: very long signatures, not practical

How to optimize stateless scheme

- usually a combination of multiple techniques
- deterministic/pseudorandom leaf selection in the tree
- using *few-time* signature schemes instead of WOTS in leaves
    - just for message signing
    - secure signing of few messages (e.g. 4)
    - smaller tree (less leaves)
    - examples: HORS, HORST, FORS
- change the structure from a single huge tree to multiple levels of smaller Merkle trees

Stateless scheme optimizations (2)

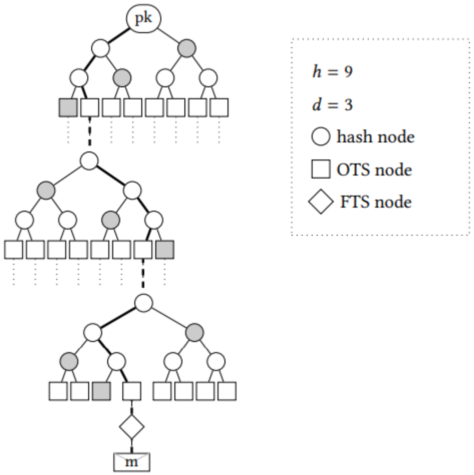


Figure 1: An illustration of a (small) SPHINCS structure.

Source: D. Bernstein et al.: The SPHINCS+ Signature Framework (2019)

PQC standardization

- SPHINCS+ if one of the three selected signature schemes for standardization
  - FIPS 205 (Draft) Stateless Hash-Based Digital Signature Standard
  - SLH-DSA based on SHAKE or SHA2 hash functions
- NIST Third Round Status Report:

  SPHINCS+ was selected for standardization because it provides a workable (albeit rather large and slow) signature scheme whose security seems quite solid and is based on an entirely different set of assumptions than those of our other signature schemes to be standardized.
- lengths in bytes (*s* – size-optimized (small), *f* – speed-optimized (fast)):

level	public key	signature	
128	32	7 856	*s*
128	32	17 088	*f*
256	64	29 792	*s*
256	64	49 856	*f*

Conclusion

- hash-based signature schemes
  - limited number of signatures
  - simple principles, usually simple requirements/assumptions
  - optimization aiming at practicality and weaker assumptions complicate construction