(To be) standardized hash-based signature schemes

Andreas Hülsing
Eindhoven University of Technology

SSR 2018
The quantum threat

- **Shor’s algorithm** breaks RSA, (EC)DSA, (EC)DH,…

- **Grover’s algorithm** asymptotically reduces complexity of brute-force search attacks by a square-root factor.
Why care today

- **EU** launched a one billion Euro project on quantum technologies
- Similar range is spent in **China**
- **US** administration passed a bill on spending $1.275 billion US dollar on quantum computing research
- **Google**, **IBM**, **Microsoft**, **Alibaba**, and others run their own research programs.
It’s a question of risk assessment
Long-lived systems

• Development time easily 10+ years
• Lifetime easily 10+ years
• At least make sure you got a secure update channel!
Merkle’s hash-based signatures
[Lam79,Mer89]

No new hardness assumptions

Provably (post-quantum) secure if (post-quantum) secure hash function is used

Basic concept extremely easy

Stateful
Basic Construction
Lamport OTS [Lam79]

Message \( M = b_1, \ldots, b_m \), OWF \( H \) \( \ast \) = n bit

[Diagram of Lamport OTS]

SK

PK

Sig

10.01.2019

https://huelsing.net
Merkle’s Hash-based Signatures

SIG = (i=2, , , , , )
Winternitz-OTS
Lamport-OTS in MSS

SIG = (i=2, , , , , )

Verification:

1. Verify

2. Verify authenticity of

We can do better!
WOTS in MSS

\[ \text{SIG} = (i=2, X, 0, 0, 0) \]

Verification:

1. Compute \( \bigcirc \) from \( \bigcirc \)
2. Verify authenticity of \( \bigcirc \)

Steps 1 + 2 together verify \( \bigcirc \)
Function chains

Hash function $h : \{0,1\}^n \rightarrow \{0,1\}^n$

Parameter $w$

Chain: $c^i(x) = h(c^{i-1}(x)) = h \circ h \circ \cdots \circ h$

$c^0(x) = x$

$c^1(x) = h(x)$

$c^{w-1}(x)$
WOTS

Winternitz parameter $w$ (usually a power of 2), security parameter $n$, message length $m$, hash function $h$

**Key Generation:** Compute $l$, sample $h_k$

$c^0(s_{k_1}) = s_{k_1}$

$c^1(s_{k_1})$

$c^1(s_{k_{l-1}})$

$c^0(s_{k_{l-1}}) = s_{k_{l-1}}$

$pk_1 = c^{w-1}(s_{k_1})$

$pk_2 = c^{w-1}(s_{k_{l-1}})$
WOTS Signature generation

Signature:
\( \sigma = (\sigma_1, \ldots, \sigma_\ell) \)

\( c^0(sk_1) = sk_1 \)

\( c^0(sk_\ell) = sk_\ell \)

\( \sigma_1 = c^{b_1}(sk_1) \)

\( \sigma_\ell = c^{b_\ell}(sk_\ell) \)

\( pk_\ell = c^{w-1}(sk_\ell) \)

10.01.2019
https://huelsing.net
WOTS Signature Verification

Verifier knows: M, w

Signature: $\sigma = (\sigma_1, \ldots, \sigma_\ell)$

$\sigma_1$

$\sigma_\ell$

$\mathbf{b}_1 \quad \mathbf{b}_2 \quad \mathbf{b}_3 \quad \mathbf{b}_4 \quad \ldots \quad \mathbf{b}_m \quad \mathbf{b}_{m+1} \quad \mathbf{b}_{m+2} \quad \ldots \quad \mathbf{b}_l$

$\mathbf{c}^1(\sigma_1) \quad \mathbf{c}^2(\sigma_1) \quad \mathbf{c}^3(\sigma_1) \quad \mathbf{c}^{w-1-b_1}(\sigma_1) \quad \mathbf{c}^{w-1-b_\ell}(\sigma_\ell)$

$\mathbf{p}_{k_1} \quad \mathbf{p}_{k_\ell}$

10.01.2019

https://huelsing.net
Multi-Tree MSS
Multi-Tree MSS

Uses multiple layers of trees to reduce key generation time

-> Key generation
(= Building first tree on each layer)
$\Theta(2^h) \rightarrow \Theta(d \times 2^{h/d})$

-> Worst-case signing times
$\Theta(h/2) \rightarrow \Theta(h/2d)$

-> Increases signature size by d-1 one-time signatures
Standardization in CFRG (adoption by NIST announced)
Competing proposals in IRTF’s CFRG

XMSS (RFC 8391)
- WOTS + MSS + Multi-tree (WOTS-T + XMSS-T + MT)
  - w = 16
  - h = 10, 16, 20
  - 8 MT (h,d) combinations (trees on all layers have same height)
  - SHA2-256 required; SHA2-512, SHAKE-128, SHAKE-256 optional

LMS (draft-mcgrew-hash-sigs-13)
- WOTS + MSS + Multi-tree (WOTS-LM + LMS + HSS=MT)
  - w = 2, 4, 16, 256
  - h = 5, 10, 15, 20, 25
  - 6 HSS combinations with two trees of independent height

- Only SHA2-256
Intermezzo: Multi-target attacks

- WOTS & Lamport need hash function \( h \) to be one-way
- Multi-tree of total height 60 with WOTS (\( w=16 \)) leads \( > 2^{60} \cdot 67 \approx 2^{66} \) images.
- Inverting one of them allows existential forgery (at least massively reduces complexity)
- \( q \)-query brute-force succeeds with probability \( \Theta \left( \frac{q}{2^{n-66}} \right) \) conventional and \( \Theta \left( \frac{q^2}{2^{n-66}} \right) \) quantum
- We loose 66 bits of security! (33 bits quantum)
New abstraction: Tweakable Hash Function

• Mitigation: Separate targets

• Common approach:
  • In addition to hash function and "input“ take
    • Hash "Address“ (uniqueness in key pair)
    • Hash "key“ used for all hashes of one key pair
      (uniqueness among key pairs)

• Tweakable Hash Function:
  \[ T \left( PP, TW, MSG \right) \rightarrow MD \]

  PP: Public parameters
  TW: Tweak
  MSG: Message
  MD: Message Digest
Security of tweakable hashes

• Necessary notion: Single-function multi-target-collision resistance for distinct tweaks

• Intuition:
  • Adversary gets black box access to $T(PP, \cdot, \cdot)$ for random PP.
  • Adversary can adaptively query with restriction to use each tweak only once.
  • Adversary receives PP and has to find second-preimage for one of its previous queries (such that PP and TW are the same).

• For length-preserving case: A property to guarantees that a preimage finder outputs a „new preimage“. E.g., assume that T is almost regular. (Subject of ongoing research)
Instantiating the tweakable hash (for SHA2)

XMSS
• $K = \text{SHA2}(\text{pad}(PP) || TW)$, $BM = \text{SHA2}(\text{pad}(PP) || TW+1)$, $MD = \text{SHA2}(\text{pad}(K) || MSG \oplus BM)$
• Standard model proof if $K$ & $BM$ were random,
• (Q)ROM proof when generating $K$ & $BM$ as above (modeling those SHA2 invocations as RO)
• Tight proof is currently under revision

LMS
• $MD = \text{SHA2}(PP || TW || MSG)$
• QROM proof assuming SHA2 is QRO
• ROM proof assuming SHA2 compression function is RO
• Proofs are essentially tight
Instantiating the tweakable hash

• LMS is factor 3 faster but leads to slightly larger signatures at same security level
• LMS makes somewhat stronger assumptions about the security properties of the used hash function
• More research on direct constructions needed
Stateless hash based signatures
SPHINCS
About the statefulness

• Works great for some settings

• However....
  ... back-up
  ... multi-threading
  ... load-balancing
ELIMINATE THE STATE
SPHINCS (2015)

• Stateless Scheme
• XMSS$^\text{MT}$ + HORST + (pseudo-)random index
• Collision-resilient
• Deterministic signing
• SPHINCS-256:
  • 128-bit post-quantum secure
  • Hundred of signatures / sec
  • 41 kb signature
  • 1 kb keys
Few-Time Signature Schemes
HORS [RR02]

Message M, OWF H, CRHF H’ = n bit
Parameters $t=2^a, k$, with $m = ka$ (typical $a=16, k=32$)

```
<table>
<thead>
<tr>
<th>SK</th>
<th>PK</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sk_1$</td>
<td>$pk_1$</td>
</tr>
<tr>
<td>$sk_2$</td>
<td>$pk_1$</td>
</tr>
<tr>
<td>$sk_t$</td>
<td>$pk_t$</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>SK</th>
<th>PK</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sk_{t-1}$</td>
<td>$pk_{t-1}$</td>
</tr>
<tr>
<td>$sk_t$</td>
<td>$pk_t$</td>
</tr>
</tbody>
</table>
```
HORS mapping function

Message $M$, OWF $H$, CRHF $H'$

Parameters $t = 2^a, k$, with $m = ka$ (typical $a = 16, k = 32$)
HORS

Message M, OWF H, CRHF H’
Parameters $t = 2^a, k$, with $m = ka$ (typical $a = 16, k = 32$)

SK

PK

$H'(M)$

Sig

https://sphincs.org
HORS Security

• $M$ mapped to $k$ element index set $M^i \in \{1, ..., t\}^k$
• Each signature publishes $k$ out of $t$ secrets
• Either break one-wayness or...

• $r$-Subset-Resilience: After seeing index sets $M^i_j$ for $r$ messages $msg_j$, $1 \leq j \leq r$, hard to find $msg_{r+1} \neq msg_j$ such that $M^i_{r+1} \in \bigcup_{1 \leq j \leq r} M^i_j$.

• Best generic attack: $\text{Succ}_{r-SSR}(A,q) = q(rk/ t)^k$
→ Security shrinks with each signature!
HORST

Using HORS with MSS requires adding PK ($tn$ bits) to MSS signature. (SPHINCS-256: $n = 256, t = 2^{16}, k = 32$)

HORST: Merkle Tree on top of HORS-PK
- New PK = Root
- Publish Auth-Paths for HORS signature values
- PK can be computed from Sig

- With optimizations: $tn \rightarrow (k (\log t - x + 1) + 2^x)n$
  - E.g. SPHINCS-256: 2 MB $\rightarrow$ 16 KB
- Use randomized message hash

https://sphincs.org
Sphinx+ Joint work with Daniel J. Bernstein, Christoph Dobrainsig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe
SPHINCS\(^+\) (our NIST submission)

- Strengthened security gives smaller signatures
- Collision- and multi-target attack resilient (XMSS tweakable hash)
- Fixed length signatures (far easier to compute than Octopus (\(\rightarrow\) Gravity-SPHINCS))
- Small keys, medium size signatures (lv 3: 17kB)
- Sizes can be much smaller if q\_sign gets reduced
- THE conservative choice
FORS (Forest of random subsets)

- Parameters $t, a = \log t, k$ such that $ka = m$
Verifiable index selection
(and optionally non-deterministic randomness)

• SPHINCS:

\[(\text{idx}||\mathbf{R}) = \text{PRF}(\mathbf{SK}. \text{prf}, \mathbf{M})\]
\[\text{md} = H_{\text{msg}} (\mathbf{R}, \mathbf{PK}, \mathbf{M})\]

• SPHINCS\(^+\):

\[\mathbf{R} = \text{PRF}(\mathbf{SK}. \text{prf}, \text{OptRand}, \mathbf{M})\]
\[(\text{md}||\text{idx}) = H_{\text{msg}} (\mathbf{R}, \mathbf{PK}, \mathbf{M})\]

https://sphincs.org
Verifiable index selection

Implements FORS security

- SPHINCS: Attacks could target “weakest” HORST key pair
- SPHINCS*: Every hash query ALSO selects FORS key pair
  - Leads to notion of interleaved target subset resilience
Instantiations

• SPHINCS⁺-SHAKE256

• SPHINCS⁺-SHA-256

• SPHINCS⁺-Haraka
# Instantiations (small vs fast)

<table>
<thead>
<tr>
<th></th>
<th>$n$</th>
<th>$h$</th>
<th>$d$</th>
<th>$\log(t)$</th>
<th>$k$</th>
<th>$w$</th>
<th>bitsec</th>
<th>sec level</th>
<th>sig bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPHINCS$^+$-128s</td>
<td>16</td>
<td>64</td>
<td>8</td>
<td>15</td>
<td>10</td>
<td>16</td>
<td>133</td>
<td>1</td>
<td>8080</td>
</tr>
<tr>
<td>SPHINCS$^+$-128f</td>
<td>16</td>
<td>60</td>
<td>20</td>
<td>9</td>
<td>30</td>
<td>16</td>
<td>128</td>
<td>1</td>
<td>16976</td>
</tr>
<tr>
<td>SPHINCS$^+$-192s</td>
<td>24</td>
<td>64</td>
<td>8</td>
<td>16</td>
<td>14</td>
<td>16</td>
<td>196</td>
<td>3</td>
<td>17064</td>
</tr>
<tr>
<td>SPHINCS$^+$-192f</td>
<td>24</td>
<td>66</td>
<td>22</td>
<td>8</td>
<td>33</td>
<td>16</td>
<td>194</td>
<td>3</td>
<td>35664</td>
</tr>
<tr>
<td>SPHINCS$^+$-256s</td>
<td>32</td>
<td>64</td>
<td>8</td>
<td>14</td>
<td>22</td>
<td>16</td>
<td>255</td>
<td>5</td>
<td>29792</td>
</tr>
<tr>
<td>SPHINCS$^+$-256f</td>
<td>32</td>
<td>68</td>
<td>17</td>
<td>10</td>
<td>30</td>
<td>16</td>
<td>254</td>
<td>5</td>
<td>49216</td>
</tr>
</tbody>
</table>
Comparison to SPHINCS-128 at same security level

<table>
<thead>
<tr>
<th></th>
<th>Signing</th>
<th>Verifying</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>median cycles</td>
<td>median cycles</td>
<td>bytes</td>
</tr>
<tr>
<td><strong>SPHINCS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$^+$ (n = 24, h = 55, d = 11, b = 8, k = 30, w = 16)</td>
<td>67 017 940</td>
<td>1 911 684</td>
<td>21 288</td>
</tr>
<tr>
<td><strong>SPHINCS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$^+$ (n = 24, h = 51, d = 17, b = 9, k = 30, w = 16)</td>
<td>40 117 282</td>
<td>2 724 094</td>
<td>29 256</td>
</tr>
<tr>
<td><strong>SPHINCS-128</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n=32, h=60, d=12, t=2^16, k=32, w = 16)</td>
<td>51 636 372</td>
<td>1 451 004</td>
<td>41 000</td>
</tr>
</tbody>
</table>
Hash-based Signatures in NIST „Competition“

• SPHINCS+
  • FORS as few-time signature
  • XMSS-T tweakable hash

• Gravity-SPHINCS
  • PORS as few-time signature
  • Requires collision-resistance -> no tweakable hash

• (PICNIC)
Conclusion

• Standardized hash-based signatures are available
  • Both got their pros & cons (independent comparison would be nice)
  • If you can deal with state: Please play around (and tell us about)

• For stateless
  • A lot of trade-offs
  • Need requirements to fine-tune
  • Further comparison is needed (not easy to compare security)
  • Fault-attacks (and mitigations) need further attention

• Tweakable hash functions
  • Might be interesting also in other settings
  • Dedicated constructions would be nice
Thank you!

Questions?

For references, literature & longer lectures see https://huelsing.net