Hash-based Signatures and upcoming topics

Andreas Hülsing
Eindhoven University of Technology

ESA Workshop on
"Secure Communications for Space Missions in the Post-Quantum Era"
June 27, 2022
Outline

Part I – HBS

1. Background
2. Standardization
   • XMSS & LMS
   • SPHINCS$^+$
3. Recent results
   • Updated security proof

Part II - Next steps

1. Formally verified proofs
2. From primitives to communication
   • PQ-WireGuard
   • PQNoise
3. QROM Security
4. A note on QKD
Hash-based Signature Schemes
[Mer89]

- Conservative choice
- Only secure hash function needed
- First standardized PQC signatures (XMSS, LMS)
- One NIST round 3 scheme (SPHINCS\textsuperscript{+})
- Fast verification
- Small keys
RSA – EC-DSA – Falcon - Dilithium...

Intractability Assumption

Cryptographic hash function

Digital signature scheme

RSA, LH, SVP, MQ...
Hash-based One-time Signatures

[Lam79]

Spy wants to signal HQ that they safely arrived in the field
Hash-based One-time Signatures

When arrived safely, send $z = x$.

If $H(z) = y$
   
   Spy arrived

Else
   
   Problems
Hash-based One-time Signatures

\[y_1 = H(x_1)\]
\[y_2 = H(x_2)\]

If they are attacking, send \(z = x_1\).
If they are pulling back, send \(z = x_2\).

If \(H(z) = y_1\)
   \(They\ are\ attacking!\)
Else if \(H(z) = y_2\)
   \(It's\ safe\ to\ send\ buses.\)
Else
\(Something\ went\ wrong\)
Hash-based One-time Signatures
[Lam79]

• Send n pairs --> Allows to sign hashes of length n
• Can be optimized using "hash chains" (WOTS)
• See my homepage for detailed summer school talks
Merkle’s Hash-based Signatures

\[ \text{SIG} = (i=2, \text{OTS}_) \]
Generic Security

<table>
<thead>
<tr>
<th></th>
<th>OW</th>
<th>SPR</th>
<th>CR</th>
<th>UD</th>
<th>PRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>(\Theta(2^n))</td>
<td>(\Theta(2^n))</td>
<td>(\Theta(2^{n/2}))</td>
<td>(\Theta(2^n))</td>
<td>(\Theta(2^n))</td>
</tr>
<tr>
<td>Quantum</td>
<td>(\Theta(2^{n/2}))</td>
<td>(\Theta(2^{n/2}))</td>
<td>(\Theta(2^{n/3}))</td>
<td>(\Theta(2^{n/2}))</td>
<td>(\Theta(2^{n/2}))</td>
</tr>
</tbody>
</table>

Can be tightly related to very basic quantum distinguishing problems over boolean functions.
Categorizing hash-based signatures
Stateful vs Stateless

**Stateful (LMS & XMSS)**
- Secret key has to be updated after every signature
- Requires advanced & secure key management:
  - No copies == no backup!
  - No parallel use (of same key)
- Fast & small

**Stateless (SPHINCS / SPHINCS*)**
- Secret key stays the same
- Key management like today
- Penalty in performance
Multi-target attacks

• WOTS & Lamport need hash function $h$ to be one-way

• Hypertree 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 lose 66 bits of security! (33 bits quantum)
Multi-target attacks: Mitigation

• Mitigation: Separate targets [HRS16]

• Common approach:
  • In addition to hash function description and „input“ take
    • Hash „Address“ (uniqueness in key pair)
    • Hash „key“ used for all hashes of one key pair (uniqueness among key pairs)
Multi-target attacks: Mitigation

• Mitigation: Separate targets [HRS16]
• Common approach:
  • In addition to hash function description and „input“ take
    • Hash „Address“ (uniqueness in key pair)
    • Hash „key“ used for all hashes of one key pair (uniqueness among key pairs)
New intermediate abstraction: Tweakable Hash Function

- Tweakable Hash Function:

\[ \text{Th}(P, T, M) \rightarrow MD \]

P: Public parameters (one per key pair)
T: Tweak (one per hash call)
M: Message
MD: Message Digest

Security properties are determined by instantiation of tweakable hash!
# Categorizing HBS

<table>
<thead>
<tr>
<th>Constructing tweakable HF assuming SHA2 / SHA3 are good keyed HF</th>
<th>Stateful</th>
<th>Stateless</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMSS / Pyramid-robust</td>
<td>SPHINCS / SPHINCS⁺-robust</td>
<td></td>
</tr>
</tbody>
</table>

| Assuming SHA2 / SHA3 is a good tweakable HF | LMS / Pyramid-simple | SPHINCS⁺-simple |
The stateful schemes

**Abstract**

This note describes the eXtended Merkle Signature Scheme (XMSS), a hash-based digital signature system that is based on existing descriptions in scientific literature. This note specifies Winternitz One-Time Signature Plus (WOTS+), a one-time signature scheme; XMSS, a single-tree scheme; and XMSS\(^*\)MT, a multi-tree variant of XMSS. Both XMSS and XMSS\(^*\)MT use WOTS+ as a main building block. XMSS provides cryptographic digital signatures without relying on the conjectured hardness of mathematical problems. Instead, it is proven that it only relies on the properties of cryptographic hash functions. XMSS provides strong security guarantees and is even secure when the collision resistance of the underlying hash function is broken. It is suitable for compact implementations, is relatively simple to implement, and naturally resists side-channel attacks. Unlike most other signature systems, hash-based signatures can so far withstand known attacks using quantum computers.

This document is a product of the Crypto Forum Research Group (CFRG) in the IRTF. This has been reviewed by many researchers, both in the research group and outside of it. The Acknowledgements section lists many of them.
LMS vs XMSS

• Difference mostly limited to "tweakable hash function"
• LMS is factor 3-4 faster
• LMS makes somewhat stronger assumptions about the security properties of the used hash function
• More research on construction of "tweakable hash functions" needed
## XMSS / XMSS-T Implementation

### C Implementation, using OpenSSL [HRS16]

<table>
<thead>
<tr>
<th></th>
<th>Sign (ms)</th>
<th>Signature (kB)</th>
<th>Public Key (kB)</th>
<th>Secret Key (kB)</th>
<th>Bit Security classical/quantum</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMSS</td>
<td>3.24</td>
<td>2.8</td>
<td>1.3</td>
<td>2.2</td>
<td>236 / 118</td>
<td>h = 20, d = 1,</td>
</tr>
<tr>
<td>XMSS-T</td>
<td>9.48</td>
<td>2.8</td>
<td>0.064</td>
<td>2.2</td>
<td>256 / 128</td>
<td>h = 20, d = 1,</td>
</tr>
<tr>
<td>XMSS</td>
<td>3.59</td>
<td>8.3</td>
<td>1.3</td>
<td>14.6</td>
<td>196 / 98</td>
<td>h = 60, d = 3,</td>
</tr>
<tr>
<td>XMSS-T</td>
<td>10.54</td>
<td>8.3</td>
<td>0.064</td>
<td>14.6</td>
<td>256 / 128</td>
<td>h = 60, d = 3,</td>
</tr>
</tbody>
</table>

Intel(R) Core(TM) i7 CPU @ 3.50GHz
XMSS-T uses message digest from Internet-Draft
All using SHA2-256, w = 16 and k = 2

27/06/2022  https://huelsing.net
LMS / XMSS performance

[LMS vs XMSS: Comparison of two Hash-Based Signature Standards Panos Kampanakis & Scott Fluhrer, eprint 2017-349]

<table>
<thead>
<tr>
<th>Operation</th>
<th>LMS</th>
<th>XMSS</th>
<th>XMSS / LMS ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMSSMT_SHA2-256_W16_H20_D2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK Gen</td>
<td>0.89 s</td>
<td>3.26 s</td>
<td>3.66</td>
</tr>
<tr>
<td>Sign</td>
<td>1.21 ms</td>
<td>4.72 ms</td>
<td>3.90</td>
</tr>
<tr>
<td>Verify</td>
<td>0.339 ms</td>
<td>1.76 ms</td>
<td>5.19</td>
</tr>
<tr>
<td>XMSSMT_SHA2-256_W16_H40_D2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK Gen</td>
<td>720 s</td>
<td>3340 s</td>
<td>4.64</td>
</tr>
<tr>
<td>Sign</td>
<td>1.91 ms</td>
<td>7.70 ms</td>
<td>4.03</td>
</tr>
<tr>
<td>Verify</td>
<td>0.350 ms</td>
<td>1.75 ms</td>
<td>5.00</td>
</tr>
</tbody>
</table>

Table 6: Measured time per operation for LMS and XMSS

Machine unknown. Guess: general purpose CPU from ~2017
LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4.
Fabio Campos, Tim Kohlstadt, Steffen Reith, and Marc Stöttinger. Africacrypt '20

<table>
<thead>
<tr>
<th></th>
<th>LMS</th>
<th>XMSS_ROBUST</th>
<th>ratio&lt;sup&gt;a&lt;/sup&gt;</th>
<th>XMSS_SIMPLE</th>
<th>ratio&lt;sup&gt;b&lt;/sup&gt;</th>
<th>XMSS_SIMPLE+PRE</th>
<th>ratio&lt;sup&gt;c&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>key gen&lt;sup&gt;d&lt;/sup&gt;</td>
<td>3774.88</td>
<td>23631.70</td>
<td>6.26</td>
<td>7792.23</td>
<td>2.06</td>
<td>7586.15</td>
<td>2.01</td>
</tr>
<tr>
<td>sign&lt;sup&gt;d&lt;/sup&gt;</td>
<td>3791.15</td>
<td>23642.03</td>
<td>6.23</td>
<td>7796.39</td>
<td>2.05</td>
<td>7596.24</td>
<td>2.00</td>
</tr>
<tr>
<td>verify&lt;sup&gt;d&lt;/sup&gt;</td>
<td>2.65</td>
<td>13.07</td>
<td>4.93</td>
<td>3.57</td>
<td>1.34</td>
<td></td>
<td>4.20</td>
</tr>
</tbody>
</table>

<sup>a</sup> XMSS_ROBUST/LMS  
<sup>b</sup> XMSS_SIMPLE/LMS  
<sup>c</sup> XMSS_SIMPLE+PRE/LMS  
<sup>d</sup> All results (apart from ratio) are given in 10^6 clock cycles.
SPHINCS
Stateless hash-based signatures

[NY89, Gol87, Gol04]

Goldreich’s approach [Gol04]:
Security parameter $\lambda = 128$
Use binary tree as in Merkle, but...

• ...for security
  • pick index $i$ at random;
  • requires huge tree to avoid index collisions (e.g., height $h = 2\lambda = 256$).

• ...for efficiency:
  • use binary certification tree of OTS key pairs (= Hypertree with $d = h$),
  • all OTS secret keys are generated pseudorandomly.
SPHINCS \([BHH+15]\)

- Select index pseudo-randomly
- Use a few-time signature key-pair on leaves to sign messages
  - Few index collisions allowed
  - Allows to reduce tree height
- Use hypertree: Use \(d \ll h\).
SPHINCS+

Joint work with Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, 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, Bas Westerbaan
Instantiations (small vs fast)

Table 3: Example parameter sets for SPHINCS$^+$ targeting different security levels and different tradeoffs between size and speed. Note that these parameter sets have been update for round 3. The column labeled “bitsec” gives the bit security computed as described in Section 9; the column labeled “sec level” gives the security level according to the levels specified in Section 4.A.5 of the Call for Proposals. As explained later, for Haraka the security level is limited to 2: i.e., it is 1 for $n = 16$, and 2 for $n = 24$ or $n = 32$.

<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>63</td>
<td>7</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>133</td>
<td>1</td>
<td>7 856</td>
</tr>
<tr>
<td>SPHINCS$^+$-128f</td>
<td>16</td>
<td>66</td>
<td>22</td>
<td>6</td>
<td>33</td>
<td>16</td>
<td>128</td>
<td>1</td>
<td>17 088</td>
</tr>
<tr>
<td>SPHINCS$^+$-192s</td>
<td>24</td>
<td>63</td>
<td>7</td>
<td>14</td>
<td>17</td>
<td>16</td>
<td>193</td>
<td>3</td>
<td>16 224</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>35 664</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>29 792</td>
</tr>
<tr>
<td>SPHINCS$^+$-256f</td>
<td>32</td>
<td>68</td>
<td>17</td>
<td>9</td>
<td>35</td>
<td>16</td>
<td>255</td>
<td>5</td>
<td>49 856</td>
</tr>
</tbody>
</table>
# Speed

<table>
<thead>
<tr>
<th></th>
<th>key generation (μs)</th>
<th>signing (μs)</th>
<th>verification (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPHINCShASHA2-128s-simple</td>
<td>84 964 790</td>
<td>644 740 090</td>
<td>861 478</td>
</tr>
<tr>
<td>SPHINCShASHA2-128s-robust</td>
<td>175 257 460</td>
<td>1 328 848 352</td>
<td>1 827 104</td>
</tr>
<tr>
<td>SPHINCShASHA2-128fsimple</td>
<td>1 334 220</td>
<td>33 651 546</td>
<td>2 150 290</td>
</tr>
<tr>
<td>SPHINCShASHA2-128f-robust</td>
<td>2 748 026</td>
<td>68 541 846</td>
<td>4 801 338</td>
</tr>
<tr>
<td>SPHINCShASHA2-192s-simple</td>
<td>125 310 788</td>
<td>1 246 378 060</td>
<td>1 444 030</td>
</tr>
<tr>
<td>SPHINCShASHA2-192s-robust</td>
<td>260 903 972</td>
<td>2 517 396 082</td>
<td>3 103 732</td>
</tr>
<tr>
<td>SPHINCShASHA2-192fsimple</td>
<td>1 928 970</td>
<td>55 320 742</td>
<td>3 492 210</td>
</tr>
<tr>
<td>SPHINCShASHA2-192f-robust</td>
<td>4 063 066</td>
<td>113 484 456</td>
<td>7 552 358</td>
</tr>
<tr>
<td>SPHINCShASHA2-256s-simple</td>
<td>80 943 202</td>
<td>1 025 721 040</td>
<td>1 986 974</td>
</tr>
<tr>
<td>SPHINCShASHA2-256s-robust</td>
<td>339 101 780</td>
<td>3 912 132 754</td>
<td>8 294 732</td>
</tr>
<tr>
<td>SPHINCShASHA2-256fsimple</td>
<td>5 067 546</td>
<td>109 104 452</td>
<td>3 559 052</td>
</tr>
<tr>
<td>SPHINCShASHA2-256f-robust</td>
<td>21 327 470</td>
<td>435 984 168</td>
<td>14 938 510</td>
</tr>
</tbody>
</table>

Table 6: Runtime benchmarks for SPHINCShASHA2 on AVX2

Single core of a 3.1 GHz Intel Xeon E3-1220 CPU (Haswell)
Recent results
New tight security analysis
with Mike Kudinov (eprint)

• Tight security analysis from weaker properties [HRS16] was flawed
• No attack / non-tight proof not concerned
• New tight security proof
Part II - Next steps
Formally verified proofs
with Matthias Meijers

• Proofs of security for PQC
  • are more complicated
  • make use of new tools
  • fewer knowledgeable reviewers

• Consequences
  • More error prone
  • Reduced confidence

• Solution:
  • Use computers to verify computational proofs
  • Reduces verification effort to verification of statements
Formally verified security of Saber
with Matthias Meijers & Pierre-Yves Strub (CRYPTO '22 )

Saber

• Lattice-based KEM
• Finalist in NIST competition

Formal analysis of

• CPA-security proof for PKE, and
• correctness proof using EasyCrypt
• sufficient to apply FO-transform
Formal verification of HBS proofs
with Matthias Meijers & several more people from the Formosa team

• Formal verification using EasyCrypt

• Progress:
  • Traditional MSS scheme (schoolbook version) already verified
  • XMSS: Close to being finished (also covers critical proof part for SPHINCS+)
  • SPHINCS+: Starting

• Far more advanced than Saber proof
From PQ-secure primitives to PQ-secure communication
with Florian Weber & many more

- Process so far largely limited to new primitives
- This is cutting short
  - No plug'n'play for DH
  - Changed performance
  - Classical analysis does not necessarily apply
- We need new / updated protocols
From PQ-secure primitives to PQ-secure communication

with Florian Weber & many more

PQ-WireGuard (S&P 21)
- PQ variant of the VPN protocol
- Symbolic analysis using Tamarin
- Computational security proof
- Optimized implementation using Classic McEliece & Dagger

PQNoise (eprint)
- Generic AKE framework
- Using KEMs in place of DH
- Generic computational security proof for arbitrary patterns
QROM security
with Kathrin Hövelmanns & many more

• Practical schemes & protocols are mostly analyzed in ROM
• Considering quantum computers, we must move to quantum accessible ROM (QROM)
• Existing proof tools are still falling behind classical counterparts
• Recent results:
  • Adaptive reprogramming for the QROM [Asiacrypt 2021]
  • New One-way to hiding lemmas [TCC 2019 & eprint]
A note on QKD
(or lasers are cool, but...)

• Quantum computers break our solutions for the key exchange problem:
  • two parties that never met and need a shared secret key

• QKD solves a different problem:
  • Two parties that already share a secret key can generate more shared secret keying material
Conclusion

• HBS are a suitable alternative if you
  • can handle state, or
  • can live with suboptimal performance

• There are more topics to solve
  • Verify our proofs
  • Make our protocols post-quantum secure
  • Develop the tools to do so
Thank you!

Questions?

For references & further literature see
https://huelsing.net/wordpress/?page_id=165