Hash Based Signatures

A Survey

Scott Fluhrer
Principal Engineer
September 15, 2017
Why Hash Based Signatures?

Confidence in its security
As strong as the hash function
Postquantum (assuming the hash function is postquantum)
Basics of Hash Based Signatures
Merkle Signatures

Overall structure
Multilevel Merkle Trees

Can sign billions of messages
Stateful Hash Based Signatures
Stateful Signature Schemes

This is a straightforward implementation of a Merkle tree.

We use one bottom level OTS leaf to sign each message.
Problem with Stateful Schemes

• State Management
  You must never use the same index to issue two different signatures
  That means that you must remember which you used
  What if the disk write fails?
  Managing multiple signers are problematic
  Issues with VM state cloning
LMS vs XMSS

These are two different proposals for doing stateful signatures.

They are mostly similar (signature sizes, public key sizes, etc), however there are two major differences between the two...
In XMSS, they randomize each hash that they perform.

Because XMSS always hashes random values, if you can generate an XMSS forgery, you must be able to generate (second) preimages.
In LMS, they prepend a unique hash identifier

The proof shows that, if the Merkle Damgård compression function acts randomly, then the probability of the attacker finding a forgery is tiny.
LMS vs XMSS

Performance

XMSS takes 3x–5x more time for every operation

(*) Sign and Verify benchmarks assumes short messages
Bottom Line

• XMSS has a stronger security model
• LMS is much faster (for short messages)

Is there a Real Difference?

• Not if you’re happy with the LMS security model
• Not if the messages you’re signing are large
Stateless Hash Based Signatures – SPHINCS
Stateless Hash Based Signatures

SPHINCS

XMSS and LMS has the disadvantage that we need to track state.

SPHINCS is a hash based signature method that doesn’t require that.
SPHINCS

Internal Organization

Hypertree

HORST

H = 60
SPHINCS Pluses and Minuses

Pluses

• No more state
• Looks like a traditional signature method
• Up to $2^{50}$ signatures per public key (before security of private key starts to erode)

Minuses

• Large signatures (41000 bytes)
• Relatively large signature generation time
Hybrid Hash Based Signatures
Intermediate Approach: Hybrid Signatures

The problem with state is keeping track of it long-term. What if we kept all state in memory?
The Hybrid Approach

Key Generation
- Public key
  - On Disk

Key Expansion
- Private key
  - In Memory

Signature Generation
- Running state
  - Signatures
What does this gain us?

We have less concern about state

• No issues about not being able to write the update to the disk
• No issue with multiple signers
• No issues about cloning, *as long as the clone doesn’t copy RAM*

Signatures can also be shorter and faster to generate
## The Trade-Offs

<table>
<thead>
<tr>
<th>XMSS/LMS</th>
<th>Hybrid</th>
<th>SPHINCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smaller signatures</td>
<td>Intermediate sized signatures</td>
<td>Large signatures</td>
</tr>
<tr>
<td>Fast siggen</td>
<td>Less problems with state</td>
<td>Slow siggen</td>
</tr>
<tr>
<td>Problems with long term state</td>
<td>Not fully defined</td>
<td>Practically perfect otherwise</td>
</tr>
</tbody>
</table>

**XMSS/LMS**
- Smaller signatures
- Fast siggen
- Problems with long term state

**Hybrid**
- Intermediate sized signatures
- Less problems with state
- Not fully defined

**SPHINCS**
- Large signatures
- Slow siggen
- Practically perfect otherwise