SPHINCS+C
Compressing SPHINCS+ With (Almost) No Cost

Andreas Hülsing, Mikhail Kudinov, Eyal Ronen, Eylon Yogev
SPHINCS+

- A hash-based signature scheme
- One of the selected NIST PQ digital schemes
  - One of the most secure and robust schemes
- Has a “small” and “fast” variant (for each security level)
- Actually allows a wide range of tradeoffs between:
  - Sig-Size, Sig-Gen-Time, Sig-Ver-Time
  - In general: faster Sig-Gen-Time and Sig-Ver-Time -> larger Sig-Size
**SPHINCS+C**

- Same structure with new more efficient primitives: WOTS+C, FORS+C
- Minor code changes compared to SPHINCS+
- Allows a new realm of better tradeoffs than SPHINCS+:
  - E.g., smaller Sig-Size with same Sig-Gen-Time

<table>
<thead>
<tr>
<th>Security Level</th>
<th>Small Signature Size</th>
<th>Fast Signature Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPHINCS+</td>
<td>SPHINCS+C</td>
</tr>
<tr>
<td>128-bit</td>
<td>7856</td>
<td>6304 (−20%)</td>
</tr>
<tr>
<td>192-bit</td>
<td>16224</td>
<td>13776 (−16%)</td>
</tr>
<tr>
<td>256-bit</td>
<td>29792</td>
<td>26096 (−13%)</td>
</tr>
</tbody>
</table>
Intuition for WOTS+C and FORS+C

- **Hash and Sign**
  - $d = HASH(r|m)$, $m \in \{0,1\}^*$, $d \in \{0,1\}^n$, $r \leftarrow \$$
  - $\sigma = SIGN(d)$

- **SIGN accepts any value of $d$**
  - SIGN may be “compressed” (size, run time) for some sub-domains

- **Basic idea**
  - Find “good” sub-domain $D_C$ and compress SIGN/VER for it
  - For signing, we add an incrementing counter to the hash
  - Search for cnt value such that $d_c = HASH(r||cnt||m) \in D_C$
  - $\sigma_c = SIGN_C(d_c)||cnt$
  - Verifier checks that $HASH(r||cnt||m) \in D_C$ and $VER_C(d, \sigma) = 1$
Intuition for WOTS+C and FORS+C

- Wait a minute, the Sig-Gen running time is not constant!
  - Yes, but this is actually OK

- Is this secure against side-channel attacks?
  - Yes, if original SPHINCS+ is “Constant time” crypto then so is SPHINCS+C
  - “Constant time” means independence of running time and secret inputs
  - Our run time variance only depends on the message and public values
  - Variance is independent from any secret values and doesn't leak any information about them
Intuition for WOTS+C and FORS+C

- Won't some signatures take a really long time?
  - We bound the probability $p$ for a signature will run more than $f(p)$ time the expected time.
  - E.g., $f(2^{-16}) < 3$, $f(2^{-32}) < 5$ and $f(2^{-64}) < 9$
  - Can optimize for parameter sets with lower variance.
    - E.g., for SPHINCS+C-256f $f(2^{-64}) < 1.2$

<table>
<thead>
<tr>
<th></th>
<th>expected hash calls</th>
<th>log($t'$)</th>
<th>2−8</th>
<th>f(p) for probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPHINCS+C-128s</td>
<td>$2^{20.9}$</td>
<td>18</td>
<td>1.6</td>
<td>2.3 3.1 3.8 4.5 5.3 6.0 6.7</td>
</tr>
<tr>
<td>SPHINCS+C-128f</td>
<td>$2^{16.7}$</td>
<td>8</td>
<td>1.0</td>
<td>1.0 1.1 1.1 1.1 1.1 1.1 1.1</td>
</tr>
<tr>
<td>SPHINCS+C-192s</td>
<td>$2^{21.7}$</td>
<td>12</td>
<td>1.0</td>
<td>1.0 1.1 1.1 1.1 1.1 1.1 1.1</td>
</tr>
<tr>
<td>SPHINCS+C-192f</td>
<td>$2^{17.4}$</td>
<td>13</td>
<td>1.2</td>
<td>1.5 1.8 2.0 2.3 2.6 2.9 3.1</td>
</tr>
<tr>
<td>SPHINCS+C-256s</td>
<td>$2^{21.5}$</td>
<td>19</td>
<td>1.8</td>
<td>2.8 3.7 4.7 5.7 6.6 7.6 8.6</td>
</tr>
<tr>
<td>SPHINCS+C-256f</td>
<td>$2^{18.4}$</td>
<td>10</td>
<td>1.0</td>
<td>1.0 1.1 1.1 1.1 1.1 1.1 1.1</td>
</tr>
</tbody>
</table>
Intuition for WOTS+C and FORS+C

- Won’t some signatures take a really long time?
  - We also run experiments to compare variability in SIG-GEN time with SPHINXS+ 
From WOTS+ and FORS to WOTS+C and FORS+C
Function chains in WOTS*

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

Parameter $w$

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

$c^0(x) = x$

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

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

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

\[ \sigma = (\sigma_1, \ldots, \sigma_{\ell}) \]

\[ \sigma = c^{b_1}(sk_1) \]

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

\[ \sigma = c^{b_\ell}(sk_\ell) \]

\[ pk_1 = c^{w-1}(sk_1) \]

\[ pk_\ell = c^{w-1}(sk_\ell) \]
WOTS+C

We remove the checksum chains by forcing it to always a pre-defined value.

**Signing:**

Instead of signing the message $m$, we sign $d = h(s||m)$, where $s$ a salt.

Search for $s$ s.t. $d$ has a checksum $S$, add $s$ to the signature.

$S$ is pre-defined to be the expected checksum.

Signer run-time is usually **reduced**! More work to find salt, but no checksum chains to calculate.

**Verifying:**

Verifier run-time is **reduced**.

No need to verify the checksum chains.

Only compute $d = h(s||m)$ and verify that $d$ has checksum $S$ (and verify the signature).

Can use the same technique to reduce more chains (at the cost of increasing Sig-Gen-Time).
WOTS+C

\[ \text{sk}_1 \rightarrow \cdots \rightarrow a_1 \rightarrow \cdots \rightarrow \text{pk}_1 \]

\[ \text{sk}_2 \rightarrow \cdots \rightarrow a_2 \rightarrow \cdots \rightarrow \text{pk}_2 \]

\[ \vdots \]

\[ \text{sk}_{\text{len}1} \rightarrow \cdots \rightarrow a_{\text{len}1} \rightarrow \cdots \rightarrow \text{pk}_{\text{len}1} \]

\[ \text{sk}_{\text{len}1+1} \rightarrow \cdots \rightarrow a_{\text{len}1+1} \rightarrow \cdots \rightarrow \text{pk}_{\text{len}1+1} \]

\[ \text{sk}_{\text{len}2} \rightarrow \cdots \rightarrow \text{pk}_{\text{len}2} \]

Removed in SPHINCS+C
FORS+C

- FORS includes multiple Merkle trees, opening one leaf in each tree
- Using similar techniques, we remove the last tree of the FORC+ signature
- **Idea:** force the hash for the last tree to always open the first leaf (leaf index 0)
- Find a salt $s$ that satisfies the above
- **Tweak:** we can make the last tree larger than other to gain savings
- Verifier run-time is **reduced** (simply check that last tree has index 0)
SPHINCS+C
SPHINCS+C Parameter Sets

- As a starting point we can use the original SPHINCS+ parameter sets
- This results in a “compressed” version of SPHINCS+ that is strictly better
  - Faster Key-Gen-Time, Sig-Gen-Time, Sig-Ver-Time
  - Smaller Sig-Size

<table>
<thead>
<tr>
<th></th>
<th>Key Generation</th>
<th>Signature</th>
<th>Verification</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPHINCS+</td>
<td>Compressed</td>
<td>SPHINCS+</td>
<td>Compressed</td>
</tr>
<tr>
<td>SHAKE-128s</td>
<td>721.4</td>
<td>649.8 (-10%)</td>
<td>5398.0</td>
<td>4964.5 (-8%)</td>
</tr>
<tr>
<td>SHAKE-128f</td>
<td>10.8</td>
<td>9.7 (-11%)</td>
<td>256.3</td>
<td>232.4 (-9%)</td>
</tr>
<tr>
<td>SHAKE-192s</td>
<td>1068.6</td>
<td>962.3 (-10%)</td>
<td>9133.3</td>
<td>8283.6 (-9%)</td>
</tr>
<tr>
<td>SHAKE-192f</td>
<td>15.1</td>
<td>13.4 (-12%)</td>
<td>380.6</td>
<td>347.6 (-9%)</td>
</tr>
<tr>
<td>SHAKE-256s</td>
<td>652.6</td>
<td>648.2 (-1%)</td>
<td>7573.0</td>
<td>7367.8 (-3%)</td>
</tr>
<tr>
<td>SHAKE-256f</td>
<td>44.4</td>
<td>38.5 (-13%)</td>
<td>860.3</td>
<td>763.4 (-11%)</td>
</tr>
</tbody>
</table>
SPHINCS+C Parameter Sets

- However, we can do better.
- We can optimize parameter sets for different constraints and use cases.
- E.g., we optimized SPHINCS+C parameters to:
  - Minimize Sig-Size
  - Keeping Sig-Gen-Time at least as fast as SPHINCS+

<table>
<thead>
<tr>
<th>Key Generation</th>
<th>Signature</th>
<th>Verification</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPHINCS+</td>
<td>SPHINCS+C</td>
<td>SPHINCS+</td>
</tr>
<tr>
<td>SHAKE-128s</td>
<td>721.1</td>
<td>341.1 (−53%)</td>
<td>5398.0</td>
</tr>
<tr>
<td>SHAKE-128f</td>
<td>10.8</td>
<td>8.6 (−20%)</td>
<td>256.3</td>
</tr>
<tr>
<td>SHAKE-192s</td>
<td>1068.6</td>
<td>501.2 (−53%)</td>
<td>9133.3</td>
</tr>
<tr>
<td>SHAKE-192f</td>
<td>15.1</td>
<td>12.9 (−15%)</td>
<td>380.6</td>
</tr>
<tr>
<td>SHAKE-256s</td>
<td>652.6</td>
<td>432.0 (−34%)</td>
<td>7573.0</td>
</tr>
<tr>
<td>SHAKE-256f</td>
<td>44.4</td>
<td>37.7 (−15%)</td>
<td>860.3</td>
</tr>
</tbody>
</table>
Improved Tradeoff with SPHINCS+C
SPHINCS+C Parameter Sets

- SPHINCS+C can provide better tradeoffs compared to SPHINCS+
- We are looking for feedback on real-world requirements and tradeoffs
  - Sig-Gen-Time Vs. Sig-Ver-Time
  - Sig-Size Vs. Sig-Ver-Time
  - Low $q_{\text{sign}}$ variants
- The paper includes a sage script for finding suitable parameter sets
Future Work

- In the paper we propose other optimization that require bigger code changes and are not included in SPHINCS+C
  - Interleaved Trees for better FORS compression
  - Small trees of FORS+C
  - Soft-state-full variant to XMSS based on a tree of FORS+C
    - Only need to make sure you don’t pass the signature number limit

- Fine-tuning parameter sets choice
Conclusion

● We presented SPHINCS+C a “compressed” variant of SPHINCS+
  ○ Based on WOTS+C and FORS+C variants of WOTS+ and FORS used in SPHINCS+
  ○ Full \textit{tight} security proof as in SPHINCS+
● SPHINCS+C allows for better tradeoffs and optimization of parameter sets
● WOTS+C optimizations can also be used in XMSS
● Improved tradeoffs and optimization also for low \( q_{\text{sign}} \) variants

● Paper available at: ia.cr/2022/778
● Code: https://github.com/eyalr0/sphincsplusC/
● Any questions?