FPGA-BASED ACCELERATOR FOR POST-QUANTUM SIGNATURE SCHEME SPHINCS-256

Dorian Amiet¹, Andreas Curiger² and Paul Zbinden¹
¹ HSR Hochschule für Technik, Rapperswil, Switzerland
² Securosys SA, Zürich, Switzerland
CHES 2018
12.09.2018
Quantum Computer Progress

www.qubitcounter.com
## Impact on Current Algorithms

<table>
<thead>
<tr>
<th>Function</th>
<th>Algorithm</th>
<th>Key length/Hash length (bits)</th>
<th>Security level (bits)</th>
<th>Quantum Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PKI: Signing, Key Exchange...</td>
<td>RSA-3072</td>
<td>3072</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ECC-256</td>
<td>256</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>Symmetric Encryption</td>
<td>AES-128</td>
<td>128</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>256</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>Hash</td>
<td>SHA-256</td>
<td>256</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>SHA3-512</td>
<td>512</td>
<td>512</td>
<td>256</td>
</tr>
</tbody>
</table>
Agenda

- Hash-based signatures
  - OTS (one-time signature)
  - Merkle trees
  - SPHINCS-256
- SPHINCS-256 FPGA implementation
- Adjustments to SPHINCS+
- SPHINCS+ FPGA implementation
- Performance results

New, unpublished results!
Post-Quantum Signature Algorithms…

- ...enable secure signing while an adversary has a quantum computer
- Several approaches:
  - Lattice-based
  - Code-based
  - Supersingular isogeny
  - Others
Post-Quantum Signature Algorithms…

- enable secure signing while an adversary has a quantum computer

- Several approaches:
  - Lattice-based
  - Code-based
  - Supersingular isogeny
  - Others

All signing protocols need a hash function (message digest)
Post-Quantum Signature Algorithms…

- ...enable secure signing while an adversary has a quantum computer

- Several approaches:
  - Lattice-based
  - Code-based
  - Supersingular isogeny
  - Others

  All signing protocols need a hash function (message digest)

- Hash based signature schemes
  - Security relies on hardness of (second-) pre-image attack
  - Cryptanalysis: Hash functions are very well analyzed and understood
  - If hash functions are broken, all signing protocols are broken

  => Simply the most conservative choice in terms of security
Example: OTS with 256 bit security

1. Generate 2x256 random numbers, each 256 bits long
   - $X_{0,0}$, $X_{0,1}$, $X_{2,0}$ ... $X_{255,1}$
   - $X_{i,j}$ = private key

<table>
<thead>
<tr>
<th>rand 0</th>
<th>rand 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{0,0}$</td>
<td>$X_{0,1}$</td>
</tr>
<tr>
<td>$X_{1,0}$</td>
<td>$X_{1,1}$</td>
</tr>
<tr>
<td>$X_{2,0}$</td>
<td>$X_{2,1}$</td>
</tr>
<tr>
<td>$X_{...,0}$</td>
<td>$X_{...,1}$</td>
</tr>
<tr>
<td>$X_{255,0}$</td>
<td>$X_{255,1}$</td>
</tr>
</tbody>
</table>
Lamport One-Time Signature (OTS)

Example: OTS with 256 bit security

1. Generate 2x256 random numbers, each 256 bits long
   - \( X_{0,0}, X_{0,1}, X_{2,0} \ldots X_{255,1} \)
   - \( X_{i,j} = \) private key

2. Calculate all digests from random numbers
   - \( Y_{0,0} = h(X_{0,0}), Y_{0,1} = h(X_{0,1}), \ldots, Y_{255,1} = h(X_{255,1}) \)
   - \( Y_{i,j} = \) public key

<table>
<thead>
<tr>
<th>( X_{0,0} )</th>
<th>( Y_{0,0} )</th>
<th>( X_{0,1} )</th>
<th>( Y_{0,1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_{1,0} )</td>
<td>( Y_{1,0} )</td>
<td>( X_{1,1} )</td>
<td>( Y_{1,1} )</td>
</tr>
<tr>
<td>( X_{2,0} )</td>
<td>( Y_{2,0} )</td>
<td>( X_{2,1} )</td>
<td>( Y_{2,1} )</td>
</tr>
<tr>
<td>( X_{\ldots,0} )</td>
<td>( Y_{\ldots,0} )</td>
<td>( X_{\ldots,1} )</td>
<td>( Y_{\ldots,1} )</td>
</tr>
<tr>
<td>( X_{255,0} )</td>
<td>( Y_{255,0} )</td>
<td>( X_{255,1} )</td>
<td>( Y_{255,1} )</td>
</tr>
</tbody>
</table>
Lamport One-Time Signature (OTS)

Example: OTS with 256 bit security

1. Generate 2x256 random numbers, each 256 bits long
   - $X_{0,0}, X_{0,1}, X_{2,0} \ldots X_{255,1}$
   - $X_{i,j} =$ private key

2. Calculate all digests from random numbers
   - $Y_{0,0} = h(X_{0,0}), Y_{0,1} = h(X_{0,1}), \ldots, Y_{255,1} = h(X_{255,1})$
   - $Y_{i,j} =$ public key

3. Sign:
   1. Calculate digest from message $d = h(m)$
   2. For $i = 0$ to $255$
      1. If $d_i = 0$, then $v_i \leq X_{i,0}$
      2. Else $v_i \leq X_{i,1}$

   $h(m) = 0b010\ldots1$

   $\Rightarrow$ Signature($m$) = ($X_{0,0}, X_{1,1}, X_{2,0}, \ldots, Y_{255,1}$)
W-OTS+ Shorter Signatures for Hash-Based Signature Schemes

- Sign a few bits per random number
- Increases processing time
- Decreases key and signature sizes

\[ h(h(h(x)))) \]
Sign "0b11" (= public key(i))

\[ h(h(x)) \]
Sign "0b10"

\[ h(x) \]
Sign "0b01"

\[ x \]
Sign "0b00" (= private key(i))
Signature system which security is based only on security of hash function

Quantum secure

Very fast

One signature per key pair
Merkle Tree

Public key for 4 signatures

N_{1,1} = h(N_{2,0} || N_{3,0})

N_{3,0} = h(Y_3)

4 W-OTS+ key pairs

Y_0, Y_1, Y_2, Y_3

X_0, X_1, X_2, X_3
Merkle Tree

Public key for 4 signatures

\[ N_{1,1} = h(N_{2,0} || N_{3,0}) \]

\[ N_{3,0} = h(Y_3) \]

4 W-OTS+ key pairs

\[ N_{0,2} \]
\[ N_{0,1} \]
\[ N_{0,0} \]
\[ N_{1,0} \]
\[ N_{2,0} \]
\[ N_{3,0} \]

\[ Y_0 \]
\[ Y_1 \]
\[ Y_2 \]
\[ Y_3 \]

\[ X_0 \]
\[ X_1 \]
\[ X_2 \]
\[ X_3 \]
Merkle Tree

+ Signature system which security is based **only** on security of hash function
+ Quantum secure
+ Fast operations
  - State-based
    => Check-list required: Which W-OTS+ key pairs (leaves of the tree) are already used?
SPHINCS

- Make a hyper-tree (tree of trees)
  - Increases number of leaves dramatically
- Use a FTS (few-time signature) at bottom layer instead of OTS
- Choose starting point at random

Dorian Amiet, FPGA-based Accelerator for SPHINCS-256, CHES 2018, 12.09.2018
- Make a hyper-tree (tree of trees)
  - Increases number of leaves dramatically
- Use a FTS (few-time signature) at bottom layer instead of OTS
- Choose starting point at random

Source: https://sphincs.cr.yp.to/

=> Stateless, practical, hash-based, incredibly nice cryptographic signatures  (SPHINCS)
### SPHINCS-256 Operation Count

<table>
<thead>
<tr>
<th>Function</th>
<th>Signing</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Part</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>HORST</td>
</tr>
<tr>
<td>BLAKE-256</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ChaCha12</td>
<td>0</td>
<td>32,768</td>
</tr>
<tr>
<td>$\Pi_{\text{ChaCha}}$</td>
<td>0</td>
<td>193,410</td>
</tr>
<tr>
<td>BLAKE-512</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
SPHINCS-256 Core Top

Host → I/O RAM → Control unit

addr → 256 instruction

data copy

addr

256

BLAKE (slow)

256

ChaCha12 (highly pipelined)

256

addr

Cache RAM

512
Simple Power Analysis

WOTS root layer 11 = SPHINCS public key

WOTS layer 11 signs WOTS root layer 10
WOTS_A = f(secret key, layer, HORST address)

WOTS layer 1 to 10
WOTS root layer 0

WOTS_C.compresses 32 WOTS_k

14. WOTS round = WOTS_A
WOTS layer 0 signs HORST root
0. WOTS round = WOTS_A

HORST_C.compresses HORST_A within a tree structure into one node of 256 bits

HORST signs the message digest D

public data

secret data

HORST

12x WOTS

SPHINCS-256 Simple Power Analysis

FPGA Core Voltage [V]

Tick [1]

0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4

0 0.5 1.0 1.5 2.0 2.5 × 10^7

Dorian Amiet, FPGA-based Accelerator for SPHINCS-256, CHES 2018, 12.09.2018
SPHINCS+

- Submitted to the NIST post-quantum project

- Some adjustments to SPHINCS-256
  - Few-Time signature is now more efficient (security, processing time, signature size)
  - Change underlying hash function
  - Masks are generated (PRNG) => reduces key sizes

- Several instances
  - Security level 1, 3, and 5 (≈ 128, 192, and 256 bit)
  - Different hash functions
    - SHAKE-256 (SHA-3)
    - SHA-256
    - Haraka
  - Always a fast (larger signature) and a small (slower processing) version
N = 128, 192, or 256
## Performance Results

<table>
<thead>
<tr>
<th>Instance</th>
<th>Sign</th>
<th>FPGA Resources</th>
<th>Sign</th>
<th>Clock</th>
<th>T sign</th>
<th>T verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPHINCS-256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>19k</td>
<td>38k</td>
<td>36</td>
<td>805k</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>525</td>
<td>1.07</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+-SHAKE256-128s</td>
<td>7.9</td>
<td>49k</td>
<td>73k</td>
<td>15.5</td>
<td>5,275k</td>
<td>17.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300*</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+-SHAKE256-128f</td>
<td>16.6</td>
<td>47k</td>
<td>73k</td>
<td>15.5</td>
<td>410k</td>
<td>1.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300*</td>
<td>0.19</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+-SHAKE256-192s</td>
<td>16.7</td>
<td>50k</td>
<td>74k</td>
<td>22.5</td>
<td>9,569k</td>
<td>31.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300*</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+-SHAKE256-192f</td>
<td>34.8</td>
<td>50k</td>
<td>74k</td>
<td>22.5</td>
<td>530k</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300*</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+-SHAKE256-256s</td>
<td>29.1</td>
<td>50k</td>
<td>76k</td>
<td>30</td>
<td>9,025k</td>
<td>30.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300*</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+-SHAKE256-256f</td>
<td>48</td>
<td>52k</td>
<td>76k</td>
<td>30</td>
<td>1,169k</td>
<td>3.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>300*</td>
<td>0.28</td>
<td></td>
</tr>
</tbody>
</table>

*Clock frequency of SHAKE-256 pipeline runs at 600 MHz

All results are related to Xilinx Kintex-7 device (XC7K325T-2)
Performance Comparison

Dorian Amiet, FPGA-based Accelerator for SPHINCS-256, CHES 2018, 12.09.2018
FPGA Implementation SPHINCS-256
- >600 sign/s, >15000 verifications/s for SPHINCS-256

FPGA Implementation SPHINCS+-SHAKE256-128f
- >700 sign/s, >5000 verifications/s for

SPA: Protected

DPA: Robust
- We tried hard, but could not extract any key bits.
Thank you

This work was supported by Innosuisse
Why is SPHINCS+ slower than SPHINCS-256?

- Factor two is lost due to the mask computation
- The hash function SHAKE-256 needs more computational effort than ChaCha12
- L-tree computation is faster than the calculation of SHAKE-256 with a long input.
  - The latter holds only for our highly pipelined FPGA implementation and is caused by pipeline stalls.
## Implementation Results

<table>
<thead>
<tr>
<th>Ref</th>
<th>Scheme</th>
<th>Security Classic</th>
<th>Security PQ</th>
<th>FPGA</th>
<th>Area LUT/FF/DSP/BRAM</th>
<th>f MHz</th>
<th>t ms</th>
<th>t·area s·LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
<td>SPHINCS-256</td>
<td>256</td>
<td>128</td>
<td>K7</td>
<td>19,067/38,132/3/36</td>
<td>525</td>
<td>1.53</td>
<td>29.4</td>
</tr>
<tr>
<td>[PDG14]</td>
<td>BLISS-IV</td>
<td>192</td>
<td>?</td>
<td>S6</td>
<td>6,438/6,198/5/7</td>
<td>135</td>
<td>0.35</td>
<td>2.25</td>
</tr>
<tr>
<td>[ACZ16]</td>
<td>ECDSA-256</td>
<td>128</td>
<td>0</td>
<td>V7</td>
<td>6,816/4,442/20/0</td>
<td>225</td>
<td>1.49</td>
<td>10.2</td>
</tr>
<tr>
<td>[ACZ16]</td>
<td>ECDSA-521</td>
<td>256</td>
<td>0</td>
<td>V7</td>
<td>8,273/7,689/64/0</td>
<td>161</td>
<td>5.02</td>
<td>41.5</td>
</tr>
<tr>
<td>[SA14]</td>
<td>RSA-2048</td>
<td>112</td>
<td>0</td>
<td>V7</td>
<td>3,558 slices/54/0</td>
<td>399</td>
<td>5.68</td>
<td>≈60</td>
</tr>
<tr>
<td>[BHH+15]</td>
<td>SPHINCS-256</td>
<td>256</td>
<td>128</td>
<td>Haswell CPU E3-1275 (1 core)</td>
<td>3500</td>
<td>14.7</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>