ARMed SPHINCS:
Computing a 41 KB signature in 16 KB of RAM

Andreas Hülsing[1], **Joost Rijneveld**[2], Peter Schwabe[2]

Technische Universiteit Eindhoven[1]
Radboud University, Nijmegen[2]
The Netherlands

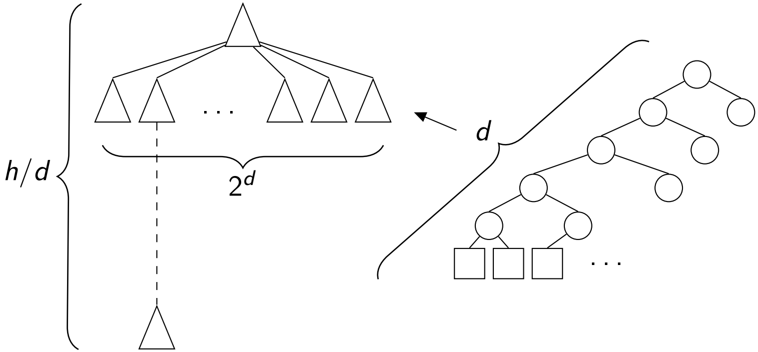
2016-03-07
PKC 2016

SPHINCS

- SPHINCS: Stateless, practical, hash-based, incredibly nice cryptographic signatures [BHHLNPSW15].
- Post-quantum
  - Hash functions do not fall to Shor (but halved by Grover)
- Hash-based schemes: conservative choice
  - One-way functions necessary for signatures [Rom90]
  - Tight security reductions
- Collision resilient

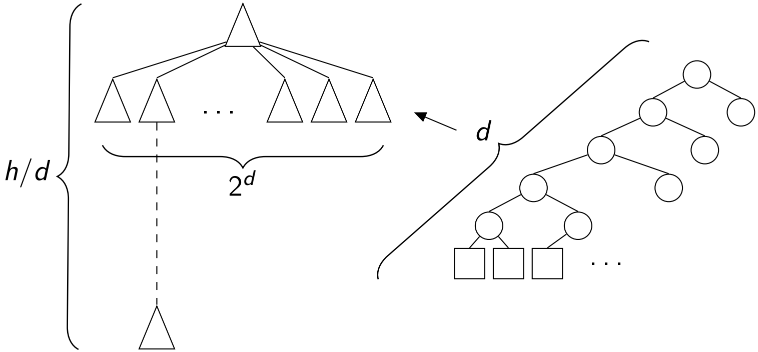
SPHINCS-256

- Large hash-tree, height $h = 60$
- Every $d = 12$-th layer: sign child node using an OTS
  - Effectively a hypertree of $h/d = 5$ Merkle trees [Mer90]
  - Trade signature size for time
- Sign messages using 2^{60} leaf nodes



SPHINCS-256

- Large hash-tree, height $h = 60$
- Every $d = 12$-th layer: sign child node using an OTS
  - Effectively a hypertree of $h/d = 5$ Merkle trees [Mer90]
  - Trade signature size for time
- Sign messages using 2^{60} leaf nodes
- No need to remember index: **stateless** [Gol87]



SPHINCS-256

- 41KB signatures, 1KB keys
- OTS
- Hash functions
- Key expansion function
- FTS

SPHINCS-256

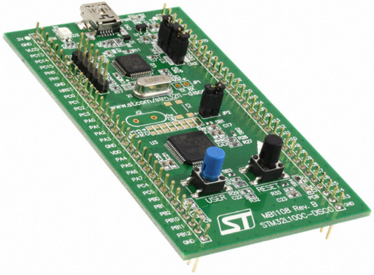
- 41KB signatures, 1KB keys
- OTS: *Winternitz OTS variant (WOTS+)* [Hül13]
- Hash functions: *BLAKE* [ANWW13], π_{ChaCha} [Ber08]
- Key expansion function: $ChaCha_{12}$
- FTS: *HORST* [BHHLNPSW15]

SPHINCS-256

- 41KB signatures, 1KB keys
- OTS: *Winternitz OTS variant (WOTS+)* [Hül13]
- Hash functions: *BLAKE* [ANWW13], π_{ChaCha} [Ber08]
- Key expansion function: $ChaCha_{12}$
- FTS: *HORST* [BHHLNPSW15]
  - Contains 16-layer Merkle tree (so $2^{16} = 65536$ leafs)
  - Goal: 32 authentication paths, root node
  - Paths start at (deterministically chosen) 'random' leafs
  - Complete tree takes approx. 2MB RAM..

Platform

- STM32L100C development board
- Cortex M3, ARMv7-M
- libopencm3 firmware
- 32MHz, 32-bit architecture
- 16 registers
- 256KB Flash
- **16KB RAM**

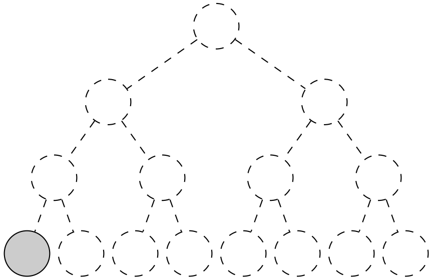


Treehash

- HORST tree is too large: 2MB!

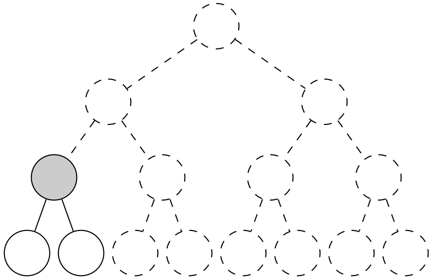
Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
  - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



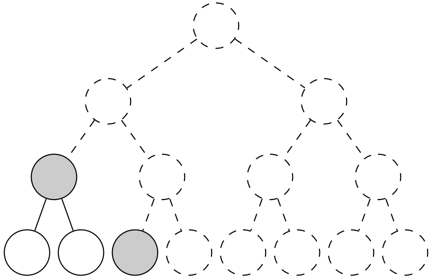
Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
  - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



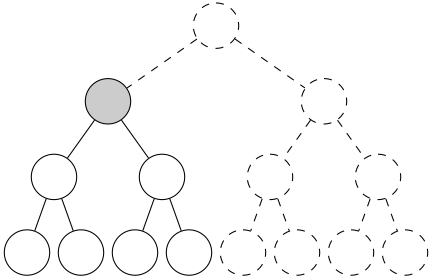
Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
  - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



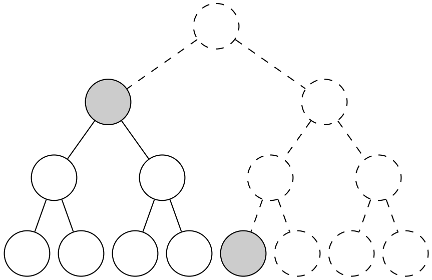
Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
  - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



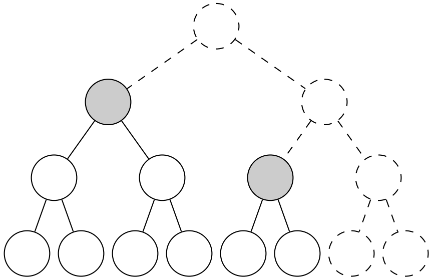
Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
  - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



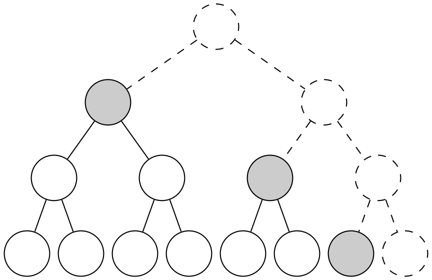
Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
  - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



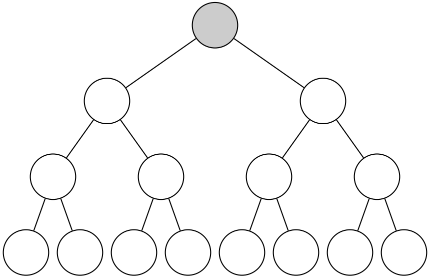
Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
  - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



Treehash

- HORST tree is too large: 2MB!
- Treehash [Mer90]: only remember relevant nodes
    - Maintain a stack: at most $log(n) = 16$ nodes (or $log(8) = 3$, in the example below)



Treehash considerations

- Goal: construct 32 HORST authentication paths
- Identify relevant nodes (320 of 131071)

Treehash considerations

- Goal: construct 32 HORST authentication paths

- Identify relevant nodes (320 of 131071)
- Identify relevant rounds ($<$ 320 of 65536)

Treehash considerations

- Goal: construct 32 HORST authentication paths

- Identify relevant nodes (320 of 131071)
- Identify relevant rounds ($<$ 320 of 65536)
- Identify relevant nodes in relevant rounds (bitmasks)

Treehash considerations

- Goal: construct 32 HORST authentication paths

- Identify relevant nodes (320 of 131071)
- Identify relevant rounds (< 320 of 65536)
- Identify relevant nodes in relevant rounds (bitmasks)
- Observation: sort masks by round index
  - Simply maintain a pointer
  - Iterate while performing treehash

Treehash considerations

- Goal: construct 32 HORST authentication paths

- Identify relevant nodes (320 of 131071)
- Identify relevant rounds (< 320 of 65536)
- Identify relevant nodes in relevant rounds (bitmasks)
- Observation: sort masks by round index
  - Simply maintain a pointer
  - Iterate while performing treehash

- Output in the appropriate order..

Streaming

- Cannot store signature $\rightarrow$ stream out immediately
- HORST inherently not ordered!

Streaming

- Cannot store signature $\rightarrow$ stream out immediately
- HORST inherently not ordered!
    - Re-arrange on the host
    - Tags (832 bytes total; $64 + 640 + 128$)
    - Alternatively: traverse tree on host

Streaming

- Cannot store signature $\rightarrow$ stream out immediately
- HORST inherently not ordered!
  - Re-arrange on the host
  - Tags (832 bytes total; $64 + 640 + 128$)
  - Alternatively: traverse tree on host

- Cannot store expanded key material
- Interleave ChaCha$_{12}$ and Treehash

Streaming

- Cannot store signature $\rightarrow$ stream out immediately
- HORST inherently not ordered!
  - Re-arrange on the host
  - Tags (832 bytes total; $64 + 640 + 128$)
  - Alternatively: traverse tree on host

- Cannot store expanded key material
- Interleave ChaCha$_{12}$ and Treehash

- Streaming message input
  - Blockwise BLAKE512
  - Stream twice: once for randomness, once for digest

ChaCha$_{12}$

- Core computation: ChaCha permutation
  - 685818 calls per signature
  - 65% of all computations
- 48 quarter-rounds of ADD, XOR and ROR
- Costs 542 cycles
  - *Note:* slightly improved since proceedings version

ChaCha_{12}

- Core computation: ChaCha permutation
  - 685818 calls per signature
  - 65% of all computations
- 48 quarter-rounds of ADD, XOR and ROR
- Costs 542 cycles
  - *Note:* slightly improved since proceedings version
- 512 bit state: 16 words of 32 bits each
  - Fit precisely in the 16 registers..

ChaCha$_{12}$

- Core computation: ChaCha permutation
  - 685818 calls per signature
  - 65% of all computations
- 48 quarter-rounds of ADD, XOR and ROR
- Costs 542 cycles
  - *Note:* slightly improved since proceedings version
- 512 bit state: 16 words of 32 bits each
  - Fit precisely in the 16 registers..
  - .. but we must preserve PC and SP
  - $\rightarrow$ Reorder to minimize and group stack access

ChaCha_{12}

- Core computation: ChaCha permutation
  - 685818 calls per signature
  - 65% of all computations
- 48 quarter-rounds of ADD, XOR and ROR
- Costs 542 cycles
  - *Note:* slightly improved since proceedings version
- 512 bit state: 16 words of 32 bits each
  - Fit precisely in the 16 registers..
  - .. but we must preserve PC and SP
  - $\rightarrow$ Reorder to minimize and group stack access
- Rotates on ARMv7 are (almost always) free!
  - `eor r6, r6, r11, ROR #29`

Performance

- Works on 16KB RAM ✓
    - Uses less than 7KB

Performance

- Works on 16KB RAM ✓
  - Uses less than 7KB
- Recall: 32MHz clock frequency
- Key generation: 28 205 671 cycles (0.88 seconds)
- Signing: 589 018 151 cycles (18.41 seconds)
- Verification: 16 414 251 cycles (0.51 seconds)
  - (of which approx. 10M spent communicating)
- *Note:* slightly improved since proceedings version

Performance

- Works on 16KB RAM ✓
  - Uses less than 7KB
- Recall: 32MHz clock frequency
- Key generation: 28 205 671 cycles (0.88 seconds)
- Signing: 589 018 151 cycles (18.41 seconds)
- Verification: 16 414 251 cycles (0.51 seconds)
  - (of which approx. 10M spent communicating)
- *Note:* slightly improved since proceedings version

- On 4-core Haswell:

  "[..] signs hundreds of messages per second."

Cost of the state

- Implemented XMSS^{MT} [HRB13], configured similarly
  - BLAKE and ChaCha primitives, 256 bit
  - Two layers, subtrees with 2^{10} leafs each
- XMSS^{MT}: Merkle trees linked with WOTS+

Cost of the state

- Implemented XMSS^{MT} [HRB13], configured similarly
  - BLAKE and ChaCha primitives, 256 bit
  - Two layers, subtrees with 2^{10} leafs each
- XMSS^{MT}: Merkle trees linked with WOTS+
- Stateful: process leafs incrementally
- BDS traversal [BDS08], store partial trees ($k = 6$)

Cost of the state

- Implemented XMSS^{MT} [HRB13], configured similarly
  - BLAKE and ChaCha primitives, 256 bit
  - Two layers, subtrees with 2^{10} leafs each
- XMSS^{MT}: Merkle trees linked with WOTS+

- Stateful: process leafs incrementally
- BDS traversal [BDS08], store partial trees ($k = 6$)

- Key generation: 8 857 708 189 cycles (276.80 seconds)
- Avg. signing: 19 441 021 cycles (0.61 seconds)
- Verification: 4 961 447 cycles (0.16 seconds)
- *Note:* slightly improved since proceedings version

Conclusions

- Stateless is expensive, but not prohibitively so
  - Signing 30x as expensive as XMSS^{MT}
  - Verification similar to XMSS^{MT}
  - (Key generation much cheaper)
- Feasible on limited platforms
  - Verification is practical
  - Non-interactive signatures (high latency)
- Further algorithmic improvements desirable
- Code is available (public domain):
  `https://joostrijneveld.nl/papers/armedsphincs/`

Reference I

Daniel J. Bernstein, Diana Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Peter Schwabe and Zooko Wilcox O'Hearn.
SPHINCS: Stateless, practical, hash-based, incredibly nice cryptographic signatures.
In Marc Fischlin and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 368-397. Springer, 2015.

John Rompel.
One-way functions are necessary and sufficient for secure signatures.
In *Proceedings of the twenty-second annual ACM symposium on theory of computing*, pages 387–394. ACM, 1990.

Ralph Merkle.
A certified digital signature.
In Gilles Brassard, editor, *Advances in Cryptology – Crypto '89*, volume 435 of *LNCS*, pages 218-238. Springer-Verlag, 1990.

Reference II

Andreas Hülsing.
W-OTS+ – shorter signatures for hash-based signature schemes.
In Amr Youssef, Abderrahmane Nitaj and Aboul-Ella Hassanien, editors, *Progress in Cryptology – AFRICACRYPT 2013*, volume 7918 of *LNCS*, pages 173-188. Springer, 2013.

Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn and Christian Winnerlein.
BLAKE2: Simpler, smaller, fast as MD5.
In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, volume 7954 of *LNCS*, pages 119-135. Springer, 2013.

Daniel J. Bernstein.
ChaCha, a variant of Salsa20.
SASC 2008: The State of the Art of Stream Ciphers, 2008.

Reference III

Oded Goldreich.
Two remarks concerning the Goldwasser-Micali-Rivest signature scheme.
In Andrew M. Odlyzko, editor, *Advances in Cryptology – Crypto '86*, volume 263 of *LNCS*, pages 104-110. Springer-Verlag, 1987.

Andreas Hülsing, Lea Rausch and Johannes Buchmann.
Optimal Parameters for $XMSS^{MT}$.
In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, volume 8128 of *LNCS*, pages 194-208. Springer, 2013.

Johannes Buchmann, Erik Dahmen and Michael Schneider.
Merkle tree traversal revisited.
In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, volume 5299 of *LNCS*, pages 63-78. Springer, 2008.