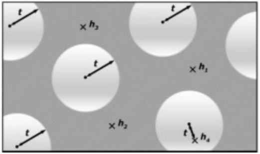
SPHINCS: practical stateless hash-based signatures

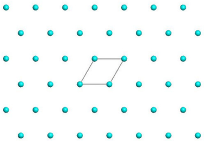
Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing,
Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou,
Peter Schwabe, Zooko Wilcox-O'Hearn

9 December 2014

Post-quantum signatures

Several issues for lattice-based, code-based, and multivariate quadratics-based signatures:

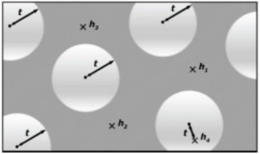


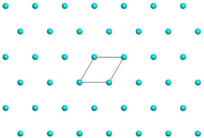


- Signature and/or key size are too big.
- Signature generation or verification is too slow.

Post-quantum signatures

Several issues for lattice-based, code-based, and multivariate quadratics-based signatures:

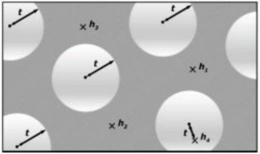


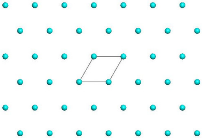


- Signature and/or key size are too big.
- Signature generation or verification is too slow.
- Where to find high-confidence secure parameters?

Post-quantum signatures

Several issues for lattice-based, code-based, and multivariate quadratics-based signatures:





- Signature and/or key size are too big.
- Signature generation or verification is too slow.
- Where to find high-confidence secure parameters?

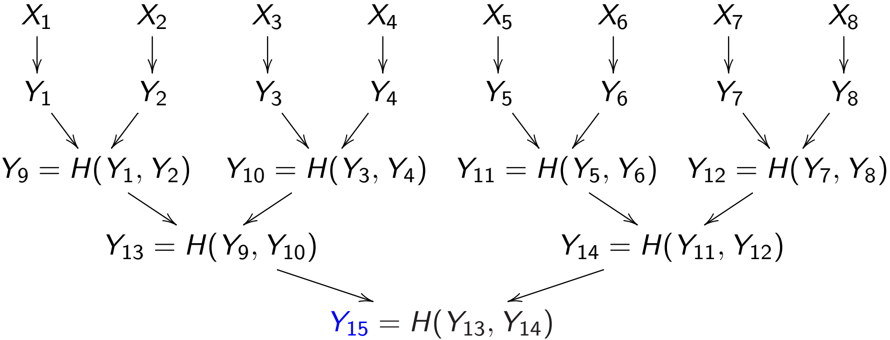
All signatures need hash functions anyways . . .

Hash-based signatures

- 1979 Lamport one-time signature scheme.
- Fix a k-bit one-way function $G : \{0,1\}^k \to \{0,1\}^k$ and hash function $H : \{0,1\}^* \to \{0,1\}^k$.
- Signer's secret key X: $2k$ strings $x_1[0], x_1[1], \ldots, x_k[0], x_k[1]$, each k bits. Total: $2k^2$ bits.
- Signer's public key Y: $2k$ strings $y_1[0], y_1[1], \ldots, y_k[0], y_k[1]$, each k bits, computed as $y_i[b] = G(x_i[b])$. Total: $2k^2$ bits.
- Signature $S(X, r, m)$ of a message m: $r, x_1[h_1], \ldots, x_k[h_k]$ where $H(r, m) = (h_1, \ldots, h_k)$.
- Must never use secret key more than once.
- Usually choose $G = H$ (restricted to k bits).
- 1979 Merkle extends to more signatures.

8-time Merkle hash tree

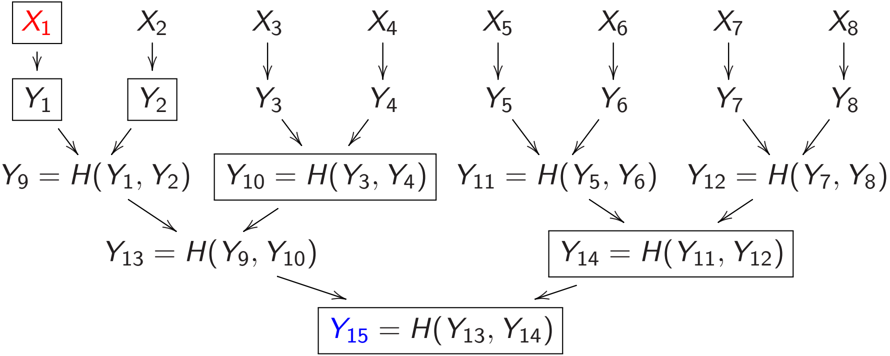
Eight Lamport one-time keys $Y_1, Y_2, \ldots, Y_8$ with corresponding $X_1, X_2, \ldots, X_8$, where $X_i = (x_{i,1}[0], x_{i,1}[1], \ldots, x_{i,k}[0], x_{i,k}[1])$ and $Y_i = (y_{i,1}[0], y_{i,1}[1], \ldots, y_{i,k}[0], y_{i,k}[1])$.



The Merkle public key is Y_{15}.

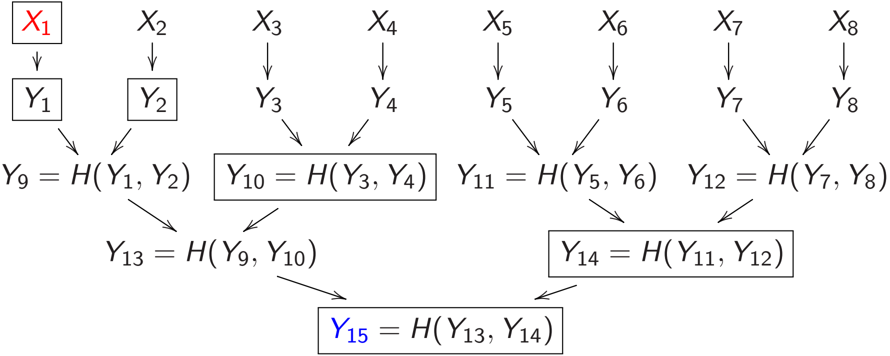
Signature in 8-time Merkle hash tree

First message has signature is $(S(X_1, r, m), Y_1, Y_2, Y_{10}, Y_{14})$.



Signature in 8-time Merkle hash tree

First message has signature is $(S(X_1, r, m), Y_1, Y_2, Y_{10}, Y_{14})$.

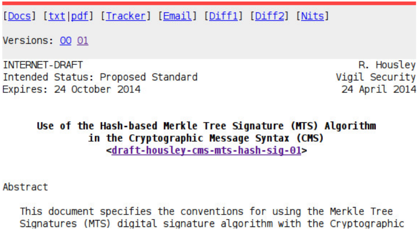


Verify by checking signature $S(X_1, r, m)$ on m against Y_1. Link Y_1 against public key Y_{15} by computing $Y'_9 = H(Y_1, Y_2)$, $Y'_{13} = H(Y'_9, Y_{10})$, and comparing $H(Y'_{13}, Y_{14})$ with Y_{15}.

Pros and cons

Pros:

- Post quantum
- Only need secure hash function
- Small public key
- Security well understood
- Fast
- Proposed for standards http://tools.ietf.org/html/draft-housley-cms-mts-hash-sig-01



[Docs] [txt|pdf] [Tracker] [Email] [Diff1] [Diff2] [Nits]

Versions: 00 01

INTERNET-DRAFT	R. Housley
Intended Status: Proposed Standard	Vigil Security
Expires: 24 October 2014	24 April 2014

Use of the Hash-based Merkle Tree Signature (MTS) Algorithm
in the Cryptographic Message Syntax (CMS)
<draft-housley-cms-mts-hash-sig-01>

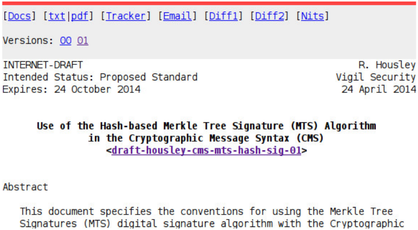
Abstract

This document specifies the conventions for using the Merkle Tree Signatures (MTS) digital signature algorithm with the Cryptographic

Pros and cons

Pros:

- Post quantum
- Only need secure hash function
- Small public key
- Security well understood
- Fast
- Proposed for standards http://tools.ietf.org/html/draft-housley-cms-mts-hash-sig-01



[Docs] [txt|pdf] [Tracker] [Email] [Diff1] [Diff2] [Nits]

Versions: 00 01

INTERNET-DRAFT R. Housley
Intended Status: Proposed Standard Vigil Security
Expires: 24 October 2014 24 April 2014

Use of the Hash-based Merkle Tree Signature (MTS) Algorithm in the Cryptographic Message Syntax (CMS)
<draft-housley-cms-mts-hash-sig-01>

Abstract

This document specifies the conventions for using the Merkle Tree Signatures (MTS) digital signature algorithm with the Cryptographic

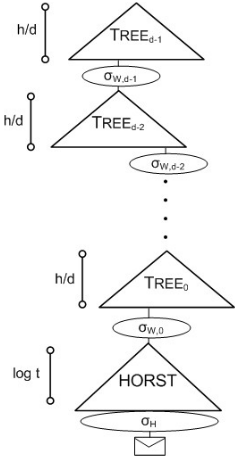
Cons:

- Biggish signature and secret key
- Stateful

  Adam Langley "for most environments it's a huge foot-cannon."

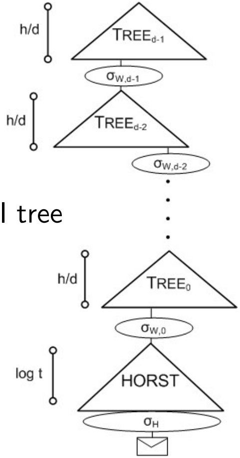
SPHINCS signature

- Stateless signature
- 128-bit post-quantum security
- Practical speed
- Practical signature size



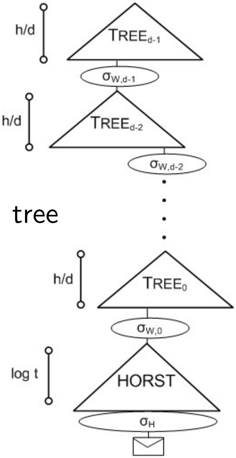
SPHINCS signature

- Stateless signature
- 128-bit post-quantum security
- Practical speed
- Practical signature size
- Use (pseudo-)random index in bottom level tree
- Generate secrets from master secret



SPHINCS signature

- Stateless signature
- 128-bit post-quantum security
- Practical speed
- Practical signature size
- Use (pseudo-)random index in bottom level tree
- Generate secrets from master secret
- Introduce new Few-Time Signature (FTS) HORS with Trees (HORST)
- New analysis of r-subset-resilience



SPHINCS achievements

Fast implementation, e.g., on Intel Haswell (titan0):

Key generation	3 182 996 cycles
Verification	1 438 120 cycles
Signature	51 035 880 cycles

SPHINCS achievements

Fast implementation, e.g., on Intel Haswell (titan0):

Key generation	3 182 996 cycles
Verification	1 438 120 cycles
Signature	51 035 880 cycles

Most importantly

SPHINCS achievements

Fast implementation, e.g., on Intel Haswell (titan0):

Key generation	3 182 996 cycles
Verification	1 438 120 cycles
Signature	51 035 880 cycles

Most importantly