SPHINCS: practical stateless hash-based signatures

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

2 April 2015

Traditional hash-based signatures: security vs. usability

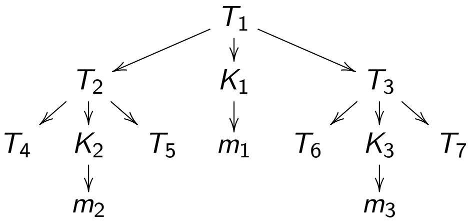
> *There's still a critical flaw in this! Recall that using a one-time signature twice **completely breaks the system**. Because of that, the signature scheme is "stateful". This means that, when signing, the signer absolutely must record that a one-time key has been used so that they never use it again. If the private key was copied onto another computer and used there, then the whole system is broken.*
>
> *That limitation might be ok in some situations, and actually means that one can build forward-secure signature schemes: schemes where signatures prior to a key compromise can still be trusted. Perhaps for a CA where the key is in an HSM that might be useful. However, for most environments it's a **huge foot-cannon**.*
>
> —Adam Langley, "Hash based signatures", 2013 (emphasis added)

Beyond hash-based: factoring-based foot-cannons!

Modern understanding of essential structure of
“provably secure” 1984 Goldwasser–Micali–Rivest:

One-time signature scheme based on factoring.

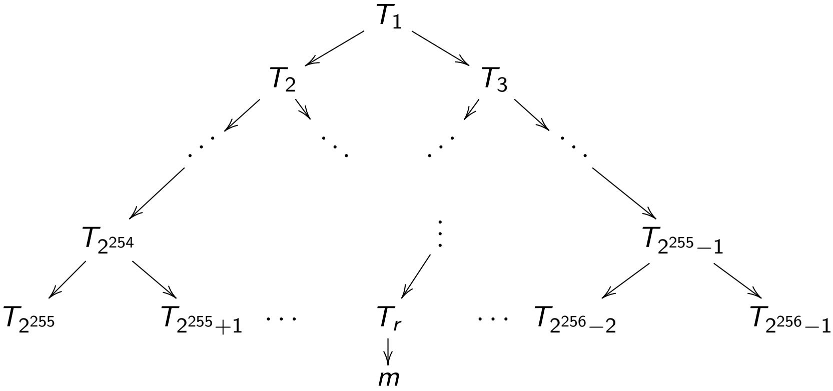
Stateful many-time signature scheme:
use one-time public key K_i to sign ith message;
use one-time public key T_1 to sign (T_2, T_3, K_1);
use one-time public key T_2 to sign (T_4, T_5, K_2);
use one-time public key T_3 to sign (T_6, T_7, K_3); etc.



ELIMINATE
THE STATE

Huge trees (1987 Goldreich), keys on demand (Levin)

Signer chooses random $r \in \{2^{255}, 2^{255}+1, \ldots, 2^{256}-1\}$,
uses one-time public key T_r to sign message;
uses one-time public key T_i to sign (T_{2i}, T_{2i+1}) for $i < 2^{255}$.
Generates ith secret key as $H_k(i)$ where k is master secret.



It works, but signatures are painfully long

0.6 MB for hash-based Goldreich signature using short-public-key Winternitz-16 one-time signatures.

Would dominate traffic in typical applications, and add user-visible latency on typical network connections.

It works, but signatures are painfully long

0.6 MB for hash-based Goldreich signature using
short-public-key Winternitz-16 one-time signatures.

Would dominate traffic in typical applications,
and add user-visible latency on typical network connections.

Example:
Debian operating system is designed for frequent upgrades.
At least one new signature for each upgrade.
Typical upgrade: one package or just a few packages.
1.2 MB average package size.
0.08 MB median package size.

It works, but signatures are painfully long

0.6 MB for hash-based Goldreich signature using
short-public-key Winternitz-16 one-time signatures.

Would dominate traffic in typical applications,
and add user-visible latency on typical network connections.

Example:
Debian operating system is designed for frequent upgrades.
At least one new signature for each upgrade.
Typical upgrade: one package or just a few packages.
1.2 MB average package size.
0.08 MB median package size.

Example:
HTTPS typically sends multiple signatures per page.
1.8 MB average web page in Alexa Top 1000000.

New: SPHINCS-256

Reasonable sizes.

0.041 MB signature.
0.001 MB public key.
0.001 MB private key.

Reasonable speeds.

Benchmarks of our public-domain software on Haswell:
51.1 million cycles to sign. (RSA-3072: 14.2 million.)
1.5 million cycles to verify. (RSA-3072: 0.1 million.)
3.2 million cycles for keygen. (RSA-3072: 950 million.)

Designed for 2^{128} post-quantum security,

even for a user signing more than 2^{50} messages:
2^{20} messages/second continuously for more than 30 years.
Yes, we did the analysis of quantum attacks.

Ingredients of SPHINCS (and SPHINCS-256)

Drastically reduce tree height (to 60).

Replace one-time leaves with few-time leaves.

Optimize few-time signature size *plus* key size.
New few-time HORST, improving upon HORS.

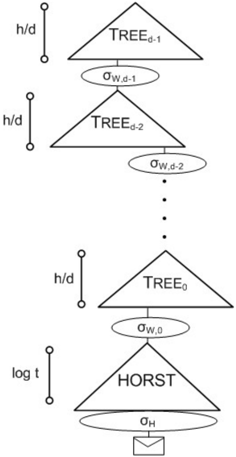
Use hyper-trees (12 layers), as in GMSS.

Use masks, as in XMSS and XMSS$^{\mathrm{MT}}$,
for standard-model security proofs.

Optimize short-input (256-bit) hashing speed.
Use sponge hash (with ChaCha12 permutation).

Use fast stream cipher (again ChaCha12).

Vectorize hash software and cipher software.



See paper for details: sphincs.cr.yp.to