Analysis and testing of SPHINCS algorithm

Sicurezza dei sistemi informatici - Giulia Milan

SPHINCS is a stateless post-quantum hash-based signature scheme

https://sphincs.org/

Outline

- Post-quantum cryptography
- Hash-based signatures
- SPHINCS
- Test in TLS use case scenario
- Possible improvements

Post-quantum cryptography

Standard asymmetric cryptography

It relies on **hard** problems:

- integer factorisation problem (RSA)
- discrete logarithm problem (DSA, ECDSA)

A hard problem can not be solved in **polynomial time** by a classical computer.

↓

If these problems turn out to be **solvable** in a polynomial time the security of standard cryptography algorithms is **broken**.

Post-quantum cryptography

Quantum computing and Shor’s algorithm

1980 - Quantum computing

- Introduction of the idea of **quantum computers**
- Machines that exploit quantum mechanical phenomena to solve mathematical problems that are considered difficult or intractable for standard computers

1994 - Shor’s algorithm

- Invention of the **Shor’s quantum algorithm**
- Algorithm which factors large numbers in polynomial time, assuming the availability of a quantum computer
- Until now the largest integer factored is 21

Post-quantum cryptography

Standard cryptography threats

- Many organisations such as Google and IBM are racing to create **practical** quantum computers
- With a large-scale quantum computer, it will be possible to **break** many of the public-key cryptosystems currently used in our applications
- This would compromise the **confidentiality** and **integrity** of all digital communications on the Internet
- We do not know **when** today's classic cryptography will be actually broken
- It is **difficult** and **time-consuming** to pull and replace existing cryptography from production software

Post-quantum cryptography

Need of PQC and its goals

Develop new cryptosystems that need to be resistant to an attacker equipped with a quantum computer.

They should:

- rely on **different hard mathematical problems** than standard cryptography
  - resistant to being solved by a large-scale quantum computer
- be **secure** against both a **quantum** and a **classical** computer
- **interoperate** with existing communications protocols and networks

Post-quantum cryptography

PQC projects

2016 - IETF Open Quantum Safe project

- developing and prototyping quantum-resistant cryptography
- implementing algorithms inside the **liboqs** library

2016 - NIST Post-quantum crypto project

- select proposed PQC algorithms
- second round of the standardization process
    - 9 signature algorithms and 17 key exchange schemes
    - **SPHINCS+** is one of the digital signature algorithms in the second round of standardization

Post-quantum cryptography

PQC algorithms

Many **classes** of mathematical problems and **strategies** that are conjectured to be quantum resistant exists.

There exists different categories of PQ cryptosystems:

1. Multivariate cryptography
2. Lattice-based cryptography
3. Code-based cryptography
4. **Hash-based cryptography**

Hash-based signatures

Main characteristics

- Uses **hash functions** for digitally signing documents
- Its security is based entirely on **hash function properties**
    - It does not rely on the hardness of mathematical problems
    - Therefore, if the underlying function turns out to be breakable in the future, it can be easily **replaced** with a new hash function
- The only security assumption is the **second preimage** or **collision resistance** of the hash function
    - This assumption is necessary for any digital signature scheme, but all other schemes require also additional security assumptions

Hash-based signatures

Main primitives

1. Cryptographic hash functions
2. One-time signatures (OTS)
3. Few-time signatures (FTS)
4. Merkle trees

Hash-based signatures

Main primitives

1. Cryptographic hash functions
2. One-time signatures (OTS)
3. Few-time signatures (FTS)
4. Merkle trees

Signature specific properties of **authentication**, **integrity** and **non-repudiation** are provided mainly by the cryptographic hash function chosen.

Hash-based signatures

Main primitives

1. Cryptographic hash functions
2. One-time signatures (OTS)
3. Few-time signatures (FTS)
4. Merkle trees

Security of hash-based signatures (HBS) is based on the correct implementations of the last three building blocks.

Hash-based signatures

High level structure

In OTS and FTS schemes:

- The public key is a **commitment** of the secret key
- The **signature** of a message consists of revealing some information of the secret key from which the verifier can recompute the commitment
- A signing key can only be used to sign respectively a **single** or **few** messages securely

These schemes are combined with **Merkle tree** structures to produce a many-time signature scheme:

- Basic binary tree
- Every node is a hash of its children
- The root is the public key
- The leaves are the hashes of the OTS or FTS public keys

Hash-based signatures

Stateful property

In general, HBS are based on OTS:

1. the signer needs to ensure that the OTS private key is never reused
    - keep track of a **state**
2. most of HBS strongly relies on iterating over signing keys in order
    - the state could be represented by some information on how many signatures were already made with the key
3. Practically, it can be difficult to deal with a state
    - the state management requirement is **not acceptable** for many applications

To overcome these limitations, **stateless** signature schemes have been proposed, such as **SPHINCS**.

SPHINCS

Stateless post-quantum hash-based signature scheme (2016)

Main components:

1. Stateless hypertree construction
2. Binary hash trees
3. WOTS+
4. HORST

SPHINCS

Stateless post-quantum hash-based signature scheme

Main components:

1. Stateless hypertree construction → SPHINCS relies upon a **hypertree** structure, which is a generalisation of the first stateless construction proposed by Goldreich.
2. Binary hash trees
3. WOTS+
4. HORST

SPHINCS

Stateless post-quantum hash-based signature scheme

Main components:

1. Stateless hypertree construction
2. Binary hash trees
3. WOTS+
4. HORST

Binary hash trees are used to create the hypertree structure. These trees are similar to the classical **Merkle trees**, with some changes.

SPHINCS

Stateless post-quantum hash-based signature scheme

Main components:

1. Stateless hypertree construction
2. Binary hash trees
3. WOTS+
4. HORST

The **intermediate nodes** of the hypertree are WOTS+ trees, that are one-time signature schemes.

Each WOTS+ tree signs the public key of the next tree. The last WOTS+ tree signs with a WOTS+ signature the HORST public key.

SPHINCS

Stateless post-quantum hash-based signature scheme

Main components:

1. Stateless hypertree construction
2. Binary hash trees
3. WOTS+
4. HORST

The **leaves** of the hypertree are HORST trees. HORST are HORS, few-time signature schemes, combined with trees.

The usage of a FTS instead of a OTS is preferred since their security decreases **gradually** with the multiple usage of the few-time key, while in OTS this would cause a complete break in security.

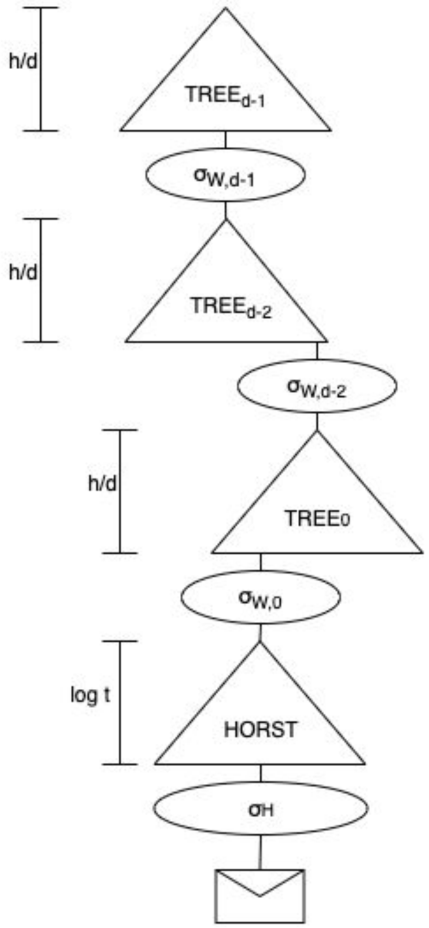
SPHINCS

Stateless post-quantum hash-based signature scheme

Main components:

1. Stateless hypertree construction
2. Binary hash trees
3. WOTS+
4. HORST

The structure of a SPHINCS **signature**:



SPHINCS

Stateless post-quantum hash-based signature scheme

- Designed so that the security of the algorithm can be based on **weak** standard-model **assumptions** of the underlying cryptographic hash function
- It exploits a randomised tree-based structure, in which indexes are selected **randomly** rather than sequentially

However, the stateless property has some **downsides** with respect to stateful schemes:

- Increase in **signing time**
- Increase in **signature size**

SPHINCS

Enhanced version SPHINCS+ (2019)

Main improvements:

- FORS instead of HORST
- Multi-target attack protection and tweakable hash functions
- Tree-less WOTS+ public key compression
- Verifiable index selection

NIST current proposals of SPHINCS+ variants:

- Usage of **SHAKE256**, **SHA-256** or **Haraka** hash functions
- Distinction of “**robust**” and “**simple**” variants, with simpler instantiations of the tweakable hash functions

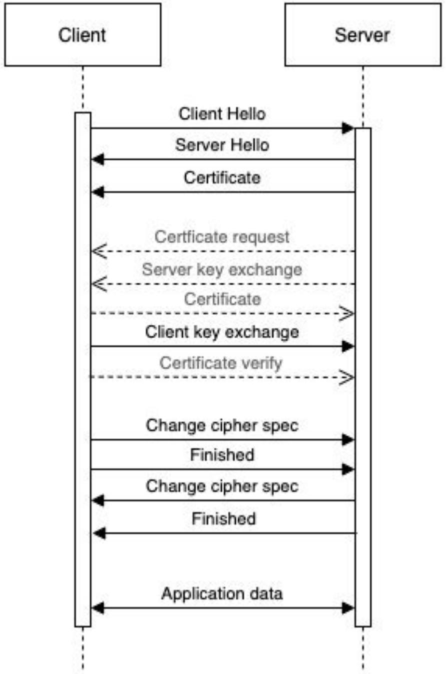
Test in TLS use case scenario

TLS handshake in a nutshell

1. **Cipher suite negotiation phase**
2. **Authentication phase***
3. Key agreement phase
4. **(Certificate verify from server)**
5. Final phase

*We consider a classic web scenario in which the client is not authenticated:

- authentication phase comprehends only the server authentication
- done with X.509 certificates



Test in TLS use case scenario

SPHINCS+ integration

1. **Cipher suite negotiation phase**
2. **Authentication phase**
3. Key agreement phase
4. **(Certificate verify from server)**
5. Final phase

"Client Hello" and "Server Hello" messages will negotiate the PQ signature algorithm.

TLS extension for specifying signature algorithms.

Test in TLS use case scenario

SPHINCS+ integration

1. Cipher suite negotiation phase
2. **Authentication phase**
3. Key agreement phase
4. (Certificate verify from server)
5. Final phase

Server will send its PQ certificate or its chain of certificates:

- New Algorithm Identifiers for X.509
- Add the **PQ public key** of the subject and the specific PQ signature algorithm used
- Certificate is **signed** by the issuer using his PQ private key

- Increase of the **size** of the single certificate, and so the size of the certificate chain
- TLS **fragmentation** of records to split packets before sending them

Test in TLS use case scenario

SPHINCS+ integration

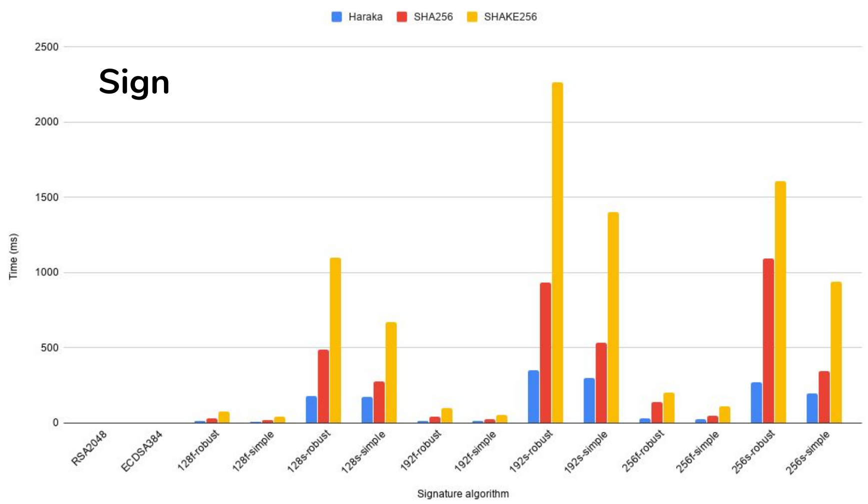
1. Cipher suite negotiation phase
2. Authentication phase
3. Key agreement phase
4. **Certificate verify from server**
5. Final phase

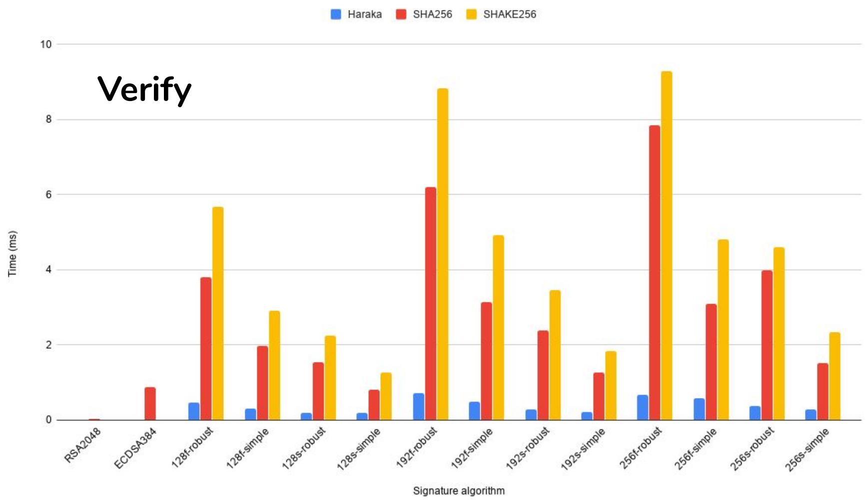
New step introduced in addition to the standard schema.

The server will sign the transcripts of the handshake and transmit a post-quantum “**Certificate verify**” message.

The message contains a PQ signature which the client will verify along with the signatures in the certificate chain.

Testbed I with liboqs test suite

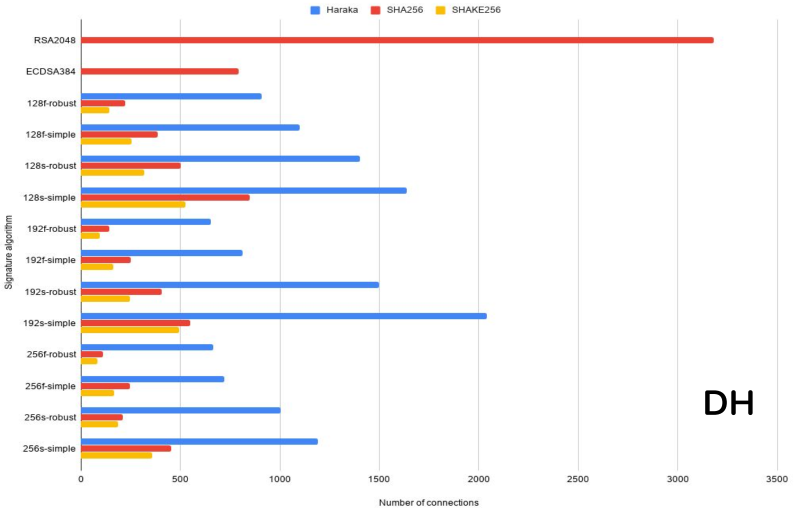


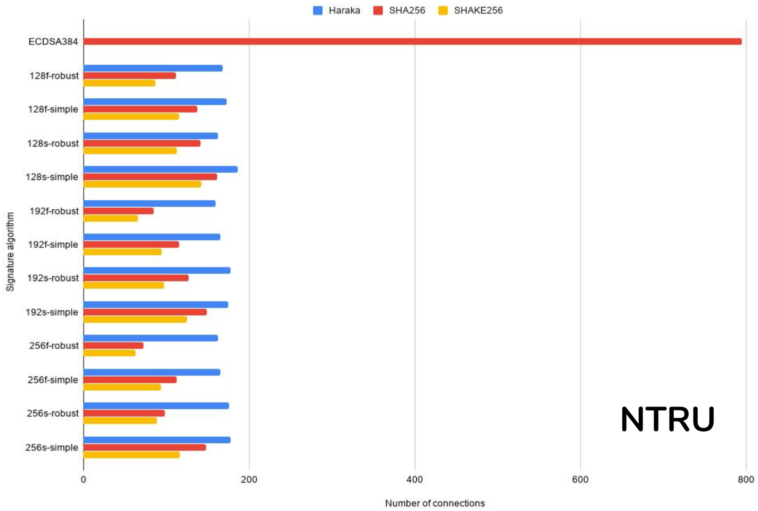


Average times for **Sign** and **Verify** operations for RSA, ECDSA and SPHINCS+:

- **Sign**: RSA (0.533 ms), ECDSA (1.01 ms), SPHINCS+ Haraka (129.68 ms)
- **Verify**: RSA (0.02 ms), ECDSA (0.87 ms), SPHINCS+ Haraka (0.39 ms)
- Performances heavily rely on the velocity of the underlying hash function
- SPHINCS+ variants with Haraka have times comparable to ECDSA values
- Other SPHINCS+ variants are significantly slower compared to RSA and ECDSA

Testbed II with OpenSSL 1.1.1





Number of TLS **connections per second** for RSA, ECDSA and SPHINCS+ with **DH** and **NTRU** key exchange:

- **DH**: RSA (3180), ECDSA (794), SPHINCS+ Haraka (1136)
- **NTRU**: ECDSA with DH (794), SPHINCS+ Haraka (171)
- Average number of connections with SPHINCS+ in 1 second is less than half of connections with RSA
- SPHINCS+ variants with Haraka perform better than ECDSA
- Other SPHINCS+ variants have lower values than ECDSA
- A standard TLS handshake is 4 times faster than a post-quantum handshake

Test in TLS use case scenario

Considerations

- Huge overhead introduced by SPHINCS+ variants by “sign” and “verify” operations:
  - Haraka hash function provides TLS connections **comparable** to ECDSA ones
  - Remaining hash functions make SPHINCS+ authentication much **slower** than RSA and ECDSA authentication mechanisms
  - Acceptable for **non-interactive applications**, when there is the need for **long term security** and for **stateless** schemes
  - For **interactive applications**, like TLS, further algorithmic improvements to these schemes could enable deployment on a broader scale
- Significant overhead introduced by adding a PQ key exchange in TLS handshake time
  - A standard DH handshake with ECDSA is still more than 4 times faster than the NTRU handshake with any SPHINCS+ variant

Possible improvements

For testing SPHINCS+

In general, results of tests reflect state-of-art results.

Note that for all tests in the TLS use case scenario:

- A **single certificate** was issued and sent from the server to the client with a self-signed certificate:
    - For more data, use longer certificate chains with intermediate Certification Authorities
        - Significant impact on the total handshake time
        - Significant impact on the number of connection established per second
- Only **SPHINCS+** was tested as a PQ signature algorithm:
    - For more data, compare SPHINCS+ also to other signature schemes
        - Computation of overhead with respect to other PQ signatures, if present

Possible improvements

For improving the deployment of SPHINCS+

For deployment on a broader scale:

- Faster hardware implementations and algorithmic improvements
- More studies on the **security** of SPHINCS+ scheme:
  - Most of studies refer to the previous version SPHINCS:
    - Fault injection attacks targeting SPHINCS but expanded to SPHINCS+
    - Differential power analysis attack only for SPHINCS
- Real world scenario integrations:
  - Supported but not directly integrated inside **OpenSSL** and **OpenSSH**
  - Integration in VPN applications like **OpenVPN**