SPHINCS+

Submitted to the NIST Post Quantum Project

Hash-based Signature Schemes

- These were developed as **one-time signature** schemes, but later extended to **many-times signatures** by **Merkle.**
- The **security** of these relies solely on the properties of the used **hash function**.
- Merkle’s tree-based signature scheme required fixing at key-generation time the **number of signatures** to be made.
- Most importantly, the system required users to remember a **state**: some information to remember how many signatures were already made with the key.
- Then, **SPHINCS** was designed as a **‘stateless’** hash-based signature scheme.

SPHINCS+

- Stateless hash-based signature scheme
- The basic idea behind the working of SPHINCS+ is to **authenticate** a huge number of few-time signature (FTS) key pairs using a so-called **hypertree**.
- **FTS** schemes are signature schemes that allow a key pair to produce a small number of signatures.
- For each new message, a **(pseudo)random** FTS key pair is chosen to sign the message. The signature consists of the **FTS signature** and the **authentication information** for that FTS key pair.
- The authentication information is roughly a **hypertree signature**, i.e. a signature using a certification tree of **Merkle tree signatures**.

SPHINCS+

Stateless Hash-based Signature Scheme

The basic idea behind the working of SPHINCS+ is to authenticate a huge number of few-time signature (FTS) key pairs using a so-called **hypertree.**

The signature consists of the FTS signature and the authentication information for that FTS key pair

Organisation

We first describe the components comprising the construction of SPHINCS+

- **WOTS+:** the OTS used in SPHINCS+.
- **XMSS:** the MTS used, and how it is used to do HT signatures.
- **FORS:** the FTS used.

Tweakable Hash Functions

- A tweakable hash function takes a **public seed** PK.seed and **context information** in form of an address ADRS in addition to the message input.

$$\mathbf{T}_\ell : \mathbb{B}^n \times \mathbb{B}^{32} \times \mathbb{B}^{\ell n} \rightarrow \mathbb{B}^n,$$
$$\mathrm{md} \leftarrow \mathbf{T}_\ell(\mathbf{PK}.\mathtt{seed}, \mathbf{ADRS}, M)$$

  And mapping and ***ln***-byte message ***M*** to an ***n***-byte **hash value md** using an *n*-byte seed PK.seed and a 32-byte address ADRS.

- The **tweak** (hash function address) might be interpreted as a **nonce**.
- This allows to make the hash function calls for each key pair and position in the virtual tree structure of SPHINCS+ **independent** from each other.

PRF and Message Digest

- SPHINCS+ makes use of a pseudorandom function PRF for **pseudorandom key generation**:

$$\mathbf{PRF} : \mathbb{B}^n \times \mathbb{B}^{32} \to \mathbb{B}^n.$$

- In addition, SPHINCS+ uses a pseudorandom function PRF_msg to generate **randomness** for the **message compression**:

$$\mathbf{PRF_{msg}} : \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^* \to \mathbb{B}^n.$$

- To **compress** the message to be signed, SPHINCS+ uses an additional **keyed hash function** H_{msg} that can process arbitrary length messages:

$$\mathbf{H_{msg}} : \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^* \to \mathbb{B}^m.$$

WOTS+ One-Time Signatures

- WOTS+ is a OTS scheme; while a private key can be used to sign **any message**, each private key MUST NOT be used to sign **more than a single** message.

Parameters

- **n:** the **security parameter**; it is the message length as well as the length of a private key, public key, or signature element in bytes. The value of n determines the in- and output length of the tweakable hash function used for WOTS+.
- **w:** the **Winternitz parameter**; it is an element of the set {4, 16, 256}.
- **len:** the **number** of ***n*-byte-string** elements in a WOTS+ private key, public key, and signature. It is computed as len = len_1 + len_2, with

$$\mathtt{len}_1 = \left\lceil \frac{8n}{\log(w)} \right\rceil, \ \mathtt{len}_2 = \left\lfloor \frac{\log\left(\mathtt{len}_1(w-1)\right)}{\log(w)} \right\rfloor + 1$$

The WOTS+ Key Pair

- In the context of SPHINCS+, the WOTS+ **private key** (a length len array of n-byte strings) is derived from a **secret seed** SK.seed that is part of the SPHINCS+ private key, and a WOTS+ **key generation address** using **PRF**.
- A WOTS+ **key pair** defines a virtual structure that consists of len hash chains of length *w*. Each of the len stings of n-bytes in the private key defines the **start node** for one hash chain. The public key is the **tweakable hash** of the **end nodes** of these hash chains.
- The corresponding public key is derived by applying **F iteratively** for **w repetitions** to each of the n-bit values in the private key, effectively constructing len hash chains. Here, **F** is parameterized by the address of the WOTS+ key pair, as well as the height of the F invocation and its specific chain, in addition to a seed PK.seed that is part of the SPHINCS+ public key.

WOTS+ Signature Generation

- A WOTS+ signature is a **length len array** of n-byte strings. The WOTS+ signature is generated by mapping a message *M* to len integers between *0* and $w - 1$. To this end, the message is transformed into len_1 base-w numbers.
- Next, we compute a checksum $C = \Sigma_{i=1}^{len_1}(w - 1 - m_i)$ over these values, represented as string of $\mathbf{len_2}$ base-*w* values $\boldsymbol{C = (C_1, \ldots, C_{len2})}$. This checksum is necessary to prevent message forgery: an increase in at least one $\boldsymbol{m_i}$ leads to a decrease in at least one C_i and vice-versa.
- Each of the base-*w* integers is used to select a node from a different hash chain. The **signature** is formed by concatenating the selected nodes.

WOTS+ Signature Verification

- In order to verify a WOTS+ **signature** on a message M, the verifier computes a WOTS+ **public key value** from the signature.
- This can be done by “completing” the **chain computations** starting from the signature values, using the base-w values of the message hash and its checksum. The result then has to be **verified**.
- When used in SPHINCS+, the output value is verified by using it to compute a **SPHINCS+ public key**.

The SPHINCS+ Hypertree

(Fixed Input-Length) XMSS [eXtended Merkle Signature Scheme]

- It is based on the **Merkle signature scheme**. It authenticates $2^{h'}$ WOTS+ public keys using a binary tree of height ***h'***. Hence, an **XMSS key pair** for height ***h'*** can be used to sign $2^{h'}$ different messages.
- Each **node** in the binary tree is an n-byte value which is the **tweakable hash** of the concatenation of its **two child nodes**.
- The **leaves** are the **WOTS+ public keys**.
- The **XMSS public key** is the **root node** of the tree.
- In SPHINCS+, the **XMSS secret key** is the single secret seed that is used to generate **all** WOTS+ secret keys.

- An **XMSS signature** in the context of SPHINCS+ consists of the **WOTS+ signature** on the **message** and the so-called **authentication path**.
- The latter is a **vector** of **tree nodes** that allow a verifier to compute a value for the **root** of the tree starting from a **WOTS+ signature**. A verifier **computes** the **root value** and verifies its correctness.

XMSS Parameters

h' : the height (number of levels - 1) of the tree.

n : the length in bytes of messages as well as of each node.

w : the Winternitz parameter as defined for WOTS+.

XMSS Public Key Generation

- Computation of the **root** of the binary hash tree.

XMSS Signature

- An XMSS signature is a ((len + *h'*) * n)-byte string consisting of
  - a WOTS+ signature sig taking len * n bytes,
  - the authentication path AUTH for the leaf associated with the used WOTS+ key pair taking *h'* * n bytes.

XMSS Signature Verification

- An XMSS signature is used to compute a candidate XMSS public key, i.e., the root of the tree.

The Hypertee

- The SPHINCS+ hypertree HT is a variant of XMSS-MT. A HT is a tree of several layers of XMSS trees.
- The trees on top and intermediate layers are used to sign the public keys, i.e., the root nodes, of the XMSS trees on the respective next layer below.
- Trees on the lowest layer are used to sign the actual messages, which are FORS public keys in SPHINCS+.

Parameters

- In addition to all XMSS parameters, a HT requires the **hypertree height h** and the **number of tree layers *d***. The same tree height $h' = h/d$ and the same Winternitz parameter w are used for all tree layers.

HT Signature

- A HT signature SIG_{HT} is a byte string of length $(h + d * len) * n$. It consists of ***d*** XMSS signatures (of $(h/d + len) * n$ bytes each).

FORS: Forest Of Random Subsets

- A few-time signature scheme (FTS).
- FORS uses **parameters** k and *t* = 2^*a* (example parameters are t = 215, k = 10). FORS signs strings of length ka bits.
- The **private key** consists of kt random n-byte strings grouped into k sets, each containing t n-byte strings. The private key values are pseudorandomly generated.
- To construct the FORS **public key**, we first construct k binary hash trees on top of the sets of private key elements. Each of the t values is used as a leaf node, resulting in k trees of height a.

FORS Signature and Verification

- Given a message of ka bits, we extract k strings of a bits. Each of these bit strings is interpreted as the index of a single leaf node in each of the k FORS trees. The signature consists of these nodes and their respective authentication paths.
- The verifier reconstructs each of the root nodes using the authentication paths and uses Th_k to reconstruct the public key.

SPHINCS+

Key Pair

- The **public key** consists of two n-bit values: the **root node** of the top tree in the hypertree, and a **random public seed** PK.seed.
- In addition, the **private key** consists of two more n-bit random seeds: SK.seed, to generate the WOTS+ and FORS secret keys, and SK.prf, used below for the randomized message digest.

SPHINCS+ Signature

- It should come as no surprise that the **signature** consists of a **FORS signature** on a **digest of the message**, a **WOTS+ signature** on the corresponding **FORS public key,** and a series of authentication paths and WOTS+ signatures to authenticate that WOTS+ public key.
- To **verify** this chain of paths and signatures, the verifier iteratively reconstructs the **public keys** and **root nodes** until the root node at the top of the SPHINCS+ hypertree is reached.

Theoretical Performance Estimates

Key Generation

- Generating the SPHINCS+ private key and PK.seed requires three calls to a secure random number generator. Next we have to generate the top tree. For the leaves we need to do 2^ h/d WOTS+ key generations (len calls to PRF for generating the sk and wlen calls to F for the pk) and we have to compress the WOTS+ public key (one call to Tlen). Computing the root of the top tree requires (2^h/d – 1) calls to H.

Signing

- For randomization and message compression we need one call to PRFmsg, and one to Hmsg.
- The FORS signature requires kt calls to PRF and F. Further, we have to compute the root of k binary trees of height log t which adds k(t – 1) calls to H.
- Finally, we need one call to Tk. Next, we compute one HT signature which consists of d trees similar to the key generation.
- Hence, we have to do d(2^h/d) times len calls to PRF and wlen calls to F as well as d(2^h/d) calls to Tlen. For computing the root of each tree we get additionally d(2h/d – 1) calls to H.

Verification

- First we need to compute the message hash using Hmsg. We need to do one FORS verification which requires k calls to F (to compute the leaf nodes from the signature elements), k log t calls to H (to compute the root nodes using the leaf nodes and the authentication paths), and one call to Tk for hashing the roots.
- Next, we have to verify d XMSS signatures which takes < wlen calls to F and one call to Tlen each for WOTS+ signature verification1 . It also needs dh/d calls to H for the d root computations.

- The classical security level, or bit security of SPHINCS+ against generic attacks can be computed as

$$b = -\log\left(\frac{1}{2^{8n}} + \sum_{\gamma}\left(1-\left(1-\frac{1}{t}\right)^{\gamma}\right)^{k}\binom{q}{\gamma}\left(1-\frac{1}{2^{h}}\right)^{q-\gamma}\frac{1}{2^{h\gamma}}\right).$$

- The quantum security level, or bit security of SPHINCS+ against generic attacks can be computed as

$$b = -\frac{1}{2}\log\left(\frac{1}{2^{8n}} + \sum_{\gamma}\left(1-\left(1-\frac{1}{t}\right)^{\gamma}\right)^{k}\binom{q}{\gamma}\left(1-\frac{1}{2^{h}}\right)^{q-\gamma}\frac{1}{2^{h\gamma}}\right).$$

Thank You