UDC 004 056 55 **DOI:10.30837/rt.2019.3.198.17**

D. TELEVNYI

THE KUPYNA HASH FUNCTION APPLICATION TO SPHINCS+ SIGNATURES

Introduction

Digital signatures (DSAs) are crucial elements in any system that requires data protection. The most used signatures are based on asymmetric pairs.

In recent years, there has been a substantial amount of research on quantum computers – machines that exploit quantum mechanical phenomena to solve mathematical problems that are difficult or intractable for conventional computers. If large-scale quantum computers are ever built, they will be able to break many of the public-key cryptosystems currently in use.

Hash-based signature schemes were developed as one-time signature schemes in the late 1970s by Lamport and extended to more signatures by Merkle.

As for the 2nd round there are 9 Digital signature candidates. SPHINCS+ (former SPHINCS) is in the list. The algorithm can be briefly described as a stateless hash-based signature scheme. It uses many components from XMSS but works with larger keys and signature to eliminate state.

The scheme can be used with different hash functions.

The main goal of this paper is to analyze the application of the national standard hash function the scheme of the NIST submission candidate SPHINCS+.

The NIST candidate SPHINCS+

In 40 years since Lamport's scheme, many ideas were introduced to improve performance, practicality and theoretical aspects of hash-based signatures. As a result, XMSS were introduced that's in the last stage of being standardized by the CFRG as the first post-quantum signature scheme. The only downside of XMSS is being stateful, which makes it not fit the standard definition of signature schemes.

In 2017 NIST (National institute of standards and technology) announced the 1st round candidate submissions for both post quantum Public-key Encryption and Key-establishment Algorithms and Digital Signatures.

As for the 2nd round there are 9 Digital signature candidates. SPHINCS+ (former SPHINCS) is in the list. The algorithm can be briefly described as a stateless hash-based signature scheme. It uses many components from XMSS but works with larger keys and signature to eliminate state.

SPHINCS [1] was designed by Bernstein, Hopwood, Hülsing, Lange, Niederhagen, Papachristodoulou, Schneider, Schwabe, and Wilcox-O'Hearn as a stateless hash-based signature scheme and was the first signature scheme to propose parameters to resist quantum cryptanalysis.

At a high level, SPHINCS works the following way. The basic idea is to authenticate a huge number of few-time signature (FTS) key pairs using a so-called hyper-tree. FTS schemes are signature schemes that allow a keypair to produce a small number of signatures, e.g., in the order of ten for our parameter sets. For each new message, a (pseudo)random FTS key pair is chosen to sign the message. The signature consists then of the FTS signature and the authentication information for that FTS key pair. The authentication information is a hyper-tree signature, i.e. a signature using a certification tree of Merkle tree signatures.

SPHINCS uses several parameters and functions [1]. The main security parameter is $n \in N$. The functions include two short-input cryptographic hash functions $F:\{0,1\}^n \rightarrow \{0,1\}^n$ *and* $H:\{0,1\}^{2n} \rightarrow \{0,1\}^n$; one arbitrary-input randomized hash function $H:\{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^m$, for $m = poly(n)$; a family of pseudo-random generators $G_\lambda:\{0,1\}^n \rightarrow \{0,1\}^{\lambda n}$ for different values of λ; an ensemble of pseudo-random function families $F_\lambda:\{0,1\}^\lambda \times \{0,1\}^n \rightarrow \{0,1\}^n$; and a pseudo-random function family $F:\{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^{2n}$ that supports arbitrary input lengths.

The security parameter n is also the output length of all cryptographic function families besides $H_{msg.}$

The Winternitz parameter w determines the number and length of the hash chains per WOTS+ instance. A greater value for w linearly increases the length of the hash chains.

The algorithm uses a hyper-tree with a total height of h and the hyper-tree consists of d layers of trees, each having height h/d. The height of the hyper-tree h determines the number of FORS instances.

The tree-based few-time signature scheme HORST has a space-time tradeoff which is controlled by two parameters $k \in N$ and $t = 2\tau$ with $\tau \in N$ and $k\tau = m$.

More detailed description of WOTS+ and HORST structures can be found in Specification paper [5]. The developers also propose SPHINCS-256 as a tested and verified set of parameters for the digital signature of the security strength *128* bit.

The hyper-tree as much as the whole signature can be visualized as it's shown in figure 1. It contains d trees (each consisting of a binary hash tree that authenticates the root nodes of $2^{h/d}$ L-Trees which in turn each have the public key no des of one WOTS+ key pair as leaves). Each tree authenticates the tree below using a WOTS+ signature σ_W, i. The only exception is $Tree_0$ which authenticates a HORST public key using a WOTS+ signature. Finally, the HORST key pair is used to sign the message.

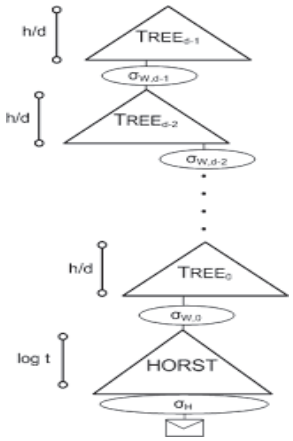

Fig. 1. SPHINCS hyper-tree

The general structure of SPHINCS can be described as a superposition of trees sets.

– *The SPHINCS hyper-tree* of height h. The root of tree is a part of the public key. The leaves are the *HORST* instances, the tree is divided in d layers containing so-called Merkle-trees.

– *Merkle-trees* of height h/d. The leaves are roots of the WOTS+ public key compression trees.

– *The WOTS public key compression trees* are L-trees of height $log_2 l$, for l – leaves amount. The leaves of this tree are components of a WOTS public key. The associated WOTS instance signs a tree root at the next layer.

– *The HORST public key compression trees* are Merkle trees of height $\tau = log_2 t$ where t is the number of public key elements in the HORST instances.

There were some changes made since the 2nd round submissions. SPHINCS+ introduced several details changes: multi-target attack protection [3], tree-less WOTS+ Public Key Compression (instead of tree-based compression), FORS (forest of random subsets) replaced HORST. [2].

The National Standard Hash Kupyna

In 2014 Ukraine released the new hash standard DSTU 7564:2014. It [dstu] defines the parameter set and modes of the Kupyna hash. As mentioned in the description the construct is based on the Even-Mansour scheme with Davies-Mayer compress function and inner permutation block from Kalyna (DSTU 7664:2014). The hash function supports several modes, defined as Kupyna-n. The standard [dstu] defines the following modes of Kupyna-256, 384, 512. [4]

The Kupyna hash is resistant to the second-preimage search attack. [4] The collision attacks described in «Analysis of the Kupyna-256 Hash Function» paper [5] showed that collision attacks are possible on the round-reduced hash up to 5 rounds. The time complexity of the attacks on round-reduced hash is shown in Table 1.

Table 1

Complexity of collision attack
on the reduced hash function

rounds	Time complexity
4	2^{67}
5	2^{120}

The collision attack on the compression function up to 7 rounds included semi-free-start collisions and was based on the rebound attack on Grøstl using SuperBox matching [5 – 7].

The time complexity results are shown in Table 2.

Table 2

Time complexity of the collision attack
on the compression function

rounds	Time complexity
6	2^{70}
7	2^{125}

In the SPHINCS design description [1] authors announced that the hash functions must be collision-resilient to be adopted in the signature scheme. In the updated version SPHINCS+ [2] hash functions must also be second-preimage search resistant due to possible multi-target attack [3].

The Kupyna hash is collision-resilient and preimage resistant if it works in modes as defined in the national standard [4]. Thus it can be applied to the SPHINCS+ signature scheme.

Kupyna-n application and benchmarking

There are tables of benchmark results given in [5]. The authors used the following Hash function families: SHA-256, SHAKE-256, Haraka-256.

The reference code can be found in the project site. [8] Each implementation case contains the signature scheme implementation alongside with hash function implementation. Authors released two versions: with suffix “s” – generates rather small signature and keys and “f” – for more quicker signature.

There are test cases applied to each reference implementation. The cases includes testing keypair and signature generation for WOTS+, FORS and the whole scheme. The PQCGenKAT_sign generates signatures for different message lengths.

The implementation is defined in a separate file and linked to signature within a source file. The header file “*hash.h*” contains the following interface listed below.

```
void prf_addr(unsigned char *out, ...);

void gen_message_random(unsigned char *R, const unsigned char *sk_seed..);

void hash_message(unsigned char *digest, uint64_t *tree, uint32_t *leaf_idx,
                  const unsigned char *R..);

void thash(unsigned char *out, const unsigned char *in, unsigned int
inblock..);
```

The src files contains the implementation of given functions with calls to the specific hash function. The Kupyna hash was implemented by the national standard description specs. [1]

The benchmark was held for signature schemes using 256 bit hash functions. The list included SHA-256, Haraka-256, SHAKE-256.

The Kupyna-256 function was used for implementation of the `hash_message()` function that computes the message hash using randomness, the public key, and the message. `gen_message_random` computes the message-dependent randomness R, using a secret seed as a key for HMAC, and an optional randomization value prefixed to the message. HMAC contains Kupyna-256 as a hash.

The signature testing runs on the following specs: *Core i3 6006U, 4GB RAM, Ubuntu 18.04, kernel 4.19.72 SMP x86_64.* The SHA-256 function was loaded from *openssl* shared lib from *libssl-dev* package. The compiler is *gcc 7*, CFLAGS = -Wall-Wextra-Wpedantic -O3

The test results for both “small signature length” and “fast signing” is listed in the Table 3 below.

Table 3

Run-time results

Signature name	Time (keygen + sign + ver)	Signature length
sphincs-shake256-256s	7.3851s	~ 29 kBytes
sphincs-shake256-256f	0.86s	~ 49 kBytes
sphincs-sha256-256s	6.83s	~ 29 kbytes
sphincs-sha256-256f	0.84s	~ 49 kBytes
sphincs-haraka256-256s	13.21s	~29 kBytes
sphincs-haraka256-256f	1.49s	~ 49kBytes
sphincs-sha-kup256-256s	6.82s	~29 kBytes
sphincs-sha-kup256-256f	0.81s	~ 49kBytes

The time measuring is performed by `clock_gettime()` function calls with CLOCK_PROCESS_CPUTIME_ID resolution. This function is common to Gnu libc and utilizes Intel processors TCS.

The measurements include key pair generation, message signing, verifying the signature with public key.

Conclusions

There has been made some changes in SPHINCS since the first submission to NIST as a Digital Signature candidate. The new signature called SPHINCS+ now requires hash function families to be second-preimage search resistant as well due to possible multi-targeting attacks. The Wirnetnitz signatures for public key compressions is now tree-less which became possible with introducing inner tweakable hash functions. The HORST scheme was also replaced by FORS (forest of random subsets) concept.

The Kupyna hash is collision-resilient and preimage resistant if it works in modes as defined in the national standard [4]. So it can be applied to the SPHINCS+ signature scheme.

The SHINCS+ signature scheme declared different hash families could be used for message signing. For practical part Kupyna-256 function was applied for randomness generation as a part of HMAC and message hashing. Authentication path calculation utilizes the SHA-256 hash.

The signature scheme was presented for 2 cases. The first case is generating rather small signatures (approximately 29 kbytes) for much longer time (a few seconds). This case is marked with -s suffix. The fast signing (marked with suffix -f) produces a longer signature (about 49 kBytes). The signature in this case contains all necessary data (seeds, full authentication path) to make verification more flawless.

The results shows the signatures have better performance when applying more lightweight hashes (SHA-2 /256, Kupyna -256) rather than using Keccak sponge (SHAKE). Haraka is a fast postquantum hash built to work on processors with AES-NI instructions. When used as a reference implementation (un-optimized) it shows itself less perfomant. In case when optimization is applied by vectorizing calculations and utilizing instruction sets the perfomance can be drastically increased.

The National standard [dstu] hash function shows good performance when used with other hashes for authentication in the signature scheme.

References:

1. Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederha-gen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical Stateless Hash-Based Signatures. In Elisabeth Oswaldand Marc Fischlin, editors, EUROCRYPT 2015, volume 9056 of LNCS, pages 368–397.Springer Berlin Heidelberg, 2015.

2. Jean-Philippe Aumasson, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe. SPHINCS$^+$ – Submission to the 2nd round of the NIST post-quantum project. Specification document (part of the submission package). 2019-03-14

3. Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks inhash-based signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, andBo-Yin Yang, editors,PKC 2016, volume 9614 of LNCS, pages 387–416. Springer BerlinHeidelberg, Berlin, Heidelberg, 2016.

4. Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Artem Boiko, Oleksandr Dyrda, Viktor Dolgov, Andrii Pushkaryov. A New Standard of Ukraine: The Kupyna Hash Function. Cryptology ePrint Archive. Report 2015/885, 2015. https://eprint.iacr.org/2015/885.pdf

5. Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Analysis of the Kupyna-256 Hash Function, Graz University of Technology, Austria, Cryptology ePrint Archive. Report 2015/956, 2015. https://eprint.iacr.org/2015/956.pdf

6. Mendel F., Rechberger C., Schl affer M., Thomsen S.S.: Rebound attacks on the reduced Grøstl hash function. In: Pieprzyk, J. (ed.) Topics in Cryptology – CT-RSA 2010. LNCS. vol. 5985. P. 350–365. Springer (2010)

7. Jean J., Naya-Plasencia M., Peyrin T. Improved rebound attack on the finalist Grøstl. In: Canteaut, A. (ed.) Fast Software Encryption – FSE 2012. LNCS. vol. 7549. P. 110–126. Springer (2012)

8. Peter Schwabe (September 23, 2019) SPHINCS+ Stateless hash-based signatures. Software Reference Implementation. Retrieved from https://sphincs.org/software.html

9. Klintsevich K., Okeya, Vuillaume C., Buchmann J., Dahmen E. Merkle signatures with virtually unlimited signature capacity. 5th International Conference on Applied Cryptography and Network Security. ACNS07, 2007.

Kharkiv National University of Radio Electronics *Received 11.09.2019*