K-XMSS and K-SPHINCS$^+$: Hash based Signatures with Korean Cryptography Algorithms

Minjoo Sim[1], Siwoo Eum[1], Gyeongju Song[1], Yujin Yang[1], Wonwoong Kim[1], and Hwajeong Seo[1*]
[1]Hansung University, 116, Samseongyo-ro 16-gil, Seongbuk-gu, Seoul, Republic of Korea
minjoos9797@gmail.com, shuraatum@gmail.com, thdrudwn98@gmail.com,
yujin.yang34@gmail.com, dnjsdndeee@gmail.com, hwajeong84@gmail.com

Abstract

Hash-Based Signature(HBS) uses a hash function to construct a digital signature scheme, where its security is guaranteed by the collision resistance of the hash function used. To provide sufficient security in the post-quantum environment, the length of hash should be satisfied. Modern HBS can be classified into stateful and stateless schemes. Two representative stateful and stateless HBS are XMSS and SPHINCS$^+$, respectively. In this paper, we propose two HBS schemes: K-XMSS and K-SPHINCS$^+$, which replace internal hash functions of XMSS and SPHINCS$^+$ with Korean cryptography algorithms. We also propose the reference implementation of K-XMSS and K-SPHINCS$^+$ employing LSH and two hash functions based on block ciphers (i.e. CHAM and LEA) as the internal hash function.

Keywords: XMSS, SPHINCS$^+$, Korean Cryptography Algorithms, Hash based Signatures, Software Implementations

1 Introduction

Hash-Based Signature(HBS) [1] schemes guarantee security with collision resistance of the hash function used. HBS schemes were developed in the 1970s by Lamport [2] and extended by Merkle [3]. As the threat to quantum computers increases, interest in the field is also on the rise. The hash function can respond to the threat of quantum computers by increasing the output length. For this reason, HBS schemes have attracted attention, and XMSS [4] has proven the feasibility of HBS. Recently, stateless SPHINCS [5], a variant of XMSS that does not need to maintain state, has been proposed. For the NIST(National Institute of Standards and Technology) Post-Quantum Cryptography standardization project [6], SPHINCS$^+$ [7], an improved version of SPHINCS, has been proposed.

In this paper, we propose variants of XMSS and SPHINCS$^+$ (i.e. K-XMSS and K-SPHINCS$^+$) by replacing the current hash function setting with Korean cryptography algorithms (i.e. LSH hash function and hash function based on Korean block ciphers).

1.1 Contributions

1.1.1 First implementation Korean version of XMSS and SPHINCS$^+$

To the best of our knowledge, this is the first trial to implement Korean version of hash-based cryptography schemes (i.e. K-XMSS and K-SPHINCS$^+$) with Korean hash functions. The original XMSS

The 6th International Symposium on Mobile Internet Security (MobiSec'22), December 15-17, 2022, 2021, Jeju Island, Republic of Korea
Article No. 1, pp. 1-7

*Corresponding author: Department of IT Convergence Engineering, Hansung University, 116, Samseongyo-ro 16-gil, Seongbuk-gu, Seoul, Tel: +82-02-760-4114

produces HBS through the use of SHA2 and SHAKE hash functions. The original SPHINCS$^+$ produces HBS using the SHA2, SHAKE, and HARAKA hash functions. In response, we proposed to generate HBS using Korean hash functions (i.e. LSH, CHAM, and LEA). As the result of evaluation performance, LSH showed the best performance among Korean hash functions in K-XMSS and K-SPHINCS$^+$.

1.1.2 Hash Function Based on Korean Block cipher

We implemented a hash function using Korean block ciphers. *Tandem DM* scheme was applied to use the Korean block cipher as a hash function. *Tandem DM* can generate a hash value having a length of $2m$-bit by applying a block cipher algorithm using an m-bit block length and a $2m$-bit key length. In this approach, we implemented hash functions using Korean block ciphers by applying LEA and CHAM Korean block ciphers.

2 Related Works

2.1 eXtended Merkle Signature Scheme(XMSS)

XMSS [4] is a stateful Hash-Based-Signature(HBS) scheme based on the Merkle Signature Scheme (MSS) [8], and uses WOTS$^+$(i.e. Winternitz One Time Signature Plus) [9] as the main building block. XMSS uses one key pair (i.e. private key and public key), n is the security parameter, l is the length in bytes, and H is the tree height. XMSS can generate up-to 2^H signatures, which is illustrated on Figure 1. To ensure the security of XMSS, the used key pair (i.e. WOTS$^+$ key) should not be used again.

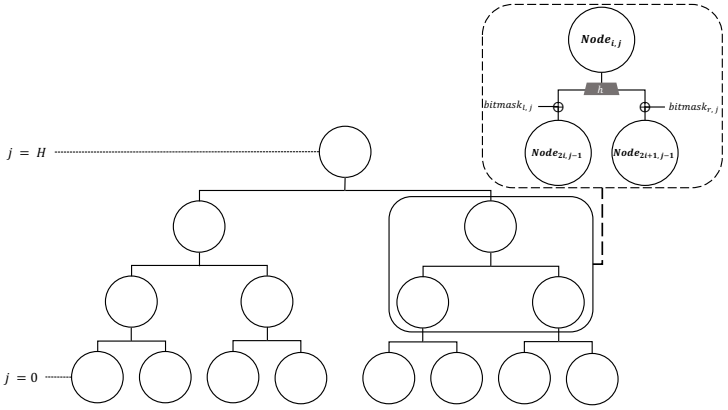

Figure 1: Tree structure of XMSS; Bitmask is chosen uniformly at random from $(b_{l,j}||b_{r,j} \in \{0,1\}^{2n})$.

As a result, the root node of the Merkle tree becomes the final XMSS public key. The bit length of this XMSS public key is $2(H + \lceil log_2 l \rceil + 1)n$, the signature length of XMSS is $(l + H)n$, and the private key of XMSS is less than $2n$.

2.2 SPHINCS$^+$

SPHINCS$^+$ [7] is a stateless Hash-Based-Signature(HBS) scheme that improves the speed and signature size of SPHINCS [5]. The main contribution of SPHINCS$^+$ is the introduction of FORS (i.e. few-time signature scheme). SPHINCS$^+$ uses functions with cryptographic properties and each parameter

is defined as follows: h and d is parameters of Hyper-Tree, b and k is parameters of FORS, and w is parameter of Winternitz. The structure of SHPINCS$^+$ is shown in Figure 2.

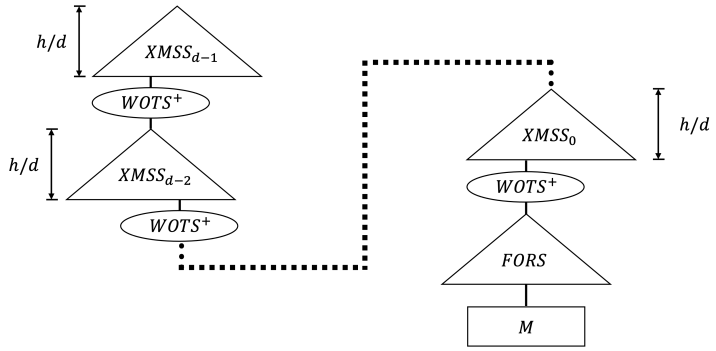

Figure 2: Overview of SPHINCS$^+$ structure.

SPHINCS$^+$ is a hyper-tree of height h and consists of d tree. In a hyper-tree, layer$(d-1)$ has a single tree and layer$(d-2)$ has $2^{h/d}$ trees. The root of the layer$(d-2)$ tree is signed using the WOTS$^+$ key pair in the layer$(d-1)$ tree.

2.3 Hash Function Based on Block cipher

Hash function based on block cipher uses a block cipher algorithm instead of a hash round function [10]. Several structures have been proposed to output the desired length of hash. We utilized the *Tandem DM* structure to implement hash functions using block ciphers. *Tandem DM* structure applies a block cipher algorithm using a key length of $2m$-bit when the block length is m-bit, and the output hash length is $2m$-bit. Figure 3 shows the *Tandem DM* Scheme.

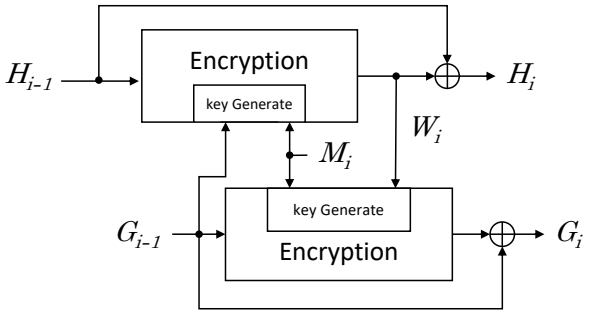

Figure 3: $2m$-bit hash round function based on m-bit block cipher with $2m$-bit key.

3 Proposed Method

3.1 Hash Function Based on Block cipher

In this paper, we construct hash function based on the Tandem DM scheme and utilize CHAM [11] and LEA [12] as the underlying block ciphers. Tandem DM scheme and block ciphers are described in Section 2.3. Algorithm 1 is a description for Figure 3.

The process of Algorithm 1 is as follows. The message received as input is divided into block size to proceed by the number of iterations. The iteration is repeated by the message length divided by the

block size. In this paper, the message length is assumed to be a multiple of the block size. Lines 4 and 7 perform key initialization. In line 4, G_i for the upper bit and $M[i]$ for the lower bit are used as a key. In line 7, $M[i]$ for the upper bit and W for the lower bit is used as a key. The initialized key generates a roundkey to be used for encryption through the *Roundkey generate* function. Then, H_i and G_i generate an encrypted value through an *Encryption* function, and finally XOR with W and G_i. If the CHAM and LEA algorithms are applied to the *Roundkey generate* function and *Encryption* function, hash values can be obtained through LEA and CHAM block ciphers.

Algorithm 1 Tandem DM scheme of hash function based on block cipher

Input: M (Message), ML (Message Length)
Output: Hash value

1: `n = Block size`
2: **for** $i = 0$ to ML/n **do**
3: $M[i]$: `Size of Block size`
4: $Key \leftarrow G_i, M[i]$ `(if` G_0`, use a initialization Vector)`
5: $RK \leftarrow RoundKey\ \ Generate(Key)$
6: $W \leftarrow Encrytion(H_i, RK)$ `(if` H_0`, use a initialization Vector)`
7: $Key \leftarrow M[i], W$
8: $RK \leftarrow RoundKey\ \ Generate(Key)$
9: $TEMP \leftarrow Encrytion(G_i, RK)$ `(if` G_0`, use a initialization Vector)`
10: $H_{i+1} \leftarrow H_i \oplus W_i$
11: $G_{i+1} \leftarrow G_i \oplus TEMP$
12: **end for**
13: **return** $Hash\ value \leftarrow H, G$

3.2 K-XMSS

In this paper, we replaced the hash functions (i.e. SHA2 and SHAKE) used in the original XMSS to Korean cryptography algorithms (i.e. K-XMSS). In particular, LSH [13] hash function and hash function based on block cipher (CHAM and LEA) are utilized, which are given in Section 3.1. We developed the code based on the basic C reference [14] provided by [15].

K-XMSS adopted the same parameters and structures utilized in XMSS. K-XMSS is performed on security parameters n of 256 and 512. Since the Winternitz parameter w of the original XMSS is fixed to 16, the value of w is also fixed to 16. For the $n = 32$ setting, K-XMSS uses LSH-256, CHAM ,and LEA. For the $n = 64$ setting, K-XMSS uses LSH-512.

3.3 K-SPHINCS$^+$

Similar to K-XMSS, we replaced the hash functions (i.e. SHA2, SHAKE, and HARAKA) used in the original SPHINCS$^+$ to Korean hash functions (LSH, CHAM, and LEA). Like K-XMSS, CHAM and LEA are block ciphers, so they are used as hash functions through the methods described in Section 3.1. We set the hash function parameters (i.e. n, h, d, k, w) used in SPHINCS$^+$ to be the same in K-SPHINCS$^+$. LSH is applicable to 256 and 512-bit outputs, and CHAM and LEA are applicable to 256-bit outputs. For this reason, we implement it based on hash function-256.

4 Evaluation

Implementations were evaluated on a MacBook Pro 16 with the Intel i7-9750H processor that can be clocked up to 2.6 GHz. Implementation is carried out on the Xcode framework.

4.1 K-XMSS vs XMSS

XMSS was evaluated using `test/speed.c` included in the basic C reference code provided by [15]. K-XMSS was also evaluated on the same setting by changing existing hash functions to Korean hash functions. SHA2 in XMSS used the OpenSSL library, and in the case of SHAKE, the optimally implemented code for XMSS operations. In contrast, for the hash function in our implementation, K-XMSS, the hash function LSH uses a basic C reference and uses the hash function our implementations based on CHAM and LEA. Table 1 and Table 2 show the performance evaluation results of K-XMSS and XMSS.

Among Korean Hash Functions, it was confirmed that LSH was significantly faster than other hash ciphers. When comparing SHA2 and SHAKE, it was confirmed that the performance of SHAKE was about 2 times faster than that of SHA2. Therefore, the performance of XMSS_SHA2 and K-XMSS_LSH was compared. As a result, it was confirmed that the LSH performance was about 2 times lower than that of the SHA2 during the entire operation process.

4.2 K-SPHINCS$^+$ vs SPHINCS$^+$

SPHINCS$^+$ was evaluated based on the simple code of the PQClean project [16], and K-SPHINCS$^+$ was evaluated by changing the hash function to a Korean hash function for the same code. Table 3 and Table 4 show the performance evaluation results of K-SPHINCS$^+$ and SPHINCS$^+$.

Among Korean hash functions, it was confirmed that LSH was significantly faster than other hash ciphers. When comparing the performance of SHA2, SHAKE, and HARAKA, it was confirmed that the performance of SHA2 was about 2 faster than that of SHAKE or HARAKA. Therefore, the performance of SPHINCS$^+$SHA2 and K-SPHINCS$^+$LSH was compared. As a result, it was confirmed that the LSH performance was about 2 times lower than that of SHA2 in the entire operation process.

5 Conclusion

We proposed K-XMSS, K-SHPINCS$^+$, which changed the hash functions of XMSS and SHPINCS$^+$ (i.e. SHA2, SHAKE, and HARAKA) to Korean hash functions (i.e. LSH, CHAM, and LEA). In particular, we used Korean block ciphers (i.e. CHAM and LEA) by changing them into hash functions. In this process, the internal hash functions used in K-XMSS and K-SPHINCS$^+$ used reference codes from LSH. And there is no prior research on hash functions based on block ciphers CHAM and LEA. Therefore, we used the CHAM and LEA hash function C code we implemented. As the result of the performance evaluation, it was confirmed that among Korean hash functions, LSH was significantly faster than other hash ciphers(i.e. CHAM and LEA). But, their performance was evaluated to be lower than that of SHA2, SHAKE, and HARAKA. This paper focused on the feasibility of Korean variant of HBS. For that reason, the performance metric is not optimal. In the current version, we utilized the basic C reference version. We believe that this performance can be further optimized by adopting the optimal implementation code (e.g. AVX2 or NEON).

Table 1: K-XMSS evaluation on MacBook Pro (Intel i7-9750H@2.6GHz); GK: Generating Keypair, CS: Creating Signature, VS: Verifying Signature, mid: median, avg: average, Algorithm indicates XMSS-[Hash function]_[h]_[n in bits].

Algorithm	GK		CS		VS	
	[sec]	[10^9cc]	mid [10^6cc]	avg [10^6cc]	mid [10^6cc]	avg [10^6cc]
LSH_10_256	8.28	21.45	31.64	44.78	10.91	11.22
LSH_10_512	17.17	44.52	65.86	93.00	21.69	22.37
CHAM_10_256	47.67	123.60	179.06	256.36	63.39	63.63
LEA_10_256	103.65	268.68	388.48	553.72	154.91	152.95

Table 2: Original XMSS evaluation on MacBook Pro (Intel i7-9750H@2.6GHz); GK: Generating Keypair, CS: Creating Signature, VS: Verifying Signature, mid: median, avg: average, Algorithm indicates XMSS-[Hash function]_[h]_[n in bits].

Algorithm	GK		CS		VS	
	[sec]	[10^9cc]	mid [10^6cc]	avg [10^6cc]	mid [10^6cc]	avg [10^6cc]
SHA2_10_256	3.53	9.17	13.54	19.13	4.62	4.63
SHA2_10_512	7.22	18.71	27.47	39.19	9.58	9.79
SHAKE_10_256	1.50	3.89	5.62	8.20	2.16	2.23
SHAKE_10_512	6.19	16.04	28.40	36.00	8.07	8.15

Table 3: K-SPHINCS$^+$ evaluation on MacBook Pro (Intel i7-9750H@2.6GHz); GK: Generating Keypair, CS: Creating Signature, VS: Verifying Signature, mid: median, avg: average, Algorithm indicates SPHINCS$^+$-[Hash function]_[n in bits].

Algorithm	GK		CS		VS	
	avg[sec]	mid[10^6cc]	avg [sec]	mid [10^9cc]	avg [sec]	mid [10^6cc]
LSH_256	0.04	108.54	0.88	2.29	0.02	60.88
CHAM_256	0.24	637.79	4.95	12.90	0.13	328.60
LEA_256	0.52	1,341.08	10.59	27.11	0.28	733.32

Table 4: Original SPHINCS$^+$ evaluation on MacBook Pro (Intel i7-9750H@2.6GHz); GK: Generating Keypair, CS: Creating Signature, VS: Verifying Signature, mid: median, avg: average, Algorithm indicates SPHINCS$^+$-[Hash function]-256f-simple.

Algorithm	GK		CS		VS	
	avg[sec]	mid[10^6cc]	avg [sec]	mid [10^9cc]	avg [sec]	mid [10^6cc]
SHA256	0.02	44.19	0.35	0.92	0.01	24.74
SHAKE256	0.04	94.63	0.07	1.79	0.02	49.19
HARAKA	0.03	89.10	0.76	2.01	0.02	52.70

6 ACKNOWLEDGEMENT

This work was partly supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (<Q|Crypton>, No.2019-0-00033, Study on Quantum Security Evaluation of Cryptography based on Computational Quantum Complexity, 25%) and this work was partly supported by Institute for Information & communications Technol-

ogy Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services, 50%) and this work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2020R1F1A1048478, 25%).

References

[1] Johannes Buchmann, Erik Dahmen, and Michael Szydlo. *Hash-based Digital Signature Schemes*, pages 35–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[2] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Citeseer, 1979.

[3] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[4] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[5] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 368–397. Springer, 2015.

[6] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[7] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2129–2146, 2019.

[8] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[9] Andreas Hülsing. W-OTS+–shorter signatures for hash-based signature schemes. In *International Conference on Cryptology in Africa*, pages 173–188. Springer, 2013.

[10] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Annual international cryptology conference*, pages 368–378. Springer, 1993.

[11] Dongyoung Roh, Bonwook Koo, Younghoon Jung, Il Woong Jeong, Dong-Geon Lee, Daesung Kwon, and Woo-Hwan Kim. Revised version of block cipher CHAM. In *International Conference on Information Security and Cryptology*, pages 1–19. Springer, 2019.

[12] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In *International Workshop on Information Security Applications*, pages 3–27. Springer, 2013.

[13] Dong-Chan Kim, Deukjo Hong, Jung-Keun Lee, Woo-Hwan Kim, and Daesung Kwon. LSH: a new fast secure hash function family. In *International Conference on Information Security and Cryptology*, pages 286–313. Springer, 2014.

[14] Xmss reference code. `https://github.com/XMSS/xmss-reference`. Accessed : 2022-09-06.

[15] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle signature scheme. In *RFC 8391*. IRTF, 2018.

[16] Nist pqc project. `https://csrc.nist.gov/Projects/post-quantum-cryptography`. Accessed : 2022-09-06.