K-XMSS and K-SPHINCS+ : Hash based Signature with Korean Cryptography Algorithms

Minjoo Sim, Siwoo Eum, Gyeongju Song,
Yujin Yang, Wonwoong Kim and **Hwajeong Seo**

https://youtu.be/QdiVGacvn2E

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

Contents

Our contribution

- First implementation Korean version of XMSS and SPHINCS+
  - Proposed to generate HBS using Korean hash function(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+**

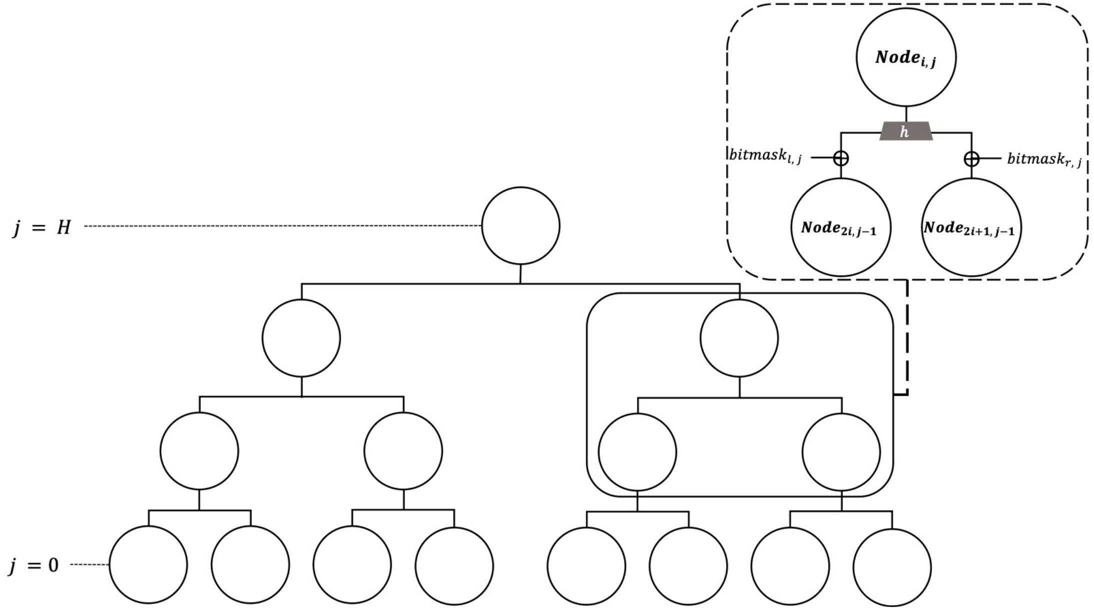
- Hash Function Based on Korean Block cipher
  - Applying the ***Tandem DM* scheme** to use the Korean block cipher as a hash function
  - Implemented hash functions using Korean block ciphers by applying **LEA and CHAM**

XMSS

- **Stateful Hash-Based-Signature(HBS) scheme** based on the Merkle Signature Scheme(MSS)
- Using WOTS+(Winternitz One Time Signature Plus) as the main building block
- Using one key pair consisting of a private key and a public key
- To ensure the security of XMSS, **the used key pair should not be used again**

XMSS

- As a result, the root node of the Merkle tree becomes the final XMSS public key



Security level	Length in bytes	Tree height
n	l	H
public key	**private key**	**signature**
$2(H + \lceil log2l \rceil + 1)n$	$< 2n$	$(l + H)n$

SPHINCS+

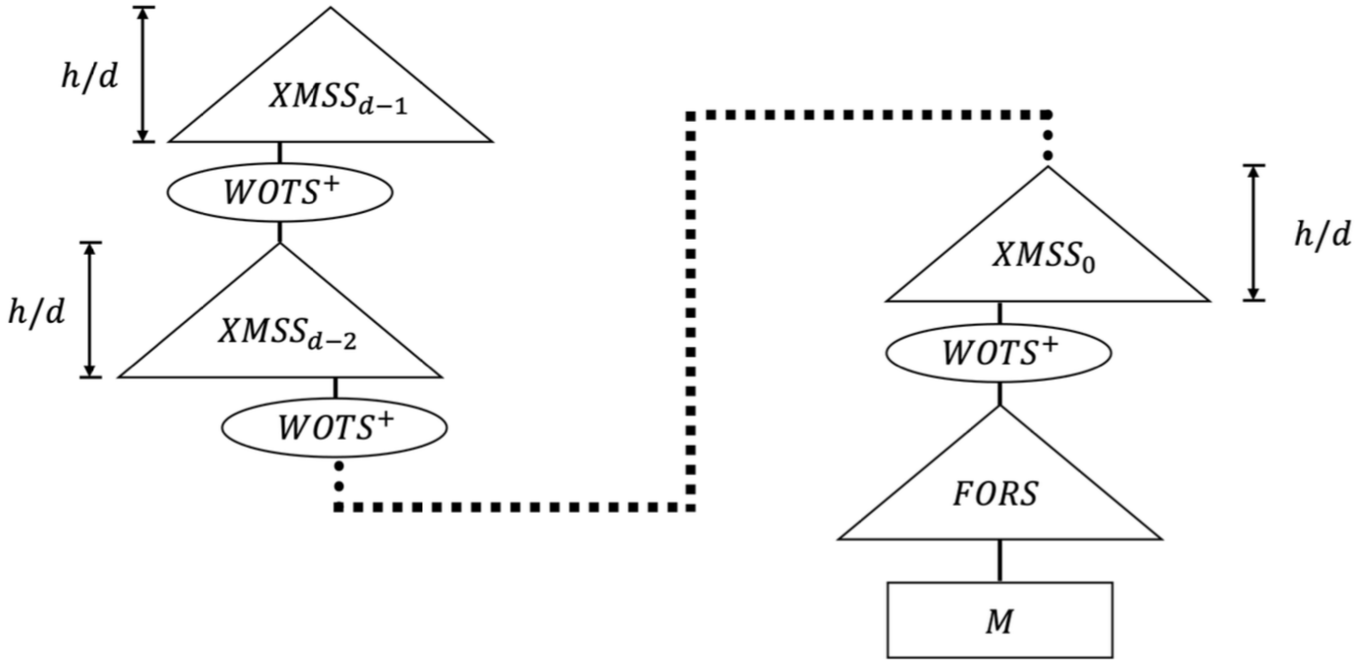
- **Stateless HBS scheme**
- Improve the speed and signature size of SPHINCS
- The main contribution of SPHINCS+
  - **Introduction of FORS**(FORS is few-time signature scheme)
  - **Selecting leaf nodes**

Hyper-Tree	FORS	Winternitz
h and d	b and k	w

Parameter of SPHINCS+

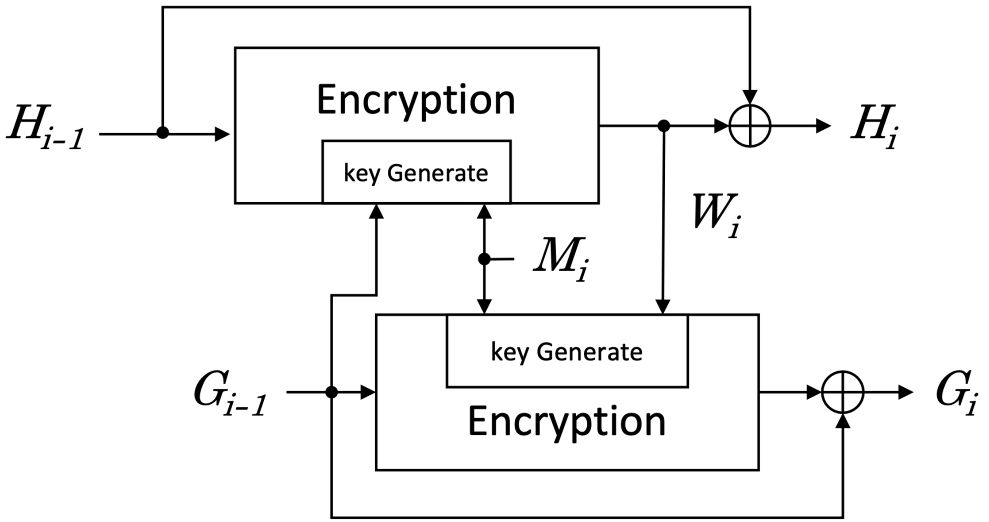
SPHINCS+

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



Hash Function Based on Block cipher

- Hash function based on block cipher uses a block cipher algorithm instead of a hash round function
- Several structures have been proposed to output the desired length of hash
- **Tandem DM** structure applies a block cipher algorithm
  - **Key length of 2m-bit when the block length is m-bit**
  - **Output hash length is 2m-bit**



Hash Function Based on Block cipher

- We construct hash function based on the **Tandem DM scheme** and utilize **CHAM** and **LEA** as the underlying block ciphers
- Algorithm 1 show Tandem DM scheme
  - In line 4, G_i for upper bit and $M[i]$ for lower bit are used as the key
  - In line 7, G_i for upper bit and W for lower bit are used as the key

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** $Hash\ value \leftarrow H, G$

key initialization

Generate a roundkey through the Roundkey generate function

Generate an encrypted value through an Encryption function

K-XMSS

- We replaced the hash functions (SHA2 and SHAKE) used in the original XMSS to Korean cryptography algorithms
- Utilized **LSH** hash function and hash function based on block cipher (**CHAM** and **LEA**)
- We developed the code based on the basic C reference of XMSS
- K-XMSS adopted the same parameters and structures utilized in XMSS

w	n	Hash Function
16	32	LSH-256, CHAM and LEA
	64	LSH-512

Parameter of K-XMSS

K-SPHINCS+

- We replaced the hash functions (SHA2, SHAKE, and HARAKA) used in the original SPHINCS+ to Korean hash functions (LSH, CHAM, and LEA)
- We set the hash function parameters (n, h, d, k and w) used in SPHINCS+ to be the same in K- SPHINCS+
- Implemented it based on hash function-256(LSH-256, CHAM and LEA)

Evaluation

- **K-XMSS vs XMSS**
- XMSS was evaluated using test/speed.c included in the basic C reference code
- SHA2 in XMSS used the OpenSSL library, and in the case of SHAKE, the optimally implemented code for XMSS operations
- In contrast, LSH uses a basic C reference and the hash function our implementations based on CHAM and LEA

Evaluation

- **K-XMSS vs XMSS**
- **LSH** was significantly **faster than other Korean hash ciphers**
- SHAKE was about 2 times faster than that of SHA2
- As the result, **LSH performance was about 2 times lower than that of the SHA2** during the entire operation process

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

Evaluation

- **K-SPHINSC+ vs SPHINSC+**
- SPHINCS+ was evaluated based on the simple code of the PQClean project
- K-SPHINCS+ was evaluated by changing the hash function to a Korean hash function for the same code

Evaluation

- **K-SPHINSC+ vs SPHINSC+**
- **LSH** was significantly **faster than other Korean hash ciphers**
- SHA2 was about 2 faster than that of SHAKE or HARAKA
- As the result, it was confirmed that the **LSH performance was about 2 times lower than that of SHA2** in the entire operation process

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

Conclusion

- We proposed K-XMSS, K-SHPINCS+, which changed the hash functions of XMSS and SHPINCS+ to Korean hash functions (LSH, CHAM, and LEA)
- As the result, LSH was significantly faster than other hash ciphers of K-XMSS and K-SPHINCS+
- But their performance was evaluated to be lower than that of SHA2, SHAKE, and HARAKA
- In Future work, this performance can be further optimized by adopting the o ptimal implementation code (e.g. AVX2 or NEON)

Thank you!