Journal of Modern Technology and Engineering
Vol.5, No.3, 2020, pp.234-241

A PERFORMANCE COMPARISON OF SOME HASH FUNCTIONS IN HASH-BASED SIGNATURE

Melike Karatay[1*], **Erdem Alkım**[2], **Necla Kırcalı Gürsoy**[1], **Mehmet Kurt**[1]

[1]Ege University, İzmir, Turkey
[2]Ondokuz Mayıs University, Samsun, Turkey

Abstract. Cryptographic signatures are widely used in internet communications and many other areas. There are a number of digital signature schemes based on mathematically difficult problems. Hash-based digital signatures are making use of hash function as their core component. Because of the speed of hash functions, hash-based signatures can be very fast comparing to other signature schemes. The security of hash-based signatures are relying on the actual security of hash functions and it can be used any hash function that provide desired security level. However, the performance in the signature scheme depends on the hash function that used to create signatures. Although number of hardware accelerators available for different hash functions in some platforms, in most case, only one hash function supported by those accelerators. Thus depending on the choice of hash functions, there will always be some users that need to use plain C implementation. In this article, the signature was created by using different hash functions in SPHINCS signature scheme and the effect of these hash functions on the efficiency of the signature scheme was measured implementing them on plain C programming language. Consequently, the empirical contributions of this paper are to present the computational results and the performance comparison using different hash functions in SPHINCS signature scheme at the first time in the literature.

Keywords: Performance Analysis, Hash Functions, Hash-based Signature, Cryptography.
AMS Subject Classification: 94A62, 94A60.
Corresponding author: Melike Karatay, Ege University, Bornova, İzmir, Turkey, Tel.: +905055623691, e-mail: *karataymlk9@gmail.com*
Received: 18 August 2020; Accepted: 22 November 2020; Published: 24 December 2020.

1 Introduction

Cryptographic signatures or digital signatures are mathematical structures that we use it to ensure the integrity of digital data. The most important application areas are authentication, integrity and non-repudiation. Signature schemes generally sign data using asymmetric encryption systems. The size of the data to be signed can be very large. Therefore, the hash functions are used in the digital signature before asymmetric encryption (Fiat & Shamir, 1986).

The hash-based signature was proposed by Leslie Lamport in 1979 and hash-based signature uses only hash functions. This proposed signature scheme is actually a one-time signature (OTS), which means each message is signed with a new key pair (Lamport, 1979). Nowadays, hash-based signature schemes are quite different from the signature scheme proposed by Lamport. Signature schemes created with OTS are not practical. But OTS can be transform into multi-time signature scheme by using a binary tree structure called the Merkle tree. In the Merkle tree, the root node is the OTS public key and its leaves are hash values generated from the root. If the Merkle tree is used then all messages can be signed with a single OTS public key (Merkle, 1989).

There are many hash-based signature schemes that use the Merkle tree. SPHINCS, XMSS,

and BPQS are some examples of these signature schemes. Hash-based signature schemes are generally stateful where the private key needs to be constantly updated. Unlike these hash-based signatures, SPHINCS is stateless. SPHINCS's signature sizes are larger than other hash-based signatures and use a few-time signature scheme called HORST (Bernstein et al., 2015). According to that, the hash values in the Merkle tree root are rarely allowed to collisions.

Although different hash functions can be used in hash-based signatures, they can often provide similar security levels. Thus hash-based signature schemes can have different efficiencies for same security levels. In this work, we have used different hash functions that have same security in a stateless hash-based signature scheme, SPHINCS, and analyzed effects of different cryptographic hash functions on efficiency of hash-based signatures.

Although most platforms, like embedded systems, smart cards, and desktop CPUs has hardware accelerators for specific hash functions, in most case they support only one hash function. As a result, no matter which hash function is chosen some users needs to create signatures without using that accelerators. Thus in this paper we only considered plain C implementations of hash functions and show their effect on the efficiency of the signature scheme to give platform independent comparison.

2 Hash Functions

Hash functions produce fixed and short length outputs regardless of the input length that is different for any input. There are three security definitions that cryptographic hash functions have to provide. These definitions are collision resistance, pre-image resistance and second pre-image resistance. Collision is the same hash output of two different inputs. This weakness is the most important security definition. Hash functions are one-way functions. If the input message can be estimated from the hash output, it is called pre-image. The third weakness is second pre-image and this weakness is like a collision. In the second pre-image, when an input message and its hash value are known, there is another message with the same hash value.

The standard hash functions Secure Hash Algorithm-1 (SHA-1) and Secure Hash Algorithm-2 (SHA-2) are functions of the Message Digest (MD) family. That is, they use the same structures and operations as the MD functions. In previous years, collision weakness was found in SHA-1 function. Therefore, the SHA-1 function has become insecure. The SHA-2 function is still secure. However, since SHA-1 and SHA-2 use the same structures, National Institute of Standards and Technology (NIST) has issued a new hash function standard called Secure Hash Algorithm-3 (SHA-3). SHA-3 differs from other hash functions and is a permutation-based function.

Permutation-based hash functions perform operations on the one permutation, such as SHA-3 function. performing operations on a single permutation such as encryption, hash, MAC (Message Authentication Code) is particularly advantageous in terms of performance. After the security weaknesses found in the SHA-1 and MD functions, the security of the other hash functions using the same structure must be reconsidered. Therefore, permutation-based hash functions are also assertive in terms of security. Hash functions using different structures in the cryptographic signature schemes because the hash functions are fast and require lower memory space.

2.1 SHA-256 Function

The SHA-2 was developed after the security weakness was found in the SHA-1. SHA-2 consists of six functions: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256. These six functions produce hash outputs of four different lengths such as 224-bits, 256-bits, 384-bits, and 512-bits. SHA-256 is a function of the SHA-2 family that generates a 256-bits hash value. The SHA-256 function performs operations on 512-bit message blocks and it has sixteen 32-bit word values. The SHA-256 hash function, like all other hash functions, has a padding rule. If

length of the message is not a multiple of 512-bit length, the padding rule is applied to the last block. The padding rule is to put 1 bit in the first of the empty bit positions and 0 bit in the remaining positions (Fips, 2004).

In SHA-256, logical bitwise AND, XOR, ROT and SHR are used. While ROT operation is a cyclic bit shift operation on the word, SHR operation is a non-cyclic bit shift operation on the word. The functions formed by these operations process the input message for 64 rounds.

2.2 Keccak Function

The SHA-3 (Secure Hash Algorithm-3), also known as Keccak, is permutation-based, unlike other standardized hash functions. SHA-3 contains six functions: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, and SHAKE256. Keccak consists of two parameters which are called bitrate (r) and capacity (c), and $r + c$ determines the permutation size. In addition, the capacity value is the security parameter. Keccak has an iterative structure consisting of rounds and the number of round is determined by the permutation length. The round function includes five steps: ι, χ, π, ρ and θ.

The multi padding rule is applied as in SHA-2 if the input message is not a multiple of r-bit length. Since Keccak has a Sponge structure, the hash value is created in two phases called absorbing phase and squeezing phase. In the absorbing phase, the input is grouped into r-bit blocks and these blocks are processed by Keccak permutation. Squeezing is the phase where the hash output is generated, and the output size is set at this phase (Bertoni et al., 2017).

In the SHA-3, the permutation length is 1600 bits. Other parameters are $r = 1088$-bit and $c = 512$-bit due to $b = r + c$. ι, χ, π, ρ and θ step operations are repeated for 24 rounds in the SHA-3. The bits to be processed with Keccak permutations are represented by a three-dimensional structure called the state. In the SHA-3, the state is indicated by a and its size is 5x5x64. The bit positions are indicated by x, y and z on the state. Also, indexes start from 0 in all Keccak step operations and the round constants value is expressed as RC.

$$\theta : a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1],$$

$$\rho : a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2],$$

$$\text{with satisfying } 0 \leq t \leq 24 \wedge \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \in GF(5)^{2x2},$$

$$t = -1 \text{ if } x = y = 0,$$

$$\pi : a[x][y] \leftarrow a[x'][y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} x' \\ y' \end{pmatrix},$$

$$\chi : a[x] \leftarrow a[x] + (a[x+1] + 1)a[x+2],$$

$$\iota : a \leftarrow a + RC[i_r].$$

$$RC[i_r][0][0][2^j - 1] = rc[j + 7i_r] \text{ for all } 0 \leq j \leq l,$$

$$rc[t] = (x^t \bmod x^8 + x^6 + x^5 + x^4 + x^1) \bmod x \in GF(2)[x].$$

The only difference between Shake algorithm and Keccak algorithm is that the Shake can produce arbitrary length output. Also, a padding difference is available in the Shake. In fact, the Shake algorithm is the extendable output function (XOF) of SHA-3. In the Shake algorithm, the squeezing phase is repeated until the desired hash output length is obtained. Extending the hash output does not mean increased security. The longer hash outputs may be needed.

2.3 Gimli Function

The Gimli function is a permutation-based authenticated encryption and hash function. Like other permutation-based encryption systems, it performs its operations in a single permutation. The 384-bit permutation is shown as a state and the state size is 3x4x32,where it has the word size indicated by 32 bits. The Gimli permutation has 24 rounds, and each round includes a non-linear layer. Also, the linear mixing layer is done in every second round, and in every fourth round has constant addition operation (Bernstein et al., 2017).

All symmetric crypto-systems necessarily involve a non-linear step. The Gimli permutations provide non-linearity with the 96-bit SP-box in the non-linear layer. The SP-box is parallelly applied to every column on the state. In the linear layer of the Gimli permutations, swap operations are performed on the rows in applied on the state. They are divided into two types as big swap and small swap as shown in Figure 1. Starting with the first round, small swap is applied first, then the swaps are applied once in two rounds, respectively. In the Gimli permutation, rounds are indexed from 1 to 24. When the round index is multiples of four, the first word of the state is XORed by the round constant.

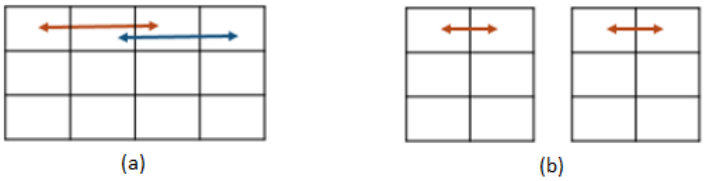

Figure 1: Gimli Big Swap (a) and Small Swap (b) Processes

In the Gimli hash function, the initial state bits are all zero. The received message as input is divided into 16-byte message blocks. Every message block is XORed with the first 16 bytes of the state and the Gimli permutation is applied to the state. Let the length of the final message block be k, where $0 \leq k \leq 15$. In this case, the following operations are taken: (1) The final message block is XORed with the first k-bytes of the state, (2) 1 is XORed with the (k)th byte of the state, (3) 1 is XORed with the last byte of the state and (4) the Gimli permutation is applied to the state. All message blocks are processed in this way. All of these processes are called absorbing phase. After the absorbing phase is completed, the squeezing phase is started where to generate the 32-byte hash value is generated. The first 16 bytes of the state are output. After that, the Gimli permutation is applied to the state again, and then the first 16 bytes of the state are output. Thus, the 32-byte hash output is generated (Bernstein et al., 2017).

2.4 Xoodyak Function

Xoodoo is a symmetric encryption system with 48-byte permutation. The 48-byte permutation is placed on the 3x4x32 matrix. The Xoodoo permutation includes a round function and has an iterative structure. The round function consists of five steps. These steps are such as the mixing layer, two types of the a plane shifting, the addition of round constants, and the non-linear layer. Xoodyak, also known as Cyclist, is the function of the Xoodoo permutations used to generate the hash value. The parameters used in the Xoodyak are hash length, the input block length and the output block length. The Xoodyak consists of two phases; absorbing and squeezing, and also it includes XOF (Daemen et al., 2018).

The Xoodyak function needs 12 rounds to provide the claimed security. It is a fast algorithm and can be used in many areas of cryptography, not just the hash functions. As in other permutation-based systems, Xoodyak performs all cryptographic operations on a single permutation. It is similar to Keccak structure. As in Keccak, it absorbs the blocks of the input message with Xoodoo permutations during the absorbing phase. In the squeezing phase, it outputs the

hash. Since Xoodyak also includes XOF function, the squeezing phase can be run as many times as desired.

3 Hash-Based Signature Schemes

Cryptographic signature schemes consist of asymmetric crypto-systems and hash functions. Asymmetric crypto-systems are slower than symmetric crypto-systems. Therefore, hash functions are used to increase performance of signature schemes. Hash-based signature schemes are examples of cryptographic signature schemes. The difference between hash-based signature schemes and other signature schemes is that the hash-based signature schemes use only hash functions. Besides, hash-based signature schemes provide post-quantum security (Song, 2014).

OTS schemes are schemes which a key pair is only used to sign one message. The OTS scheme can be generated from any one-way function. Signing and verification algorithms with OTS are very fast and low-cost. The disadvantage of the OTS scheme is that the signature length, the public and private keys are very large. For signing each message in the OTS scheme, a new key pair is required (Lamport, 1979). In addition to Lamport's OTS scheme, OTS schemes such as Winternitz OTS (WOTS) and Winternitz OTS Tree (WOTST) are also available (Bernstein et al., 2015).

The most important features are the size of signature, signing and verification speed for signature schemes. Generating a new key pair to sign each message slows down the system. If the Merkle tree is used, there is no need to generate a new key pair for each message. The Merkle tree is a binary tree that holds hash values in its leaf nodes. Non-leaf nodes hold the hash values of the lower level nodes. The Merkle root is the root node of this hash tree. The Merkle tree plays a major role in verifying large data. To prove that any hash value is a leaf of the Merkle tree, the logarithm of the number of leaves is processed (Merkle, 1989).

In cryptographic signature schemes, The Merkle tree allows the signing of multiple messages with the same public key. The Merkle tree starts signing with an OTS scheme. A many-time signature scheme is named a full signature. A full signature stores the index of the used OTS key pair in the tree. The OTS key pairs use the leaves of the tree from left to right because of making certain that each OTS key pair is only used once. In this case, small secret keys generated by the OTS, small public keys and small signatures are obtained.

A lot of hash-based signature schemes are stateful, namely signature requires updating the secret key, unlike digital signature schemes. Signing with stateful hash-based signature schemes requires keeping state of the used OTS keys and ensures that the OTS keys are never reused. Goldreich recommended a stateless hash-based signature scheme. Each OTS key pair corresponds to a non-leaf node, and it is used to sign the hash value of the public keys of its two child nodes in Goldreich's system (Bernstein et al., 2015; Oded, 2009).

A few-time signature (FTS) is a signature scheme intended to sign a few messages. Stateless signature schemes sign with a FTS. FTS schemes let security to decrease more and more in case a few-time key is used more than once. Examples of FTS schemes are Hash to Obtain Random Subset (HORS) and Hash to Obtain Random Subset Tree (HORST).

3.1 SPHINCS

SPHINCS, which is a stateless hash-based signature scheme, provides 2^{128} post-quantum security. Signatures generated by SPHINCS are 41 KB, each of private and public keys is 1 KB in size. SPHINCS significantly reduces signature size using advances of two new ideas together. The first idea is that SPHINCS replaces the leaf OTS with a hash-based FTS (here, HORS and HORST) scheme to improve the security level of randomized index selection. SPHINCS has a few index crashes that this case permits of a lesser tree height for the corresponding security standard. SPHINCS-256 decreases the whole tree height from 256 to 60 while providing 2^{128} se-

curity towards quantum attackers. The next idea is that SPHINCS inspects Goldreich's method as a hyper-tree establishment with h layers of trees of height 1, and generalizes to a hyper-tree with d layers of trees of height h/d (Figure 2). This brings a balance among signature size and time according to the number of layers d. The SPHINCS presents an exact security reduction to some standard-model properties of hash functions.

HORS is a fast hash-based FTS. The disadvantage of the HORS is to have large public keys. The HORST, namely HORS with trees, has been introduced as a better FTS for SPHINCS. The HORST sacrifices runtime to scale down the size of public key and the combined size of a signature and a public key when compared to the HORS (Bernstein et al., 2015).

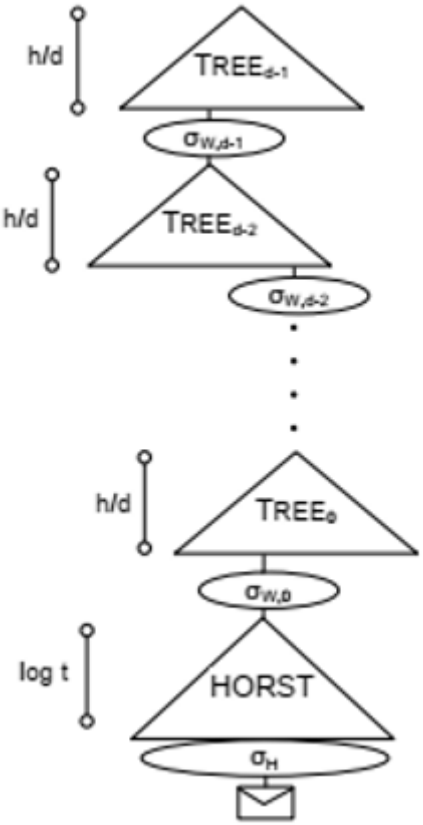

Figure 2: A SPHINCS signature structure (Bernstein et al., 2015)

In the case of a hash function with n-bit outputs, a hyper-tree of height h, and d layers, the signature size is $|\sigma| \approx d \cdot |\sigma_{OTS}| + h \cdot n$ where the size of a one-time signature $|\sigma_{OTS}|$ is nearly $O(n^2)$. Therefore, smaller full signatures are considered to decreasing the number of layers.

4 Results and Analysis

To compare effect of selected cryptographic hash functions, we implemented plain C version of Xoodyak, Gimli, SHAKE-256 and SHA-256 hash functions in the SPHINCS hash-based signing scheme. We used GCC version 7 to compile our implementations, and all measurements are taken on AMD A8-7410 APU while hyperthreading and turboboost disabled.

The speed measurement program was run 6 iterations for each version. At the end of all iterations, median and average values were computed. As seen in the Table 1, for the SPHINCS signature scheme, the effect of the selected cryptographic hash function can be as big as 6x, thus

selecting proper hash function is an important design decision for hash-based signature schemes. The SHA-256 hash function is the fastest during key generation, signing and verification. The hash function having the slowest key generation, signing and verification speeds is Xoodyak in the SPHINCS signature scheme.

Table 1: Speed of Sphincs Signature Scheme With Different Hash Functions

		Key Generation	Signing	Verification
Xoodyak	Median	6373168820 cyc	74464259994 cyc	111830350 cyc
	Average	6373172255 cyc	74463091634 cyc	112043026 cyc
Gimli	Median	2317458696 cyc	27014787078 cyc	41192062 cyc
	Average	2317449447 cyc	27014394727 cyc	40599169 cyc
SHAKE-256	Median	1179399910 cyc	13481173912 cyc	19999154 cyc
	Average	1179383854 cyc	13481078835 cyc	19975982 cyc
SHA-256	Median	1059877178 cyc	12509474092 cyc	18490526 cyc
	Average	1059865427 cyc	12509010547 cyc	18429730 cyc

5 Conclusion

Cryptographic hash functions are fast one-way functions along with the security they provide. They are used in digital signatures because of their security and fastness. In addition, hash-based signature schemes perform signing operations using only hash functions. For this reason, the hash functions used by the hash-based signature schemes affect the performance of the signature scheme. In this study, the effects of Xoodyak, Gimli, SHAKE-256 and SHA-256 functions on the SPHINCS hash-based signature scheme were investigated. While evaluating this performance, the effect of the hash function used in the key generation, signing and verification stages of the SPHINCS hash-based signature scheme was examined. As a result of these investigations, the fastest hash function in the SPHINCS hash-based signing scheme is SHA-256. The Gimli and SHAKE-256 functions have values close to the performance provided by the SHA-256 function, while the slowest hash function is Xoodyak. This is due to the structures that Xoodyak function uses differently from other hash functions.

6 Acknowledgment

This work is supported by Ege University Scientific Research Projects Coordination Unit with the project number 480.

References

Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P. & Wilcox-O'Hearn, Z. (2015). SPHINCS: practical stateless hash-based signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 368-397). Springer, Berlin, Heidelberg. doi: 10.1007/978-3-662-46800-5-15

Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.-X., Todo, Y. & Viguier, B. (2017). Gimli: a cross-platform permutation. Lecture Notes in Computer Science, Fischer, W. (ed.), CHES 2017: Cryptographic Hardware and Embedded Systems: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings (pp. 299-320). Springer, Cham. doi: 10.1007/978-3-319-66787-4_15

Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2009). Keccak specifications. Submission to nist (round 2), 320-337.

Daemen, J., Hoffert, S., Van Assche, G., and Van Keer, R. (2018). Xoodoo cookbook, IACR Cryptology ePrint Archive 2018, 767.

Fiat, A., & Shamir, A. (1986). How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques* (pp. 186-194). Springer, Berlin, Heidelberg.

Fips, P. (2004). Secure Hash Standard, National Institute of Standards and Technology, US Department of Commerce, 180-2.

Lamport, L. (1979). Constructing digital signatures from a one-way function (Vol. 238). Technical Report CSL-98, SRI International.

Merkle, R.C. (1989). A certified digital signature. In *Conference on the Theory and Application of Cryptology* (pp. 218-238). Springer, New York, NY.

Oded, G. (2009). Foundations of cryptography: Volume 2, basic applications.

Song, F. (2014). A note on quantum security for post-quantum cryptography. In *International Workshop on Post-Quantum Cryptography* (pp. 246-265). Springer, Cham.