SPHINCS-Simpira: Fast Stateless Hash-based Signatures with Post-quantum Security

Shay Gueron[1,2] and Nicky Mouha[3,4]

[1] University of Haifa, Israel.
[2] Intel Corporation, Israel Development Center, Haifa, Israel.
[3] National Institute of Standards and Technology, Gaithersburg, MD, USA.
[4] Project-team SECRET, Inria, France.

Abstract. We introduce SPHINCS-Simpira, which is a variant of the SPHINCS signature scheme with Simpira as a building block. SPHINCS was proposed by Bernstein et al. at EUROCRYPT 2015 as a hash-based signature scheme with post-quantum security. At ASIACRYPT 2016, Gueron and Mouha introduced the Simpira family of cryptographic permutations, which delivers high throughput on modern 64-bit processors by using only one building block: the AES round function. The Simpira family claims security against structural distinguishers with a complexity up to 2^{128} using classical computers. In this document, we explain why the same claim can be made against quantum computers as well. Although Simpira follows a very conservative design strategy, our benchmarks show that SPHINCS-Simpira provides a 1.5× speed-up for key generation, a 1.4× speed-up for signing 59-byte messages, and a 2.0× speed-up for verifying 59-byte messages compared to the originally proposed SPHINCS-256.

Keywords: Simpira, SPHINCS, post-quantum security, hash-based signature, AES-NI.

1 Introduction

Although it is not known how long it will be before large-scale quantum computers can be built, the advent of such computers will make most currently-used standards for public-key cryptography insecure. The goal of post-quantum cryptography is to ensure that systems remain secure against quantum computers.

Hash-based signatures are one of the most promising candidates for post-quantum digital signatures. The advantage of hash-based signatures is that the choice of secure parameters is better understood than for other constructions, against attacks using classical as well as quantum computers. Modern hash-based signatures can be quite efficient in terms of key storage, signature sizes, and computation times. A recently proposed hash-based signature scheme is SPHINCS [3], introduced by Bernstein et al. at EUROCRYPT 2015.

At ASIACRYPT 2016, Gueron and Mouha introduced a family of cryptographic permutations named Simpira [13], which supports inputs of $128 \times b$ bits,

where b is a positive integer. These permutations can be used in various applications, such as encryption, authentication, and hashing.

In this paper, we will explain how Simpira can be used inside SPHINCS-256. In particular, we will instantiate the $F : \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ and $H : \{0,1\}^{512} \rightarrow \{0,1\}^{256}$ functions of SPHINCS by Simpira with respectively $b = 2$ and $b = 4$, combined with a Davies-Meyer feedforward.

The main goal of this paper will be to provide benchmarking results of the resulting construction, and to compare them with the original SPHINCS-256. As a side result, we will also argue that Simpira can provide 2^{128} security not just against classical computers but also against quantum computers, so that the security proof of SPHINCS can be carried over.

To improve the readability of this paper, we will use Simpira to refer to Simpira v2, which is the published version of the design that appeared at ASIACRYPT 2016. Also, SPHINCS will refer to SPHINCS-256, which a specific instantiation of SPHINCS that was introduced in the SPHINCS paper at EUROCRYPT 2016.

Outline. First, we discuss hash functions, digital signatures and hash-based signatures in Sect. 2. In Sect. 3, we describe SPHINCS and in particular the F and H functions inside SPHINCS that dominate its performance. The description of Simpira is briefly recalled in Sect. 4. We explain how Simpira satisfies the design requirements of SPHINCS in Sect. 5, and in particular its resistance against quantum adversaries. In Sect. 6, we benchmark SPHINCS-Simpira, and compare its performance to the original SPHINCS design. A comparison with Haraka, another AES-based proposal to speed up SPHINCS, is made in Sect. 7. We conclude the paper in Sect. 8.

2 Preliminaries

A hash function h is a function that transforms an input of an arbitrary length into an output of a fixed length. It can be that the length of the input is fixed by the application, in which case it is acceptable that h only supports inputs of this specific length.

The requirements of a hash function depend on the application. In a very informal way, we will state some common requirements for hash functions:

- *preimage resistance*, that is, given an output, it should be infeasible to find a corresponding input to the hash function,
- *second-preimage resistance*, that is, given an input, it should be infeasible to find another input that hashes to the same output,
- *undetectability*, that is, it should be infeasible to distinguish the outputs from uniformly random values,
- *collision resistance*, that is, it should be infeasible to find two distinct inputs that hash to the same output,

– *indifferentiability* from a random oracle, that is, the hash function should "behave" as a random oracle when it is instantiated with an "ideal" compression function (or an "ideal" permutation for permutation-based hash functions).

A (digital) signature scheme consists of three algorithms:

– a *key generation algorithm* that draws a private key uniformly at random from the space of possible private keys, and generates a corresponding public key,
– a *signature algorithm* that given a private key and a message, generates a signature for this message, and
– a *verification algorithm* that given a public key, a message, and a signature, outputs "valid" if the signature is a correct signature for the given message, or "invalid" otherwise.

The goal of a signature algorithm is to ensure that an adversary cannot generate a forgery, that is, a new message and a corresponding signature, generated without knowledge of the private key.

Traditional signatures schemes such as RSA [24], DSA [22] or ECDSA [22] derive their security from the difficulty of number-theoretic problems such as the factorization or the discrete logarithm problem. Large-scale quantum computers are expected to render these signature schemes completely insecure due to Shor's algorithm [25], which runs in polynomial time with respect to the size of the input.

When post-quantum security is required, hash-based signatures can present an attractive alternative to traditional signature schemes. In fact, the very first digital signature scheme was a hash-based signature scheme, and was proposed around 1975 by Lamport [9,18]. In Lamport's proposal, the private key consists of two secret values, and the public key consists of the corresponding hashes of these values. Then, to sign a one-bit message, reveal either the first or the second secret value, depending on whether the message bit is zero or one. By repeating this process, an arbitrary number of bits can be processed.

A downside of Lamport's signature scheme is that the size of the private and public keys grows quickly with the number of messages that need to be signed. Another, perhaps more devastating downside of Lamport's signature scheme, is that it is a *stateful* signature scheme. That is, the signer needs to indicate which public keys are used to sign a particular message, and therefore needs to keep track of the number of messages signed so far. The security of the scheme critically depends on this property. As pointed out by Langley [19], this is a "huge foot-cannon" in many environments, for example when a private key needs to be copied from one device to another.

Many improved hash-based signature schemes have been proposed to address the shortcomings of Lamport's signature scheme. For example, the private keys can be generated from a master key using a secure key derivation function

(KDF) [21], and Merkle trees [20] can be used to shorten the public keys. However most, if not all, practical hash-based signature schemes are stateful. To also overcome this problem, the *stateless* hash-based signature scheme SPHINCS was recently proposed.

3 Hash Functions in SPHINCS

For a full specification of SPHINCS, we refer to [3]. For the purposes of this paper, it is sufficient to know that the performance of SPHINCS is largely determined by two functions:

- F, a 256-bit-to-256-bit hash function, and
- H, a 512-bit-to-256-bit hash function.

The precise security requirements of F and H are stated in the SPHINCS paper [3]. For this paper, it is sufficient to recall the informal definitions of Sect.2, and to say that F is required to be preimage resistant, second-preimage resistant and undetectable, and that H is required to be second-preimage resistant. These security properties should hold for both classical and quantum adversaries. In the SPHINCS security proof, F and H are not required to be collision resistant, nor indifferentiable from a random oracle.

To instantiate F and H, we will use two (unkeyed) cryptographic permutations of the Simpira family: P_{256} and P_{512} on 256-bit and 512-bit inputs respectively. More specifically, let $F(m) = P_{256}(m) \oplus m$, and $H(m) = \lfloor P_{512}(m) \oplus m \rfloor$, where $\lfloor \cdot \rfloor$ truncates the input to the first 256 bits.

When P_{256} and P_{512} are random permutations, this construction of F and H was shown to be optimally preimage, second-preimage and collision-resistant by Black et al. [6], and can trivially be shown to be undetectable as well. However, it is not indifferentiable from a random oracle, as shown by Chang et al. [8]. To see this, observe that it is easy to find a fixed point for F (pick $m = P_{256}^{-1}(0)$), whereas it is hard to find a fixed point for a random oracle. For H, a similar differentiability attack applies. But as we noted earlier, the SPHINCS security proof does not require F and H to be secure in the indifferentiability framework.

Note that Simpira comes with simple security arguments and comfortable security margins against all structural distinguishers with up to 2^{128} queries. This allows us to model P_{256} and P_{512} as random permutations for adversaries making up to 2^{128} queries, from which the security results of F and H follow. It may be considered to reduce the number of rounds of Simpira at a future point in time, and perhaps only when Simpira is used in SPHINCS. However, we decided to implement Simpira with the same number of rounds as in its design document, so that we can reuse Simpira's claim against structural distinguishers in this paper.

For fast software implementations of SPHINCS, it is recommended to evaluate F and H on multiple independent inputs in parallel. This property is important for the fast vectorized implementation of the original SPHINCS, and will be used in this paper as well. In the following section, we will explain how to use Simpira to instantiate P_{256} and P_{512} respectively.

4 Description of Simpira

We now recall the specification of Simpira for the parameters that are relevant to this paper: $b = 2$ and $b = 4$, corresponding to 256-bit and 512-bit inputs respectively. A complete description, together with a design rationale, a security analysis and benchmarks, can be found in the Simpira design document [13].

An algorithmic specification of the Simpira design of Fig. 3 is given in Fig. 4. It uses one round of AES as a building block, which corresponds to the `AESENC` instruction on Intel processors (see Alg. 1). Its input is a 128-bit xmm register, which stores the AES 4×4 matrix of bytes as shown in Fig. 1. For additional details, we refer to [12].

s_0	s_4	s_8	s_{12}
s_1	s_5	s_9	s_{13}
s_2	s_6	s_{10}	s_{14}
s_3	s_7	s_{11}	s_{15}

Fig. 1. The internal state of AES can be represented by a 4×4 matrix of bytes, or as a 128-bit xmm register value $s = s_{15} \| \ldots \| s_0$, where s_0 is the least significant byte.

The G-function[5] is specified in Alg. 2. It is parameterized by a counter c and by the number of subblocks b. Here, `SETR_EPI32` converts four 32-bit values into a 128-bit value, using the same byte ordering as the `_mm_setr_epi32()` compiler intrinsic. Fig. 2 shows how the constants can be expressed using the 4×4 byte matrix of AES.

`0x00` $\oplus\, c_0 \oplus b_0$	`0x10` $\oplus\, c_0 \oplus b_0$	`0x20` $\oplus\, c_0 \oplus b_0$	`0x30` $\oplus\, c_0 \oplus b_0$
$c_1 \oplus b_1$	$c_1 \oplus b_1$	$c_1 \oplus b_1$	$c_1 \oplus b_1$
$c_2 \oplus b_2$	$c_2 \oplus b_2$	$c_2 \oplus b_2$	$c_2 \oplus b_2$
$c_3 \oplus b_3$	$c_3 \oplus b_3$	$c_3 \oplus b_3$	$c_3 \oplus b_3$

Fig. 2. The constants used inside the $G_{c,b}$ function of Alg. 2, expressed as a 4×4 matrix of bytes. Here, $c = c_4 \| \ldots \| c_0$ and $b = b_4 \| \ldots \| b_0$ are 32-bit integers, where the least significant byte is c_0 and b_0 respectively.

Both the input and output of Simpira consist of b subblocks of 128 bits. The arrays use zero-based numbering, and array subscripts should be taken modulo the number of elements of the array. The subblock shuffle is done implicitly: we

[5] The F-function of the Simpira paper [13] is called G here to avoid confusion with the F-function of the SPHINCS design document [3].

do not reorder the subblocks at the end of a Feistel round, but instead we apply the G-functions to other subblock inputs in the subsequent round.

As a result of this implementation choice, Simpira for $b = 2$ and $b = 4$ and their reduced-round variants are not always equivalent to a (generalized) Feistel with identical rounds. For example, for $b = 2$ the G-function is alternatingly applied from left to right and from right to left. When the number of rounds is odd, this is not equivalent to a Feistel with identical rounds: the two output subblocks will be swapped.

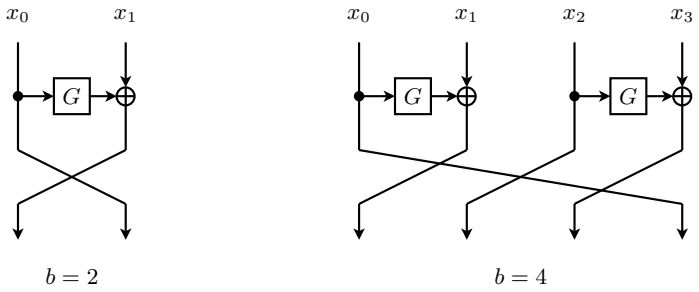

Fig. 3. One round of the Simpira construction for $b = 2$ and $b = 4$ In both cases, the total number of rounds is 15. G is shorthand for $G_{c,b}$, where c is a counter that is initialized by one, and incremented after every evaluation of $G_{c,b}$. Every $G_{c,b}$ consists of two AES round evaluations, where the round constants that are derived from (c, b).

5 Post-Quantum Security of Simpira

We refer to [13] for the security claims of Simpira in the classical (non-quantum) setting, where it is argued to be secure against structural distinguishers with a complexity up to 2^{128}. In this section, we argue why Simpira is also secure in a post-quantum setting against attacks with the same complexity.

In particular, we evaluate the security of F and H of SPHINCS when instantiated by Simpira with respectively $b = 2$ and $b = 4$, combined with a Davies-Meyer feedforward, as described in Sect. 3. We explain why this construction is undetectable, preimage resistant, second-preimage resistant and collision-resistant up to 2^{128} operations, both in a classical and in a quantum setting.

A common misunderstanding is that the complexity of a quantum algorithm is the square root of the corresponding classical algorithm. This would mean that F and H need to have a preimage and second-preimage resistance of 2^{256}. Such a claim is not made in the Simpira design document [13], as only security up to structural distinguishers with complexity up to 2^{128} is argued. Although no preimage or second-preimage attack is known, the Simpira instantiations of F and H may have preimage or second-preimage attacks with a complexity of

Algorithm 1 `AESENC` (see [12])

```
1: procedure AESENC(state, key)
2:     state ← SubBytes(state)
3:     state ← ShiftRows(state)
4:     state ← MixColumns(state)
5:     state ← state ⊕ key
6:     return state
7: end procedure
```

Algorithm 2 $G_{c,b}(x)$

```
1: procedure G_{c,b}(x)
2:     C ← SETR_EPI32(0x00 ⊕ c ⊕ b,
3:                    0x10 ⊕ c ⊕ b,
4:                    0x20 ⊕ c ⊕ b,
5:                    0x30 ⊕ c ⊕ b)
6:     return AESENC(AESENC(x, C), 0)
7: end procedure
```

Algorithm 3 Simpira ($b \in \{2, 4\}$)

```
1: procedure SIMPIRA(x_0, ..., x_{b-1})
2:     R ← 15
3:     c ← 1
4:     for r = 0, ..., R − 1 do
5:         x_{r+1} ← x_{r+1} ⊕ G_{c,b}(x_r)
6:         c ← c + 1
7:         if b = 4 then
8:             x_{r+3} ← x_{r+3} ⊕ G_{c,b}(x_{r+2})
9:             c ← c + 1
10:        end if
11:    end for
12:    return (x_0, x_1, ..., x_{b-1})
13: end procedure
```

Algorithm 4 Simpira^{-1} ($b \in \{2, 4\}$)

```
1: procedure SIMPIRA^{-1}(x_0, ..., x_{b-1})
2:     R ← 15
3:     c ← bR/2
4:     for r = R − 1, ..., 0 do
5:         if b = 4 then
6:             x_{r+3} ← x_{r+3} ⊕ G_{c,b}(x_{r+2})
7:             c ← c − 1
8:         end if
9:         x_{r+1} ← x_{r+1} ⊕ G_{c,b}(x_r)
10:        c ← c − 1
11:    end for
12:    return (x_0, x_1, ..., x_{b-1})
13: end procedure
```

Fig. 4. Alg. 2 specifies $G_{c,b}$ using the `AESENC` operation that is defined in Alg. 1. Alg. 3 and Alg. 4 specify Simpira and its inverse for $b = 2$ and $b = 4$, where both the input and output consist of b subblocks of 128 bits. Note that all arrays use zero-based numbering, and array subscripts should be taken modulo the number of elements of the array.

less than 2^{256}, possibly as the result of a distinguisher that makes more than 2^{128} evaluations.

This incorrect interpretation of post-quantum security is likely the result of some confusion about Grover's algorithm [10, 11], a quantum algorithm that inverts a function with a s-bit output using about $2^{s/2}$ operations. As $s = 256$ in our case, this corresponds to about 2^{128} quantum operations. Note that only the digest size s determines the complexity of Grover's algorithm, and not any particular structure about the function. When the function is a random oracle, Grover's algorithm was shown to be optimal by Bennett et al. [1].

But more efficient quantum algorithms exist to invert particular functions. The discrete logarithm problem comes in mind, for which Shor [25,26] provided a quantum algorithm that is polynomial-time in the size of the input. How-

ever, Shor's algorithm crucially relies on the special structure of the function. Here in particular, it uses the fact that exponentiation can be performed using the square-and-multiply algorithm. For Simpira, however, no such structure is present. Therefore, although the Simpira design document only claims security up to 2^{128} in the classical setting, also no quantum algorithm is known that can find preimages or second-preimages for F or H in less than 2^{128} quantum operations.

Furthermore, there is no algorithm that can find collisions in less than 2^{128} operations (classical or quantum). Note that a quantum algorithm by Brassard, Høyer and Tapp [7] is frequently claimed to find collisions with a lower complexity. We refer to Bernstein [2] for an explanation of why the Brassard-Høyer-Tapp quantum algorithm does not outperform classical collision search algorithms. Bernstein's argument is used to argue the security of SPHINCS against multi-target preimage attacks, and applies regardless of the primitives by which it is instantiated.

Undetectability can be seen as a rather broad notion, as there are many ways in which the outputs of F (or H) could be distinguished from uniformly random values. Perhaps the first output bit is biased towards zero, the XOR of the output bits is biased towards one, the probability of finding a collision is higher than for a uniformly random distribution,... Undetectability requires security against all of these distinguishers, and many more. In a classical setting, Simpira claims security against structural distinguishers up to 2^{128} queries, which includes all detectability attacks. In this claim, no specific computational model is imposed on the distinguisher, so that it carries over to the post-quantum setting as well. Note that F (or H) is evaluated on uniformly random inputs, so the quantum computer has no control over the inputs, but can base its decision only on the observation of the outputs of F (or H) in the "real world," or uniformly random values in the "ideal world."

We point out that we do not consider the *fully-quantum model*, in which an adversary can query an oracle implementing a cryptographic primitive in a quantum superposition of different state. Under that assumption, exponential speed-ups are possible to attack many commonly-used constructions, such as Even-Mansour [17], and even CBC-MAC, GCM, and OCB [15]. Currently, the security of most hash-based constructions (including the original SPHINCS) is not yet well-understood in the fully-quantum model. Neither do we consider that better alternatives to Grover's algorithm may be discovered in the future to invert cryptographic hash functions. Any progress in this area will likely have an impact on SPHINCS with the current parameter choices, regardless of the underlying hash functions.

6 Benchmarks of SPHINCS-Simpira

We measured the performance of SPHINCS on the latest Intel processor, Architecture Codename Skylake, with Hyper-Threading and Turbo Boost disabled. Our measurements consider both the original SPHINCS, as well as SPHINCS-

Simpira where the F and H functions are replaced by Simpira with respectively $b = 2$ and $b = 4$ and a Davies-Meyer feedforward, as explained in Sect. 6. As none of the other parameters are changed, both the original SPHINCS and SPHINCS-Simpira have a private-key size of 1 088 bytes, a public-key size of 1 056 bytes, and the size of each signature is 41 000 bytes.

The authors of SHPINCS made an optimized implementation available as part of eBACS [4]. To ensure the fairest possible comparison, we use the latest SPHINCS implementation on eBACS in the SUPERCOP benchmarking framework (version 20170105). In particular, we perform 32 runs to measure the average and mean speed of the key generation algorithm (`crypto_sign_keypair`), the signature algorithm (`crypto_sign`) on 59-byte messages, and the verification algorithm (`crypto_sign_open`) on 59-byte messages. The results are shown in Table 1.

Table 1. Cycle counts on Intel Skylake for the `crypto_sign_keypair`, `crypto_sign`, and `crypto_sign_open` operations of both the original SPHINCS and SPHINCS-Simpira, using the SUPERCOP benchmarking framework. Messages are chosen to be 59 bytes long.

	SPHINCS (orig.)	SPHINCS-Simpira	speed-up
Cycles to generate a key pair			
median	2 841 398	1 875 254	1.52×
average	2 844 989	1 878 209	1.51×
Cycles to sign 59 bytes			
median	43 993 732	32 549 062	1.35×
average	44 007 958	32 568 235	1.35×
Cycles to verify 59 bytes			
median	1 283 710	629 086	2.04×
average	1 286 054	631 346	2.04×

7 Comparison with Haraka

Haraka [16] has been proposed as another AES-based hash function to replace the F and H functions of SPHINCS. The first version of Haraka was vulnerable to an attack by Jean [14] due to a weak choice of round constants, but was subsequently updated to prevent the attack.

Haraka was announced before Simpira, and is even faster than Simpira when used in SPHINCS. An additional advantage of the F and H functions of Haraka is that it is optimized for both latency and throughput, whereas Simpira focuses only on optimizing throughput. Although the latency of F and H is not a bottleneck in SPHINCS, having a low latency can be an advantage in other applications.

It is difficult to make a comparison between Haraka and Simpira. Although both AES-based, they are entirely different designs, and make different security claims as well. Simpira claims security against structural distinguishers up to 2^{128} queries in a classical setting, inspired by the hermetic sponge strategy of SHA-3 [5, 23]. From this, the security of F and H in SPHINCS-Simpira against preimage, second-preimage and undetectability attacks follow. Haraka makes a stronger claim against preimage and second-preimage resistance (up to 2^{256} queries in a classical setting), but makes no claim against other non-randomness properties. No explicit claim of undetectability is made for Haraka, but this seems implied as SPHINCS is the target application.

We hope that the different designs and security claims of Haraka and Simpira will be an interesting starting point for discussions, and that this may lead to a better understanding of the pre-quantum and post-quantum security of AES-based designs in general.

8 Conclusion

For modern 64-bit processors that nowadays all have AES instructions, Simpira offers a family of high-throughput cryptographic permutations of various input sizes. This makes it an interesting candidate for the 256-bit-to-256-bit and 512-bit-to-256-bit hash functions that dominate the performance of SPHINCS, when these are combined with a Davies-Meyer feedforward.

However, Simpira only claims security up to 2^{128} queries using classical computers, and no security against quantum computers. By recalling that the complexity of Grover's algorithm depends on the output size of the function, and not on any particular structure of the function, we claim that Simpira has the same security both pre-quantum and post-quantum.

SPHINCS-Simpira then provides the same post-quantum security claims as the original SPHINCS. However, our benchmarks show that SPHINCS-Simpira gives a 1.5× speed-up for generating key pairs, a 1.4× speed-up for signing 59-byte messages, and a 2.0× speed-up for verifying the signatures.

References

1. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.V.: Strengths and Weaknesses of Quantum Computing. SIAM J. Comput. 26(5), 1510–1523 (1997), `https://doi.org/10.1137/S0097539796300933`
2. Bernstein, D.J.: Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? SHARCS '09: Special-purpose Hardware for Attacking Cryptographic Systems pp. 105–116 (2009)
3. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical Stateless Hash-Based Signatures. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria,

April 26-30, 2015, Proceedings, Part I. LNCS, vol. 9056, pp. 368–397. Springer (2015), https://doi.org/10.1007/978-3-662-46800-5_15
4. Bernstein, D.J., Lange, T.: eBACS: ECRYPT benchmarking of cryptographic systems. http://bench.cr.yp.to (accessed 2017-02-21)
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic Sponge Functions, available at http://sponge.noekeon.org/CSF-0.1.pdf
6. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. LNCS, vol. 2442, pp. 320–335. Springer (2002)
7. Brassard, G., Hoyer, P., Tapp, A.: Quantum algorithm for the collision problem. arXiv preprint quant-ph/9705002 (1997)
8. Chang, D., Lee, S., Nandi, M., Yung, M.: Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In: Lai, X., Chen, K. (eds.) Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4284, pp. 283–298. Springer (2006), https://doi.org/10.1007/11935230_19
9. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Information Theory 22(6), 644–654 (1976), https://doi.org/10.1109/TIT.1976.1055638
10. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996. pp. 212–219. ACM (1996), http://doi.acm.org/10.1145/237814.237866
11. Grover, L.K.: Quantum Mechanics Helps in Searching for a Needle in a Haystack. Phys. Rev. Lett. 79, 325–328 (Jul 1997), http://link.aps.org/doi/10.1103/PhysRevLett.79.325
12. Gueron, S.: Intel® Advanced Encryption Standard (AES) New Instructions Set. Available at: https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set (September 2012), Revision 3.01
13. Gueron, S., Mouha, N.: Simpira v2: A Family of Efficient Permutations Using the AES Round Function. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 95–125 (2016), https://doi.org/10.1007/978-3-662-53887-6_4
14. Jean, J.: Cryptanalysis of Haraka. IACR Trans. Symmetric Cryptol. 2016(1), 1–12 (2016), http://tosc.iacr.org/index.php/ToSC/article/view/531
15. Kaplan, M., Leurent, G., Leverrier, A., Naya-Plasencia, M.: Breaking Symmetric Cryptosystems Using Quantum Period Finding. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 207–237. Springer (2016), https://doi.org/10.1007/978-3-662-53008-5_8
16. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka v2 – Efficient Short-Input Hashing for Post-Quantum Applications. IACR Trans. Symmetric Cryptol. 2016(2), 1–29 (2016), http://tosc.iacr.org/index.php/ToSC/article/view/563

17. Kuwakado, H., Morii, M.: Security on the quantum-type Even-Mansour cipher. In: Proceedings of the International Symposium on Information Theory and its Applications, ISITA 2012, Honolulu, HI, USA, October 28-31, 2012. pp. 312–316. IEEE (2012), http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6400943
18. Lamport, L.: Constructing Digital Signatures from a One Way Function. Tech. Rep. SRI-CSL-98, SRI International Computer Science Laboratory (October 1979)
19. Langley, A.: Hash based signatures. https://www.imperialviolet.org/2013/07/18/hashsig.html (2013)
20. Merkle, R.C.: A Certified Digital Signature. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer (1989), https://doi.org/10.1007/0-387-34805-0_21
21. National Institute of Standards and Technology: Recommendation for Key Derivation Using Pseudorandom Functions (Revised) (U.S. Department of Commerce, Washington, D.C.). NIST Special Publication 800-108 (October 2009), https://doi.org/10.6028/NIST.SP.800-108
22. National Institute of Standards and Technology: Digital Signature Standard (DSS) (U.S. Department of Commerce, Washington, D.C.). NIST Federal Information Processing Standards Publication 186-4 (July 2013), https://doi.org/10.6028/NIST.FIPS.186-4
23. National Institute of Standards and Technology: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions (U.S. Department of Commerce, Washington, D.C.). NIST Federal Information Processing Standards Publication 202 (August 2015), https://doi.org/10.6028/NIST.FIPS.202
24. Rivest, R.L., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Commun. ACM 21(2), 120–126 (1978), http://doi.acm.org/10.1145/359340.359342
25. Shor, P.W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. pp. 124–134. IEEE Computer Society (1994), https://doi.org/10.1109/SFCS.1994.365700
26. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comput. 26(5), 1484–1509 (1997), https://doi.org/10.1137/S0097539795293172