SHA FAMILY FUNCTIONS

Stelian DUMITRA
PdD, Department of Economic Informatics Doctoral School
The Bucharest University of Economic Studies
Bucharest, Romania

E-mail: stelian.dumitra@endava.com; stelu20d@yahoo.com

Abstract

This paper presents a general overview regarding SHA family functions. A lot of hash functions were proposed in the last three decades and most of them are based upon the MD construction, especially the MD4 family. The most popular hash functions that belong to the MD4 family are: MD5, the SHA family (SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) and the RIPEMD family (RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320). In 2004, collision attacks against MD5 and SHA-0 were demonstrated by Xiaoyun Wang and one year later she extended a theoretical attack against SHA-1. NIST took the Wang attack into serious consideration and decided to open a competition to develop the next secure hash algorithm, named SHA-3. After five years of competition NIST selected Keccak as winner. Keccak is a family of sponge functions. Also, this paper describes the sponge construction and its security. Various comparisons between SHA-3, SHA-2 and SHA-1 regarding the security strengths, performances and construction details are summarized. The conclusions are presented at the end of the paper.

Keywords: *Keccak, SHA-3, hash function, collision resistance, preimage attack, sponge construction*

1. Introduction

Cryptographic hash functions play an important role in the current cryptographic protocols and they are used to ensure data integrity, data origin authentication, password protection, random number generation and the likes of them. They produce a fixed-length output, *digest*, that can be also treated as a *fingerprint* of the input data [7]. A lot of security protocols and applications use hash functions: digital signature scheme for authentication data such as DSS (Digital Signature Scheme), XML signature, computing MAC (Message Authentication Code)/HMAC (Keyed-Hash Message Authentication Code), secure communication protocols such as SSH (Secure Shell Host), SFTP (Secure File Transfer Protocol), SSL (Secure Socket Layer), IPSec (Internet Protocol Security) etc, Kerberos protocol for authentication and data integrity, PGP (Pretty Good Privacy), S/MIME (Secure/Multipurpose Internet Mail Extensions) for integrity of e-mail messages.

A hash function ($h: M \rightarrow M_h$) is a function that transforms a variable-length input into a fixed-length output – *hash value* (for example, 128, 160, 224, 256, 384, 512 bits). In practice there are two classes of hash functions:

1. *One Way Hash Functions – OWHF*
2. *Collisions Resistant Hash Functions – CRHF*

An OWHF function satisfies the following properties

- Given a digest value $d \in M_h$ it is computationally infeasible to find a message $m \in M$ so that $d = h(m)$.
- Given a message m_1 it is computationally infeasible to find another message $m_2 \neq m_1$ so that $h(m_1) \neq h(m_2)$.

For a CRHF function it is computationally infeasible to find two different messages m_1 and m_2 so that $h(m_1) = h(m_2)$. This means that the digests are almost unique for each given message. The OWHF functions are also known as *one-way weak collision resistance* and CRHF functions are also known as *strong collision resistance*.

Another class of hash functions is MAC (Message Authentication Codes) that is a function of the symmetric key *k* and the message *m*, $m = MAC_k(x)$. A lot of hash functions were proposed in the last three decades and most of them are based upon the MD (Merkle-Damgård) construction [8], especially the MD4 family [1]. The most popular hash functions that belongs to the MD4 family are: MD5 [2], the SHA family (SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) [3] and the RIPEMD family (RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320) [4, 5, 6]. The MD4 digest algorithm was developed by Ronald Rivest and the digest length is 128 bits. All operations are bitwise Boolean functions: AND, OR, XOR and negation. Boer, Bosselaers and Hans Dobbertin demonstrated weaknesses in MD4 and this algorithm is not recommended for secure hashing [9, 10]. A new strengthened version of MD4 was introduced by Rivest in 1991 and computes a 128-bit output, called MD5 and possess a collision resistance of about 2^{64}. As MD4, this presented potential weaknesses because pseudo-collisions were found on its compression function. In 1993, the US National Institute of Standard and Technology (NIST) published a new message digest standard, SHA (Secure Hash Algorithm). The first version was SHA-0 and, in 1994, a new version was published, SHA-1, derived from SHA-0 with some changes. SHA-0 and SHA-1 produce an output length of 160 bit. In the absence of analytical attacks, the maximum collision resistance of SHA-0 and SHA-1 is of about 2^{80}. The known attack on SHA-0 was developed by Joux and Chabaud [11], a differential attack that finds two messages hashing with the same value in about 2^{61}evaluations. In 2000 NIST introduced three more variants of SHA-1: SHA-256, SHA-384 and SHA-512, functions that produce a message digest with a length of: 256, 384 and 512 bits. These functions were adopted as standard, SHA-2, by FIPS in 2002. A new modification of SHA-1 was introduced in 2004, SHA-224, to fit the security level of 3DES. This function was also included in SHA-2 standard.

In 2004, collision attacks against MD5 and SHA-0 were demonstrated by Xiaoyun Wang [12]. One year later Wang extended a theoretical attack against SHA-1 and it was claimed that a collision search would take 2^{69} steps [13]. An improved version of this attack was presented by Wang in August 2005 with the time complexity of 2^{63}(a brute-force search would require 2^{80} operations) [14]. Other cryptographic attacks on SHA-1 were proposed by Christophe De Cannière and Christian Rechberger [15], Grechnikov [16], Stéphane Manuel [17], Cameron McDonald, Philip Hawkes and Josef Pieprzyk [18], Marc Stevens [19].

Although no serious flaws were disclosed against SHA-2, NIST took the Wang attack into serious consideration and decided to open a competition to develop the next secure

hash algorithm, named SHA-3. NIST did not plan to replace SHA-2 with SHA-3, as it considered that both functions should co-exist. Below is the timeline of the SHA-3 selection process [20]:

- November 2, 2007: NIST announces a request for a new cryptographic hash function – SHA-3 [21]
- October 31, 2008: Submission deadline. 64 submissions were received from the international cryptography community.
- December 10, 2008: The first round began. NIST selected 51 algorithms for Round 1.
- July 24, 2009: The second round was announced. NIST selected 14 algorithms for Round 2.
- December 9, 2010: The third round was announced and 5 algorithms were selected:
  - *BLAKE* by Jean-Philippe Aumasson, Luca Henzen, Willi Meier and Raphael C.-W.Phan
  - *Grøstl* by Lars Ramkilde Knudsen, Praveen Gauravaram, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer and Søren S. Thomsen
  - *JH* by Hongjun Wu
  - *Keccak* by Joan Daemen, Guido Bertoni, Michaël Peeters and Gilles Van Assche
  - *Skein* by Bruce Schneier, Niels Ferguson, Stefan Lucks, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jesse Walker, Jon Callas
- October 2, 2012: NIST selected Keccak as winner.

Table 1. General information about SHA-3 finalists and SHA-2 in bits

Algorithm	Domain Extender	Underlying Primitive	Primitive size	Hash size
BLAKE	HAIFA	Block cipher	k=512 b=512	224 256
			k=1024 b=1024	384 512
Grøstl	Grøstl	A pair of permutations	512 512	224 256
			1024 1024	384 512
JH	JH	Permutation	1024	224 256 384 512
Keccak	Sponge	Permutation	1600	224 256 384 512
Skein	UBI	Tweakable block cipher	k=512 b=512 t=128	224 256 384 512
SHA-2	MD	Block cipher	k=512 b=256	224 256
			k=1024 b=512	384 512

2. The SHA-2 Hash Family

The SHA-2 functions: SHA-224, SHA-256, SHA-384, SHA-512 are the next generation of SHA-1 function standardized by US NIST in 2002 [3]. Initially there were two functions in this standard: SHA-256 and SHA-512. Later, in addition, another two truncated versions were standardized: SHA-224 and SHA-384. The SHA-2 functions are described in detail below.

Padding the Message. The message *M* of length *l* bits is right-padded with a binary *"1"* followed by *k* zero bits, followed by *s-bits* suffix containing the binary length of the original message.

$$\begin{aligned} &l+1+k \equiv 448 \, mod \, 512, \; s=64, && \text{for SHA-224 and SHA-256} \\ &l+1+k \equiv 896 \, mod \, 1024, \; s=128, && \text{for SHA-384 and SHA-512} \end{aligned} \tag{1}$$

The length of the padded message should be a multiple of 512 bits. The padded message must be parsed into N-512 bit blocks, for SHA-224/256, respectively N-1024 bit blocks, for SHA-384/512, $M^{(1)}, M^{(2)}, \dots, M^{(N)}$.

Computing the Message Digest. The algorithm uses 64 (resp. 80) round functions for processing a single message block. An input block $M^{(i)}$ is expressed as $M_0^{(i)} M_1^{(i)} .. M_{15}^{(i)}$, where $M_t^{(i)}$ are 32-bit (resp. 64-bit) words. The words of the message schedule are labeled $W_0, W_1, \dots, W_t$, where t denotes the number of the rounds – 1, 63 for SHA-224/256, respectively 79 for SHA-384/512.

After the padding phase, eight working state registers *a, b, c, d, e, f, g, h* are initialized with the $(i-1)^{th}$ hash value $H_0^{(i-1)}, H_1^{(i-1)}, \dots, H_7^{(i-1)}$. The initial hash $H^{(0)}$ must be initialized with 32-bit constants, for SHA-224/256, respectively 64-bit constants for SHA-384/512. The steps for computing the message digest are described below:

Step 1. Preparing the message schedule, $\{W_t\}$***:***

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \; (or \; 79) \end{cases} \tag{2}$$

For SHA-224/256, the functions σ_0 and σ_1 are defined as:

$$\begin{aligned} \sigma_0(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \\ \sigma_1(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \end{aligned} \tag{3}$$

For SHA-384/512, the functions σ_0 and σ_1 are defined as:

$$\begin{aligned} \sigma_0(x) &= ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x) \\ \sigma_1(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x) \end{aligned} \tag{4}$$

Step 2. Initializing the variables a, b, c, d, e, f, g, h with the $(i-1)^{th}$ ***hash value:***

$$a = H_0^{(i-1)}, \dots, h = H_7^{(i-1)} \tag{5}$$

Step 3. For t = 0 to 63, the following values are calculated:

$$
\begin{aligned}
&T_1 = h + \sum_1(e) + Ch(e,f,g) + K_t + W_t \\
&T_2 = \sum_0(a) + Maj(a,b,c) \\
&h = g \\
&g = f \\
&f = e \\
&e = d + T_1 \\
&d = c \\
&c = b \\
&b = a \\
&a = T_1 + T_2
\end{aligned}
\tag{6}
$$

For SHA-224/256, the functions $\sum_0$ and $\sum_1$ are defined as:

$$
\begin{aligned}
\sum_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\
\sum_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)
\end{aligned}
\tag{7}
$$

For SHA-384/512, the functions $\sum_0$ and $\sum_1$ are defined as:

$$
\begin{aligned}
\sum_0(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \\
\sum_1(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)
\end{aligned}
\tag{8}
$$

Ch and *Maj* are two logical functions that operate on 32-bit (64-bit) words and three variables, *x, y, z*.

$$
\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)
\end{aligned}
\tag{9}
$$

K_t are round constants on 32-bit (64-bit) words, where $0 \leq t \leq 63$ or $0 \leq t \leq 79$.

Step 4. Computing the i^{th} intermediate hash value $H^{(i)}$:

$$
H_0^{(i)} = a + H_0^{(i-1)}, \ldots, H_7^{(i)} = h + H_7^{(i-1)} \tag{10}
$$

After processing $M^{(N)}$ the resulting message digest of the message, M, is:

$$
H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)} \tag{11}
$$

SHA-224/SHA-384 is defined in the same manner as SHA-256/SHA-512 with the following differences: uses different constants initialization, $H^{(0)}$ and the message digest is truncated at 224/384 bits as: $H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)}$, respectively

$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)}$.

More details about implementing SHA-2 are described in [3].

3. Merkle-Damgård construction

Merkle-Damgård construction is a hash construction method which was designed by R. Merkle [22] and I. Damgård [23] in 1989.

It transforms a compression function, $h:\{0,1\}^{m_c} \times \{0,1\}^{n} \rightarrow \{0,1\}^{m_c}$, into a hash function. The m_c denotes the size of the chaining value and n denotes the block size for the compression function. Most of the hash functions are built upon MD construction (Figure 1). It begins with a padding step where the message M is padded so that the message length becomes a multiple of message block length, n. The most used procedure is: *the message M of length l bits is right-padded with a binary "1" followed by k zero bits, followed by s-bit suffix containing the binary length of the original message, so that*

$$l + 1 + k + s \equiv 0 (mod\ n). \tag{12}$$

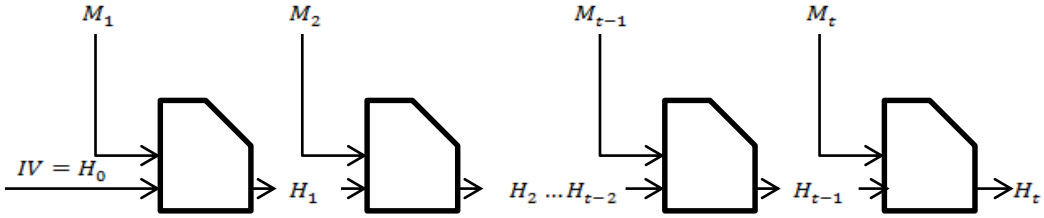

Figure 1. The Merkle-Damgård costruction

The message is divided into block of n bits each, $M = M_1 M_2 \ldots M_t$. An initial chaining value is set for the hash function, publicly known initialization vector, *IV*, and a process is repeated *t* times:

$$\begin{aligned} H_0 &= IV \in \{0,1\}^{m_c} \\ H_i &= h(H_{i-1}, M_i), i = 1, 2, .., t \end{aligned} \tag{13}$$

The final H_t is outputted as the hash value, i.e. $H(M) = H_t$.

If the compression function is collision resistant then the hash function itself is collision resistant, so the collision resistance is preserved. Also, the pre-image resistance and second pre-image resistance of the compression function are preserved [22, 23].

4. Sponge construction

The sponge construction is a mode of operation, based on a fixed-length permutation (or transformation) *f*, a padding rule and a parameter bitrate *r*, which builds a function mapping variable-length input to variable-length output [24]. The permutation *f* operates on a fixed number of bits, the width *b*. The value $c = b - r$ is called the *capacity* [25]. This construction is used for building hash functions and stream ciphers. When it is used as a hash function, called SHA-2 Replacement Mode, the sponge function receives a variable-length input and produces a fixed-length output (SHA-3 224/256/384/512). If a sponge function is used as a stream cipher, called Variable-length Output Mode, it receives a fixed-length input and produces a variable-length output [26]. Also, the sponge functions are used for generating pseudo-random bits. A sponge construction can be expressed as a random permutation,

where the construction is called a P-sponge (random sponge), or random function, where the construction is called a T-sponge [27]. Before the sponge construction phase is performed a pre-processing phase where all the bits of the state are initialized to zero, the message is padded to a multiple of *r* and cut into block of *r* bits.

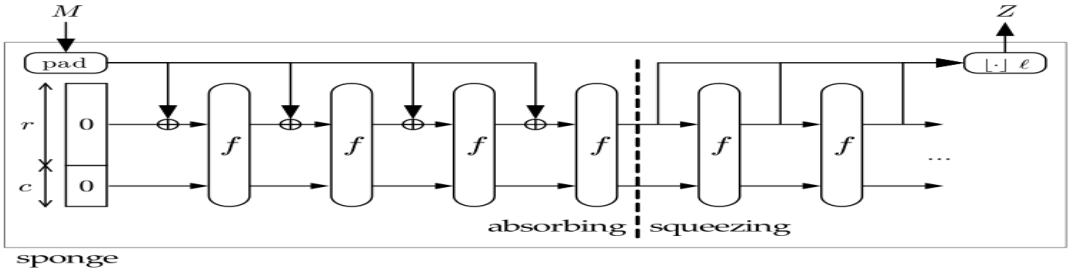

Figure 2. The sponge construction (Source: [24], p.13)

The construction consists of two phases: the absorbing phase and the squeezing phase.

- In the absorbing phase the *r*-bit input message blocks are XOR-ed and overwritten to the state, the *f* is applied to this state. After all blocks are processed, second phase is applied.
- In the squeezing phase a part of the state is returned as output blocks and *f* is applied to the state. The process is repeated in the same manner until the number of the chosen blocks by the user is achieved.

The difference between the compression functions of MD construction and the functions from sponge construction is that a function used in sponge construction maps 1 bit input into 1 bit output. When *f* is expressed as a random permutation the lower bound for the complexity of a collision is $min(2^{n/2}, 2^{c/2})$ and of a pre-image and second pre-image is $min(2^{n}, 2^{c/2})$, where *n* is the hash size. If c ≥ 2n and *f* is a random function, then the sponge construction is differentiable from a random oracle, the strength against signature forgery is increased from $2^{n/2}$ to 2^{n}. More about theory of sponge constructions and their security properties are provided in [24].

5. Keccak hash function

Keccak, became the new SHA-3 standard, is a family of sponge functions. It can be used in two principle modes:

- **SHA-2 Replacement Mode** – SHA-3 produces a fixed-length output of 224, 256, 384 or 512 bits
- **Variable-length Output Mode** – SHA-3 can generate arbitrarily many output bits, so it can be used as a stream cipher or pseudorandom bit generator.

In the pre-processing phase the message *m* is padded as follows:

$pad(m) = m \parallel P10^{*}1$, where P is the bit string representation of the message m, followed by a 1, then by a smallest number of 0s and then again a 1, so that $len(P10^{*}1)\ mod\ r \equiv 0$.

In the case of SHA-3, the width of the state, *b*, is:

$b = r + c = 5 \cdot 5 \cdot 2^{l},\ l = 0,1,\ldots,6$, so $b \in \{25, 50, 100, 200, 400, 800, 1600\}$

The values $b = 25$ and $b = 50$ are not used in practice. For SHA-3 a state of $b = 1600$ bits is used and $r \in \{1088, 1344\}$. The parameters of SHA-3 are represented in bits, in Table 2:

Table 2. The parameters of SHA-3

b (state)	r	c	security level	hash output
1600	1344	256	128	224
1600	1344	256	128	256
1600	1088	512	256	384
1600	1088	512	256	512

5.1 The Keccak-f permutation

Keccak-f[b] is a permutation over Z_2^b. The state (figure-3) consists of a 5×5 array of 64-bit words, a three-dimensional array of elements of *GF(2)*, *a[5][5][w]*, where $w = 2^l$. An element is denoted as *a[x][y][z]*, $x, y \in Z_5$, $z \in Z_w$. The string representation of the state is denoted as *s* and its bits are indexed from *0* to *b-1*.

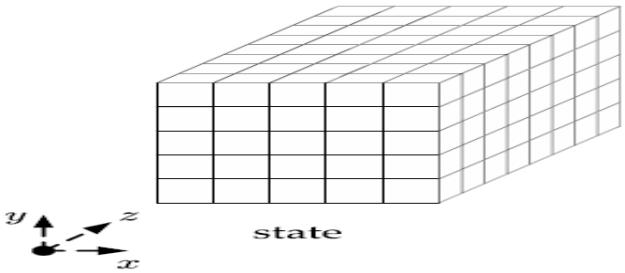

Figure 3. The state of Keccak where each small cube represents one bit (Source: [25], p.11)

The mapping between the bits of *a* and those *s* is:

$$s[w(5y + x) + z] = a[x][y][z] \tag{14}$$

The *Keccak-f[b]* function consists of $n_r = 12 + 2l$ rounds, where each round consists of *b* bits.

Table 3. Number of rounds within Keccak-f (for SHA-3: b=1600, $n_r = 24$)

state width b	number of rounds n_r
25	12
50	14
100	16
200	18
400	20
800	22
1600	24

Each round consists of five sub-rounds, denoted by Greek letters: $\theta(theta), \rho(rho), \pi(pi), \chi(chi)\ and\ \iota(iota)$ [25], [28].

$\theta(a)$:

$$c[x] = a[x,0] \oplus a[x,1] \oplus a[x,2] \oplus a[x,3] \oplus a[x,4], x = 0,1,2,3,4$$
$$d[x] = c[x-1] \oplus rot(c[x+1],1), \quad x = 0,1,2,3,4$$
$$a[x,y] = a[x,y] \oplus d[x], \quad x,y = 0,1,2,3,4$$

$\rho(a)$:

$$a[x][y][z] = a[x][y][z-(t+1)(t+2)/2], where\ 0 \le t < 24, \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

$\pi(a)$:

$$a[x][y] = a[x'][y'],\ where \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} \quad (15)$$

$\chi(a)$:

$$a[x] = a[x] \oplus (a[x+1] \oplus 1) \wedge a[x+2]$$

$\iota(a)$:

$a[0,0] = a[0,0] \oplus RC[i_r]$, where $RC[i_r]$ - round constants, $0 \le i_r \le n_r - 1$

5.2 Keccak vs. SHA-2

After collision attacks against MD5 and SHA-0 were demonstrated by Wang in 2004 and theoretical attacks against SHA-1 were proved to exist serious doubts that these will pose a practical threat against SHA-2 arose. NIST decided to simultaneously use two standards: SHA-2 and SHA-3. The SHA-3 functions are alternatives to the SHA-2 functions. These two standards use different design principles: SHA-2 uses MD principle and Keccak uses Sponge construction. The security strengths, performances and a various details regarding construction of the functions SHA-1, SHA-2 and SHA-3 are summarized in Table 4.

Table 4. Comparison of SHA functions (Source: [28, 29])

Function	Output Size (bits)	Internal State size (bits)	Block size (bits)	Max mes-sage size(bits)	Rounds	Example Perfor-mance (MiB/s)[29]	Security Strengths in Bits		
							Collision	Preimage	2nd Preimage
SHA-1	160	160 (5x32)	512	$2^{64}-1$	80	192	<80	162	160-L(M)
SHA-224	224	256 (8x32)	512	$2^{64}-1$	64	139	112	224	min(224, 256-L(M))
SHA-256	256						128	256	256-L(M)
SHA-384	384	512 (8x64)	1024	$2^{128}-1$	80	154	192	384	384
SHA-512	512						256	512	512-L(M)
SHA-512/224	224						112	224	224
SHA-512/256	256						128	256	256
SHA3-224	224	1600 (5x5x64)	1152	Unlimited	24	-	112	224	224
SHA3-256	256		1088				128	256	256
SHA3-384	384		832				192	384	384
SHA3-512	512		576				256	512	512
SHAKE128	d		1344				min(d/2, 128)	≥min(d, 128)	min(d, 128)
SHAKE256	d		1088				min(d/2, 256)	≥min(d, 256)	min(d, 256)

Keccak, as hash function, provides 224, 256, 384 and 512 bit output sizes as well as SHA-2. Also, Keccak can be used as a stream cipher or pseudorandom bit generator, so it

supports variable output length and plays well with HMAC and KDFs. Both, SHA-2 and SHA-3, support N/2 bit collision resistance, N preimage resistance. SHA-3, SHA-384, SHA-512/224 and SHA-512/256 support N bit second preimage resistance, while SHA-256 and SHA-512 support N-L(M) second preimage resistance, where N is the output size in bits and *L(M)* is a function defined as $[log_2(len(M) / B)]$, with B - the block length of the function. Keccak is very hardware friendly and is better suited for embedded applications that are power or cost constrained, but is slower than SHA-2 in software area, it overall has a good performance, fairly high quality, in-depth analysis [31].

Many performance comparisons of the SHA-3 finalists and SHA-2 can be found at [20].

5.3 Security of Keccak

The SHA-3 hash functions were designed to resist collision, pre-image, second preimage or length-extension attacks, resistance which should be equal or exceeds the resistance that the corresponding SHA-2 functions provide [28]. More about the theory of sponge construction and their security properties can be found at [24], [30]. The security of Keccak has been thoroughly researched by a number of cryptanalysts [20].

Aumasson and Khovratovich provided two possible distinguishers on reduced-round Keccak-f[1600]. First, they detected non-ideal behavior in the algebraic description of the permutation applying cube-testers. Second, the authors tried to solve the constrained-input constrained-output (CICO) problem using automated algebraic techniques [32, 33].

Aumasson and Meier presented *zero-sum distinguishers*. This distinguisher was applied to the inner permutation of the hash function of Keccak and succeeded up to 16 rounds [33, 34].

Boura and Canteaut extended the *zero-sum distinguisher* of Aumasson and Meier to 18 rounds by analyzing the Walsh spectrum of the non-linear part and bounding the degree of the rounds more tightly [33], [35].

Boura and Canteaut extended their zero-sum distinguishers to 20 rounds [33], [36].

Morawiecki and Srebrny used SAT-solver techniques to find preimages for three rounds of Keccak, with 40 unknown message bits [33], [37].

Various research papers regarding the security analysis of Keccak can be found at [33].

6 Conclusions

A general overview regarding SHA family functions was presented. A brief design of SHA-2 and Keccak algorithms was described. Many cryptographic attacks against MD5 and SHA-0 were demonstrated by cryptanalysts and these functions were finally broken. Wang extended a theoretical attack against SHA-1. After these attacks, NIST opened a new competition for the next secure hash algorithm, named SHA-3. Also, NIST decided to simultaneously use two standards: SHA-2 and SHA-3. Five finalists in this competition are BLAKE, Grøstl, JH, Keccak and Skein. Keccak was announced as the winner. SHA-2 and Keccak are designed completely differently: SHA-2 uses MD construction and Davies-Meyer compression function, while Keccak is based on the Sponge construction. If an attack could work on SHA-2, the same attack would not work on SHA-3. Both functions support the same hash lengths and Keccak can be used as a stream cipher or pseudorandom bit generator because it supports variable output length. Also, Keccak has higher performance in hardware implementations than SHA-2. Various comparisons regarding construction, performance and security strength were summarized.

References

1. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family, Federal Register, Vol. 72, No. 212, 62212, http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf, Friday, November 2, 2007
2. Aumasson, J.P., Khovratovich, D., **First Analysis of Keccak**, comment on the NIST Hash Competition, https://131002.net/data/papers/AK09.pdf, 2009
3. Aumasson, J.P., Meier, W., **Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi**, CHES 2009 rump session, https://131002.net/data/papers/AM09.pdf, 2009
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., **Cryptographic sponge functions**, version 0.1, http://sponge.noekeon.org/CSF-0.1.pdf, January 2011
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., **The KECCAK reference**, version 3.0, http://keccak.noekeon.org/Keccak-reference-3.0.pdf, January 2011
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., **On the security of the keyed sponge construction**, Symmetric Key Encryption Workshop, http://sponge.noekeon.org/SpongeKeyed.pdf, February 2011
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., **The Keccak sponge function family, Third-party cryptanalysis**, http://keccak.noekeon.org/third_party.html, 2012
8. Boer, B., Bosselaers, A., **An attack on the last two rounds of MD4**, Advances in Cryptology – Crypto'91, LNCS, vol. 576, Springer, Berlin Heidelberg, 1992, pp 194-203
9. Boura, C., Canteaut, A., **A zero-sum property for the Keccak-f permutation with 18 rounds**, comment on the NIST Hash Competition, https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/zero_sum.pdf, 2010
10. Boura, C., Canteaut, A., **Zero-Sum Distinguishers for Iterated Permutations and Application to Keccak-f and Hamsi-256**, Selected Areas in Cryptography, LNCS, vol. 6544, Springer, Berlin Heidelberg, 2011, pp 1-17
11. Burr, B.: National Institute of Standards and Technology (NIST), **SHA3 WHERE WE'VE BEEN WHERE WE'RE GOING**, http://csrc.nist.gov/groups/ST/hash/sha-3/documents/burr_dimacs2013_presentation.pdf, 2013
12. Chabaud, F., Joux, A., **Differential collisions in SHA-0**, Advances in Cryptology – Crypto'98, LNCS, vol. 1462, Springer, Berlin Heidelberg, 1998, pp 56-71
13. Damgård, I., **A Design Principle for Hash Functions**, Advances in Cryptology — CRYPTO' 89 Proceedings, LNCS, vol. 435, Springer, Berlin Heidelberg, 1990, pp 416-427.
14. De Cannière, C., Rechberger, C., **Finding SHA-1 Characteristics: General Results and Applications**, Advances in Cryptology – ASIACRYPT 2006, LNCS, vol. 4284, Springer, Berlin Heidelberg, 2006, pp 1-20.
15. Dobbertin, H., **Cryptanalysis of MD4**, Journal of Cryptology, LNCS, vol. 11, issue 4, Springer, Berlin Heidelberg, 1998, pp 253-271
16. Dobbertin, H., Bosselaers, A. and Preneel, B., **The hash function RIPEMD-160**, http://homes.esat.kuleuven.be/~bosselae/ripemd160.html, February 2012
17. Gauravaram, P., Millan, W., Dawson, E., Viswanathan, K., **Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction**, LNCS, vol. 4058, Springer, Berlin Heidelberg, 2006, pp 407-420

18. Grechnikov, E.A., **Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics**, Cryptology ePrint Archive: Report 2010/413, 2010
19. Madhuravani, B., Murthy, D.S.R., **Cryptographic Hash Functions: SHA Family**, International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 2, issue 4, Springer, Berlin Heidelberg, 2013
20. Manuel, S., **Classification and generation of disturbance vectors for collision attacks against SHA-1**, Designs, Codes and Cryptography, vol. 59, issue 1-3, Springer, US, 2011, pp 247-263.
21. Matusiewicz, K., **Analysis of Modern Dedicated Cryptographic Hash Functions, PhD thesis,** Macquarie University, 2007.
22. McDonald, C., Hawkes, P., Pieprzyk, J., **Constructing Nonlinear Differentials in SHA-1**, Cryptology ePrint Archive: Report 2009/259, 2009
23. Mendel, F., Nad, T., Scherz, S., Schläffer, M., **Differential Attacks on Reduced RIPEMD-160**, LNCS, vol. 7483, Springer, Berlin Heidelberg, 2012, pp 23-38
24. Merkle, R., **One Way Hash Functions and DES**, Advances in Cryptology - CRYPTO' 89 Proceedings, LNCS, vol. 435, Springer, Berlin Heidelberg, 1990, pp 428-446.
25. Morawiecki, P., Srebrny, M., **A SAT-based preimage analysis of reduced Keccak hash functions**, Cryptology ePrint Archive: Report 2010/285, (2010)
26. National Institute of Standards and Technology (NIST), **FIPS 180-4**, http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, 2012.
27. National Institute of Standards and Technology (NIST), **Third Round Report of the SHA-3 Cryptographic Hash Algorithm Competition, NISTIR 7896**, http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf, 2012
28. National Institute of Standards and Technology (NIST), **SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, DRAFT FIPS PUB 202**, http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf, 2014
29. Paar, C., Pelzl, J., **SHA-3 and The Hash Function Keccak**, Springer, http://professor.unisinos.br/linds/teoinfo/Keccak.pdf, 2010
30. Rivest, R.L., **The MD4 message digest algorithm**, Advances in Cryptology - Crypto'90, Springer-Verlag, 1991, pp 303-311
31. Rivest, R.L., **The MD5 message-digest algorithm**, Request for Comments (RFC 1320), Internet Activities Board, Internet Privacy Task Force, 1992.
32. Stevens, M., **Counter-Cryptanalysis**, Advances in Cryptology – CRYPTO 2013, LNCS, vol. 8042, Springer, Berlin Heidelberg, 2013, pp 129-146.
33. Wang, G., Wang, S., **Preimage Attack on Hash Function RIPEMD**, LNCS, vol. 5451, Springer, Berlin Heidelberg, 2009, pp 274-284
34. Wang, X., Yu, H., **How to Break MD5 and Other Hash Functions**, Advances in Cryptology – EUROCRYPT 2005, LNCS, vol. 3494, Springer, Berlin Heidelberg, 2005, pp 19-35
35. Wang, X., Yin, Y.L., Yu, H., **Finding Collisions in the Full SHA-1**, Shoup, V.(ed.) CRYPTO 2005, LNCS, vol. 3621, Springer, Berlin Heidelberg, 2005, pp 17-36.
36. Wang, X., Yao, A., Yao, F., **Cryptanalysis of SHA-1**, Cryptographic Hash Workshop hosted, NIST, October 2005
37. http://en.wikipedia.org/wiki/SHA-3