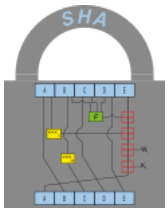
SHA-3

الگوریتم هش امن 3 (**SHA-3** آخرین عضو است) امن الگوریتم خانواده ای از استانداردها، Hash در تاریخ 5 اوت، سال NIST منتشر شده توسط 2015. [4][5] مرجع پیاده سازی کد منبع به اختصاص داده شده بود مالکیت عمومی از طریق چشم پوشی . [6] اگر چه بخشی از همان CC0 است داخلی کاملا SHA-3 ،مجموعه از استانداردها SHA-2 و SHA-1 مانند ساختار از MD5 متفاوت از .

الگوریتم هش امن



مفاهیم

توابع هش · SHA · DSA

استانداردهای اصلی

SHA-0 · SHA-1 · SHA-2 · SHA-3

---

v t e

SHA-3 (Keccak)	
عمومی	
طراحان	گیدو برتون، جوآن دیمن ، مایکل پترز و ژیل ون Assche .
اولین بار منتشر شد	2015
سلسله	(SHA-0)، SHA-1 ، SHA-2 ، SHA-3
صدور گواهینامه	FIPS PUB 202
جزئیات	
اندازه بروزنامه	خودسرانه
ساختار	ساخت اسفنج
دور	24
سرعت	12.6 پیکسل بر روی دستگاه معمولی x86-64

برای Keccak-f [1600] و XORing 1024 بیتی [1] که تقریبا با SHA3-256 مطابقت دارد.
بهترین انتقاد عمومی
Preimage attack on Keccak-512 reduced to 8 rounds, requiring $2^{511.5}$ time and 2^{508} memory[2] Zero-sum distinguishers exist for the full 24-round Keccak-f[1600], though they cannot be used to attack the hash function itself[3]

SHA-3 is a subset of the broader cryptographic primitive family **Keccak** (/ˈkɛtʃæk/, or /ˈkɛtʃɑːk/),[7][8] designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche, building upon RadioGatún. Keccak's authors have proposed additional uses for the function, not (yet) standardized by NIST, including a stream cipher, an

authenticated encryption system, a "tree" hashing scheme for faster hashing on certain architectures,[9][10] and AEAD ciphers Keyak and Ketje.[11][12]

Keccak is based on a novel approach called sponge construction.[13] Sponge construction is based on a wide random function or random permutation, and allows inputting ("absorbing" in sponge terminology) any amount of data, and outputting ("squeezing") any amount of data, while acting as a pseudorandom function with regard to all previous inputs. This leads to great flexibility.

NIST does not currently plan to withdraw SHA-2 or remove it from the revised

Secure Hash Standard. The purpose of SHA-3 is that it can be directly substituted for SHA-2 in current applications if necessary, and to significantly improve the robustness of NIST's overall hash algorithm toolkit.[14]

History

The Keccak algorithm is the work of Guido Bertoni, Joan Daemen (who also co-designed the Rijndael cipher with Vincent Rijmen), Michael Peeters, and Gilles Van Assche. It is based on earlier hash function designs PANAMA and RadioGatún. PANAMA was designed by Daemen and Craig Clapp in 1998.

RadioGatún, a successor of PANAMA, was designed by Daemen, Peeters, and Van Assche, and was presented at the NIST Hash Workshop in 2006.[15]

In 2006 NIST started to organize the NIST hash function competition to create a new hash standard, SHA-3. SHA-3 is not meant to replace SHA-2, as no significant attack on SHA-2 has been demonstrated. Because of the successful attacks on MD5, SHA-0 and SHA-1,[16] NIST perceived a need for an alternative, dissimilar cryptographic hash, which became SHA-3.

After a setup period, admissions were to be submitted by the end of 2008. Keccak

was accepted as one of the 51 candidates. In July 2009, 14 algorithms were selected for the second round. Keccak advanced to the last round in December 2010.[17]

During the competition, entrants were permitted to "tweak" their algorithms to address issues that were discovered. Changes that have been made to Keccak are:[18][19]

- The number of rounds was increased from $12 + \ell$ to $12 + 2\ell$ to be more conservative about security.
- The message padding was changed from a more complex scheme to the

simple 10^*1 pattern described below.

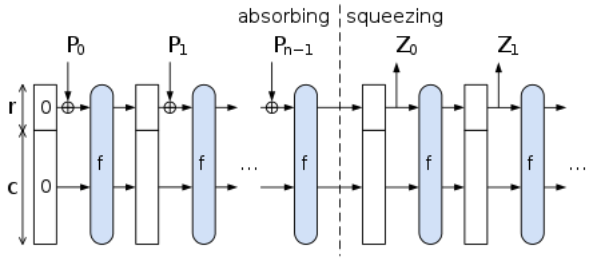
- The rate *r* was increased to the security limit, rather than rounding down to the nearest power of 2.

On October 2, 2012, Keccak was selected as the winner of the competition.[7]

In 2014, the NIST published a draft FIPS 202 "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions".[20] FIPS 202 was approved on August 5, 2015.[21]

On August 5, 2015 NIST announced that SHA-3 had become a hashing standard.[22]

Design



The sponge construction for hash functions. P_i are input, Z_i are hashed output. The unused "capacity" c should be twice the desired resistance to collision or preimage attacks.

SHA-3 uses the sponge construction,[13][23] in which data is "absorbed" into the sponge, then the result is "squeezed" out. In the absorbing phase, message blocks are XORed into a subset of the state, which is then transformed as a whole using a permutation function *f*. In the "squeeze"

phase, output blocks are read from the same subset of the state, alternated with the state transformation function *f*. The size of the part of the state that is written and read is called the "rate" (denoted *r*), and the size of the part that is untouched by input/output is called the "capacity" (denoted *c*). The capacity determines the security of the scheme. The maximum security level is half the capacity.

Given an input bit string *N*, a padding function pad, a permutation function *f* that operates on bit blocks of width *b*, a rate *r* and an output length *d*, we have capacity $c = b - r$ and the sponge construction $Z = \text{sponge}[f,\text{pad},r](N,d)$,

yielding a bit string Z of length d, works as follows:[24]:18

- pad the input N using the pad function, yielding a padded bit string P with a length divisible by r (such that $n = \text{len}(P)/r$ is integer),
- break P into n consecutive r-bit pieces $P_0, \ldots, P_{n-1}$
- initialize the state S to a string of b 0 bits.
- absorb the input into the state: For each block P_i,
    - extend P_i at the end by a string of c 0 bits, yielding one of length b,
    - XOR that with S and

        - apply the block permutation *f* to the result, yielding a new state *S*
- initialize *Z* to be the empty string
- while the length of *Z* is less than *d*:
    - append the first *r* bits of *S* to *Z*
    - if *Z* is still less than *d* bits long, apply *f* to *S*, yielding a new state *S*.
- truncate *Z* to *d* bits

The fact that the internal state *S* contains *c* additional bits of information in addition to what is output to *Z* prevents the length extension attacks that SHA-2, SHA-1, MD5 and other hashes based on the Merkle–Damgård construction are susceptible to.

In SHA-3, the state *S* consists of a 5 × 5 array of *w* = 64-bit words, *b* = 5 × 5 × *w* = 5 × 5 × 64 = 1600 bits total. Keccak is also defined for smaller power-of-2 word sizes *w* down to 1 bit (25 bits total state). Small state sizes can be used to test cryptanalytic attacks, and intermediate state sizes (from *w* = 8, 200 bits, to *w* = 32, 800 bits) can be used in practical, lightweight applications.[11][12]

For SHA-3-224, SHA-3-256, SHA-3-384, and SHA-3-512 instances, r is greater than d, so there is no need for additional block permutations in the squeezing phase; the leading d bits of the state are the desired hash. However, SHAKE-128

and SHAKE-256 allow an arbitrary output length, which is useful in applications such as optimal asymmetric encryption padding.

Padding

To ensure the message can be evenly divided into r-bit blocks, padding is required. SHA-3 uses the pattern 10^*1 in its padding function: a 1 bit, followed by zero or more 0 bits (maximum $r - 1$) and a final 1 bit.

The maximum of $r - 1$ 0 bits occurs when the last message block is $r - 1$ bits long. Then another block is added after

the initial 1 bit, containing $r - 1$ 0 bits before the final 1 bit.

The two 1 bits will be added even if the length of the message is already divisible by *r*.[24]:5.1 In this case, another block is added to the message, containing a 1 bit, followed by a block of $r - 2$ 0 bits and another 1 bit. This is necessary so that a message with length divisible by *r* ending in something that looks like padding does not produce the same hash as the message with those bits removed.

The initial 1 bit is required so messages differing only in a few additional 0 bits at the end do not produce the same hash.

The position of the final 1 bit indicates which rate *r* was used (multi-rate padding), which is required for the security proof to work for different hash variants. Without it, different hash variants of the same short message would be the same up to truncation.

The block permutation

The block transformation *f*, which is Keccak-f[1600] for SHA-3, is a permutation that uses **xor**, **and** and **not** operations, and is designed for easy implementation in both software and hardware.

It is defined for any power-of-two word size, $w = 2^{\ell}$ bits. The main SHA-3 submission uses 64-bit words, $\ell = 6$.

The state can be considered to be a $5 \times 5 \times w$ array of bits. Let $a[i][j][k]$ be bit $(5i + j) \times w + k$ of the input, using a little-endian bit numbering convention and row-major indexing. I.e. i selects the row, j the column, and k the bit.

Index arithmetic is performed modulo 5 for the first two dimensions and modulo w for the third.

The basic block permutation function consists of $12 + 2\ell$ rounds of five steps, each individually very simple:

θ

Compute the parity of each of the $5w$ (320, when $w = 64$) 5-bit columns, and exclusive-or that into two nearby columns in a regular pattern. To be precise,

$a[i][j][k] \leftarrow a[i][j][k] \oplus \text{parity(a[0...4]}$

ρ

Bitwise rotate each of the 25 words by a different triangular number 0, 1, 3, 6, 10, 15, ... To be precise, *a*[0][0] is not rotated, and for all $0 \le t < 24$,

$a[i][j][k] \leftarrow a[i][j][k-(t+1)(t+2)/2]$,

where $\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

π

Permute the 25 words in a fixed pattern. $a[j][2i+3j] \leftarrow a[i][j]$.

χ

Bitwise combine along rows, using $x \leftarrow x \oplus (\neg y \ \& \ z)$. To be precise, $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k]$. This is the only non-linear operation in SHA-3.

ι

Exclusive-or a round constant into one word of the state. To be precise, in round n, for $0 \le m \le \ell$, $a[0][0][2^m-1]$ is exclusive-ORed with bit $m + 7n$ of a degree-8 LFSR sequence. This breaks the symmetry that is preserved by the other steps.

Speed

The speed of SHA-3 hashing of long messages is dominated by the computation of f = Keccak-f[1600] and XORing S with the extended P_i, an operation on b = 1600 bits. However, since the last c bits of the extended P_i are 0 anyway, and XOR with 0 is a noop, it is sufficient to perform XOR operations only for r bits ($r = 1600 - 2 \times 224 = 1152$ bits for SHA3-224, 1088 bits for SHA3-256, 832 bits for SHA3-384 and 576 bits for SHA3-512). The lower r is (and, conversely, the higher $c = b - r = 1600 - r$), the less efficient but more secure the hashing becomes since fewer bits of the

message can be XORed into the state (a quick operation) before each application of the computationally expensive *f*. The authors report the following speeds for software implementations of Keccak-f[1600] plus XORing 1024 bits,[1] which roughly corresponds to SHA3-256:

- 57.4 cpb on IA-32, Intel Pentium 3[25]
- 41 cpb on IA-32+MMX, Intel Pentium 3
- 20 cpb on IA-32+SSE, Intel Core 2 Duo or AMD Athlon 64
- 12.6 cpb on a typical x86-64-based machine
- 6–7 cpb on IA-64

For the exact SHA3-256 on x86-64, Bernstein measures 11.7–12.25 cpb depending on the CPU.[26]:7 SHA-3 has been criticized for being slow in software – SHA2-512 is more than twice as fast as SHA3-512 and SHA-1 is more than three times as fast on an Intel Skylake processor clocked at 3.2 GHz.[27] The authors have reacted to this criticism by suggesting to use SHAKE128 and SHAKE256 instead of SHA3-256 and SHA3-512, at the expense of cutting the preimage resistance in half (but while keeping the collision resistance). With this, performance is on par with SHA2-256 and SHA2-512. To further increase speed, the authors suggest using

KangarooTwelve, a variant of Keccak with half the number of rounds (trading safety margin for speed) and using parallelizable tree hashing to exploit the availability of parallelism in the processor.

However, in hardware implementations, SHA-3 is notably faster than all other finalists,[28] and also faster than SHA-2 and SHA-1.[27]

Instances

The NIST standard defines the following instances, for message *M* and output length *d*:[24]:20,23

Instance	Output size *d*	rate *r* = block size	capacity *c*	Definition	Security Strengths in Bits		
					Collision	Preimage	2nd Preimage
SHA3-224(*M*)	224	1152	448	Keccak[448] (*M* || 01, 224)	112	224	224
SHA3-256(*M*)	256	1088	512	Keccak[512] (*M* || 01, 256)	128	256	256
SHA3-384(*M*)	384	832	768	Keccak[768] (*M* || 01, 384)	192	384	384
SHA3-512(*M*)	512	576	1024	Keccak[1024] (*M* || 01, 512)	256	512	512
SHAKE128(*M*, *d*)	*d*	1344	256	Keccak[256] (*M* || 1111, *d*)	min(*d*/2,128)	≥min(*d*,128)	min(*d*,128)
SHAKE256(*M*, *d*)	*d*	1088	512	Keccak[512] (*M* || 1111, *d*)	min(*d*/2,256)	≥min(*d*,256)	min(*d*,256)

With the following definitions

- Keccak[*c*](*N*, *d*) = sponge[Keccak-f[1600], pad10*1, *r*](*N*, *d*)[24]:20
- Keccak-f[1600] = Keccak-p[1600, 24][24]:17
- *c* is the capacity
- *r* is the rate = 1600 − *c*

- *N* is the input bit string

Note that the appended postfixes are written as bit strings, not hexadecimal digits.

The SHA-3 instances are the drop-in replacements for SHA-2, with identical security claims. SHAKE instances are so called XOF's, Extendable Output Functions. For example, SHAKE128(M, 256) can be used as a hash function with a 256 bit length and 128 bit overall security.

Note that all instances append some bits to the message, the rightmost of which represent the domain separation suffix.

The purpose of this is to ensure that it is not possible to construct messages that produce the same hash output for different applications of the Keccak hash function. The following domain separation suffixes exist:[24][29]

Suffix	Meaning
...0	reserved for future use
01	SHA-3
...11	RawSHAKE

RawSHAKE is the basis for the Sakura coding for tree hashing, which has not been standardized yet. However, the SHAKE suffix has been carefully chosen so that it is forward compatible with Sakura. Sakura appends 0 for a chaining hop or 1 for a message, then 10^*0 for a non-final (inner) node or 1 for a final

node, before it applies RawSHAKE. Sequential hashing corresponds to a hop tree with a single message node, which means that 11 is appended to the message before RawSHAKE is applied. Thus, the SHAKE XOFs append 1111 to the message, i.e., 1 for message, 1 for final node, and 11 for the RawSHAKE domain separation suffix.[29]:16

Since 10^*1 padding always adds at least two bits, in byte aligned libraries there are always six unused zero bits. Therefore, these appended extra bits never make the padded message longer.

Security against quantum

attacks

There is a general result (Grover's algorithm) that quantum computers can perform a structured preimage attack in $\sqrt{2^d} = 2^{d/2}$, while a classical brute-force attack needs 2^d. A structured preimage attack implies a second preimage attack[30] and thus a collision attack. A quantum computer can also perform a birthday attack, thus break collision resistance, in $\sqrt[3]{2^d} = 2^{d/3}$[31] (although that is disputed[32]). Noting that the maximum strength can be *c/2*, this gives the following upper[33] bounds on the quantum security of SHA-3:

Instance	Security Strengths in Bits			
	Collision (Brassard et al.)	Collision (Bernstein)	Preimage	2nd Preimage
SHA3-224(*M*)	74⅔	112	112	112
SHA3-256(*M*)	85⅓	128	128	128
SHA3-384(*M*)	128	192	192	192
SHA3-512(*M*)	170⅔	256	256	256
SHAKE128(*M*, *d*)	min(*d*/3,128)	min(*d*/2,128)	≥min(*d*/2,128)	min(*d*/2,128)
SHAKE256(*M*, *d*)	min(*d*/3,256)	min(*d*/2,256)	≥min(*d*/2,256)	min(*d*/2,256)

It has been shown that the Merkle–Damgård construction, as used by SHA-2, is collapsing and, by consequence, quantum collision-resistant,[34] but for the sponge construction used by SHA-3, the authors provide proofs only for the case that the block function *f* is not efficiently invertible; Keccak-f[1600], however, is efficiently invertible, and so their proof does not apply.[35]

Capacity change controversy

In February 2013 at the RSA Conference, and then in August 2013 at CHES, NIST announced they would select different values for the capacity, i.e. the security parameter, for the SHA-3 standard, compared to the submission.[36][37] The changes caused some turmoil.

The hash function competition called for hash functions at least as secure as the SHA-2 instances. It means that a *d*-bit output should have *d*/2-bit resistance to collision attacks and *d*-bit resistance to preimage attacks, the maximum achievable for *d* bits of output. Keccak's security proof allows an adjustable level

of security based on a "capacity" c, providing $c/2$-bit resistance to both collision and preimage attacks. To meet the original competition rules, Keccak's authors proposed $c=2d$. The announced change was to accept the same $d/2$-bit security for all forms of attack and standardize $c=d$. This would have sped up Keccak by allowing an additional d bits of input to be hashed each iteration. However, the hash functions would not have been drop-in replacements with the same preimage resistance as SHA-2 anymore; it would have been cut in half, making it vulnerable to advances in quantum computing, which effectively would cut it in half once more.[30]

In September 2013, Daniel J. Bernstein suggested on the NIST hash-forum mailing list[38] to strengthen the security to the 576-bit capacity that was originally proposed as the default Keccak, in addition to and not included in the SHA-3 specifications.[39] This would have provided at least a SHA3-224 and SHA3-256 with the same preimage resistance as their SHA-2 predecessors, but SHA3-384 and SHA3-512 would have had significantly less preimage resistance than theirs. In late September, the Keccak team responded by stating that they had proposed 128-bit security by setting $c = 256$ as an option already in their SHA-3 proposal.[40] Although the reduced

capacity was justifiable in their opinion, in the light of the negative response, they proposed raising the capacity to $c = 512$ bits for all instances. This would be as much as any previous standard up to the 256-bit security level, while providing reasonable efficiency,[41] but not the 384/512 bit preimage resistance offered by SHA2-384/512. The authors tried to justify that with the claim that "claiming or relying on security strength levels above 256 bits is meaningless."

In early October 2013, Bruce Schneier criticized NIST's decision on the basis of its possible detrimental effects on the acceptance of the algorithm, saying:

> *There is too much mistrust in the air. NIST risks publishing an algorithm that no one will trust and no one (except those forced) will use.*[42]

Paul Crowley, a cryptographer and senior developer at an independent software development company, expressed his support of the decision, saying that Keccak is supposed to be tunable and there is no reason for different security levels within one primitive. He also added:

>

> *Yes, it's a bit of a shame for the competition that they demanded a certain security level for entrants, then went to publish a standard with a different one. But there's nothing that can be done to fix that now, except re-opening the competition. Demanding that they stick to their mistake doesn't improve things for anyone.*[43]

There was also some confusion that internal changes were made to Keccak. The Keccak team clarified this, stating

that NIST's proposal for SHA-3 is a subset of the Keccak family, for which one can generate test vectors using their reference code submitted to the contest, and that this proposal was the result of a series of discussions between them and the NIST hash team.[44] Also, Bruce Schneier corrected his earlier statement, saying:

> *I misspoke when I wrote that NIST made "internal changes" to the algorithm. That was sloppy of me. The Keccak permutation remains unchanged. What NIST proposed was reducing the hash*

> *function's capacity in the name of performance. One of Keccak's nice features is that it's highly tunable.*[42]

In response to the controversy, in November 2013 John Kelsey of NIST proposed to go back to the original $c = 2d$ proposal for all SHA-2 drop-in replacement instances.[45] These changes were confirmed in the April 2014 draft.[46] This proposal was implemented in the final release standard in August 2015.[4]

The reduced-capacity forms were published as SHAKE128 and SHAKE256, where the number indicates the security level and the number of bits of output is variable, but should be twice as large as the required collision resistance.

Examples of SHA-3 variants

The following hash values are from NIST.gov:[47]

```
SHA3-224("")
6b4e03423667dbb73b6e15454f0
eb1abd4597f9a1b078e3f5b5a6b
c7
SHA3-256("")
```

```
a7ffc6f8bf1ed76651c14756a06
1d662f580ff4de43b49fa82d80a
4b80f8434a
SHA3-384("")
0c63a75b845e4f7d01107d852e4
c2485c51a50aaaa94fc61995e71
bbee983a2ac3713831264adb47f
b6bd1e058d5f004
SHA3-512("")
a69f73cca23a9ac5c8b567dc185
a756e97c982164fe25859e0d1dc
c1475c80a615b2123af1f5f94c1
1e3e9402c3ac558f500199d95b6
d3e301758586281dcd26
SHAKE128("", 256)
7f9c2ba4e88f827d6160455076 0
5853ed73b8093f6efbc88eb1a6e
```

```
acfa66ef26
SHAKE256("", 512)
46b9dd2b0ba88d13233b3feb743
eeb243fcd52ea62b81b82b50c27
646ed5762fd75dc4ddd8c0f200c
b05019d67b592f6fc821c49479a
b48640292eacb3b7c4be
```

Changing a single bit causes each bit in the output to change with 50% probability, demonstrating an avalanche effect:

```
SHAKE128("The quick brown
fox jumps over the lazy
dog", 256)
f4202e3c5852f9182a0430fd814
```

```
4f0a74b95e7417ecae17db0f8cf
eed0e3e66e
SHAKE128("The quick brown
fox jumps over the lazy
dof", 256)
853f4538be0db9621a6cea659a0
6c1107b1f83f02b13d18297bd39
d7411cf10c
```

Comparison of SHA functions

In the table below, *internal state* means the number of bits that are carried over to the next block.

Comparison of SHA functions

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds	Operations	S
MD5 (as reference)		128	128 (4 × 32)	512	Unlimited[49]	64	And, Xor, Rot, Add (mod 2^{32}), Or	
SHA-0		160	160 (5 × 32)	512	$2^{64} - 1$	80	And, Xor, Rot, Add (mod 2^{32}), Or	
SHA-1								
SHA-2	*SHA-224* *SHA-256*	224 256	256 (8 × 32)	512	$2^{64} - 1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	
	SHA-384 *SHA-512*	384 512	512 (8 × 64)	1024	$2^{128} - 1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	
	SHA-512/224 *SHA-512/256*	224 256						
SHA-3	*SHA3-224* *SHA3-256* *SHA3-384* *SHA3-512*	224 256 384 512	1600 (5 × 5 × 64)	1152 1088 832 576	Unlimited[51]	24[52]	And, Xor, Rot, Not	
	SHAKE128 *SHAKE256*	*d* (arbitrary) *d* (arbitrary)		1344 1088				r r

References

1. *Keccak implementation overview*

Version 3.2 , section 3.1

2. *http://eprints.qut.edu.au/82454/2/__staffhome.qut.edu.au_staffgroupm$_meaton_Desktop_Draft%20paper_Pieprzyk.pdf*

3. *http://keccak.noekeon.org/Keccak-submission-3.pdf*

4. *https://www.nist.gov/itl/csd/201508_sha3.cfm*

5. *https://www.nist.gov/manuscript-publication-search.cfm?pub_id=919061*

6. *KeccakReferenceAndOptimized-3.2.zip mainReference.c "The Keccak sponge function, designed by Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van*

Assche. For more information, feedback or questions, please refer to our website: http://keccak.noekeon.org/Implementation by the designers, hereby denoted as "the implementer". To the extent possible under law, the implementer has waived all copyright and related or neighboring rights to the source code in this file. https://creativecommons.org/publicdomain/zero/1.0/ "

7. *"NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition" . NIST. 2012-10-02. Retrieved 2012-10-02.*

8. *Cruz, José R.C. (7 May 2013). "Keccak: The New SHA-3 Encryption Standard" . Dr. Dobbs.*

9. *Guido Bertoni; Joan Daemen; Michaël Peeters; Gilles Van Assche. "The Keccak sponge function family: Specifications summary" . Retrieved 2011-05-11.*

10. *NIST, Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition , sections 5.1.2.1 (mentioning "tree mode"), 6.2 ("other features", mentioning authenticated encryption), and 7 (saying "extras" may be standardized in the future)*

11. *Daemen, Joan, CAESAR submission: Ketje v1 (PDF)*

12. *Daemen, Joan, CAESAR submission: Keyak v1 (PDF)*

13. *Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. "Sponge Functions" . Ecrypt Hash Workshop 2007.*

14. ."Federal Register Vol 72. No 212" *(PDF).*

15. *"The road from Panama to Keccak via RadioGatun" (PDF).*

16. *Stevens, Marc; Bursztein, Elie; Karpman, Pierre; Albertini, Ange; Markov, Yarik. "The first collision for full SHA-1" (PDF). Retrieved 23 February 2017.*

17. *"NIST Computer Security Division – The SHA-3 Cryptographic Hash Algorithm Competition, November 2007 – October 2012" .*

18. *"Keccak parameter changes for round 2"* .

19. *"Simplifying Keccak's padding rule for round 3"* .

20. *"SHA-3 standardization"* . *NIST. Retrieved 2015-04-16.*

21. *National Institute of Standards and Technology (Aug 5, 2015). "Federal Information Processing Standards: Permutation-Based Hash and Extendable-Output Functions, etc"* . *Retrieved 5 Aug 2015.*

22. *"Announcing Approval of Federal Information Processing Standard (FIPS) 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,*

and Revision of the Applicability Clause of FIPS 180-4, Secure Hash Standard" . *2015-08-05.*

23. *Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction"* . *EuroCrypt 2008.*

24. *http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf*

25. *"about 41 cycles/byte [...] represents a 40% speedup compared to an implementation using only 32-bit instructions". By formula* $\frac{1}{x} \times 1.40 = \frac{1}{41}$ *we obtain* $x = 57.4$

26. http://cr.yp.to/hash/sha3opt-20120104.pdf

27. http://keccak.noekeon.org/is_sha3_slow.html

28. *Guo, Xu; Huang, Sinan; Nazhandali, Leyla; Schaumont, Patrick (Aug 2010), "Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations" (PDF), NIST 2nd SHA-3 Candidate Conference: 12, retrieved 2011-02-18 Keccak is second only to Luffa, which did not advance to the final round.*

29. http://keccak.noekeon.org/Sakura.pdf

30. *http://cr.yp.to/hash/quantumsha3-20101112.pdf*

31. *https://link.springer.com/chapter/10.1007%2FBFb0054319 citeseerx*

32. *http://cr.yp.to/hash/collisioncost-20090823.pdf*

33. *http://www.scottaaronson.com/papers/collision.pdf*

34. *https://eprint.iacr.org/2016/508.pdf*

35. *https://eprint.iacr.org/2017/282.pdf*

36. *John Kelsey. "SHA3, Where We've Been, Where We're Going" (PDF). RSA Conference 2013.*

37. *John Kelsey. "SHA3, Past, Present, and Future" . CHES 2013.*

38. *"NIST hash forum mailing list" .*

39. *"The Keccak SHA-3 submission" (PDF). 2011-01-14. Retrieved 2014-02-08.*

40. *"On 128-bit security" .*

41. *"A concrete proposal" . 2 October 2013.*

42. *"Schneier on Security: Will Keccak = SHA-3?" .*

43. *"LShift: Why I support the US Government making a cryptography standard weaker" .*

44. *"Yes, this is Keccak!" .*

45. *"Moving Forward with SHA-3" (PDF).*

46. *NIST Computer Security Division (CSD). "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions" (PDF). NIST.*

47. *"NIST.gov – Computer Security Division – Computer Security Resource Center"* .

48. *http://bench.cr.yp.to/results-hash.html#amd64-skylake*

49. *"The MD5 Message-Digest Algorithm"* . *Retrieved 2016-04-18. "In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used."*

50. *"Announcing the first SHA1 collision"* . *Retrieved 2017-02-23.*

51. *"The Sponge Functions Corner"* . *Retrieved 2016-01-27.*

52. *"The Keccak sponge function family"* . *Retrieved 2016-01-27.*

External links

- The Keccak web site