Security Evaluation Report on SHA-224, SHA-512/224, SHA-512/256, and the six SHA-3 Functions

Submitted for CRYPTREC

March 8 2015

Donghoon Chang
IIIT-Delhi, India
donghoon@iiitd.ac.in

Executive summary

This report contains an extensive evaluation of the security of SHA-224, SHA-512/224, SHA-512/256, and the six SHA-3 functions (SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, and SHAKE256) against *best known practical and theoretical* collision attacks, preimage attacks, second-preimage attacks, distinguishing attacks. Moreover, the report contains the security evaluation of the security of message authentication codes (MACs), authenticated encryption, and stream ciphers based on them against *best known practical and theoretical* forgery attacks and key recovery attacks.

We conclude that

- SHA-224, SHA-512/224, SHA-512/256, and the six SHA-3 functions provide the optimal security with a good security margin against the current best known collision/pseudo-collision attacks and preimage/pseudo-preimage attacks.
- *Except for SHA-224*, all other functions SHA-512/224, SHA-512/256, and the six SHA-3 functions provide the optimal security with a good security margin against the current best known second-preimage attacks.
  - There is a long-message second-preimage attack on SHA-224 with complexity less than 2^{224}.
- All the underlying functions of SHA-224, SHA-512/224, SHA-512/256, and all the six SHA-3 functions provide the optimal security with a good security margin against the current best known distinguishing attacks.
- *Except for SHA-224*, HMAC based on all other functions SHA-512/224, SHA-512/256, and the six SHA-3 functions provide the optimal security with a good security margin against the current best known forgery and key-recovery attacks.
  - There are best forgery and key-recovery attacks on HMAC-SHA-224 with complexity less than 2^{224}.
- MAC (Hash with a prefix key) and stream cipher and authenticated encryption (Keyak) based on the six SHA-3 functions provide the optimal security with a good security margin against the current best known forgery and key-recovery attacks.

Table of Contents

1 Algorithms

1.1 SHA-224

SHA-224 [47] is a truncated version of SHA-256 with a different initial value. In other words, SHA-224 is exactly same as SHA-256 except its initial value and its final truncation, where the 224-bit message digest is obtained by truncating the final hash value to its left-most 224 bits.

Given an input message string M of any length, a sequence of N 512-bit blocks $M^{(1)}||M^{(2)}||\cdots||M^{(N)}$ is defined by the following padding rule pad which appends the bit '1' to the end of the message, followed by k zero bits, where k is the smallest non-negative integer such that the bit-length of pad(M) is a multiple of 512, and then finally appends the 64-bit binary representation of the bit-length of M as follows:

$$\text{Let } M^{(1)}||M^{(2)}||\cdots||M^{(N)}=\text{pad}(M)=M||10^k||\text{bin}_{64}(|M|)$$

And by the big endian order, each $M^{(i)}$ is converted into sixteen 32-bit words as follows:

$$M^{(i)} \Longrightarrow W_0^{(i)}||W_1^{(i)}||W_2^{(i)}||\cdots||W_{15}^{(i)}$$

The initial value IV of SHA-224 consists of eight 32-bit words, which represent the second thirty-two bits of the fractional parts of the square roots of the 9th through 16th primes. The IV is defined as follows:

$$IV = H_0^{(0)}||H_1^{(0)}||H_2^{(0)}||H_3^{(0)}||H_4^{(0)}||H_5^{(0)}||H_6^{(0)}||H_7^{(0)} = \begin{cases} H_0^{(0)} &= 0xc1059ed8 \\ H_1^{(0)} &= 0x367cd507 \\ H_2^{(0)} &= 0x3070dd17 \\ H_3^{(0)} &= 0xf70e5939 \\ H_4^{(0)} &= 0xffc00b31 \\ H_5^{(0)} &= 0x68581511 \\ H_6^{(0)} &= 0x64f98fa7 \\ H_7^{(0)} &= 0xbefa4fa4 \end{cases}$$

SHA-224 uses a sequence of sixty-four constant 32-bit words, which represent the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers. Each constant is used only in one step. Therefore the compression function of SHA-224 consists of sixty-four steps.

$$\begin{array}{rllllll} K_t^{\{256\}} = & 0x428a2f98 & 0x71374491 & 0xb5c0fbcf & 0xe9b5dba5 & 0x3956c25b & 0x59f111f1 \\ & 0x923f82a4 & 0xab1c5ed5 & 0xd807aa98 & 0x12835b01 & 0x243185be & 0x550c7dc3 \\ & 0x72be5d74 & 0x80deb1fe & 0x9bdc06a7 & 0xc19bf174 & 0xe49b69c1 & 0xefbe4786 \\ & 0x0fc19dc6 & 0x240ca1cc & 0x2de92c6f & 0x4a7484aa & 0x5cb0a9dc & 0x76f988da \\ & 0x983e5152 & 0xa831c66d & 0xb00327c8 & 0xbf597fc7 & 0xc6e00bf3 & 0xd5a79147 \\ & 0x06ca6351 & 0x14292967 & 0x27b70a85 & 0x2e1b2138 & 0x4d2c6dfc & 0x53380d13 \\ & 0x650a7354 & 0x766a0abb & 0x81c2c92e & 0x92722c85 & 0xa2bfe8a1 & 0xa81a664b \\ & 0xc24b8b70 & 0xc76c51a3 & 0xd192e819 & 0xd6990624 & 0xf40e3585 & 0x106aa070 \\ & 0x19a4c116 & 0x1e376c08 & 0x2748774c & 0x34b0bcb5 & 0x391c0cb3 & 0x4ed8aa4a \\ & 0x5b9cca4f & 0x682e6ff3 & 0x748f82ee & 0x78a5636f & 0x84c87814 & 0x8cc70208 \\ & 0x90befffa & 0xa4506ceb & 0xbef9a3f7 & 0xc67178f2 & \text{for} & 0 \le t \le 63 \end{array}$$

Let $W_t^{(i)}$ be the word to be used at t-th step. Then, $W_t^{(i)}$ is defined as follows:

$$W_t^{(i)} = \sigma_1(W_{t-2}^{(i)}) + W_{t-7}^{(i)} + \sigma_0(W_{t-15}^{(i)}) + W_{t-16}^{(i)} \qquad \text{for} \quad 16 \le t \le 63$$

SHA-224 uses six logical functions, where each function operates on 32-bit words, as follows:

$$\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \vee (\neg x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \\
\Sigma_0^{\{256\}}(x) &= x^{>>>2} \oplus x^{>>>13} \oplus x^{>>>22} \\
\Sigma_1^{\{256\}}(x) &= x^{>>>6} \oplus x^{>>>11} \oplus x^{>>>25} \\
\sigma_0^{\{256\}}(x) &= x^{>>>7} \oplus x^{>>>18} \oplus x^{>>3} \\
\sigma_1^{\{256\}}(x) &= x^{>>>17} \oplus x^{>>>19} \oplus x^{>>10}
\end{aligned}$$

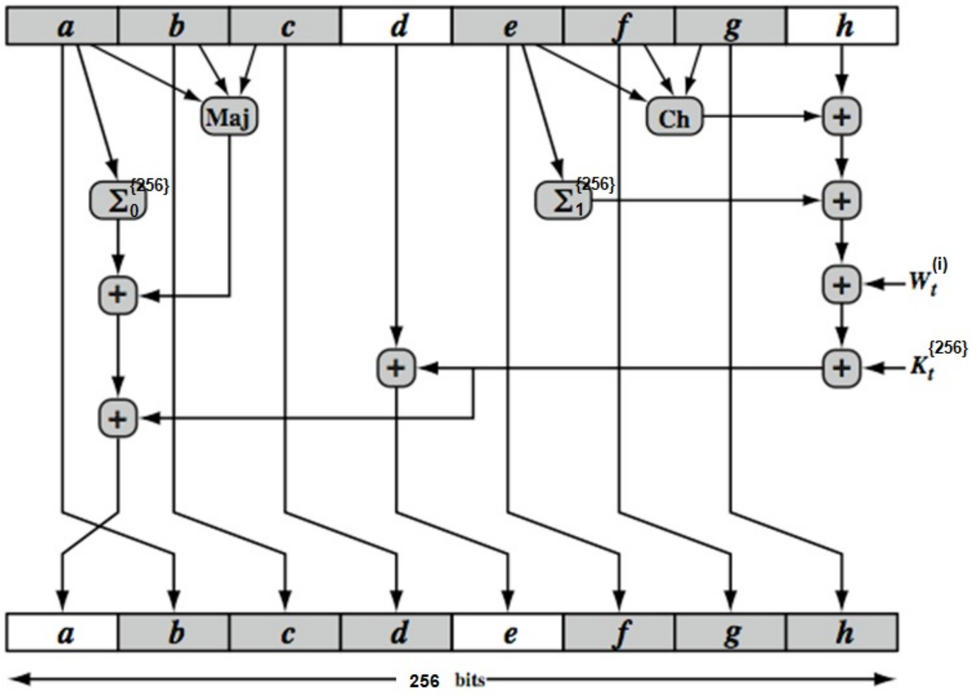


Fig. 1. The Step Function of SHA-224 and SHA-256 [52].

The whole process of producing a 224-bit digest, $H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}$, is as follows:

For i=1 to N:

1. Intialize the eight working variables, a, b, c, d, e, f, g, and h, with the (i-1)$^{\text{st}}$ hash value:

$$\begin{aligned}
a &= H_0^{(i-1)} \\
b &= H_1^{(i-1)} \\
c &= H_2^{(i-1)} \\
d &= H_3^{(i-1)}
\end{aligned}$$

$$\begin{aligned} e &= H_4^{(i-1)} \\ f &= H_5^{(i-1)} \\ g &= H_6^{(i-1)} \\ h &= H_7^{(i-1)} \end{aligned}$$

2. For $t = 0$ to 63:

$$\begin{aligned} T_1 &= h + \textstyle\sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t^{(i)} \\ T_2 &= h + \textstyle\sum_0^{\{256\}}(a) + Maj(a,b,c) \\ h &= g \\ g &= f \\ f &= e \\ e &= d + T_1 \\ d &= c \\ c &= b \\ b &= a \\ a &= T_1 + T_2 \end{aligned}$$

3. Compute the i^{th} intermediate hash value $H^{(i)}$:

$$\begin{aligned} H_0^{(i)} &= a + H_0^{(i-1)} \\ H_1^{(i)} &= b + H_1^{(i-1)} \\ H_2^{(i)} &= c + H_2^{(i-1)} \\ H_3^{(i)} &= d + H_3^{(i-1)} \\ H_4^{(i)} &= e + H_4^{(i-1)} \\ H_5^{(i)} &= f + H_5^{(i-1)} \\ H_6^{(i)} &= g + H_6^{(i-1)} \\ H_7^{(i)} &= h + H_7^{(i-1)} \end{aligned}$$

1.2 SHA-512/224 and SHA-512/256

SHA-512/224 and SHA-512/256 [47] are truncated versions of SHA-512 with different initial values. In other words, SHA-512/224 and SHA-512/256 are exactly same as SHA-512 except their initial values and their final truncation, where the 224-bit message digest for SHA-512/224 (the 256-bit message digest for SHA-512/256) is obtained by truncating the final hash value to its left-most 224 bits (its left-most 256 bits for SHA-512/256).

Given a input message string M of any length, a sequence of N 1024-bit blocks $M^{(1)}||M^{(2)}||\cdots||M^{(N)}$ is defined by the following padding rule pad which appends the bit '1' to the end of the message, followed by k zero bits, where k is the smallest non-negative integer such that the bit-length of pad(M) is a multiple of 1024, and then finally appends the 128-bit binary representation of the bit-length of M as follows:

$$\text{Let } M^{(1)}||M^{(2)}||\cdots||M^{(N)}=\text{pad}(M)=M||10^k||\text{bin}_{128}(|M|)$$

And by the big endian order, each $M^{(i)}$ is converted into sixteen 64-bit words as follows:

$$M^{(i)} \Longrightarrow W_0^{(i)}||W_1^{(i)}||W_2^{(i)}||\cdots||W_{15}^{(i)}$$

Unlike SHA-224, SHA-256, SHA-384, and SHA-512, the initial values for SHA-512/224 and SHA-512/256 are not defined by the fractional parts of the cube roots of prime numbers, but defined by calling SHA-512 hash function with a tweaked $IV' = (IV \oplus 0xa5a5a5a5a5........a5)$, where

IV is the original initial value of SHA-512, as follows:

In case of SHA-512/224,

$$IV_{512/224} = H_0^{(0)}||H_1^{(0)}||H_2^{(0)}||H_3^{(0)}||H_4^{(0)}||H_5^{(0)}||H_6^{(0)}||H_7^{(0)} = \text{SHA-512(“SHA-512/224”) with } IV'$$

In case of SHA-512/256,

$$IV_{512/256} = H_0^{(0)}||H_1^{(0)}||H_2^{(0)}||H_3^{(0)}||H_4^{(0)}||H_5^{(0)}||H_6^{(0)}||H_7^{(0)} = \text{SHA-512(“SHA-512/256”) with } IV'$$

Then, the initial value of SHA-512/224, which consists of eight 64-bit words, is defined as follows:

$$IV_{512/224} = H_0^{(0)}||H_1^{(0)}||H_2^{(0)}||H_3^{(0)}||H_4^{(0)}||H_5^{(0)}||H_6^{(0)}||H_7^{(0)} = \begin{cases} H_0^{(0)} & = 0x8c3d37c819544da2 \\ H_1^{(0)} & = 0x73e1996689dcd4d6 \\ H_2^{(0)} & = 0x1dfab7ae32ff9c82 \\ H_3^{(0)} & = 0x679dd514582f9fcf \\ H_4^{(0)} & = 0x0f6d2b697bd44da8 \\ H_5^{(0)} & = 0x77e36f7304c48942 \\ H_6^{(0)} & = 0x3f9d85a86a1d36c8 \\ H_7^{(0)} & = 0x1112e6ad91d692a1 \end{cases}$$

And, the initial value of SHA-512/256, which consists of eight 64-bit words, is defined as follows:

$$IV_{512/256} = H_0^{(0)}||H_1^{(0)}||H_2^{(0)}||H_3^{(0)}||H_4^{(0)}||H_5^{(0)}||H_6^{(0)}||H_7^{(0)} = \begin{cases} H_0^{(0)} & = 0x22312194fc2bf72c \\ H_1^{(0)} & = 0x9f555fa3c84c64c2 \\ H_2^{(0)} & = 0x2393b86b6f53b151 \\ H_3^{(0)} & = 0x963877195940eabd \\ H_4^{(0)} & = 0x96283ee2a88effe3 \\ H_5^{(0)} & = 0xbe5e1e2553863992 \\ H_6^{(0)} & = 0x2b0199fc2c85b8aa \\ H_7^{(0)} & = 0x0eb72ddc81c52ca2 \end{cases}$$

SHA-512/224 and SHA-512/256 uses the same sequence of eighty constant 64-bit words of SHA-512, which represents the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers. Each constant is used only in one step. Therefore the compression function of SHA-512/224 and SHA-512/256 consist of sixty-four steps, respectively.

$$K_t^{\{512\}} = \begin{array}{llll}
0x428a2f98d728ae22 & 0x7137449123ef65cd & 0xb5c0fbcfec4d3b2f & 0xe9b5dba58189dbbc \\
0x3956c25bf348b538 & 0x59f111f1b605d019 & 0x923f82a4af194f9b & 0xab1c5ed5da6d8118 \\
0xd807aa98a3030242 & 0x12835b0145706fbe & 0x243185be4ee4b28c & 0x550c7dc3d5ffb4e2 \\
0x72be5d74f27b896f & 0x80deb1fe3b1696b1 & 0x9bdc06a725c71235 & 0xc19bf174cf692694 \\
0xe49b69c19ef14ad2 & 0xefbe4786384f25e3 & 0x0fc19dc68b8cd5b5 & 0x240ca1cc77ac9c65 \\
0x2de92c6f592b0275 & 0x4a7484aa6ea6e483 & 0x5cb0a9dcbd41fbd4 & 0x76f988da831153b5 \\
0x983e5152ee66dfab & 0xa831c66d2db43210 & 0xb00327c898fb213f & 0xbf597fc7beef0ee4 \\
0xc6e00bf33da88fc2 & 0xd5a79147930aa725 & 0x06ca6351e003826f & 0x142929670a0e6e70 \\
0x27b70a8546d22ffc & 0x2e1b21385c26c926 & 0x4d2c6dfc5ac42aed & 0x53380d139d95b3df \\
0x650a73548baf63de & 0x766a0abb3c77b2a8 & 0x81c2c92e47edaee6 & 0x92722c851482353b \\
0xa2bfe8a14cf10364 & 0xa81a664bbc423001 & 0xc24b8b70d0f89791 & 0xc76c51a30654be30 \\
0xd192e819d6ef5218 & 0xd69906245565a910 & 0xf40e35855771202a & 0x106aa07032bbd1b8 \\
0x19a4c116b8d2d0c8 & 0x1e376c085141ab53 & 0x2748774cdf8eeb99 & 0x34b0bcb5e19b48a8 \\
0x391c0cb3c5c95a63 & 0x4ed8aa4ae3418acb & 0x5b9cca4f7763e373 & 0x682e6ff3d6b2b8a3 \\
0x748f82ee5defb2fc & 0x78a5636f43172f60 & 0x84c87814a1f0ab72 & 0x8cc702081a6439ec \\
0x90befffa23631e28 & 0xa4506cebde82bde9 & 0xbef9a3f7b2c67915 & 0xc67178f2e372532b \\
0xca273eceea26619c & 0xd186b8c721c0c207 & 0xeada7dd6cde0eb1e & 0xf57d4f7fee6ed178 \\
0x06f067aa72176fba & 0x0a637dc5a2c898a6 & 0x113f9804bef90dae & 0x1b710b35131c471b \\
0x28db77f523047d84 & 0x32caab7b40c72493 & 0x3c9ebe0a15c9bebc & 0x431d67c49c100d4c \\
0x4cc5d4becb3e42b6 & 0x597f299cfc657e2a & 0x5fcb6fab3ad6faec & 0x6c44198c4a475817 \\
\text{for } 0 \leq t \leq 79 & & &
\end{array}$$

Let $W_t^{(i)}$ be the word to be used at t-th step. Then, $W_t^{(i)}$ is defined in the same way of SHA-512 as follows:

$$W_t^{(i)} = \sigma_1(W_{t-2}^{(i)}) + W_{t-7}^{(i)} + \sigma_0(W_{t-15}^{(i)}) + W_{t-16}^{(i)} \qquad \text{for} \quad 16 \leq t \leq 79$$

SHA-512/224 and SHA-512/256 use the same six logical functions with SHA-512, where each function operates on 64-bit words, as follows:

$$\begin{array}{ll}
Ch(x,y,z) & = (x \wedge y) \vee (\neg x \wedge z) \\
Maj(x,y,z) & = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \\
\Sigma_0^{\{512\}}(x) & = x^{>>>28} \oplus x^{>>>34} \oplus x^{>>>39} \\
\Sigma_1^{\{512\}}(x) & = x^{>>>14} \oplus x^{>>>18} \oplus x^{>>>41} \\
\sigma_0^{\{512\}}(x) & = x^{>>>1} \oplus x^{>>>8} \oplus x^{>>7} \\
\sigma_1^{\{512\}}(x) & = x^{>>>19} \oplus x^{>>>61} \oplus x^{>>6}
\end{array}$$

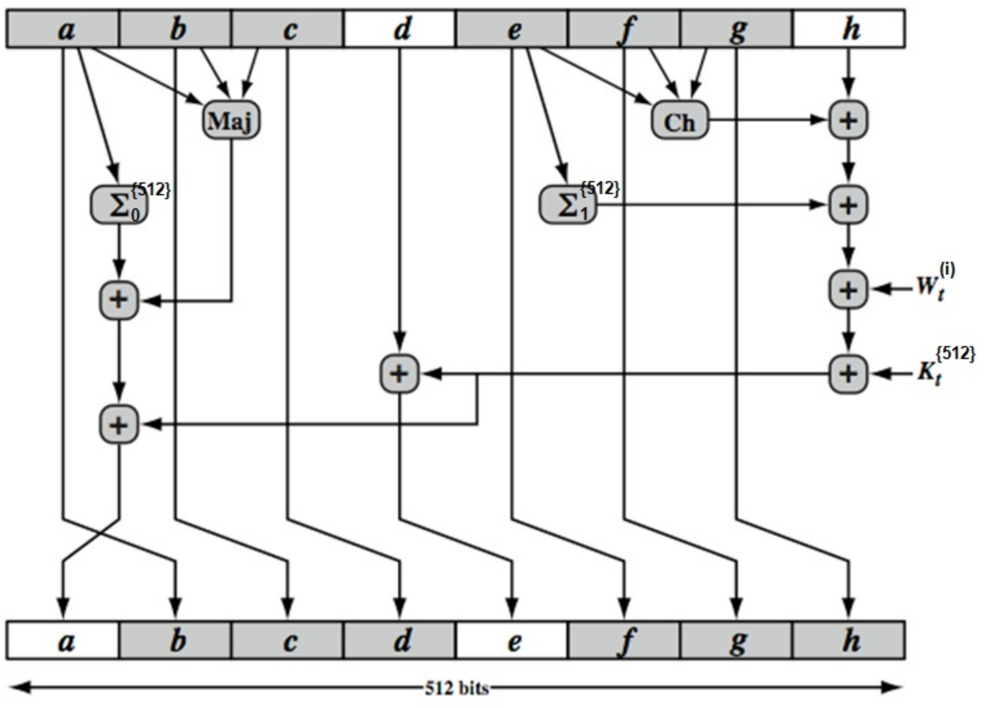


Fig. 2. The Step Function of SHA-512/224 and SHA-512/256 [52].

The whole process of producing a 224-bit digest for SHA-512/224 (a 256-bit digest for SHA-512/256), which is the left-most 224 bits (the left-most 256 bits for SHA-512/256) of $H_0^{(N)}||\cdots||H_7^{(N)}$, as follows:

For i=1 to N:

1. Intialize the eight working variables, a, b, c, d, e, f, g, and h, with the (i-1)st hash value:

$$\begin{aligned} a &= H_0^{(i-1)} \\ b &= H_1^{(i-1)} \\ c &= H_2^{(i-1)} \\ d &= H_3^{(i-1)} \\ e &= H_4^{(i-1)} \\ f &= H_5^{(i-1)} \\ g &= H_6^{(i-1)} \\ h &= H_7^{(i-1)} \end{aligned}$$

2. For $t = 0$ to 79:

$$\begin{aligned}
T_1 &= h + \textstyle\sum_1^{\{512\}}(e) + Ch(e,f,g) + K_t^{\{512\}} + W_t^{(i)} \\
T_2 &= h + \textstyle\sum_0^{\{512\}}(a) + Maj(a,b,c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}$$

3. Compute the i^{th} intermediate hash value $H^{(i)}$:

$$\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)} \\
H_1^{(i)} &= b + H_1^{(i-1)} \\
H_2^{(i)} &= c + H_2^{(i-1)} \\
H_3^{(i)} &= d + H_3^{(i-1)} \\
H_4^{(i)} &= e + H_4^{(i-1)} \\
H_5^{(i)} &= f + H_5^{(i-1)} \\
H_6^{(i)} &= g + H_6^{(i-1)} \\
H_7^{(i)} &= h + H_7^{(i-1)}
\end{aligned}$$

1.3 The Six SHA-3 Functions

The Draft FIPS FUB 202 [48] specifies the SHA-3 family which consists of four cryptographic hash functions and two expandable output functions (XOFs). According to digest bit-lengths, the four cryptographic hash functions are named SHA3-224, SHA3-256, SHA3-384, and SHA-512, where each SHA3-n produces n-bit digests. The two SHA-3 XOFs are named SHAKE128 and SHAKE256, where these numerical suffixes '128' and '256' indicate the security strength. Unlike the four SHA-3 cryptographic hash functions, the two SHA-3 XOFs can produce output of any desired length.

All the six SHA-3 functions are based on the same underlying permutation, called Keccak-p[1600,24].

The Permutation Keccak-p[1600,24]. Let $x, y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_{64}$. Let $a[x][y][z]$ represent each bit of 1600-bit by the values of x, y, z. The permutation Keccak-p[1600,24] is an iterated permutation on 1600-bit, consisting of a sequence of 24 rounds $\mathbf{R}$. A round $\mathbf{R}$=$\iota \circ \chi \circ \pi \circ \rho \circ \theta$ consists of five steps:

$$\begin{aligned}
\theta:\ & a[x][y][z] \leftarrow a[x][y][z] \oplus \textstyle\bigoplus_{y'=0}^{4} a[x-1][y'][z] \oplus \bigoplus_{y'=0}^{4} a[x+1][y'][z-1] \\
\rho:\ & a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2], \\
& \qquad \text{with } t \text{ satisfying } 0 \le t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ in } \mathbf{GF}(5)^{2\times 2}, \\
& \qquad \text{or } t = -1 \text{ if } x = y = 0, \\
\pi:\ & a[x][y] \leftarrow a[x'][y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}, \\
\chi:\ & a[x] \leftarrow a[x] \oplus (a[x+1] \oplus 1) a[x+2], \\
\iota:\ & a \leftarrow a \oplus \mathbf{RC}[i_r],
\end{aligned}$$

where the round constants $\mathbf{RC}[i_r][0][0][2^j-1] = \text{rc}[j+7i_r]$ for all $0 \leq j \leq \ell$ and $0 \leq i_r \leq 23$, and all other values of $\mathbf{RC}[i_r][x][y][z]$ are zero. The values $\text{rc}[t] \in \mathbf{GF}(2)$ are defined as $\text{rc}[t] = (x^t \bmod x^8 + x^6 + x^5 + x^4 + 1) \bmod x$ in $\mathbf{GF}(2)[x]$.

Comparison with Keccak-*f*. The Keccak-f family of permutations, originally defined in [10], is the specialization of the Keccak-p family to the case that $n_r = 12 + 2l$:

$$\text{Keccak-}f[b] = \text{Keccak-}p[b,\, 12+2l].$$

Consequently, the Keccak-p[1600, 24] permutation, which underlies the six SHA-3 functions, is equivalent to Keccak-f[1600].

The sponge construction, denoted by sponge$[f,\text{pad},r](M,d)$ is a framework for domain extensions of the six SHA-3 functions, where f is a function mapping $(r+c)$-bit stings to $(r+c)$-bit stings, pad is a padding rule by appending an appropriate string to the given message M, and d indicates the digest size. The sponge construction is illustrated in Fig. 3.

Then, Keccak$[c]$ is specified as follows:

Keccak$[c](M,d)$=sponge[Keccak-p[1600,24],pad10*1,1600-c](M,d),

where pad10*1(M)=$M||10^t1$ with the minimum-non-negative integer t making the size of pad10*1(M) a multiple of (1600-c).

The four SHA-3 hash functions and the two SHA-3 XOFs are specified as follows:

The Four SHA-3 Hash Functions:

$$\begin{aligned}
\text{SHA3-224}(M) &= \text{Keccak}[448](M||01, 224);\\
\text{SHA3-256}(M) &= \text{Keccak}[512](M||01, 256);\\
\text{SHA3-384}(M) &= \text{Keccak}[768](M||01, 384);\\
\text{SHA3-512}(M) &= \text{Keccak}[1024](M||01, 512);
\end{aligned}$$

The Two SHA-3 XOFs:

$$\begin{aligned}
\text{SHAKE128}(M,d) &= \text{Keccak}[256](M||1111, d);\\
\text{SHAKE256}(M,d) &= \text{Keccak}[512](M||1111, d);
\end{aligned}$$

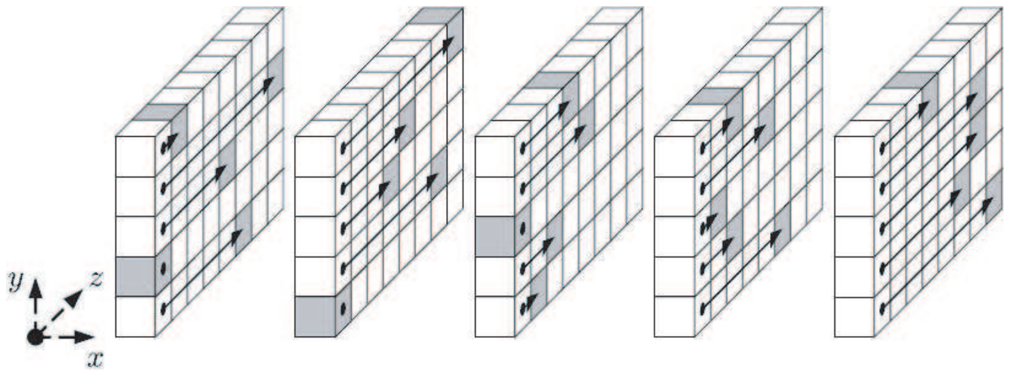


Fig. 6. The Function ρ of KECCAK-p [6]: Note that x=y=0 is depicted at the center of the slice.

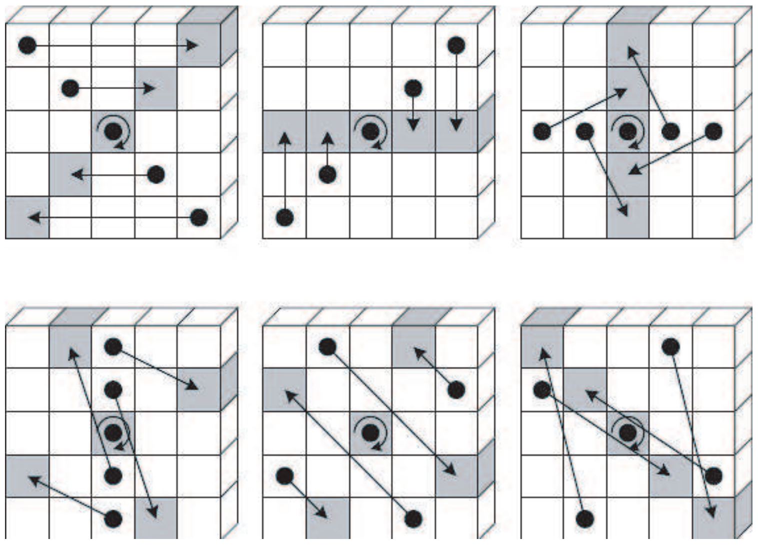


Fig. 7. The Function π of KECCAK-p [6]: Note that x=y=0 is depicted at the center of the slice.

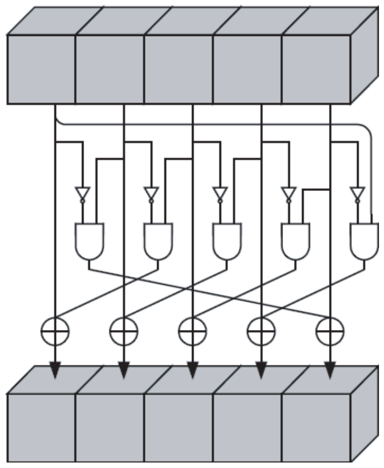


Fig. 8. The Function χ of KECCAK-p [6].

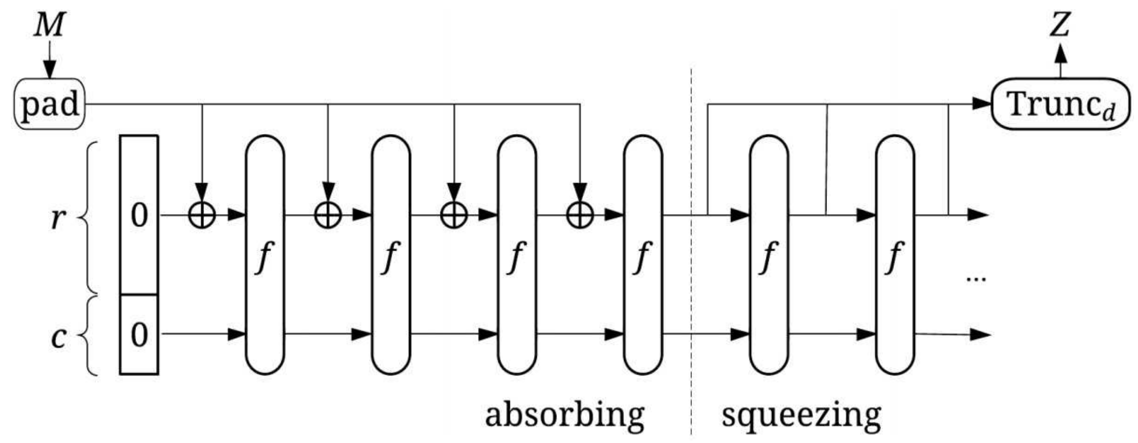
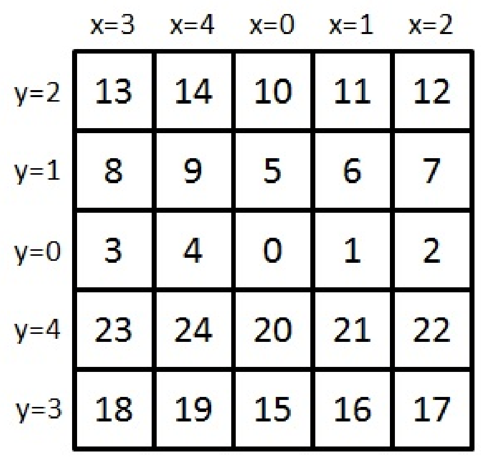


Fig. 3. The sponge construction: $Z = \text{SPONGE}[f,\text{pad},d]$ [8].



	x=3	x=4	x=0	x=1	x=2
y=2	13	14	10	11	12
y=1	8	9	5	6	7
y=0	3	4	0	1	2
y=4	23	24	20	21	22
y=3	18	19	15	16	17

Fig. 4. Slice Index Numbering of KECCAK-p state [6]: Note that x=y=0 is depicted at the center of the slice.

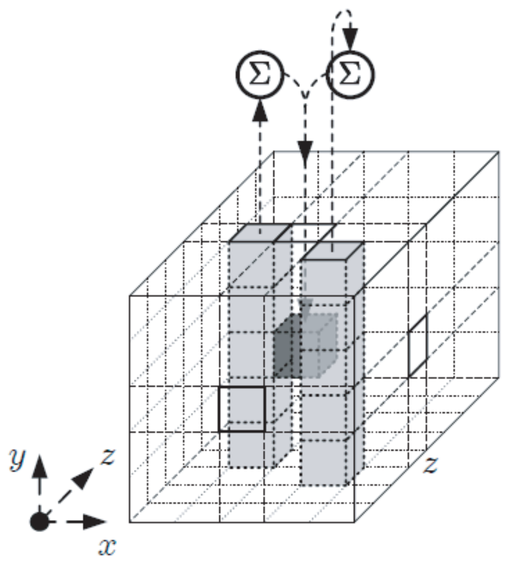


Fig. 5. The Function θ of KECCAK-p [6].

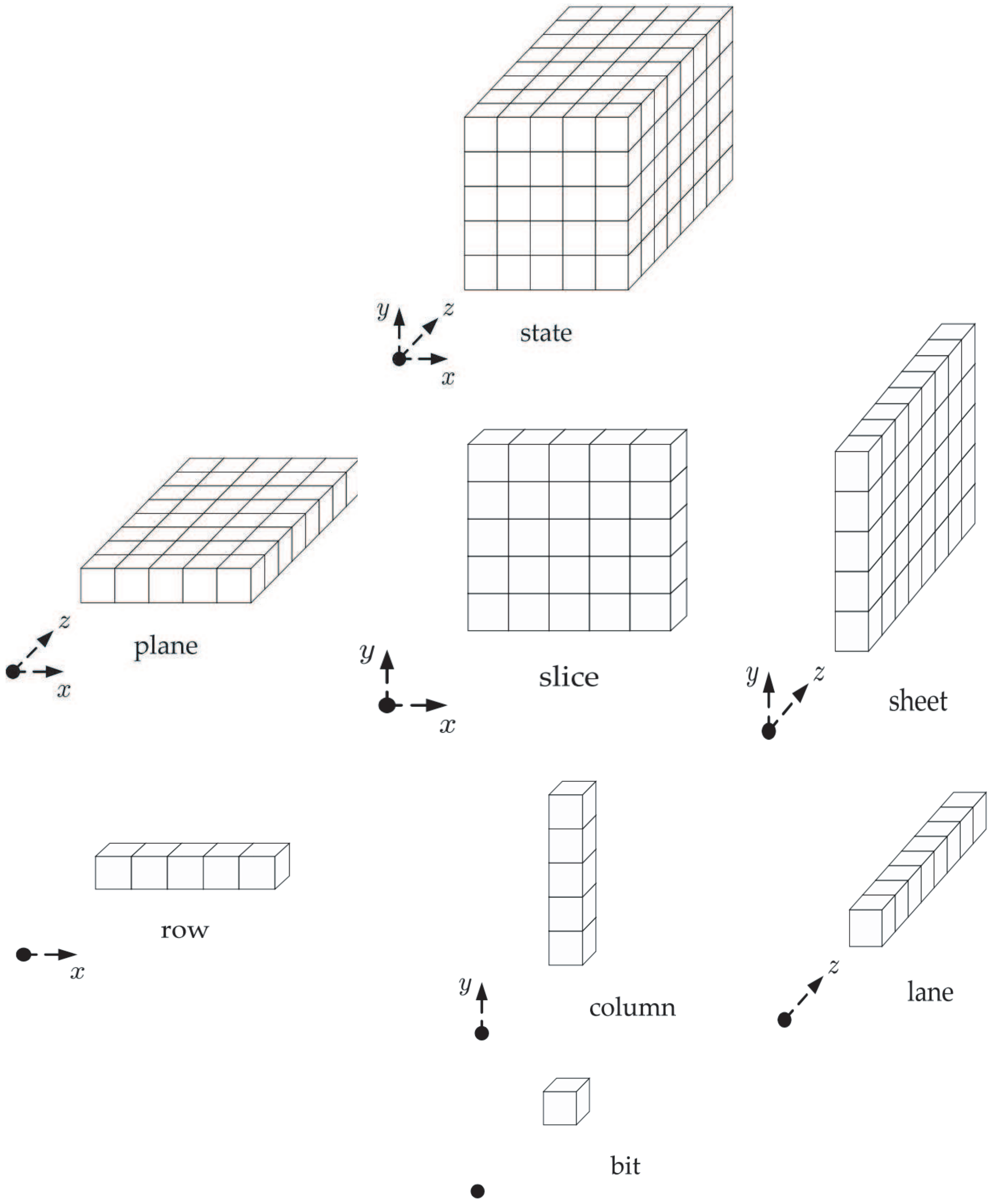


Fig. 9. Naming conventions for parts of Keccak-p state [6].

2 Security Overview

This section describes the security overview of SHA-224, SHA-512/224, SHA-512/256, and SHA-3 functions and their application to MAC, stream cipher, and authenticated encryption. All the detailed results in each table will be explained in Sect. 3, Sect. 4, Sect. 5, and Sect. 6. In other words, all the tables in this section shows the summary of security analyses explained in this report. For that purpose, we has examained and, in this report, has contained all of current best practical and theoretical collision-type, preimage-type, second-preimage, distinguishing attacks on each of SHA-224, SHA-512/224, SHA-512/256, and SHA-3 functions. In order to explain each attack technique, we will use some examples along with a simple additional explanation.

Especially, in Sect. 5, we will evaluate the domain separation between SHA-224 and SHA-256, and the domain separation among SHA-512, SHA-512/224, SHA-512/256, and the domain separation among the six SHA-3 functions. For the domain separation between SHA-224 and SHA-256, they use different initial values. For the domain separation among SHA-512, SHA-512/224, SHA-512/256, SHA-512/224 and SHA-512/256 use different initial values by calling SHA-512 with a tweaked initial value. For the domain separation among the six SHA-3 functions, they use two layers of padding rule; the multi-rate padding 10*1 as the outer padding, and a partition-padding approach as the inner padding, which is explained in detail in Sect. 5.

In Table 1, except for SHA-224, all other functions provides optimal security against the existing collision, preimage, and second-preimage attacks, because the sizes of their internal states are at least double of their hash output sizes. In case of SHA-224, we can only guarantee min(224,256-$L(M)$)-bit security against Kelsey-Schneier's long-message second-preimage attack [37], where $L(M)$ is defined as $\lceil \log_2(\mathrm{len}(M)/512) \rceil$, which is explained in Sect. 4.1.

Algorithm	Output Size	Security Strengths in Bits		
		Collision	Preimage	2nd Preimage
SHA-224	224	112	224	min(224,256-$L(M)$)
SHA-512/224	224	112	224	224
SHA-512/256	256	128	256	256
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE128	d	min(d/2,128)	$\geq$min(d,128)	min(d,128)
SHAKE256	d	min(d/2,256)	$\geq$min(d,256)	min(d,256)

Table 1. Security Strengths of SHA-2 and SHA-3 functions [48]: $L(M)$ is defined as $\lceil \log_2(\mathrm{len}(M)/512) \rceil$.

In Table 2, we can see the best known practical collision-type attacks against SHA-2, and SHA-3 functions. Here, practical attacks means that examples for collisions or semi-free-start collisions are actually given, which are explained in detail in Sect. 3.

Algorithm	Attack Type	Target	Rounds	Percent Broken	Practical?	Reference
SHA-224	Semi-free-start Collision	Hash Function	38/64	59%	Example Given	[43]
SHA-512/224	Semi-free-start Collision	Hash Function	38/80	47.5%	Example Given	[32]
SHA-512/256	Semi-free-start Collision	Hash Function	38/80	47.5%	Example Given	[32]
SHA3-224	Collision	Hash Function	4/24	16.7%	Example Given	[22]
SHA3-256	Collision	Hash Function	4/24	16.7%	Example Given	[22]
SHA3-384	Collision	Hash Function	3/24	12.5%	Example Given	[23]
SHA3-512	Collision	Hash Function	3/24	12.5%	Example Given	[23]
SHAKE128	Collision	Hash Function	4/24	16.7%	Example Given	[22]
SHAKE256	Collision	Hash Function	4/24	16.7%	Example Given	[22]

Table 2. Best Known Practical Collision-type Attacks against SHA-2, and SHA-3 functions

In Table 3, we can see the best known theoretical collision-type attacks against SHA-2, and SHA-3 functions, which are explained in detail in Sect. 3.

Algorithm	Attack Type	Target	Rounds	Percent Broken	CF Call	Reference
SHA-224	-	-	-	-	-	-
SHA-512/224	-	-	-	-	-	-
SHA-512/256	-	-	-	-	-	-
SHA3-224	-	-	-	-	-	-
SHA3-256	Collision	Hash Function	5/24	21%	2^{115}	[23]
SHA3-384	Collision	Hash Function	4/24	16.7%	2^{147}	[23]
SHA3-512	-	-	-	-	-	-
SHAKE128	Collision	Hash Function	5/24	21%	2^{115}	[23]
SHAKE256	Collision	Hash Function	5/24	21%	2^{115}	[23]

Table 3. Best Known Theoretical Collision-type Attacks against SHA-2, and SHA-3 functions

In Table 4, we can see the best known practical preimage-type and second-preimage attacks against SHA-2, and SHA-3 functions, which are explained in detail in Sect. 3.

Algorithm	Attack Type	Target	Rounds	Percent Broken	Practical?	Reference
SHA-224	-	-	-	-	-	-
SHA-512/224	-	-	-	-	-	-
SHA-512/256	-	-	-	-	-	-
SHA3-224	Preimage Second-Preimage	hash function	≤ 3	12.5%	2^{34}	[45]
SHA3-256	Preimage Second-Preimage	hash function	≤ 3	12.5%	2^{34}	[45]
SHA3-384	Preimage Second-Preimage	hash function	≤ 3	12.5%	2^{34}	[45]
SHA3-512	Preimage Second-Preimage	hash function	≤ 3	12.5%	2^{34}	[45]
SHAKE128	Preimage Second-Preimage	hash function	≤ 3	12.5%	2^{34}	[45]
SHAKE256	Preimage Second-Preimage	hash function	≤ 3	12.5%	2^{34}	[45]

Table 4. Best Known Practical Preimage-type and Second-preimage Attacks against SHA-2, and SHA-3 functions

In Table 5, we can see the best known theoretical preimage-type and second-preimage attacks against SHA-2, and SHA-3 functions, which are explained in detail in Sect. 3.

Algorithm	Attack Type	Target	Rounds	Percent Broken	CF Call	Reference
SHA-224	Pseudo Preimage	Hash Function	52/64	81.25%	2^{255}	[38]
SHA-512/224	Pseudo Preimage	Hash Function	57/80	71.25%	2^{511}	[38]
SHA-512/256	Pseudo Preimage	Hash Function	57/80	71.25%	2^{511}	[38]
SHA3-224	Preimage Second-Preimage	Hash Function	7/24	29.2%	Time: $2^{218.11}$ Memory: $2^{180.12}$	[19]
SHA3-256	Preimage Second-Preimage	Hash Function	8/24	33.3%	Time: $2^{255.64}$ Memory: $2^{254.03}$	[19]
SHA3-384	Preimage Second-Preimage	Hash Function	8/24	33.3%	Time: $2^{378.74}$ Memory: $2^{324.06}$	[19]
SHA3-512	Preimage Second-Preimage	Hash Function	9/24	37.5%	Time: $2^{511.70}$ Memory: $2^{510.2}$	[19]
SHAKE128	Preimage Second-Preimage	Hash Function	6/24	25%	Expected Time: near 2^{128} Memory: ?	[19]
SHAKE256	Preimage Second-Preimage	Hash Function	8/24	33.3%	Time: $2^{378.74}$ Memory: $2^{324.06}$	[19]

Table 5. Best Known Theoretical Preimage-type and Second-Preimage Attacks against SHA-2, and SHA-3 functions

In Table 6, we can see the best known practical distinguishing attacks and differential properties for SHA-2 and SHA-3 Functions, which are explained in detail in Sect. 3.

Algorithm	Target	Rounds	Fraction of Target Analyzed	Underlying Function Call	Reference
SHA-224	Compression Function	47/64	73.4%	Example Given	[14]
SHA-512/224	Compression Function	48/80	60%	Example Given	[53]
SHA-512/256	Compression Function	48/80	60%	Example Given	[53]
SHA3-224 SHA3-256 SHA3-384 SHA3-512 SHAKE128 SHAKE256	Permutation	9/24 10/24	37.5% 42%	$2^{29.83}$ $2^{59.67}$	[2]

Table 6. Best Known Practical Distinguishing Attacks and Differential Properties for SHA-2 and SHA-3 Functions

In Table 7, we can see the best known theoretical distinguishing attacks and differential properties for SHA-2 and SHA-3 Functions, which are explained in detail in Sect. 3.

Algorithm	Target	Rounds	Fraction of Target Analyzed	CF Call	Reference
SHA-224	-	-	-	-	-
SHA-512/224	-	-	-	-	-
SHA-512/256	-	-	-	-	-
SHA3-224	Permutation	12	50%	2^{128}	[2]
SHA3-256	Permutation	13	54.17%	2^{243}	[2]
SHA3-384	Permutation	14	58.33%	2^{256}	[2]
SHA3-512	Permutation	14	58.33%	2^{256}	[2]
SHAKE128	Permutation	11	45.83%	2^{81}	[2]
SHAKE256	Permutation	13	54.17%	2^{243}	[2]

Table 7. Best Known Theoretical Distinguishing Attacks and Differential Properties for SHA-2 and SHA-3 Functions

According to existing analyses, we can summarize the best attack complexities on HMAC based SHA-224, SHA-512/224, and SHA-512/256 as shown in Table 8. As we can see, when the key size

(k) is same as the hash output size, except for SHA-224, the best known attacks on SHA-512/224 and SHA-512/256 are the exhaustive search over the possible key space. Based on existing attacks, SHA-512/224 provides better security than SHA-224.

Algorithm HMAC-SHA-v	ℓ	s	Existential Forgery $\min(2^k, O(2^{\ell/2})$ [50])	Universal Forgery $\min(2^k, O(\ell \cdot 2^{\ell-s})$ [35])	Internal State Recovery $\min(2^k, O(2^{\ell-s})$ [41, 24])	Key Recovery $\min(2^k, O(2^{3\ell/4})$ [35])
v=224	256	55	$\min(2^k, O(2^{128})$ [50])	$\min(2^k, O(2^{200})$ [35])	$\min(2^k, O(2^{201})$ [41, 24])	$\min(2^k, O(2^{192})$ [35])
v=512/224	512	118	$\min(2^k, O(2^{256})$ [50])	$\min(2^k, O(2^{393})$ [35])	$\min(2^k, O(2^{394})$ [41, 24])	$\min(2^k, O(2^{384})$ [35])
v=512/256	512	118	$\min(2^k, O(2^{256})$ [50])	$\min(2^k, O(2^{393})$ [35])	$\min(2^k, O(2^{394})$ [41, 24])	$\min(2^k, O(2^{384})$ [35])

Table 8. Best Known Attack Complexity of HMAC based on SHA-224, SHA-512/224, and SHA-512/256: k is the key size, and ℓ is the internal state size, 2^s is the maximum block length of message.

Table 9 shows the summary of Attacks on MAC, Stream Cipher, and Authenticated Encryption based on reduced versions of SHA-3 hash functions, which is explained in detail in Sect. 6.

Mode	Rounds	Type of Attack	Generic complexity	Attack complexity
MAC	5	Key Recovery	2^{128}	2^{36}
MAC	6	Key Recovery	2^{128}	2^{36}
MAC	7	Key Recovery	2^{128}	2^{97}
MAC	7	Forgery	2^{128}	2^{65}
MAC	8	Forgery	2^{256}	2^{129}
AE (Keyak)	6	Key Recovery	2^{128}	2^{36}
AE (Keyak)	7	Key Recovery	2^{128}	2^{76}
AE (Keyak)	7	Forgery	2^{128}	2^{65}
Stream Cipher	6	Key Recovery	2^{128}	2^{35}
Stream Cipher	8	Keystream Prediction	2^{256}	2^{128}
Stream Cipher	9	Keystream Prediction	2^{512}	2^{256}

Table 9. Summary of Attacks on MAC, Stream Cipher, and Authenticated Encryption based on Keccak [25]

3 Detailed Security Analysis

3.1 SHA-224

SHA-224 [47] is a truncated version of SHA-256 with a different initial value. In other words, SHA-224 is exactly same as SHA-256 except its initial value and its final truncation, where the 224-bit message digest is obtained by truncating the final hash value to its left-most 224 bits. Therefore, existing cryptanalytic techniques to SHA-256, which do not depend only on the initial value of SHA-256, can be applied to SHA-224.

Best Practical Collision-type Attack on SHA-256 The current best practical collision-type attack on SHA-256 is a semi-free-start collision attack on 38-step of SHA-256 [43]. Let IV be the initial value of SHA-256. Actually, we can consider a family of hash functions,$\{H_{iv}\}_{iv \in \mathcal{IV}}$, by varying the initial value of SHA-256. So, we can describe SHA-256(M) as $H_{IV}(M)$. A semi-free-start collision of SHA-256 means that there are two (IV', M) and (IV', M') such that $H_{IV'}(M) = H_{IV'}(M')$, where IV' is different from the initial value IV of SHA-256 and $M \neq M'$.

Now, we start to explore a practical semi-free-start collision attack, with 2^{37} time complexity, on the 38 steps of SHA-256 given in [43]. Let $f(\cdot, \cdot)$ be the compression function of SHA-256. Before finding a semi-free-start collision for SHA-256, we want to find a collision (IV', M) and (IV', M') pair for f, where $|IV'| = 256$ and $|M| = |M'| = 512$ and $M \neq M'$. Once we can find such collision pair for f, the collision (IV', M) and (IV', M') pair is also a semi-free-start collision pair for SHA-256, because SHA-256 is based on Merkle-Damgård domain extension [20, 44], or called MD construction, with the message length padding after message.

From now on, we focus only on finding two a collision (IV', M) and (IV', M') pair for f.

By the big endian order, M and M' are converted into sixteen 32-bit words as follows:

$$\begin{aligned} M &\Longrightarrow W_0||W_1||W_2|| \cdots ||W_{15} \\ M' &\Longrightarrow W'_0||W'_1||W'_2|| \cdots ||W'_{15} \end{aligned}$$

Then, expanded message words W_t and W'_t, which are used at t-th step, are defined as follows:

$$\begin{aligned} W_t &= \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} \qquad &\text{for} \quad 16 \leq t \leq 37 \\ W'_t &= \sigma_1(W'_{t-2}) + W'_{t-7} + \sigma_0(W'_{t-15}) + W'_{t-16} \qquad &\text{for} \quad 16 \leq t \leq 37 \end{aligned}$$

As we can see from Fig. 10, [43] considered W_t and W'_t satisfying the following conditions:

$$\begin{aligned} W_i &\neq W'_i \text{ for } i \in \{7, 8, 10, 15, 23, 24\}, \\ W_i &= W'_i \text{ for } i \notin \{7, 8, 10, 15, 23, 24\}. \end{aligned}$$

W_i	7	8	10	15	23	24
0						
1						
2						
3						
4						
5						
6						
7	x					
8		x				
9						
10			x			
11						
12						
13						
14						
15				x		
16						
17			x	x		
18						
19						
20						
21						
22	x			x		
23	x	x			x	
24		x				x
25			x		x	
26			x			x
27						
28						
29						
30				x	x	
31				x		x
32						
33						
34						
35						
36						
37						

Fig. 10. Message word differences and message word conditions for the attacks on the 38 steps of SHA-256 [43]. Rows show the individual steps of the message expansion to compute W_i. Columns (and highlighted rows) show those expanded message words which contain a difference. An occurrence of a message word in the message expansion equation is denoted by 'x'. For all rows which are not highlighted but contain an 'x', the message differences must cancel [43].

Now, a question arises, "How can we 1) find a 18-step differential characteristic from step 7 to step 24 and 2) find a confirming message pair?"

To search for a differential characteristic and a confirming message pair, [43] use the same approach and automatic search tool as in [42]. But, [43] improved the selection of starting points for the search (message words which contain differences) and the search strategy.

Similar to [42], the basic idea of the search algorithm for differential characteristics consists of three parts which are decision (guessing), deduction (propagation), and backtracking (correction). The search algorithm proceeds as follows:

- Let U be a set of bits. Repeat the following until U is empty:
  1. **Decision(Guessing)**
     (a) Pick randomly (or according to some heuristic) a bit in U.
     (b) Impose new constraints on this bit.
  2. **Deduction(Propagation)**

   (a) Propagate the new information to other variables and equations as described in [42].
   (b) If an inconsistency is detected start backtracking, else continue with step 1-(a).
3. **Backtracking(Correction)**
   (a) Try a different choice for the decision bit.
   (b) If all choices result in an inconsistency, mark the bit as critical.
   (c) Continue with step 1-(a).

In the deduction, [43] use generalization conditions on bits described in Fig. 11. A generalized condition takes all 16 possible conditions on a pair of bits into account. More in detail, through the following three steps [43], we can find a characteristic and its conforming message pair.

- **Stage 1:** We first search for a consistent differential characteristic in the message expansion. Hence, we only add unconstrained bits '?' or 'x' of W_i to the set U_1. Furthermore, we try to reduce the number of conditions after step 15 in the message expansion. In this case, it is more likely to find confirming message pairs in the last stage of the search. To get a sparser characteristic in this area, we pick decision bits more often from the last few steps of the message expansion.
- **Stage 2:** Once we have found a differential characteristic for the message expansion, we continue with searching for a differential characteristic in the state update. We add all unconstrained bits '?' or 'x' of chaining variables a ,b , c, d, e, f, g, and h of Fig. 1 to the set U_2. Note that we pick decision bits more often from a ,b , c, and d, since this results in sparser characteristics for a ,b , c, and d. Similar to Stage 1, experiments have shown that in this case, confirming message pairs are easier to find in the last stage.
- **Stage 3:** In the last stage, we search for confirming inputs. We only pick decision bits '-' which are constrained by linear two-bit conditions, similar as in [42]. This ensures that those bits which influence a lot of other bits are guessed first. Additionally, at least all bits influenced by two-bit conditions propagate as well. This way, inconsistent characteristics can be detected earlier and valid solutions are found faster.

$(X_i, X_i{}^*)$	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$	(X_i, X_i^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓	3	✓	✓	-	-
-	✓	-	-	✓	5	✓	-	✓	-
x	-	✓	✓	-	7	✓	✓	✓	-
0	✓	-	-	-	A	-	✓	-	✓
u	-	✓	-	-	B	✓	✓	-	✓
n	-	-	✓	-	C	-	-	✓	✓
1	-	-	-	✓	D	✓	-	✓	✓
#	-	-	-	-	E	-	✓	✓	✓

Fig. 11. Notation for all generalized conditions on a pair of bits [16].

As written in [43], after Stage 3 finishes, we already get a confirming message pair which results in a semi-free-start collision. The corresponding differential characteristic for 38 steps of SHA-256 is given in Fig. 12.

i	∇A_i	∇E_i	∇W_i
-4	--------------------------------	--------------------------------	
-3	--------------------------------	--------------------------------	
-2	--------------------------------	--------------------------------	
-1	--------------------------------	--------------------------------	
0	--------------------------------	--------------------------------	--------------------------------
1	--------------------------------	--------------------------------	--------------------------------
2	--------------------------------	--------------------------------	--------------------------------
3	--------------------------------	--------------------------------	--------------------------------
4	--------------------------------	--------------------------------	--------------------------------
5	--------------------------------	--------------------1----1------	--------------------------------
6	--------------------------------	1--00--1-------1-10111---0-1--0-	--------------------------------
7	-n----u-n-u---u-------n-nn------	nu-11-0uuun101-uuuuuuu1u-u-1--01	--nnnnn--nn--un----nuuu-nn------
8	----nn-n--n--un-uu-u-u-u-n------	n01nn-1n10000-1u1uuunu00n0-nn0n1	0000011011101--1100nuuuuuuuuuu00
9	u---u-n--nuuu---n-uuu--u-------n	000n00u10001n101nu0u000111-u11un	--------------------------------
10	--------------------------------	00unn00110n1001u11u-101u111u0uun	--------unnnnnnnu---------------
11	--------------------------------	1u1-11u11-n01n100n10u1-11u100011	------1-------------------------
12	--------------------------------	0uu1u1u0u1uu1n01nn111u011n-01010	----------1-----01--------------
13	--------------------------------	n0010uu01-00n1-01n0nu10u10-1-nuu	----------------0--------------1
14	--------------------------------	101-110-1-0010-10--111-010---100	--------------------------------
15	----u----------n----------------	0-1-u01----00--n-00----10----111	-----------------------------u--
16	-----------------------------u--	----n-n1---01-un11------nuu-nu01	--------------------------------
17	--------------------------------	-0--n-1---0nnnnn-nuu1---011-1-un	--------------------------------
18	--------------------------------	----0-1---00000--000--1011101100	--------------------------------
19	--------------------------------	0---u-00nuuuuuuu0001--0011011011	--------------------------------
20	--------------------------------	1--1--11100111---1--0unnnnnnnn0-	--------------------------------
21	--------------------------------	----1---11111111------00000000--	--------------------------------
22	--------------------------------	----------------------11111111--	--------------------------------
23	--------------------------------	--------------------------------	----n---------un----------------
24	--------------------------------	--------------------------------	-----------------------------n--
25	--------------------------------	--------------------------------	--------------------------------
26	--------------------------------	--------------------------------	--------------------------------
27	--------------------------------	--------------------------------	--------------------------------
28	--------------------------------	--------------------------------	--------------------------------
29	--------------------------------	--------------------------------	--------------------------------
30	--------------------------------	--------------------------------	--------------------------------
31	--------------------------------	--------------------------------	--------------------------------
32	--------------------------------	--------------------------------	--------------------------------
33	--------------------------------	--------------------------------	--------------------------------
34	--------------------------------	--------------------------------	--------------------------------
35	--------------------------------	--------------------------------	--------------------------------
36	--------------------------------	--------------------------------	--------------------------------
37	--------------------------------	--------------------------------	--------------------------------

Fig. 12. Differential characteristic for the semi-free-start collision attack on SHA-256 reduced to 38 steps [43]. Bits with gray background have at least one additional condition. [43].

According to [43], it took 8 hours on a single CPU to find the confirming message pair of Fig. 13, which is equivalent to about 2^{37} SHA-256 evaluations.

h_0	ba75b4ac c3c9fd45 fce04f3a 6d620fdb 42559d01 b0a0cd10 729ca9bc b284a572
m	4f5267f8 8f8ec13b 22371c61 56836f2b 459501d1 8078899e 98947e61 4015ef31
	06e98ffc 4babda4a 27809447 3bf9f3be 7b3b74e1 065f711d 6c6ead5e a1781d54
m^*	4f5267f8 8f8ec13b 22371c61 56836f2b 459501d1 8078899e 98947e61 7e73f1f1
	06e99000 4babda4a 277f1447 3bf9f3be 7b3b74e1 065f711d 6c6ead5e a1781d50
Δm	00000000 00000000 00000000 00000000 00000000 00000000 00000000 3e661ec0
	00001ffc 00000000 00ff8000 00000000 00000000 00000000 00000000 00000004
h_1	9312c19e d18b19eb d9c3c91f 36c4e589 4ab410cb 692af674 72cfd427 8e5a0d0a

Fig. 13. Example of a semi-free-start collision for 38 steps of SHA-256 [43].

Security of SHA-224 against Theoretical Preimage-type Attacks Till now, most of preimage-type attacks are focused on SHA-256 and SHA-512 so we need to check whether the preimage-type attacks on SHA-256 can be also applied to SHA-224.

Table 10 summarizes known theoretical preimage-type attacks on SHA-256.

Published in	Year	Attack Method	Attack	Round	Complexity
Preimages for step-reduced SHA-2 [1]	2009	Meet-in-the-Middle	Preimage	42/64 43/64	$2^{251.7}$ $2^{254.9}$
Advanced meet-in-the-middle preimage attacks [34]	2010	Meet-in-the-Middle	Preimage	42/64	$2^{248.4}$
Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family [38]	2012	Biclique	Preimage Pseudo-Preimage	45/64 52/64	$2^{255.5}$ 2^{255}

Table 10. Best Known Theoretical-but-Marginal Preimage-type Attacks against SHA-256

As shown in Fig. 14, the meet-in-the-middle preimage attacks[1, 34] need to match the values of two internal states generated by forward and backward directions. However, unlike SHA-256, SHA-224's internal state size is 24-bit-longer than its hash output size of 224-bit, whereas the internal state size and the hash output size of SHA-256 are same as 256-bit. Usually the attack complexity based on meet-in-the-middle attack approach depends on the size of the internal state. Moreover, as shown in Table 10, the attack complexities of meet-in-the-middle preimage attacks on SHA-256 are already beyond than the general preimage attack complexity 2^{224} for SHA-224. Therefore, it is expected that SHA-224 provides a better security than SHA-256 against meet-in-the-middle preimage attacks.

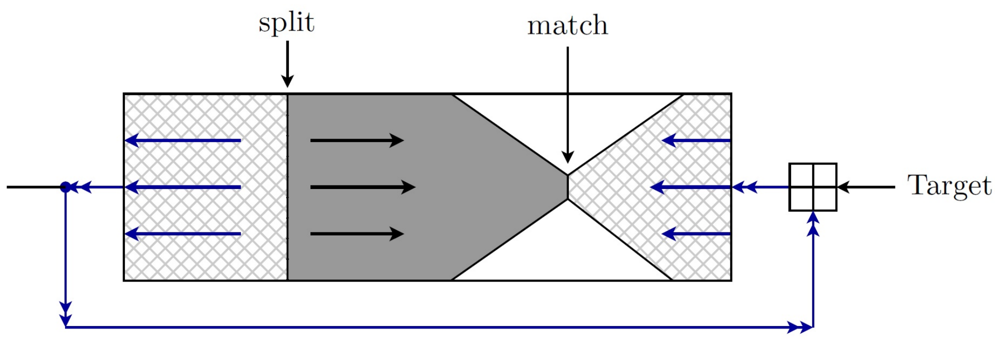


Fig. 14. Meet-in-the-Middle Pseudo-Preimage Attack against Davies-Meyer Hash Functions [34].

As shown in Fig. 15, biclique-based preimage attack [38] also need to match the values of two internal states generated by forward and backward directions. Therefore, it is also expected that SHA-224 provides a better security than SHA-256 against biclique-based preimage-type attacks.

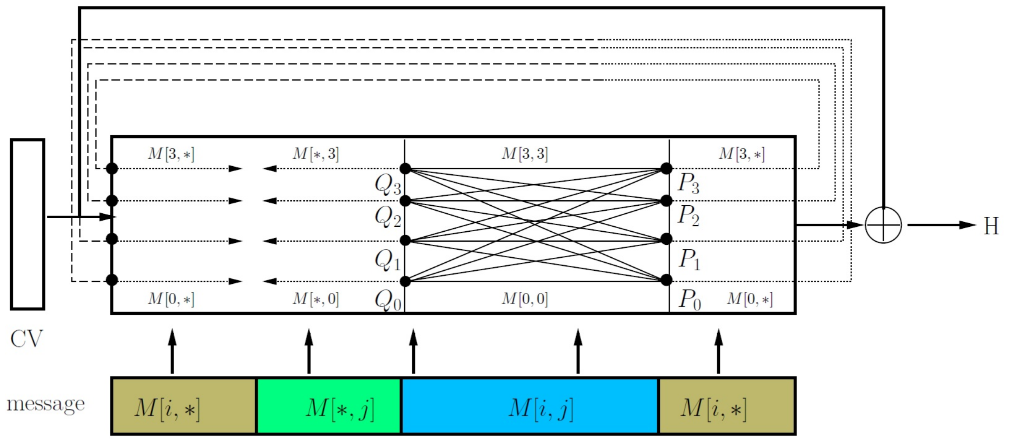


Fig. 15. Biclique of dimension 2 in the meet-in-the-middle attack [38].

Best Practical Distinguishing Attack on SHA-256 The current best practical distinguishing attack on SHA-256 is a second-order differential collision attack (with about 2^{46} time complexity) on 47 steps out of 64 steps of the compression function of SHA-256 [14]. Let f be the compression function of SHA-256.

Definition 1. *A second-order differential collision for f is a two-tuple (a_1, a_2) together with a value y such that*

$$f(y + a_1 + a_2) - f(y + a_1) - f(y + a_2) + f(y) = 0$$

In order to understand the second-order differential collision attack approach on the compression function of SHA-256, firstly we define some basic notations. In cases of SHA-1 and SHA-2 hash functions, they follow Davies-Meyer construction which is a well known method to turn a block cipher into a compression function. The Davis-Meyer compression function is as follows:

$f(y) = E(y) + \tau_n(y)$, where $y = (k||x) \in \{0,1\}^{k+n}$ and E is the underlying block cipher and $\tau_n(y)$ represents the n least significant bits of y.

The underlying block cipher E is split into two subparts, $E{=}E_1 \circ E_0$. Let us assume that there are two differential characteristics as follows:

- **Characteristic 1**: $E_0^{-1}(y + \beta) - E_0^{-1}(y) = \alpha$ holds with probability p_0
- **Characterisitc 2**: $E_1(y + \gamma) - E_1(y) = \delta$ holds with probability p_1.

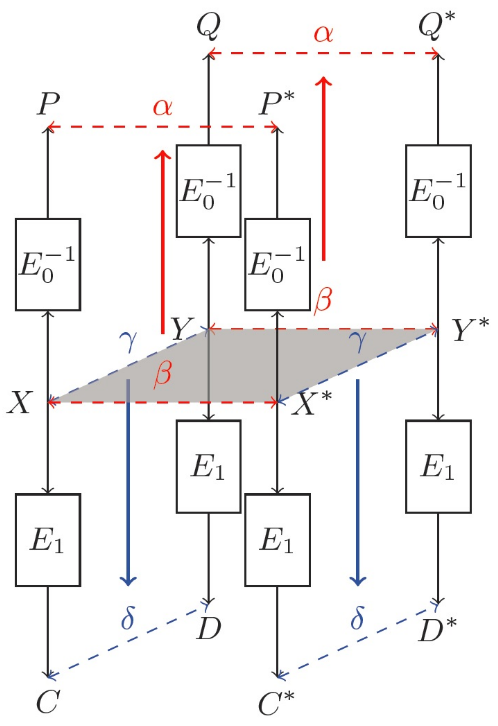


Fig. 16. Schematic view of the attack of Second-Order Differential Collision Attacks [14].

Using these two differential characteristics, we can construct a second-order differential collision for the block cipher E by the following attack procedure [14] (See Fig. 16):

1. Choose a random value for X and compute $X^* = X + \beta$, $Y = X + \gamma$, and $Y^* = X^* + \gamma$.
2. Compute backward from X, X^*, Y, Y^* using E_0^{-1} to obtain P, P^*, Q, Q^*.
3. Compute forward from X, X^*, Y, Y^* using E_1 to obtain C, C^*, D, D^*.
4. Check if $P^* - P{=}Q^* - Q$ and $D - C{=}D^* - C^*$ is fulfilled.

Due to Chracteristic 1 and 2,

$$P^* - P{=}Q^* - Q{=}\alpha \text{ holds with probability at least } p_0^2,$$
$$D - C{=}D^* - C^*{=}\delta \text{ holds with probability at least } p_1^2$$

Therefore, the above attack procedure succeeds with $p_0^2 \times p_1^2$. Also, we can get the following relations [14]:

$$Q^* - Q - P^* + P = 0 \text{ and } E(Q^*) - E(P^*) - E(Q) + E(P) = 0$$

If we set $\alpha := a_1$ and the difference $Q - P := a_2$ we can get the following second-order differential collision relation as follows:

$$E(P + a_1 + a_2) - E(P + a_1) - E(P + a_2) + E(P) = 0$$

Now, we are going to apply the above second-order differential collision attack strategy to 47 steps out of 64 steps of the compression function of SHA-256. The underlying block cipher E, which consists of 47 steps, is split into two subparts, $E{=}E_1 \circ E_0$, where E_0 is the first 22 steps and E_1 is the last 23 steps. Fig. 17 indicates the characteristic of E_0, which will hold with probability 2^{-28}, and Fig. 18 indicates the characteristic of E_1, which will hold with probability 2^{-72}. Though these probabilities are too low, through the message modification techniques, we can significantly reduce the complexity for finding values satisfying the two characteristics, because there is no secret key in case of hash functions.

backward

i	chaining value	message	prob
0	B: -3 E: +10 +24 +29 H: -12 -17 +23		2^{-10}
1	C: -3 F: +10 +24 +29		2^{-4}
2	D: -3 G: +10 +24 +29		2^{-4}
3	E: -3 H: +10 +24 +29		2^{-7}
4	F: -3		2^{-1}
5	G: -3		2^{-1}
6	H: -3	+3	2^{-1}
7			1
⋮	⋮	⋮	⋮
20			1
21		+17 +28	1
22	A: +17 +28 E: +17 +28		

Fig. 17. Differential characteristic for steps 1-22 using signed-bit-differences [14].

forward

i	chaining value	message	prob
22	B: +3 +12 +14 +19 +23 +32 C: +25 E: -3 -7 -13 F: -12 -23 G: -25 H: -1 +3 +7 +14 +15 +24 +26 +28 -30	-25	2^{-22}
23	C: +3 +12 +14 +19 +23 +32 D: +25 F: -3 -7 -13 G: -12 -23 H: -25		2^{-13}
24	A: -25 D: +3 +12 +14 +19 +23 +32 G: -3 -7 -13 H: -12 -23		2^{-10}
25	B: -25 E: +14 +19 +32 H: -3 -7 -13		2^{-7}
26	C: -25 F: +14 +19 +32		2^{-4}
27	D: -25 G: +14 +19 +32		2^{-4}
28	E: -25 H: +14 +19 +32		2^{-4}
29	F: -25		2^{-1}
30	G: -25		2^{-1}
31	H: -25	+25	1
32			1
⋮	⋮	⋮	⋮
45			1
46		-7 -18 -22	2^{-6}
47	A: -7 -18 -22 E: -7 -18 -22		

message modification

Fig. 18. Differential characteristic for steps 23-47 using signed-bit-differences [14]. Note that conditions imposed by the characteristic in steps 23-30 are fulfilled in a deterministic way using message modification techniques [14].

Finally, the second-order differential collision attack on 47 steps of the compression function SHA-256 [14] was confirmed by providing an example of second-order differential collision as shown in Fig. 19.

y	89456784	4ef9daf6	0ab509f5	3fdf6c93	fe7afc67	b03ad81a	fd306df9	1d14cadd
	daea3041	70f45fd7	4a03bf20	c13c961c	6a12c686	fc7be50c	7b060fc2	0ee1e276
	630c3c7e	734246a4	88401eb0	9aac88c1	4b6bca45	b777c1e6	5537cdb1	9b5bc93b
a_1	00000000	00000000	00000000	00000000	00000000	00000000	00000004	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	fffffffc	00000000	fffffffc	10800200	00000000	ff800000	803ef414
a_2	2335e851	20f48326	69151911	f5cb76c2	b9d69e31	32685b9c	90cceff7	081ebbf7
	967c8864	a43138d1	7e9a3eec	c39cf7d3	5914e008	8d0d3b73	e077c63f	d29db1b0
	742b8c01	92248811	a119f182	dd829be5	e3e1802e	21130e9f	1dacd7d3	8acf11fe

Fig. 19. Example of a second-order differential collision $f(y+a_1+a_2) - f(y+a_1) - f(y+a_2) + f(y) = 0$ for 47 steps of the SHA-256 compression function [14].

3.2 SHA-512/224 and SHA-512/256

SHA-512/224 and SHA-512/256 [47] are truncated versions of SHA-512 with different initial values. In other words, SHA-512/224 and SHA-512/256 are exactly same as SHA-512 except their initial values and their final truncation, where the 224-bit message digest for SHA-512/224 (the 256-bit message digest for SHA-512/256) is obtained by truncating the final hash value to its left-most 224 bits (its left-most 256 bits for SHA-512/256).

Unlike SHA-224, SHA-256, SHA-384, and SHA-512, the initial values for SHA-512/224 and SHA-512/256 are not defined by the fractional parts of the cube roots of prime numbers, but defined by calling SHA-512 hash function with a tweaked $IV' = (IV \oplus 0xa5a5a5a5a5........a5)$, where IV is the original initial value of SHA-512. Therefore, existing cryptanalytic techniques to SHA-512, which do not depend only on the initial value of SHA-512, can be applied to SHA-512/224 and SHA-512/256.

Best Practical Collision-type Attack on SHA-512 The current best practical collision-type attack on SHA-512 is a semi-free-start collision attack on the 38 steps (out of 80) of SHA-512 [32]. Let IV be the initial value of SHA-512. Actually, we can consider a family of hash functions,$\{H_{iv}\}_{iv \in \mathcal{IV}}$, by varying the initial value of SHA-512. So, we can describe SHA-512(M) as $H_{IV}(M)$. A semi-free-start collision of SHA-512 means that there are two (IV', M) and (IV', M') such that $H_{IV'}(M) = H_{IV'}(M')$, where IV' is different from the initial value IV of SHA-512 and $M \neq M'$.

Now, we start to explore a practical semi-free-start collision attack, with $2^{40.5}$ time complexity, on the 38 steps of SHA-512 given in [32]. Let $f(\cdot, \cdot)$ be the compression function of SHA-512. Before finding a semi-free-start collision for SHA-512, we want to find a collision (IV', M) and (IV', M') pair for f, where $|IV'| = 512$ and $|M| = |M'| = 1024$ and $M \neq M'$. Once we can find such collision pair for f, the collision (IV', M) and (IV', M') pair is also a semi-free-start collision pair for SHA-512, because SHA-512 is based on Merkle-Damgård domain extension [20, 44], or called MD construction, with the message length padding after message.

From now on, we focus only on finding two a collision (IV', M) and (IV', M') pair for f.

By the big endian order, M and M' are converted into sixteen 64-bit words as follows:

$$M \Longrightarrow W_0||W_1||W_2||\cdots||W_{15}$$
$$M' \Longrightarrow W'_0||W'_1||W'_2||\cdots||W'_{15}$$

Then, expanded message words W_t and W'_t, which are used at t-th step, are defined as follows:

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} \qquad \text{for} \quad 16 \leq t \leq 37$$
$$W'_t = \sigma_1(W'_{t-2}) + W'_{t-7} + \sigma_0(W'_{t-15}) + W'_{t-16} \qquad \text{for} \quad 16 \leq t \leq 37$$

As we can see from Fig. 20, [32] considered W_t and W'_t satisfying the following conditions:

$$W_i \neq W'_i \text{ for } i \in \{7, 8, 10, 15, 23, 24\},$$
$$W_i = W'_i \text{ for } i \notin \{7, 8, 10, 15, 23, 24\}.$$

W_i	7	8	10	15	23	24
0						
1						
2						
3						
4						
5						
6						
7	x					
8		x				
9						
10			x			
11						
12						
13						
14						
15				x		
16						
17			x	x		
18						
19						
20						
21						
22	x			x		
23	x	x			x	
24		x				x
25			x		x	
26			x			x
27						
28						
29						
30				x	x	
31				x		x
32						
33						
34						
35						
36						
37						

Fig. 20. Message word differences and message word conditions for the attacks on the 38 steps of SHA-512 [43, 32]. Rows show the individual steps of the message expansion to compute W_i. Columns (and highlighted rows) show those expanded message words which contain a difference. An occurrence of a message word in the message expansion equation is denoted by 'x'. For all rows which are not highlighted but contain an 'x', the message differences must cancel [43, 32].

Now, a question arises, "How can we 1) find a 18-step differential characteristic from step 7 to step 24 and 2) find a confirming message pair?"

To search for a differential characteristic and a confirming message pair, [32] uses the same approach and automatic search tool as in [43]. But, [32] improved the selection of starting points for the search (message words which contain differences) and the search strategy.

stage	decision bit	decision rule		
		probability	choice 1	choice 2
1–2	?	1	-	x
	x	1/2	u	n
		1/2	n	u
3	-	1/2	0	1
		1/2	1	0

Fig. 21. Decision rules in different search stages [32].

Similar to [43], the guess-then-determine attack [32] consists of three stages. The rules of the guessing strategy are given in Fig. 21 and the three stages are as follows:

– **Stage 1:** We first search for a consistent differential characteristic in the message expansion. Hence, we only add unconstrained bits ('?') and difference bits ('x') of W_i to the set U.
– **Stage 2:** We continue with the search for a differential characteristic in the state update. Hence, we add all unconstrained bits ('?') and difference bits ('x') of chaining variables a ,b , c, d, e, f, g, and h of Fig. 2 to the set U. We pick decision bits more often from a ,b , c, and d, since this results in sparser characteristics for a ,b , c, and d. Experiments have shown that in this case, confirming message pairs are easier to find in the last stage.
– **Stage 3:** In the last stage, we search for confirming message pairs by guessing bits without difference ('-'). We only pick decision bits of eight chaining variables and W_i which are constrained by two-bit conditions, similar as in [42]. This serves as a preselection heuristic for the branching look-ahead in Fig. 22.

```
Algorithm   Look-ahead branching heuristic for differential cryptanalysis
  Let $U$ be a set of undetermined bits and $s_{\max}$ the limit of look-ahead candidates.
  repeat
    Guessing
      1. pick a bit $v \in U$ randomly and increment $s$
      2. impose new constraints on this bit $v$
    Propagation
      3. propagate the new information to other variables and equations
      4. if an inconsistency is detected, return $v$ as the decision bit
         else count the number $m$ of additional variables that were assigned due to
         this guess and save the pair $(v, m)$ in a list $L$.
    Update
      5. remove all variables that were assigned due to the guess $v$ from the set $U$
      6. undo all changes to restore the original assignment
  until $U$ is empty or $s \geq s_{\max}$
  return $v^*$ from $L$ with the highest score $m$ as the decision bit
```

Fig. 22. Look-ahead branching heuristic for differential cryptanalysis [32].

As written in [32], after Stage 3 finishes, we already get a confirming message pair which results in a semi-free-start collision. The corresponding differential characteristic for 38 steps of SHA-512 is given in Fig. 23.

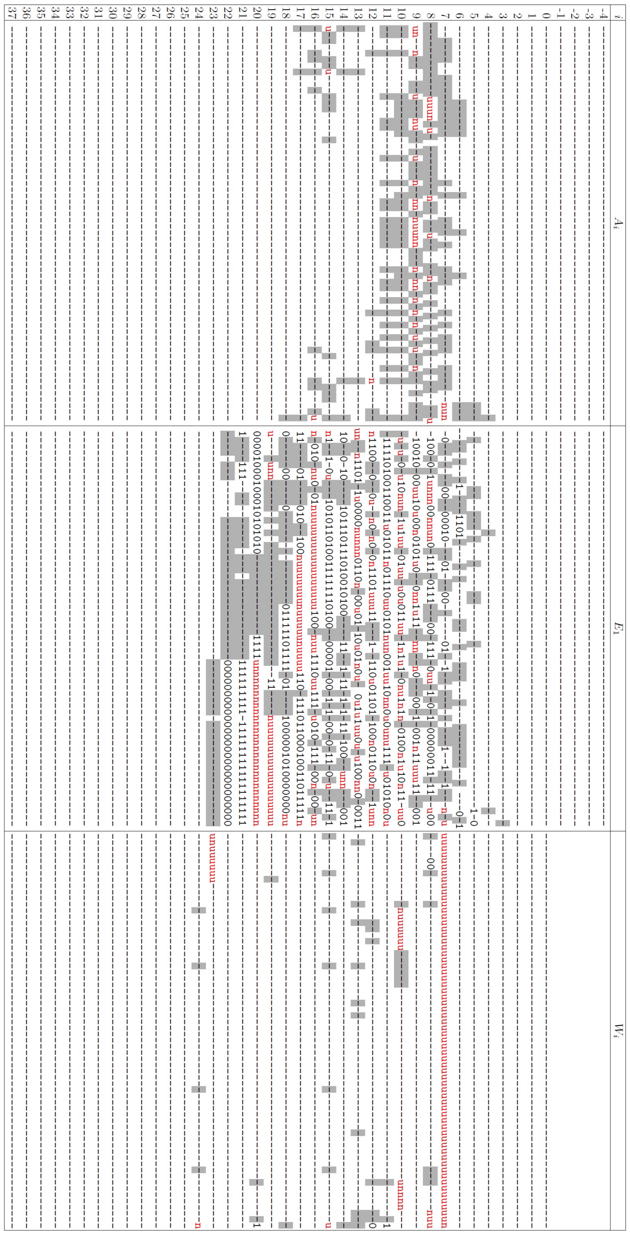


Fig. 23. Differential characteristic for a semi-free-start-collision of SHA-512 reduced to 38 steps (bits with two-bit conditions highlighted) [32].

Using the improvements in the branching heuristic proposed in [32], it took 5441 seconds (≈ 1.5 hours) on a cluster with 40 CPUs to find the confirming message pair of Fig. 24, which is equivalent to a complexity of about $2^{40.5}$ evaluations of the SHA-512 compression function evaluations.

h_0	e8626f53a3771964	2ae427b8c5065790	c8fd5a1628fc3337	0f362d297f82f987
	89166a0c022ffc40	c2c49c30e629239f	d1fa8bd692843025	ad4bba64c797e6ec
m	610519a88f0d2809	3addc83f01c8b179	84afa7a2772c6141	ad539854e64c9cce
	85450b73549b2085	7296b5291f31c0d9	fc978d9624e2c2cc	fffffffffffffffe
	92114cb9d2f4cd9b	34a3198b79871212	cca7f43154e38081	ac0598a589168fe1
	f32ae6a0070a8d2e	755aa5cada87e894	4b9bd7df3c94b667	65291f2b80cc8c51
m^*	610519a88f0d2809	3addc83f01c8b179	84afa7a2772c6141	ad539854e64c9cce
	85450b73549b2085	7296b5291f31c0d9	fc978d9624e2c2cc	0000000000000001
	92114cb9d2f4cd9c	34a3198b79871212	cca8143154e38079	ac0598a589168fe1
	f32ae6a0070a8d2e	755aa5cada87e894	4b9bd7df3c94b667	65291f2b80cc8c50
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	ffffffffffffffff
	0000000000000007	0000000000000000	000fe000000000f8	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000001
h_1	946a28eedc3b2ff6	c4573d0a13ea6268	11f07b04b06900dd	897c606e4053bbe4
	2406aae9d58504b4	89b237932b061ba8	663402cb4bb1972c	d99c062dce945423

Fig. 24. Example of a semi-free-start collision for 38 steps of SHA-512 [32].

Security of SHA-512/224 and SHA-512/256 against Theoretical Preimage-type Attacks Till now, preimage-type attacks on SHA-512 are only available so we need to check whether the preimage-type attacks on SHA-512 can be also applied to SHA-512/224 and SHA-512/256.

Table 11 summarizes known theoretical preimage-type attacks on SHA-512.

Published in	Year	Attack Method	Attack	Round	Complexity
Preimages for step-reduced SHA-2 [1]	2009	Meet-in-the-Middle	Preimage	42/80	$2^{502.3}$
				46/80	$2^{511.5}$
Advanced meet-in-the-middle preimage attacks [34]	2010	Meet-in-the-Middle	Preimage	42/80	$2^{494.6}$
Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family [38]	2012	Biclique	Preimage	50/80	$2^{511.5}$
			Pseudo-Preimage	57/80	2^{511}

Table 11. Best Known Theoretical Preimage-type Attacks against SHA-512

As already shown in Fig. 14, the meet-in-the-middle preimage attacks[1, 34] need to match the values of two internal states generated by forward and backward directions. However, unlike SHA-512, SHA-512/224 and SHA-512/256's internal state sizes are 288-bit-longer and 256-bit-longer than their hash output sizes of 224-bit and 256-bit, respectively, whereas the internal state size and the hash output size of SHA-512 are same as 512-bit. Usually the attack complexity based on meet-in-the-middle attack approach depends on the size of the internal state. Moreover, as shown in Table 11, the attack complexities of meet-in-the-middle preimage attacks on SHA-512 are already beyond than the general preimage attack complexity 2^{224} and 2^{256} for SHA-512/224 and SHA-512/256, respectively. Therefore, it is expected that SHA-512/224 and SHA-512/256 provide better security than SHA-512 against meet-in-the-middle preimage attacks.

As shown in Fig. 15, biclique-based preimage attack [38] also need to match the values of two internal states generated by forward and backward directions. Therefore, it is expected that SHA-

512/224 and SHA-512/256 provide better security than SHA-512 against biclique-based preimage-type attacks.

Best Practical Distinguishing Attack on SHA-512 The current best practical distinguishing attack on SHA-512 is a boomerang attack (with about 2^{51} time complexity) on 48 steps out of 80 steps of the compression function of SHA-512 [53]. Let H be 48-step reduced version of the compression function of SHA-512, H_0 and H_1 be two sub-ciphers: $H = H_1 \circ H_1$. The boomerang attack on a compression function can be described as follows [53]. Especially, in [53], H_0 is the first 23 steps of and H_1 is the last 25 steps.

step	Δw_i	Δa	Δb	Δc	Δd	Δe	Δf	Δg	Δh
		64			23,25,30,36,46,50	0	0	28,36	25,30
1			64			23,46,50			28,36
2				64			23,46,50		
3					64			23,46,50	
4						64			23,46,50
5							64		
6								64	
7									64
8	64								
9-22									
23	56,57,63	56,63				56,63			

Fig. 25. The top differential path used for boomerang attack on SHA-512 [53].

step	Δw_i	Δa	Δb	Δc	Δd	Δe	Δf	Δg	Δh
23	33,40		2,7,13,23,27,64	41		5,13	2,7	41	$\Sigma_1(\Delta e_{23})$
24	41			2,7,13,23,27,64	41		5,13	2,7	41
25		41			2,7,13,23,27,64			5,13	2,7
26			41			23,27,64			5,13
27				41			23,27,64		
28					41			23,27,64	
29						41			23,27,64
30							41		
31								41	
32									41
33	41								
34-47									
48	33,40	33,40	0	0	0	33,40	0	0	0

Fig. 26. The bottom differential path used for boomerang attack on SHA-512 [53].

- Choose a random chaining value $v^{(1)}$ and a message $w^{(1)}$, compute $v^{(2)} = v^{(1)} + \beta$, $v^{(3)} = v^{(1)} + \gamma$, $v^{(4)} = v^{(3)} + \beta$ and $w^{(2)} = w^{(1)} + \beta_w$, $w^{(3)} = w^{(1)} + \gamma_w$, $w^{(4)} = w^{(3)} + \beta_w$. We get a quartet $S = \{(v^{(i)}, w^{(i)}) | i = 1, 2, 3, 4\}$.
- Compute backward from the quartet S using H_0^{-1} to obtain the initial values, IV_1, IV_2, IV_3, and IV_4.
- Compute forward from the quartet S using H_1 to obtain the output values h_1, h_2, h_3, and h_4.
- Check whether $IV_2 - IV_1 = IV_4 - IV_3 = \alpha$ and $h_3 - h_1 = h_4 - h_2 = \delta$ are fulfilled.

Let Δa be the XOR difference between a and a', and $\Delta a : i$ $(1 \le i \le 64)$ is used to denote that the i-th bit of a is different from the i-th bit of a', and the rest of the bits of a and a' are the same. Let w_i be the expanded 64-bit message word at i-th step. Fig. 25 is the characteristic for H_0 and Fig. 26 is the characteristic for H_1 [53].

Then, the boomerang attack procedure on the 48-step reduced version of the compression function of SHA-256, which is described in [53], is as follows:

1. Randomly select eleven 64-bit message words $w_i^{(1)}$ $(22 \leq \leq 32)$, and a 512-bit chaining variables $v_{23}^{(1)}$=$(a_{23}^{(1)}, b_{23}^{(1)}, \cdots, h_{23}^{(1)})$. Modify the messages $w_i^{(1)}$ $(22 \leq i \leq 32)$ to meet the conditions in Fig. 28. Compute $v_i^{(1)}$ $(23 \leq i \leq 33)$. Modify $v_{23}^{(1)}$ and $w_i^{(1)}$ $(22 \leq i \leq 32)$ so that $v_i^{(1)}$ $(24 \leq i \leq 33)$ satisfy all the 59 conditions in Fig. 29.
2. Let $w_i^{(2)} = w_i^{(1)} \oplus \Delta w_i^{(1,2)}$, $w_i^{(3)} = w_i^{(1)} \oplus \Delta w_i^{(1,3)}$, $w_i^{(4)} = w_i^{(2)} \oplus \Delta w_i^{(1,3)}$ $(22 \leq i \leq 32)$, where let $w_i^{(j_1,j_2)}$ denote the XOR difference of $w_i^{(j_1)}$ and $w_i^{(j_2)}$. The message differences $\Delta w_i^{(1,2)}$ and $\Delta w_i^{(1,3)}$ are defined in Fig. 27. Compute $v_i^{(j)}$ $(j = 2, 3, 4; 23 \leq i \leq 33)$. Modify $v_{23}^{(1)}$ and $w_i^{(1)}$ $(22 \leq i \leq 32)$ so that $v_i^{(1)}$ $(24 \leq i \leq 33)$ satisfy all the 59 conditions in Fig. 29 in one side and 18 conditions at step 23 in Fig. 29 in the other side. Check whether $v_{33}^{(1)} \oplus v_{33}^{(3)} = v_{33}^{(2)} \oplus v_{33}^{(4)} = 0$, which will hold with about 2^{-41}, because there are 41 more conditions in step 24-31 in Fig. 29. If yes, goto the next step. (Note that in [53], 2^{-40} is used rather than 2^{-41}, which is not clear) Otherwise, go back to step 1.
3. Select five 64-bit message words $w_i^{(1)}$ $(17 \leq \leq 21)$ randomly, which means that there are 320-bit freedom to choose. Let $w_i^{(2)} = w_i^{(1)}$ $(17 \leq i \leq 21)$. Compute $w_i^{(1)}$ and $w_i^{(2)}$ $(33 \leq i \leq 47, 0 \leq i \leq 16)$ in forward and backward directions separately. Let $w_i^{(3)} = w_i^{(1)}$ and $w_i^{(4)} = w_i^{(2)}$ when $33 \leq i \leq 37$. Compute $w_i^{(3)}$ and $w_i^{(4)}$ when $38 \leq i \leq 47$ and $0 \leq i \leq 21$ by the message expansion.
4. Compute $v_{22}^{(j)}, v_{21}^{(j)}, \cdots, v_0^{(j)}$ $(j = 1, 2, 3, 4)$ in forward direction. Check whether $v_0^{(2)} - v_0^{(1)} = v_0^{(4)} - v_0^{(3)}$ and $v_{48}^{(2)} - v_{48}^{(1)} = v_{48}^{(4)} - v_{48}^{(3)}$, which can hold with probability 2^{-51}, because the probability of steps 22 to 1 of the top differential path is about 2^{-45} (but, it was not explained in [53] how the authors got this probability 2^{-45}.), and the probability of the message expansion is 2^{-6}. If yes, output $w_i^{(j)}$ $(j = 1, 2, 3, 4; 0 \leq i \leq 15)$ and $v_1^{(j)}$ $(j = 1, 2, 3, 4)$. Otherwise, go to step 3.

Hence, the complexity of the 48-step attack is $2^{40} + 2^{45} \times 2^6 \approx 2^{51}$, while its generic best attack requires the complexity 2^{256} by the zero-sum distinguishing attack [2]. And [53] provided an example of boomerang distinguisher as shown in Fig. 30.

i	$\Delta w^{(1,2)}$	$\Delta w^{(1,3)}$
22	4180000000000000	0000008100000000
23	8000000000000000	0000010000000000
24	0502081000000002	0
25	0200100000000004	0
26	2804080001020010	0
27	1008000002000020	0
28	00825520891408a1	0
29	c184220110080140	0
30	0504804080200408	0
31	0001008000400800	0
32	2$ab$100$a$291089050	0000010000000000

Fig. 27. Message differences in steps 23 to 33 [53].

message	conditions
$w_{22}^{(1)}$	$w_{22,15}^{(1)} = w_{22,14}^{(1)}$, $w_{22,44}^{(1)} = w_{22,43}^{(1)} \oplus w_{22,35}^{(1)} \oplus w_{22,34}^{(1)} \oplus w_{22,15}^{(1)} \oplus w_{22,14}^{(1)}$, $w_{22,48}^{(1)} = w_{22,47}^{(1)} \oplus w_{22,14}^{(1)} \oplus w_{22,15}^{(1)} \oplus w_{22,6}^{(1)} \oplus w_{22,5}^{(1)}$, $w_{22,57}^{(1)} = w_{22,56}^{(1)} \oplus 1$
$w_{23}^{(1)}$	$w_{23,41}^{(1)} = w_{22,33}^{(1)} \oplus w_{23,34}^{(1)} \oplus w_{23,40}^{(1)}$, $w_{23,42}^{(1)} = w_{23,40}^{(1)} \oplus w_{23,35}^{(1)} \oplus w_{23,34}^{(1)} \oplus 1$, $w_{23,48}^{(1)} = w_{23,40}^{(1)} \oplus w_{23,41}^{(1)} \oplus w_{23,47}^{(1)} \oplus 1$
$w_{24}^{(1)}$	$w_{24,2}^{(1)} = w_{22,21}^{(1)} \oplus w_{22,63}^{(1)} \oplus w_{22,8}^{(1)}$, $w_{24,37}^{(1)} = w_{22,57}^{(1)} \oplus w_{22,35}^{(1)} \oplus w_{22,44}^{(1)}$, $w_{24,37}^{(1)} = w_{22,57}^{(1)} \oplus w_{22,35}^{(1)} \oplus w_{22,44}^{(1)}$, $w_{24,44}^{(1)} = w_{22,63}^{(1)} \oplus w_{22,41}^{(1)} \oplus w_{22,50}^{(1)}$, $w_{24,50}^{(1)} = w_{22,6}^{(1)} \oplus w_{22,48}^{(1)} \oplus w_{22,57}^{(1)}$, $w_{24,57}^{(1)} = w_{22,12}^{(1)} \oplus w_{22,54}^{(1)} \oplus w_{22,63}^{(1)}$, $w_{24,59}^{(1)} = w_{22,15}^{(1)} \oplus w_{22,57}^{(1)}$
$w_{25}^{(1)}$	$w_{25,3}^{(1)} = w_{23,22}^{(1)} \oplus w_{23,64}^{(1)} \oplus w_{23,9}^{(1)}$, $w_{25,45}^{(1)} = w_{23,64}^{(1)} \oplus w_{23,42}^{(1)} \oplus w_{23,51}^{(1)}$, $w_{25,58}^{(1)} = w_{23,13}^{(1)} \oplus w_{23,55}^{(1)} \oplus w_{23,64}^{(1)}$
$w_{26}^{(1)}$	$w_{26,5}^{(1)} = w_{24,24}^{(1)} \oplus w_{24,2}^{(1)} \oplus w_{24,11}^{(1)}$, $w_{26,18}^{(1)} = w_{24,37}^{(1)} \oplus w_{24,15}^{(1)} \oplus w_{24,24}^{(1)}$, $w_{26,25}^{(1)} = w_{24,44}^{(1)} \oplus w_{24,22}^{(1)} \oplus w_{24,31}^{(1)}$, $w_{26,44}^{(1)} = w_{24,63}^{(1)} \oplus w_{24,41}^{(1)} \oplus w_{24,50}^{(1)}$, $w_{26,51}^{(1)} = w_{24,6}^{(1)} \oplus w_{24,48}^{(1)} \oplus w_{24,57}^{(1)}$, $w_{26,60}^{(1)} = w_{24,15}^{(1)} \oplus w_{24,57}^{(1)}$, $w_{26,62}^{(1)} = w_{24,17}^{(1)} \oplus w_{24,59}^{(1)}$
$w_{27}^{(1)}$	$w_{27,6}^{(1)} = w_{25,25}^{(1)} \oplus w_{25,3}^{(1)} \oplus w_{25,12}^{(1)}$, $w_{27,26}^{(1)} = w_{25,45}^{(1)} \oplus w_{25,23}^{(1)} \oplus w_{25,32}^{(1)}$, $w_{27,52}^{(1)} = w_{25,7}^{(1)} \oplus w_{25,49}^{(1)} \oplus w_{25,58}^{(1)}$, $w_{27,57}^{(1)} = w_{27,19}^{(1)} \oplus w_{27,39}^{(1)} \oplus w_{27,6}^{(1)} \oplus w_{24,44}^{(1)} \oplus 1$, $w_{27,61}^{(1)} = w_{25,16}^{(1)} \oplus w_{25,58}^{(1)}$
$w_{28}^{(1)}$	$w_{28,1}^{(1)} = w_{26,20}^{(1)} \oplus w_{26,62}^{(1)} \oplus w_{26,7}^{(1)}$, $w_{28,6}^{(1)} = w_{26,25}^{(1)} \oplus w_{26,3}^{(1)} \oplus w_{26,12}^{(1)}$, $w_{28,8}^{(1)} = w_{26,27}^{(1)} \oplus w_{26,5}^{(1)} \oplus w_{26,14}^{(1)}$, $w_{28,12}^{(1)} = w_{26,31}^{(1)} \oplus w_{26,9}^{(1)} \oplus w_{26,18}^{(1)}$, $w_{28,19}^{(1)} = w_{26,38}^{(1)} \oplus w_{26,16}^{(1)} \oplus w_{26,25}^{(1)}$, $w_{28,21}^{(1)} = w_{26,40}^{(1)} \oplus w_{26,18}^{(1)} \oplus w_{26,27}^{(1)}$, $w_{28,25}^{(1)} = w_{26,44}^{(1)} \oplus w_{26,22}^{(1)} \oplus w_{26,31}^{(1)}$, $w_{28,28}^{(1)} = w_{26,47}^{(1)} \oplus w_{26,25}^{(1)} \oplus w_{26,34}^{(1)}$, $w_{28,32}^{(1)} = w_{26,51}^{(1)} \oplus w_{26,29}^{(1)} \oplus w_{26,38}^{(1)}$, $w_{28,38}^{(1)} = w_{26,57}^{(1)} \oplus w_{26,35}^{(1)} \oplus w_{26,44}^{(1)}$, $w_{28,41}^{(1)} = w_{26,60}^{(1)} \oplus w_{26,38}^{(1)} \oplus w_{26,47}^{(1)}$, $w_{28,43}^{(1)} = w_{26,62}^{(1)} \oplus w_{26,40}^{(1)} \oplus w_{26,49}^{(1)}$, $w_{28,45}^{(1)} = w_{26,64}^{(1)} \oplus w_{26,42}^{(1)} \oplus w_{26,51}^{(1)}$, $w_{28,47}^{(1)} = w_{26,2}^{(1)} \oplus w_{26,44}^{(1)} \oplus w_{26,53}^{(1)}$, $w_{28,50}^{(1)} = w_{26,5}^{(1)} \oplus w_{26,47}^{(1)} \oplus w_{26,56}^{(1)}$, $w_{28,56}^{(1)} = w_{26,11}^{(1)} \oplus w_{26,53}^{(1)} \oplus w_{26,62}^{(1)}$
$w_{29}^{(1)}$	$w_{29,7}^{(1)} = w_{27,26}^{(1)} \oplus w_{27,4}^{(1)} \oplus w_{27,13}^{(1)}$, $w_{29,9}^{(1)} = w_{27,28}^{(1)} \oplus w_{27,6}^{(1)} \oplus w_{27,15}^{(1)}$, $w_{29,14}^{(1)} = w_{22,56}^{(1)} \oplus w_{24,59}^{(1)}$, $w_{29,15}^{(1)} = w_{22,57}^{(1)} \oplus w_{27,59}^{(1)} \oplus 1$, $w_{29,20}^{(1)} = w_{27,39}^{(1)} \oplus w_{27,17}^{(1)} \oplus w_{27,26}^{(1)}$, $w_{29,21}^{(1)} = w_{22,63}^{(1)} \oplus w_{24,2}^{(1)} \oplus w_{29,8}^{(1)} \oplus 1$, $w_{29,29}^{(1)} = w_{27,48}^{(1)} \oplus w_{27,26}^{(1)} \oplus w_{27,35}^{(1)}$, $w_{29,33}^{(1)} = w_{27,52}^{(1)} \oplus w_{27,30}^{(1)} \oplus w_{27,39}^{(1)}$, $w_{29,42}^{(1)} = w_{27,61}^{(1)} \oplus w_{27,39}^{(1)} \oplus w_{27,48}^{(1)}$, $w_{29,43}^{(1)} = w_{22,56}^{(1)} \oplus w_{24,37}^{(1)} \oplus w_{29,34}^{(1)}$, $w_{29,44}^{(1)} = w_{22,56}^{(1)} \oplus w_{29,34}^{(1)} \oplus w_{29,43}^{(1)} \oplus w_{22,57}^{(1)} \oplus w_{29,35}^{(1)} \oplus 1$, $w_{29,46}^{(1)} = w_{27,1}^{(1)} \oplus w_{27,43}^{(1)} \oplus w_{27,52}^{(1)}$, $w_{29,47}^{(1)} = w_{22,56}^{(1)} \oplus w_{24,50}^{(1)} \oplus w_{29,5}^{(1)}$, $w_{29,48}^{(1)} = w_{22,56}^{(1)} \oplus w_{29,6}^{(1)} \oplus w_{22,57}^{(1)} \oplus w_{29,5}^{(1)} \oplus w_{29,47}^{(1)}$, $w_{29,50}^{(1)} = w_{24,44}^{(1)} \oplus w_{29,41}^{(1)} \oplus w_{22,63}^{(1)}$, $w_{29,51}^{(1)} = w_{27,6}^{(1)} \oplus w_{27,48}^{(1)} \oplus w_{27,57}^{(1)}$, $w_{29,54}^{(1)} = w_{24,57}^{(1)} \oplus w_{29,12}^{(1)} \oplus w_{22,63}^{(1)}$, $w_{29,55}^{(1)} = w_{24,57}^{(1)} \oplus w_{29,13}^{(1)} \oplus w_{27,19}^{(1)} \oplus w_{27,61}^{(1)} \oplus 1$, $w_{29,56}^{(1)} = w_{22,56}^{(1)}$, $w_{29,57}^{(1)} = w_{22,57}^{(1)}$, $w_{29,58}^{(1)} = w_{29,6}^{(1)} \oplus w_{29,48}^{(1)} \oplus w_{29,57}^{(1)} \oplus w_{29,7}^{(1)} \oplus w_{29,49}^{(1)} \oplus 1$, $w_{29,63}^{(1)} = w_{22,63}^{(1)}$, $w_{29,64}^{(1)} = w_{27,19}^{(1)} \oplus w_{27,61}^{(1)}$
$w_{30}^{(1)}$	$w_{30,4}^{(1)} = w_{28,23}^{(1)} \oplus w_{28,1}^{(1)} \oplus w_{28,10}^{(1)}$, $w_{30,11}^{(1)} = w_{28,30}^{(1)} \oplus w_{28,8}^{(1)} \oplus w_{28,17}^{(1)}$, $w_{30,22}^{(1)} = w_{28,41}^{(1)} \oplus w_{28,19}^{(1)} \oplus w_{28,28}^{(1)}$, $w_{30,32}^{(1)} = w_{28,51}^{(1)} \oplus w_{28,29}^{(1)} \oplus w_{28,38}^{(1)}$, $w_{30,33}^{(1)} = w_{30,11}^{(1)} \oplus w_{30,20}^{(1)} \oplus w_{30,32}^{(1)} \oplus w_{30,10}^{(1)} \oplus w_{30,19}^{(1)} \oplus 1$, $w_{30,39}^{(1)} = w_{28,58}^{(1)} \oplus w_{28,36}^{(1)} \oplus w_{28,45}^{(1)}$, $w_{30,42}^{(1)} = w_{25,45}^{(1)} \oplus w_{25,3}^{(1)} \oplus w_{30,22}^{(1)} \oplus w_{30,9}^{(1)} \oplus w_{28,6}^{(1)} \oplus w_{28,48}^{(1)} \oplus w_{28,57}^{(1)}$, $w_{30,45}^{(1)} = w_{30,23}^{(1)} \oplus w_{30,32}^{(1)} \oplus w_{30,44}^{(1)} \oplus w_{30,22}^{(1)} \oplus w_{30,31}^{(1)} \oplus 1$, $w_{30,48}^{(1)} = w_{28,3}^{(1)} \oplus w_{28,45}^{(1)} \oplus w_{28,54}^{(1)}$, $w_{30,51}^{(1)} = w_{28,6}^{(1)} \oplus w_{28,48}^{(1)} \oplus w_{28,57}^{(1)}$, $w_{30,52}^{(1)} = w_{30,30}^{(1)} \oplus w_{30,39}^{(1)} \oplus w_{30,51}^{(1)} \oplus w_{30,29}^{(1)} \oplus w_{30,38}^{(1)}$, $w_{30,54}^{(1)} = w_{30,32}^{(1)} \oplus w_{30,41}^{(1)} \oplus w_{30,51}^{(1)} \oplus w_{30,29}^{(1)} \oplus w_{30,38}^{(1)} \oplus 1$, $w_{30,57}^{(1)} = w_{28,12}^{(1)} \oplus w_{28,54}^{(1)} \oplus w_{28,63}^{(1)}$, $w_{30,59}^{(1)} = w_{28,14}^{(1)} \oplus w_{28,56}^{(1)}$, $w_{30,64}^{(1)} = w_{25,3}^{(1)} \oplus w_{30,22}^{(1)} \oplus w_{30,9}^{(1)} \oplus 1$
$w_{31}^{(1)}$	$w_{31,12}^{(1)} = w_{29,31}^{(1)} \oplus w_{29,9}^{(1)} \oplus w_{29,18}^{(1)}$, $w_{31,23}^{(1)} = w_{29,42}^{(1)} \oplus w_{29,20}^{(1)} \oplus w_{29,29}^{(1)}$, $w_{31,40}^{(1)} = w_{29,59}^{(1)} \oplus w_{29,37}^{(1)} \oplus w_{29,46}^{(1)}$, $w_{31,49}^{(1)} = w_{29,4}^{(1)} \oplus w_{29,46}^{(1)} \oplus w_{29,55}^{(1)}$
$w_{32}^{(1)}$	$w_{32,5}^{(1)} = w_{30,24}^{(1)} \oplus w_{30,2}^{(1)} \oplus w_{30,11}^{(1)}$, $w_{32,7}^{(1)} = w_{30,26}^{(1)} \oplus w_{30,4}^{(1)} \oplus w_{30,13}^{(1)}$, $w_{32,13}^{(1)} = w_{30,33}^{(1)} \oplus w_{30,11}^{(1)} \oplus w_{30,20}^{(1)}$, $w_{32,16}^{(1)} = w_{30,35}^{(1)} \oplus w_{30,13}^{(1)} \oplus w_{30,22}^{(1)}$, $w_{32,20}^{(1)} = w_{30,39}^{(1)} \oplus w_{30,17}^{(1)} \oplus w_{30,26}^{(1)}$, $w_{32,25}^{(1)} = w_{30,45}^{(1)} \oplus w_{30,23}^{(1)} \oplus w_{30,32}^{(1)}$, $w_{32,29}^{(1)} = w_{30,48}^{(1)} \oplus w_{30,26}^{(1)} \oplus w_{30,35}^{(1)}$, $w_{32,32}^{(1)} = w_{30,51}^{(1)} \oplus w_{30,29}^{(1)} \oplus w_{30,38}^{(1)} \oplus 1$, $w_{32,34}^{(1)} = w_{30,51}^{(1)} \oplus w_{30,29}^{(1)} \oplus w_{30,38}^{(1)} \oplus 1$, $w_{32,38}^{(1)} = w_{30,57}^{(1)} \oplus w_{30,35}^{(1)} \oplus w_{30,44}^{(1)}$, $w_{32,40}^{(1)} = w_{30,59}^{(1)} \oplus w_{30,37}^{(1)} \oplus w_{30,46}^{(1)}$, $w_{32,47}^{(1)} = w_{23,47}^{(1)} \oplus 1$, $w_{32,49}^{(1)} = w_{30,4}^{(1)} \oplus w_{30,46}^{(1)} \oplus w_{30,55}^{(1)}$, $w_{32,53}^{(1)} = w_{30,8}^{(1)} \oplus w_{30,50}^{(1)} \oplus w_{30,59}^{(1)}$, $w_{32,54}^{(1)} = w_{30,9}^{(1)} \oplus w_{30,51}^{(1)} \oplus w_{30,60}^{(1)}$, $w_{32,56}^{(1)} = w_{30,11}^{(1)} \oplus w_{30,53}^{(1)} \oplus w_{30,62}^{(1)}$, $w_{32,58}^{(1)} = w_{27,58}^{(1)}$, $w_{32,60}^{(1)} = w_{30,15}^{(1)} \oplus w_{30,57}^{(1)}$, $w_{32,62}^{(1)} = w_{30,17}^{(1)} \oplus w_{30,59}^{(1)}$

Fig. 28. The message conditions in $w_{22}^{(1)} - w_{32}^{(1)}$ [53].

steps	conditions
23	$a^{(1)}_{23,56} = w^{(1)}_{22,56} \oplus 1,\ a^{(1)}_{23,62} = w^{(1)}_{22,63} \oplus a^{(1)}_{23,3} \oplus w^{(1)}_{22,56} \oplus a^{(1)}_{23,4} \oplus a^{(1)}_{23,57},\ a^{(1)}_{23,63} = w^{(1)}_{22,63}$
	$b^{(1)}_{23,41} = a^{(1)}_{23,41}$
	$c^{(1)}_{23,2} = a^{(1)}_{23,2},\ c^{(1)}_{23,7} = a^{(1)}_{23,7},\ c^{(1)}_{23,13} = a^{(1)}_{23,13},\ c^{(1)}_{23,23} = a^{(1)}_{23,23},\ c^{(1)}_{23,27} = a^{(1)}_{23,27},\ c^{(1)}_{23,63} = b^{(1)}_{23,63} \oplus 1,\ c^{(1)}_{23,64} = a^{(1)}_{23,64}$
	$e^{(1)}_{23,13} = b^{(1)}_{23,13} \oplus 1,\ e^{(1)}_{23,56} = w^{(1)}_{22,56} \oplus 1,\ e^{(1)}_{23,63} = w^{(1)}_{22,63}$
	$f^{(1)}_{23,2} = b^{(1)}_{23,2} \oplus 1,\ f^{(1)}_{23,7} = b^{(1)}_{23,7} \oplus 1$
	$g^{(1)}_{23,41} = c^{(1)}_{23,41},\ g^{(1)}_{23,63} = f^{(1)}_{23,63}$
24	$a^{(1)}_{24,2} = b^{(1)}_{24,2},\ a^{(1)}_{24,7} = b^{(1)}_{24,7},\ a^{(1)}_{24,13} = b^{(1)}_{24,13},\ a^{(1)}_{24,23} = b^{(1)}_{24,23},\ a^{(1)}_{24,27} = b^{(1)}_{24,27},\ a^{(1)}_{24,41} = b^{(1)}_{24,41},\ a^{(1)}_{24,63} = a^{(1)}_{23,63} \oplus 1,\ a^{(1)}_{24,64} = b^{(1)}_{24,64}$
	$e^{(1)}_{24,2} = 1,\ e^{(1)}_{24,5} = 0,\ e^{(1)}_{24,7} = 1,\ e^{(1)}_{24,13} = 0$
25	$a^{(1)}_{25,41} = h^{(1)}_{24,41},\ a^{(1)}_{25,36} = a^{(1)}_{25,30} \oplus h^{(1)}_{24,41} \oplus h^{(1)}_{25,2} \oplus 1,\ a^{(1)}_{25,46} = a^{(1)}_{25,35} \oplus h^{(1)}_{24,41} \oplus h^{(1)}_{25,7} \oplus 1,\ a^{(1)}_{25,52} = a^{(1)}_{25,47} \oplus h^{(1)}_{24,41} \oplus d^{(1)}_{25,13}$
	$e^{(1)}_{25,5} = 1,\ e^{(1)}_{25,13} = 0,\ e^{(1)}_{25,23} = f^{(1)}_{25,23} \oplus 1,\ e^{(1)}_{25,27} = f^{(1)}_{25,27},\ e^{(1)}_{25,64} = f^{(1)}_{25,64}$
26	$e^{(1)}_{26,23} = d^{(1)}_{25,23},\ e^{(1)}_{26,27} = d^{(1)}_{25,27},\ e^{(1)}_{26,41} = c^{(1)}_{26,41},\ e^{(1)}_{26,46} = e^{(1)}_{26,23} \oplus e^{(1)}_{26,19} \oplus e^{(1)}_{23,5} \oplus 1,\ e^{(1)}_{26,54} = e^{(1)}_{26,31} \oplus e^{(1)}_{26,27} \oplus e^{(1)}_{23,13} \oplus 1,\ e^{(1)}_{26,64} = e^{(1)}_{26,37} \oplus e^{(1)}_{26,41} \oplus e^{(1)}_{26,23} \oplus g^{(1)}_{26,23} \oplus 1$
27	$e^{(1)}_{27,23} = 0,\ e^{(1)}_{27,27} = 0,\ e^{(1)}_{27,64} = 0$
	$a^{(1)}_{27,41} = b^{(1)}_{27,41}$
28	$e^{(1)}_{28,23} = 1,\ e^{(1)}_{28,27} = 1,\ e^{(1)}_{28,41} = f^{(1)}_{28,41},\ e^{(1)}_{28,64} = 1$
29	$e^{(1)}_{29,41} = d^{(1)}_{28,41},\ e^{(1)}_{29,45} = e^{(1)}_{29,41} \oplus e^{(1)}_{29,4} \oplus h^{(1)}_{29,27} \oplus 1,\ e^{(1)}_{29,18} = e^{(1)}_{29,14} \oplus d^{(1)}_{28,41} \oplus h^{(1)}_{29,64} \oplus 1,\ e^{(1)}_{29,64} = e^{(1)}_{29,37} \oplus d^{(1)}_{28,41} \oplus h^{(1)}_{29,23} \oplus 1$
30	$e^{(1)}_{30,41} = 0$
31	$e^{(1)}_{31,41} = 1$

Fig. 29. The conditions of chaining variables in the middle steps [53].

$IV^{(1)}$	d51d68d22cd614bb	ad109f079123bc43	3e30194750de9356	b934d669f648b886
	2788083c8af206a4	f53a6844e79ca3ff	83333924f0fb45ee	aeca4ed80990f3c1
$IV^{(2)}$	551d68d22cd614bb	ad109f079123bc43	be30194750de9356	3936f6621588b886
	a788083c8af206a4	753a4844e79ca3ff	0333591cf87b45ee	2ec84edfead0f3c1
$IV^{(3)}$	3ca41aa7cc2ed702	d28a0787d13ece62	aaa0ccee378c5884	45960268826fa783
	126c152e3ed3c3d8	90227712dcb66469	c96f7308aa86be3c	5adecf0ca7c8cff9
$IV^{(4)}$	bca41aa7cc2ed702	d28a0787d13ece62	2aa0ccee378c5884	c5982260a1afa783
	926c152e3ed3c3d8	10225712dcb66469	496f9300b206be3c	dadccf148908cff9
$M^{(1)}$	7897cf7f1c02fa18	c0e30c69c197577d	f6016b4df4a5101b	44cf12bc7c5f7f89
	d28a43112a41160f	a481e26554edd575	8a4f5ecd8ee90f42	0c10896df299f0a3
	8bd715591505422b	82f9e09643a6f94e	8ae783224a988778	d858b794e8b95a4a
	d98d2e211f08b5e3	3185a2321c2013d0	493b7695ecb8bc63	40dde2bb03f050f7
$M^{(2)}$	7897cf7f1c02fa18	c0e30c69c197577d	f6016b4df4a5101b	44cf12bc7c5f7f89
	d28a43112a41160f	a481e26554edd575	8a4f5ecd8ee90f42	8c10896df299f0a3
	8bd715591505422b	82f9e09643a6f94e	8ae783224a988778	d858b794e8b95a4a
	d98d2e211f08b5e3	3185a2321c2013d0	493b7695ecb8bc63	40dde2bb03f050f7
$M^{(3)}$	2ec928b5e9b2bae2	da67703373f8f947	c4c2b463d9c34453	a4d359b70a54809d
	829416361d1acc84	49208682435343aa	8a4c7b5efe34b2e8	d6bcd7d0a70c5663
	ef5a6123cadba871	a134d5cebfae6e21	e32944037719f06e	81033c0b86b9f18e
	9d4d5849a78a6aa9	4634d6dd6a193ca7	783f014e5106c88e	bcd2f996a68b63f7
$M^{(4)}$	2ec928b5e9b2bae2	da67703373f8f947	c4c2b463d9c34453	a4d359b70a54809d
	829416361d1acc84	49208682435343aa	8a4c7b5efe34b2e8	56bcd7d0a70c5663
	ef5a6123cadba871	a134d5cebfae6e21	e32944037719f06e	81033c0b86b9f18e
	9d4d5849a78a6aa9	4634d6dd6a193ca7	783f014e5106c88e	bcd2f996a68b63f7

Fig. 30. Example of a quart satisfying $H(IV^{3)}, M^{(3)}) - H(IV^{(1)}, M^{(1)}) - H(IV^{(4)}, M^{(4)}) + H(IV^{(2)}, M^{(2)}) = 0$ for 48 steps of the SHA-512 compression function. [53].

3.3 SHA3-224

As shown in Sect. 1.3, SHA3-224 is defined as follows:

$$\text{SHA3-224}(M) = \textsc{Keccak}[448](M||01, 224)$$

However, since all known cryptanalytic results were done only for Keccak[448]$(M, 224)$ without the two-bit 10 padding, the results cannot be directly applied to SHA3-224. Nonetheless, the results on Keccak[448]$(M, 224)$ clearly show the security strength of SHA3-224, because the attack techniques on Keccak[448]$(M, 224)$ can be also applied to SHA3-224 with a small change by considering the two additional padding bits. Therefore, in this subsection, we focus on describing all known cryptanalytic results on Keccak[448]$(M, 224)$. For simplicity, we will call Keccak[448]$(M, 224)$ Keccak-224(M).

Best Practical Collision-type Attack on Keccak-224 The best practical collision attack on Keccak-224 is the attack on the four round version of Keccak-224, which is described in [22]. They even provided a collision pair which was found within a few minutes on a single PC. In this subsection, we will describe their attack idea step by step. Note that A round function R of Keccak-224 consists of five steps, θ, ρ, π, χ, and ι.

```
|----------------|----------------|---------------1|-------4--------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|-------4--------|----8-----------|
|----------------|----------------|---------------1|----------------|----8-----------|

|----------------|----------------|----------------|--8-------------|2---------------|
|4---------------|----------------|----------------|----------------|2---------------|
|----------------|----------------|----------------|--8-------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|4---------------|----------------|----------------|----------------|----------------|

|----------------|----------------|----------------|----------------|----------------|
|-----------8----|-----------2----|----------------|----------------|----------------|
|----------------|----------------|-----------1----|----------------|-----------1----|
|---------1------|-------4--------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

Fig. 31. A Double Kernel trail for 2-round characteristic leading to collisions on Keccak-224 and Keccak-256 [27, 46, 22]: The probability of the first transition is 2^{-12}. The probability of the second transition is 1, since there are no active Sboxes which affect the output. $-$ indicates the four-bit zero difference. Note that the order of lanes is from left to right.

For understanding the attack, we need to know the notion of a *column parity kernel* or *CP-kernel* that was defined in the Keccak submission document [9]: a 1600-bit state difference is in the CP-kernel if all of its columns have even parity. It is easy to see that if a state differece a is in the CP-kernel, a is a fixed state of θ, that is, $a = \theta(a)$. Since ρ and π just reorder the bits of the state, the total hamming weight of a state is preserved. In other words, when let $a' = \pi(\rho(a))$, then $HW(a) = HW(a')$, where $HW(x)$ means the hamming weight of x. Also, when a' is a low hamming weight differential state, $\chi(a')$ will be same as a' with a high probability. Since ι is the operation of adding a constant, there is no change of difference and hamming weight. Therefore, if an input state a of a round function R is in CP-kernel, then $HW(a) = HW(R(a))$ with a high probability.

In [27] and [46], the authors provided differential characteristics that input differential states of the first and second rounds stay in the CP-kernel, which are named *double kernel trails* in [46]. The 2-round characteristic in Fig. 31 is an example of a double kernel trail, where $-$ indicates the four-bit zero difference. Since KECCAK-224 takes the first 224 bits of the final 1600-bit state, the 2-round characteristic in Fig. 31 provides a collision after two rounds in case of KECCAK-224. Later, the 2-round characteristic in Fig. 31 will be used for 4-round practical collision attack on KECCAK-224 in [22].

In [22], the authors backwardly extended one more round from the 2-round characteristic of Fig. 31 as shown in Fig. 32. The characteristic is a 3-round characteristic leading to collisions on KECCAK-224 with probability 2^{-24} [22]. As we can see from Fig. 32, the hamming weight of the input differential state of the 3-round characteristic is very high due to the diffusion effect by θ.

```
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2676F26B|357C4789AF3-6AF1|78D3526BC4A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E265EF26B|357C4789AF3-4AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF226C4D78366789|C4DAE35E2656F26B|35FC4789AF3-6AF1|78D3526BC6A74C4D|

|----------------|----------------|---------------1|-------4--------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|-------4--------|----8-----------|
|----------------|----------------|---------------1|----------------|----8-----------|

|----------------|----------------|----------------|--8-------------|2---------------|
|4---------------|----------------|----------------|----------------|2---------------|
|----------------|----------------|----------------|--8-------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|4---------------|----------------|----------------|----------------|----------------|

|----------------|----------------|----------------|----------------|----------------|
|-----------8----|-----------2----|----------------|----------------|----------------|
|----------------|----------------|-----------1----|----------------|-----------1----|
|---------1------|-------4--------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

Fig. 32. A 3-round characteristic leading to a collision on KECCAK-224 with probability 2^{-24} [22]. Note that the order of lanes is from left to right.

Let $\varDelta T$ be the input differential state of the 3-round characteristic of Fig. 32 as shown in Fig. 33.

```
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2676F26B|357C4789AF3-6AF1|78D3526BC4A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E265EF26B|357C4789AF3-4AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF226C4D78366789|C4DAE35E2656F26B|35FC4789AF3-6AF1|78D3526BC6A74C4D|
```

Fig. 33. A Target Difference ΔT: This is the input differential state of the 3-round characteristic shown in Fig. 32. Note that the order of lanes is from left to right.

Here, we focus on collision attack on 4-round Keccak-224. For any hash output size, $r+c$=1600, where r is the bitrate and c is the capacity. In case of Keccak-224, r=1152 and c=448, where the capacity size is defined as 2×224. Also, as the padding rule, pad10*1 is used. In case of Keccak-224, given any message M, pad10*1(M)=$M||10^t1$, where t is the least non-negative integer such that the bit-size of pad10*1(M) is a multiple of r(=1152).

Let R be the round function of Keccak-224 and let p be the 8-bit pad 10000001. In order to find a 4-round collision using the characteristic of Fig. 32, [22] provides an answer to the following question,

"Given a target difference ΔT, how can we find two different 1144-bit M and M' such that $R(M||p||0^{448}) \oplus R(M'||p||0^{448}) = \Delta T$?"

[22] developed an algorithm, called *the target difference algorithm (TDA)*, to solve the above problem. Given a target difference ΔT, the TDA efficiently produces message pairs (M, M')'s such that $R(M||p||0^{448}) \oplus R(M'||p||0^{448}) = \Delta T$.

And Let $DDT(\delta^{in}, \delta^{out})$ be $\{x \in \{0,1\}^5 | \chi_{|5}(x) \oplus \chi_{|5}(x \oplus \delta^{in}) = \delta^{out}\}$.

In order to answer to the above question, we need to understand the following important property, which is already described in [22],

"For any non-zero 5-bit output difference δ^{out} to Keccak Sbox $\chi_{|5}$, the set of possible input differences, $\{\delta^{in}|DDT(\delta^{in}, \delta^{out}) > 0\}$, contains at least 5 (and upto 17) 2-dimensional affine subspaces,"

A 2-dimensional affine subspace means that it can be described by $\{x, x \oplus y, x \oplus z, x \oplus y \oplus z\}$, where x is a 5-bit value, and y and z ($y \neq z$) are any non-zero 5-bit values. Therefore, the above property shows that there are at least five 2-dimensional affine subspaces which are contained in $\{\delta^{in}|DDT(\delta^{in}, \delta^{out}) > 0\}$.

Firstly, given a ΔT, we know the possible 1600-bit input different of χ from $\{\delta^{in}|DDT(\delta^{in}, \delta^{out}) > 0\}$, where δ^{out} is defined according to ΔT. But, our concern is not $\{\delta^{in}|DDT(\delta^{in}, \delta^{out}) > 0\}$, but an affine subspace $\{x, x \oplus y, x \oplus z, x \oplus y \oplus z\}$ which are contained in $\{\delta^{in}|DDT(\delta^{in}, \delta^{out}) > 0\}$, where δ^{out} is defined according to ΔT.

Since the bit-size of ΔT is 1600, we can divide ΔT into 320 5-bit differences and take only non-zero 5-bit differences (say that there are t non-zero 5-bit differences from ΔT), denoted by δ_i^{out} for $1 \leq i \leq t$. And for each δ_i^{out}, let us choose one affine subspace, say $AS_i = \{x_i, x_i \oplus y_i, x_i \oplus z_i, x_i \oplus y_i \oplus z_i\}$ for $1 \leq i \leq t$, which are contained in $\{\delta^{in}|DDT(\delta^{in}, \delta_i^{out}) > 0\}$. Therefore, any 5-bit difference, say s_i, in AS_i can be possible to produce its corresponding five-bit output difference δ_i^{out}.

Let L^{-1} be $\theta^{-1} \circ \rho^{-1} \circ \pi^{-1}$.

Then, our interest is how can we select $(s_1, s_2, ..., s_t)$ such that the last 456-bit of $L^{-1}(s)$ is $p||0^{448}$, where s be the 1600-bit input difference of χ which is determined by $(s_1, s_2, ..., s_t)$. For this, firstly, [22] suggests to describe each AS_i (for all i) by *three* linear equations over 1-bit.

For each 5-bit s_i, we denote $s_i = s_{i,1}||s_{i,2}||s_{i,3}||s_{i,4}||s_{i,5}$. For $AS_i = \{x_i, x_i \oplus y_i, x_i \oplus z_i, x_i \oplus y_i \oplus z_i\}$, we denote $x_i = x_{i,1}||x_{i,2}||x_{i,3}||x_{i,4}||x_{i,5}$, $y_i = y_{i,1}||y_{i,2}||y_{i,3}||y_{i,4}||y_{i,5}$, and $z_i = z_{i,1}||z_{i,2}||z_{i,3}||z_{i,4}||z_{i,5}$.

Therefore, we can form the following five linear equations and each bit of s_i is defined by two 1-bit variable a and b only.

$$\begin{aligned}
s_{i,1} &= x_{i,1} \oplus a \cdot y_{i,1} \oplus b \cdot z_{i,1} \\
s_{i,2} &= x_{i,2} \oplus a \cdot y_{i,2} \oplus b \cdot z_{i,2} \\
s_{i,3} &= x_{i,3} \oplus a \cdot y_{i,3} \oplus b \cdot z_{i,3} \\
s_{i,4} &= x_{i,4} \oplus a \cdot y_{i,4} \oplus b \cdot z_{i,4} \\
s_{i,5} &= x_{i,5} \oplus a \cdot y_{i,5} \oplus b \cdot z_{i,5}
\end{aligned}$$

Any s_i is possible as long as there are a and b satisfying the above five linear equations. Since there are two 1-bit variables a and b, we can reduce the number of equations from 5 to 3 by replacing a and b as follows.

- Since for any i y_i and z_i are non-zero 5-bit differences and $y_i \neq z_i$, there should exist a j such that $y_{i,j} \neq z_{i,j}$. Without loss of generality, assume $j = 1$ and $y_{i,j} = 1$, so $z_{i,1} = 0$.
- Rearrange the first equation and get a new linear equation, $a = s_{i,1} \oplus x_{i,1}$, and apply this to the second equation.
- The second equation is changed into a new equation $b = s_{i,2} \oplus x_{i,2} \oplus (s_{i,1} \oplus x_{i,1}) \cdot y_{i,2}$.
- Apply these two equations to the remaining three equations by replacing a and b.
- Finally, we get the following three equations and any s_i is a possible input difference of $chi_{|5}$ when the three equations holds. That is, given ΔT, in order to find s_i, we have to solve the following three linear equations with five variables $s_{i,1}$, $s_{i,2}$, $s_{i,3}$, $s_{i,4}$, and $s_{i,5}$.

$$\begin{aligned}
s_{i,3} &= x_{i,3} \oplus (s_{i,1} \oplus x_{i,1}) \cdot y_{i,3} \oplus (s_{i,2} \oplus x_{i,2} \oplus (s_{i,1} \oplus x_{i,1}) \cdot y_{i,2}) \cdot z_{i,3} \\
s_{i,4} &= x_{i,4} \oplus (s_{i,1} \oplus x_{i,1}) \cdot y_{i,4} \oplus (s_{i,2} \oplus x_{i,2} \oplus (s_{i,1} \oplus x_{i,1}) \cdot y_{i,2}) \cdot z_{i,4} \\
s_{i,5} &= x_{i,5} \oplus (s_{i,1} \oplus x_{i,1}) \cdot y_{i,5} \oplus (s_{i,2} \oplus x_{i,2} \oplus (s_{i,1} \oplus x_{i,1}) \cdot y_{i,2}) \cdot z_{i,5}
\end{aligned}$$

Since $1 \leq i \leq t$, we can construct 3t linear equations with $5t$ variables in total. Also, since the last 456-bit of $L^{-1}(s)$ should be $p||0^{448}$, 456 linear equations are added. Therefore, we have to solve $(3t + 456)$ equations with $5t$ variables. For example, for $t = 310$, as shown in [22], we can get a solution subset of dimension at least 164, that is, we can find 2^{164} s's which are possible input difference of χ and satisfy the padding constraint given by $p||0^{448}$. Moreover, given any s, we can calculate actual values using $DDT(\delta^{in}, \delta^{out})$. Therefore, we can find 2^{164} (M,M') pairs from 2^{164} s's by applying L^{-1}. In this way, the target difference algorithm was developed in [22].

Now, we only need to connect one of these (M,M') pairs to the 3-round characteristic given in Fig. 32 with probability 2^{-24}. Finally, we can get a 4-round collision as shown in Fig. 34.

```
M1=

FAC7AC69 2710BE04 8462C382 7ABF1BF9 D065CD30 DB64DEB8 1410CD30 C837D79B 22E446B7 31E9BD55
A6B2074C C86E32CC DE50A10A F7BAAA58 D96CBC88 9FBD75F6 5E0D735A D22AA663 16A574AA 7DB08692
558AB029 109B4D30 86CE5DCA 13A295C7 E7C9D94B 648794D2 62EE3CF8 69439337 8CAB9F15 AC7C3267
90F41CBE A20E6893 B4781F24 0BA37647 F29A67A0 81F628D0

M2=

CE5FBC81 47710FCC 462C92E0 48F5D2CF F92F6EC3 053E64E1 570780B9 F838EC54 8F74809F 66B4AC6F
70DD1843 BF34F0C5 5010C89A D8791148 D5CC073E 3239AEBC 7DF48D79 0EC7767B FB081604 AFA975B9
F8EFAE0F ED793473 479E931C F2F80A74 7192D08F 5EB5AB27 F1CAC04E F394232D 48656B2A A3471644
DB74E60A 05FB3B18 41DC27C3 8384BF53 32534C3E 811C00B5

Output=

826B10DC 0670E4E1 5B510CDA AB876AA8 B50557ED 267932FB AA4D38E8
```

Fig. 34. A collision for 4-round Keccak-224 [22].

Best Practical Preimage and Second-preimage Attack on SHA3-224 There are two types of practical preimage attacks on reduced versions of underlying permutation of SHA3-224. One is the dedicated attack utilizing a round structure of the underlying permutation of SHA3-224. The other is the general attack (not depending on specific structure) using the SAT solver algorithms. Till now, the best dedicated attack is a 2-round preimage attack on Keccak-224 and Keccak-256 with time complexity 2^{33} and memory complexity 2^{29} [46]. The best SAT-solver-based general attack is a 3-round preimage attack on Keccak-p with time complexity about 2^{34} when the message size is 40-bit and hash output size is 1024-bit [45].

Dedicated Practical Preimage Attack on 2-round Keccak-224 and Keccak-256 [46]: *See Fig. 35. As explained in [46], we are given a hash value, which is 4 (in case of* Keccak-*256* according to the index ordering in Fig. 4) out of the 5 white lanes in the most right slice #4, in Fig. 35, that represents the final state after the permutation. In Fig. 35, each white lane is a lane known and fixed, each colored one, a not-yet-fixed lane. The fifth lane is not known but we can choose a random value for it and fix it.

See Fig. 36: $a_0, a_1, \cdots, e_0, e_1$ are message conditions such that $a_0 = a_1, \cdots, e_0 = e_1$ to bypass θ's diffusion effect. Therefore, there are 318 ($=4 \times 64 + 62$) degrees of freedom for the message, because we can choose any values for a_0, b_0, c_0, e_0 and can freely choose the first 62 bits of d_0 due to 10*1 padding rule. The number k under x_i means that x_i at slice z on the left side gets moved to slice $z + k$ (mod 64) after the transformation

In the forward direction from #1 to #2, we can compute the known white lanes in #2 after θ, ρ, and π. In the backward direction from #4 to #3, we can compute the values of 5 white lanes in #3 from the fixed value of 5 white lanes in #4. Now, we only need to find suitable 318 bit message bits which enables the forward and backward directions to be matched. In this way, we can find preimages of 2-round Keccak-224 and Keccak-256 with time complexity 2^{33} and memory complexity 2^{29} [46].

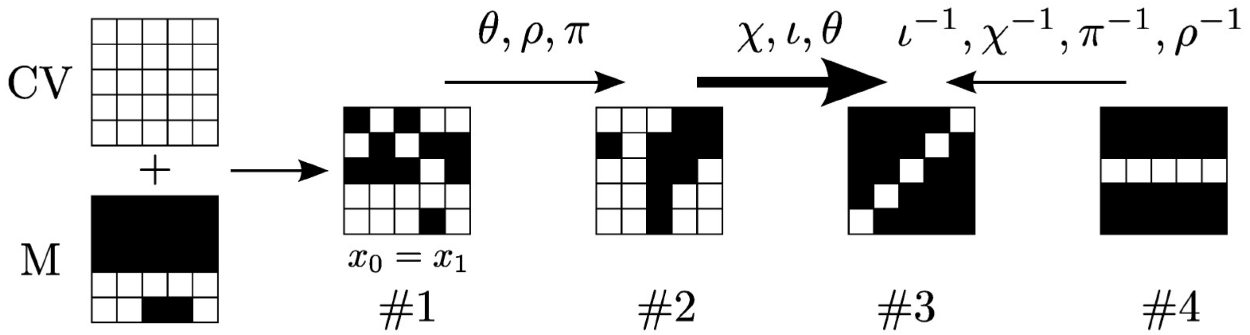


Fig. 35. Strategy of 2-Round Preimage Attack on Keccak-224 and Keccak-256 [46]: the numbering of indexes follows Fig. 4. Each square represents a 64 bit lane. Each white lane is a lane known and fixed, each colored one, a not-yet-fixed lane.

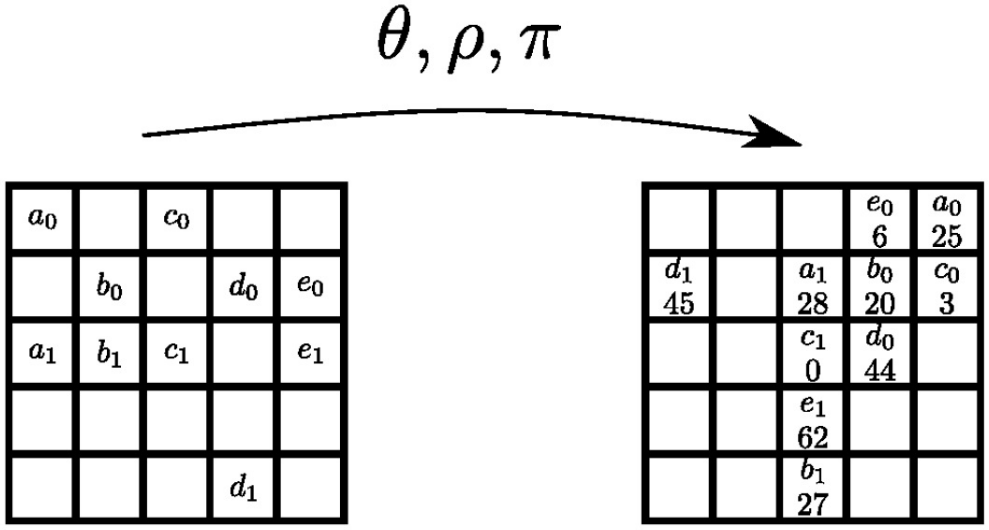


Fig. 36. Shows how the bits $a_0, a_1, \cdots, e_0, e_1$ get moved by θ, ρ, and π [46].

Practical Preimage Attack on 3-round Keccak Using SAT solvers[45]: Before we explain about SAT and the application of SHA solvers to preimage attacks on hash functions, let us consider the following example in [49].

$$p \vee q \vee r \qquad p \vee \neg q \vee \neg r \qquad \neg p \vee q \vee \neg r \qquad \neg p \vee \neg q \vee r$$

$p \vee q \vee r$, $p \vee \neg q \vee \neg r$, $\neg p \vee q \vee \neg r$, and $\neg p \vee \neg q \vee r$ are called "*clauses*". Especially, the above four clauses are related to $p \oplus q \oplus r = 1$. More precisely, any solution (p, q, r) of the boolean equation '$p \oplus q \oplus r = 1$' makes the values of above all the four clauses '1'. A formula $(p \vee q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r)$ is a conjunction of the four clauses, which is called a conjunctive normal form(CNF). We say that if there is a solution (p, q, r) making all the four clauses of the formula '1', we say that the formula in CNF is *satisfiable*. In case of XOR operation with n literals, 2^{n-1} clauses are required to form a formular in CNF related to a problem of finding a solution of the XOR operation with n literals.

Another example is for 32-bit modular addition [49]. As explained in [49], a full-adder circuit takes three inputs, x, y, and c_{in} (the three bits to be summed), and outputs two bits, s (the lower bit of the sum) and c_{out} (the carry bit). We can give s and c_{out} as functions as follows [49]:

$$\begin{aligned} s(x, y, c_{in}) &= x \oplus y \oplus c_{in} \\ c_{out}(x, y, c_{in}) &= ((x \oplus y) \wedge c_{in}) \vee (x \wedge y) \end{aligned}$$

Each of the binary XOR gates requires 4 clauses to encode, while each of the binary AND and OR gates requires 3 clauses. Encoding a binary 32-bit adder circuit therefore requires 32×4 variables (in addition to the 3×32 variables for x, y, and s) and $32 \times (2 \times 4 + 3 \times 3) = 544$ clauses [49]. In this way, all the operations in a hash function can be described in clauses. And a problem of finding a preimage of a hash function can be reduced to finding a solution for the CNF of all the clauses.

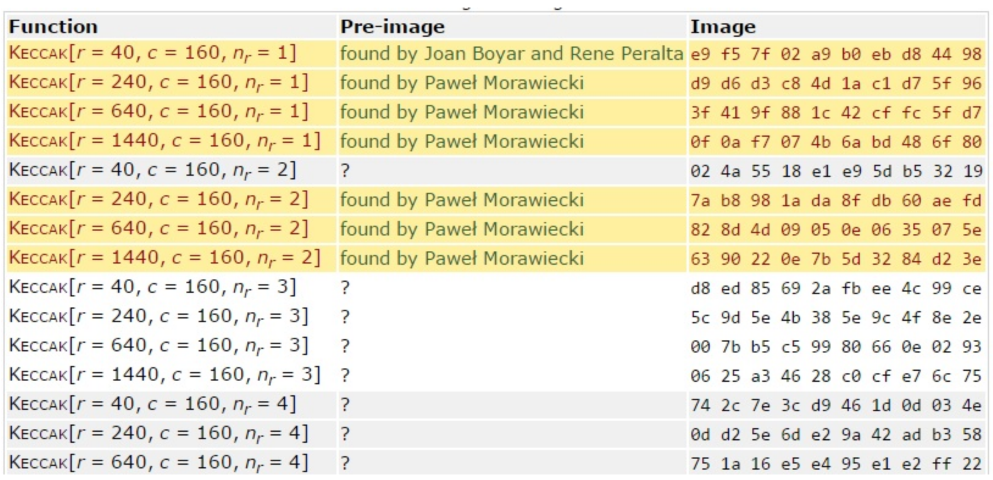
Morawiecki and Srebrny [45] developed CryptLogVer for generating CNF of a hash function along with a given preimage which is passed to the SAT solver PrecoSAT [13].

Input parameters				Attack times [secs]	
Function	Number of rounds	Message size [bits]	Hash size [bits]	SAT-solver attack	Exhaustive search
Keccak[1024,576]	3	24	1024	2^{0}	2^{1}
Keccak[1024,576]	3	32	1024	$2^{3,3}$	2^{9}
Keccak[1024,576]	3	40	1024	$2^{10,8}$	2^{17}
Keccak[120,80]	3	24	80	$2^{2,5}$	$2^{-2,9}$
Keccak[120,80]	3	32	80	$2^{5,7}$	2^{5}
Keccak[120,80]	3	40	80	$2^{15,7}$	2^{13}
Keccak[24,26]	4	24	24	$2^{12,1}$	$2^{-13,5}$
Keccak[24,26]	5	24	24	$2^{12,8}$	$2^{-13,1}$

Fig. 37. Preimage attacks: SAT-based attacks vs. exhaustive search. [45]: The experiments were carried out on a 4-core Intel Xeon 2.5 GHz which was a part of Grid'5000 system. [33]

Finally, Morawiecki and Srebrny [45] had found a preimage for the 3-round Keccak-f[1600] with 40 unknown message bits with about time complexity 2^{34}, which is faster than the generic attack complexity 2^{40}. However, this attack cannot be applied to preimage attacks on reduced-round Keccak-224, because they only considered a specific situation that a message size is only 40-bit, which is much smaller than the bit-rate size 1152 of Keccak-224.

Fig. 38 shows current status of preimage challenges [5]. According to Fig. 38, there is no practical attack finding preimages for more than two-round version of Keccak.

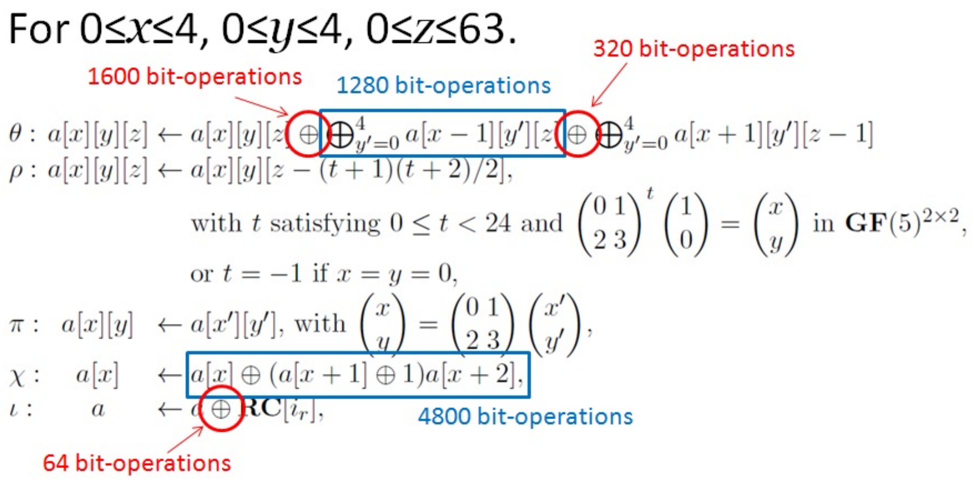


Function	Pre-image	Image
KECCAK[r = 40, c = 160, n_r = 1]	found by Joan Boyar and Rene Peralta	e9 f5 7f 02 a9 b0 eb d8 44 98
KECCAK[r = 240, c = 160, n_r = 1]	found by Paweł Morawiecki	d9 d6 d3 c8 4d 1a c1 d7 5f 96
KECCAK[r = 640, c = 160, n_r = 1]	found by Paweł Morawiecki	3f 41 9f 88 1c 42 cf fc 5f d7
KECCAK[r = 1440, c = 160, n_r = 1]	found by Paweł Morawiecki	0f 0a f7 07 4b 6a bd 48 6f 80
KECCAK[r = 40, c = 160, n_r = 2]	?	02 4a 55 18 e1 e9 5d b5 32 19
KECCAK[r = 240, c = 160, n_r = 2]	found by Paweł Morawiecki	7a b8 98 1a da 8f db 60 ae fd
KECCAK[r = 640, c = 160, n_r = 2]	found by Paweł Morawiecki	82 8d 4d 09 05 0e 06 35 07 5e
KECCAK[r = 1440, c = 160, n_r = 2]	found by Paweł Morawiecki	63 90 22 0e 7b 5d 32 84 d2 3e
KECCAK[r = 40, c = 160, n_r = 3]	?	d8 ed 85 69 2a fb ee 4c 99 ce
KECCAK[r = 240, c = 160, n_r = 3]	?	5c 9d 5e 4b 38 5e 9c 4f 8e 2e
KECCAK[r = 640, c = 160, n_r = 3]	?	00 7b b5 c5 99 80 66 0e 02 93
KECCAK[r = 1440, c = 160, n_r = 3]	?	06 25 a3 46 28 c0 cf e7 6c 75
KECCAK[r = 40, c = 160, n_r = 4]	?	74 2c 7e 3c d9 46 1d 0d 03 4e
KECCAK[r = 240, c = 160, n_r = 4]	?	0d d2 5e 6d e2 9a 42 ad b3 58
KECCAK[r = 640, c = 160, n_r = 4]	?	75 1a 16 e5 e4 95 e1 e2 ff 22

Fig. 38. Pre-image challenges and status [5]

Best Theoretical-but-Marginal Preimage and Second-preimage Attack on SHA3-224 The best theoretical-but-marginal preimage attack on SHA3-224 is the attack [19] on the 7-round version of SHA3-224, which requires time complexity $2^{218.11}$ and memory complexity $2^{180.12}$. Since the same attack technique is applied to all other variants of SHA3 family, we firstly describe its attack idea using the slides presented by [19] at the SHA-3 workshop in 2014. Note that this preimage attack can be also considered as a second-preimage attack.

The attack's complexity is going to be measured in terms of number of bit-operations such as $\oplus$ and $\wedge$, and so on. Fig. 39 shows that the number of bit-operations of each round of KECCAK-p is, in total, at least 8064 (=1600+1280+320+4800+64) bit-operations to compute one round.



For $0 \le x \le 4$, $0 \le y \le 4$, $0 \le z \le 63$.

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

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

$$\text{with } t \text{ satisfying } 0 \le t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ in } \mathbf{GF}(5)^{2\times 2},$$

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

$$\pi: a[x][y] \leftarrow a[x'][y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix},$$

$$\chi: a[x] \leftarrow \boxed{a[x] \oplus (a[x+1] \oplus 1) a[x+2]},$$

$$\iota: a \leftarrow a \oplus \mathbf{RC}[i_r],$$

Fig. 39. Number of Bit-operations of each Round of KECCAK-p: In total, at least 8064 (=1600+1280+320+4800+64) bit-operations are required to compute one round.

Fig. 40 shows that general preimage attack complexity for Keccak-o, where o is the hash output size requires $r \times 8064 \times 2^o$ bit-operations to find its preimage with high probability.

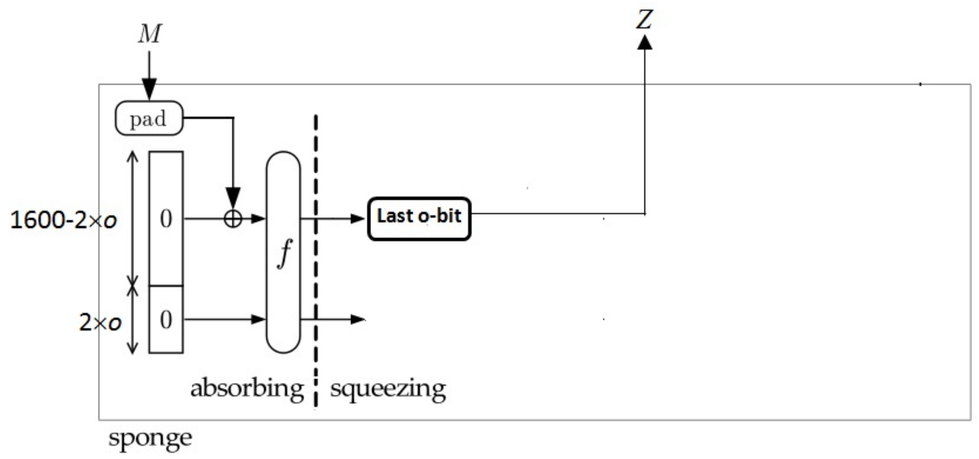


Fig. 40. General Preimage Attack Complexity for Keccak-o, where o is the hash output size: So, given a o-bit hash value Z, we need $r \times 8064 \times 2^o$ bit-operations to find its preimage with high probability.

Polynomial Enumeration *(used by Dinur and Shamir [26]):* Given a boolean function f_i ($1 \leq i \leq b$) with n-bit input and degree d, where f_i is the i-th output bit of f, polynomial enumeration algorithm is a way of constructing the truth table of f_i by the following two steps:

1. Compute coefficients of f_i with time complexity $\sum_{j=0}^{d}(2^j \times \binom{n}{j})$.
2. Construct the truth table of f_i using the fast Moebius transformation with time complexity $n \times 2^{n-1}$.

The Fast Moebius Transformation*:* This transforms the coefficient array of a boolean function to its truth table array. For example, see Fig. 41.

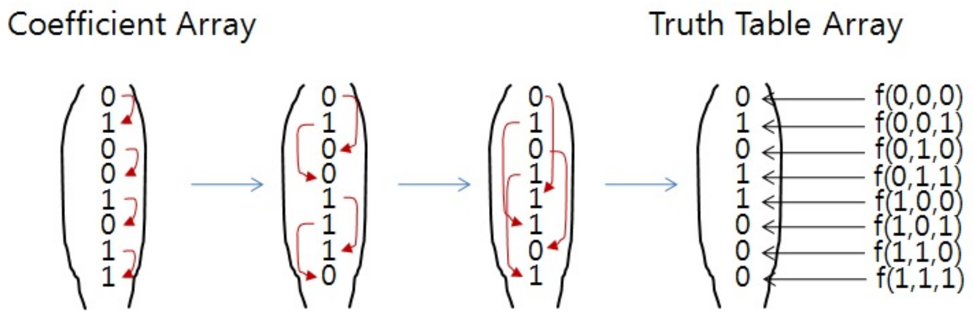


Fig. 41. The Fast Moebius Transformation in case that $f(x_1, x_2, x_3) = x_1 \oplus x_1x_2x_3 \oplus x_1x_2 \oplus x_3$: Generally, for n variables, $n \times 2^{n-1}$ 1-bit XOR operations are required.

Preimage Attack on a hash function H using Polynomial Enumeration *(used by Dinur and Shamir [26]):* Given a o-bit hash output Z, the attack consists of the following two steps:

1. By polynomial enumeration algorithm, efficiently find messages M's which partially match over b bits of the given o-bit hash value, where b is a value less than o.
2. if there is M s.t. $H(M) = Z$, then return M else goes to Step 1.

Improving Polynomial Enumeration *(by Bernstein [12]):* Given a boolean function f_i ($1 \leq i \leq b$) with n-bit input and degree d, where f_i is the i-th output bit of f, polynomial enumeration algorithm is a way of constructing the truth table of f_i by the following two steps:

1. Compute coefficients of f_i with time complexity $\sum_{j=0}^{d}(j \times \binom{n}{j})$.
2. Construct the truth table of f_i using the fast Moebius transformation with time complexity $n \times 2^{n-1}$.

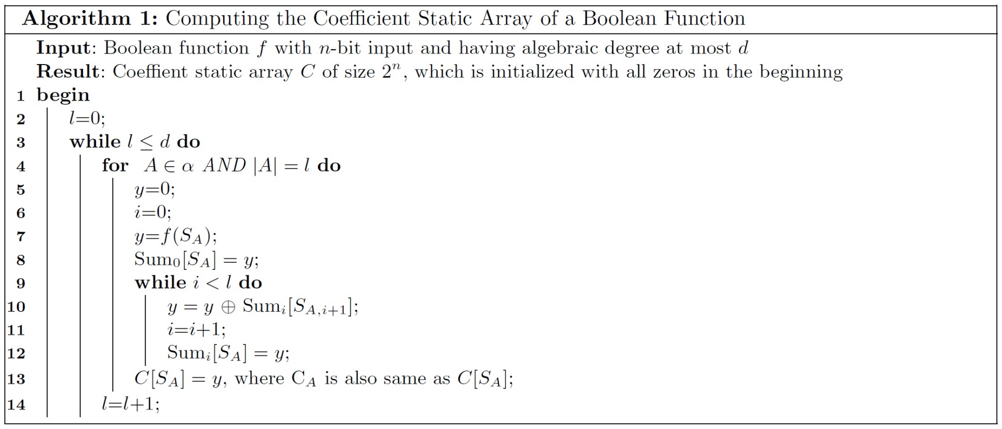
Bernstein [12] suggested an idea of storing the partial sums and reusing them to speed-up the polynomial enumeration algorithm. So, he could improve $\sum_{j=0}^{d}(2^j \times \binom{n}{j})$ into $\sum_{j=0}^{d}(j \times \binom{n}{j})$. But, Bernstein did not define the algorithm in detail. Later, by [19], as shown in Fig. **??** His idea was formally described in [19].

Application to 6, 7, 8 rounds of Keccak-512 *(by Bernstein [12, 21]):* The followings are results:

– 6 rounds: 2^{176} bits of memory give a workload reduction by a factor 50 (6 bits)
– 7 rounds: 2^{320} bits of memory give a workload reduction by a factor 37 (5 bits)
– 8 rounds: 2^{508} bits of memory give a workload reduction by a factor 1.4 (half a bit)

Further Improving Bernstein's results *(by Chang et al. [19]):* The results of [19] can be summarized as follows:

– Bernstein only described the idea of improving Step 1 complexity. However, overall time and memory complexity of his attack is not clear.
  - Result 1: Based on Bernstein's idea, [19] made Algorithm 1 (in Fig. 42) for generating the coefficient array of a boolean function with detailed time and memory complexity.
  - Result 2: We provide a general preimage attack methodology on hash functions using Result 1 and meet-in-the-middle-matching technique.
  - Result 3: Using Result 2, as an example, we further improve Bernstein's result upto 9 rounds of Keccak-512.



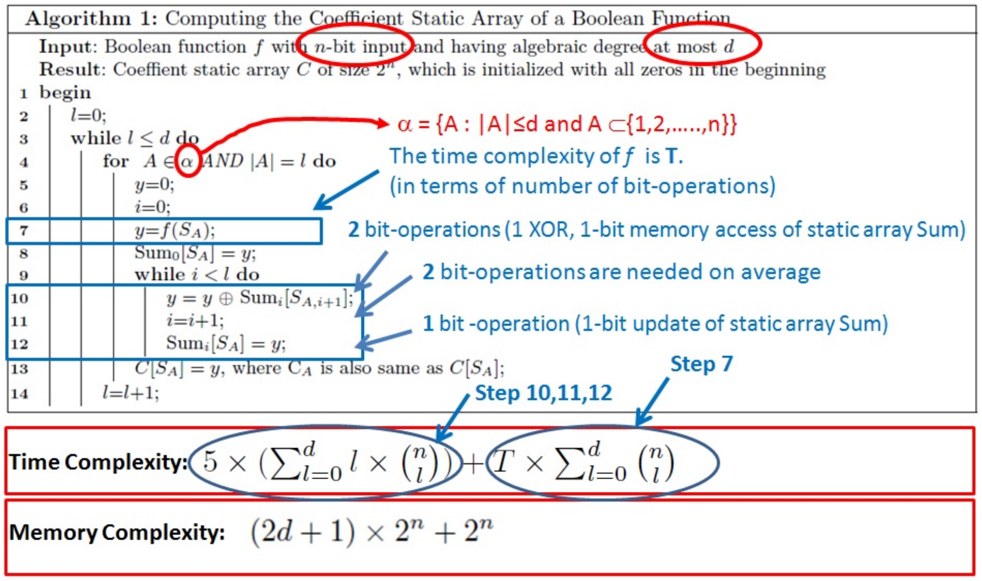
Algorithm 1: Computing the Coefficient Static Array of a Boolean Function

Input: Boolean function f with n-bit input and having algebraic degree at most d

Result: Coeffient static array C of size 2^n, which is initialized with all zeros in the beginning

```
1  begin
2      $l$=0;
3      while $l \leq d$ do
4          for $A \in \alpha$ $AND$ $|A| = l$ do
5              $y$=0;
6              $i$=0;
7              $y$=$f(S_A)$;
8              $\mathrm{Sum}_0[S_A] = y$;
9              while $i < l$ do
10                 $y = y \oplus \mathrm{Sum}_i[S_{A,i+1}]$;
11                 $i$=$i$+1;
12                 $\mathrm{Sum}_i[S_A] = y$;
13             $C[S_A] = y$, where $C_A$ is also same as $C[S_A]$;
14         $l$=$l$+1;
```

Fig. 42. Algorithm 1 for Computing the Coefficient Static Array of a Boolean Function (Result 1)[19]: This is followed by Bernstein's idea in [12]



Algorithm 1: Computing the Coefficient Static Array of a Boolean Function

Input: Boolean function f with n-bit input and having algebraic degree at most d

Result: Coeffient static array C of size 2^n, which is initialized with all zeros in the beginning

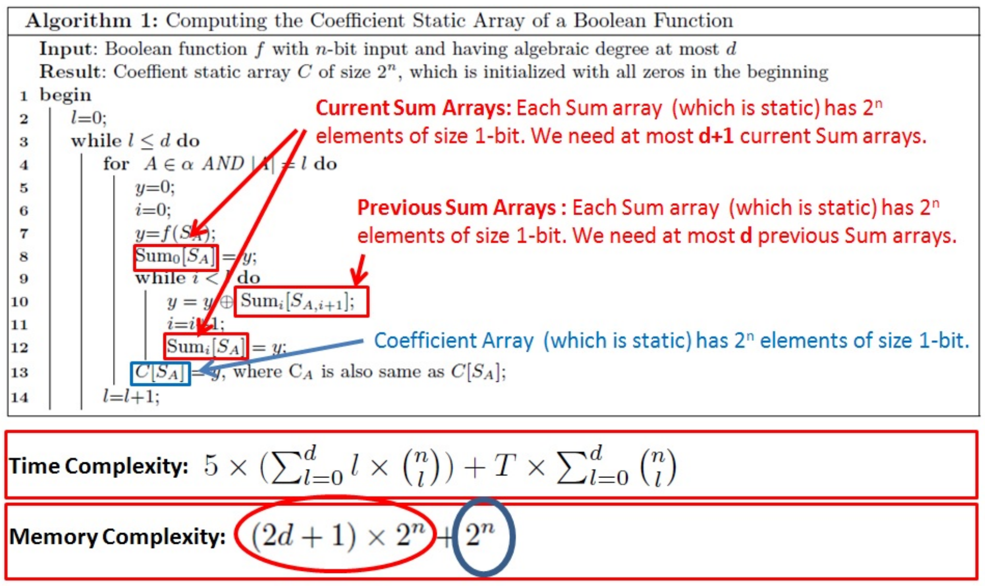
```
1  begin
2      $l$=0;
3      while $l \leq d$ do
4          for $A \in \alpha$ $AND$ $|A| = l$ do
5              $y$=0;
6              $i$=0;
7              $y$=$f(S_A)$;
8              $\mathrm{Sum}_0[S_A] = y$;
9              while $i < l$ do
10                 $y = y \oplus \mathrm{Sum}_i[S_{A,i+1}]$;
11                 $i$=$i$+1;
12                 $\mathrm{Sum}_i[S_A] = y$;
13             $C[S_A] = y$, where $C_A$ is also same as $C[S_A]$;
14         $l$=$l$+1;
```

$\alpha = \{A : |A| \leq d$ and $A \subset \{1,2,\ldots,n\}\}$
The time complexity of f is **T**.
(in terms of number of bit-operations)
2 bit-operations (1 XOR, 1-bit memory access of static array Sum)
2 bit-operations are needed on average
1 bit -operation (1-bit update of static array Sum)
Step 7
Step 10,11,12

Time Complexity: $5 \times \left(\sum_{l=0}^{d} l \times \binom{n}{l}\right) + T \times \sum_{l=0}^{d} \binom{n}{l}$

Memory Complexity: $(2d+1) \times 2^n + 2^n$

Fig. 43. Time Complexity of Algorithm 1 for Computing the Coefficient Static Array of a Boolean Function (Result 1) [19]: This is followed by Bernstein's idea in [12]



```
Algorithm 1: Computing the Coefficient Static Array of a Boolean Function
  Input: Boolean function $f$ with $n$-bit input and having algebraic degree at most $d$
  Result: Coeffient static array $C$ of size $2^n$, which is initialized with all zeros in the beginning
begin
    $l$=0;
    while $l \leq d$ do
        for $A \in \alpha$ AND $|A| = l$ do
            $y$=0;
            $i$=0;
            $y=f(S_A)$;
            $\mathrm{Sum}_0[S_A] = y$;
            while $i < l$ do
                $y = y \oplus \mathrm{Sum}_i[S_{A,i+1}]$;
                $i=i+1$;
                $\mathrm{Sum}_i[S_A] = y$;
            $C[S_A] = y$, where $C_A$ is also same as $C[S_A]$;
        $l=l+1$;
```

Current Sum Arrays: Each Sum array (which is static) has 2^n elements of size 1-bit. We need at most **d+1** current Sum arrays.

Previous Sum Arrays : Each Sum array (which is static) has 2^n elements of size 1-bit. We need at most **d** previous Sum arrays.

Coefficient Array (which is static) has 2^n elements of size 1-bit.

Time Complexity: $5 \times (\sum_{l=0}^{d} l \times \binom{n}{l}) + T \times \sum_{l=0}^{d} \binom{n}{l}$

Memory Complexity: $(2d+1) \times 2^n + 2^n$

Fig. 44. Memory Complexity of Algorithm 1 for Computing the Coefficient Static Array of a Boolean Function (Result 1) [19]: This is followed by Bernstein's idea in [12]

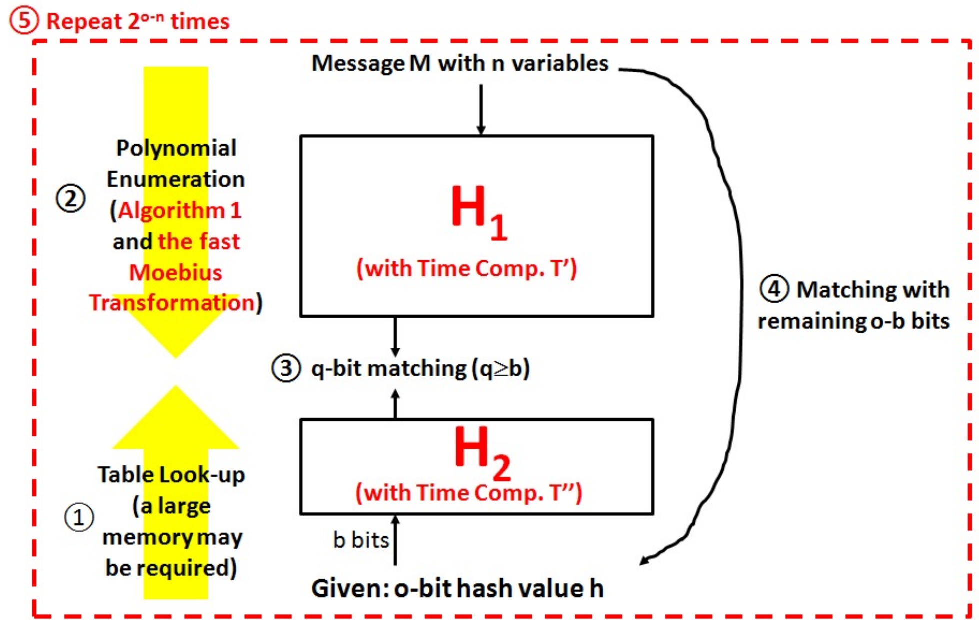
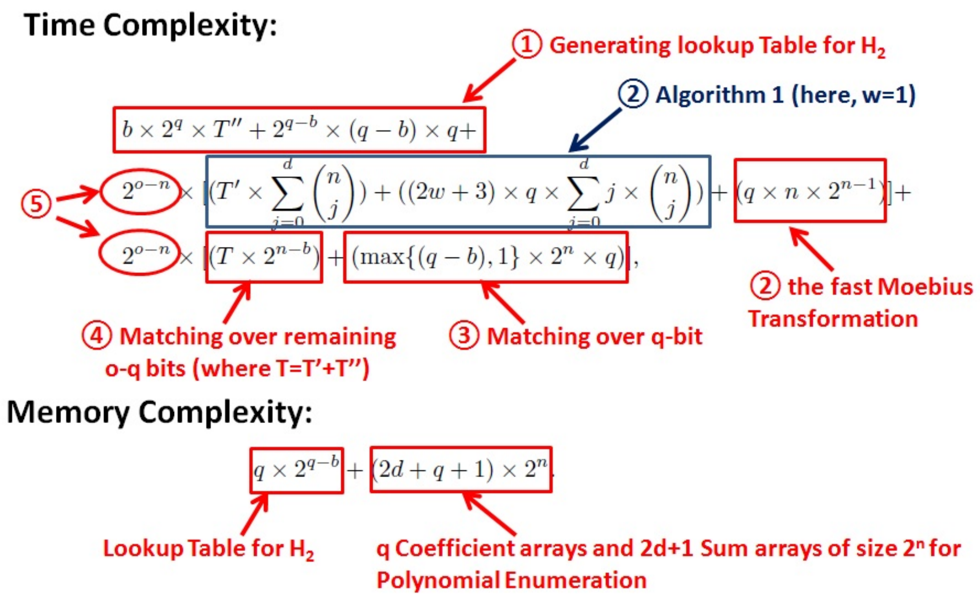

⑤ Repeat 2^{o-n} times
Message M with n variables
② Polynomial Enumeration (Algorithm 1 and the fast Moebius Transformation)
H_1 (with Time Comp. T')
③ q-bit matching (q≥b)
④ Matching with remaining o-b bits
① Table Look-up (a large memory may be required)
H_2 (with Time Comp. T'')
b bits
Given: o-bit hash value h

Fig. 45. General Preimage Attack on $H = H_2 \circ H_1$ (Result 2) [19]


Time Complexity:
① Generating lookup Table for H_2
② Algorithm 1 (here, w=1)
⑤
② the fast Moebius Transformation
④ Matching over remaining o-q bits (where T=T'+T'')
③ Matching over q-bit
Memory Complexity:
Lookup Table for H_2
q Coefficient arrays and 2d+1 Sum arrays of size 2^n for Polynomial Enumeration

$$b \times 2^q \times T'' + 2^{q-b} \times (q-b) \times q +$$
$$2^{o-n} \times [(T' \times \sum_{j=0}^{d} \binom{n}{j}) + ((2w+3) \times q \times \sum_{j=0}^{d} j \times \binom{n}{j})) + (q \times n \times 2^{n-1})] +$$
$$2^{o-n} \times [(T \times 2^{n-b}) + (\max\{(q-b),1\} \times 2^n \times q)],$$

$$q \times 2^{q-b} + (2d+q+1) \times 2^n.$$

Fig. 46. Complexity of General Preimage Attack on $H = H_2 \circ H_1$ (Result 2) [19]

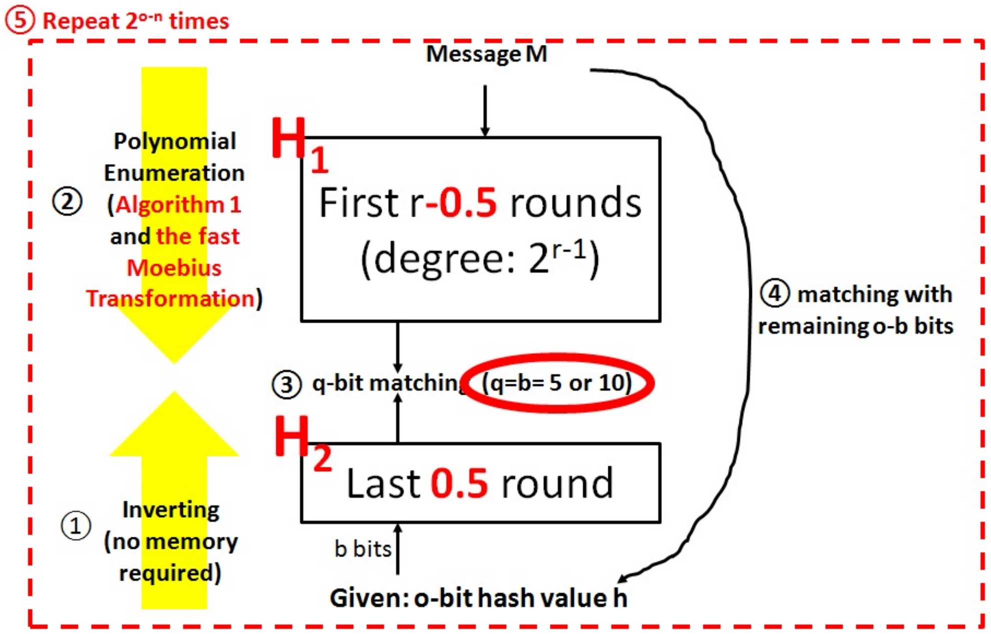
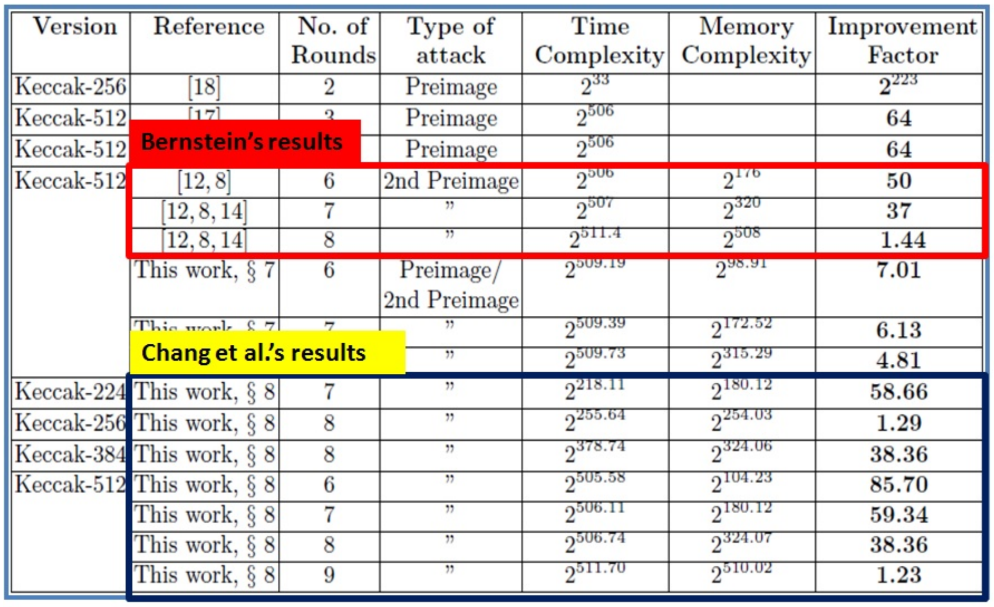
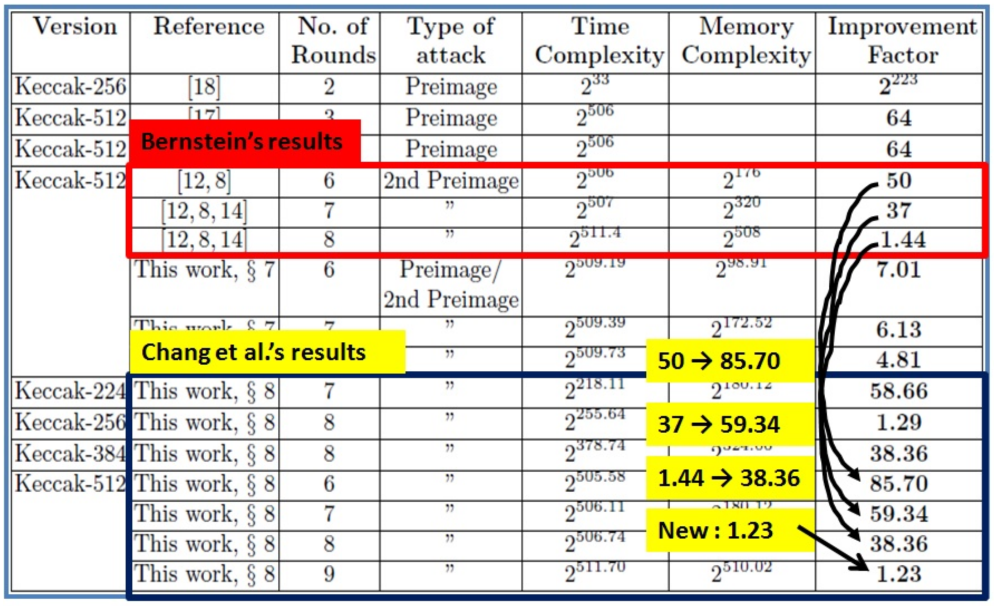

⑤ Repeat 2^{o-n} times
Message M
H_1
First r-0.5 rounds (degree: 2^{r-1})
② Polynomial Enumeration (Algorithm 1 and the fast Moebius Transformation)
④ matching with remaining o-b bits
③ q-bit matching (q=b= 5 or 10)
H_2
Last 0.5 round
① Inverting (no memory required)
b bits
Given: o-bit hash value h

Fig. 47. Application of General Preimage Attack to KECCAK (Result 3) [19]


Bernstein's results
Chang et al.'s results

Version	Reference	No. of Rounds	Type of attack	Time Complexity	Memory Complexity	Improvement Factor
Keccak-256	[18]	2	Preimage	2^{33}		2^{223}
Keccak-512	[17]	3	Preimage	2^{506}		64
Keccak-512	[illegible]	[illegible]	Preimage	2^{506}		64
Keccak-512	[12, 8]	6	2nd Preimage	2^{506}	2^{176}	50
	[12, 8, 14]	7	"	2^{507}	2^{320}	37
	[12, 8, 14]	8	"	$2^{511.4}$	2^{508}	1.44
	This work, § 7	6	Preimage/ 2nd Preimage	$2^{509.19}$	$2^{98.91}$	7.01
	This work, § 7	7	"	$2^{509.39}$	$2^{172.52}$	6.13
	[illegible]	[illegible]	"	$2^{509.73}$	$2^{315.29}$	4.81
Keccak-224	This work, § 8	7	"	$2^{218.11}$	$2^{180.12}$	58.66
Keccak-256	This work, § 8	8	"	$2^{255.64}$	$2^{254.03}$	1.29
Keccak-384	This work, § 8	8	"	$2^{378.74}$	$2^{324.06}$	38.36
Keccak-512	This work, § 8	6	"	$2^{505.58}$	$2^{104.23}$	85.70
	This work, § 8	7	"	$2^{506.11}$	$2^{180.12}$	59.34
	This work, § 8	8	"	$2^{506.74}$	$2^{324.07}$	38.36
	This work, § 8	9	"	$2^{511.70}$	$2^{510.02}$	1.23

Fig. 48. 1st and 2nd Preimage Attacks on 6, 7, 8, 9 rounds of Keccak (Result 3) [19]


Bernstein's results
Chang et al.'s results
50 → 85.70
37 → 59.34
1.44 → 38.36
New : 1.23

Version	Reference	No. of Rounds	Type of attack	Time Complexity	Memory Complexity	Improvement Factor
Keccak-256	[18]	2	Preimage	2^{33}		2^{223}
Keccak-512	[17]	3	Preimage	2^{506}		64
Keccak-512	[illegible]	[illegible]	Preimage	2^{506}		64
Keccak-512	[12, 8]	6	2nd Preimage	2^{506}	2^{176}	50
	[12, 8, 14]	7	"	2^{507}	2^{320}	37
	[12, 8, 14]	8	"	$2^{511.4}$	2^{508}	1.44
	This work, § 7	6	Preimage/ 2nd Preimage	$2^{509.19}$	$2^{98.91}$	7.01
	This work, § 7	7	"	$2^{509.39}$	$2^{172.52}$	6.13
	[illegible]	[illegible]	"	$2^{509.73}$	[illegible]	4.81
Keccak-224	This work, § 8	7	"	$2^{218.11}$	[illegible]	58.66
Keccak-256	This work, § 8	8	"	$2^{255.64}$	[illegible]	1.29
Keccak-384	This work, § 8	8	"	$2^{378.74}$	[illegible]	38.36
Keccak-512	This work, § 8	6	"	$2^{505.58}$	[illegible]	85.70
	This work, § 8	7	"	$2^{506.11}$	[illegible]	59.34
	This work, § 8	8	"	$2^{506.74}$	[illegible]	38.36
	This work, § 8	9	"	$2^{511.70}$	$2^{510.02}$	1.23

Fig. 49. Comparison between Bernstein's results and Chang *et al.*'s results (Result 3) [19]

Best Practical Distinguishing Attack on SHA3-224 The following zero-sum structures were investigated to find distinguishers of the underlying permutation of SHA3-224 [39, 2]. According to Fig. 51, zero-sum structures for 9-round and 10-round versions of the underlying permutation of SHA3-224 can be found with practical time complexity $2^{29.83}$ and $2^{59.67}$, respectively.

Definition 2. *[15] Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$. A zero-sum for F of size K is a subset $\{x_1, \cdots, x_K\} \subset \mathbb{F}_2^n$ of elements which sum to zero and for which the corresponding images by F also sum to zero, i.e.,*

$$\sum_{i=1}^{K} x_i = \sum_{i=1}^{K} F(x_i) = 0$$

The above zero-sum structures can be easily found using the concept of higher-order derivatives of a function. Higher-order derivatives of a function and its related property were well studied in [40].

Definition 3. *[40] Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$. For any $a \in \mathbb{F}_2^n$ the derivative of F with respect to a is the function $D_aF(x) = F(x+a) + F(x)$. For any k-dimensional subspace V of $\mathbb{F}_2^n$ and for any basis of V, $\{a_1, \cdots, a_k\}$, the k-th order derivative of F with respect to V is the function defined by*

$$D_V F(x) = D_{a_1} D_{a_2} \cdots D_{a_k} F(x) = \sum_{v \in V} F(x+v), \forall x \in \mathbb{F}_2^n$$

And for every subspace V of dimension $(\geq deg(F) + 1)$,

$$D_V F(x) = \sum_{v \in V} F(x+v) = 0, \forall x \in \mathbb{F}_2^n$$

For example, as shown in Fig. 50, the degree of the forward direction of 6-round Keccak-p is at most 60 and the degree of the backward direction of 4-round Keccak-p is at most 27. This is because the algebraic normal form (ANF) of the round function has degree 2 and the ANF of the inverse of round function has degree 3. Therefore, once we choose a subspace of dimension 61 in the middle, we can find zero-sum structures by an application of higher-order derivative concept.

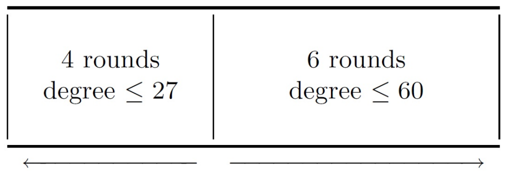


Fig. 50. Structure of the 10-round Distinguisher for Keccak-p [2].

Fig. 51 clearly shows how much time complexity is required to find zero-sum distinguishers for some numbers of rounds of Keccak-p [2]. According to Fig. 51, zero-sum structures for 9-round and 10-round versions of Keccak-p can be found with practical time complexity $2^{29.83}$ and $2^{59.67}$.

type of	backwards		forwards		total	
bounds	#rounds	degree $\leq$	#rounds	degree $\leq$	#rounds	complexity
1 slice	2	9	4	9	6	$2^{8.41}$
1 lane	3	9	3	8	6	$2^{9.00}$
1 lane	3	9	4	15	7	$2^{14.77}$
1 slice	3	17	5	17	8	$2^{16.58}$
1 lane	4	27	5	30	9	$2^{29.83}$
1 lane	4	27	6	60	10	$2^{59.67}$
1-per-row	5	81	6	60	11	$2^{59.54}$
1-per-row	5	81	7	128	12	$2^{127.73}$
1-per-row	6	243	7	128	13	$2^{242.88}$
1-per-row	6	243	8	256	14	$2^{255.77}$
2-per-row	6	243	9	512	15	$2^{511.68}$
anywhere	6	729	10	1024	16	$2^{1023.88}$

Fig. 51. Parameters of the best distinguisher for various total number of rounds. The columns "type of bounds" gives the type of bounds used, either with respect to bits in a same slice (only if order ≤ 25), in one lane (only if order ≤ 25), at most two per row (3^{n-1}, 2^n, only with order ≤ 640), or anywhere in the state (3^n, 2^n). For consistency, we give the normalized complexity in terms of evaluations of the permutation (assuming that computing a round has the same complexity as computing an inverse round), e.g., the complexity given is $2^{10} \times 26 = 2^{8.42}$ for the attack on the first line, since it requires 2^{10} evaluations of the two rounds of the inverse permutations, out of six rounds in total in the permutation considered [2].

Best Theoretical-but-Marginal Distinguishing Attacks on Keccak-256 A zero-sum structure of the full 24-round KECCAK-p can be found with time complexity 2^{1575} [29], which is still far beyond the general collision attack complexity $2^{n/2}$ and the general preimage attack complexity 2^n, where n is a hash output size (n=224 for KECCAK-224, n=256 for KECCAK-256, n=384 for KECCAK-384, n=512 for KECCAK-512). This marginal zero-sum distingushing attack on the full 24-round KECCAK-p can be applied to any of KECCAK-224, KECCAK-256, KECCAK-384, and KECCAK-512. Let us take a look at this marginal attack.

forward		backward	
# rounds	bound on deg(R^r)	# rounds	bound on deg(R^{-r})
1	2	1	3
2	4	2	9
3	8	3	27
4	16	4	81
5	32	5	243
6	64	6	729
7	128	7	1309
8	256	8	1503
9	512	9	1567
10	1024	10	1589
11	1408	11	1596
12	1536	12	1598
13	1578	13	1599
14	1592		
15	1597		
16	1599		

Fig. 52. Upper bounds on the degree of several rounds of KECCAK-p and of its inverse [15].

As shown in Fig. 52, [15] provided upper bounds on the degree of several rounds of KECCAK-p and of its inverse. Soon later [28] improved the results of [15] as shown in Fig. 53.

round	Old	New
1	3	3
2	9	9
3	27	27
4	81	81
5	243	243
6	729	729
7	1309	1309
8	1503	1454
9	1567	1532
10	1589	1566
11	1596	1583
12	1598	1591
13	1599	1595
14		1597
15		1598
16		1599

Fig. 53. Comparison of the Upper Bounds on Deg(R^{-r}) [28]: Old means the results of [15] and New means the improved results of [28].

By the same authors [29], the Upper Bounds on Deg(R^{-r}) were further improved as shown in Fig. 54.

Round	Old	Improved Bound
1	3	3
2	9	9
3	27	27
4	81	81
5	243	243
6	729	729
7	1309	1164
8	1503	1382
9	1567	1491
10	1589	1545
11	1596	1572
12	1598	1586
13	1599	1593
14	1599	1596
15	1599	1598
16	1599	1599

Fig. 54. The Further Improved Upper Bounds on Deg(R^{-r}) [29]: Old means the results of [15].

From Fig. 52, Fig. 53, and Fig. 54, the degree of the backward direction of 11-round KECCAK-p is upper-bounded by 1572 and the degree of the forward direction of 12-round KECCAK-p is upper-bounded by 1536.

[15, 29, 28] considered how to extend one more round without increasing the degrees. The main question is described in Fig. 55. How can we connect the forward direction and backward direction?

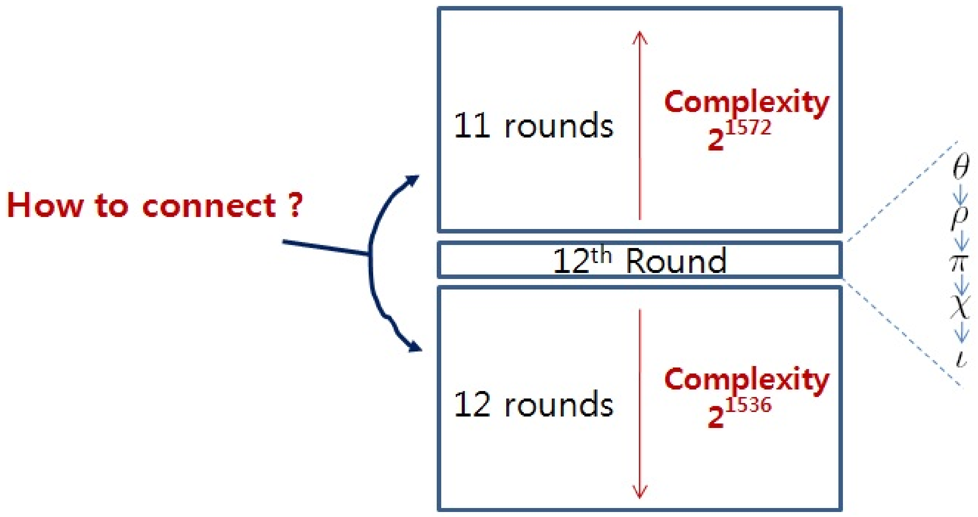


Fig. 55. The Zero-sum Distinguishing Attack Strategy on the full 24-round KECCAK-p [29].

Fig. 56 shows that the starting point will be the input states of χ function at 12th Round.

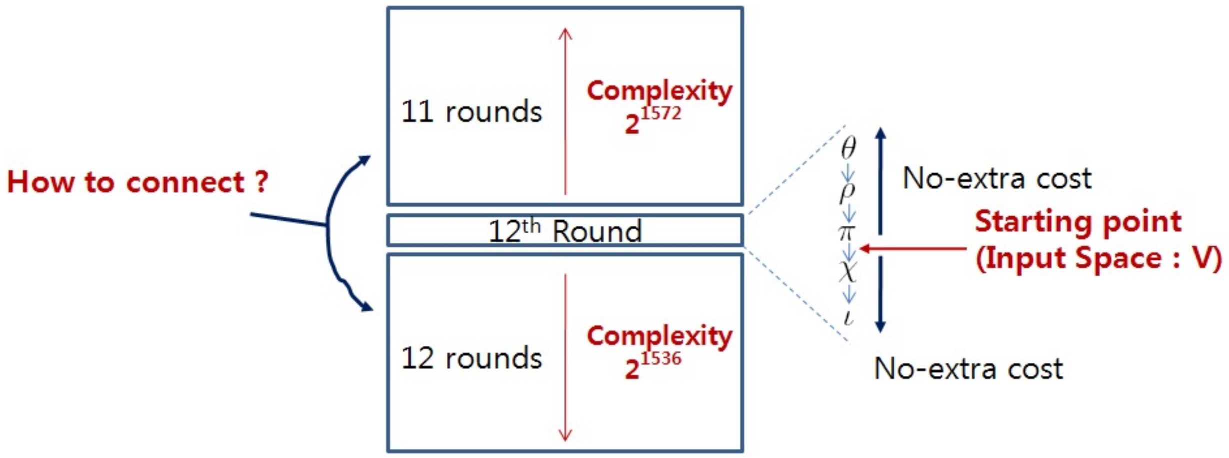


Fig. 56. The Zero-sum Distinguishing Attack Strategy on the full 24-round KECCAK-p from the Middle of 12th Round [29].

As we know, the non-linear operation, repetations of the 5-bit Sbox operation in parallel, of χ function is done over each 5-bit row independently in parellel. Therefore, since the 5-bit Sbox is bijective, if we consider all possible 5-bit values, which forms 5-dimensional subspace, for a particular 5-bit row, we can expect that all 32 possible outputs, which will again form 5-dimensional subspace, will come after the Sbox operation. So, if we construct a subspace V in a way that for each row, either all 32 5-bit values appear in the row or only one particular 5-bit value appears in the row. Once we form such subspace V with dimension t for a t, $\{\chi(x)|x \in V\}$ also will be a subspace with dimension t. Therefore, we can extend the second half round of 12th round (χ and ι functions) without any extra cost for the zero-sum attack. Moreover, the first half (θ, ρ, and π) can be inverted without increasing any degree because they are linear functions.

Since the degree of the backward direction of 11-round underlying permutation of SHA3-224 is upper-bounded by 1572 and the degree of the forward direction of 12-round underlying permuation of SHA3-224 is upper-bounded by 1536 and the Sbox input size is 5-bit, we have to choose a subspace V with dimention 1575 (= the minimum value which is a multiple of 5 and bigger than 1572 and 1536.) in a way that for each row, either all 32 5-bit values appear in the row or only one particular 5-bit value appears in the row. Therefore, a zero-sum structure of the full 24-round underlying permutation of SHA3-224 can be found with time complexity 2^{1575} [29].

However, compared to the security strength of SHA3-224, which has 224-bit security strength, the time complexity 2^{1575} is too high. Let's find the best attack with complexity less than 2^{224}. According to Fig. 51, 7-round in the forward direction has degree of 128 and 5-round in the backward direction has degree of 81. Therefore, we can find a zero-sum distinguisher for 12-round underlying permutation of SHA3-224 with about complexity 2^{128}, which is less than 2^{224}. As we learned from Fig. 55, we can extend one more round by considering all the possible inputs of each active row. According to Fig. 52, we can find a zero-sum distinguisher for 12-round underlying permutation of SHA3-224 with about complexity 2^{130}, which is less than 2^{224}.

3.4 SHA3-256

As shown in Sect. 1.3, SHA3-256 is defined as follows:

$$\text{SHA3-256}(M) = \textsc{Keccak}[512](M||01, 256)$$

However, since all known cryptanalytic results were done only for $\textsc{Keccak}[512](M, 256)$ without the two-bit 10 padding, the results cannot be directly applied to SHA3-256. Nonetheless, the results on $\textsc{Keccak}[512](M, 256)$ clearly show the security strength of SHA3-256, because the attack techniques on $\textsc{Keccak}[512](M, 256)$ can be also applied to SHA3-256 with a small change by considering the two additional padding bits. Therefore, in this subsection, we focus on describing all known cryptanalytic results on $\textsc{Keccak}[512](M, 256)$. For simplicity, we will call $\textsc{Keccak}[512](M, 256)$ $\textsc{Keccak}\text{-}256(M)$.

Best Practical Collision-type Attack on Keccak-256 The best practical collision attack on KECCAK-256 is the attack on the four round version of KECCAK-256, which is described in [22]. They even provided a collision pair which was found within a few minutes on a single PC. In this subsection, we will describe their attack idea step by step. Note that a round function R of KECCAK-256 consists of five steps, θ, ρ, π, χ, and ι.

In the same way of the 4-round practical collision attack on KECCAK-224 in Sect. 3.3, [22] provided the following collision of Fig. 58 on the 4-round KECCAK-256 using the 3-round characteristic of Fig. 57, which is same as one of Fig. 32.

```
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2656F26B|357C4789AF3-6AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E2676F26B|357C4789AF3-6AF1|78D3526BC4A74C4D|
|26978AF134CB835E|AF224C4D78366789|C4DAE35E265EF26B|357C4789AF3-4AF1|78D3526BC6A74C4D|
|26978AF134CB835E|AF226C4D78366789|C4DAE35E2656F26B|35FC4789AF3-6AF1|78D3526BC6A74C4D|

|----------------|----------------|---------------1|-------4--------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|-------4--------|----8-----------|
|----------------|----------------|---------------1|----------------|----8-----------|

|----------------|----------------|----------------|--8-------------|2---------------|
|4---------------|----------------|----------------|----------------|2---------------|
|----------------|----------------|----------------|--8-------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|4---------------|----------------|----------------|----------------|----------------|

|----------------|----------------|----------------|----------------|----------------|
|-----------8----|-----------2----|----------------|----------------|----------------|
|----------------|----------------|-----------1----|----------------|-----------1----|
|---------1------|-------4--------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

Fig. 57. A 3-round characteristic leading to a collision Keccak-256 with probability 2^{-24} [22]. Note that the order of lanes is from left to right.

```
M1=

C4F31C32 4C59AE6D 5D19F0F4 25C4E44B D8853032 8D5E12F2 BB6E6EE2 27C33B1E 6C091058 EB9002D5
3BA4A06F 4A0CC7F1 CCB55E51 8D0DD983 2B0A0843 9B21D3B0 53679075 526DDED2 48294844 6FF4ED2C
1ACE2C15 471C1DC7 D4098568 F1EBF639 EAF7B257 09FDAE87 688878E6 4875EB30 C9C32D80 3C9E6FCB
3C2ABCFA E6A4807B 2AB281B8 812332B3

M2=

A4D30EF7 80BB8F69 90C048DF EB7213B9 A6650424 3A65F63E 8C268881 B651B81F AADAFA3C EE2CA5C3
DB78EAC2 C8EAE779 442F9C35 3221E287 B3017A5A 90790712 1B1C8BDC E08B10A8 9A9D25CA 1BE7AAAC
4E2F3E9C 73717DAD 5566015A A198CFB9 5A1CA8C2 A0E3348A AE6C0BB1 3980F9E4 A4FA8B91 6E81A989
89A9BCAA E12BF1F1 30EF9595 812E8B45

Output=

61FB1891 F326B6D5 24DD94DF 73274984 05DA9A1D 3FD359B9 78B8393B F2E7990B
```

Fig. 58. A collision for 4-round Keccak-256 [22]. Note that the order of lanes is from left to right.

Best Efficient-but-theoretical Collision-type Attack on Keccak-256 The best efficient-but-theoretical collision-type attack on Keccak-256 is a collision attack on the 5-round reduced version of Keccak-256 with complexity 2^{115} using generalized internal differences [23]. As shown in [23], this same technique can be also applied to show a practical collision attack on the 3-round version of Keccak-384 and an efficient-but-theoretical collision attack on the 4-round version of Keccak-384 with complexity 2^{147} and a practical collision attack on the 3-round version of Keccak-512, which will be described in Sect. 3.5 and 3.6 for evaluating SHA3-384 and SHA3-512.

As said in [23], this new attack approach using internal differentials is a special type of subset cryptanalysis, which tries to track the statistical evolution of a certain set of values. Here, an internal differential means the statistical evolution of the differences between parts of only one plaintext. This is different from the standard differential attacks in a sense that the attack follows the statistical evolution of the difference between two difference plaintexts.

The new attack using internal differentials on Keccak can be also called "Squeeze attacks" [23].

Squeeze Attacks [23]. Assume that a hash function maps a set S of possible inputs into a set D of possible outputs. In order to find a collision for the hash function with a high probability, by the birthday attack complexity, we need to compute hash output for every input in a set $S' \subseteq S$, where $|S'| = \sqrt{|D|}$. Now, consider a variant of this attack. Instead of considering a collision over the set D, let's consider a collision over a set $D' \subseteq D$ and let D' contain a fraction q of the points in D. Assume that the probability of picking an input in S' whose output is in D' is p. Then, in order to find a collision, we need to compute hash outputs for $\sqrt{q|D|}/p$ inputs in S'. In order to guarantee a better performance of this new variant approach, we need an assumption that $p^2 > q$. For an example, see Fig. 59.

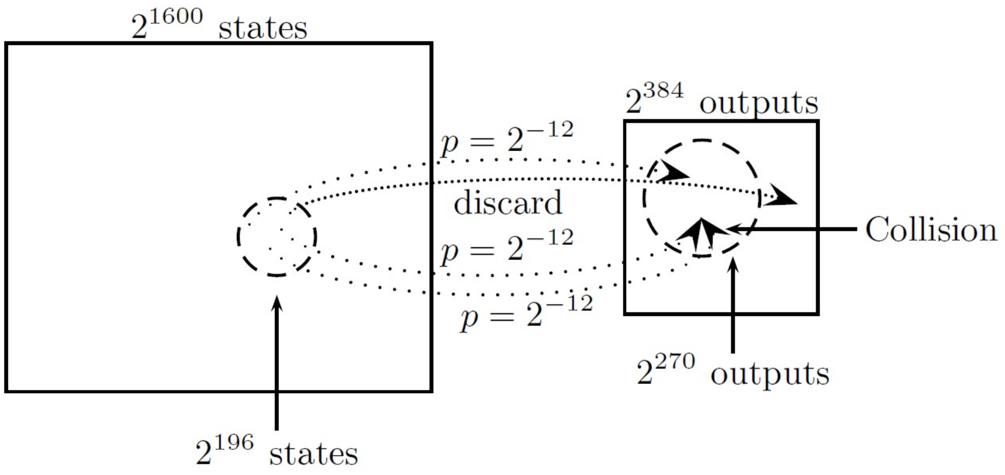


Fig. 59. A Squeeze Attack with $|S| = 2^{1600}$; $|S'| = 2^{196}$; $|D| = 2^{384}$; $|D'| = 2^{270}$; $p = 2^{-12}$ [23].

Internal Difference Sets [23]. Each internal state of Keccak family can be described with 1600-bit. Let $a[x][y][z]$ represent each bit of 1600-bit by the values of x, y, z, where $x, y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_{64}$. Given a rotation index $i \in \{1, 2, 4, 8, 16, 32\}$, let us consider a set of symmetric states $a[x][y][z]$ such that $a[x][y][z] = a[x][y][z + i]$. Given a 1600-bit symmetric state a, let $a_1 = \pi(a)$, $a_2 = \rho(a)$, $a_3 = \theta(a)$, and $a_4 = \chi(a)$. Then, it is easy to check that a_1, a_2, a_3, and a_4 will be also symmetric states. For $i = 16$, a symmetric state $a[x][y][z]$ is composed of four repetitions of slices 0-15 (See an example in Fig. 60.). Each such sequence of slices (0-15, 16-31, 32-47, 48-63) is called a *consecutive slice set* or *CSS* in short. However, after applying ι, the symmetric property would not be preserved. However, the influence by ι is small so the state remains close to being symmetric.

```
|169D169D169D169D|A965A965A965A965|3EC73EC73EC73EC7|9025902590259025|C264C264C264C264|
|A34BA34BA34BA34B|0F330F330F330F33|4902490249024902|3D683D683D683D68|613D613D613D613D|
|C684C684C684C684|B368B368B368B368|589B589B589B589B|5F335F335F335F33|E27AE27AE27AE27A|
|22E822E822E822E8|3D583D583D583D58|B37AB37AB37AB37A|1047104710471047|D525D525D525D525|
|60F360F360F360F3|C3E4C3E4C3E4C3E4|37FA37FA37FA37FA|8193819381938193|69BA69BA69BA69BA|
```

Fig. 60. A symmetric state with i=16 [23]: The state is described as a matrix of 5×5 lanes of 64 bits, ordered from left to right, where each lane is given in hexadecimal using the little-endian format. Each lane of the state consists of 4 repetitions of a 16-bit word [23].

‘*Internal differences*’ are defined in this way [23]: Internal differences measure how close the state is to a symmetric state. In case that $i = 16$, the internal differences can be obtained by computing the XOR differences between the first consecutive slice set, and each of the three other ones, denoted by the triplet $(\Delta_1, \Delta_2, \Delta_3)$. So, an internal difference in Keccak can be defined by the set of states with a fixed value of $(\Delta_1, \Delta_2, \Delta_3)$. Especially, when all 4 CSS’s are equal, the internal difference is called a *zero internal difference.*

‘*Internal difference set*’ is defined in this way [23]: An internal difference set $\{v + w : w$ is symmetric$\}$ is defined by using a *single representative state* v and adding to it *all* the fully symmetric states. Given a rotation index i, the internal difference set is denoted by $[i, v]$.

The Evolution of Internal Difference Through Keccak’s Permutation [23]. It is clear that the zero internal difference passes with probability 1 all the four operations π, ρ, θ, χ. However, the addition of a constant by the operation ι affects the characteristic. Since each operation except for χ is an affine mapping, the followings hold:

$$\begin{aligned}\pi([i,v]) &= [i,\pi(v)],\\ \rho([i,v]) &= [i,\rho(v)],\\ \theta([i,v]) &= [i,\theta(v)],\\ \iota([i,v]) &= [i,\iota(v)].\end{aligned}$$

The Evolution of Internal Difference Through χ [23]. Since χ is a nonlinear mapping, $\chi([i,v]) \neq [i, chi(v)]$. However, when a state of an internal difference which is not symmetric enters the χ function, it is possible to consider the possible outcomes in terms of “distance” from the zero internal difference. For a detailed explanation, the authors defines a *rotated row set*, $\{r(y,z), r(y,z+16), r(y,z+32), r(y,z+48)\}$ (in case that $i = 16$), where a row $r(y,z)$ is in the first CSS and other three rows are in the other CSS’s (see Fig. 61.). We say that a rotated set $\{r(y,z), r(y,z+16), r(y,z+32), r(y,z+48)\}$ (in case that $i = 16$) is *inactive* only when $r(y,z) = r(y,z+16) = r(y,z+32) = r(y,z+48)$.

```
|0001000100010001|0001000100010001|0001000100010001|0001000100010001|0001000100010001|
```

Fig. 61. A rotated row set [23]: The first five lanes of a state in which the 20 bits of the first rotated row set for i=16 are set to 1. The lanes are ordered from left to right, where each lane is given in hexadecimal using the little-endian format. [23].

In [23], a rotated row set is called *sparse* when it contains at most two distinct input value (See Fig. 62.). When a rotated row set is sparse, there is only a single input difference between the two groups of Sboxes. So, we can use the difference distribution table also in the more general case of $i \neq 32$, when the rotated row set is sparse.

```
|0001000100010001|0001000000010000|0000000000000000|0001000100010001|0000000100000001|
```

Fig. 62. A sparse rotated row set [23]: The first five lanes (given in the format of Example of Fig. 61) of a state in an internal difference in which the first rotated row set is sparse for i=16. The (binary) value of r(0,0) and r(0,32) is 10011, while the value of r(0,16) and r(0,48) is 11010. In this example, the internal difference fixes the difference of 01001 between the two groups of rows. The value of the other rows is zero. [23].

Choosing the Value of the Rotation Index [23]. Assume that a hash output would be one of 2^d restricted values with probability p. Then, we can find a collision pair, whose hash output is a restricted value, with complexity $p^{-1} \cdot 2^{d/2}$. Let n be a hash output size of Keccak-n. Since the padding of Keccak is the 10*1 padding, we should be able to generate messages (1600-2n-2)-bit M's such that $M||11||0^{2n} \in [i, \mathbf{0}]$ for a fixed $i \in \{1, 2, 4, 8, 16, 32\}$. Since the padding "11", we can choose $(i-2)$ bit-values for the lane containing the "11" padding. When r is the bitrate (or message block, in case of Keccak-n, the bitrate is 1600-2n), we can choose values only for $r/64$ lanes and there is i-bit freedom for such lanes except for the lane containing the padding "11". Therefore, we can generate $2^{r\cdot(i/64)-2}$ initial states which are symmetric. Hence, we have to ensure that $2^{r\cdot(i/64)-2} \geq p^{-1} \cdot 2^{d/2}$.

Extending Internal Differential Characteristics [23]. [23] proposes a way of extending 1.5 rounds more from an internal differential characteristic by aggregating internal output differences of χ function in the second last round, using *affine subspaces.*

In [10], it was observed that since the algebraic degree of the Keccak Sbox is only 2, all the possible output differences of the Sboxes form an affine subspace. For example, a two-dimensional affine subspace has a form of $\{\Delta, \Delta \oplus \Delta_1, \Delta \oplus \Delta_2, \Delta \oplus \Delta_1 \oplus \Delta_2\}$ $(= \Delta \oplus \{0, \Delta_1, \Delta_2, \Delta_1 \oplus \Delta_2\})$. More precisely, as shown in Fig. 63, a given non-zero difference Δ_{in}, the number of possible output differences Δ_{out} is 4 or 8 or 16, and the set of such possible output differences, $\{\Delta_{out} | \exists v \text{ s.t. } \chi(v) \oplus (v \oplus \Delta_{in}) = \Delta_{out}\}$, will be two- or three- or four-dimensional affine subspace. For example, when $\Delta_{in} = 10$, the set of possible output differences will be $\{01, 09, 11, 19\}$ from Fig. 63, which can be described by $\Delta \oplus \{0, \Delta_1, \Delta_2, \Delta_1 \oplus \Delta_2\}$), where $\Delta = 01$, $\Delta_1 = 08$, $\Delta_2 = 10$. Moreover, each vector of the two-dimensional affine subspace, $\Delta \oplus \{0, \Delta_1, \Delta_2, \Delta_1 \oplus \Delta_2\}$, is described as $\Delta \oplus c_1\Delta_1 \oplus c_2\Delta_2$ using two variables $c_1 (= 0 \text{ or } 1)$ and $c_2 (= 0 \text{ or } 1)$. In case of a three-dimensional subspace, it can be described by three variables.

Δ_{out} / Δ_{in}	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
00	32	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
01	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-
02	-	-	8	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	8	-	-	-	-	-	-	-	-	-	-	-	-
03	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-
04	-	-	-	-	8	8	8	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
05	-	-	-	-	4	-	4	-	-	-	-	-	4	-	4	-	-	-	-	-	-	4	-	4	-	-	-	-	-	4	-	4
06	-	-	-	-	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	4	4	4	4	-	-	-	-	-	-	-	-
07	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2
08	-	-	-	-	-	-	-	-	8	-	8	-	8	-	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
09	-	4	-	4	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	-	4	-	4
0A	-	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	-	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4
0B	-	4	4	-	-	-	-	-	-	-	-	-	-	4	4	-	-	4	4	-	-	-	-	-	-	-	-	-	-	4	4	-
0C	-	-	-	-	-	-	-	-	4	4	4	4	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0D	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-
0E	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2
0F	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	-	-	-	-
10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	8	-	-	-	8	-	-	-	8	-	-	-	8	-	-	-
11	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-	-	4	-	-
12	-	-	4	4	-	-	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	4	4
13	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	-	-	-	-	4	4	4	4	-	-	-	-	4	4
15	-	4	-	-	-	-	-	4	-	4	-	-	-	-	-	4	4	-	-	-	-	-	4	-	4	-	-	-	-	-	4	-
16	-	-	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	4	4	-	-
17	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-	-	-	2	2	2	2	-	-
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	4	-	4	-	4	-	4	-
19	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2
1A	-	-	-	-	-	-	-	-	4	-	-	4	4	-	-	4	4	-	-	4	4	-	-	4	-	-	-	-	-	-	-	-
1B	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-	-	2	2	-
1C	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1D	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	2	-	2	-	2	-	2	-	2	-	2	-	2	-	2	-
1E	-	-	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-
1F	-	2	2	-	2	-	-	2	2	-	-	2	-	2	2	-	2	-	-	2	-	2	2	-	-	2	2	-	2	-	-	2

Fig. 63. The differential distribution table of the χ when viewed as Sbox. The first bit of a row is viewed as the least significant bit. Given input difference Δ_{in} and output difference Δ_{out} the number in the table shows the size of the solution set $\{v|\chi(v) \oplus (v \oplus \Delta_{in}) = \Delta_{out}\}$. Differences are in hex number. [27].

So, we can describe the evolution of internal differences through χ as follows:

$\chi([i, \hat{u}]) = [i, \hat{\mathbf{u}}]$, where $\hat{u}$ is a specific internal input difference and $\hat{\mathbf{u}}$ is the symbolic form to describe all the possible internal output differences using allocated variables from each rotated row set.

In order to calculate the number of possible final hash outputs after the truncation, in case of KECCAK-224 and KECCAK-256, we have to know the number of possible first 320-bit values of an 1600-bit input state of χ at the last round. In case of KECCAK-384 and KECCAK-512, we have to know the number of possible first 640-bit values of an 1600-bit input state of χ at the last round. Especially, for $n = 384$, a further improvement is possible [23]: for $n = 384$, we can know the first

320 bits of the hash output from the first 320-bit input of χ at the last round and the remaining last 64-bit of the hash output is defined by the next 192-bit input (= input value for next three lanes, where each lane size is 64-bit) because the last 64-bit output is determined by three lanes by the definition of χ. Therefore, in case of KECCAK-384, we can know the number of possible final hash outputs from the number of possible first 512(=320+192) bits of χ's input at the last round.

Now, let's count the number of possible input states of χ in the last round. Let t be the weight of the 1600-bit internal input difference of χ function in the second last round. For example, see Fig. 64. In this case, $t = 12$ to allocate 24 variables, say t'=24, and extend the characteristic beyond χ function, because every row of input internal differences of χ is at most 1, and according to Fig. 63, there are only four possible output difference when its weight of input difference has 1. So, each non-zero difference bit of 11 non-zero difference bits allocates two variables. Therefore, in total, 24 variables are allocated.

In cases of KECCAK-224 and KECCAK-256, we only consider the possible number of values for the first five input lanes of χ at the last round, which is $32^i (= 2^{5i})$, since there are i rotated row sets and there 32 possible cases for each set. And assume that t' variables are allocated at the second last round. Therefore, the total number of possible values for the final hash output will be $2^{t'+5i}$.

In cases of KECCAK-384, the number of possible first 320-bit of hash outputs is influenced by the number of possible values for the first five input lanes of χ at the last round, which is $32^i (= 2^{5i})$. And the number of possible remaining 64-bit of the hash outputs will be $min(2^{64}, 2^{3i})$, because we are considering only three lanes, not five lanes. And assume that t' variables are allocated at the second last round. Therefore, the total number of possible values for the final hash output will be $2^{t'+5i} \cdot min(2^{64}, 2^{3i})$.

In cases of KECCAK-512, we need to consider the possible number of values for the first ten input lanes of χ at the last round, which is $(32^i)^2 (= 2^{10i})$. And assume that t' variables are allocated at the second last round. Therefore, the total number of possible values for the final hash output will be $2^{t'+10i}$.

COLLISION ATTACK ON 5-ROUND KECCAK-256 [23]. As we learned already, we can see the following evolution of an internal difference set $[i, v]$ through π, ρ, θ, ι, and χ as follows:

$$\begin{aligned}
\pi([i,v]) &= [i, \pi(v)] \text{ with probability 1} \\
\rho([i,v]) &= [i, \rho(v)] \text{ with probability 1} \\
\theta([i,v]) &= [i, \theta(v)] \text{ with probability 1} \\
\iota([i,v]) &= [i, \iota(v)] \text{ with probability 1} \\
\pi^{-1}([i,v]) &= [i, \pi^{-1}(v)] \text{ with probability 1} \\
\rho^{-1}([i,v]) &= [i, \rho^{-1}(v)] \text{ with probability 1} \\
\theta^{-1}([i,v]) &= [i, \theta^{-1}(v)] \text{ with probability 1} \\
\iota^{-1}([i,v]) &= [i, \iota^{-1}(v)] \text{ with probability 1} \\
\chi([i,v]) &= [i, v'] \text{ with probability } p \text{ whose value depends on the values of } v \text{ and } v'
\end{aligned}$$

Using the above evolution information of each function, as shown in Fig. 64, [23] found an 2.5-round internal differential characteristic with probability 2^{37} from the input difference of the second round R1 to the output difference of R3.5, where a rotation index value i is 32. This characteristic will be used to attack on the 5-round collision attack on KECCAK-256 with about complexity 2^{115}.

```
|BCE-1EDC--------|68A-49EE--------|4A6-2999--------|916-3-39--------|68A--D8---------|
|BCE-9E54--------|68A-49EA--------|4A6-2999--------|916-3-99--------|6CA--D8---------|
|FCE-9E54--------|68A-49EE--------|4A2-2999--------|916-3-99--------|68A--D8---------| R1
|BCE-9E54--------|68A-49EE--------|4A6-2999--------|916-3-99--------|68A--D8---------|
|BCE-9E54--------|68A-49EE--------|4A6-2B99--------|916-3-99--------|69A--D8---------|
```

$$\downarrow \mathrm{L}$$

```
|----8-88--------|----4-----------|-------2--------|----------------|------4---------|
|-------A--------|----4-----------|-------2--------|----------------|------4---------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \chi \quad (p = 2^{-21})$$

```
|----8-8A--------|----4-----------|-------2--------|----------------|------4---------|
|-------8--------|----4-----------|-------2--------|----------------|------4---------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \iota$$

```
|-------8--------|----4-----------|-------2--------|----------------|------4---------|
|-------8--------|----4-----------|-------2--------|----------------|------4---------|
|----------------|----------------|----------------|----------------|----------------| R2
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \mathrm{L}$$

```
|-------8--------|-4--------------|----------------|----------------|----------------|
|----------------|-4--------------|----------------|----------------|----------------|
|----8-----------|------8---------|----------------|----------------|----------------|
|-------2--------|------8---------|----------------|----------------|----------------|
|8---------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \chi \quad (p = 2^{-16})$$

```
|-------8--------|-4--------------|----------------|----------------|----------------|
|----------------|-4--------------|----------------|----------------|----------------|
|----8-----------|------8---------|----------------|----------------|----------------|
|------82--------|------8---------|----------------|----------------|----------------|
|8---------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \iota$$

```
|8---8-82--------|-4--------------|----------------|----------------|----------------|
|----------------|-4--------------|----------------|----------------|----------------|
|----8-----------|------8---------|----------------|----------------|----------------| R3
|------82--------|------8---------|----------------|----------------|----------------|
|8---------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \mathrm{L}$$

```
|8---8-82--------|------4---------|----------------|----------------|----------------|
|----------------|----------------|---4------------|--1-------------|----------------|
|-8--------------|----------------|----------------|----------------|---2------------| R3.5
|----------------|----------------|---2------------|----------------|----------------|
|----------------|----------------|----------------|---1-4----------|----------------|
```

Fig. 64. The 2.5 round internal differential characteristic with probability 2^{-37} used in the 5-round collision attack on Keccak-256 [23]: This characteristic has a rotation index value of $i = 32$.

In Fig. 64, let us consider the internal output difference of R3.5 which is shown in Fig. 65, where the internal output difference of R3.5 is also the internal input difference of χ in the fourth round. We can see that the difference weight of any two rows of every rotated row set for $i = 32$ is 1, because the values, 1,2,4,8, in Fig. 65 are described by four-bit 0001, 0010,0100, and 1000, whose hamming weights are 1 only. Then, the number of possible differences between rows in each rotated row set after χ function is exactly four. More precisely, according to the differential distribution table of the χ in Fig. 63, when $\Delta_{in} = 01$, Δ_{output} will be one of $\{01, 09, 11, 19\}$. When $\Delta_{in} = 02$, Δ_{output} will be one of $\{02, 03, 12, 13\}$. When $\Delta_{in} = 04$, Δ_{output} will be one of $\{04, 05, 06, 07\}$. When $\Delta_{in} = 08$, Δ_{output} will be one of $\{08, 0A, 0C, 0E\}$. When $\Delta_{in} = 10$, Δ_{output} will be one of $\{10, 14, 18, 1C\}$. Moreover, each of $\{01, 09, 11, 19\}$, $\{02, 03, 12, 13\}$, $\{04, 05, 06, 07\}$, $\{08, 0A, 0C, 0E\}$, and $\{10, 14, 18, 1C\}$ forms two-dimensional subspaces by two 0-1 variables.

Let $[i, v]$ be the internal difference set representing the difference set in Fig. 65. Since the weight of v has 12, the internal output difference set of χ function will be described by $[i, \mathbf{v}']$, where $\mathbf{v}'$ is represented by the 24 variables. Then, $[i, \mathbf{v}']$ will be evolved as follows:

$$
\begin{array}{l}
[i, \mathbf{v}'] \Rightarrow_{\iota} [i, \mathbf{v}'_1)](= [i, \iota(\mathbf{v}')]) \text{ at R3 with probability 1} \\
[i, \mathbf{v}'_1] \Rightarrow_{\theta} [i, \mathbf{v}'_2)](= [i, \theta(\mathbf{v}'_1)]) \text{ at R4 with probability 1} \\
[i, \mathbf{v}'_2] \Rightarrow_{\rho} [i, \mathbf{v}'_3)](= [i, \rho(\mathbf{v}'_2)]) \text{ at R4 with probability 1} \\
[i, \mathbf{v}'_3] \Rightarrow_{\pi} [i, \mathbf{v}'_4)](= [i, \pi(\mathbf{v}'_3)]) \text{ at R4 with probability 1}
\end{array}
$$

Finally, $[i, \mathbf{v}'_4)]$ will be the internal difference set of χ function in the last round R4. Now, our concern is the number of possible hash outputs. Especially, in case of Keccak-256, the values of hash output can be determined only by the first 320-bit of the input state of χ function. According to the internal differential characteristic of Fig. 64, we see that the input states of χ function will be in $[i, \mathbf{v}'_4)]$ with probability 2^{37} and the first 320-bit of the input state of χ is defined by combining 2^{5i} symmetric states and 2^{24} possible values for 24 variables. So, the number of possible first 320-bit of the input state of χ function will be $2^{5i} \cdot 2^{24}$. When $i = 32$, it will be 2^{184}.

```
|8---8-82--------|------4---------|----------------|----------------|----------------|
|----------------|----------------|---4------------|--1-------------|----------------|
|-8--------------|----------------|----------------|----------------|---2------------| R3.5
|----------------|----------------|---2------------|----------------|----------------|
|----------------|----------------|----------------|---1-4----------|----------------|
```

Fig. 65. The internal output difference of R3.5 described in Fig. 64 [23]: This characteristic difference has a rotation index value of $i = 32$.

Finally, we only need to show how to find 1086-bit M's such that $\mathrm{R}(M||11||0^{512})$ in the internal input difference of R1 in Fig. 66. This can be done in a similar way with the target difference algorithm (TDA), which was already explained in detail to attack on SHA3-224. In this case, we consider the internal difference, so a new algorithm, called *a target internal difference algorithm (TIDA)*, can be developed.

```
|BCE-1EDC--------|68A-49EE--------|4A6-2999--------|916-3-39--------|68A--D8---------|
|BCE-9E54--------|68A-49EA--------|4A6-2999--------|916-3-99--------|6CA--D8---------|
|FCE-9E54--------|68A-49EE--------|4A2-2999--------|916-3-99--------|68A--D8---------| R1
|BCE-9E54--------|68A-49EE--------|4A6-2999--------|916-3-99--------|68A--D8---------|
|BCE-9E54--------|68A-49EE--------|4A6-2B99--------|916-3-99--------|69A--D8---------|
```

Fig. 66. The internal input difference of R1 described in Fig. 64 [23]: This characteristic difference has a rotation index value of $i = 32$.

Therefore, according to *the sqeeze attack*, we can find a collision by the complexity $2^{37} \cdot 2^{184/2}$ (=2^{129}) which is higher than the birthday attack complexity. So, [23] further considered a message modification in TIDA to quickly find messages M's to follow the internal characteristic described in Fig. 64, which improves the 2^{129} time complexity of the basic attack by a multiplicative factor which is between 2^{14} and 2^{21}. Therefore, with about 2^{115} complexity, a collision for 5-round reduced version of Keccak-256 can be found.

Best Practical Preimage and Second-preimage Attack on SHA3-256 As shown from the evaluation part of SHA3-224, the current best practical preimage attack is on 3-round Keccak only with 40-bit messages [45]. But it is not clear how to utilize the result of [45] to find a preimage or second-preimage of 3-round SHA3-224.

Best Theoretical-but-Marginal Preimage and Second-preimage Attacks on SHA3-256 The best theoretical-but-marginal preimage attack on SHA3-256 is the attack [19] on the 8-round version of SHA3-256, which requires time complexity $2^{255.64}$ and memory complexity $2^{254.03}$ with improvement factor 1.29. For a detailed explanation, see the evaluation part of SHA3-224. Note that this preimage attack is also considered as the second-preimage attack.

Best Practical Distinguishing Attack on SHA3-256 A zero-sum structure of the 9- and 10-round underlying permutation of SHA3-256 can be found with practical time complexity $2^{29.83}$ and $2^{59.67}$, respectively, which was already explained in the evaluation part of SHA3-224.

Best Theoretical-but-Marginal Distinguishing Attacks on SHA3-256 A zero-sum structure of the full 24-round underlying permutation of SHA3-256 can be found with time complexity 2^{1575}, which was already explained in the evaluation part of SHA3-224. However, compared to the security strength of SHA3-256, which has 256-bit security strength, the time complexity 2^{1575} is too high. Let's find the best attack with complexity less than 2^{256}. According to Fig. 51, 7-round in the forward direction has degree of 128 and 6-round in the backward direction has degree of 243. Therefore, we can find a zero-sum distinguisher for 13-round underlying permutation of SHA3-256 with about complexity 2^{243}, which is less than 2^{256}. As we learned from Fig. 55, we can extend one more round by considering all the possible inputs of each active row. According to Fig. 52, we can find a zero-sum distinguisher for 13-round underlying permutation of SHA3-256 with about complexity 2^{245}, which is less than 2^{256}.

3.5 SHA3-384

As shown in Sect. 1.3, SHA3-384 is defined as follows:

$$\text{SHA3-384}(M) = \textsc{Keccak}[768](M||01, 384)$$

However, since all known cryptanalytic results were done only for KECCAK[768]$(M, 384)$ without the two-bit 10 padding, the results cannot be directly applied to SHA3-384. Nonetheless, the results on KECCAK[768]$(M, 384)$ clearly show the security strength of SHA3-384, because the attack techniques on KECCAK[768]$(M, 384)$ can be also applied to SHA3-384 with a small change by considering the two additional padding bits. Therefore, in this subsection, we focus on describing all known cryptanalytic results on KECCAK[768]$(M, 384)$. For simplicity, we will call KECCAK[768]$(M, 384)$ KECCAK-384(M).

```
|--8-------------|--4-------------|----------------|----------------|----------------|
|--8-------------|----------------|----------2-----|----------------|----------------|
|----------------|--4-------------|----------2-----|----------------|----------------| R0
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \mathrm{L}$$

```
|--8-------------|----------------|---------------1|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|--8-------------|---------8------|----------------|----------------|----------------|
|----------------|---------8------|---------------1|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \chi \quad (p = 2^{-12})$$

```
|--8------------1|----------------|---------------1|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|--8-------------|---------8------|----------------|----------------|----------------|
|----------------|---------8------|---------------1|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \iota$$

```
|--8-------------|----------------|---------------1|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|--8-------------|---------8------|----------------|----------------|----------------| R1
|----------------|---------8------|---------------1|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \mathrm{L}$$

```
|--8-------------|----------------|----------------|----------------|----------------|
|----------------|----------------|-4--------------|-------------1--|----------------|
|----------------|----------------|----------------|----------------|----------------| R1.5
|----------------|----------------|------2---------|------------8---|----------------|
|4---------------|----------------|----------------|----------------|----------------|
```

Fig. 67. The 1.5-round internal differential characteristic with probability 2^{-12} used in order to find collisions in 3-round Keccak-384 [23]:The characteristic has a rotation index value of i=4 [23].

Best Practical Collision-type Attack on Keccak-384 The best practical collision attack on KECCAK-384 is the attack on the three-round version of KECCAK-384, which is described in [23]. They even provided a collision pair which was found within a minute on a single PC. The collision can be found by the sqeeze attack using the internal differential characteristic in Fig. 67 for $i = 4$. As we can see, the final internal difference has six non-zero-bit internal difference in Fig. 67. So, in the same way for attacking KECCAK-256, we can calculate the bound on the output subset by allocating two 0-1 variables corresponding to each of six non-zero-bit internal differences (in total, there are 12 variable allocations). We need to know the first 512-bit of the input states of χ function in the last round (R2) to determine 384-bit hash outputs. So, we can bound the number of possible hash output by calculating the number of possible first 512-bit of the input states of χ function as the last round. Since $i = 4$, there are $2^{8 \cdot 4}$ symmetric states on the first 512-bit and 12 0-1 variables influence the 512-bit. Therefore, there are 2^{44} possible first 512-bit values which bound the number of possible hash outputs as 2^{44}.

Therefore, we only need to try 2^{22} initial states, which confirm the characteristic in Fig. 67, to find a collision. It was shown in [23] that for $n = 384$, we have r=832 and we can choose a sufficient number of $2^{-12} \cdot 2^{r \cdot (i/64)-2)} = 2^{38}$ messages that satisfy the constraints, where i=4. Finally, [23] found a collision pair, which is shown in Fig. 68, in less than a minute on a single PC.

```
M1=

FFFFFFFF FF7FFFFF BBBBBBBB BBFBBBBB 44444444 44444444 FFFFFFFF FFFFFFFF 99999999 99999999
44444444 44C44444 44444444 44444444 44644444 44444444 AAAAAAAA AAAAAAAA 66666666 66666666
44444444 44444444 DDDDDDDD DD9DDDDD DDFDDDDD DDDDDDDD

M2=

33333333 33B33333 55555555 55155555 AAAAAAAA AAAAAAAA 77777777 77777777 44444444 44444444
66666666 66E66666 EEEEEEEE EEEEEEEE 11311111 11111111 CCCCCCCC CCCCCCCC FFFFFFFF FFFFFFFF
11111111 11111111 99999999 99D99999 DDFDDDDD DDDDDDDD

Output=

99999991 11199999 4440C444 405C60DC 00000000 0C100010 777677F7 73F77767 3550F597 55D57155
66666664 66666666
```

Fig. 68. A collision in 3-round Keccak-512 [23]:The messages were found using Characteristic of Fig. 67 [23].

Best Efficient-but-theoretical Collision-type Attack on Keccak-384 The best efficient-but-theoretical collision-type attack on Keccak-384 is a collision attack on the 4-round reduced version of Keccak-384 with complexity 2^{147} using generalized internal differences [23]. This attack is one-round extension of the 3-round practical collision attack on Keccak-384. The internal differential characteristic used for the attack is the combination of Fig. 67 and Fig. 69.

As we can see, the final internal difference in Fig. 69 has a relatively high weight of 88. It is written in [23] that "there are 20 non-sparse rotated row sets, whose 4 Sboxes assume (exactly) 3 values. However, it is easy to verify that in 18 out of the 20 non-sparse rotated row sets, the possible values for the input state of χ function as the third round, are still contained in an affine subspace whose dimension is at most 4."

```
|--8-------------|----------------|----------------|----------------|----------------|
|----------------|----------------|-4--------------|-------------1--|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|------2---------|------------8---|----------------|
|4---------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \chi \quad (p = 2^{-12})$$

```
|--8-------------|----------------|----------------|----------------|----------------|
|----------------|----------------|-4--------------|-------------1--|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|------2---------|------------8---|----------------|
|4---------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \iota$$

```
|--8---------8-82|----------------|----------------|----------------|----------------|
|----------------|----------------|-4--------------|-------------1--|----------------|
|----------------|----------------|----------------|----------------|----------------| R2
|----------------|----------------|------2---------|------------8---|----------------|
|4---------------|----------------|----------------|----------------|----------------|
```

$$\downarrow \mathrm{L}$$

```
|8-8------------8|-8-82488---4----|-81-------------|-4----1----8----|--------6--12-4-|
|----------4----2|------18--481---|----------------|1-1-491----8----|------------2-4-|
|91----8----1-1-4|----------4-8--1|4----------8----|---------18--481|-----------1----| R2.5
|-----C--24-8----|----------------|---1-----2-2-922|--1-----81------|---4----2-------|
|------------4-8-|8--2----1-------|--C--24-8-------|----------------|22---1-----2-2-9|
```

Fig. 69. The 1-round extension of Characteristic of Fig. 67 used in the collision attack on 4-round Keccak-384 [23]:The characteristic has a rotation index value of i=16 (this applies to the full 2.5-round characteristic used in the 4-round attack) and probability 2^{-12}. The total probability of the full 2.5-round characteristic is 2^{-24} [23].

[23] exploited this observation, they alloted 140 0-1 variables for all but 2 rotated row sets to extend one more round in the same way with practical 3-round collision attack on Keccak-384. Finally, they could bound the number of hash output by $2^{10+128+132} = 2^{270}$ and the expected time complexity of the attack is bounded by $2^{12} \cdot 2^{270/2} = 2^{147}$, which is 2^{45} times faster than the 2^{192} complexity of the birthday attack.

Also, it is written in [23] that we have sufficiently many degrees of freedom to find a collision. We need to try about 2^{147} initial states. According to the calculation of the degrees of freedom, we have $2^{-12} \cdot 2^{r \cdot (i/64)-2} = 2^{196}$ states that satisfy the constraints, where $i = 4$ and r=832.

Best Practical Preimage and Second-preimage Attack on SHA3-384 As shown from the evaluation part of SHA3-224, the current best practical preimage attack is on 3-round Keccak only with 40-bit messages [45]. But it is not clear how to utilize the result of [45] to find a preimage or second-preimage of 3-round SHA3-384.

Best Theoretical-but-Marginal Preimage and Second-preimage Attacks on SHA3-384 The best theoretical-but-marginal preimage attack on SHA3-384 is the attack [19] on the 8-round version of SHA3-384, which requires time complexity $2^{378.74}$ and memory complexity $2^{324.06}$ with improvement factor 38.36. For a detailed explanation, see the evaluation part of SHA3-224. Note that this preimage attack is also considered as the second-preimage attack.

Best Practical Distinguishing Attack on SHA3-384 A zero-sum structure of the 9- and 10-round underlying permutation of SHA3-384 can be found with practical time complexity $2^{29.83}$ and

$2^{59.67}$, respectively, which was already explained in the evaluation part of SHA3-224.

Best Theoretical-but-Marginal Distinguishing Attacks on SHA3-384 A zero-sum structure of the full 24-round underlying permutation of SHA3-384 can be found with time complexity 2^{1575}, which was already explained in the evaluation part of SHA3-224. However, compared to the security strength of SHA3-384, which has 384-bit security strength, the time complexity 2^{1575} is too high. Let's find the best attack with complexity less than 2^{384}. According to Fig. 51, 8-round in the forward direction has degree of 256 and 6-round in the backward direction has degree of 243. Therefore, we can find a zero-sum distinguisher for 14-round underlying permutation of SHA3-384 with about complexity 2^{256}, which is less than 2^{384}. As we learned from Fig. 55, we can extend one more round by considering all the possible inputs of each active row. According to Fig. 52, we can find a zero-sum distinguisher for 14-round underlying permutation of SHA3-384 with about complexity 2^{260}, which is less than 2^{384}.

3.6 SHA3-512

As shown in Sect. 1.3, SHA3-512 is defined as follows:

$$\text{SHA3-512}(M) = \textsc{Keccak}[1024](M||01, 512)$$

However, since all known cryptanalytic results were done only for $\textsc{Keccak}[1024](M, 512)$ without the two-bit 10 padding, the results cannot be directly applied to SHA3-384. Nonetheless, the results on $\textsc{Keccak}[1024](M, 512)$ clearly show the security strength of SHA3-512, because the attack techniques on $\textsc{Keccak}[1024](M, 512)$ can be also applied to SHA3-512 with a small change by considering the two additional padding bits. Therefore, in this subsection, we focus on describing all known cryptanalytic results on $\textsc{Keccak}[1024](M, 512)$. For simplicity, we will call $\textsc{Keccak}[1024](M, 512)$ $\textsc{Keccak}\text{-}512(M)$.

Best Practical Collision-type Attack on Keccak-512 The best practical collision attack on Keccak-384 is the attack on the three-round version of Keccak-512, which is described in [23]. They even provided a collision pair which was found in less than an hour on a single PC. The collision can be found by the sqeeze attack using the internal differential characteristic in Fig. 70 for $i = 4$. As we can see, the final internal difference has 11 non-zero-bit internal difference in Fig. 70. So, in the same way for attacking Keccak-256, we can calculate the bound on the output subset by allocating two 0-1 variables corresponding to each of 11 non-zero-bit internal differences (in total, there are 22 variable allocations). We need to know the first 640-bit of the input states of χ function in the last round (R2) to determine 512-bit hash outputs. So, we can bound the number of possible hash output by calculating the number of possible first 640-bit of the input states of χ function as the last round. Since $i = 4$, there are $2^{2 \cdot 5 \cdot 4}$ symmetric states on the first 640-bit and 22 0-1 variables influence the 640-bit. Therefore, there are 2^{62} possible first 640-bit values which bound the number of possible hash outputs as 2^{62}.

```
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------| R0
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
                                         ↓ R
|---------------1|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------| R1
|----------------|----------------|----------------|----------------|----------------|
|----------------|----------------|----------------|----------------|----------------|
                                         ↓ L
|---------------1|----1-----------|----------------|----------------|------------8---|
|----------------|----------2-----|----------------|----2-----------|----------------|
|---------------2|----------------|----------------|-------------2--|----------------| R1.5
|--------1-------|----------------|-------------4--|----------------|----------------|
|----------------|----------------|-----1----------|----------------|---------------4|
```

Fig. 70. The 1.5-round internal differential characteristic with probability 1 used in order to find collisions in 3-round Keccak-512 [23]:The characteristic has a rotation index value of i=4 [23].

Therefore, we only need to try 2^{31} initial states, which confirm the characteristic in Fig. 70, to find a collision. It was shown in [23] that for $n = 512$, we have r=576 and we can choose a sufficient number of $2^{r\cdot(i/64)-2)} = 2^{34}$ messages that satisfy the constraints, where i=4. Finally, [23] found a collision pair, which is shown in Fig. 71, in less than an hour on a single PC.

```
M1=

88888888 88888888 66666666 66666666 AAAAAAAA AAAAAAAA 77777777 77777777 BBBBBBBB BBBBBBBB
BBBBBBBB BBBBBBBB 11111111 11111111 88888888 88888888 CCCCCCCC CCCCCCCC

M2=

AAAAAAAA AAAAAAAA 88888888 88888888 EEEEEEEE EEEEEEEE 99999999 99999999 99999999 99999999
99999999 99999999 88888888 88888888 CCCCCCCC CCCCCCCC CCCCCCCC CCCCCCCC

Output=

56BCC94B C4445644 D7655451 5DD96555 71FA7332 3BA30B23 958408C5 64407664 41805414 11190901
6ABAA8BA A8ABAEFA 7EF8AEEE ECCE68DC 4EC8ACEC DD5D5CCC
```

Fig. 71. A collision in 3-round Keccak-512 [23]:The messages were found using Characteristic of Fig. 70 [23].

Best Practical Preimage and Second-preimage Attacks on SHA3-512 As shown from the evaluation part of SHA3-224, the current best practical preimage attack is on 3-round Keccak only with 40-bit messages [45]. But it is not clear how to utilize the result of [45] to find a preimage or second-preimage of 3-round SHA3-512.

Best Theoretical-but-Marginal Preimage and Second-preimage Attacks on SHA3-512 The best theoretical-but-marginal preimage attack on SHA3-512 is the attack [19] on the 9-round version of SHA3-512, which requires time complexity $2^{511.70}$ and memory complexity $2^{510.02}$ with improvement factor 1.23. This detailed explanation was already described in the evaluation part for SHA3-224. Note that this preimage attack is also considered as the second-preimage attack.

Best Practical Distinguishing Attack on SHA3-512 A zero-sum structure of the 9- and 10-round underlying permutation of SHA3-512 can be found with practical time complexity $2^{29.83}$ and $2^{59.67}$, respectively, which was already explained for evaluation of SHA3-224.

Best Theoretical-but-Marginal Distinguishing Attacks on SHA3-512 A zero-sum structure of the full 24-round underlying permutation of SHA3-512 can be found with time complexity 2^{1575}, which was already explained in the evaluation part of SHA3-224. However, compared to the security strength of SHA3-512, which has 512-bit security strength, the time complexity 2^{1575} is too high. Let's find the best attack with complexity less than 2^{512}. According to Fig. 51, 8-round in the forward direction has degree of 256 and 6-round in the backward direction has degree of 243. Therefore, we can find a zero-sum distinguisher for 14-round underlying permutation of SHA3-512 with about complexity 2^{256}, which is less than 2^{512}. As we learned from Fig. 55, we can extend one more round by considering all the possible inputs of each active row. According to Fig. 52, we can find a zero-sum distinguisher for 14-round underlying permutation of SHA3-512 with about complexity 2^{260}, which is less than 2^{512}.

3.7 SHAKE128

Best Practical Collision Attack on SHAKE128. The capacity size of SHAKE128 is $c = 256$, which less than any other capacity size of the SHA3-224, SHA3-256, SHA3-384, and SHA3-512. Therefore, from the attack point of view, there is more freedom on messages in SHAKE128, compared to other SHA-3 functions. For a further domain separation purpose as explained in Sect. 4.3, SHAKE128 has "1111" as its inner 4-bit padding in addition to 10*1 padding as its outer padding. In Sect. 3.4, we learned that there is a practical collision attack on 4-round KECCAK-256 [22]. It looks that the attack procedure used for 4-round KECCAK-256 still works for SHAKE128 when the output size d of SHAKE128 is less than or equal to 320-bit, because there are zero-differences on the first 320-bit ((see the first five lanes on the first row of the last differential state of Fig. 32)) of the hash output as shown in Fig. 32.

Best Theoretical Collision Attack on SHAKE128. The capacity size of SHAKE128 is $c = 256$. For a further domain separation purpose as explained in Sect. 4.3, SHAKE128 has "1111" as its inner 4-bit padding in addition to 10*1 padding as its outer padding. Note that KECCAK-256 uses the 10*1 padding only. In Sect. 3.4, we learned that there is a theoretical collision attack on 5-round KECCAK-256 with complexity 2^{115} [23]. It looks that the attack procedure used for 5-round KECCAK-256 can still work for SHAKE128 when $231 \leq d \leq 320$, where d is the output size of SHAKE256. This is because 1) the attack complexity 2^{115} is meaningful only for $d > 230$, and 2), as explained in the evaluation part of KECCAK-256, the internal differential characteristic in Fig. 64 is 2^{-37} and the number of possible 320-bit hash outputs is 2^{184}, and by the help of the squeeze attack and message modification in TIDA, we can find a collision over 320-bit with about complexity 2^{115} in the same way.

Best Practical Preimage and Second-preimage Attack on SHAKE128. As shown from the evaluation part on KECCAK-224, the current best practical preimage attack is on 3-round KECCAK only with 40-bit messages [45]. But it is not clear how to utilize the result of [45] to find a preimage of 3-round SHAKE128. Note that this preimage attack is also considered as the second-preimage attack.

Best Theoretical Preimage and Second-preimage Attacks on SHAKE128. As described in [48], NIST claims that the security strength of SHAKE128 in terms of preimage resistance is min(d,128), where d is the output size, since we can only guarantee at most 128-bit security due to the capacity size $c = 256$. As shown in Fig. 48, it seems to be expected (but, it is required to verify) that the current best theoretical preimage attack would be an attack on 6 rounds of Keccak-128 with complexity less than the general preimage attack complexity 2^{128}, which can be applied to SHAKE128 with d=128. Note that this preimage attack is also considered as the second-preimage attack.

Best Practical Distinguishing Attack on SHAKE128. A zero-sum structure of the 9- and 10-round underlying permutation of SHAKE128 can be found with practical time complexity $2^{29.83}$ and $2^{59.67}$, respectively, which was already explained in the evaluation part of SHA3-224.

Best Theoretical-but-Marginal Distinguishing Attacks on SHAKE128. A zero-sum structure of the full 24-round underlying permutation of SHAKE128 can be found with time complexity 2^{1575}, which was already explained at the evaluation part of SHA3-224. However, compared to the security strength of SHAKE128, which has 128-bit security strength, the time complexity 2^{1575} is too high. Let's find the best attack with complexity less than 2^{128}. According to Fig. 51, 6-round in the forward direction has degree of 60 and 5-round in the backward direction has degree of 81. Therefore, we can find a zero-sum distinguisher for 11-round underlying permutation of SHAKE128 with about complexity 2^{81}, which is less than 2^{128}. As we learned from Fig. 55, we can extend one more round by considering all the possible inputs of each active row. According to Fig. 52, we can find a zero-sum distinguisher for 11-round underlying permutation of SHAKE128 with about complexity 2^{85}, which is less than 2^{128}.

3.8 SHAKE256

Best Practical Collision Attack on SHAKE256. The capacity sizes of SHAKE256 and SHA3-256 has the same capacity size as $c = 512$. For further domain separation purpose as explained in Sect. 4.3, SHAKE256 has "1111" as its inner 4-bit padding in addition to 10*1 padding as its outer padding. Note that Keccak-256 uses the 10*1 padding only. In Sect. 3.4, we learned that there is a practical collision attack on 4-round Keccak-256 [22]. Though there is the 4-bit loss of freedom due to "1111" padding compared to the attack on 4-round Keccak-256, the attack procedure used for 4-round Keccak-256 still works for SHAKE256 when the output size d of SHAKE256 is less than or equal to 320-bit, because there are zero-differences on the first 320-bit ((see the first five lanes on the first row of the last differential state of Fig. 32)) of the hash output as shown in Fig. 32.

Best Theoretical Collision Attack on SHAKE256. The capacity sizes of SHAKE256 and SHA3-256 has the same capacity size as $c = 512$. For further domain separation purpose as explained in Sect. 4.3, SHAKE256 has "1111" as its inner 4-bit padding in addition to 10*1 padding as its outer padding. Note that Keccak-256 uses the 10*1 padding only. In Sect. 3.4, we learned that there is a theoretical collision attack on 5-round Keccak-256 with complexity 2^{115} [23]. Though there is the 4-bit loss of freedom due to "1111" padding compared to the attack on 5-round Keccak-256, the attack procedure used for 5-round Keccak-256 can still work for SHAKE256 when the output size d of SHAKE256 is less than or equal to 320-bit. This is because, as explained in the evaluation part of Keccak-256, the internal differential characteristic in Fig. 64 is 2^{-37} and the number of possible 320-bit hash outputs is 2^{184}, and by the help of the squeeze attack and message modification in TIDA, we can find a collision over 320-bit with about complexity 2^{115} in the same way.

Best Practical Preimage and Second-preimage Attack on SHAKE256. As shown from the evaluation part on Keccak-224, the current best practical preimage attack is on 3-round Keccak only with 40-bit messages [45]. But it is not clear how to utilize the result of [45] to find a preimage of 3-round SHAKE256. Note that this preimage attack is also considered as the second-preimage attack.

Best Theoretical Preimage and Second-preimage Attacks on SHAKE256. As described in [48], NIST claims that the security strength of SHAKE256 in terms of preimage resistance is min(d,256), where d is the output size, since we can only guarantee at most 256-bit security due to the capacity size $c = 512$. As shown in Fig. 48, the current best theoretical preimage attack is an attack on 8 rounds of Keccak-256 with complexity less than the general preimage attack complexity 2^{256}, which can be applied to SHAKE256 with d=256. Note that this preimage attack is also considered as the second-preimage attack.

Best Practical Distinguishing Attack on SHAKE256. A zero-sum structure of the 9- and 10-round underlying permutation of SHAKE256 can be found with practical time complexity $2^{29.83}$ and $2^{59.67}$, respectively, which was already explained in the evaluation part of SHA3-224.

Best Theoretical-but-Marginal Distinguishing Attacks on SHAKE256. A zero-sum structure of the full 24-round underlying permutation of SHAKE256 can be found with time complexity 2^{1575}, which was already explained in the evaluation part of SHA3-224. However, compared to the security strength of SHAKE256, which is 256-bit security strength, the time complexity 2^{1575} is too high. Let's find the best attack with complexity less than 2^{256}. According to Fig. 51, 7-round in the forward direction has degree of 128 and 6-round in the backward direction has degree of 243. Therefore, we can find a zero-sum distinguisher for 13-round underlying permutation of SHAKE256 with about complexity 2^{243}, which is less than 2^{256}. As we learned from Fig. 55, we can extend one more round by considering all the possible inputs of each active row. According to Fig. 52, we can find a zero-sum distinguisher for 13-round underlying permutation of SHAKE256 with about complexity 2^{245}, which is less than 2^{256}.

4 Security Evaluation of Domain Extensions

4.1 Security Analysis of Each Domain Extension against Second-preimage Attacks

In 2005, Kelsey and Schneier [37] introduced a general second preimage-finding attack on the MD hash functions with message length padding (called SMD hash functions)such as MD5, SHA-1, SHA-2 with about the birthday attack complexity. Their attack works even when the compression function is a random oracle of fixed input length. Here, we want to explain the idea of Kelsey-Schneier's general second-preimage attacks. For that, we follow the way of explaining the attacks from Chang's PhD thesis [17].

Let $\text{SPRING}_{t_1}^{t_2}(IV;X)$ be a set of message M's such that for all $M \in \text{SPRING}_{t_1}^{t_2}(IV;X)$, $t_1 \leq ||M|| \leq t_2$ and $|\text{SPRING}_{t_1}^{t_2}(IV;X)|=t_2-t_1+1$ and $\text{MD}^f(\text{IV}, M)=X$ and all element have different block-lengths and X is a fixed value. Assume that $t_1 = k$ and $t_2 = 2^k + k - 1$. Let t be $2^k + k + 1$. Their attack consists of the following four steps.

1. the target message M is given (see Fig. 72.).
2. we use $\text{SPRING}_{t_1}^{t_2}(IV;X)$. (see Fig. 73.).
3. we compute $f(X, M_i^*) = h_i^*$ for $1 \leq i \leq 2^{n-k}$, where M_i^*'s are chosen arbitrarily (see Fig. 74.).
4. According to the birthday paradox, with a high probability, there exist h_l^* and h_j such that $h_l^* = h_j$ for $1 \leq l \leq 2^{n-k}$ and $k+1 \leq j \leq 2^k + k$ (see Fig. 75.).
   - Therefore, given a message of $2^k + k + 1$ blocks and $\text{SPRING}_{t_1}^{t_2}(IV;X)$, with the complexity 2^{n-k} we have a second preimage M' such that $pad(M') = S||M_l^*||M_{j+1}||\cdots||M_{2^k+k+1}$ and $SMD^f(IV, M') = h_t$ and S has j-1 block length and is the element of $\text{SPRING}_{t_1}^{t_2}(IV;X)$.

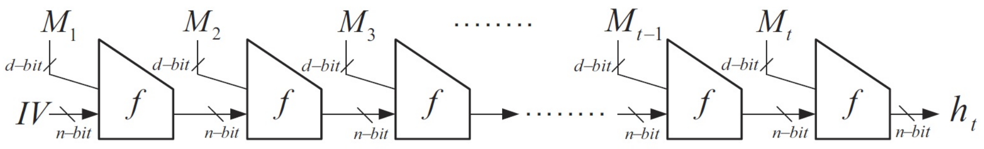


Fig. 72. General Second Preimage Attack: Step 1 [37]

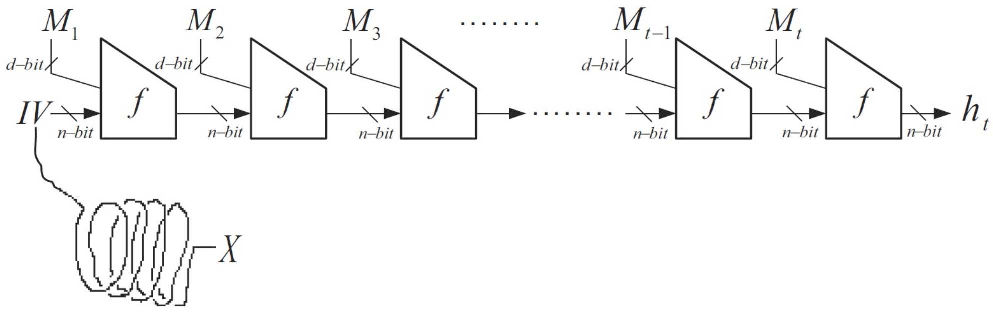


Fig. 73. General Second Preimage Attack: Step 2 [37]

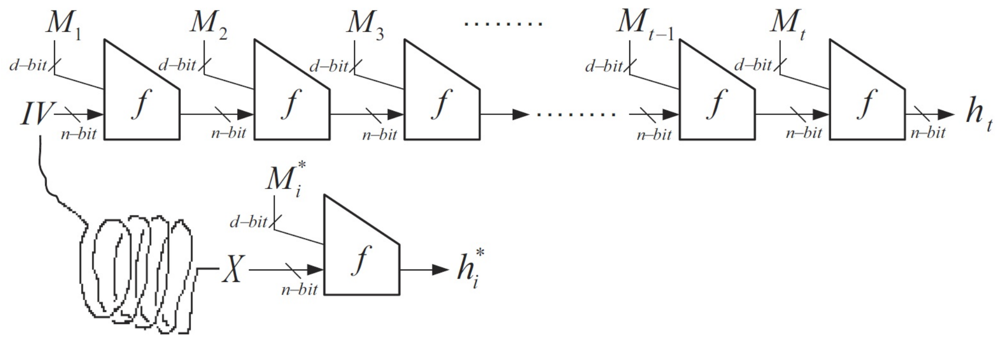


Fig. 74. General Second Preimage Attack: Step 3 [37]

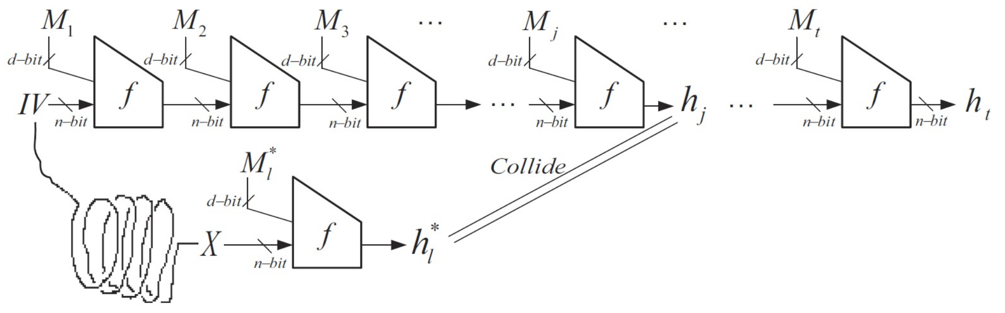
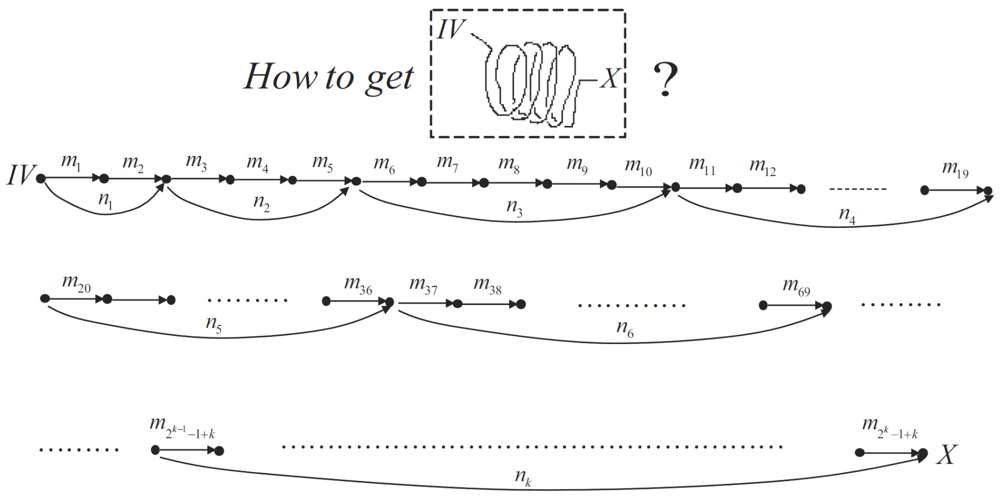


Fig. 75. General Second Preimage Attack: Step 4 [37]



How to get
IV
X
?
m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_{10} m_{11} m_{12} m_{19}
n_1 n_2 n_3 n_4
m_{20} m_{36} m_{37} m_{38} m_{69}
n_5 n_6
$m_{2^{k-1}-1+k}$
m_{2^k-1+k}
n_k
X

Fig. 76. A General Method of Expandable Messages [37]

General Method for Obtaining SPRING$^{t_2}_{t_1}(IV;X)$ with complexity $k2^{n/2}$. See Figure 76. In order to construct SPRING$^{t_2}_{t_1}(IV;X)$, Kelsey and Schneier [37] used Joux's multicollision attack method as shown in Figure **??**. At first, with the birthday attack complexity we can get $m_1||m_2$ and n_1 such that $f(f(IV,m_1),m_2) = f(IV,n_1) = h_2$, n_i and m_i are n bits. Next, with the birthday attack complexity we can get $m_3||m_4||m_5$ and n_2 such that $f(f(f(h_2,m_3),m_4),m_5) = f(h_2,n_2)$. Likewise, for other cases, we can have the results shown in Figure **??**. Therefore, with the complexity $k2^{n/2}$ we can construct $\mathrm{SPRING}_{t_1}^{t_2}(IV;X)$ where t_1 is k and t_2 is 2^k+k-1.

Method for Obtaining $\mathrm{SPRING}_{t_1}^{t_2}(IV;X)$ in the case of MD4-style Hash Functions such as MD4, MD5 and SHA-1 with complexity $2^{n/2+1}$. Kelsey and Schneier [37] described another method for constructing $\mathrm{SPRING}_{t_1}^{t_2}(IV;X)$ using the fixed points of the compression functions of MD4-style hash functions such as MD4, MD5 and SHA-1. The compression functions f's of MD4-style hash functions have the same structure as follows: $f(h_{i-1},M_i)=Inv(h_{i-1},M_i)+h_{i-1}$ where Inv is an efficiently-invertible function with given M_i. So, for any M_i, we can easily get h^* such that $Inv(h^*,M_i)=0$. In other words, $f(h^*,M_i)=h^*$ where h^* is a fixed point of f when a M_i is given. See Figure 77. In the same way, with the complexity 2^s we can compute fixed points p_i's for $1\le i\le s$ where $s=2^{n/2}$. And with the complexity $s=2^{n/2}$ we can compute q_i's for $1\le i\le s$. Then, according to the birthday paradox, with the high probability there exist q_i and p_j such that $q_i=p_j$ for some i and j.

Finally, with the complexity $2^{n/2+1}$ we can construct $\mathrm{SPRING}_{t_1}^{t_2}(IV;X)$ using m_i and n_j where $X=q_i$ or p_j, t_1 and t_2 are any values. See Figure 78.

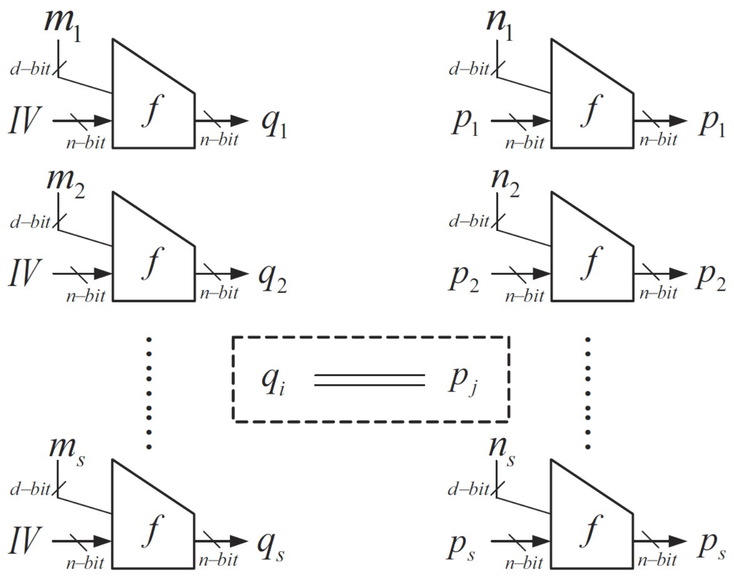


Fig. 77. A General Method of Expandable Messsages in case of SHA-256 style Hash Function [37]: Step 1

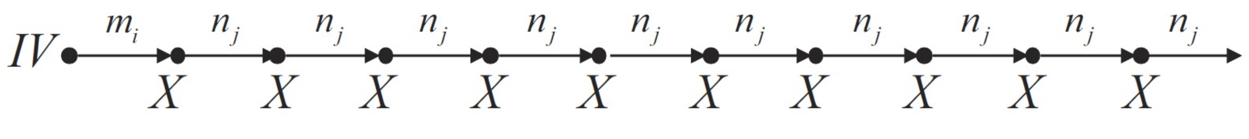


Fig. 78. A General Method of Expandable Messsages in case of SHA-256 style Hash Function [37]: Step 2

Security of Domain Extensions of SHA-224 against second-preimage attacks. The complexity of Kelsey-Schneier's general second-preimage attack on SHA-224 depends on the internal state size. In case of SHA-224, the state size is 256 bits and it is very easy to find fixed points so the attack complexity would be $O(2^{128})$, which is much less than the optimal security bound of hash functions with hash output sizes of 224 or 256 bits.

Security of Domain Extensions of SHA-512/224 and SHA-512/256 against second-preimage attacks. The complexity of Kelsey-Schneier's general second-preimage attack on SHA-224 depends on the internal state size. In cases of SHA-512/224 and SHA-512/256, their internal state sizes are 512 bits only so the Kelsey-Schneier's attack complexity on them would be $O(2^{256})$, which is beyond than the general second-preimage attack complexity 2^{224} for SHA-512/224. We know from [18] that any indifferentiable attack on chopMD with n-bit internal state and $(n-s)$-bit hash output requires at least $O(min(2^{n-s-1}, \frac{2^s}{3(n-s)+1}))$ queries when the compression function of chopMD is assumed to be the random oracle. In case of SHA-512/224, n=512 and s=288, the attack complexity would be $O(min(2^{223}, \frac{2^{288}}{3\times 224+1})) = O(2^{223})$ which shows that SHA-512/224 is guaranteed to provide the optimal security against the second-preimage attacks. In case of SHA-512/256, n=512 and s=256, the attack complexity would be $O(min(2^{255}, \frac{2^{256}}{3\times 256+1})) = O(2^{246})$ which shows that SHA-512/224 is guaranteed to provide at least the almost optimal security against the second-preimage attacks.

Security of Domain Extensions of SHA3-224, SHA3-256, SHA3-384, SHA3-512 against second-preimage attacks. The four SHA3 hash functions are based on the sponge construction, not based on the MD construction. So, Kelsey-Schneier's general second-preimage attacks cannot be directly applied to the four SHA3 hash functions. It was shown in [7] that the sponge construction provides at least $2^{c/2}$ indifferentiable security, where c is the capacity size. Since the capacity sizes of the four SHA-3 hash functions are double of their hash outputs, the four SHA-3 hash function provides the optimal security against second-preimage attacks.

Security of Domain Extensions of SHAKE128 and SHAKE256 against second-preimage attacks. It was shown in [7] that the sponge construction provides at least $2^{c/2}$ indifferentiable security, where c is the capacity size. In case of SHAKE128, c=256, so, we can only guarantee the 128-bit security against the second preimage attacks, regardless of its output size. In case of SHAKE256, c=512, so, we can only guarantee the 256-bit security against the second preimage attacks, regardless of its output size.

4.2 Domain Separation of SHA-224, SHA-512/224 and SHA-512/256

The initial value IV_{224} of SHA-224 consists of eight 32-bit words, which represent the second thirty-two bits of the fractional parts of the square roots of the 9th through 16th primes. On the other hand, $IV_{512/224}$ and $IV_{512/256}$ are defined by using the SHA-512 hash function as follows:

$$IV_{512/224}\text{=SHA-512(“SHA-512/224”) with } IV' = IV_{512} \oplus \text{0xa5a5a5...a5}$$
$$IV_{512/256}\text{=SHA-512(“SHA-512/256”) with } IV' = IV_{512} \oplus \text{0xa5a5a5...a5}$$

Each domain extension can be described in Fig. 79, Fig. 80, and Fig. 81.

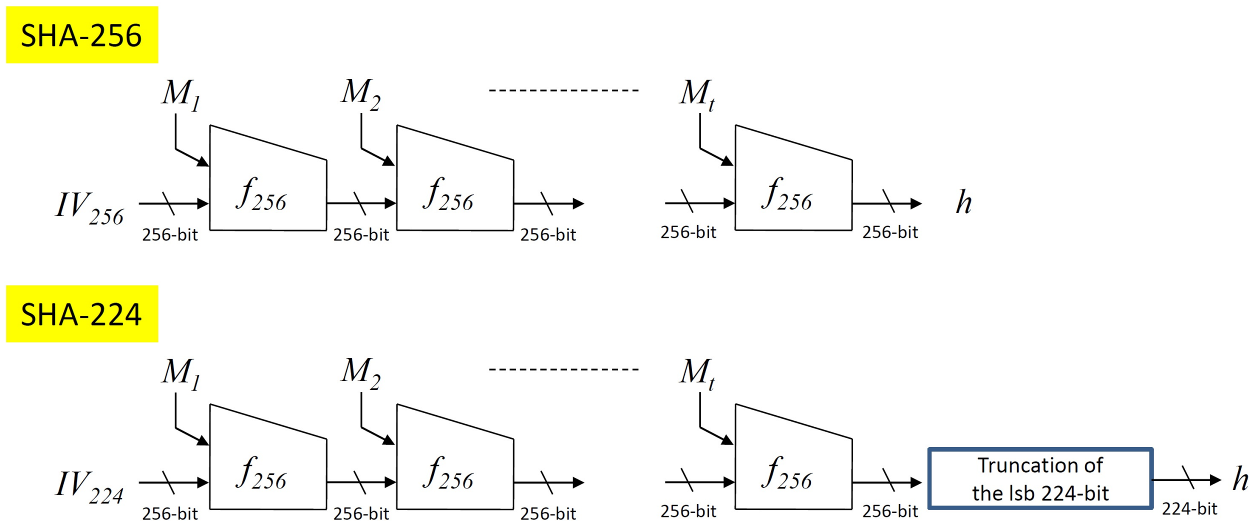


Fig. 79. Domain Extensions of SHA-256 and SHA-224, where f_{256} is the compression function of SHA-256.

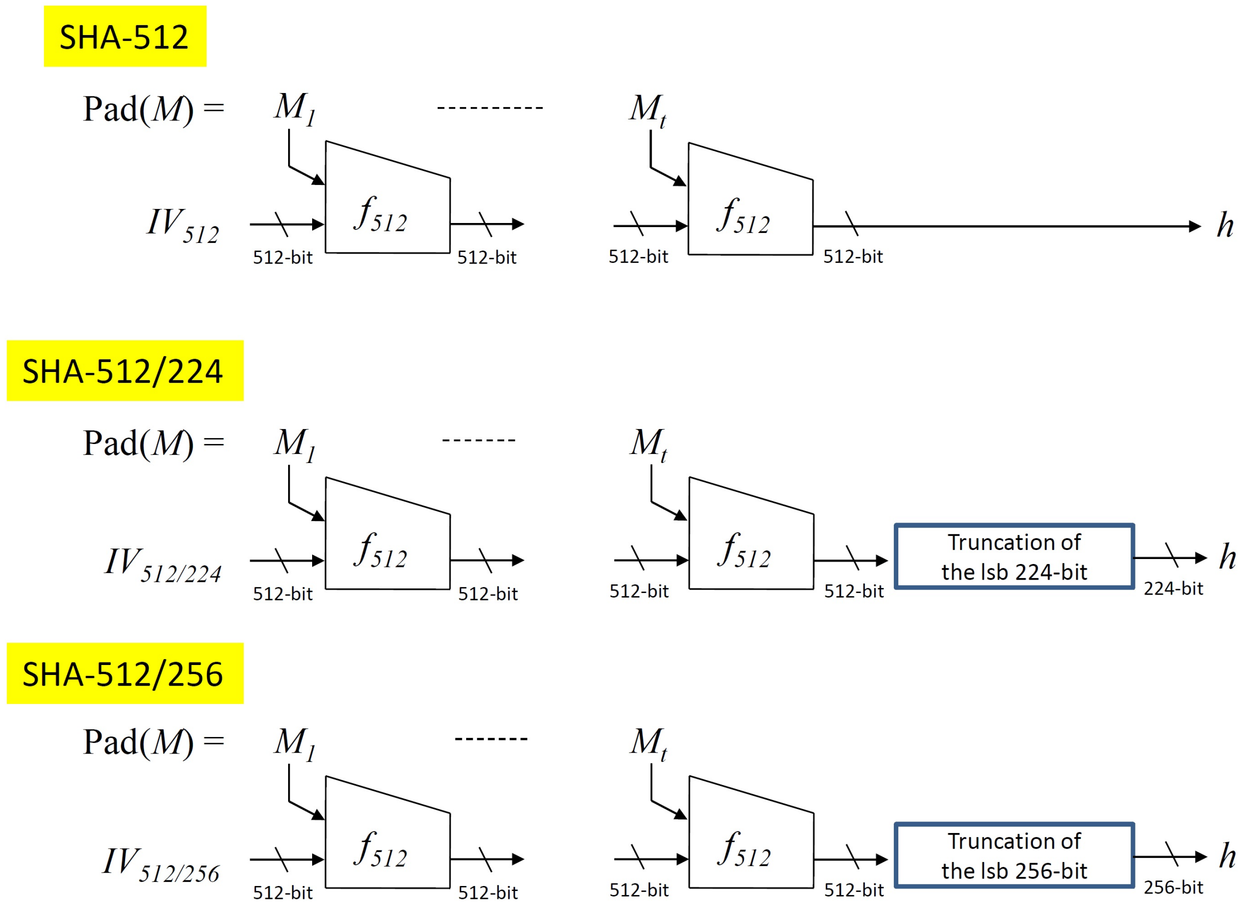


Fig. 80. Domain Extensions of SHA-512, SHA-512/224, and SHA-512/256, where Pad is the padding rule of SHA-512. Unlike SHA-224, the initial values of SHA-512/224 and SHA-512/256 are generated by calling SHA-512 for one-block padded message.

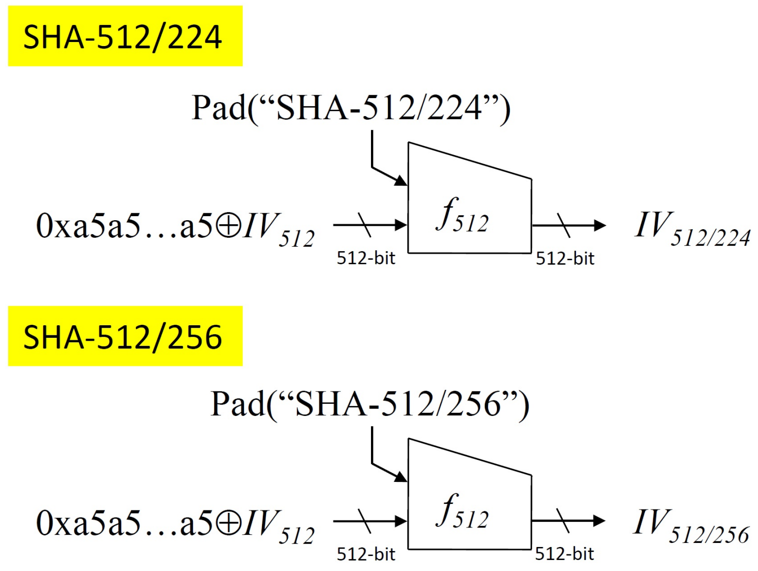


Fig. 81. Initial Values $IV_{512/224}$ and $IV_{512/256}$ of SHA-512/224, and SHA-512/256: The initial values of SHA-512/224 and SHA-512/256 are generated using $IV_{512} \oplus$ 0xa5a5....a5, where IV_{512} is the initial value of SHA-512, Pad is the padding rule of SHA-512, and f_{512} is the compression function of SHA-512.

Now, a question arises: What if we only use IV_{512} rather than $IV_{512} \oplus$ 0xa5a5a5...a5? If we we use the same IV_{512} without tweak for generating $IV_{512/224}$ and $IV_{512/256}$, we can easily show that SHA-512 and SHA-512/224 (or 'SHA-512 and SHA-512/256') are related and dependent on each other. Let us define a message M' such that Pad(M'):=(Pad("SHA-512/224")$||M_1|| \cdots ||M_{t-1}$) for an integer $t > 1$. Then, let us define another message M such that Pad(M)=($M_1|| \cdots ||M_{t-1}||M_t$), where $M_1, \cdots, M_{t-1}$ were already defined from Pad(M'). Once we know SHA-512(M)=h', then we can calculate h(=SHA-512/224(M)) without calling SHA-512/224 but calculating $h = f_{512}(h', M_t)$. Therefore, we can easily find a dependency between two hash outputs of SHA-512 and SHA-512/224 if IV_{512} is used rather than $IV_{512} \oplus$ 0xa5a5a5...a5. Likewise, we can see that it is also easy to find a dependency between two hash outputs of SHA-512 and SHA-512/256 if IV_{512} is used rather than $IV_{512} \oplus$ 0xa5a5a5...a5.

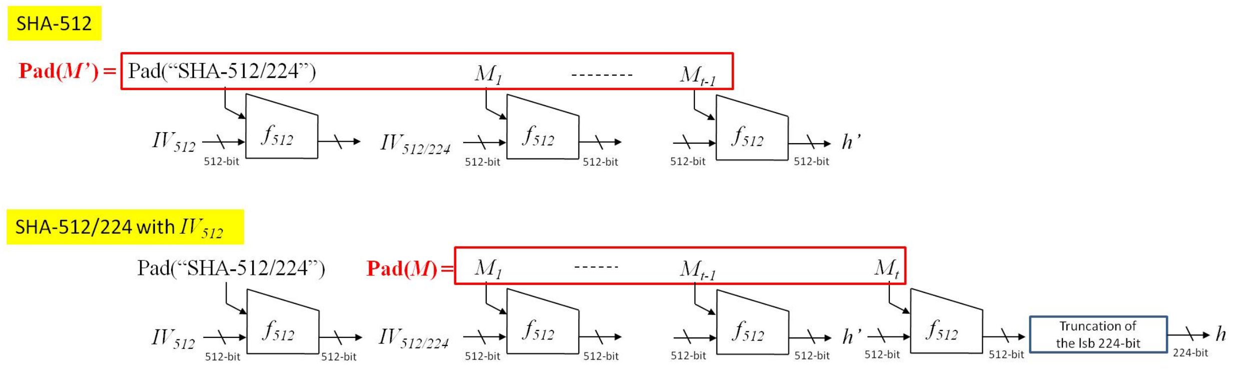


Fig. 82. The reason of Generating $IV_{512/224}$ not from IV_{512}: if $IV_{512/224}$ is generated only from IV_{512} without any tweak on it, it is easy to show that SHA-512 and SHA-512/224 are not independent.

4.3 Domain Separation of the SHA-3 Functions

The padding rule Pad's of the six SHA-3 functions are defined by composing two padding rules; one is the outer padding Pad_{out}, called multi-rate padding 10^*1, and the other is the inner padding Pad_{in}, say partition-padding. Therefore, the padding Pad of each of SHA-3 functions is described as $\text{Pad(M)}=\text{Pad}_{out}(\text{Pad}_{in}((\text{M}))$. As shown in [7], the sponge construction, on which SHA-3 functions are defined, is indifferentiably secure in random oracle model and in the single stage setting model [51], under the condition that the last block of padded message should be non-zero.

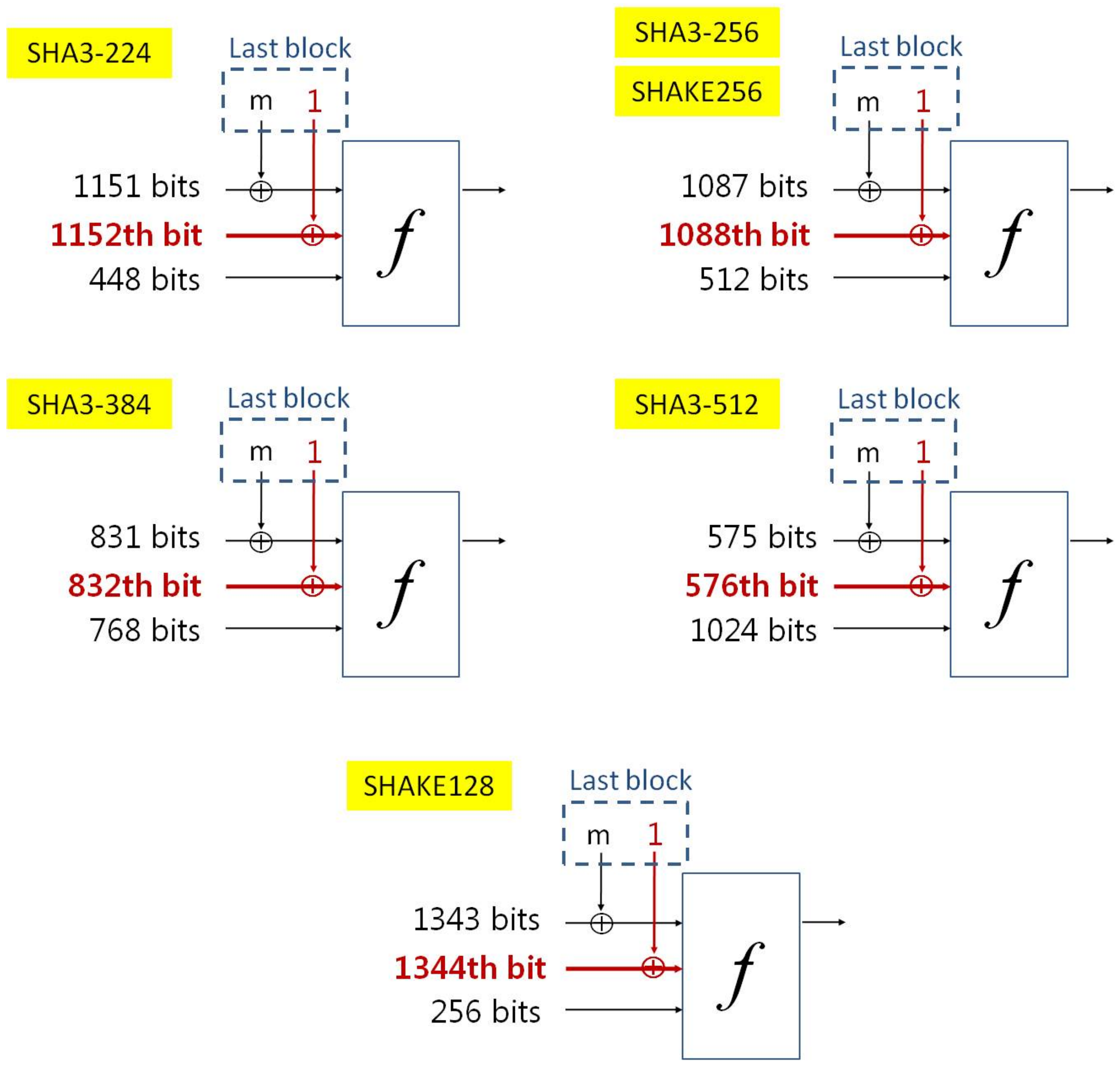


Fig. 83. The last block of padded message after outer padding 10^*1 of the SHA-3 functions

Using multi-rate padding 10^*1 as the Outer Padding (Domain Separation between Two domain extensions when their bitrates are different.) As we can see from Fig. 83, the bitrate of SHA3-n is 1600-n for n=224,256,384,512. Therefore, after applying the multi-rate padding 10^*1 as the outer padding is that in case of SHA3-224 the 1152-th bit of the last 1600-bit input chaining state will be XORed with the bit '1', in case of SHA3-256 the 1088-th bit of the last 1600-bit input chaining state will be XORed with the bit '1', in case of SHA3-384 the 832-th bit of the last 1600-bit input chaining state will be XORed with the bit '1', and in case of SHA3-512 the 576-th bit of the last 1600-bit input chaining state will be XORed with the bit '1'. Therefore, we can see that the last blocks of padded messages for any two different variants of SHA-3 family will be always different and non-zero, which will guarantee that there is no dependency on the hash outputs of different variants of SHA-3 family with help of the result of indifferentiable security of the sponge construction [7].

Using Partition-Padding Approach as the Inner Padding(Domain Separation between Two domain extensions when their bitrates are same.) In order to explain the necessity of the inner padding, firstly let us see the cases of SHA3-256 and SHAKE-256.

$$\begin{aligned}\text{SHA3-256}(M) &= \textsc{Keccak}[512](M\|01, 256);\\ \text{SHAKE256}(M,d) &= \textsc{Keccak}[512](M\|1111, d);\end{aligned}$$

As we can see, the capacity c is same as 512 for SHA3-256 and SHAKE256. SHA3-256 uses "01" padding as its inner padding and SHAKE256 uses "1111" as its inner padding. More in detail, the inner paddings are defined in Fig 85. The reason why we need such inner padding is that we need to guarantee that hash outputs will be independent even for the same capacity size. If there is no such inner padding, it is easy to find any dependency on the outputs of SHA3-256 and SHAKE256.

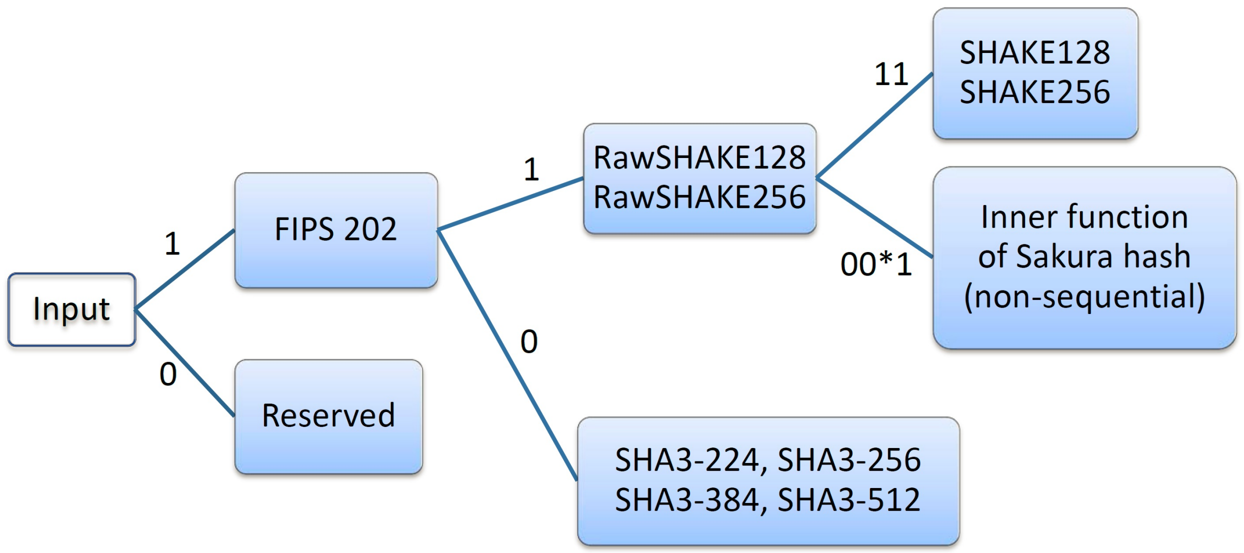


Fig. 84. Inner Padding Methods via Padding-partition. [30] : Note that the padding bits are reversely ordered.

As we can see examples in Fig. 85, for various purposes we can partition inner padding bits for domain separation.

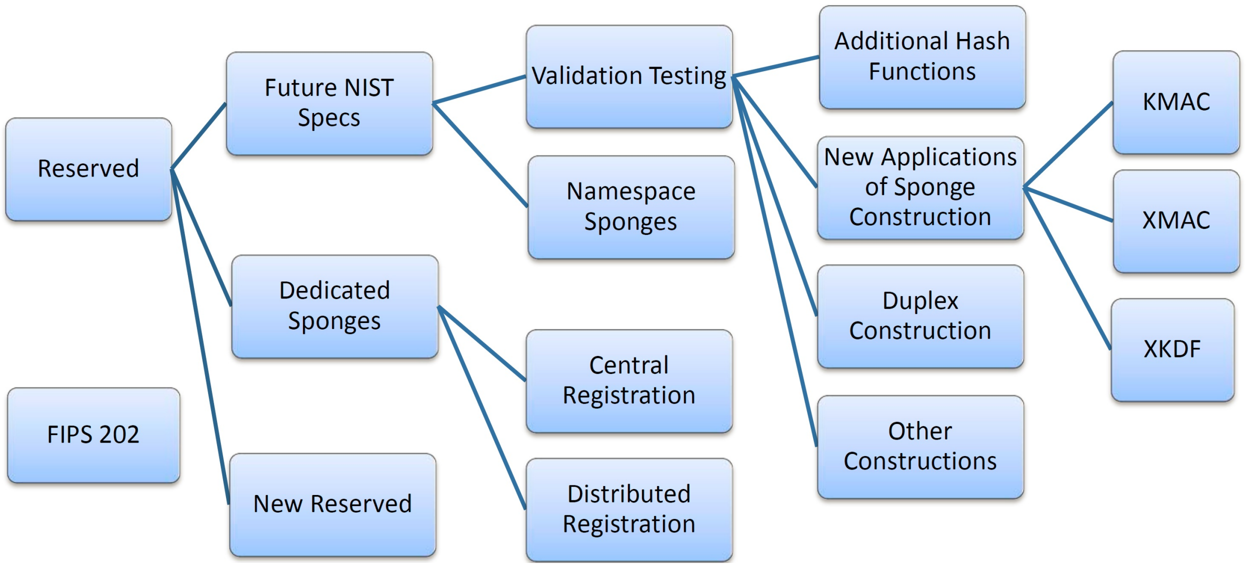


Fig. 85. Example for Structure of Extensions [30]

5 Evaluation of HMAC based on the Domain Extension of SHA-224, SHA-512/224, SHA-512/256, and the Four SHA-3 Hash Functions

In this section, we examine the security of HMAC based on the chopMD, which is the domain extension of SHA-224, SHA-512/224, and SHA-512/256. A chopMD hash function is a MD hash function with its final output truncation.

HMAC [4]. Let K be a n-bit key. We define $\overline{K} = K||0^{b-n}$ where b indicates the size of the message block of a hash algorithm H (ex. $b = 512$ for SHA-224, $b = 1024$ for SHA-512/224 and SHA-512/256).opad is formed by repeating the byte '0x36' as many times as needed to get a b-bit block, and ipad is defined similarly using the byte '0x5c'. Then, HMAC is defined as follows:

$$\text{HMAC}_K(M) = H(\overline{K} \oplus \mathsf{opad}||H(\overline{K} \oplus \mathsf{ipad}||M)).$$

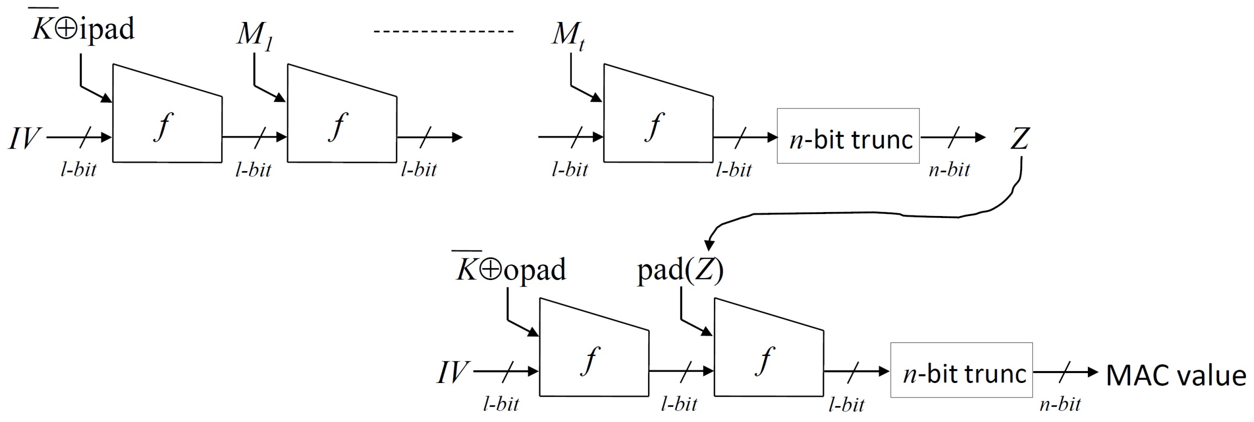


Fig. 86. HMAC construction: for SHA-224, ℓ=256 and n=224, for SHA-512/224, ℓ=512 and n=224, for SHA-512/256, ℓ=512 and n=256

In this section, we consider the following four attacks.

- **Existential Forgery**: the attacker builds valid pair (M,T), without having queried M.
- **Universal Forgery**: the attacker first receives a message M sent as a challenge, and then builds valid pair (M,T), without having queried M.
- **Internal State Recovery**: the attacker recovers any internal state during any MAC value computation.
- **Key Recovery**: the attacker recovers the secret-key K used in the MAC algorithm.

According to existing analyses, we can summarize the best attack complexities on HMAC based SHA-224, SHA-512/224, and SHA-512/256 as shown in Table 12. As we can see, when the key size (k) is same as the hash output size, except for SHA-224, the best known attacks on SHA-512/224 and SHA-512/256 are the exhaustive search over the possible key space. Based on existing attacks, SHA-512/224 provides better security than SHA-224.

Algorithm HMAC-SHA-v	ℓ	s	Existential Forgery $\min(2^k, O(2^{\ell/2})$ [50])	Universal Forgery $\min(2^k, O(\ell \cdot 2^{\ell-s})$ [35])	Internal State Recovery $\min(2^k, O(2^{\ell-s})$ [41, 24])	Key Recovery $\min(2^k, O(2^{3\ell/4})$ [35])
v=224	256	55	$\min(2^k, O(2^{128})$ [50])	$\min(2^k, O(2^{200})$ [35])	$\min(2^k, O(2^{201})$ [41, 24])	$\min(2^k, O(2^{192})$ [35])
v=512/224	512	118	$\min(2^k, O(2^{256})$ [50])	$\min(2^k, O(2^{393})$ [35])	$\min(2^k, O(2^{394})$ [41, 24])	$\min(2^k, O(2^{384})$ [35])
v=512/256	512	118	$\min(2^k, O(2^{256})$ [50])	$\min(2^k, O(2^{393})$ [35])	$\min(2^k, O(2^{394})$ [41, 24])	$\min(2^k, O(2^{384})$ [35])

Table 12. Best Known Attack Complexity of HMAC based on SHA-224, SHA-512/224, and SHA-512/256: k is the key size, and ℓ is the internal state size, 2^s is the maximum block length of message.

Security of HMAC based on the four SHA-3 hash functions. There are four SHA-3 hash functions, SHA3-224, SHA3-256, SHA3-384, and SHA3-512. The capacity size of each SHA-3 hash function is double of its hash output size, which is similar to SHA-512/224 and SHA-512/256. Therefore, we can at least guarantee that the best known forgery, key-recovery, internal-state recovery attacks on HMAC based on the MD hash functions such as SHA-224, SHA-256, SHA-512 cannot be applied to HMAC based on SHA-3 hash functions with less complexity than $min(2^k, 2^n)$, where k is the key size and n is the hash output size.

6 Evaluation of MAC, Stream Cipher, and AE based on the SHA-3 Functions

In this section, we explain results of [25] for MACs, stream ciphers, and AEs based on SHA-3 functions. Let us see attack ideas.

Key Recovery Attack using Linear Superpolys [25]. As an example, let's consider a boolean function $f(x_0, x_1, x_2, x_3) = x_0 + x_0x_1x_2 + x_0x_1x_3 + x_2$. And f can be described in the following ways:

$$\begin{aligned}
f(x_0, x_1, x_2, x_3) &= x_0(1 + x_1x_2 + x_1x_3) + x_2,\\
f(x_0, x_1, x_2, x_3) &= x_1(x_0x_2 + x_0x_3) + x_0 + x_2,\\
f(x_0, x_1, x_2, x_3) &= x_0x_1(x_2 + x_3) + x_0 + x_2.
\end{aligned}$$

We say $t_{\{0\}} = x_0$, $t_{\{1\}} = x_1$, $t_{\{0,1\}} = x_0x_1$, $P_{t_{\{0\}}}(x_0, x_1, x_2, x_3) = 1+x_1x_2+x_1x_3$, $P_{t_{\{1\}}}(x_0, x_1, x_2, x_3) = x_0x_2+x_0x_3$, $P_{t_{\{0,1\}}}(x_0, x_1, x_2, x_3) = x_2+x_3$, $Q_{t_{\{0\}}}(x_0, x_1, x_2, x_3) = x_2$, $P_{t_{\{1\}}}(x_0, x_1, x_2, x_3) = x_0+x_2$, $P_{t_{\{0,1\}}}(x_0, x_1, x_2, x_3) = x_0 + x_2$.

So, we can describe f in the following ways:

$$\begin{aligned}
f(x) &= t_{\{0\}} \cdot P_{t_{\{0\}}}(x) + Q_{t_{\{0\}}}(x),\\
f(x) &= t_{\{1\}} \cdot P_{t_{\{1\}}}(x) + Q_{t_{\{1\}}}(x),\\
f(x) &= t_{\{0,1\}} \cdot P_{t_{\{0,1\}}}(x) + Q_{t_{\{0,1\}}}(x).
\end{aligned}$$

Genearally let $f : \{0,1\}^n \to \{0,1\}$ and let $I \subseteq \{0, 1, 2, \cdots, n-1\}$. Then, f can be described as follows:

$$f(x) = t_I \cdot P_{t_I}(x) + Q_{t_I}(x).$$

where none of the terms in $Q_{t_I}(x)$ is divisible by t_I.

The key-recovery attack using the linear superpolys consists of two phases, preprocessing (offline) phase and online phase.

Preprocessing (Offline) Phase. Let us consider a boolean function $f(v, x)$ where $v = (v_1, \cdots, v_{d-1})$ are (d-1) public variables (variables controlled by the attacker, e.g. a message or a nonce) and $x = (x_1, \cdots, x_n)$ are n secret key variables. Let $I = \{1, \cdots, d-1\}$ and we define $C_I = \{(b_1, ..., b_{d-1}) | 1 \le \forall j \le d-1, b_j \in \{0,1\}\}$, which is the set of all binary vectors of the length $d-1$.

$$\sum_{v \in C_I} f(v, x) = P_{t_I}(\underbrace{1, \cdots, 1}_{d-1\ times}, x) = L(x)$$

where, $L(x)$ is called the *superpoly* of C_I.

Assuming the degree of $f(v, x)$ is d, then, according to [40], we can write

$$L(x) = a_1x_1 + \cdots a_nx_n + c$$

If the boolean function f's algebraic normal form is not public and we only know input-output relations of f and the degree d of f, we interpolate the linear coefficients of $L(x)$ as follows:

- fine the constant $c = \sum_{v \in C_I} f(v, 0)$

– find $a_i = \sum_{v \in C_I} f(v, 0, \cdots, \underbrace{1}_{x_i}, 0, \cdots, 0))$

Online Phase. In online phase, our target is to find the secret key variables x. For that, we need n linearly independent equations to find x by the Gaussian elimination. In order to get a linear equation, an attacker has to make 2^{d-1} chosen plaintext queries v's for every $v \in C_I$ to an oracle $f(\cdot, x)$ to get the value $f(v, x)$, where x are n secret key variables, and he can calculate the value of $L(x)$ by summing up all the values of $f(v, x)$'s as follows:

$$\sum_{v \in C_I} f(v, x) = P_{t_I}(\underbrace{1, \cdots, 1}_{d-1\ times}, x) = L(x)$$

Let the value of $L(x)$ be b_{t_I}. We have to repeat this procedure for different boolean functions f's until he can get n linearly independent equations.

Divide-and-Conquer Key Recovery Attack using Partial-key-dependent Constant Superpolys [25]. For example, let the secret key size be 128-bit. Assume that constant superpolys of some cubes, where each cube is formed by d public variables, are defined only by the first 64-bit value of the 128-bit key. Then, the idea of the divide-and-conquer key recovery attack [25] is as follows:

1. (Offline) For each of 2^{64} possible cases of the first 64-bit value of the secret key, the attacker precomputes and stores the values of the constant superpolys depending only on the first 64-bit, where the value of each constant superpoly for each 64-bit case can be calculated by summing all the possible outputs such as MAC values or internal states or keystreams, etc.
2. (Online) Once the 128-bit secret key is fixed, the attacker makes 2^d queries by considering all the possible values for the d public variables and sums all the returned values to get the values of superpolys. Then, we can get the first 64-bit value of the secret key by finding matching values with the summed values from the precomputed table.

Forgery Attack using a Zero Superpoly [25]. As we learned from the zero-sum distinguisher, The zero-sum structures can be easily found using the concept of higher-order derivatives of a function. Higher-order derivatives of a function and its related property were well studied in [40]. Let us repeat one of his results as follows:

Definition 4. *[40] Let F be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$. For any $a \in \mathbb{F}_2^n$ the derivative of F with respect to a is the function $D_aF(x) = F(x + a) + F(x)$. For any k-dimensional subspace V of $\mathbb{F}_2^n$ and for any basis of V, $\{a_1, \cdots, a_k\}$, the k-th order derivative of F with respect to V is the function defined by*

$$D_V F(x) = D_{a_1} D_{a_2} \cdots D_{a_k} F(x) = \sum_{v \in V} F(x + v), \forall x \in \mathbb{F}_2^n$$

And for every subspace V of dimension ($\geq deg(F) + 1$),

$$D_V F(x) = \sum_{v \in V} F(x + v) = 0, \forall x \in \mathbb{F}_2^n$$

Let us consider a boolean function $f(v, x)$ of degree d where $v = (v_1, \cdots, v_{d+1})$ are $(d+1)$ public variables (variables controlled by the attacker, e.g. a message or a nonce) and $x = (x_1, \cdots, x_n)$ are n secret key variables. According to the above result of [40], the following equation holds (in other words, the superpoly of C_I is zero):

$$\sum_{v \in C_I} f(v,x) = 0$$

where the degree of $f(v,x)$ is d, $C_I = \{(b_1, ..., b_{d+1}) | 1 \leq \forall j \leq d+1, b_j \in \{0,1\}\}$, which is the set of all binary vectors of the length $d+1$.

Therefore, once we know $\sum_{v \in C_I - \{v'\}} f(v,x)$ through $2^{d+1}-1$ queries, we can know that $f(v',x) = \sum_{v \in C_I - \{v'\}} f(v,x)$ without any additional query, which is a successful forgery attack, because $f(v',x) \oplus \sum_{v \in C_I - \{v'\}} f(v,x)$ should be zero according to the above result of [40].

Keystream Prediction [25]. Higher-order derivatives of a function and its related property were well studied in [40]. Let us consider a boolean function $f(IV,x)$ of degree d where $IV = (v_1, \cdots, v_d)$ are d public initial variables (initial variables controlled by the attacker) and $x = (x_1, \cdots, x_n)$ are n secret key variables. According to the above result of [40], the following equation holds:

$$\sum_{IV \in C_I} f(IV, 0, \cdots, 0, 0) = \sum_{IV \in C_I} f(IV, 0, \cdots, 0, 1) = \cdots = \sum_{IV \in C_I} f(IV, 1, \cdots, 1, 1) = \text{constant}$$

where constant is either 1 or 0, the degree of $f(IV,x)$ is d, $C_I = \{(b_1, ..., b_d) | 1 \leq \forall j \leq d, b_j \in \{0,1\}\}$, which is the set of all binary vectors of the length d.

Therefore, once the attacker computes $\sum_{IV \in C_I} f(IV, 0, \cdots, 0, 0)$ (*offline phase*) and knows $\sum_{IV \in C_I - \{IV'\}} f(IV,x)$ through $2^d - 1$ queries (*online phase*), the attacker can know that $f(IV',x) = \sum_{v \in C_I - \{v'\}} f(v,x)$ without any additional query for the previously unseen initial value IV', which is a successful keystream prediction attack, because $(f(IV',x) \oplus \sum_{IV \in C_I - \{IV'\}} f(IV,x))$ should be zero according to the above result of [40].

6.1 Key-recovery Attack on MAC construction based on 5-round Keccak

[25] considered the 5-round version of KECCAK with a 1600-bit state with r=1024 and c=576 and the authors considered a MAC construction with the 128-bit key and 128-bit tag as shown in Fig. 87.

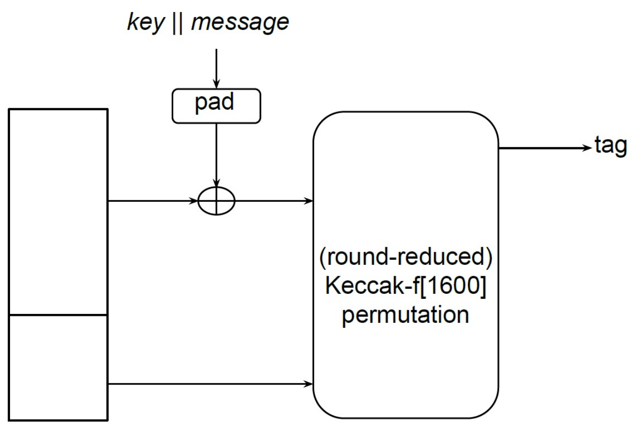


Fig. 87. MAC based on Keccak [25]

Since the degree of the 5-round version of Keccak is at most 32 (=2^5), the authors [25] chose 31 public variables $v = (v_1, \cdots, v_{31})$ among message area in Fig. 88 for a cube attack on the 5-round version of Keccak as MAC for the key recovery attack. As shown already, the superpoly of any cube with 31 variables consists of linear terms only.

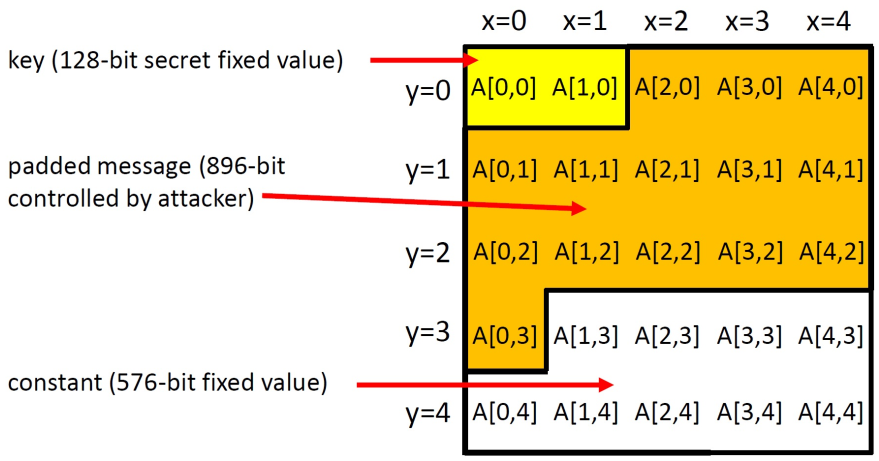


Fig. 88. Area of key, message, and constant of input state of the permutation considered in [25] for Key Recovery Attack on 5-round Keccak working as MAC: the ordering of lanes is defined according to Fig. 89

Preprocessing (Offline) Phase. Once we choose any cube with 31 variables among 896 message bits, do the following procedure to find a linear equation over 128 secret key variables by defining different f's with allocation of any fixed constant values onto the remaining 865 bits (=896-31).

– fine the constant $c = \sum_{v \in C_I} f(v, 0)$

– find $a_i = \sum_{v \in C_I} f(v, 0, \cdots, \underbrace{1}_{x_i}, 0, \cdots, 0))$

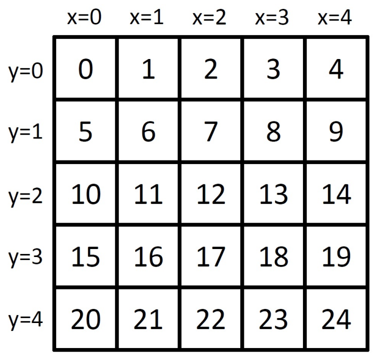


Fig. 89. Lane Odering of State considered in [25]

According to Sect. 4.1 of [25], only 20-25% of the superpolys were useful (i.e. non-constant) based on their simulation. The authors finally found 117 linearly independent equations using 19 cubes and several output bits.

Online Phase. We can guess remaining 11 additional secret key-bit values and find the 128-bit secret key by making $19 \cdot 2^{31} \approx 2^{35}$ chosen plaintext MAC queries and by using the Gaussian elimination.

6.2 Key-recovery Attack on Stream Cipher construction based on 6-round Keccak

For the 6-round version of Keccak with a 1600-bit state with r=1024 and c=576, the authors [25] considered a stream cipher construction with the 128-bit key and 128-bit initial value IV as shown in Fig. 90. As described in [25], The first 960 of the 1024 available output bits contain 960/5=192 full rows, which can be converted using χ^{-1} operating on the rows independently, so we can compute 960 bits after 5.5 rounds with probability 1. Therefore, we only need to break 5.5-round version with at most 32 degrees, because the first half round is linear.

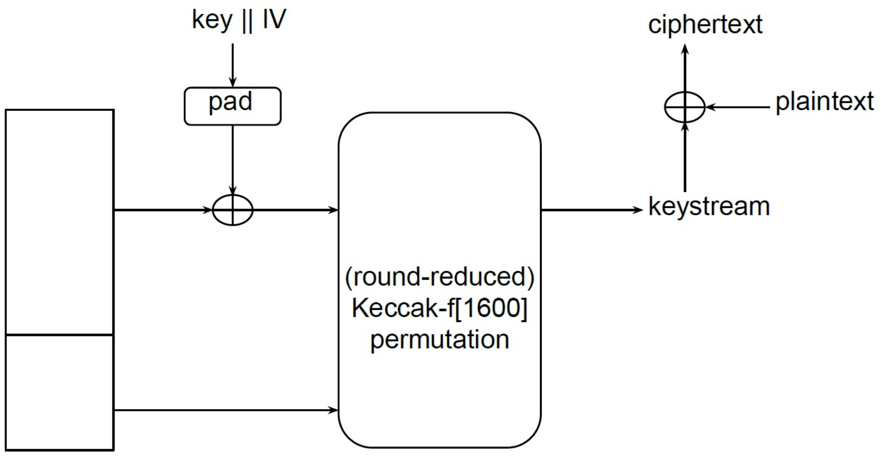


Fig. 90. Stream Cipher based on Keccak [25]

Since the degree of the 5.5-round version of Keccak is at most 32 ($=2^5$), the authors [25] chose 31 public variables $v = (v_1, \cdots, v_{31})$ among initial value area in Fig. 91 for a cube attack on the 5.5-round version of Keccak as stream cipher for the key recovery attack. As shown already, the superpoly of any cube with 31 variables consists of linear terms only.

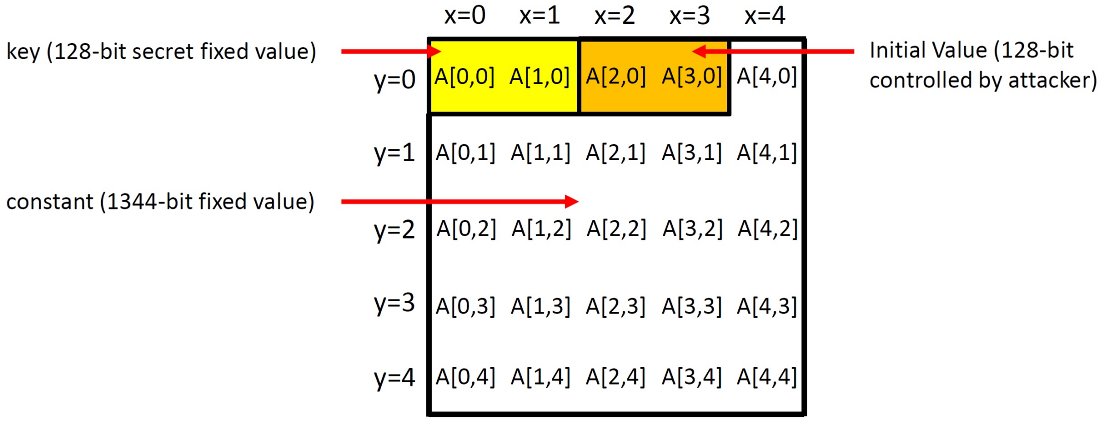

x=0
x=1
x=2
x=3
x=4
key (128-bit secret fixed value)
Initial Value (128-bit controlled by attacker)
constant (1344-bit fixed value)
y=0
y=1
y=2
y=3
y=4
A[0,0] A[1,0] A[2,0] A[3,0] A[4,0]
A[0,1] A[1,1] A[2,1] A[3,1] A[4,1]
A[0,2] A[1,2] A[2,2] A[3,2] A[4,2]
A[0,3] A[1,3] A[2,3] A[3,3] A[4,3]
A[0,4] A[1,4] A[2,4] A[3,4] A[4,4]

Fig. 91. Area of key, initial value, and constant of input state of the permutation considered in [25] for Key Recovery Attack on 6-round Keccak working as Stream Cipher: the ordering of lanes is defined according to Fig. 89

Preprocessing (Offline) Phase. Once we choose any cube with 31 variables among 128 initial value bits, do the following procedure to find a linear equation over 128 secret key variables by defining different f's with allocation of any fixed constant values onto the remaining 97 bits (=128-31).

- fine the constant $c = \sum_{v \in C_I} f(v, 0)$
- find $a_i = \sum_{v \in C_I} f(v, 0, \cdots, \underbrace{1}_{x_i}, 0, \cdots, 0))$

According to Sect. 4.2 of [25], the authors finally found 128 linearly independent equations using 25 cubes and several output bits.

Online Phase. Therefore, we can find the 128-bit secret key by making $25 \cdot 2^{31} \approx 2^{36}$ chosen IV queries and by using the Gaussian elimination.

6.3 Key-recovery Attack on Authenticated Encryption based on 6-round Keyak

For the 6-round version of Keccak with a 1600-bit state with r=1348 and c=252, the authors [25] considered Keyak family algorithms [11] with the 128-bit key, 128-bit nonce, and 128-bit tag as shown in Fig. 92. The attack procedure using the chosen nonce attack is same as the key-recovery attack on 6-round version of stream cipher in Sect. 6.2.

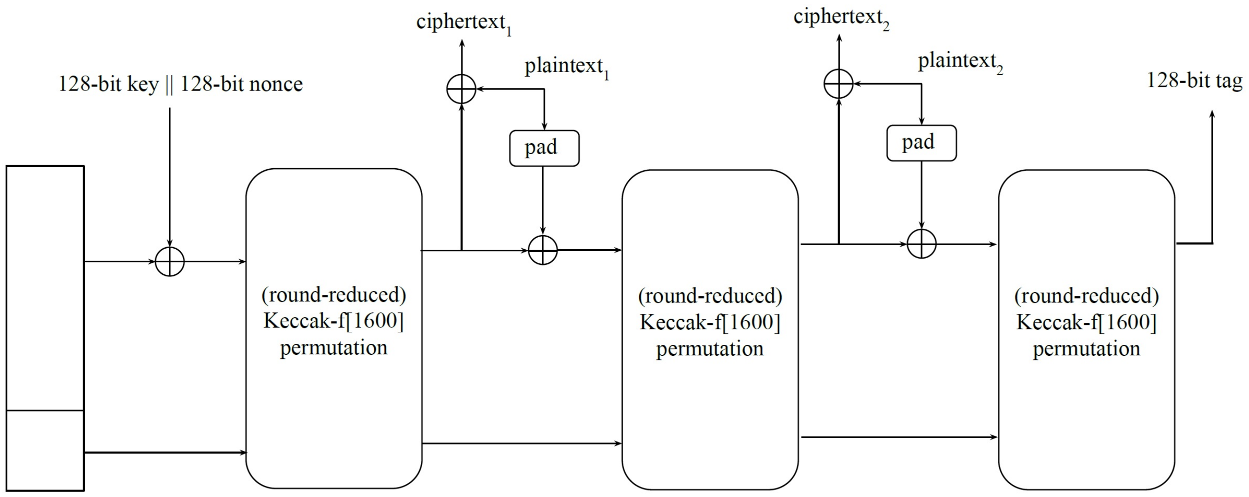

ciphertext$_1$
plaintext$_1$
ciphertext$_2$
plaintext$_2$
128-bit key || 128-bit nonce
128-bit tag
pad
pad
(round-reduced) Keccak-f[1600] permutation
(round-reduced) Keccak-f[1600] permutation
(round-reduced) Keccak-f[1600] permutation

Fig. 92. Lake Keyak processing two plaintext blocks [25] : The capacity c is 252 and the bitrate r is 1348.

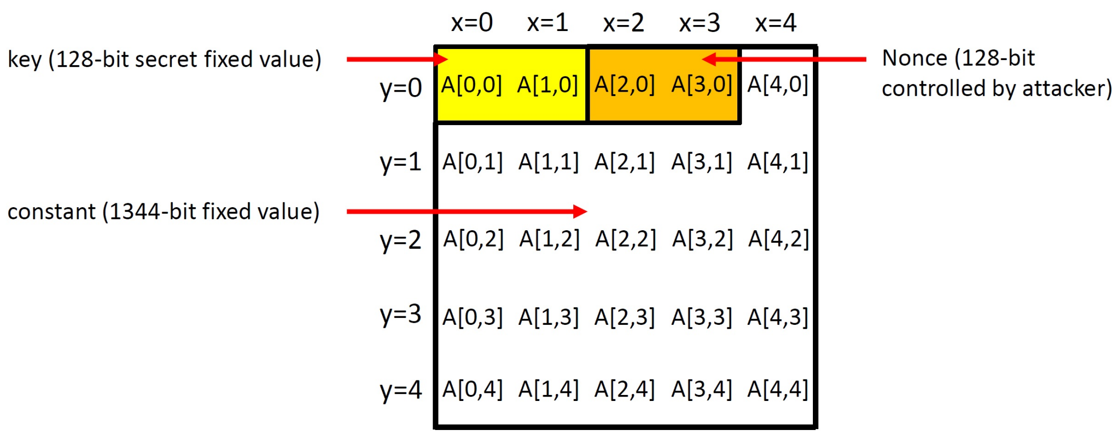

key (128-bit secret fixed value)
constant (1344-bit fixed value)
Nonce (128-bit controlled by attacker)
x=0 x=1 x=2 x=3 x=4
y=0 A[0,0] A[1,0] A[2,0] A[3,0] A[4,0]
y=1 A[0,1] A[1,1] A[2,1] A[3,1] A[4,1]
y=2 A[0,2] A[1,2] A[2,2] A[3,2] A[4,2]
y=3 A[0,3] A[1,3] A[2,3] A[3,3] A[4,3]
y=4 A[0,4] A[1,4] A[2,4] A[3,4] A[4,4]

Fig. 93. Area of key, initial value, and constant of input state of the permutation considered in [25] for Key Recovery Attack on 6-round Keyak: the ordering of lanes is defined according to Fig. 89

6.4 Forgery Attack on MAC construction based on 7-round and 8-round Keccak and 7-round Keyak

Forgery Attack on MAC construction based on 7-round Keccak [25]. In case of 7-round Keccak, its degree is at most $d = 128$. However, if we carefully define variables the public variables $v = (v_1, \cdots, v_i)$, we can reduce the degree of 7-round Keccak from $d = 128$ to $d = 64$. More precisely, as shown in Fig. 94, there are 896-bit padded message area, which can be controlled by the attacker.

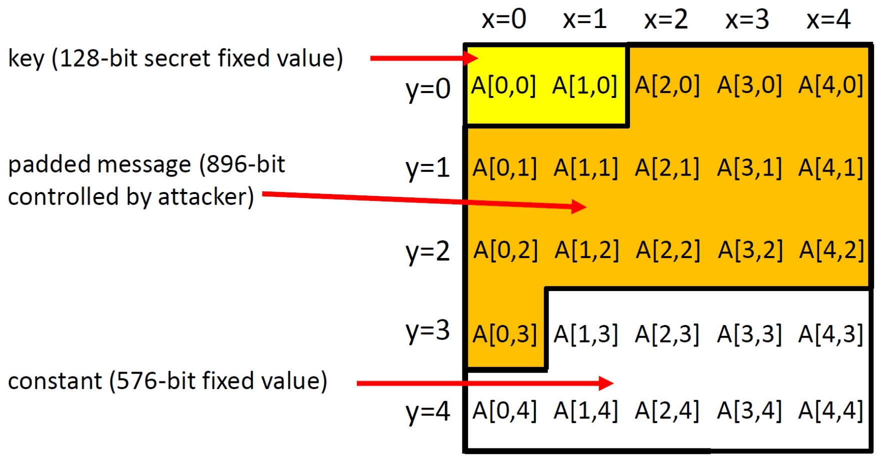

x=0
x=1
x=2
x=3
x=4
key (128-bit secret fixed value)
padded message (896-bit controlled by attacker)
constant (576-bit fixed value)
y=0
y=1
y=2
y=3
y=4
A[0,0] A[1,0] A[2,0] A[3,0] A[4,0]
A[0,1] A[1,1] A[2,1] A[3,1] A[4,1]
A[0,2] A[1,2] A[2,2] A[3,2] A[4,2]
A[0,3] A[1,3] A[2,3] A[3,3] A[4,3]
A[0,4] A[1,4] A[2,4] A[3,4] A[4,4]

Fig. 94. Area of key, message, and constant of input state of the permutation considered in [25] for Forgery Attack on 7-round Keccak working as MAC: the ordering of lanes is defined according to Fig. 89

Then, as shown in Fig. 95, we, as an attacker, define two sets of random variables, $V_1 = (v_1^1, v_2^1, \cdots, v_i^1)$ from $A[0,2]$, and $V_2 = (v_1^2, v_2^2, \cdots, v_j^2)$ from $A[1,1]$, where $i+j = 65$. Let C_1 and C_2 be any i-bit constant and j-bit constants chosen by the attacker, respectively. Then we define the i positions of A[0,3] as $V_1 \oplus C_1$ and the j positions of A[1,2] as $V_2 \oplus C_2$ such that $A[0,2] \oplus A[0,3]$ and $A[1,1] \oplus A[1,2]$ are constant regardless what values are assigned to the 65 variables. Except for the 130-bit positions defined by the 65 public random variables, remaining 766 (=896-130) bits are fixed as any constants chosen by the attacker. As we can see from Fig. 95, there is no change of positions influenced by the 65 variables after θ step, because $A[0,2] \oplus A[0,3]$ and $A[1,1] \oplus A[1,2]$ are constant regardless what values are assigned to the 65 variables. Since ρ step independently works in each lane, there is again no change of non-constant positions. As we can see from Fig. 96, the final positions influenced by the 65 variables will be determined as shown in the final status of Fig. 95.

As we can see from the final status of Fig. 95, there are no two adjacent positions (determined by the random variables) in a row and each of the 130 positions influenced by the 65 random variables are described by a boolean function with degree 1. By the definition of χ step, after χ step, there is no increase of degree because there are no two adjacent positions (influenced by the random variables) in a row. So, there is no increase of degree after the first round, and the degree of 7-round Keccak will be 64 (=2^6) only, not 128 (=2^7). Therefore, as explained in [25], the forgery attack works by collecting $2^{65} - 1$ tags for $2^{65} - 1$ chosen message queries by varing 65 public variables among message parts in Fig. 88, we can get the tag (= the sum of all $2^{65} - 1$ tags) for the last

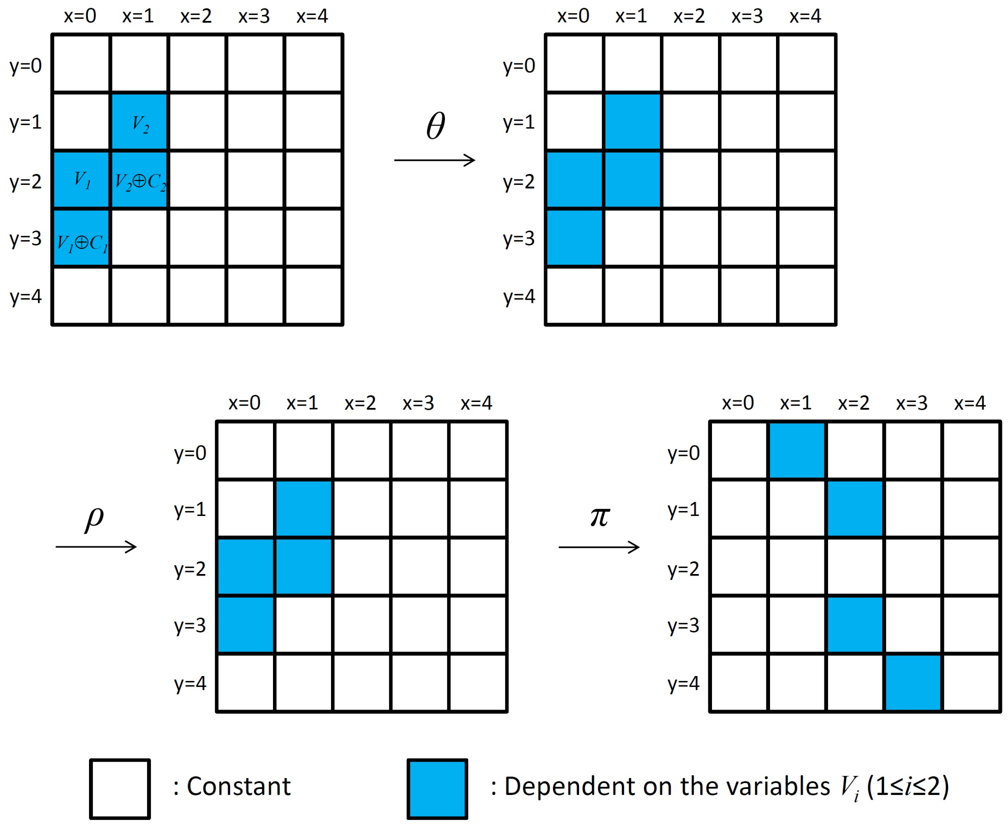


Fig. 95. Transition through the first linear part of the round (θ, ρ, π steps) for 7-round Forgery Attack: C_1 and C_2 are any constants, V_i ($1 \leq i \leq 2$) are variables.

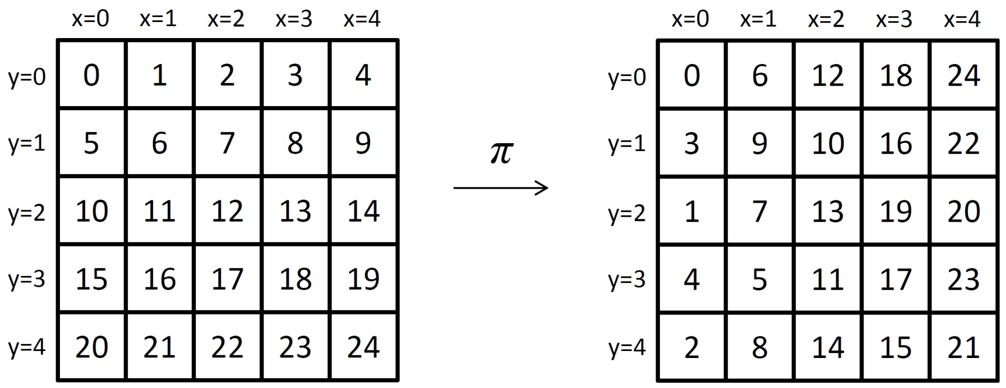


Fig. 96. Change of Positions through π step

remaining chosen message without any additional query.

Forgery Attack on MAC construction based on 8-round Keccak [25]. In case of 8-round Keccak, its degree is at most $d = 256$. However, if we carefully define variables the public variables $v = (v_1, \cdots, v_i)$, we can reduce the degree of 8-round Keccak from $d = 256$ to $d = 128$. More precisely, as shown in Fig. 97, there are 768-bit padded message area with 256-bit key size, which can be controlled by the attacker.

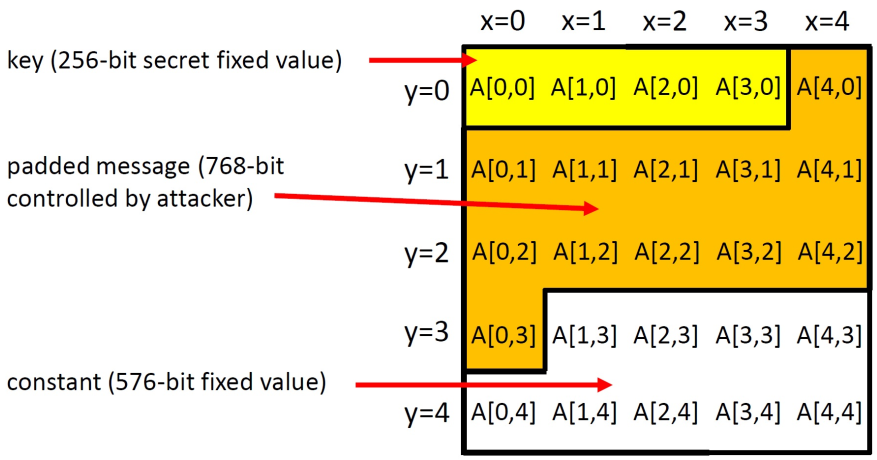

x=0
x=1
x=2
x=3
x=4
key (256-bit secret fixed value)
padded message (768-bit controlled by attacker)
constant (576-bit fixed value)
y=0
y=1
y=2
y=3
y=4
A[0,0] A[1,0] A[2,0] A[3,0] A[4,0]
A[0,1] A[1,1] A[2,1] A[3,1] A[4,1]
A[0,2] A[1,2] A[2,2] A[3,2] A[4,2]
A[0,3] A[1,3] A[2,3] A[3,3] A[4,3]
A[0,4] A[1,4] A[2,4] A[3,4] A[4,4]

Fig. 97. Area of key, message, and constant of input state of the permutation considered in [25] for Forgery Attack on 8-round Keccak working as MAC with 256-bit key: the ordering of lanes is defined according to Fig. 89

Then, as shown in Fig. 98, we, as an attacker, define three sets of random variables, $V_1 = (v_1^1, v_2^1, \cdots, v_i^1)$ from $A[2,0]$, $V_2 = (v_1^2, v_2^2, \cdots, v_j^2)$ from $A[4,0]$, and $V_3 = (v_1^3, v_2^3, \cdots, v_j^3)$ from $A[4,1]$, where $i + 2j = 129$. Let C_1 and C_2 be any i-bit constant and j-bit constants chosen by the attacker, respectively. Then we define the i positions of A[2,1] as $V_1 \oplus C_1$ and the j positions of A[4,2] as $V_2 \oplus V_3 \oplus C_2$ such that $A[2,0] \oplus A[2,1]$ and $A[4,0] \oplus A[4,1] \oplus A[4,2]$ are constant regardless what values are assigned to the 129 variables. Except for the (2i+3j)-bit positions defined by the 129 (=i+2j) public random variables, remaining $(768 - 2i + 3j)$ bits are fixed as any constants chosen by the attacker. As we can see from Fig. 98, there is no change of positions influenced by the 129 variables after θ step, because $A[2,0] \oplus A[2,1]$ and $A[4,0] \oplus A[4,1] \oplus A[4,2]$ are constant regardless what values are assigned to the 129 variables. Since ρ step independently works in each lane, there is again no change of non-constant positions. As we can see from Fig. 96, the final positions influenced by the 129 variables will be determined as shown in the final status of Fig. 98.

As we can see from the final status of Fig. 98, there are no two adjacent positions (determined by the random variables) in a row and each of the $(2i+3j)$ positions influenced by the 129 $(=i+2j)$ random variables are described by a boolean function with degree 1. By the definition of χ step, after χ step, there is no increase of degree because there are no two adjacent positions (influenced by the random variables) in a row. So, there is no increase of degree after the first round, and the degree of 8-round Keccak will be 128 $(=2^7)$ only, not 256 $(=2^8)$. Therefore, as explained in [25], the forgery attack works by collecting $2^{129} - 1$ tags for $2^{129} - 1$ chosen message queries by varing 129 $(= i + 2j)$ public variables among message parts in Fig. 97, we can get the tag (= the sum of

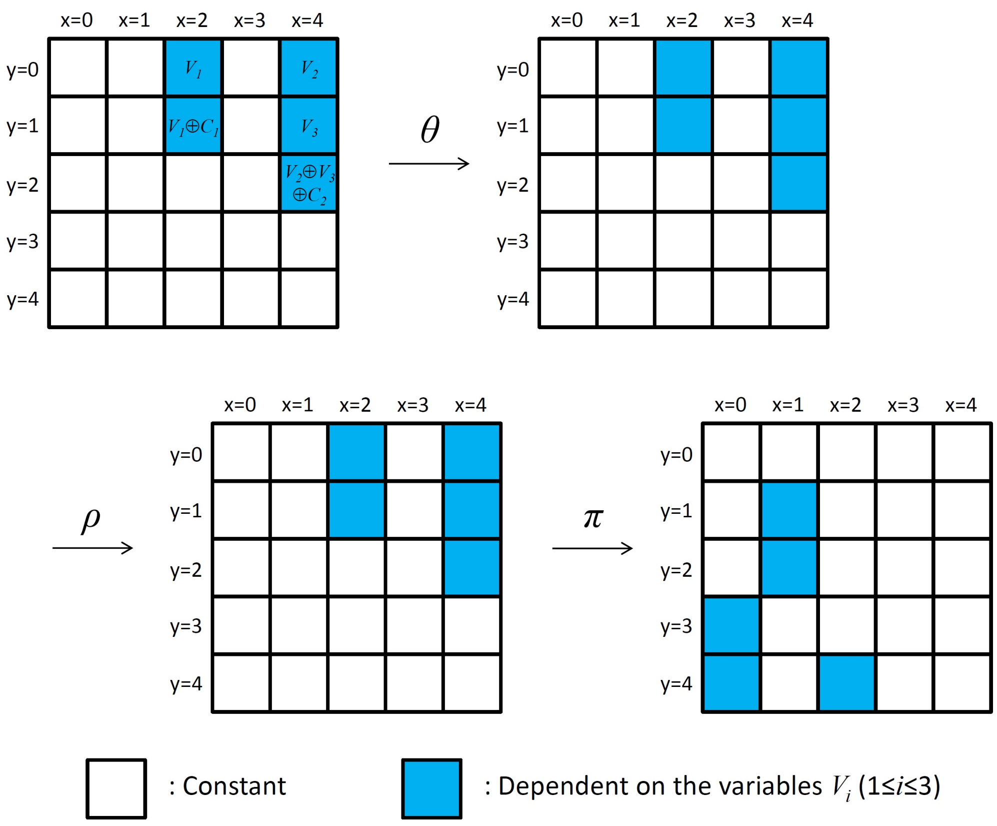


Fig. 98. Transition through the first linear part of the round $(\theta, \rho, \pi$ steps) for 8-round Forgery Attack: C_1 and C_2 are any constants, V_i $(1 \leq i \leq 3)$ are variables.

all $2^{129} - 1$ tags) for the last remaining chosen message without any additional query.

Forgery Attack on 7-round Keyak [25]. In this case, we assume that the nonce is fixed. The forgery attack is to start from the state after the first permutation call as shown in Fig. 99. In the same with the forgery attack on MAC construction based on 7-round KECCAK, we can forge the 128-bit tag with complexity 2^{65} in case of 7-round Keyak [25].

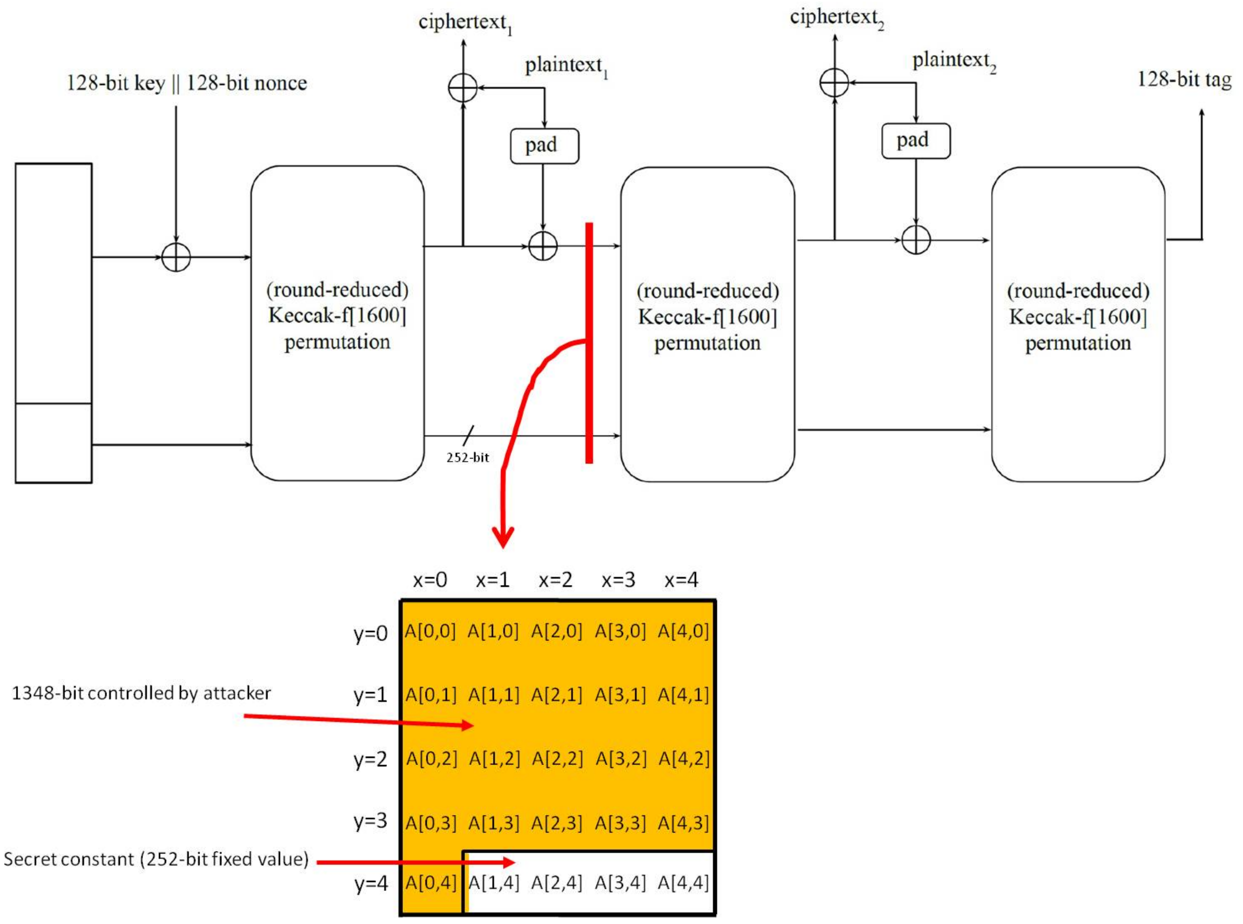

ciphertext$_1$
plaintext$_1$
128-bit key || 128-bit nonce
pad
ciphertext$_2$
plaintext$_2$
pad
128-bit tag
(round-reduced) Keccak-f[1600] permutation
(round-reduced) Keccak-f[1600] permutation
(round-reduced) Keccak-f[1600] permutation
252-bit
x=0 x=1 x=2 x=3 x=4
y=0 A[0,0] A[1,0] A[2,0] A[3,0] A[4,0]
y=1 A[0,1] A[1,1] A[2,1] A[3,1] A[4,1]
y=2 A[0,2] A[1,2] A[2,2] A[3,2] A[4,2]
y=3 A[0,3] A[1,3] A[2,3] A[3,3] A[4,3]
y=4 A[0,4] A[1,4] A[2,4] A[3,4] A[4,4]
1348-bit controlled by attacker
Secret constant (252-bit fixed value)

Fig. 99. Area of 252-bit Secret Constant and 1348-bit Values controlled by the attacker when the initial value is fixed in [25] for Forgery Attack on 7-round Keyak: the ordering of lanes is defined according to Fig. 89

6.5 Keystream Prediction for 8- and 9-round Keccak-based Stream Cipher

In this subsection, the attacker targets on predicting a keystream for a previously unseen IV' in the stream cipher mode.

- In case of keystream prediction for 8-round KECCAK-based Stream Cipher, we assume that the key size is 256-bit, the initial value size is 128-bit, the bitrate $r = 1024$, and capacity $c = 576$. Having 1024 bits of keystream, we can invert 960 bits among the 1024 bits of the keystream through ι and χ, since *chi* works on every 5-bit row independently. Therefore, in this attack, the attacker targets on 7.5 rounds, not 8 rounds.

– In case of keystream prediction for 9-round Keccak-based Stream Cipher, we assume that the key size is 512-bit, the initial value size is 256-bit, the bitrate $r = 1024$, and capacity $c = 576$. Having 1024 bits of keystream, we can invert 960 bits among the 1024 bits of the keystream through ι and χ, since χ works on every 5-bit row independently. Therefore, in this attack, the attacker targets on 8.5 rounds, not 9 rounds.

Keystream Prediction for 8-round Keccak-based Stream Cipher [25]. For the 8-round version of Keccak with a 1600-bit state with r=1024 and c=576, the authors [25] considered a stream cipher construction with the 256-bit key and 128-bit initial value IV as shown in Fig. 90. As described in [25], The first 960 of the 1024 available output bits contain 960/5=192 full rows, which can be converted using χ^{-1} operating on the rows independently, so we can compute 960 bits after 7.5 rounds with probability 1. Therefore, we only need to break 7.5 round version with at most 128 degrees, because the first half round is linear. Let $IV = (v_1, \cdots, v_{128})$ be 128 public initial variables (initial variables controlled by the attacker) and $x = (x_1, \cdots, x_{256})$ be 256 secret key variables. Let $f(IV, x)$ be the 7.5-round Keccak. In the offline process, the attacker computes $\sum_{IV \in C_I} f(IV, 0, \cdots, 0, 0)$ with time complexity 2^{128}. In the online process, the attacker knows $\sum_{IV \in C_I - \{IV'\}} f(IV, x)$ through $2^{128} - 1$ queries, and then the attacker can know that $f(IV', x) = \sum_{v \in C_I - \{v'\}} f(v, x)$ without any additional query for the previously unseen initial value IV', which is a successful keystream prediction attack, because $(f(IV', x) \oplus \sum_{IV \in C_I - \{IV'\}} f(IV, x))$ should be zero.

Keystream Prediction for 9-round Keccak-based Stream Cipher [25]. For the 9-round version of Keccak with a 1600-bit state with r=1024 and c=576, the authors [25] considered a stream cipher construction with the 512-bit key and 256-bit initial value IV. In the same way with the 8-round prediction attack, we can predict the keystream for an unused IV' with complexity 2^{256}.

6.6 Divide-and-Conquer Key Recovery Basic Attack on 6-round Keccak-based MAC

[25] considered the 6-round version of Keccak with a 1600-bit state with r=1024 and c=576 and the authors considered a MAC construction with the 128-bit key and 128-bit tag as shown in Fig. 87. Since the degree of the 6-round version of Keccak is at most 64 (=2^6). Then, as shown in Fig. 100, we, as an attacker, define a set of random variables, $V_1 = (v_1^1, v_2^1, \cdots, v_{32}^1)$ from A[2,2]. Let C_1 be any 32-bit constant. Then we define the 32 positions of A[2,3] as $V_1 \oplus C_1$ such that $A[2, 2] \oplus A[2, 3]$ is constant regardless what values are assigned to the 32 variables. As we can see from the final status of Fig. 100, any position dependent on the second 64-bit secret value assigned to A[2,0] and any position dependent on the public random variables V_1 are not adjacent to each other in a row. Also, the positions influenced by V_1 and $V_1 \oplus C_1$ are located in different rows. So, by the definition of χ step, after χ step, there is no increase of degrees among random variables and the value of any superpoly of the cube formed by all the 32 random variables V_1 after 6-round is independent of the value of the second 64-bit secret key in A[1,0]. On the other hand, as we can see from the final status of Fig. 101, A[1,0] and A[2,0] are adjacent and A[2,3] and A[3,3] are adjacent, which means that the value of any superpoly of the cube formed by all the 32 random variables V_1

after 6-round is dependent on the value of the first 64-bit secret key in A[0,0].

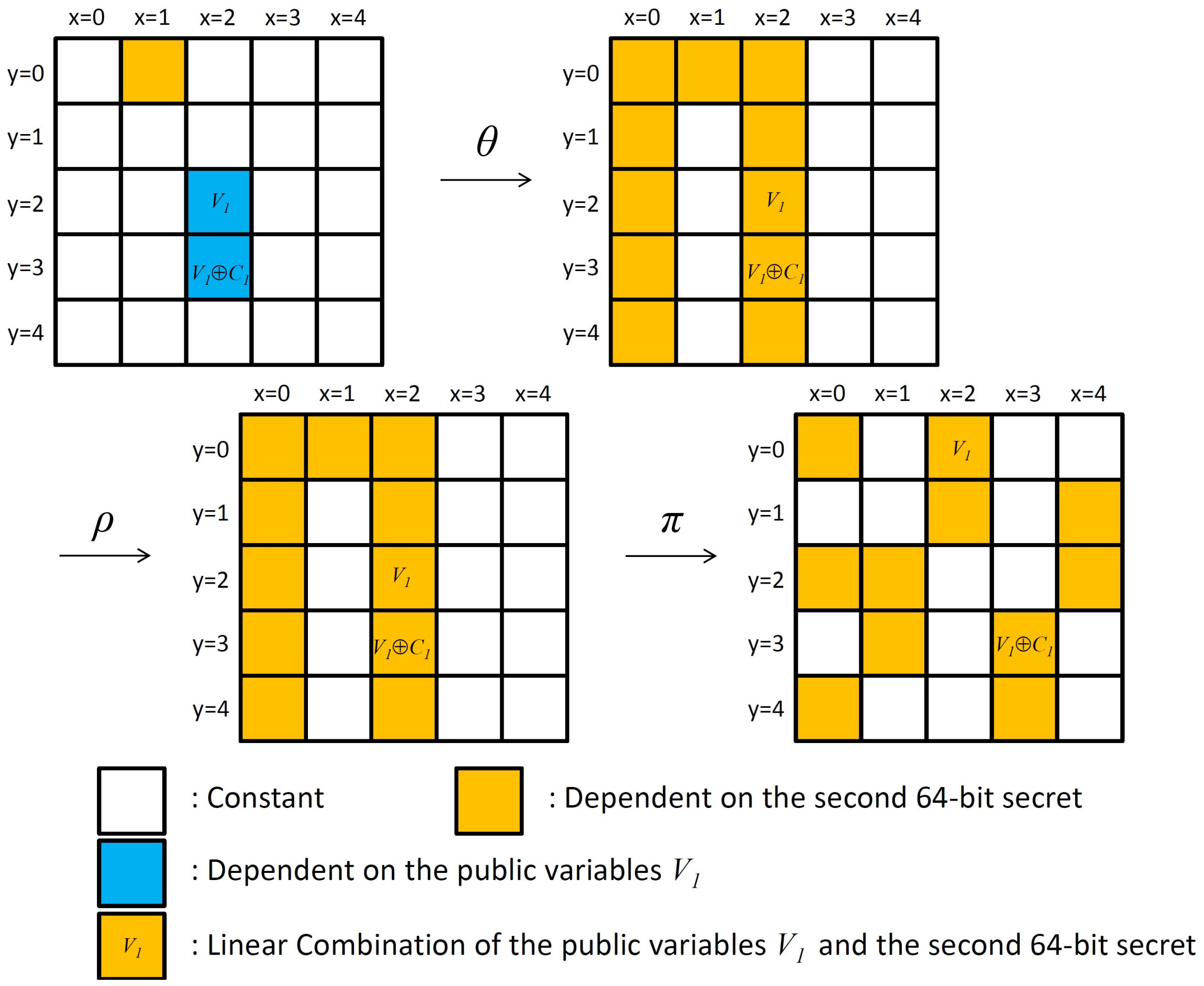

x=0
x=1
x=2
x=3
x=4
y=0
y=1
y=2
y=3
y=4
V_1
$V_1 \oplus C_1$
θ
ρ
π
: Constant
: Dependent on the second 64-bit secret
: Dependent on the public variables V_1
: Linear Combination of the public variables V_1 and the second 64-bit secret

Fig. 100. Transition generated by the second 64-bit secret value A[1,0] [25]: the ordering of lanes is defined according to Fig. 89

Therefore, the following two properties hold [25]:

- Property 1: The cube sum of each output bit after 6 rounds does not depend on the value of A[1,0].
- Property 2: The cube sums of the output bits after 6 rounds depend on the value of A[0,0].

The offline phase of the divide-and-conquer key recovery attack [25] with time complexity 2^{96} and memory complexity 2^{64} is as follows:

1. Set the capacity lanes (A[1,4], A[2,4], A[3,4], A[4,4]) to zero. Set all other state bits (besides A[0,0] and the cube variables) to an arbitrary constant.
2. For each of the 2^{64} possible values of A[0,0]:

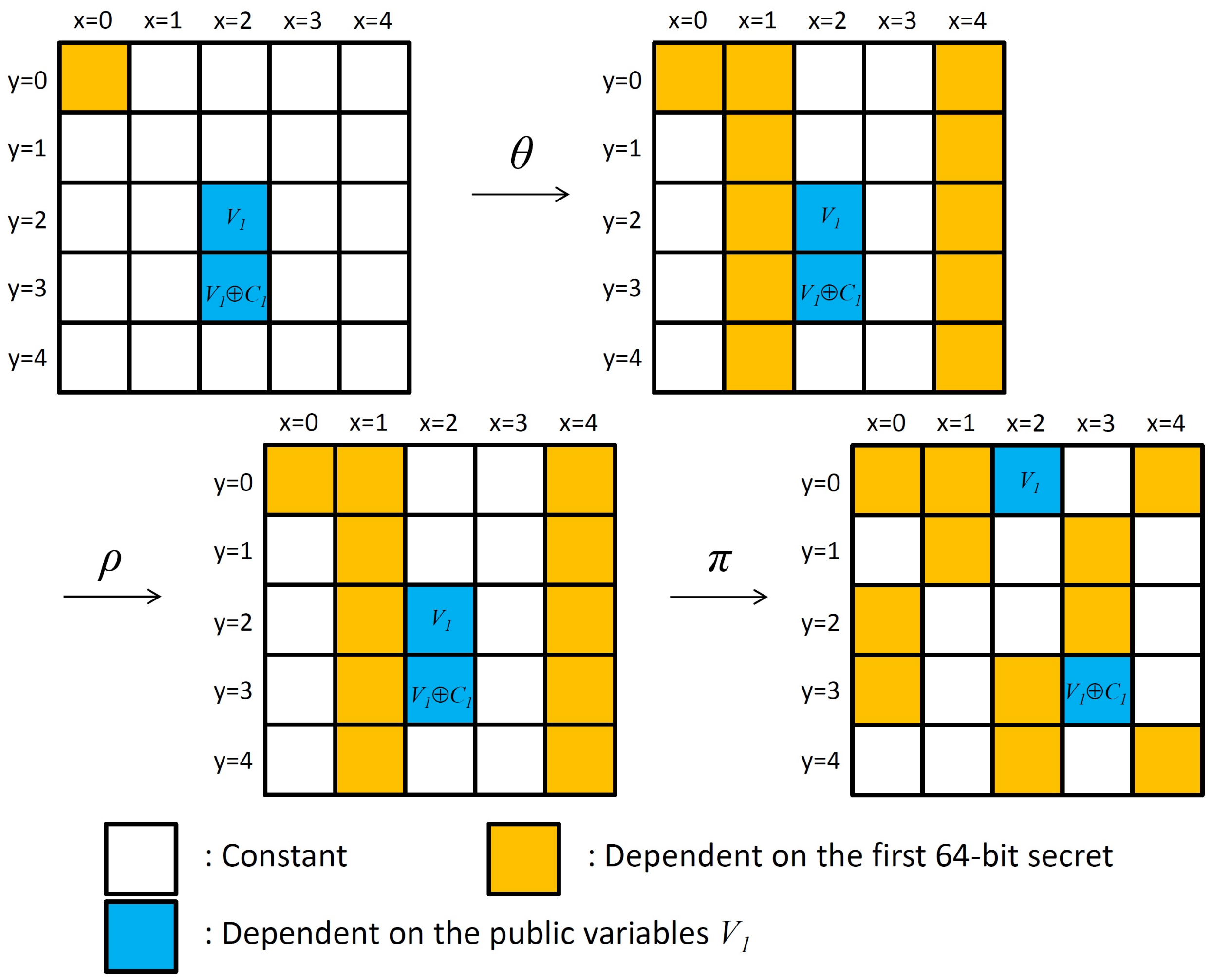


Fig. 101. Transition generated by the first 64-bit secret value A[0,0] [25]: the ordering of lanes is defined according to Fig. 89

(a) Calculate the cube sums after 6 rounds for all the output bits. Store the cube sums in a sorted list L, next to the value of the corresponding A[0,0].

The online phase of the divide-and-conquer key recovery attack [25] with time complexity 2^{32} is as follows:

1. Request the outputs for the 2^{32} messages that make up the chosen cube (using the same constant as in the offline phase).
2. Calculate the cube sums for the output bits and search them in L.
3. For each match in L, retrieve A[0,0] and store all of its possible values.

6.7 Divide-and-Conquer Key Recovery Attack on 7-round Keccak-based MAC and 7-round Keyak

In the previous subsection, The offline phase of the divide-and-conquer key recovery attack on 6-round Keccak-based MAC [25] requires the time complexity 2^{96} and memory complexity 2^{64}. On the other hand, the online phase of the divide-and-conquer key recovery attack [25] requires only the time complexity 2^{32}. So, there is a large gap between the complexities of online and offline phases. So, [25] considered how to balance those complexities by lowering the offline complexity and increasing the online complexity. In order to understand the balanced attack, we need to know a concept of impact of auxiliary variables as shown Fig. 102. If the constant assigned to A[0,1] is same as the first 64-bit value of the 128-bit secret key assigned to A[0,0], then the transition of the first half round is determined as shown Fig. 102. From the last state of Fig. 102, we can see that there will be no multiplication via χ step between any of the first 64-bit K and the public random variables V_1, because they are not adjacent to each other. So, the cube sums with the 32 random variables V_1 after 6 rounds depend neither on the value of A[0,0], nor on the auxiliary variables of A[0,1]. This observation in [25] gives rise to the balanced attack as follows:

The offline phase of the divide-and-conquer balanced key recovery attack [25] with time complexity 2^{64} and memory complexity 2^{32} is as follows:

1. Set the state bits (which are not cube variables) to zero (or an arbitrary constant). Furthermore, set A[1,0] and the 32 LSBs of A[0,0] to zero (or an arbitrary constant).
2. For each possible value of the 32 MSBs of A[0,0]:
   (a) Calculate the cube sums after 6 rounds for all the output bits. Store the cube sums in a sorted list L, next to the value of the 32 MSBs of A[0,0].

The online phase of the divide-and-conquer balanced key recovery attack [25] with time complexity 2^{66} and memory complexity 2^{32} is as follows:

1. For each possible value of the 32 LSBs of A[0,1]:
   (a) Request the outputs for the 232 messages that make up the chosen cube with the 32 LSBs of A[0,1] set according to Step 1 (setting the same constant values in the state as in the preprocessing).
   (b) Calculate the cube sums for the output bits and search them in L.
   (c) For each match in L, retrieve the 32 MSBs of A[0,0]. Assume that the 32 LSBs of A[0,0] are equal to the 32 LSBs of A[0,1] (the 32 column parities should be zero, as in the offline phase). Then, given the full 64-bit A[0,0], exhaustively search A[1,0] using trial encryptions, and if a trial encryption succeeds, return the full key A[0,0],A[1,0].

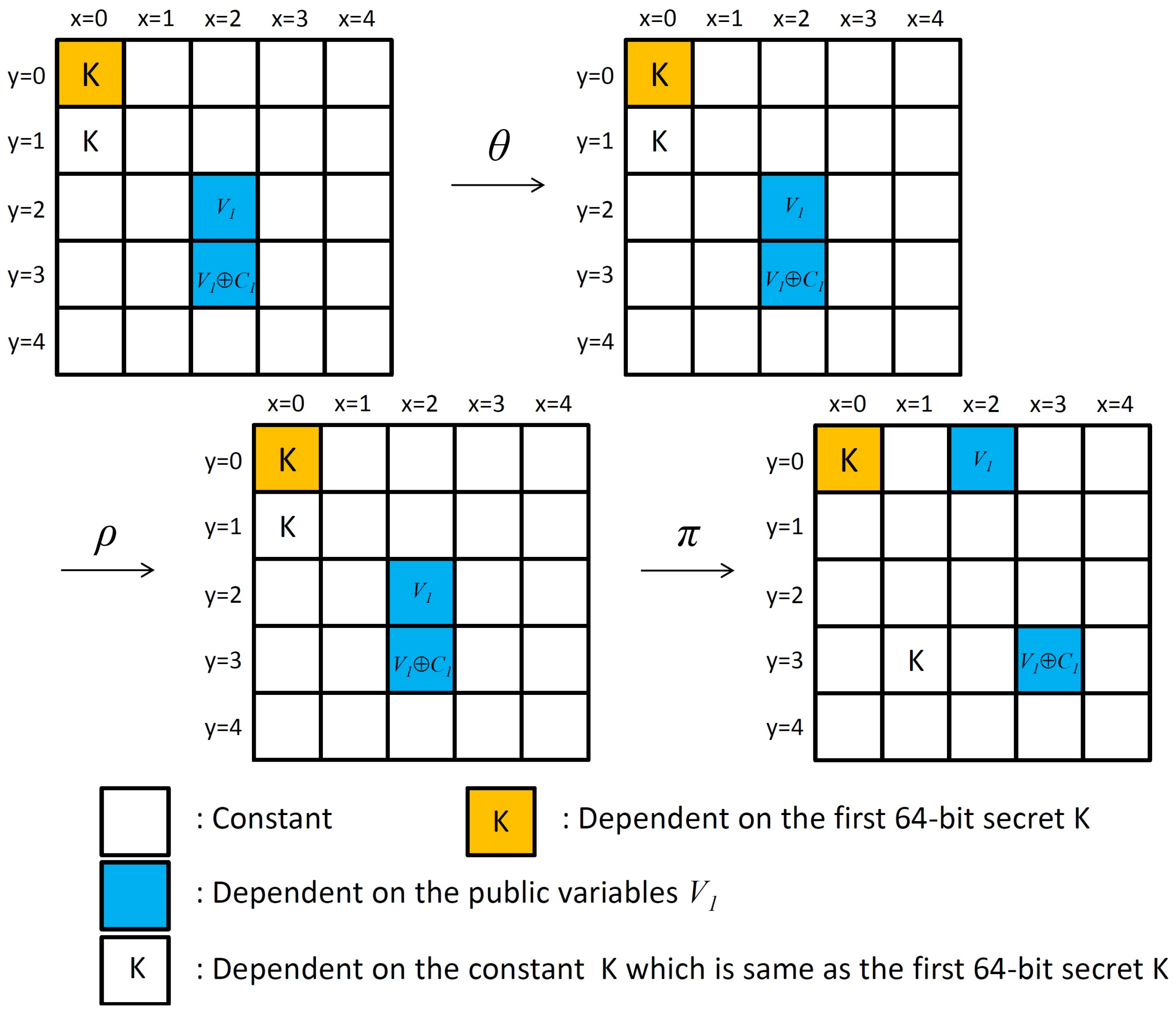

x=0
x=1
x=2
x=3
x=4
y=0
y=1
y=2
y=3
y=4
K
K
V_1
$V_1 \oplus C_1$
θ
ρ
π
: Constant
: Dependent on the first 64-bit secret K
: Dependent on the public variables V_1
: Dependent on the constant K which is same as the first 64-bit secret K

Fig. 102. Transition when the public constant A[0,1] is same as the secret A[0,0] [25]: the ordering of lanes is defined according to Fig. 89

For the key recovery attack on the 7-round Keccak-based MAC, we need 64 random variables [25]. Due to this reason, we need to increase the above 6-round attack by a factor of 2^{32}. Therefore, the data complexity of the 7-round attack is 2^{64}, its time complexity is 2^{97}, and its memory complexity remains 2^{32}.

Using this divide-and-conquer balanced key recovery approach, [25] suceededed in finding the secret key of 7-round Keyak with the time complexity 2^{76}, the data complexity 2^{75}, and the memory complexity 2^{43} of words.

References

1. K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang, *Preimages for step-reduced SHA-2*, ASIACRYPT 2009, LNCS 5912, pp. 578-597, 2009.
2. J-P. Aumasson and W. Meier, *Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi*, http://aumasson.jp/data/papers/AM09.pdf, 2009.
3. J-P. Aumasson, I. Dinur, W. Meier, and A. Shamir, *Cube testers and key recovery attacks on reduced-round md6 and trivium*, FSE 2009, LNCS , pp. 1-22, 2009.
4. M. Bellare, R. Canetti, and H. Krawczyk, *Keying hash functions for message authentication*, Advances in Cryptology – CRYPTO 1996, LNCS 1109, pp. 1-15, 1996.
5. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, http://keccak.noekeon.org/crunchy_contest.html.
6. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Keccak specifications*, Oct 2008, http://keccak.noekeon.org/Keccak-specifications.pdf.
7. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, *On the Indifferentiability of the Sponge Construction*, EUROCRYPT 2008, L 4965, Springer-Verlag, pp. 181-197, 2008.
8. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Cryptographic sponge functions*, Jan 2011, http://sponge.noekeon.org/CSF-0.1.pdf.
9. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *The Keccak SHA-3 submission*, Submission to NIST (Round 3), 2011.
10. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *The Keccak reference*, Submission to NIST (Round 3), 2011.
11. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer *Keyak*, http://keyak.noekeon.org/.
12. D. J. Bernstein, *Second preimages for 6 (7? (8??)) rounds of Keccak?*, NIST mailing list http://cr.yp.to/hash/keccak-20101127.txt, 2010.
13. Biere, A. 2009 P{re,i}coSAT@SC09. SAT 2009 competitive events booklet http://fmv.jku.at/precosat/.
14. A. Biryukov, M. Lamberger, F. Mendel, and I. Nikolić, *Second-Order Differential Collisions for Reduced SHA-256*, ASIACRYPT 2011, LNCS 7073, pp. 270-287, 2011.
15. C. Boura, A. Canteaut, and C. D. Cannière, *Higher-order differential properties of Keccak and Luffa**, https://eprint.iacr.org/2010/589.pdf, 2011.
16. D. Cannière and C. Rechberger, *Finding SHA-1 Characteristics: General Results and Applications*, ASIACRYPT 2006, LNCS 4284, pp. 1-20, 2006.
17. D. Chang, *Security Analysis of Hash Domain and Codomain Extensions*, PhD Thesis, 2008.
18. D. Chang and M. Nandi, *Improved Indifferentiability Security Analysis of chopMD Hash Function*, FSE 2008, LNCS 5086, pp. 429-443, 2008.
19. D. Chang, A. Kumar, P. Morawiecki, and S. K. Sanadhya, *1st and 2nd Preimage Attacks on 7, 8, and 9 Rounds of Keccak-224,256,384,512*, NIST SHA-3 workshop, 2014.
20. I. B. Damgård, *A design principle for hash functions*, CRYPTO 1989, LNCS 435, Springer-Verlag, pp. 416-427, 1990.
21. J. Daemen, *Response to [12]*, http://ehash.iaik.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt, 2010.
22. I. Dinur, O. Dunkelman, and A. Shamir, *New Attacks on Keccak-224 and Keccak-256*, FSE 2012, LNCS 7549, pp. 442-461.
23. I. Dinur, O. Dunkelman, and A. Shamir, *Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials*, FSE 2013, LNCS 8424, pp. 219-240, 2014.
24. I. Dinur and G. Leurent, *Improved Generic Attacks against Hash-Based MACs and HAIFA*, CRYPTO 2014, LNCS 8616, pp. 14917168, 2014.
25. I. Dinur, P. Morawiecki, J. Pieprzyk, M. Srebrny, and M. Straus, *Cube Attacks and Cube-attack-like Cryptanalysis on the Round-reduced Keccak Sponge Function*, Cryptology ePrint Archive: Report 2014/736, 2014.
26. I. Dinur and A. Shamir, *An improved algebraic attack on Hamsi-256*, FSE 2011, LNCS 6733, pp. 88-106, 2011.
27. A. Duc, J. Guo, T. Peyrin, and L. Wei, *Unaligned Rebound Attack: Application to Keccak*, FSE 2012, LNCS 7549, pp. 402-421, 2012.

28. M. Duan and X. Lai, *Improved Zero-sum Distinguisher for Full Round Keccak-f Permutation*, https://eprint.iacr.org/2011/023.pdf, 2011.

29. M. Duan and X. Lai, *Improved Zero-sum Distinguisher for Full Round Keccak-f Permutation*, Chinese Science Bulletin, Vol. 57, No. 6, pp. 694-697, Feb., 2012.

30. M. Dworkin, *Domain Extensions*, http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/documents/dworkin_domain_ext.pdf, 2014.

31. N. Eén and N. Sörensson, *Translating Pseudo-Boolean Constraints into SAT*, Journal on Satisfiability, Boolean Modeling and Computation 2, pp. 11726, 2006.

32. M. Eichlseder, F. Mendel, and M. Schläffer, *Branching Heuristics in Differential Collision Search with Applications to SHA-512*, FSE 2014, http://eprint.iacr.org/2014/302.

33. Grid'5000, www.grid5000.fr.

34. J. Guo, S. Ling, C. Rechberger, and H. Wang, *Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2*, ASIACRYPT 2010, LNCS 6477, pp. 56-75, 2010.

35. J. Guo, T. Peyrin, Y. Sasaki, and L. Wang, *Updates on Generic Attacks against HMAC and NMAC*, CRYPTO 2014, LNCS 8616, pp. 131-148, 2014.

36. A. Joux, *Multicollisions in iterated hash functions. Application to cascaded constructions*, CRYPTO 2004, LNCS 3152, pp. 306-316, 2004.

37. J. Kelsey and B. Schneier, *Second preimages on n-bit hash functions for much less than 2^n work*, EUROCRYPT 2005, LNCS 3494, pp. 474-490, 2005.

38. D. Khovratovich, C. Rechberger, and A. Savelieva, *Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family*, FSE 2012, LNCS, pp. 244-263, 2012.

39. L.R. Knudsen and V. Rijmen, *Known-key distinguishers for some block ciphers*, ASIACRYPT 2007, LNCS 4833, pp. 315-324, 2007.

40. X. Lai, *Higher order derivatives and differential cryptanalysis*, In Proc. Symposium on Communication, Coding and Cryptography17, in honor of J. L. Massey on the occasion of his 60th birthday. Kluwer Academic Publishers, 1994.

41. G. Leurent, T. Peyrin, and L. Wang, *New Generic Attacks Against Hash-based MACs*, ASIACRYPT 2013, LNCS 8270, pp. 11720, 2013.

42. F. Mendel, T. Nad, and M. Schläffer, *Finding SHA-2 Characteristics: Searching through a Minefield of Constradictions*, ASIACRYPT 2011, LNCS 7073, pp. 288-307, 2011.

43. F. Mendel, T. Nad, and M. Schläffer, *Improving Local Collisions: New Attacks on Reduced SHA-256*, EUROCRYPT 2013, LNCS 7881, pp. 262-278, 2013.

44. R. C. Merkle, *One way hash functions and DES*, CRYPTO 1989, LNCS 435, Springer-Verlag, pp. 428-446, 1990.

45. P. Morawiecki and M. Srebrny, *A SAT-based preimage analysis of reduced Keccak hash functions*, https://eprint.iacr.org/2010/285.pdf, 2010.

46. M. Naya-Plasencia, A. Röck, and W. Meier., *Practical Analysis of Reduced-Round Keccak*, INDOCRYPT 2011, LNCS 7107, pp. 23617254, 2011.

47. NIST, *Secure Hash Standard (SHS)*, FIPS PUB 180-4, http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

48. NIST, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, The Draft FIPS PUB 202, http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf.

49. V. Nossum, *SAT-based preimage attacks on SHA-1*, Master Thesis, https://www.duo.uio.no/bitstream/handle/10852/34912/thesis-output.pdf?sequence=1, 2012.

50. B. Preneel and P.C. van Oorschot, *On the Security of Two MAC Algorithms*, EUROCRYPT 1996, LNCS 1070, pp. 19-32, 1996.

51. T. Ristenpart, H. Shacham, and T/ Shrimpton, *Careful with Composition: Limitations of the Indifferentiability Framework*, EUROCRYPT 2011, LNCS 6632, pp. 487-506, 2011.

52. W. Stallings, *Chapter 12 of Cryptography and Network Security*, www.cise.ufl.edu/ñemo/crypto/stallings/ch11.ppt.

53. H. Yu and D. Bai, *Boomerang Attack on Step-Reduced SHA-512*, Inscrypt 2014, to appear, http://eprint.iacr.org/2014/945.