Cryptanalysis of Round Reduced KECCAK

Nikhil Mittal

Under supervision of

Prof. Manindra Agrawal and Dr. Shashank Singh

IIT Kanpur

nickedes@cse.iitk.ac.in

June 9, 2019

Hash Function

- A hash function is of the form $H : \{0,1\}^* \to \{0,1\}^n$.

Hash Function

- A hash function is of the form $H : \{0,1\}^* \rightarrow \{0,1\}^n$.
- It is a deterministic function.
- It takes as input an arbitrary size string and outputs a fixed size (n) string.

Hash Function

- A hash function is of the form $H : \{0,1\}^* \rightarrow \{0,1\}^n$.
- It is a deterministic function.
- It takes as input an arbitrary size string and outputs a fixed size (n) string.
- It is used in cryptographic applications such as Authentication, Digital Signatures and Integrity etc..

Cryptographic Hash Functions

- Cryptographic applications require Hash functions to satisfy the following conditions:

Cryptographic Hash Functions

- Cryptographic applications require Hash functions to satisfy the following conditions:
    - **Efficiency**: Given a message m, it is easy to compute its Hash i.e. $H(m)$.

Cryptographic Hash Functions

- Cryptographic applications require Hash functions to satisfy the following conditions:
  - **Efficiency**: Given a message m, it is easy to compute its Hash i.e. $H(m)$.
  - **Preimage Resistance**: Given $H(m)$, it is computationally hard to find the message m.

Cryptographic Hash Functions

- Cryptographic applications require Hash functions to satisfy the following conditions:
    - **Efficiency**: Given a message m, it is easy to compute its Hash i.e. $H(m)$.
    - **Preimage Resistance**: Given $H(m)$, it is computationally hard to find the message m.
    - **Second-preimage Resistance**: Given a message m, it is computationally hard to find another message m' such that $H(m) = H(m')$.

Cryptographic Hash Functions

- Cryptographic applications require Hash functions to satisfy the following conditions:
  - **Efficiency**: Given a message m, it is easy to compute its Hash i.e. $H(m)$.
  - **Preimage Resistance**: Given $H(m)$, it is computationally hard to find the message m.
  - **Second-preimage Resistance**: Given a message m, it is computationally hard to find another message m' such that $H(m) = H(m')$.
  - **Collision Resistance**: It is computationally hard to find two messages m and m' such that $H(m) = H(m')$.

Cryptographic Hash Functions

- Cryptographic applications require Hash functions to satisfy the following conditions:
  - **Efficiency**: Given a message m, it is easy to compute its Hash i.e. $H(m)$.
  - **Preimage Resistance**: Given $H(m)$, it is computationally hard to find the message m.
  - **Second-preimage Resistance**: Given a message m, it is computationally hard to find another message m' such that $H(m) = H(m')$.
  - **Collision Resistance**: It is computationally hard to find two messages m and m' such that $H(m) = H(m')$.
- Hash functions having the above properties are referred to as cryptographic hash functions.

Need For SHA-3

- MD5, SHA-1, SHA-2 are very popular hash functions and are widely used.

Need For SHA-3

- MD5, SHA-1, SHA-2 are very popular hash functions and are widely used.
- In year 2005, First practical collision attacks were found on:
  - MD5, SHA-0 and SHA-1 by Xiaoyun Wang *et al.*

Need For SHA-3

- MD5, SHA-1, SHA-2 are very popular hash functions and are widely used.
- In year 2005, First practical collision attacks were found on:
  - MD5, SHA-0 and SHA-1 by Xiaoyun Wang *et al.*
- National Institute of Standards and Technology (NIST) was worried about the security of hash functions.
- Though by that time SHA-2 family of hash functions was standardized.

Need For SHA-3

- MD5, SHA-1, SHA-2 are very popular hash functions and are widely used.
- In year 2005, First practical collision attacks were found on:
  - MD5, SHA-0 and SHA-1 by Xiaoyun Wang *et al.*
- National Institute of Standards and Technology (NIST) was worried about the security of hash functions.
- Though by that time SHA-2 family of hash functions was standardized.
- SHA-2 was also based on Merkle-Damgard construction like SHA-0, SHA-1.
- There was a possibility that it could also be attacked in a similar fashion.

SHA-3 Competition

- With this thought, in the year 2006 NIST decided to hold a competition for the next secure hash function.

SHA-3 Competition

- With this thought, in the year 2006 NIST decided to hold a competition for the next secure hash function.
- In 2008, NIST announced a competition for the Secure Hash Algorithm-3 (SHA-3).

SHA-3 Competition

- With this thought, in the year 2006 NIST decided to hold a competition for the next secure hash function.
- In 2008, NIST announced a competition for the Secure Hash Algorithm-3 (SHA-3).
- In the year 2012, NIST announced KECCAK as the winner of the competition among the five finalists viz. BLAKE, Grøstl, JH, KECCAK and Skein.

SHA-3 Competition

- With this thought, in the year 2006 NIST decided to hold a competition for the next secure hash function.
- In 2008, NIST announced a competition for the Secure Hash Algorithm-3 (SHA-3).
- In the year 2012, NIST announced KECCAK as the winner of the competition among the five finalists viz. BLAKE, Grøstl, JH, KECCAK and Skein.
- Since 2015, KECCAK has been standardized as SHA-3 by NIST.

KECCAK

- KECCAK hash function is based on sponge construction.
- SHA-3 family of hash functions is based on KECCAK.
- The SHA-3 family provides four hash functions:
  - SHA3-224, SHA3-256, SHA3-384 and SHA3-512.

KECCAK

- KECCAK hash function is based on sponge construction.
- SHA-3 family of hash functions is based on KECCAK.
- The SHA-3 family provides four hash functions:
  - SHA3-224, SHA3-256, SHA3-384 and SHA3-512.
- KECCAK's excellent resistance towards crypt-analytic attacks is one of the main reasons for its selection by NIST.
- The algorithm is a good mixture of linear as well as non-linear operations.

Sponge Construction

- A sponge construction consists of:
  - Permutation function f,
  - Parameter "rate" r, and
  - Padding rule pad.

Sponge Construction

- A sponge construction consists of:
  - Permutation function f,
  - Parameter "rate" r, and
  - Padding rule pad.
  - This construction produces a sponge function that takes as input a bit string M and generates a string of length l.

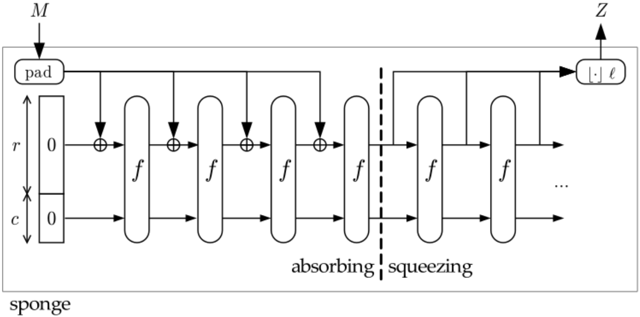


Figure: The sponge construction

KECCAK-p Permutation

- The function f in the sponge construction is denoted by KECCAK-$f[b]$.
- b is the length of input string.
- Internally, KECCAK-$f[b]$ consists of a round function p which is applied n_r number of times.
- KECCAK-$f[b]$ function is specialization of KECCAK-$p[b, n_r]$.

KECCAK State

- The state input to KECCAK-$f[b]$ consists of b bits.
- The state is divided into slices.
- Each slice is of fixed size i.e., 25 bits.
- A state S, which is a b-bit string, in KECCAK is usually denoted by a 3-dimensional grid of size $(5 \times 5 \times w)$.

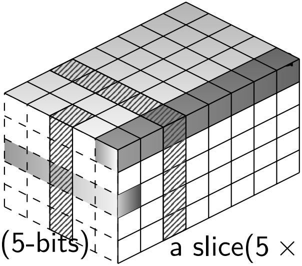


Figure: The KECCAK State

Round Function of KECCAK-*p*

- The round function *p* in KECCAK comprises of 5 step mappings.
- The KECCAK state undergoes some transformations specified by the step mapping.
- These step mappings are called θ, ρ, π, χ *and* ι.
- These transformations are applied in sequence.
- Now, we will describe these 5 step mappings in detail.

θ step mapping

- XORs each bit in the state with the parities of two neighboring columns.

θ step mapping

- XORs each bit in the state with the parities of two neighboring columns.
- For bit position $(x,\ y,\ z)$, one column is $((x-1) \bmod 5,\ z)$ and the other is $((x+1) \bmod 5,\ (z-1) \bmod w)$.

θ step mapping

- XORs each bit in the state with the parities of two neighboring columns.
- For bit position $(x,\ y,\ z)$, one column is $((x-1) \bmod 5,\ z)$ and the other is $((x+1) \bmod 5,\ (z-1) \bmod w)$.
- If we have A as the input state to θ then the output state B is:

$$B[x,\ y,\ z] = A[x,\ y,\ z] \bigoplus P[(x-1) \bmod 5,\ z] \bigoplus P[(x+1) \bmod 5,\ (z-1) \bmod w] \tag{1}$$

θ step mapping

- XORs each bit in the state with the parities of two neighboring columns.
- For bit position $(x,\ y,\ z)$, one column is $((x-1) \bmod 5,\ z)$ and the other is $((x+1) \bmod 5,\ (z-1) \bmod w)$.
- If we have A as the input state to θ then the output state B is:

$$
\begin{aligned}
B[x,\ y,\ z] = A[x,\ y,\ z] \bigoplus P[(x-1) \bmod 5,\ z] \\
\bigoplus P[(x+1) \bmod 5,\ (z-1) \bmod w]
\end{aligned}
\tag{1}
$$

- $P[x,\ z]$ represents the parity of the column $(x,\ z)$.

$$
P[x,\ z] = \bigoplus_{y=0}^{4} A[x,\ y,\ z]
$$

ρ step mapping

- ρ (**rho**): This step rotates each lane by a constant value towards the MSB.

ρ step mapping

- ρ (**rho**): This step rotates each lane by a constant value towards the MSB.
- If we have A as the input state to ρ then the output state B is:
    - $B[x, y, z] = A[x, y, z + \rho(x, y) \bmod w]$

ρ step mapping

- ρ (**rho**): This step rotates each lane by a constant value towards the MSB.
- If we have A as the input state to ρ then the output state B is:
  - $B[x, y, z] = A[x, y, z + \rho(x, y) \bmod w]$
- $\rho(x, y)$ is the constant for lane (x, y).
- The constant value $\rho(x, y)$ is specified for each lane in the construction of KECCAK.

ρ step mapping

- ρ (**rho**): This step rotates each lane by a constant value towards the MSB.
- If we have A as the input state to ρ then the output state B is:
  - $B[x, y, z] = A[x, y, z + \rho(x, y) \bmod w]$
- $\rho(x, y)$ is the constant for lane (x, y).
- The constant value $\rho(x, y)$ is specified for each lane in the construction of KECCAK.
- ρ is a linear step mapping.

π step mapping

- π (**pi**): It permutes the positions of lanes.
- The new position of a lane is determined by a matrix,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2}$$

  where (x', y') is the position of lane (x, y) after π step.
- π is a linear step mapping.

χ step mapping

- χ (**chi**): Each bit in the original state is XOR-ed with a non-linear function of next two bits in the same row.

$$B[x,\ y,\ z] = A[x,\ y,\ z] \oplus \\ ((A[(x+1) \bmod 5,\ y,\ z] \oplus 1) \cdot A[(x+2) \bmod 5,\ y,\ z])\,. \quad (3)$$

χ step mapping

- χ (**chi**): Each bit in the original state is XOR-ed with a non-linear function of next two bits in the same row.

$$B[x,\ y,\ z] = A[x,\ y,\ z] \oplus \\ ((A[(x+1) \bmod 5,\ y,\ z] \oplus 1) \cdot A[(x+2) \bmod 5,\ y,\ z])\,. \tag{3}$$

- χ is the only non-linear operation among the 5 step mappings in KECCAK.

ι step mapping

- ι (**iota**): This step mapping only modifies the $(0,\ 0)$ lane depending on the round number.
- If we have A as the input state to ι then the output state B is:

$$B[0,\ 0] = A[0,\ 0] \oplus RC_i, \tag{4}$$

where RC_i is round constant that depends on the round number.

ι step mapping

- ι (**iota**): This step mapping only modifies the $(0,\ 0)$ lane depending on the round number.
- If we have A as the input state to ι then the output state B is:

$$B[0,\ 0] = A[0,\ 0] \oplus RC_i, \tag{4}$$

where RC_i is round constant that depends on the round number.

- The remaining 24 lanes remain unaffected.

ι step mapping

- ι (**iota**): This step mapping only modifies the $(0,\ 0)$ lane depending on the round number.
- If we have A as the input state to ι then the output state B is:

$$B[0,\ 0] = A[0,\ 0] \oplus RC_i, \tag{4}$$

  where RC_i is round constant that depends on the round number.
- The remaining 24 lanes remain unaffected.
- All the rounds are identical but the symmetry is destroyed by this step due to the addition of a round constant to a particular lane.

Specification of KECCAK-$p[b, n_r]$

- Round in KECCAK is given by:
  - $\texttt{Round}(A,\ i_r) = \iota(\chi(\pi(\rho(\theta(A)))),\ i_r)$
- It consists of n_r number of rounds of $\texttt{Round}(A,\ i_r)$.
- KECCAK-$p[b,\ n_r](S)$
  - Convert S into a state array A
  - For i_r from 0 to $n_r - 1$, let $A = \texttt{Round}(A,\ i_r)$
  - Convert A into string S' of length b
  - Return S'

SHA-3 Hash Function

- The SHA-3 hash function is KECCAK-$p[b,\ 12 + 2 \cdot l]$
- $w = b/25$ and $l = \log_2(w)$
- The value of $b = 1600$, so we have $l = 6$
- Thus the f function in SHA-3 is KECCAK-$p[1600,\ 24]$
- Instances of KECCAK are denoted by KECCAK $[r,\ c]$
- Where $r = 1600 - c$ and the capacity c is chosen to be twice the size of hash output d
- To avoid generic attacks with expected cost below 2^d
- The hash function with output length d is denoted by

$$\text{KECCAK-}d \quad = \quad \text{KECCAK}\,[r := 1600 - 2 \cdot d,\ c := 2 \cdot d] \qquad (5)$$

pad10*1

- The padding rule followed by KECCAK is **pad10*1**
- Rule is that the input string is appended with a 1 bit followed by some number of 0 bits and followed by 1 bit
- The asterisk in the padding rule indicates that 0 bit is either not present or is repeated as required so that the length of output string after padding is a multiple of the block length (i.e. r)

KECCAK[r:=800-384, c:=384]

- KECCAK $[r := 800 - 384, c := 384] =$
  KECCAK-$p\,[800, 24]\,[r := 800 - 384, c := 384]$

KECCAK[r:=800-384, c:=384]

- KECCAK $[r := 800 - 384, c := 384] =$ KECCAK-$p[800, 24]$ $[r := 800 - 384, c := 384]$
- 2-round KECCAK $[r := 800 - 384, c := 384] =$ KECCAK-$p[800, 2]$ $[r := 800 - 384, c := 384]$

Observations

- **Observation 1:** If we know all the bits of a row then we can invert χ for that row. It is depicted below.

$$\boxed{a_0}\boxed{a_1}\boxed{a_2}\boxed{a_3}\boxed{a_4} \xrightarrow{\chi^{-1}} \boxed{a'_0}\boxed{a'_1}\boxed{a'_2}\boxed{a'_3}\boxed{a'_4}$$

Figure: Computation of χ^{-1} for full row

$$a'_i = a_i \oplus (a_{i+1} \oplus 1) \cdot (a_{i+2} \oplus (a_{i+3} \oplus 1) \cdot a_{i+4}) \tag{6}$$

- **Observation 2:** When only one output bit is known after χ step, then we can fix the first output bit to be the same as the input bit and the second bit as 1.

$$\boxed{a_0}\boxed{*}\boxed{*}\boxed{*}\boxed{*} \xrightarrow{\chi^{-1}} \boxed{a_0}\boxed{1}\boxed{*}\boxed{*}\boxed{*}$$

Figure: Computation of χ^{-1} when only 1-bit is known in row

Observations

- **Observation 3:**
- a_i are the output bits of χ and a'_i are the input bits
- Guo *et al.* observed that when 4 out of 5 output bits of χ are known
- Then we can obtain 4 linear relations in terms of a'_i

$$a'_i = a_i \oplus (a_{i+1} \oplus 1) \cdot (a_{i+2} \oplus (a_{i+3} \oplus 1) \cdot a_{i+4}) \tag{7}$$

- If the values of $a_0,\ a_1,\ a_2,\ a_3$ are known
- Using the Equation 7, we can eliminate the expression a_4 from the rest of the equations
- Hence obtain 4 linear relations

Notations

- State is represented by 25 lanes
- Each lane is represented by a variable which is a 32-bit array
- A variable with a number in round bracket "$(\cdot)$" represents the shift of the bits in array towards MSB
- A variable with a number in square bracket "$[\cdot]$" represents the bit value of the variable at that index.
- If there are multiple numbers in the square bracket then it represents the corresponding bit values.

2 rounds of KECCAK[r:=800-384,c:=384]

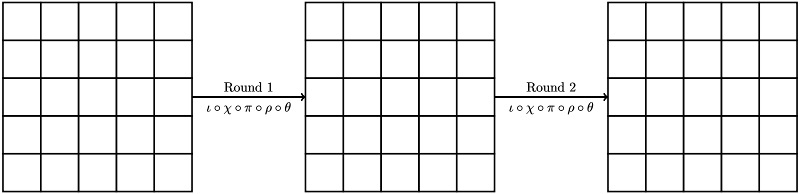


Figure: Two rounds of KECCAK[$r := 800 - 384, c := 384$]

- We will discuss a preimage attack on above structure

Final State of 2-round KECCAK[r:=800-384,c:=384]

- $c = 384 \rightarrow d = 192 \rightarrow$ hash of 6 lanes

y ↑					
4	$\star$	$\star$	$\star$	$\star$	$\star$
3	$\star$	$\star$	$\star$	$\star$	$\star$
2	$\star$	$\star$	$\star$	$\star$	$\star$
1	h_5	$\star$	$\star$	$\star$	$\star$
0	h_0	h_1	h_2	h_3	h_4
	0	1	2	3	4

$x \longrightarrow$

Figure: The Final Hash State for KECCAK$[r := 800 - 384, c := 384]$

- $r = 800 - 384 \rightarrow r = 416 \rightarrow$ Message block of 13 lanes

0	0	0	0	0
0	0	0	0	0
a_1	b_1	c_2	0	0
a_2	b_2	c_1	d_1	e_1
a_0	b_0	c_0	d_0	e_0

Figure: Setting of Initial State in the Attack

Attack

- Our aim is to find the values of $a_0,\ a_1,\ a_2,\ b_0,\ b_1,\ b_2,\ c_0,\ c_1,\ c_2,$ $d_0,\ d_1$ and $e_0,\ e_1$ variables in the initial state
- Such that they lead to a final state having first six lanes as $h_0,\ h_1,\ h_2,\ h_3,\ h_4$ and h_5
- We follow the basic idea of the attack given by Naya *et al.* in 2011
- 2 rounds of $\textsc{Keccak}[r := 800 - 384,\ c := 384]$
    - Best-known attack, has a time complexity of $O(2^{64})$
    - Based on the idea of Linear structures given by Jian Guo *et al.* in 2016

First round θ step mapping

- θ step mapping diffuses message bits to full state
- Aim : Control the diffusion
- By adding constraints on message bits
- Conditions to make column parity zero:

$$\begin{aligned} a_2 = a_0 \oplus a_1, \quad b_2 = b_0 \oplus b_1, \quad c_2 = c_0 \oplus c_1 \\ d_1 = 0, \quad d_0 = 0 \quad \text{and} \quad e_1 = e_0 \end{aligned} \tag{8}$$

State 1 to State 2

State 1

0	0	0	0	0
0	0	0	0	0
a_1	b_1	c_2	0	0
a_2	b_2	c_1	d_1	e_1
a_0	b_0	c_0	d_0	e_0

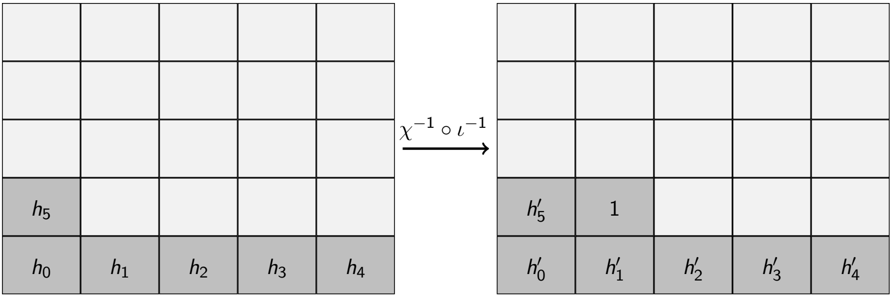
$\xrightarrow{\theta, \pi, \rho}$

State 2

$c_0(30)$	$d_1(23)$	0	0	0
$e_0(27)$	$a_2(4)$	$b_1(10)$	0	0
$b_0(1)$	$c_1(6)$	0	0	0
$d_0(28)$	$e_1(20)$	$a_1(3)$	0	0
$a_0(0)$	$b_2(12)$	$c_2(11)$	0	0

Figure: Preimage attack on 2-round KECCAK[$r := 800 - 384,\ c := 384$]

χ and ι inverse



State 4

Figure: Inversion of hash through $\chi^{-1} \circ \iota^{-1}$

State 4 to State 3

h_5				
h_0	h_1	h_2	h_3	h_4

State 4

$$\xrightarrow[\pi^{-1},\,\rho^{-1}]{\iota^{-1},\,\chi^{-1}}$$

				$h'_4(18)$
			$h'_3(11)$	
		$h'_2(21)$		
	$h'_1(20)$			1
$h'_0(0)$			$h'_5(4)$	

State 3

State 1 to 4

State 1

0	0	0	0	0
0	0	0	0	0
a_1	b_1	c_2	0	0
a_2	b_2	c_1	d_1	e_1
a_0	b_0	c_0	d_0	e_0

$\xrightarrow{\theta, \pi, \rho}$

State 2

$c_0(30)$	$d_1(23)$	0	0	0
$e_0(27)$	$a_2(4)$	$b_1(10)$	0	0
$b_0(1)$	$c_1(6)$	0	0	0
$d_0(28)$	$e_1(20)$	$a_1(3)$	0	0
$a_0(0)$	$b_2(12)$	$c_2(11)$	0	0

$\xrightarrow{\chi, \iota, \theta}$

State 3

				$h'_4(18)$
			$h'_3(11)$	
		$h'_2(21)$		
	$h'_1(20)$			1
$h'_0(0)$			$h'_5(4)$	

$\xrightarrow[\pi^{-1}, \rho^{-1}]{\iota^{-1}, \chi^{-1}}$

State 4

h_5				
h_0	h_1	h_2	h_3	h_4

Figure: Preimage attack on 2-round KECCAK$[r := 800 - 384,\ c := 384]$

State 2 to State 3

$c_0(30)$	$d_1(23)$	0	0	0
$e_0(27)$	$a_2(4)$	$b_1(10)$	0	0
$b_0(1)$	$c_1(6)$	0	0	0
$d_0(28)$	$e_1(20)$	$a_1(3)$	0	0
$a_0(0)$	$b_2(12)$	$c_2(11)$	0	0

State 2

$$\xrightarrow{\chi, \iota, \theta}$$

				$h'_4(18)$
			$h'_3(11)$	
		$h'_2(21)$		
	$h'_1(20)$			1
$h'_0(0)$			$h'_5(4)$	

State 3

Figure: Intermediate States in 2-round preimage attack on KECCAK-384

Description of Attack

- Fix d_0, d_1 as constants (with condition that $d_0 = d_1$).

Description of Attack

- Fix d_0, d_1 as constants (with condition that $d_0 = d_1$).
- In state 2, we have $11 \cdot 32$ free variables.

Description of Attack

- Fix d_0, d_1 as constants (with condition that $d_0 = d_1$).
- In state 2, we have $11 \cdot 32$ free variables.
- And we need to satisfy $7 \cdot 32$ Boolean equations and $4 \cdot 32$ column parity conditions.

Description of Attack

- Fix d_0, d_1 as constants (with condition that $d_0 = d_1$).
- In state 2, we have $11 \cdot 32$ free variables.
- And we need to satisfy $7 \cdot 32$ Boolean equations and $4 \cdot 32$ column parity conditions.
- So, we expect a solution.

Description of Attack

- Fix d_0, d_1 as constants (with condition that $d_0 = d_1$).
- In state 2, we have $11 \cdot 32$ free variables.
- And we need to satisfy $7 \cdot 32$ Boolean equations and $4 \cdot 32$ column parity conditions.
- So, we expect a solution.
- We do find the possible solution subspace.

Description of Attack

- In state 3, the values of i^{th}-slice depend on the $(i-1)^{\text{th}}$ and i^{th}-slice of state 2.

Description of Attack

- In state 3, the values of i^{th}-slice depend on the $(i-1)^{\text{th}}$ and i^{th}-slice of state 2.

- We first find the set of input message bits which satisfy the small collection of consecutive slices of state 3.

Description of Attack

- In state 3, the values of i^{th}-slice depend on the $(i-1)^{\text{th}}$ and i^{th}-slice of state 2.
- We first find the set of input message bits which satisfy the small collection of consecutive slices of state 3.
- We then merge the solutions to find message bits which satisfy large collection of consecutive slices.

Possible solutions for groups of 3 slices

- Consider a group of 3 slices (for example take the first 3 slices).

Possible solutions for groups of 3 slices

- Consider a group of 3 slices (for example take the first 3 slices).
- It contains the following message bits
  - $a_0[0, 1, 2]$, $a_1[3, 4, 5]$, $a_2[4, 5, 6]$
  - $b_0[1, 2, 3]$, $b_1[10, 11, 12]$, $b_2[12, 13, 14]$
  - $c_0[30, 31, 0]$, $c_1[6, 7, 8]$, $c_2[11, 12, 13]$
  - $e_0[27, 28, 29]$, $e_1[20, 21, 22]$

Possible solutions for groups of 3 slices

- Consider a group of 3 slices (for example take the first 3 slices).
- It contains the following message bits
  - $a_0[0, 1, 2]$, $a_1[3, 4, 5]$, $a_2[4, 5, 6]$
  - $b_0[1, 2, 3]$, $b_1[10, 11, 12]$, $b_2[12, 13, 14]$
  - $c_0[30, 31, 0]$, $c_1[6, 7, 8]$, $c_2[11, 12, 13]$
  - $e_0[27, 28, 29]$, $e_1[20, 21, 22]$
- Once we fix these message bits in the State 2, the slice 1 and slice 2 of State 3 get fixed

Possible solutions for groups of 3 slices

- Consider a group of 3 slices (for example take the first 3 slices).
- It contains the following message bits
  - $a_0[0, 1, 2]$, $a_1[3, 4, 5]$, $a_2[4, 5, 6]$
  - $b_0[1, 2, 3]$, $b_1[10, 11, 12]$, $b_2[12, 13, 14]$
  - $c_0[30, 31, 0]$, $c_1[6, 7, 8]$, $c_2[11, 12, 13]$
  - $e_0[27, 28, 29]$, $e_1[20, 21, 22]$
- Once we fix these message bits in the State 2, the slice 1 and slice 2 of State 3 get fixed
- Furthermore there is no dependency between these message bits

Possible solutions for groups of 3 slices

- Consider a group of 3 slices (for example take the first 3 slices).
- It contains the following message bits
  - $a_0[0,1,2]$, $a_1[3,4,5]$, $a_2[4,5,6]$
  - $b_0[1,2,3]$, $b_1[10,11,12]$, $b_2[12,13,14]$
  - $c_0[30,31,0]$, $c_1[6,7,8]$, $c_2[11,12,13]$
  - $e_0[27,28,29]$, $e_1[20,21,22]$
- Once we fix these message bits in the State 2, the slice 1 and slice 2 of State 3 get fixed
- Furthermore there is no dependency between these message bits
- Thus the total possible solutions for this 3-slice $= 2^{33-2\cdot 7} = 2^{19}$

Possible solutions for groups of 6 slices

- This is obtained by merging two groups of 3 slices

Possible solutions for groups of 6 slices

- This is obtained by merging two groups of 3 slices
- Consider, for example, the first two 3-slices (first 6 slices). It contains the following message bits:
  - $a_0[0-5]$, $a_1[3-8]$, $a_2[4-9]$
  - $b_0[1-6]$, $b_1[10-15]$, $b_2[12-17]$
  - $c_0[30-3]$, $c_1[6-11]$, $c_2[11-16]$
  - $e_0[27-0]$, $e_1[20-25]$

Possible solutions for groups of 6 slices

- This is obtained by merging two groups of 3 slices
- Consider, for example, the first two 3-slices (first 6 slices). It contains the following message bits:
  - $a_0[0-5]$, $a_1[3-8]$, $a_2[4-9]$
  - $b_0[1-6]$, $b_1[10-15]$, $b_2[12-17]$
  - $c_0[30-3]$, $c_1[6-11]$, $c_2[11-16]$
  - $e_0[27-0]$, $e_1[20-25]$
- During merging, we get to compute the bit values of slice 3 of the State 3 as well.

Possible solutions for groups of 6 slices

- This is obtained by merging two groups of 3 slices
- Consider, for example, the first two 3-slices (first 6 slices). It contains the following message bits:
  - $a_0[0-5]$, $a_1[3-8]$, $a_2[4-9]$
  - $b_0[1-6]$, $b_1[10-15]$, $b_2[12-17]$
  - $c_0[30-3]$, $c_1[6-11]$, $c_2[11-16]$
  - $e_0[27-0]$, $e_1[20-25]$
- During merging, we get to compute the bit values of slice 3 of the State 3 as well.
- Since, we already have the correct bit values of slice 3 of the State 3, and there is dependency between the above message bit variables, we end up having total possible solutions $= 2^{2\cdot 19-2-7} = 2^{29}$.
- There is dependency between bits $a_0[4,5], a_1[4,5]$ and $a_2[4,5]$.

Possible solutions for groups of 12 slices

- Similar to the case of 6-slice, the solution for a 12-slice is obtained by merging two 6-slices

Possible solutions for groups of 12 slices

- Similar to the case of 6-slice, the solution for a 12-slice is obtained by merging two 6-slices
- Consider, for example, the first 12 slices. It contain the following message bits:
  - $a_0[0-11]$, $a_1[3-14]$, $a_2[4-15]$
  - $b_0[1-12]$, $b_1[10-21]$, $b_2[12-23]$
  - $c_0[30-9]$, $c_1[6-17]$, $c_2[11-22]$
  - $e_0[27-6]$, $e_1[20-31]$

Possible solutions for groups of 12 slices

- Similar to the case of 6-slice, the solution for a 12-slice is obtained by merging two 6-slices
- Consider, for example, the first 12 slices. It contain the following message bits:
  - $a_0[0-11]$, $a_1[3-14]$, $a_2[4-15]$
  - $b_0[1-12]$, $b_1[10-21]$, $b_2[12-23]$
  - $c_0[30-9]$, $c_1[6-17]$, $c_2[11-22]$
  - $e_0[27-6]$, $e_1[20-31]$
- As before, here we again get the values of slice 6 of State 3

Possible solutions for groups of 12 slices

- Similar to the case of 6-slice, the solution for a 12-slice is obtained by merging two 6-slices
- Consider, for example, the first 12 slices. It contain the following message bits:
  - $a_0[0-11]$, $a_1[3-14]$, $a_2[4-15]$
  - $b_0[1-12]$, $b_1[10-21]$, $b_2[12-23]$
  - $c_0[30-9]$, $c_1[6-17]$, $c_2[11-22]$
  - $e_0[27-6]$, $e_1[20-31]$
- As before, here we again get the values of slice 6 of State 3
- Similar to 6-slice, the bit variables are not independent
- The bit variables $a_0, a_1, a_2[6,7,8,9], b_0, b_1, b_2[12]$, $e_0[27-31]$ and $e_1[27-31]$ are dependent

Possible solutions for groups of 12 slices

- Similar to the case of 6-slice, the solution for a 12-slice is obtained by merging two 6-slices
- Consider, for example, the first 12 slices. It contain the following message bits:
    - $a_0[0-11]$, $a_1[3-14]$, $a_2[4-15]$
    - $b_0[1-12]$, $b_1[10-21]$, $b_2[12-23]$
    - $c_0[30-9]$, $c_1[6-17]$, $c_2[11-22]$
    - $e_0[27-6]$, $e_1[20-31]$
- As before, here we again get the values of slice 6 of State 3
- Similar to 6-slice, the bit variables are not independent
- The bit variables $a_0, a_1, a_2[6,7,8,9], b_0, b_1, b_2[12]$, $e_0[27-31]$ and $e_1[27-31]$ are dependent
- Hence, the total possible solutions $= 2^{2\cdot 29-10-7} = 2^{41}$

Possible solutions for groups of 24 slices

- This is obtained by merging two groups of 12 slices

Possible solutions for groups of 24 slices

- This is obtained by merging two groups of 12 slices
- For example, consider the first 24 slices i.e.,
  - $a_0[0-23]$, $a_1[3-26]$, $a_2[4-27]$
  - $b_0[1-24]$, $b_1[10-1]$, $b_2[12-3]$
  - $c_0[30-21]$, $c_1[6-29]$, $c_2[11-2]$
  - $e_0[27-18]$, $e_1[20-11]$

Possible solutions for groups of 24 slices

- This is obtained by merging two groups of 12 slices
- For example, consider the first 24 slices i.e.,
  - $a_0[0-23], \;\; a_1[3-26], \;\; a_2[4-27]$
  - $b_0[1-24], \;\; b_1[10-1], \;\; b_2[12-3]$
  - $c_0[30-21], \;\; c_1[6-29], \;\; c_2[11-2]$
  - $e_0[27-18], \;\; e_1[20-11]$
- This is very much similar to the 12 slice solution
- In this case we get 34 dependencies
- The total number of possible solutions is equal to $2^{2 \cdot 41-34-7} = 2^{41}$

Final solutions

- Thus, using above method, we find the possible solutions for the remaining 6 slices and 2 slices

Final solutions

- Thus, using above method, we find the possible solutions for the remaining 6 slices and 2 slices
- For finding the solution for the last consicutive 8 slices, we merge possible solution of its consituent 6-slice and 2-slice

Final solutions

- Thus, using above method, we find the possible solutions for the remaining 6 slices and 2 slices
- For finding the solution for the last consicutive 8 slices, we merge possible solution of its consituent 6-slice and 2-slice
- Final solution space is obtained by merging the solution space of first 24 slices and the last 8 slices

Final solutions

- Thus, using above method, we find the possible solutions for the remaining 6 slices and 2 slices

- For finding the solution for the last consicutive 8 slices, we merge possible solution of its consituent 6-slice and 2-slice

- Final solution space is obtained by merging the solution space of first 24 slices and the last 8 slices

- Space complexity of the attack $= 2^{42}$

Final solutions

- Thus, using above method, we find the possible solutions for the remaining 6 slices and 2 slices
- For finding the solution for the last consicutive 8 slices, we merge possible solution of its consituent 6-slice and 2-slice
- Final solution space is obtained by merging the solution space of first 24 slices and the last 8 slices
- Space complexity of the attack $= 2^{42}$
- Time complexity of the attack $= 2^{44}$

Conclusion

- We have presented a preimage attack on the 2 rounds of round reduced $\text{KECCAK}[r := 800 - 384, c := 384]$.

Conclusion

- We have presented a preimage attack on the 2 rounds of round reduced KECCAK$[r := 800 - 384, c := 384]$.
- A practical attack with attack complexity of 2^{44}

Conclusion

- We have presented a preimage attack on the 2 rounds of round reduced $\textsc{Keccak}[r := 800 - 384, c := 384]$.
- A practical attack with attack complexity of 2^{44}
- Future work: Variant(s) of this attack for more rounds of $\textsc{Keccak}$.

Questions?

Thank You