Finding Collisions against 4-round SHA3-384 in Practical Time

Senyang Huang[1,2], Orna Agmon Ben-Yehuda[3], Orr Dunkelman[1], Alexander Maximov[4]

Dept. of Electrical and Information Technology, Lund University, Lund, Sweden

Dept. of Computer Science, University of Haifa, Haifa, Israel

CRI, University of Haifa, Haifa, Israel

Ericsson Research, Lund, Sweden

2023 . 3 . 21

Contents

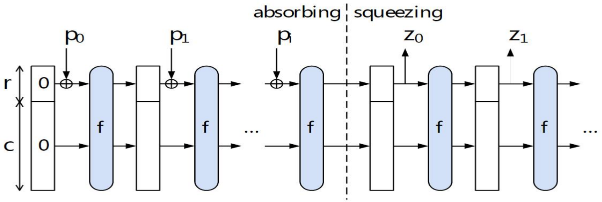
Collision Attack on Hash Functions

- Cryptographic hash functions are unkeyed primitives that accept an arbitrarily long input *message* and produce a fixed length output *hash value*, or *digest* for short.
- Hash functions are extremely useful in various cryptographic protocols authentication, password protection, commitment schemes, key exchange protocols, etc.
- One of the security requirements for a secure hash function H is that it should be computationally difficult to find a collision message pair $\{(x, y) | x \neq y, \text{s.t.} H(x) = H(y)\}$.

Keccak Sponge Function

- The Keccak sponge function family, designed by Bertoni, Daemen, Peeters, and Giles in 2007, was selected by the U.S. National Institute of Standards and Technology (NIST) in 2012 as the proposed SHA-3 cryptographic hash function.

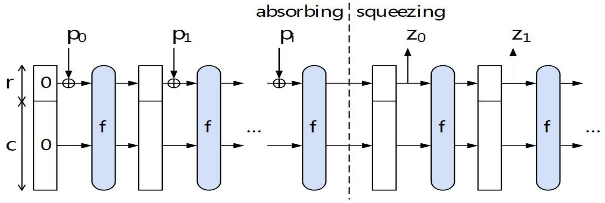
Keccak Sponge Function



Sponge Construction

- b-bit permutation f, f contains 24 rounds.
- Two parameters: bitrate r and capacity c, $b = r + c$. $b = 1600$ by default.

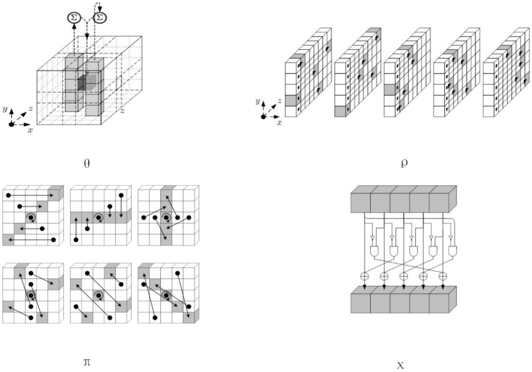
Keccak Sponge Function



Sponge Construction

- Four versions: Keccak-512, Keccak-384, Keccak-256, Keccak-224.
- SHA3-n is different from Keccak-n only in the padding rules.
- $n = c/2$.

Keccak Sponge function



The Round Function of Keccak

- Round function: $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$
- Linear layer: $L \triangleq \pi \circ \rho \circ \theta$
- χ is a nonlinear layer.

Proposition 1

(**CP-kernel Equation**) *For every i-th and j-th bits in the same column of the state A we have:*

$$A[i] \oplus A[j] = B[\sigma(i)] \oplus B[\sigma(j)],$$

where A and B are the input and output states of L, respectively, and $0 \leq i, j < 1600,\ i \neq j$. $\sigma = \pi \circ \rho$ is a combined permutation, which forms a mapping on integers $\{0, 1, \cdots, 1599\}$ such that $\sigma(i)$ is the new position of the i-th bit in the state after applying $\pi \circ \rho$.

Variant[r, c, d]	n_r	Complexity	Reference
Keccak-512	3	Practical	[DDS13]
Keccak-384	3	Practical	[DDS13]
Keccak-384	4	2^{147}	[DDS13]
SHA3-384	4	$2^{59.64}$	This work
Keccak-256	4	Practical	[DDS12][DDS14]
Keccak-256	5	2^{115}	[DDS13]
SHA3-256	5	Practical	[GLL+20]
Keccak-224	4	Practical	[DDS12][DDS14]
SHA3-224	5	Practical	[GLL+20]

Contents

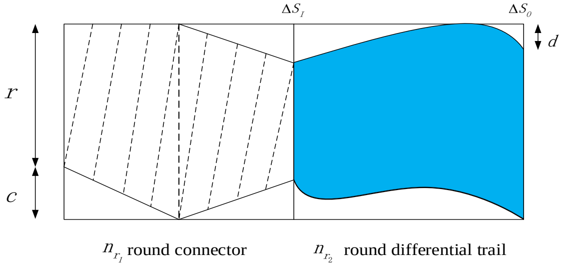
Framework of previous works

$(n_{r_1} + n_{r_2})$-round collision attacks:

- n_{r_1}-round connector: produce a brunch of message pairs (M_1, M_1'), s.t.

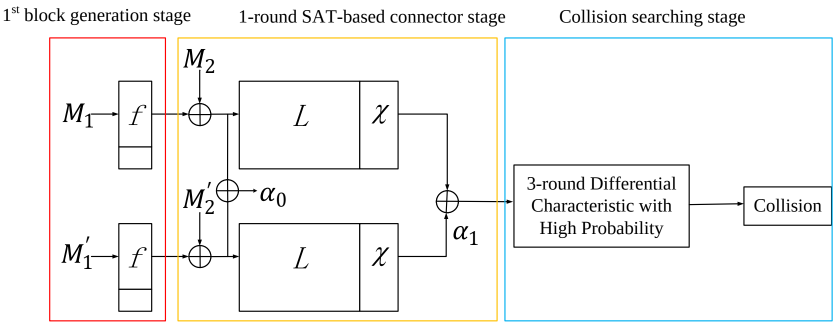
$$R^{n_{r_1}}(M_1||0^c) \oplus R^{n_{r_1}}(M_1'||0^c) = \Delta S_I$$

- Linearisation techniques: $n_{r_1} = 1$[DDS12] $\rightarrow n_{r_1} = 2$[QSLG17] $\rightarrow n_{r_1} = 3$ [SLG17]
- n_{r_2}-round high probability differential trail: $\Delta S_I \rightarrow \Delta S_O$, with first d bits of ΔS_O being zero.



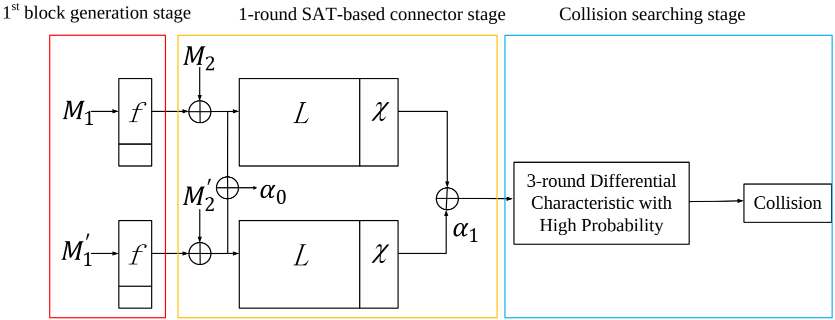
- The main drawback of previous lineariston techniques is that bit conditions are added in order to linearise the first rounds, thus consuming many degrees of **freedom**.
- As the input space of SHA-3-384 is too small for a sufficient level of degrees of **freedom**, extra bit conditions may cause contradictions making the linearisation technique infeasible.

Framework of Our Attack



- The first block is used as a pseudo random number generater.
- Our two-block collision attack can be extended to a multi-block attack, where the first few blocks can be chosen prefixes with meaningful information.

Framework of Our Attack



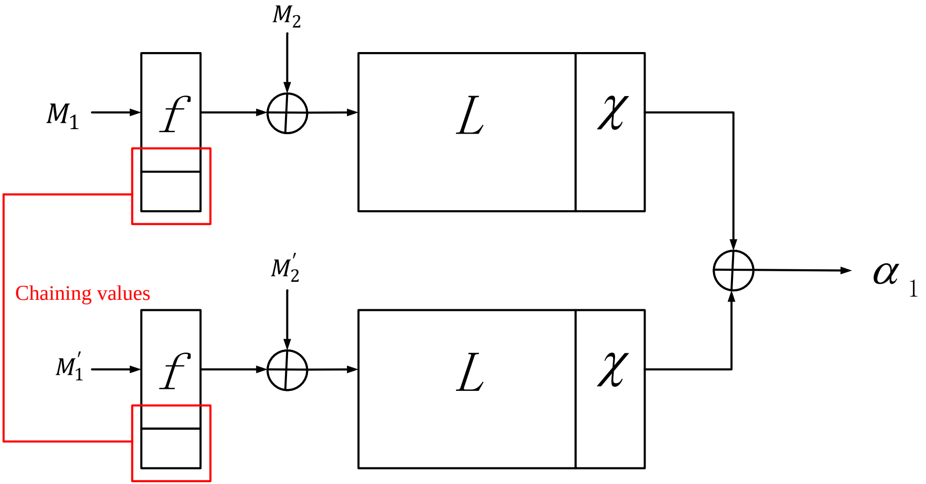
- We gain greater flexibility in choosing the differential characteristic as now we can "connect" to a wider range of input differences.
- Non-linear conditions which are useful in finding collisions (i.e., fixing intermediate bits to some values) are much easier to be satisfied using this sort of tools.

Contents

Differential Characteristic						Probability
$\alpha_1(\Delta S_I)$	7c0bc4f5b4398002 7c0bccf5b4398002 7c0bc4f5b4398000 7c0bc4f5bc398002 7c0bc4f1b4398002	2407de4bc9668001 240fde4bc9e68001 240fda4bc9e68001 240fde4fc9668001 240fde4bc9e68001	ac02095d32eb8000 ac02095d32ef8000 ac02095d32eb8000 ac02095d32eb8000 ac02095d3aeb8000	d402e98975068000 c40ae98975068000 c40ae9897d068000 c40ae98975068000 d40ae98975868000	3c05706a07f58000 3414706a05f58000 3c15706a25f58000 3c15706a05f48000 3c15706a05f58000	1
β_1	0000000000000000 0000000001008000 0010000001000000 0010000000008000 0000000000000000	0000000000000000 0000000000008010 0000000001000000 0000000000008000 0000000000000000	0000000000000000 0000000000000010 0010000000000000 0010000000000000 0000000000000010	0000000000000000 0000000000008010 0000000001000000 0000000000000000 0000000000000010	0000000000000000 0000000001000000 0010000000000000 0010000000008000 0000000000000000	2^{-26}
β_2	0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000	8000000000000000 0000000000000000 0000000000000000 0000000000000001 8000000000000000	0000000000000000 0000000080000000 0000000000000000 0000000000000000 0000000080000000	0000000000000000 0000000000000001 0000000000000001 0000000000000000 0000000000000000	0000000000000000 0000000000000000 0000000000000000 0000000000000001 0000000000000000	2^{-15}
β_3	0000000000000000 0000000000000000 0000000000000001 0000000000000000 0000000000000000	0000000000000000 0000000000000000 0000002000000000 0000000000000000 0080000000000000	0000000000000000 0000000000000000 0000000002000000 0000000000000400 0000000000000000	0000000000000000 0002000000000000 0000000000000000 0000000000000000 0000000000000000	0000000000000000 0000000010000000 0000000000000000 0000000000000000 0000000000000002	2^{-1}
$\alpha_4(\Delta S_O)$	0000000000000000 0000000000000000 0000000000000001 0000000000000000 000000000000000?	0000000000000000 0000000000000000 000000200000000? 0000000000000000 008000000000000?	0000000000000000 0000000000000000 000000?00200000? 0000000000000400 00?0000000000000	0000000000000000 0002000000000000 000000?000000000 0000000000000?00 00?0000000000000	0000000000000000 0000?00010000000 0000000000000000 0000000000000?00 0000000000000002	–

The 3-round differential characteristic in our attack adapts the second characteristic in [GLL+20, Table 9].

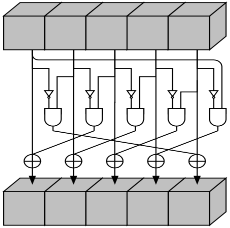
Requirements on the chaining values



Proposition 2

Suppose that the input difference is denoted as $(\delta_4, \delta_3, \delta_2, \delta_1, \delta_0)$.

1. *If the output difference of* χ *is* $(0, 0, 0, 0, 0)$*, the input difference is* $(0, 0, 0, 0, 0)$.
2. *If the output difference of* χ *is* $(0, 0, 0, 0, 1)$, $\delta_0 = 1$.
3. *If the output difference of* χ *is* $(0, 0, 0, 1, 1)$, $\delta_1 \oplus \delta_3 = 1$.

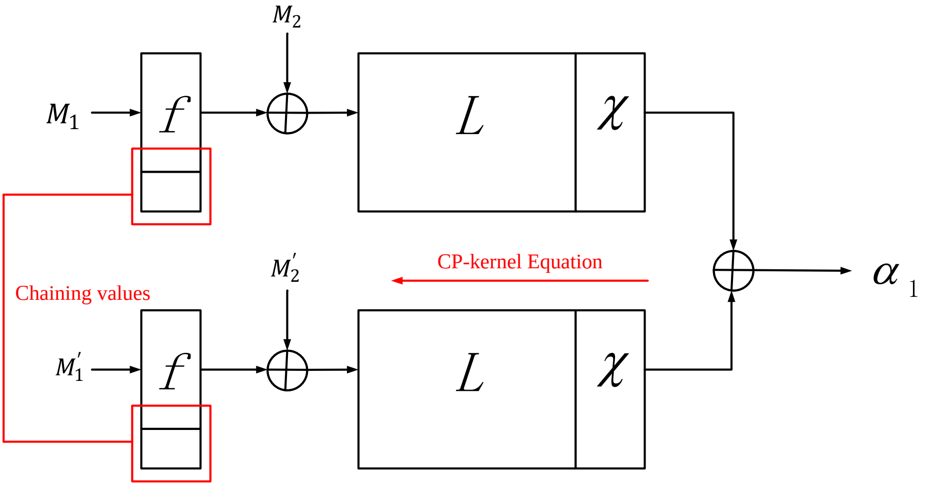


Requirements on the chaining values

Output Difference	Conditions	Output Difference	Conditions
0x1	$\delta_{in}[0] = 1$	0x3	$\delta_{in}[1] \oplus \delta_{in}[3] = 1$
0x2	$\delta_{in}[1] = 1$	0x6	$\delta_{in}[2] \oplus \delta_{in}[4] = 1$
0x4	$\delta_{in}[2] = 1$	0xc	$\delta_{in}[3] \oplus \delta_{in}[0] = 1$
0x8	$\delta_{in}[3] = 1$	0x18	$\delta_{in}[4] \oplus \delta_{in}[1] = 1$
0x10	$\delta_{in}[4] = 1$	0x11	$\delta_{in}[0] \oplus \delta_{in}[2] = 1$
0	$\delta_{in} = 0$		

Table: Summary of conditions for special output differences of χ.

Requirements on the chaining values



Requirements on the chaining values

$\alpha_0[939]+\alpha_0[1579]=0,\ \alpha_0[867]+\alpha_0[1187]=0,\ \alpha_0[868]+\alpha_0[1188]=0,$ $\alpha_0[881]+\alpha_0[1201]=0,\ \alpha_0[882]+\alpha_0[1202]=0,\ \alpha_0[883]+\alpha_0[1203]=0,$ $\alpha_0[884]+\alpha_0[1204]=0,\ \alpha_0[885]+\alpha_0[1205]=0,\ \alpha_0[886]+\alpha_0[1206]=0,$ $\alpha_0[887]+\alpha_0[1207]=0,\ \alpha_0[888]+\alpha_0[1208]=0,\ \alpha_0[889]+\alpha_0[1209]=0,$ $\alpha_0[999]+\alpha_0[1319]=0,\ \alpha_0[1000]+\alpha_0[1320]=0,\ \alpha_0[1001]+\alpha_0[1321]=0,$ $\alpha_0[1036]+\alpha_0[1356]=0,\ \alpha_0[1037]+\alpha_0[1357]=0,\ \alpha_0[1038]+\alpha_0[1358]=0,$ $\alpha_0[1039]+\alpha_0[1359]=0,\ \alpha_0[1040]+\alpha_0[1360]=0,\ \alpha_0[1088]+\alpha_0[1408]=0,$ $\alpha_0[1148]+\alpha_0[1468]=0,\ \alpha_0[1149]+\alpha_0[1469]=0,\ \alpha_0[1150]+\alpha_0[1470]=0,$ $\alpha_0[1151]+\alpha_0[1471]=0,\ \alpha_0[1216]+\alpha_0[1536]=0,\ \alpha_0[1217]+\alpha_0[1537]=0,$ $\alpha_0[1218]+\alpha_0[1538]=0,\ \alpha_0[1219]+\alpha_0[1539]=0,\ \alpha_0[1220]+\alpha_0[1540]=0,$ $\alpha_0[1277]+\alpha_0[1597]=0,\ \alpha_0[1278]+\alpha_0[1598]=0,\ \alpha_0[1279]+\alpha_0[1599]=0,$ $\alpha_0[938]+\alpha_0[1578]=0,\ \alpha_0[959]+\alpha_0[1279]=1,\ \alpha_0[998]+\alpha_0[1318]=1,$ $\alpha_0[1147]+\alpha_0[1467]=1,\ \alpha_0[836]+\alpha_0[1476]=1$
$\alpha_0[952]+\alpha_0[1592]+\alpha_0[1373]+\alpha_0[1053]=1$

Table: Conditions on chaining values

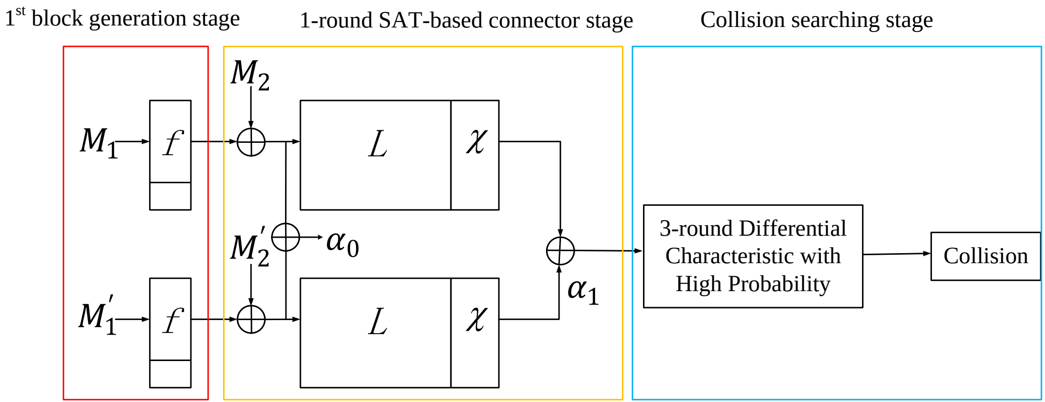
Algorithm Generating Prefix Pairs

Constant XOR Σ=0x7c00000000
$S_P = \emptyset$
Initialise an array Counter of length 2^{39} with zeros.
for each integer $i \in [0, 2^n)$ **do**
    Randomly pick a message M of 832 bits and compute the value string c.
    HashTable[c][Counter[c]]=M
    Increase Counter[c] by 1.
end for
for each integer $i \in [0, 2^n)$ **do**
    if $i < i \oplus \Sigma$ **then**
        for each integer $j \in [0, \text{Counter}[i])$ **do**
            for each integer $k \in [0, \text{Counter}[i \oplus \Sigma]]$ **do**
                $S_P = S_P \cup \{(\text{HashTable}[i][j], \text{HashTable}[i \oplus \Sigma][k])\}$
            end for
        end for
    end if
end for

Contents

Definition 3 (Connectivity Problem)

Given M_1 and M_1', find M_2 and M_2' s.t.
$R(f(M_1||0) \bigoplus (M_1||0)) \bigoplus R(f(M_1'||0) \bigoplus (M_2'||0)) = \alpha_1$.



- Solve the connectivity problems with a SAT solver directly? Time-consuming!

- ~~Solve the connectivity problems with a SAT solver directly.~~
- Filter the prefix pairs generated in the first stage $\rightarrow$ Deduce-and-sieve Algorithm

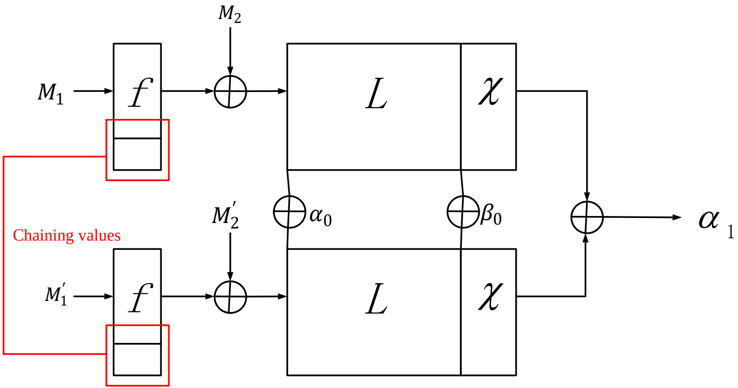
Two phases in deduce-and-sieve algorithm:

- Difference phase
- Value phase

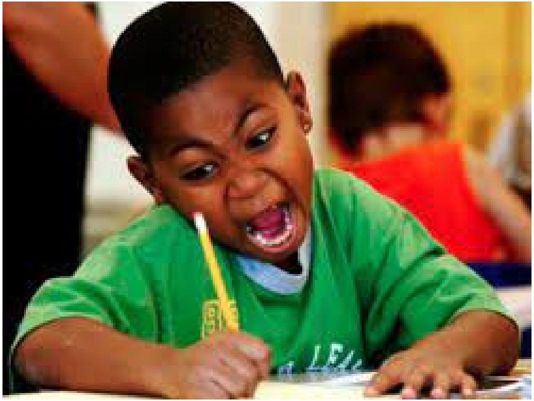
Difference Phase

Given a prefix pair (M_1, M_1') and α_1,

- the chaining values are known $\iff$ part of α_0 is known
- the conditions on β_0 should hold if the connectivity problem is solvable.



Derive New Bit Differences of α_0 and β_0



DEDUCE

- Derive from CP-kernel equations:

$$\alpha_0[i] \bigoplus \alpha_0[j] = \beta_0[\sigma(i)] \bigoplus \beta_0[\sigma(j)]$$

- Derive from bit relations:

$$\alpha_0[i] \oplus (\bigoplus_{k=0}^{4} \alpha_0[i_0 + 320 \cdot k]) \oplus (\bigoplus_{k=0}^{4} \alpha_0[j_0 + 320 \cdot k]) = \beta_0[\sigma(i)]$$

SIEVE

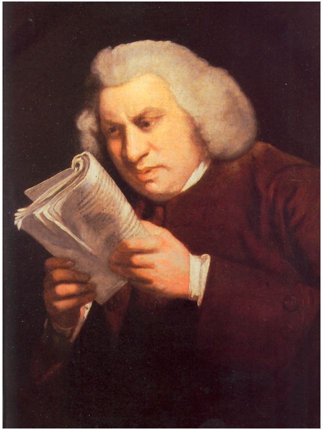


Figure: Truncated Difference Transition Table (TDTT)

Truncated Difference Transition Table (TDTT)

Definition 4

Given a truncated input difference Δ_{in}^{T} and an output difference Δ_{out}, the entry TDTT$(\Delta_{in}^{T}, \Delta_{out})$ of the S-box's TDTT is:

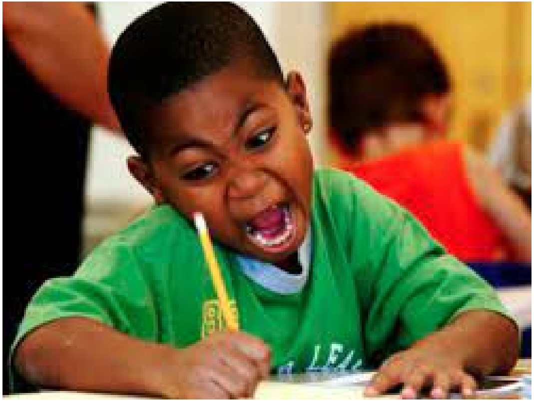
$$TDTT(\Delta_{in}^{T}, \Delta_{out}) = \begin{cases} null, & \text{if } \Delta_{in}^{T} \text{ does not deduce } \Delta_{out}, \text{ or } \Delta_{in}^{T} \text{ is irregular} \\ \Delta_{in}^{T'}, & \text{if more bits of the input difference can be derived} \\ \Delta_{in}^{T}, & \text{if no more bits can be derived} \end{cases}$$

where $\Delta_{in}^{T'}$ is the new truncated input difference, $\Delta_{in}^{T}, \Delta_{in}^{T'} \in \mathbb{F}_2^{2n}$ and $\Delta_{out} \in \mathbb{F}_2^{n}$.

E.g.

- Δ_{in}^{T} =???0?, $\Delta_{out} = 00011$
- The compatible input differences are 01001, 11001 and 11101.
- In this case, the truncated input difference should be Δ_{in}^{T} =?1?01

DEDUCE



Algorithm Discarding Prefix Pairs with TDTT

for each S-box **do**
    Deduce the output difference Δ_{out} from α_1.
    Deduce the truncated input difference Δ_{in}^T from β_0 and β_0^S.
    $T \leftarrow \text{TDTT}(\Delta_{in}^T, \Delta_{out})$
    if T=null **then**
        return 0.
    else if $T = \Delta_{in}^T$ **then**
        continue
    else if $T \neq \Delta_{in}^T$ **then**
        Find the indices of the five bits in the S-box as $i_0, i_1, \cdots, i_4$.
        for each integer $j \in [0, 5)$ **do**
            if the $(j+5)$th bit of Δ_{in} is 0 **and** $T_{j+5} = 1$ **then**
                Set $\beta_0[i_j] = T_j$, $\beta_0^S[i_j] = 1$
                Call CPkernel(α_0, β_0, α_0^S, β_0^S, $\phi_0(\sigma^{-1}(i_j))$)
            end if
        end for
    end if
end for

We can filter most of the prefix pairs applying the difference phase. But the filtering rate is not satisfying.

Two phases in deduce-and-sieve algorithm:

- Difference phase
- Value phase

Value Phase – Fixed Value Distribution Table (FVDT)

Definition 5

Given a truncated input difference Δ_{in}^{T} and an output difference Δ_{out}, the entry FVDT($\Delta_{in}^{T}, \Delta_{out}$) of the S-box's FVDT is:

$$FVDT(\Delta_{in}^{T}, \Delta_{out}) = \begin{cases} null, & \text{if } \Delta_{in}^{T} \text{ does not deduce } \Delta_{out}, \text{ or } \Delta_{in}^{T} \text{ is irregular.} \\ v, & \text{otherwise.} \end{cases}$$

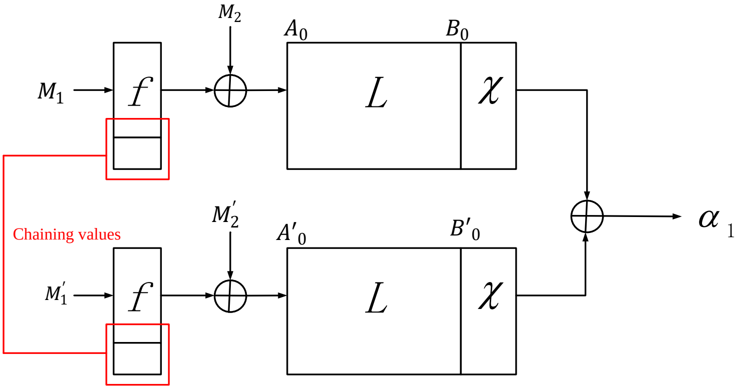
where Δ_{in}^{T}, $v \in \mathbb{F}_2^{2n}$, $\Delta_{out} \in \mathbb{F}_2^{n}$ and v is the fixed point with respect to Δ_{in}^{T} and Δ_{out}.

E.g.

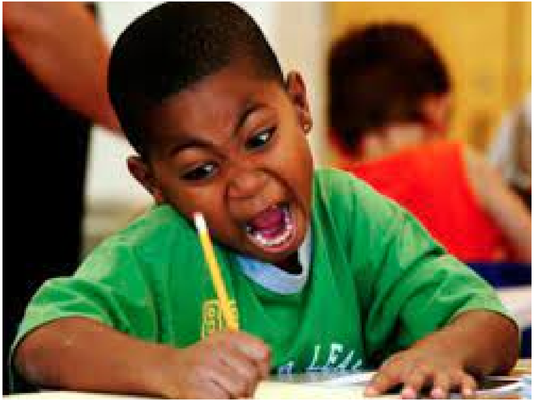
- Δ_{in}^{T} =??01?, $\Delta_{out} = 00001$
- The compatible differences are 01011 and 11011.
- The solution set is $S_T(??01?, 00001) = \{00000, 00011, 01000, 01011\} \cup \{00001, 11010\}$.

What do we have now?

- the chaining values (bit values of A_0 and A'_0) are known.
- bit values of B_0 and B'_0 are known from FVDT.

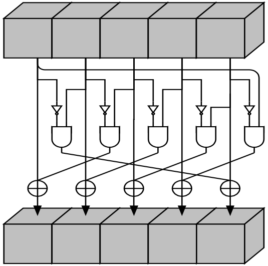


Derive New Bit Values of A, A', B and B'



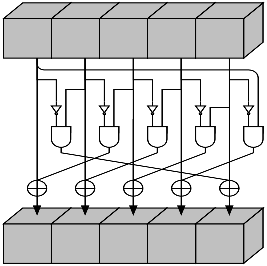
Value Phase

$x_0,\quad x_1,\quad x_2,\quad x_3,\quad x_4$



$y_0,\quad y_1,\quad y_2,\quad y_3,\quad y_4$

$x_0',\quad x_1',\quad x_2',\quad x_3',\quad x_4'$



$y_0',\quad y_1',\quad y_2',\quad y_3',\quad y_4'$

$$\left.\begin{aligned} y_0 &= x_0 \oplus (x_1 \oplus 1) \cdot x_2 \\ y_0' &= x_0' \oplus (x_1' \oplus 1) \cdot x_2' \end{aligned}\right\} \Longrightarrow \Delta y_0 = \Delta x_0 \oplus (x_1 \oplus 1) \cdot x_2 \oplus (x_1' \oplus 1) \cdot x_2'$$

Derive more input differences in some bit positions.

Algorithm Value Phase of the Deduce-and-sieve Algorithm

```
Call InitialVP(α_0, β_0, α_1, α_0^S, β_0^S, B, B', B_S, B'_S, FVDT)
for each integer i ∈ [0, 320) do
    Call CPkernel(A, B, A_S, B_S, i)
    Call CPKernel(A', B', A'_S, B'_S, i)
end for
a =Update(α_0, β_0, α_1, α_0^S, β_0^S, B, B', B_S, B'_S)
if a = 0 then
    return 0                    ▷ No new bit differences are deduced.
else
    return 1                    ▷ New bit differences are deduced.
end if
```

Algorithm Deduce-and-sieve Algorithm

DeriveSieve(M_1, $M_1^{'}$, TDTT, FVDT)
$(A, A', A_S, A'_S, B, B', B_S, B'_S, \alpha_0, \alpha_0^S, \beta_0, \beta_0^S)$=Initial($M_1$, $M_1^{'}$)
flag $= 1$
while *flag* **do**
    flag =DP(M_1, $M_1^{'}$, α_0, β_0, α_1, α_0^S, β_0^S, TDTT)
    if *flag* **then**
        flag =VP(α_0, β_0, α_1, α_0^S, β_0^S, A, A', A_S, A'_S, B, B', B_S, B'_S, FVDT)
        if *flag* $= 0$ **then**
            return 1 ▷Accept the prefix pair
        end if
    else
        return 0 ▷Discard the prefix pair
    end if
end while

Some of the generated prefix pairs have been filtered by applying the deduce-and-sieve algorithm. The connectivity problems of the remaining prefix pairs are determined by using a SAT-solver called CryptoMiniSAT.

Contents

- The method in the collision searching stage follows Guo et al.'s work [QSLG17, SLG17].
- All solutions for a corresponding connectivity problem form an affine subspace.
- Search the affine subspace exhaustively for the collision message pair.

Contents

1. 1st block generation stage: generate prefix message pairs
2. 1-round SAT-based connector stage
    - filter the prefix message pairs with deduce-and-sieve algorithm
    - solve the connectivity problems over the remaining prefix message pairs with a SAT solver
3. Collision searching stage: search for the collision suffix message pair

- The filtering rate of deduce-and-sieve algorithm is $2^{-19.42}$.
- The average running time of the deduce-and-sieve algorithm is 1.22×10^{-5}s for a prefix pair.
- The average running time of the SAT solver for every prefix pair is 0.31s
- Deduce-and-sieve algorithm outperforms the SAT solver by a factor of 2.54×10^{4} on this special type of SAT problems.

- We define a *semi-free n-bit internal collision attack* in which situation the adversary is assumed to have the capacity of modifying n-bit chaining values for each suffix message, where $n > 0$.
- From our experiments, there are 11.07 suffix seed pairs on average in $2^{41.3}$ prefix message pairs to construct semi-free 14-bit internal collision attacks.
- To build a real collision attack, we need to collect 2^{14} suffix seed pairs for the semi-free 14-bit internal collision attack.
- The time complexity of our collision attack is determined by the complexity of the second stage, which is $2^{59.64}$. The memory and data complexity are both $2^{45.92}$.

M_1	5732121a0fbfccdd a6b588d6643b6fce 2539995219a2ce0b	3df4817046b87bb1 2e17f6154a55be62 29efb889f172624b	d00adfa01cf61d66 7ed2eb58ca74dd3d 241d314913f32ec0	fbd8327932de6b42 45e995d069e01873	1e0cd531ed3dbbe1 8f1bfe1bcf516038
M_2	73d2c43d15d68ac7 cf50106808412695 911d77c7f077b8f	fa5d040dff851751 4551bf03cb0bbf25 d24e61e7e9bad037	fdf1c8f504ddc895 f4544f840a2f65a7 f0ee7da479ccdb0d	a112154efd855b32 bcce3ec44e560b73	e5b66a03d74127aa e652b76f1af97123
M_1'	5b3f3de5af8b3513 b0837ea6d3a8333a 91218525188f2fc1	d8943ff358e8dd8a eaa1ca4dff69a1cc 8170fc1f64fbf10d	41335bb30c11643c 969790479bd934d2 8d424172e8264f5c	9e205a1a7a501109 9a55270d03777022	80d3cbaa427aa316 c51cfcceb2e668bb
M_2'	a0afd65757f0e1dd dc42016f089ee317 3a67013dd90c8c1a	6be5f0a54d323649 2de8a8c03a5b75eb 243c77f1f9dec1dd	6cc4a8dcebd91fa9 9c6515d09e202385 34cd394488378778	102d4731eb8f9549 7baa86549b09ca54	5f5b8d0749cafeb 9eb057116c73aaca
H	ed3e58fde7229fec 4228cee97acc3204	bc8fc643fc5d7fa3	6d6751e1f3dceaab	5d5192031990a2ef	6f7ab88b4137642c

Table: Semi-free 4-bit Internal Collision Messages and Hash Value

Thank you for your attention!

Itai Dinur, Orr Dunkelman, and Adi Shamir.
New attacks on keccak-224 and keccak-256.
In *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, pages 442–461, 2012.

Itai Dinur, Orr Dunkelman, and Adi Shamir.
Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials.
In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 219–240, 2013.

Itai Dinur, Orr Dunkelman, and Adi Shamir.
Improved practical attacks on round-reduced keccak.
J. Cryptology, 27(2):183–209, 2014.

Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song.
Practical collision attacks against round-reduced SHA-3.
J. Cryptology, 33(1):228–270, 2020.

Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo.
New collision attacks on round-reduced keccak.
In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 216–243, 2017.

Ling Song, Guohong Liao, and Jian Guo.
Non-full sbox linearization: Applications to collision attacks on round-reduced keccak.
In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 428–451, 2017.