Term paper

on

ACE

Lightweight Cryptography

Aikata
11640090

May 1, 2020

Abstract. ACE is a cipher submitted to NIST's Lightweight Cryptography Competition. It is degined by Mark Aagaard, Riham AlTawy, Guang Gong, Kalikinkar Mandal, and Raghvendra Rohit, a team from University of Waterloo. With it's strong security and lightweightedness ACE successfully made it to ROUND- 2 of the competition. This report analyzes ACE on various existing attacks and provides some new insights on the security of the cipher. It reports weakness in the implementation of the cipher provided by the designers and new attacks based on it's permutation. The analysis is both software and harware based. ACE permutation is based on Simeck. Simeck is a family of three lightweight block ciphers designed by Yang et al., following the framework used by Beaulieu et al. from the United States National Security Agency to design SIMON and SPECK. We analyze a lot of attacks done on Simeck and also present some new possible attacks.

Keywords: ACE, SIMECK , NIST , Lightweight Cryptography, Cryptanalysis

Contents

1 Introduction

The IoT devices such as RFID, smart cards, sensors nodes play an important role in the network. These devices have limited memory and are battery operated devices. The standard cryptography algorithms (such as AES) provide good security but their performance is not acceptable on these devices because of large memory requirements to store s-boxes, large block and key sizes. To cater to such issues, NIST recommended preferring lightweight algorithms which provide same level of security and their performance is also acceptable on these devices. Following this in 2013, NIST started the Lightweight Cryptography (LWC) project, with the end goal of creating a portfolio of lightweight algorithms for authenticated encryption with associated data (AEAD), and optionally hashing, in constrained environments.

`ACE` is a candidate in NIST's lWC competition. `ACE`, often known as one of the strongest cards in a deck of cards, is an 320-bit lightweight permutation. It is designed to achieve a balance between hardware cost and software efficiency for both Authenticated Encryption with Associated Data (henceforth "AEAD") and hashing functionalities, while providing sufficient security margins. To accomplish these goals, `ACE` components and its mode of operation are adopted from well known and analyzed cryptographic primitives. The designers of `ACE` have not only submitted the designs but also analyzed the security of the cipher on varous software and hardware fronts. They analyzed how the cipher will be secure against linear cryptanalysis, differential cryptanalysis and othrs. They also ensured that along with security the cipher is also efficient, for this hardware implementation results are an important criteria for assessment and comparison. In the first round of the LWC evaluation, more than half of the candidates reported hardware implementation results or their estimates, ranging from complete implementation and analysis to partial implementation results and theoretical estimates.

The analysis of the permuattion is done along with my colleague **PINTU KUMAR**. We performed a security analysis of `ACE` and it's permutation cryptoprimitive `Simeck` by analyzing the confusion and diffusion of the cipher and using Linear, Integral, Differential, Impossible Differential, Related key cryptanalysis techniques to uncover a relationship of key bits.

1.1 My contributions in line with the running assigment

1. Study the feasibility of impossible differential cryptanalysis and Zero-Sum property in the NIST LWC Cipher that you have been allocated.
   • Done in Section 6.5 and Section 6.1 Algebraic Degree of `Simeck`32 respectively. We also found a new improved impossible differential trail.

2. You need to see how MILP based automated cryptanalysis as shown in class applies to your cipher. (In case the authors have already provided some study, you are required to reproduce the same).
   • Done in Section 6.6 MILP for `Simeck`.

3. Compute the 1-1 DDT and 1-1 LAT of the the Sbox of your cipher. Now enlist the Good/Bad Inputs and Good/Bad Outputs.
   • Done in Section 4. Analysis of TWINE Sbox(Assigned Sbox).

4. Draw a nice diagram to summarize the design of your cipher both at the top-level (including the mode of operation) and also the internal transformation. At least two figures are expected. You are encouraged to improvise over figures given in the specifications (in case it has been provided by the designers) based on your study.
   • Done in Section 2. Design of `ACE`, all the diagrams are reproduced in as modified way as possible. The complete cipher design is also re-explaind in a simpler way.

5. BCT and BDT for our cipher
   • Done in Section 4. Analysis of TWINE Sbox(Assigned Sbox), and Appendix

6. Performance metrices
   • Done in Section 6.8 Performance Metrices of ACE-STEP.

7. Allotted Research Paper
   • Done in Section 6.5.2. and Section 6.6. The implementations for the Impossible Differential property as well as the MILP property stated in the paper are done.

1.2 My contributions apart from running assigment

- Section 5. Analysis of ACE implementation, gives our contribution towards finding some weak points in the official implementation of the cipher and weakness of the cipher design for weak key and nonce classes.

- In section 6.1 Algebraic Degree of Simeck32, we found that some of the degrees they have stated for some rounds are not really correct and also gave a possible degree for those rounds.

- In Sections 6.2, 6.3 and 6.4, we studied various attacks done on Simeck, reproduced them through implementations to cover the work that has already been done.

- In Section 6.5.1 Impossible Differential Attack on Simeck in Single Key setting we present a new 13 round trail and the implementation for the property.

- In Section 6.5.2 Impossible Differential Attack on Simeck in Related Key setting and Section 6.6 MILP of Simeck we have done the implementation of the Impossible Differential property as well as of the MILP property which is used to verify the impossible trails.

- In Section 6.7 Differential Fault Attack (DFA) on Simeck we studied the only paper that exists on DFA on Simeck.

1.3 Outline

In Section 2 the Design of the cipher is described and in Section 3 we give a brief security anaysis done by the designers. In next section i.e. Section 4 we analyze TWINE sbox as part of the running assigment. In Section 5 we present the weakness in the implementation of ACE and in Section 6 we analyze the permutation of ACE that is based on Simeck With Section 7 we conclude the report and also give the future scope of the work.

2 Design of ACE

ACE is an all in one primitive that utilizes a generalized version of sLiSCP-light with state size 320-bit and a different linear layer to offer both hashing and authenticated encryption functionalities in a single hardware circuit. The AEAD algorithm (ACE-AE-k) and the HASH algorithm (ACE-H-h) are parameterized by the size k of the secret key and the length of the message digest h in bits, respectively. Both the algorithms process the same amount of data per permutation call (i.e, rate r is same) and hence r value is ignored in the individual parameters' description. The Table 1 gives the parametric description of ACE .

Table 1: Parameter set for ACE-AE-128 and ACE-H-256

Functionality	Algorithm	rate	$\|Key\|$	$\|Nonce\|$	$\|Tag\|$	$\|Hash\|$	$\|IV\|$
AEAD	ACE-AE-128	64	128	128	128	-	-
Hash	ACE-H-256	64	-	-	-	256	24

2.1 ACE AEAD algorithm

The AEAD algorithm ACE-AE-k is a combination of two algorithms, an authenticated encryption algorithm ACE-E and the verified decryption algorithm ACE-D. An authenticated encryption algorithm ACE-E takes as input a secret key K of length k bits, a public message number N (nonce) of size n bits, a block header AD (associated data) and a message M . The output of ACE-E is an authenticated ciphertext C of same length as M , and an authentication tag T of size t bits.

The decryption and verification algorithm takes as input the secret key K, nonce N , associated data AD, ciphertext C and tag T , and outputs the plaintext M of same length as C only if the verification of tag is correct, and $\perp$ if the tag verification fails.

2.2 ACE Hash algorithm

The hash algorithm takes as input a message M , and the standard initialization vector IV of length iv bits, and returns a fixed size output H, called hash or message digest. Note that IV and N refer to two different things. IV is for a HASH function and is fixed, while N is for an AEAD algorithm and never repeated for a fixed key.

2.3 The ACE permutation

ACE is an iterative permutation that takes a 320-bit state as an input and outputs an 320-bit state after iterating the step function ACE-step for s = 16 times Figure 1 . The nonlinear operation SB-64 (Figure 2) is applied on even indexed words (i.e., A, C and E, see Figure 1) and hence the permutation name.

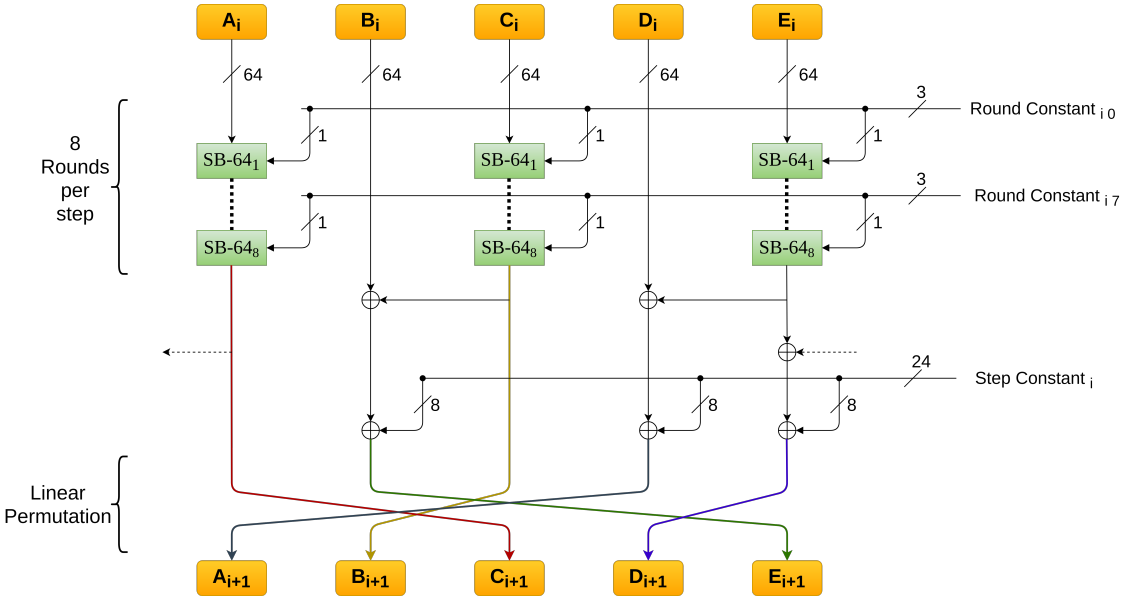


Figure 1: ACE-step

Here, Round Constant$_{ij} = (p_{ij}, q_{ij}, r_{ij})$ where $p_{ij}, q_{ij}, r_{ij} \in \{0, 1\}$ and $0 \leq j \leq 2$. A `Simeck` box is a permutation of a 64-bit input, constructed by iterating the `Simeck`-64 block cipher for 8 rounds (`SB-64`$_1$ to `SB-64`$_8$) with round constant addition $\alpha_{ij} = 1^{31}||p_{ij}$, $\beta_{ij} = 1^{31}||q_{ij}$ and $\gamma_{ij} = 1^{31}||r_{ij}$ in place of key addition, for words **A, C** and **E** respectively. Here `i` is the number of permutation round and `j` is the number of `Simeck` box round within every permutation round.

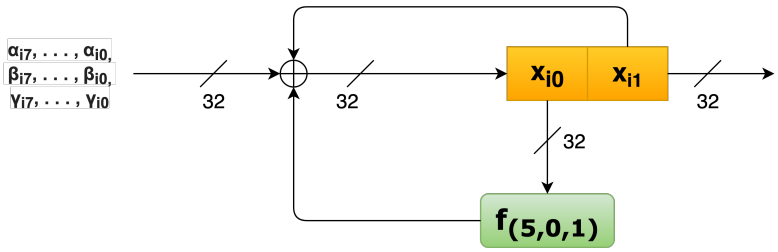


Figure 2: `Simeck` Box (`SB-64`$_i$)

The round function of `Simeck` Box - f is defined as: $f_{(5,0,1)}(x) = (L^5(x) \bigodot x) \bigoplus L^1(x)$. A round constant for ith permutation round is given as a tuple (p_i, q_i, r_i), but for every jth round within `Simeck` Box (`SB-64`) (Figure 2) we need jth bit of these constants, so we take (p_{ij}, q_{ij}, r_{ij}). There are 16 ACE-steps within every permutation (`ACE`). ACE employs a 7-bit LFSR shown in Figure 3 to generate round and step constants. List of round and step constants is given in Table 2.

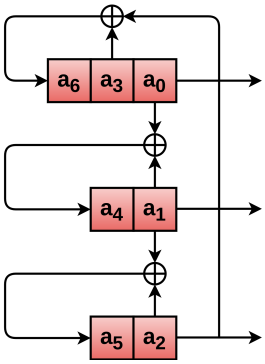


Figure 3: LFSR for generating ACE constants

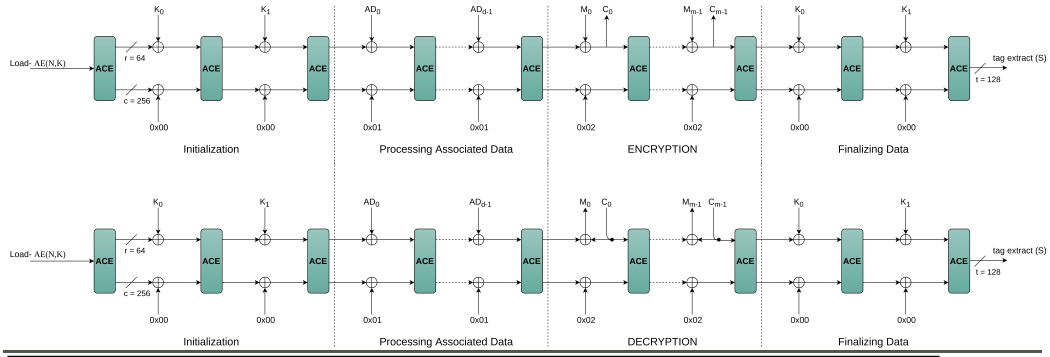
Table 2: Round and step constants of `ACE`

Step i	Round constants (rc_0^i, rc_1^i, rc_2^i)
0-3	(07, 53, 43), (0a, 5d, e4), (9b, 49, 5e), (e0, 7f, cc)
4-7	(d1, be, 32), (1a, 1d, 4e), (22, 28, 75), (f7, 6c, 25)
8 - 11	(62, 82, fd), (96, 47, f9), (71, 6b, 76), (aa, 88, a0)
12 - 15	(2b, dc, b0), (e9, 8b, 09), (cf, 59, 1e), (b7, c6, ad)
Step i	Step constants (sc_0^i, sc_1^i, sc_2^i)
0-3	(50, 28, 14), (5c, ae, 57), (91, 48, 24), (8d, c6, 63)
4-7	(53, a9, 54), (60, 30, 18), (68, 34, 9a), (e1, 70, 38)
8 - 11	(f6, 7b, bd), (9d, ce, 67), (40, 20, 10), (4f, 27, 13)
12 - 15	(be, 5f, 2f), (5b, ad, d6), (e9, 74, ba), (7f, 3f, 1f)

2.4 AEAD Algorithm: `ACE-AE-128`

The encryption `ACE-E` and decryption `ACE-D` processes of `ACE-AE-128` are shown in Figure 4. The dsigners assume a nonce-respecting adversary, i.e, for a fixed K, the nonce N is never repeated. Then considering a data limit of 2 d , k-bit security is achieved if c $\geq$ k + d + 1 and d $\ll$ c/2 . The parameter set of `ACE-AE-128`(see Table 1) with actual effective capacity 254 (2 bits are lost for domain separation) satisfies this condition, and hence `ACE-AE-128` provides 128-bit security for confidentiality, integrity and authenticity.

Note that we could use r = 192, d = 64 and obtain the same level of security . However, this would require an additional 128 XORs and cannot meet our objective to achieve both AEAD and Hash functionalities using the same hardware circuit.



Algorithm 2 ACE-$\mathcal{AE}$-128 algorithm

Authenticated encryption ACE-$\mathcal{E}(K, N, AD, M)$:
  $S \leftarrow$ Initialization(N, K)
  if $|AD| \neq 0$ **then:**
    $S \leftarrow$ Processing-Associated-Data(S, AD)
  $(S, C) \leftarrow$ Encyption(S, M)
  $T \leftarrow$ Finalization(S, K)
  return (C, T)

Initialization(N, K):
  $S \leftarrow$ load-$\mathcal{AE}(N, K)$
  $S \leftarrow$ ACE(S)
  for $i = 0$ to 1 **do:**
    $S \leftarrow (S_r \oplus K_i, S_c)$
    $S \leftarrow$ ACE(S)
  return S

Processing-Associated-Data(S, AD):
  $(AD_0||\cdots||AD_{\ell_{AD}-1}) \leftarrow$ pad$_r(AD)$
  for $i = 0$ to $\ell_{AD} - 1$ **do:**
    $S \leftarrow (S_r \oplus AD_i, S_c \oplus 0^{c-2}||01)$
    $S \leftarrow$ ACE(S)
  return S

Encryption(S, M):
  $(M_0||\cdots||M_{\ell_M-1}) \leftarrow$ pad$_r(M)$
  for $i = 0$ to $\ell_M - 1$ **do:**
    $C_i \leftarrow M_i \oplus S_r$
    $S \leftarrow (C_i, S_c \oplus 0^{c-2}||10)$
    $S \leftarrow$ ACE(S)
  $C_{\ell_M-1} \leftarrow$ trunc-msb$(C_{\ell_M-1}, |M| \bmod r)$
  $C \leftarrow (C_0, C_1, \ldots, C_{\ell_M-1})$
  return (S, C)

pad$_r(X)$:
  $X \leftarrow X||10^{r-1-(|X| \bmod r)}$
  return X

trunc-lsb(X, n):
  return $(x_{r-n}, x_{r-n+1}, \ldots, x_{r-1})$

Verified decryption ACE-$\mathcal{D}(K, N, AD, C, T)$:
  $S \leftarrow$ Initialization(N, K)
  if $|AD| \neq 0$ **then:**
    $S \leftarrow$ Processing-Associated-Data(S, AD)
  $(S, M) \leftarrow$ Decyption(S, C)
  $T' \leftarrow$ Finalization(S, K)
  if $T' \neq T$ **then:**
    return $\perp$
  else:
    return M

Decryption(S, C):
  $(C_0||\cdots||C_{\ell_C-1}) \leftarrow$ pad$_r(C)$
  for $i = 0$ to $\ell_C - 2$ **do:**
    $M_i \leftarrow C_i \oplus S_r$
    $S \leftarrow (C_i, S_c \oplus 0^{c-2}||10)$
    $S \leftarrow$ ACE(S)
  $M_{\ell_C-1} \leftarrow S_r \oplus C_{\ell_C-1}$
  $C_{\ell_C-1} \leftarrow$ trunc-msb$(C_{\ell_C-1}, |C| \bmod r)||$trunc-lsb$(M_{\ell_C-1}, r-|C| \bmod r))$
  $M_{\ell_C-1} \leftarrow$ trunc-msb$(M_{\ell_C-1}, |C| \bmod r)$
  $M \leftarrow (M_0, M_1, \ldots, M_{\ell_C-1})$
  $S \leftarrow$ ACE$(C_{\ell_C-1}, S_c \oplus 0^{c-2}||10)$
  return (S, M)

Finalization(S, K):
  for $i = 0$ to 1 **do:**
    $S \leftarrow$ ACE$(S_r \oplus K_i, S_c)$
  $T \leftarrow$ tagextract(S)
  return T

trunc-msb(X, n):
  if $n = 0$ **then:**
    return ϕ
  else:
    return $(x_0, x_1, \ldots, x_{n-1})$

Figure 4: Authenticated encryption and Verified decryption

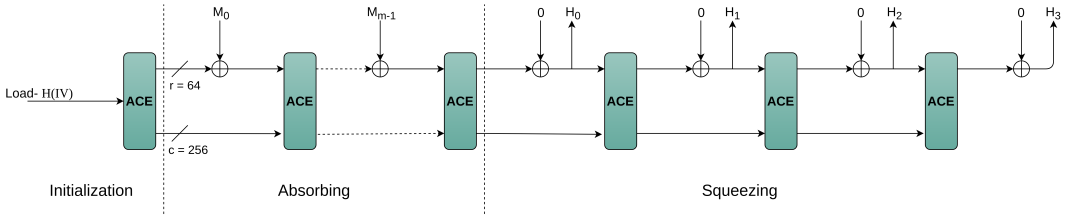
2.5 Hash Algorithm: `ACE-H-128`

The hash algorithm `ACE-H-128` takes as input a message M , and the standard initialization vector IV of length 24 bits, and then returns a 256-bit message digest H. The depiction of the ACE-H-256 is shown in Figure 5.

For a sponge based hash with b = r + c and h-bit message digest, the generic security bounds are given by:

• Collision:min(2 h/2 , 2 c/2)

• Preimage: min(2 min(h,b) , max(2 min(h,b)-r , 2 c/2))
• Second-preimage: min(2 h , 2 c/2)
Accordingly, ACE-H-256 provides 128, 192 and 128-bit securities for collision, preimage and second preimage, respectively.



Algorithm 3 ACE-$\mathcal{H}$-256 algorithm

ACE-$\mathcal{H}$-256(M, IV):
  $S \leftarrow$ Initialization(IV)
  $S \leftarrow$ Absorbing(S, M)
  $H \leftarrow$ Squeezing(S)
  return H

Initialization(IV):
  $S \leftarrow$ load-$\mathcal{H}(IV)$
  $S \leftarrow$ ACE(S)
  return S

pad$_r(M)$:
  $M \leftarrow M||10^{r-1-(|M| \bmod r)}$
  return M

Absorbing(S,M):
  $(M_0||\cdots||M_{\ell_M-1}) \leftarrow$ pad$_r(M)$
  for $i = 0$ to $\ell_M - 1$ **do:**
    $S \leftarrow$ ACE$(S_r \oplus M_i, S_c)$
  return S

Squeezing(S):
  for $i = 0$ to 2 **do:**
    $H_i \leftarrow S_r$
    $S \leftarrow$ ACE(S)
  $H_3 \leftarrow S_r$
  return $H_0||H_1||H_2||H_3$

Figure 5: Hash algorithm

2.6 State Initialization

In this section we describe how the state is initialized for both `ACE-AE-128` and `ACE-H-128`.

1. Nonce = $N_0||N_1$ (128 bits) and Key = $K_0||K_1$ (128 bits). Nonce is never repeated for a fixed key.

2.
$$\begin{array}{llllllllll}
\text{A}[7], & \text{A}[6], & \ldots, & \text{A}[0] & \leftarrow & K_0[7], & K_0[6], & \ldots, & K_0[0] & (64 \text{ bits}) \\
\text{C}[7], & \text{C}[6], & \ldots, & \text{C}[0] & \leftarrow & K_1[7], & K_1[6], & \ldots, & K_1[0] & (64 \text{ bits}) \\
\text{B}[7], & \text{B}[6], & \ldots, & \text{B}[0] & \leftarrow & N_0[7], & N_0[6], & \ldots, & N_0[0] & (64 \text{ bits}) \\
\text{E}[7], & \text{E}[6], & \ldots, & \text{E}[0] & \leftarrow & N_1[7], & N_1[6], & \ldots, & N_1[0] & (64 \text{ bits}) \\
\text{D}[7], & \text{D}[6], & \ldots, & \text{D}[0] & \leftarrow & 0x00, & 0x00, & \ldots, & 0x00 & (64 \text{ bits})
\end{array}$$

3. The rate part of the State is initialized with $S_r \leftarrow A[7], A[6], A[5], A[4], C[7], C[6], C[5], C[4]$, thus 64 bits. The remaining bytes form the capacity part of state, S_c.

4. Tag = $T_0||T_1$ (128 bits)
$T_0[7], T_0[6], \ldots, T_0[0] \longleftarrow A[7], A[6], \ldots, A[0]$
$T_1[7], T_1[6], \ldots, T_1[0] \longleftarrow C[7], C[6], \ldots, C[0]$

5. IV = $h/2||r||r'$ (24 bits) and is fixed. Eight bits are used to encode each of the used h/2, r and r' sizes and loaded in word B as follows.
   $B[7] \leftarrow 0x80$
   $B[6] \leftarrow 0x40$
   $B[5] \leftarrow 0x40$
   The remaining bytes are set to 0x00.

2.7 Padding

The padding rule (10*), denoting a single 1 followed by the required number of 0's, is applied to the message M, so that its length after padding is a multiple of r. The resulting padded message is divided into m r-bit blocks $M_0 \dots M_{m-1}$. A similar procedure is carried out on the associated data AD which results in d r-bit blocks $AD_0 \dots AD_{d-1}$. In the case where no associated data is present, no processing is necessary.

In case of AD or M whose length is a multiple of r, an additional r-bit padded block is appended to AD or M to distinguish between the processing of partial and complete blocks.

3 Designer's Security Analysis and Claims

Meet/miss-in-the-middle distinguishers cannot cover more than ten steps, because ten steps guarantees full bit diffusion in both forward and backward directions od ACE. Implying that there are no distinguishers for 16 steps of ACE.

After modelling ACE using Mixed Integer Linear Programming (MILP) the expected bounds for the minimum number of active `Simeck` boxes for all linear layers is given below in Table 3.

Table 3: Minimum number of active `Simeck` boxes for s-step ACE

step s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#Active SB	0	1	2	3	5	7	8	9	11	12	13	15	16	17	19	21

If the algebraic degree equals the number of active bits then the bit is unknown (i.e., XOR sum of the component function is unpredictable), otherwise, it is balanced in which case the XOR sum is always zero. Table 4 shows Integral distinguishers of ACE.

Table 4: Integral distinguishers of ACE

Steps s	Input division property	Balanced bits
8	$1^{32}\|\|0\|\|1^{287}$	64-127, 256-319
1	$1^{96}\|\|0\|\|1^{223}$	None
1	$1^{160}\|\|0\|\|1^{159}$	None
1	$1^{224}\|\|0\|\|1^{95}$	64-127, 256-319
1	$1^{288}\|\|0\|\|1^{31}$	None

To avoid attacks such as rotational distinguishers, slide distinguishers and invariant subspace attack which exploit the symmetric properties of a round function, round constants and step constants are added after every `Simeck` box Round and after every ACE-step.

They assume a nonce-respecting adversary, i.e, for a fixed K, the nonce N is never repeated.

4 Analysis of TWINE Sbox(Assigned Sbox)

TWINE s-box was assigned to me to study BOGI permutation etc. Table 5 gives the TWINE sbox. The next sections show the BOGI pemutation and BCT, and BDT computation for this sbox.

Table 5: TWINE Sbox

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
SB(x)	C	0	F	A	2	B	9	5	8	3	D	7	1	E	6	4

4.1 BOGI permutation

Authors of cipher GIFT [1] brought BOGI permutation into picture. They used BN2 sbox and BOGI permutations to provide linear and differential bounds as good as that of a BN3 sbox. BOGI stands for Bad Input to Good Input.The authors ensured that BOGI permutations exists for GIFT. If a single active bit transition occurs, the input and output active bit must be in BI and BO otherwise in GI and GO respectively. To analyze this 1-1 DDT and 1-1 LAT is required. The 1 - 1 DDT and 1 - 1 LAT of TWINE Sbox is given in Table 6.Using this table we found that this Sbox is not suitable for BOGI permutation mentioned in [1]. The code for this is attatched in *'implementations'* folder as (TWINE DDT, LAT, BCT and BDT.ipynb) in block 2 and 3 and also in Appendix 9.

Table 6: 1 - 1 LAT and 1 - 1 DDT

1 - 1 LAT	1000	0100	0010	0001	**1 - 1 DDT**	1000	0100	0010	0001
1000	-2	2	0	0	1000	0	2	2	2
0100	-2	0	0	0	0100	0	0	0	0
0010	0	4	0	2	0010	0	2	0	0
0001	-2	0	2	0	0001	0	0	2	0

GI = 0100
GO = 1000
BI = 1000, 0010, 0001
BO = 0100, 0010, 0001

Since, $|GO| + |GI| = 2 < 4$, BOGI permutation not suitable

4.2 BCT and BDT

BCT stands for Boomerang Connectivity Table and BDT stands for Boomerang Difference Table. Boomerang connectivity matrix of an invertible $m \times m$ S-Box S is an $2^m \times 2^m$ matrix with entry at row $\alpha \in F_m^2$ and column $\beta \in F_m^2$ equal to:

$$\#|\{x \in F_m^2 | S^{-1}(S(x) \oplus \beta) \oplus S^{-1}(S(x \oplus \alpha) \oplus \beta) = \alpha\}|$$

Boomerang Difference matrix of an invertible $m \times m$ S-Box S is an $2^m \times 2^m \times 2^m$ matrix with entry at row $\alpha \in F_m^2$ and column $\gamma \in F_m^2$ and $\beta \in F_m^2$ equal to :

$$\#|\{x \in F_m^2 | S^{-1}(S(x) \oplus \beta) \oplus S^{-1}(S(x \oplus \alpha) \oplus \beta) = \alpha \; and \; S(x) \oplus S(x \oplus \alpha) = \gamma\}|$$

BCT given Table 7 and BDT given in Table 22,23,24,25,26 and 8. In this ection only BDT for $(15, \gamma, \beta)$ where $\gamma \in F_4^2$ and $\beta \in F_4^2$ (Table 8) is shown. Rest of the BDTs are in Appendix. If we take the first columns of all these BDT Tables we'll get the DDT of Twine. The code for this is attatched in $'implementations'$ folder as (TWINE DDT, LAT, BCT and BDT.ipynb) in block 4 and 5 and also in Appendix 9. We also verified that BDT$(\alpha, \gamma, 0)$ is the same as it's DDT (also shown in block 5).

Table 7: BCT of twine

α, β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	0	2	0	0	6	0	0	0	6	2	2	4	0	0	2
2	16	0	0	2	6	2	0	2	0	0	4	2	0	0	6	0
3	16	0	2	0	0	2	2	6	6	0	0	0	0	0	2	4
4	16	0	0	2	0	0	6	0	0	2	0	4	0	6	2	2
5	16	6	4	6	0	0	2	2	0	2	2	0	0	0	0	0
6	16	2	0	0	0	4	0	2	0	6	0	0	6	2	2	0
7	16	0	0	2	2	6	2	0	2	4	0	0	6	0	0	0
8	16	6	6	4	2	2	0	0	0	0	0	0	0	2	0	2
9	16	0	0	2	0	0	0	6	4	0	2	0	2	2	0	6
10	16	2	0	0	2	0	0	4	6	2	0	2	0	0	0	6
11	16	0	2	0	2	0	6	2	0	0	0	6	2	4	0	0
12	16	0	2	0	6	0	0	0	2	2	6	0	0	2	4	0
13	16	4	6	6	0	0	0	0	2	0	0	2	2	0	2	0
14	16	2	0	0	4	0	2	0	0	0	6	0	2	0	6	2
15	16	2	0	0	0	2	4	0	2	0	2	6	0	6	0	0

Table 8: BDT of twine for $(15, \gamma, \beta)$ where $\gamma \in F_4^2$ and $\beta \in F_4^2$

15,γ, β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	4	0	0	0	0	4	0	4	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	2	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
11	2	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 Analysis of ACE implementation

We ran the official implementation of ACE given by the designers and found the results mentioned in the next two subsections. The implementations are very well written and commented.

5.1 Padding Problem

In the algorithm they mention that padding of the message has to be done before absorption and generation of ciphertext. Whereas in the implementation this is not taken care of. We can see in Figure 6 that the implementation states to do the padding beforehand whereas in the implementation message is absorbed first and given out.

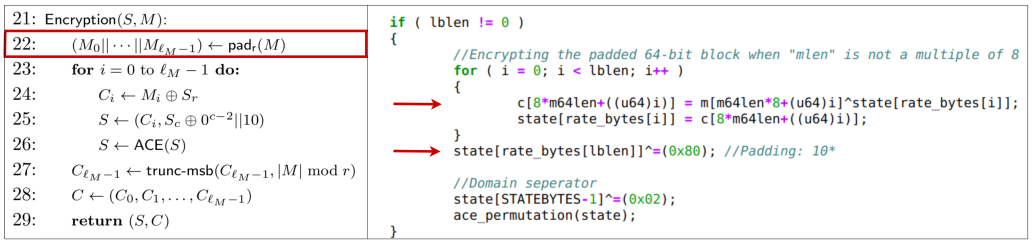


Figure 6: The algorithm and the implementation of `ACE`

This results in the pattern shown in Table 9. The rate size is 64 bits but even if we give a message smaller than that we get the output of the same size as that of the message instead of the padded output.

Table 9: Plaintexts(PT) and resultant Ciphertexts(CT) for Key = `AAAAAAAAAAAAAAAABBBBBBBBBBBBBBBB` and Nonce = `00000000000000000000000000000000`

PT	CT
99	18
9999	18C6
999999	18C69A
99999999	18C69A26
9999999999	18C69A26A5
999999999999	18C69A26A5F0
99999999999999	18C69A26A5F09C
9999999999999999	18C69A26A5F09C81
99999999999999999	18C69A26A5F09C819A
999999999999999999	18C69A26A5F09C819A4E
9999999999999999999	18C69A26A5F09C819A4E99
99999999999999999999	18C69A26A5F09C819A4E99AE
999999999999999999999	18C69A26A5F09C819A4E99AE88
9999999999999999999999	18C69A26A5F09C819A4E99AE88A4
99999999999999999999999	18C69A26A5F09C819A4E99AE88A4A8
999999999999999999999999999999	18C69A26A5F09C819A4E99AE88A4A865

5.2 Weak Key and Nonce class

We found that the cipher behaves in a very predictable way for particular Nonce and keys using which we can guess some of the hex values in the ciphertexts.

The results given in Table 10 are valid for a lot of Key and Nonce pairs aparat from what is shown here. The values marked 0 in PT $\bigoplus$ CT show that they remain unchaged after encryption.

We also found that for Key = `AAAAAAAAAAAAAAAABBBBBBBBBBBBBBBB` and Nonce = `00000000000000000000000000000000`, if all the half-words (4 bits) of plaintext are same

Table 10: Unchanged half-words

Key	Nonce	PT ⊕ CT
aaaaaaaaaaaaaaaaccccccccccccccccc	00 . . . 0	?????0?????????0
aaaaaaaaaaaaaaaaccccccccccccccccc	00 . . . 0	?????0?????????0
aaaaaaaaaaaaaaaabbbbbbbbbbbbbbbb	00 . . . 0	????0???????0???
aaaaaaaaaaaaaaaabbbbbbbbbbbbbbbb	aa . . . a	??????????0?0???

then we get a pallindromc sub-sequence of length 10 by picking the values in places which has - in − − −. − −.. − ... − − − −. This is also shown in Table 11, in first row the values in bold form a plalindromic subsequence. Values in the same positions in other rows also form a palindromic sub-sequence. Note that they also follow the result given in Table 10.

Table 11: Pallindromic Subsequence of length 10

PT	CT
0000000000000000	815F03BF3C690518
1111111111111111	904E12AE2D781409
2222222222222222	A37D219D1E4B273A
3333333333333333	B26C308C0F5A362B
4444444444444444	C51B47FB782D415C
5555555555555555	D40A56EA693C504D
6666666666666666	E73965D95A0F637E
7777777777777777	F62874C84B1E726F
8888888888888888	09D78B37B4E18D90
9999999999999999	18C69A26A5F09C81
AAAAAAAAAAAAAAAA	2BF5A91596C3AFB2
BBBBBBBBBBBBBBBB	3AE4B80487D2BEA3
CCCCCCCCCCCCCCCC	4D93CF73F0A5C9D4
DDDDDDDDDDDDDDDD	5C82DE62E1B4D8C5
EEEEEEEEEEEEEEEE	6FB1ED51D287EBF6
FFFFFFFFFFFFFFFF	7EA0FC40C396FAE7

6 Analysis of ACE permutation

ACE uses Simeck-64 (SB-64) for permutation ACE-STEP. There are 16 rounds per permutation and in each round of permutation Simeck-64 runs for 8 rounds. We did some analysis of attacks on Simeck-64 and also a new differential trail which can be used to launch improved attacks on Simeck.

6.1 Algebraic Degree of Simeck32

The Zero-sum property to estimate the degree of the cipher was very useful as in this cipher we don't have an s-box. In Hang Li et al. [2] report the algebraic degree of Simeck32 as given in Table 12. However in our implementation given in *'implementations'*[1] folder as *SIMECK_FOR_INTEGRAL_PROPERTY.ipynb* block 2 (also in Appendix 9), we found that algebraic degrees reported for Rounds 6 and 7 are not correct and they can instead be 15 and 19 respectively.

[1] My contribution

Table 12: Algebraic Degree of Simeck32

Rounds	Algebraic Degree
1	1
2	2
3	3
4	5
5	8
6	13
7	18
8	24
9	27
10	29
11	30
12	30
13	31
14	31

6.2 Integral Cryptanalysis of Simeck

The integral attack is a chosen plaintext attack that was initially developed to attack the block cipher Square [3]. The attack exploits the behaviour that the XOR sum of a structure of texts can be determined after several iterations of the round function. In order for this event to happen, the structure of plaintexts has to be in a particular format. The point before the XOR sum of the texts is used in a key recovery attack to identify possible round subkey bit values. The attack is particularly suited for block ciphers where all transformations operate on words (word-based block ciphers).

In [4] Kai Zhang et al. evaluated the security level on Simeck against integral cryptanalysis for the first time. They proposed some theoretical and experimental integral distinguishers on Simeck. More specifically, 12/14/16-round theoretical integral distinguishers on Simeck32/48/64 and some 15-round experimental integral distinguishers on Simeck32 are presented. With these integral distinguishers, Simeck32/48/64 reduced to 21/21/24 rounds respectively are attacked with integral cryptanalysis. Their findings are given in Table 13. Here A is for All property folowing bits, B is for Balanced property folowing bits and U is for Unidentified property folowing bits.

Table 13: Theoretical Integral Distinguishers on SIMECK [4]

Algorithm	Integral Form	Balanced bit
(11-round) Simeck32 (14-round)	(AACCCAACCAAACAAA AACCAAACAAAAAAAA)	$C_R(13)$
	(AAAAAAAAAAAAAAAC AAAAAAAAAAAAAAAA)	$C_R(15)$
Simeck48 (13-round)	(ACCCACCCAACCAAACAAAAAAAA ACCAACCAAACAAAAAAAAAAAAA)	$C_R(22)$
Simeck64 (15-round)	(ACCCCCCCCCCCACCCAACCAAACAAAAAAAA ACCCCCCACCCAACCAAACAAAAAAAAAAAA)	$C_R(28)$

They prepend one round to each of the integral distinguishers given in Table 13 to make them 12/15/14/16 round distnguishers. Then they use it for key recovery by decrypting 6/7/8 rounds for Simeck32/48/64. Figure 7 shows the key recovery for Simeck32 for using the 14 round distinguisher. We implemented this with the similar code as that of previous

section but changing the bit positions
footnoteMy contribution.

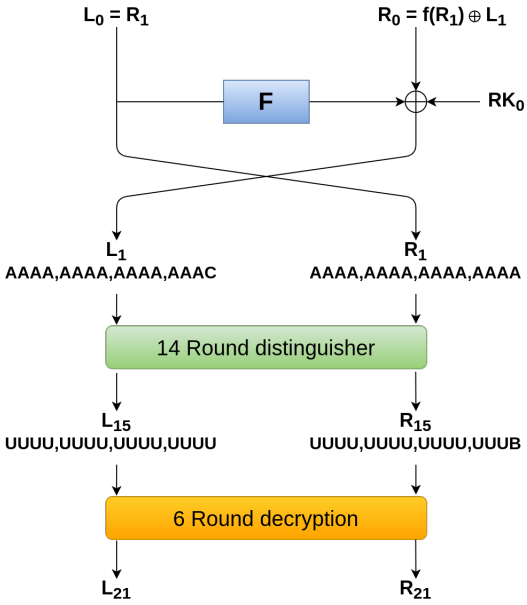


Figure 7: Integral attack on 21 round `Simeck`32 [4]

Table 14 shows the results they got for `Simeck`32/48/64.

Table 14: Summary of Integral cryptanalysis on `Simeck` [4]

Algorithm	Rounds Attacked	Data	Memory	Time	Success Prob.
`Simeck`32	12+6	$O(2^{23})$	$2^{43.59}$	2^{63}	1
	15+6	$O(2^{31})$	$2^{43.59}$	2^{63}	≈ 1
`Simeck`48	14+7	$O(2^{34})$	2^{66}	2^{95}	1
`Simeck`64	16+8	$O(2^{35})$	$2^{90.46}$	2^{127}	1

Hang Li et al. [2] extend the above attack by improving the distinguishers and decrypting 7/8/9 rounds for `Simeck`32/48/64. Their results are given in Tabel 16. Using the algebraic degrees given in Table 12 they realized that they can easily form a distinguisher for 12 rounds of `Simeck`32 instead of just 11 rounds done in previous work. Similarly they extend the distinguishers for `Simeck`48/64 to 17/20 rounds. Figure 8 shows how Integral attack is applied to `Simeck`32. The distinguishers they have used are given in Table 15.

Table 15: Theoretucal Integral Distinguishers on SIMECK [2]

Algorithm	Integral Form	Balanced bit
(12-round)	(CAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA)	$C_R(13)$
Simeck32 (14-round)	(AAAAAAAAAAAAAAAC AAAAAAAAAAAAAAAA)	$C_R(15)$
Simeck48 (17-round)	(CAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAA)	$C_R(23)$
Simeck64 (20-round)	(CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)	$C_R(31)$

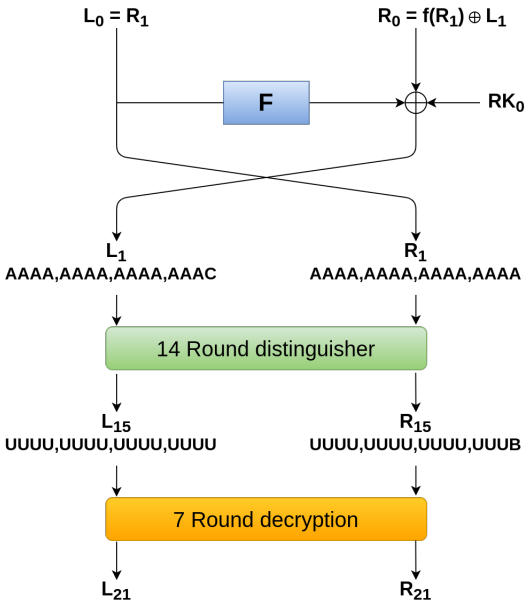


Figure 8: Integral attack on 22 round Simeck32 [2]

Table 16: Summary of Integral cryptanalysis on Simeck [2]

Algorithm	Rounds Attacked	Data	Memory	Time	Success Prob.
Simeck32	13+7	$O(2^{31})$	$2^{55.88}$	2^{63}	1
	15+7	$O(2^{31})$	$2^{55.88}$	2^{63}	1
Simeck48	18+8	$O(2^{47})$	$2^{82.52}$	2^{95}	1
Simeck64	21+9	$O(2^{63})$	$2^{109.26}$	$2^{127.3}$	1

6.3 Linear Cryptanalysis of Simeck

Linear cryptanalysis is a known plaintext attack introduced by Matsui [5] in 1993. However, the origins of the attack can be traced back to the observation regarding the S-box of the DES made by Shamir [6] in 1985. In a basic linear attack, a one-bit linear relationship between selected bits of the plaintext block P , ciphertext block C and master key block K is constructed as follows:

$$\lambda_P \cdot P \oplus \lambda_C \cdot C = \lambda_K \cdot K$$

where λ_i are bit masks and $\lambda_X \cdot X$ is a scalar product of λ_X and X. Stated differently, $\lambda_X \cdot X$ denotes the XOR sum (parity) of the bits of X masked (selected) by λ_X.

Kai Zhang et al. [7] analyzed the security evaluation on Simeck against zero-correlation linear cryptanalysis. The authors' proposed 11-, 13- and 15-round zero-correlation linear distinguishers on Simeck32/48/64 respectively and then zero-correlation linear cryptanalysis on 21, 24 and 28-round Simeck32/48/64. The linear distinguishers proposed by them are given in Table : 17 for Simeck32. Similar distinguisher can be made for 13/15 rounds of Simeck48/64 as given in [7]. The attack method is shown in Figure 9 and the results are shown in Table 18. They use miss in the middle attack to make these distinguishers. For e.g. for Simeck32 they make 11 round Distinguisher were we see a contardiction in 5th round. The implementation for this is done in *'implementations'* folder as (SIMECK_FOR ZERO Correlation linear cryptanalysis PROPERTY.ipynb)[2] and also in Appendix 9.

Table 17: 11-round ZC linear distinguisher on Simeck32

Rounds	Γ_L	Γ_R
0	1000000000000000	0000000000000000
1	0000000000000000	1000000000000000
2	1000000000000000	*1000*0000000000
3	*1000*0000000000	**100**000*00000
4	**100**000*00000	***10***00**000*
5	***10***00**000*	*********0***00*
5	***0**********	**00**********0*
6	**00**********0*	*000****10***00*
7	*000****10***00*	00000**100**000*
8	00000**100**000*	00000*1000*00000
9	00000*1000*00000	0000010000000000
10	0000010000000000	0000000000000000
11	0000000000000000	0000010000000000

Table 18: Summary of Linear cryptanalysis on Simeck [7]

Algorithm	Rounds Attacked	Data	Memory	Time
Simeck32	11+5+5	$O(2^{32})$	$2^{45.67}$	$2^{58.78}$
Simeck48	13+6+5	$O(2^{48})$	$2^{58.99}$	$2^{81.64}$
Simeck64	15+7+6	$O(2^{64})$	$2^{97.67}$	$2^{123.06}$

[2] My contribution

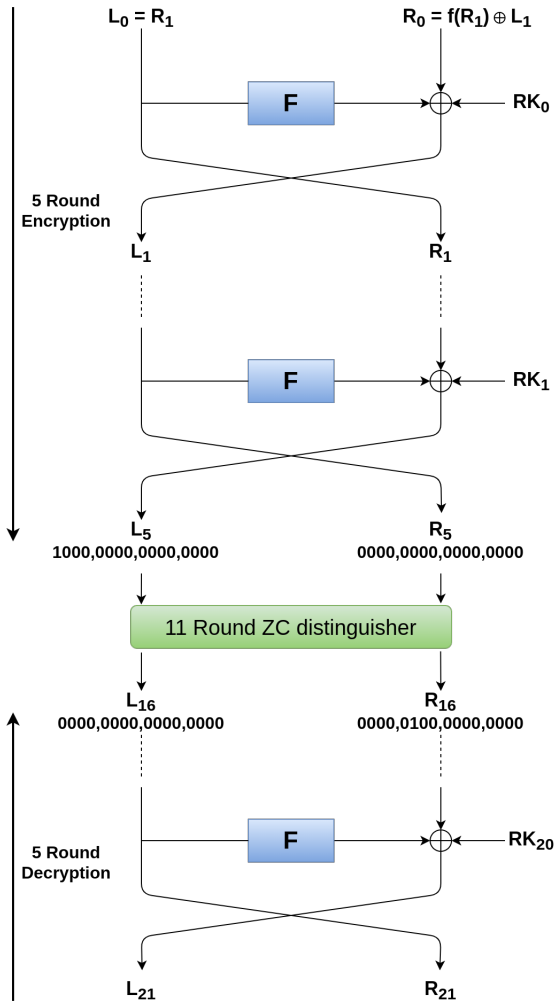


Figure 9: Initial five rounds encryption and final five rounds decryption with 11 round distiguisher on `Simeck`32[7]

6.4 Differential Cryptanalysis of `Simeck`

Differential cryptanalysis is a chosen plaintext attack that exploits the existence of a linear relationship between the encryption of two plaintexts using the same secret master key. Given a pair of plaintexts P and P' = $P \oplus \Delta_P$, the ciphertexts after R rounds are respectively Q and Q' = $Q \oplus \Delta_Q$ with high probability. The expected difference at this round is used in a key recovery attack to identify possible subkey

In [8] Kexin Qiao et al. reveal some details in implementing the dynamic key-guessing techniques and thus make it easy to launch a differential attack with these techniques on `Simeck` like blockciphers. Specfically, they write a program to calculate the complexity in dynamic key-guessing procedure and then estimate the complexities in differential cryptanalysis on family of `Simeck` block ciphers. They find a 13-round differential of `Simeck`32/64 with lower hamming weight with probability $2^{-29.64}$. Applying this differential to attack `Simeck` with dynamic key-guessing techniques. They use a differential characteristic of 13-round with probability 2^{-38}. They encrypt 3 rounds and decrypt 5 rounds and use the 13-round distiguisher to launch a 21 round attack and do dynamic key-guessing. The 13 round distinguisher they use is shown in Table 19. The results they

Table 20: Results for Differential Attacks on Round Reduced Simeck

Versions	Attacked Rounds	$\|sk\|$	λ_e	λ_r	Chosen Count	Data Complexity	Time Complexity	Success Prob.
Simeck32/64	21	53	$2^{-2.678}$	3.29	4	2^{30}	$2^{48.52}$	41.7%
Simeck32/64	22	54	2^{-1}	2.56	3	2^{32}	$2^{57.88}$	47.1%
Simeck48/96	28	75	$2^{-8.365}$	2.54	3	2^{46}	$2^{68.31}$	46.8%
Simeck64/128	34	82	$2^{-1.678}$	3.94	4	2^{63}	$2^{116.34}$	55.5%
Simeck64/128	35	118	$2^{-1.678}$	3.94	4	2^{63}	$2^{116.34}$	55.5%

got are given in Table 20.

Table 19: A differential characteristic of 13-round Simeck32 with probability 2^{-38}

Rounds	Input	differences
0	0000000000000000	0000000000000010
1	0000000000000010	0000000000000000
2	0000000000000100	0000000000000010
3	0000000000010000	0000000000001010
5	0000000000111010	0000000000010000
6	0000000000001100	0000000000111010
7	0000000000101010	0000000000001100
8	0000000000010000	0000000000101010
9	0000000000001010	0000000000010000
10	0000000000000100	0000000000001010
11	0000000000000010	0000000000000100
12	0000000000000000	0000000000000010
13	0000000000000010	0000000000000000

6.5 Impossible Differential Attack on Simeck

Differential cryptanalysis deals with the occurrences of high probability events. On the other hand, an impossible differential [9] deals with the occurrences of zero-probability events. The technique is a sieving attack which eliminates wrong subkey guesses that lead to contradictions. Eventually, only the right possible subkeys remain. A miss-in-the-middle technique is used in which two input differences are propagated to the middle of a cipher, one from the top and one from the bottom. If the resulting output differences in the middle are distinct, then a contradiction occurs. Subkey guesses that cause such an event to happen are discarded. The attack can also be used with low probability differentials and conditional characteristics (or differentials), and can be combined with linear cryptanalysis.

The basic idea is that they look for differentials (differences between pairs of plaintexts that have some percentage of holding true for the corresponding differences between pairs of ciphertexts) that are impossible; i.e., that can not happen. It turns out that in many cases these impossible differentials are more powerful than normal differentials, and can be used to attack encryption algorithms.

6.5.1 Single Key Setting

Sadeh Sadeghi and Nasour Bagheri [10] employ an improved miss-in-the-middle approach to find zero correlation linear distinguishers and impossible differentials on Simeck48 and

Simeck64. Based on this novel technique, they present zero-correlation linear approximations for 15-round Simeck48 and 17-round Simeck64 and these zero-correlation linear approximations improve the previous best result by two rounds for Simeck48 and Simeck64. Moreover, they attack 27-round Simeck48 and 31-round Simeck64 based on these zero-correlation linear distinguishers. In addition, due to the duality of zero-correlation and impossible differential, they search for the impossible differential characteristics for Simeck48 and Simeck64 so that they will be able to present 15-round Simeck48 and 17-round Simeck64 while the best previously known results were 13-round impossible differentials for Simeck48 and 15-round impossible differentials for Simeck64. Moreover, they propose impossible differential attacks on 22-round Simeck48 and 24-round Simeck64 based on these impossible differential characteristics.

Figure 10 shows an 11 Round impossible Differential trail for Simeck64. The bits in Red contradict. Here we also found a new 13 Round Trail given in Figure 11[3] . They extend this 11 round trail to 17 round by fixing some values shown in Figure 12. All these implementations[4] are given in *'implementations'* folder as SIMECK_impossible_differential_single_key and also in Appendix 9.

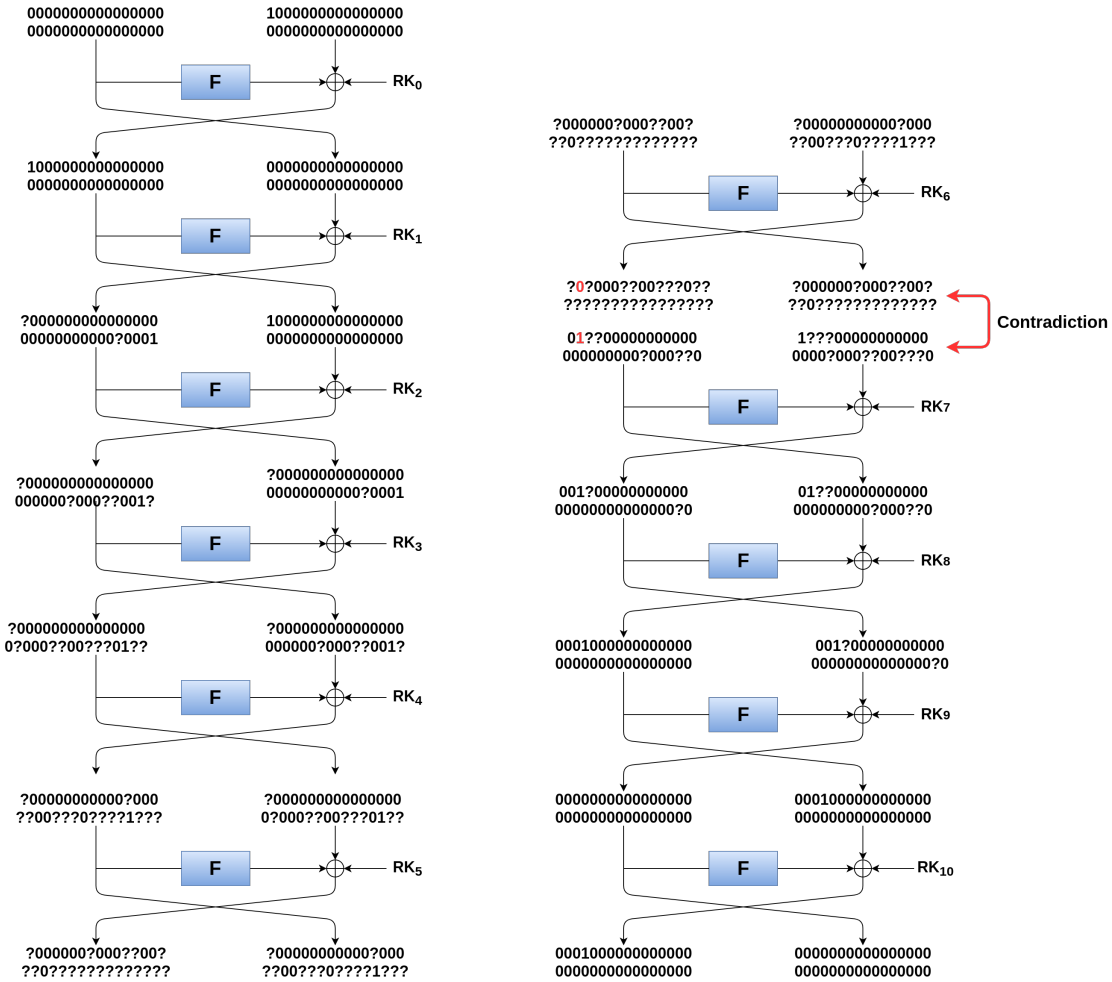


Figure 10: 11 Round impossible Differential trail for Simeck64

[3]My contribution
[4]My contribution

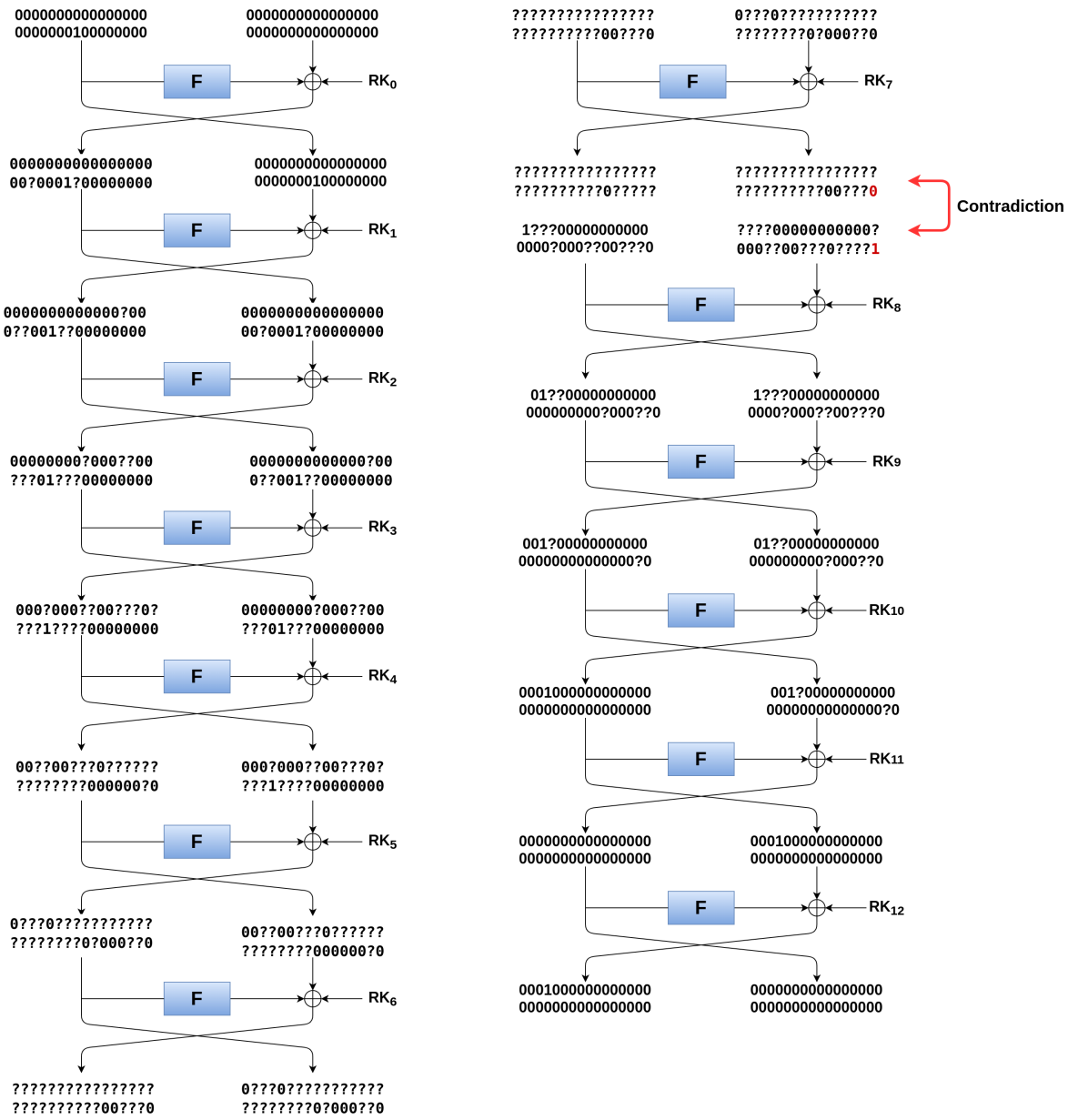


Figure 11: 13 Round impossible Differential trail for `Simeck64`

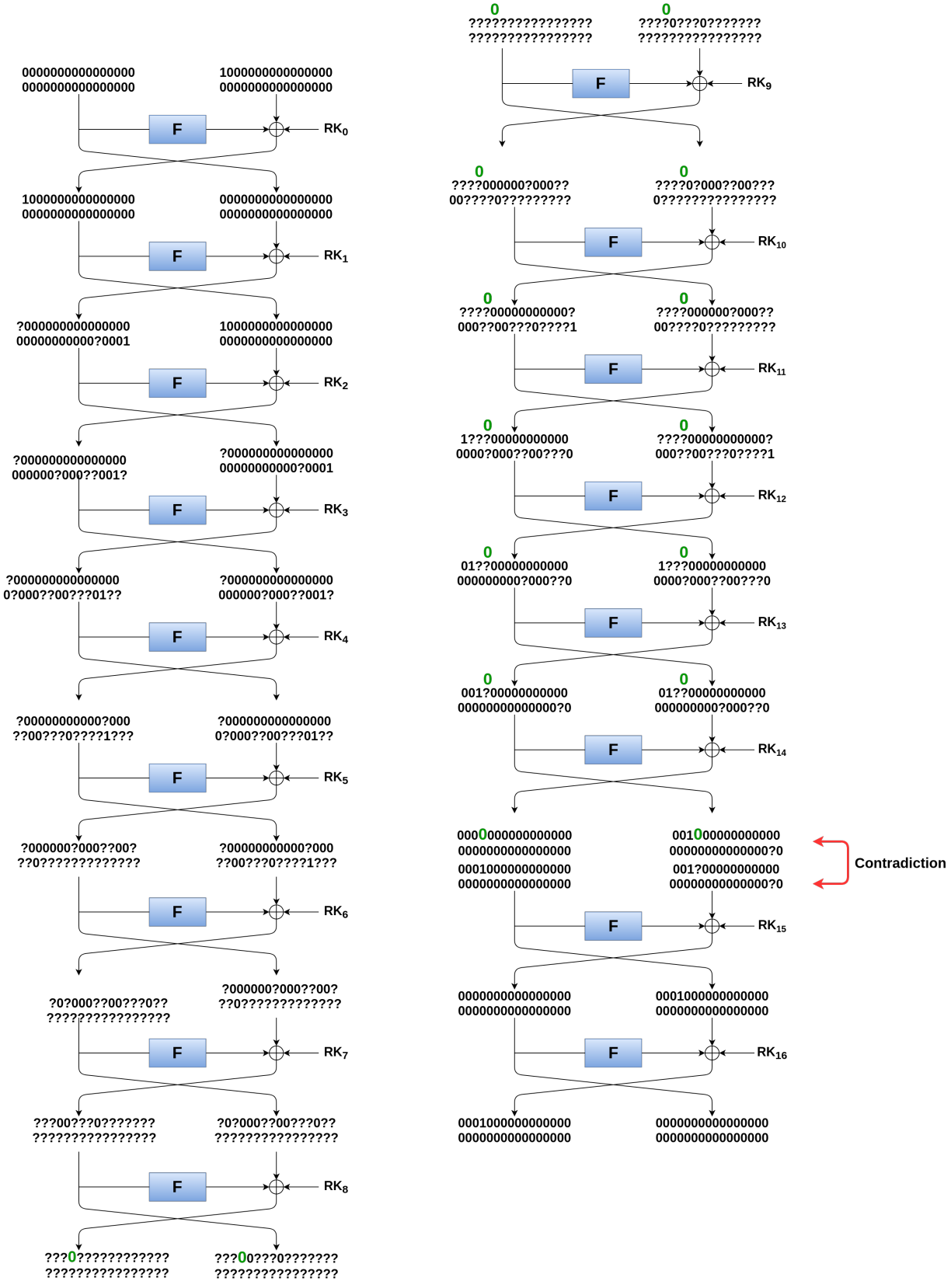


Figure 12: 17 Round impossible Differential trail for `Simeck`64

6.5.2 Related Key Setting

In the previous paper [10] Sadeh Sadeghi and Nasour Bagheri had considered single key model, here in this paper [11][5] they consider related-key impossible differential distinguishers for the variants of Simeck. They first propose some distinguishers on Simeck using the miss-in-the-middle approach. More specifically, 15/16/19-round related-key impossible differential distinguishers on Simeck32/48/64 are presented first while the best previously known results were 11/15/17-round on Simeck32/48/64 in the single-key setting. Afterwards, using MILP approach, they automatically prove that these characteristics are the best related-key impossible differentials of Simeck when we limit the input and output differences to 1 active bit.

They first show a 15-round key difference which starts with the master key differences of the form $(\Delta_{k1}||\Delta_{k2}||\Delta_{k3}||\Delta_{k4}) = (10...0||0||0||0)$ that depicted in Figure 13 and then show a 15-round related-key impossible differential against Simeck32 using this 15-round key difference depicted in Figure 14. This implementation [6] is given in *'implementations'* folder as SIMECK_impossible_differential_related_key and also in Appendix 9.

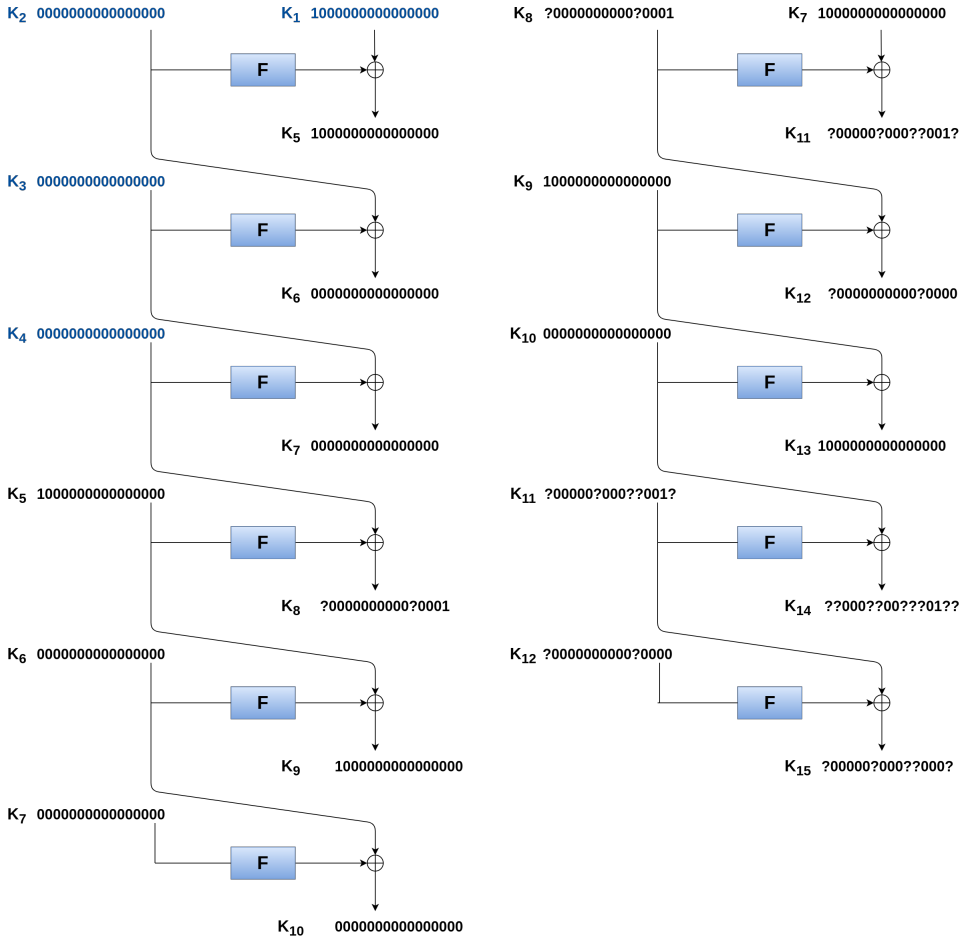


Figure 13: 15-round key differences for Simeck32 [11]

[5]Paper Assigned to us
[6]My contribution

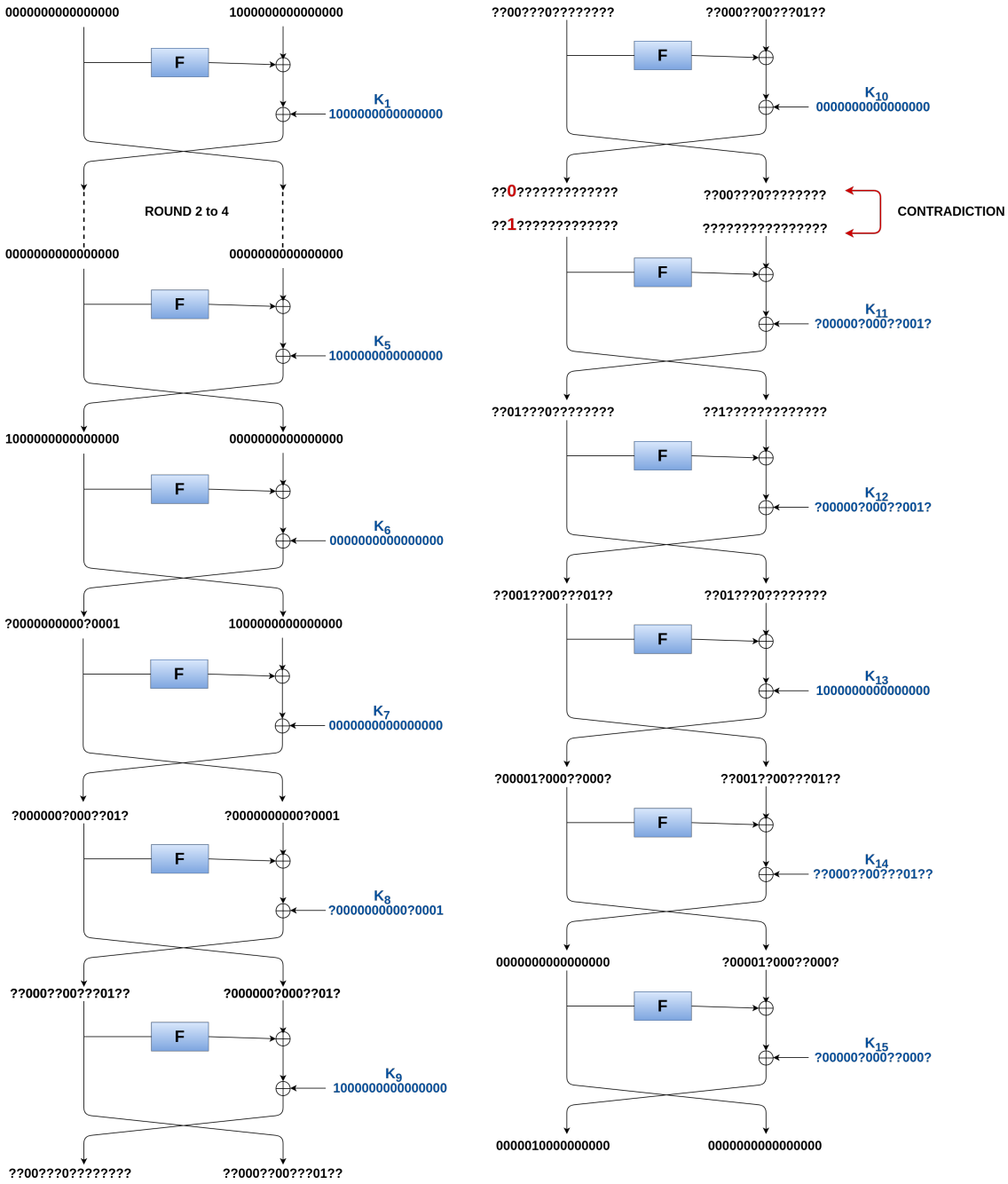


Figure 14: 15-round Related key differences for `Simeck`32 [11]

In this paper, the authors utilize MILP approach to search best related-key impossible differential characteristics for `Simeck`.They summarize all the related-key impossible differential characteristics which is obtained from the MILP method for `Simeck` in the Property 1. Note that if they limit the input and output differences to 1 active bit, MILP proves that the maximal number of rounds of relatedkey impossible differential characteristics for `Simeck`32, `Simeck`48 and `Simeck`64 is 15, 16 and 19-round respectively. Therefore, these characteristics are the best relatedkey impossible differentials of `Simeck` when they limit

the input and output differences to 1 active bit.

Property 1 : Assume that $(0, \Delta_{R_{in}}[i]) \nrightarrow (\Delta_{L_{out}}[(i+j)\%n], 0)$ with master key differences $(\Delta_{k_1}[i], 0, 0, 0)$ is a r-round related-key impossible differential for `Simeckn` where $\Delta_{R_{in}}$, $\Delta_{L_{out}}$, $\Delta_{K_1} \in F_2^n$ - $\{0\}$. Then for any i, where $0 \leq$ i $\leq$ n - 1, one can construct a set of :

- 15-round related-key impossible differentials for `Simeck`32 by choosing j = 5. For e.g. if we choose i = 0 and j = 5, we can derive a 15-round related-key impossible differential characteristic $(0, \Delta_{R_{in}}[0]) \nrightarrow (\Delta_{L_{out}}[5], 0)$.

- 16-round related-key impossible differentials for `Simeck`48 by choosing j $\in \{1, 4, 5, 6, 8, 9, 10, 14\}$.For e.g. if we pick i = 0 and choose j = 5, we can obtain a 16-round related-key impossible differential characteristic $(0, \Delta_{R_{in}}[0]) \nrightarrow (\Delta_{L_{out}}[5], 0)$ with master key differences $(\Delta_{k_1}[i], 0, 0, 0)$.

- 19-round related-key impossible differentials for `Simeck`64 by choosing j $\in \{3, 7\}$. For e.g. if we pick i = 0 and choose j = 3, we can obtain a 19-round related-key impossible differential characteristic $(0, \Delta_{R_{in}}[0]) \nrightarrow (\Delta_{L_{out}}[3], 0)$ with master key differences $(\Delta_{k_1}[i], 0, 0, 0)$

These results are verified by the MILP of `Simeck`. This implementation[7] is given in $'implementations'$ folder as SIMECK_milp.cpp and also in Appendix 9.

6.6 MILP of `Simeck`

Mixed Integer Linear Programming was first introduced by Mouha et al. [12] , which was used to minimize the number of active s-boxes in a differential or linear characteristic. Following that, Sun et al. in [13] extended Mouha et al.'s work from byte oriented ciphers to bit oriented ciphers. MILP has received many applications recently to automate cryptanalysis of block ciphers and hash functions.

Mixed Integer Linear Programming (MILP) can be used to search the best characteristic in related-key impossible differential cryptanalysis. In [14], Cui et al. proposed a MILP-based impossible differential characteristic search method for block ciphers. The MILP problem is a problem of optimizing the value of the objective function for values satisfying the linear constraint equation. In order to search for the MILP-based impossible differential characteristic, the objective function is set to the expression expressing the probability of the differential characteristic and the linear constraint expression is configured to constitute the cipher system. Therefore, with respect to the set cryptosystem, the optimum differential characteristic probability constituting the cryptosystem corresponding to the answer to the MILP problem is obtained. However, if the answer to the MILP problem cannot be obtained for a specific input or output differential, it means that the differential characteristic cannot be configured in the specified cryptosystem for that input and output differential value. Therefore, the input and output differentials become invalid differential characteristics of the given cryptosystem.

Recently, Sasaki et al. proposed a new impossible differential search tool inspired by the design and cryptanalysis aspects in [15] using MILP. They presented an approach for evaluating s-boxes including 8×8 s-boxes in impossible differential cryptanalysis which was missing in [14].

[7] My contribution

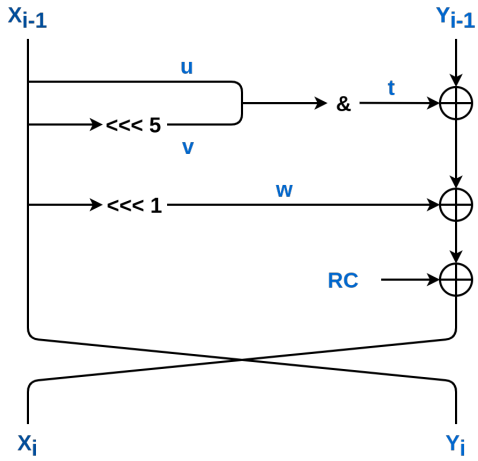


Figure 15: Diagram for `Simeck` MILP equation reference

Figure 15 shows the diagram i have drawn for `Simeck`'s MILP. The equations used are given below. We minimize the number of active bits throughout the trail as there is no s-box here. A sample .lp file for 10 rounds of `Simeck`32 is given *'implementations'* folder as $SIMECK\_32\_10.lp$. The implementation[8] of the complete MILP is given in *'implementations'* folder as SIMECK_milp.cpp and also in Appendix 9.

$$x_{ij} - y_{i+1j} - u_{ij} - v_{ij} - w_{ij} = 0$$
$$t_{ij} - u_{ij} >= 0$$
$$t_{ij} - v_{ij+5} >= 0$$
$$t_{ij} - u_{ij} - v_{ij+5} <= 0$$
$$x_{i+1j} - y_{ij} - t_{ij} - w_{ij+1} = 0$$

We also used this to verfiy the Property 1 given in previous section.Zejung Xiang et al. [16] extend this MILP for `Simeck` to search integral distinguishers of block ciphers based on division property. They also use H-representation and Convex hull for the same. The equations given above are used for division property along with stopping rules. They successfully find 14, 17 and 20 round distinguisher for `Simeck`32, `Simeck`48 and `Simeck`64 respectively.

6.7 Differential Fault Attack (DFA) on `Simeck`

In DFA, the attackers investigate the difference in values generated from the fault-free calculation and the faulty calculation to recover the secret information. The setup for the DFA attack on the encryption of a block cipher is explained as follows. The attackers perform encryption of the same plaintext, P , at least twice. The first execution of the encryption is without any faults so that a fault-free ciphertext, C, is obtained. Then, the encryption of the same plaintext, P , is repeated, during which the fault injection is performed by the attackers. After a successful fault injection occurs, a faulty ciphertext, C', is obtained. The DFA attacks recover the key using the difference of C and C' , that is, ΔC, and the knowledge about the injected fault.

Venua Nalla et al. [17] showed that `Simeck` is vulnerable to fault attacks and demonstrate two fault attacks on `Simeck`. The first is a random bit-flip fault attack which recovers the n-bit last round key of `Simeck` using on average about n/2 faults and the second is a more practical, random byte fault attack which recovers the n-bit last round key of `Simeck` using on average about n/6.5 faults.

[8] My contribution

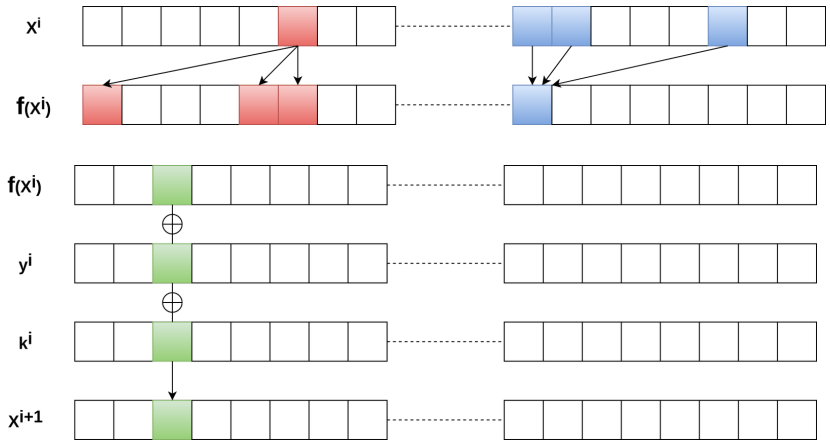


Figure 16: Bits affected

Figure 16 shows the bits that get affected by a single bit flip. They then use these bit inter-relation equations to recover two of the last round key bits per fault. Thus requiring n/2 faults. However [9], it must be noted that the recovery of last key is not sufficient to recover the master key. For that they need to recover atleast 4 consecutive round keys. Thus requiring 4 times the ault they need now.

6.8 Performance Metrices of ACE-STEP

The codes for this section are given in *'implemnetations/ace_hardware'* folder. Throughput and power are given at 100 kHz.

Table 21: ACE permutation implementation results

Key Size (bits)	Bock Size (bits)	Latency (Cycles/block)	Throughput (Kbps)	Tech (μM
128	64	128	266.66	0.065

Area (GE)	Efficiency (Kbps/KGE)	Power (μW)	Energy (μJ)
2716	98.18	1.54268	3.085

7 Conclusion and Future Scope

This report investigates the security level of ACE and it's permutation based on Simeck family of ciphers against integral cryptanalysis, linear cryptanalysis, diferential cryptanalysis, impossible differential cryptanalysis in both single key and related key settings and DFA. We also formulates MILP to verify both the impossible differential property and division property. We came up with a few weaknesses in the implementation and cipher design and also introduced a new improved impossible differential trail. We also found some incorrect observations made in some of the previous works. Through this analysis we got to learn various methods and technologies of analyzing a cipher.

The future scope of this work would be to improve the attacks done on Simeck which in turn reduces the strength of ACE. Also studying linear hulls and attacks on Simeck using linear hulls comes within the domain of future work as the results given by Lingyue Qin et al. [18] using linear hulls are really good and much better than most of the existing work.

[9] My contribution

8 Bibliography

References

[1] "GIFT: A small present (full version)," *IACR Cryptology ePrint Archive*, vol. 2017, p. 760, 2017, withdrawn. [Online]. Available: http://eprint.iacr.org/2017/760

[2] H. Li, J. Ren, and S. Chen, "Improved integral attack on reduced-round simeck," *IEEE Access*, vol. 7, pp. 118 806–118 814, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2936834

[3] J. Daemen, L. R. Knudsen, and V. Rijmen, "The block cipher square," in *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, ser. Lecture Notes in Computer Science, E. Biham, Ed., vol. 1267. Springer, 1997, pp. 149–165. [Online]. Available: https://doi.org/10.1007/BFb0052343

[4] K. Zhang, J. Guan, B. J. Hu, and D. Lin, "Integral cryptanalysis on simeck," *2016 Sixth International Conference on Information Science and Technology (ICIST)*, pp. 216–222, 2016.

[5] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, ser. Lecture Notes in Computer Science, T. Helleseth, Ed., vol. 765. Springer, 1993, pp. 386–397. [Online]. Available: https://doi.org/10.1007/3-540-48285-7_33

[6] A. Shamir, "On the security of DES," in *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, ser. Lecture Notes in Computer Science, H. C. Williams, Ed., vol. 218. Springer, 1985, pp. 280–281. [Online]. Available: https://doi.org/10.1007/3-540-39799-X_22

[7] K. Zhang, J. Guan, B. Hu, and D. Lin, "Security evaluation on simeck against zero-correlation linear cryptanalysis," *IET Information Security*, vol. 12, no. 1, pp. 87–93, 2018. [Online]. Available: https://doi.org/10.1049/iet-ifs.2016.0503

[8] K. Qiao, L. Hu, and S. Sun, "Differential security evaluation of simeck with dynamic key-guessing techniques," in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy, ICISSP 2016, Rome, Italy, February 19-21, 2016*, O. Camp, S. Furnell, and P. Mori, Eds. SciTePress, 2016, pp. 74–84. [Online]. Available: https://doi.org/10.5220/0005684400740084

[9] E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials," *J. Cryptology*, vol. 18, no. 4, pp. 291–311, 2005. [Online]. Available: https://doi.org/10.1007/s00145-005-0129-3

[10] S. Sadeghi and N. Bagheri, "Improved zero-correlation and impossible differential cryptanalysis of reduced-round SIMECK block cipher," *IET Information Security*, vol. 12, no. 4, pp. 314–325, 2018. [Online]. Available: https://doi.org/10.1049/iet-ifs.2016.0590

[11] ——, "Security analysis of SIMECK block cipher against related-key impossible differential," *Inf. Process. Lett.*, vol. 147, pp. 14–21, 2019. [Online]. Available: https://doi.org/10.1016/j.ipl.2019.02.009

[12] N. Mouha, Q. Wang, D. Gu, and B. Preneel, "Differential and linear cryptanalysis using mixed-integer linear programming," in *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. Wu, M. Yung, and D. Lin, Eds., vol. 7537. Springer, 2011, pp. 57–76. [Online]. Available: https://doi.org/10.1007/978-3-642-34704-7_5

[13] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song, "Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers," in *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, ser. Lecture Notes in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 8873. Springer, 2014, pp. 158–178. [Online]. Available: https://doi.org/10.1007/978-3-662-45611-8_9

[14] T. Cui, K. Jia, K. Fu, S. Chen, and M. Wang, "New automatic search tool for impossible differentials and zero-correlation linear approximations," *IACR Cryptology ePrint Archive*, vol. 2016, p. 689, 2016. [Online]. Available: http://eprint.iacr.org/2016/689

[15] Y. Sasaki and Y. Todo, "New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers," 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*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10212, 2017, pp. 185–215. [Online]. Available: https://doi.org/10.1007/978-3-319-56617-7_7

[16] Z. Xiang, W. Zhang, Z. Bao, and D. Lin, "Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers," in *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. H. Cheon and T. Takagi, Eds., vol. 10031, 2016, pp. 648–678. [Online]. Available: https://doi.org/10.1007/978-3-662-53887-6_24

[17] V. Nalla, R. A. Sahu, and V. Saraswat, "Differential fault attack on SIMECK," in *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC, Prague, Czech Republic, January 20, 2016*, M. Palkovic, G. Agosta, A. Barenghi, I. Koren, and G. Pelosi, Eds. ACM, 2016, pp. 45–48. [Online]. Available: https://doi.org/10.1145/2858930.2858939

[18] L. Qin, H. Chen, and X. Wang, "Linear hull attack on round-reduced simeck with dynamic key-guessing techniques," in *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. K. Liu and R. Steinfeld, Eds., vol. 9723. Springer, 2016, pp. 409–424. [Online]. Available: https://doi.org/10.1007/978-3-319-40367-0_26

9 Appendix

Table 22: BDT of twine

0,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
11	2	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
12	4	0	0	0	0	4	0	0	0	4	0	0	4	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
2,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
5	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	4	0	0	0	4	0	0	0	0	0	4	0	0	0	4	0
11	2	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 23: BDT of twine

3,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
8	2	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
15	4	0	0	0	0	0	0	4	4	0	0	0	0	0	0	4
4,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	4	0	0	0	0	0	4	0	0	0	0	4	0	4	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
5,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 24: BDT of twine

6,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	4	0	0	0	0	4	0	0	0	4	0	0	4	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
13	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
5	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	2	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
9	4	0	0	0	0	4	0	0	0	4	0	0	4	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
5	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2

Table 25: BDT of twine

9,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	4	0	0	0	0	0	0	4	4	0	0	0	0	0	0	4
8	2	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	2	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
10,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
8	4	0	0	0	0	0	0	4	4	0	0	0	0	0	0	4
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
13	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
11,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
7	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	2	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
12	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
13	4	0	0	0	0	0	4	0	0	0	0	4	0	4	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 26: BDT of twine

12,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	2	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
9	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
14	4	0	0	0	4	0	0	0	0	0	4	0	0	0	4	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	2	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	2	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
12	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14,γ,β	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	0	0	0	4	0	0	0	0	0	4	0	0	0	4	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2

Code for `Simeck`'s Linear Cryptanalysis property

```
#!/usr/bin/env python
# coding: utf-8

# In[11]:


# forward direction - encryption
input_left = "1000000000000000"
input_right= "0000000000000000"
rounds=5
for i in range(rounds):
    temp_right=[i for i in input_left]
    for j in range(16):
        if input_right[j] == '1':
            temp_right[j]="?"
            if temp_right[(j+1)%16]!='?':
                temp_right[(j+1)%16]="1"
            temp_right[(j+5)%16]="?"
        if input_right[j] == '?':
            temp_right[j]="?"
            temp_right[(j+1)%16]="?"
            temp_right[(j+5)%16]="?"
    input_left=input_right
    input_right=""
    for j in temp_right:
        input_right+=j
    print(input_left,input_right)




# In[13]:


# backward direction - encryption
input_left = "00000000000000000000000000000000"
input_right= "00000100000000000000000000000000"
rounds=6
for i in range(rounds):
    temp_left=[i for i in input_right]
    for j in range(16):
        if input_left[j] == '1':
            temp_left[j]="?"
            if temp_left[(j+1)%16]!='?':
                temp_left[(j+1)%16]="1"
            temp_left[(j+5)%16]="?"
        if input_left[j] == '?':
            temp_left[j]="?"
            temp_left[(j+1)%16]="?"
            temp_left[(j+5)%16]="?"
    input_right=input_left
    input_left=""
    for j in temp_left:
        input_left+=j
    print(input_left,input_right)



# In[ ]:
```

Code for `Simeck`'s integral property

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


NUM_ROUNDS = {
    # (block_size, key_size): num_rounds
    (32, 64): 32,
    (48, 96): 36,
    (64, 128): 44,
}


def get_sequence(num_rounds):
    if num_rounds < 40:
        states = [1] * 5
    else:
        states = [1] * 6

    for i in range(num_rounds - 5):
        if num_rounds < 40:
            feedback = states[i + 2] ^ states[i]
        else:
            feedback = states[i + 1] ^ states[i]
        states.append(feedback)
```

```
    return tuple(states)


class Simeck:
    def __init__(self, block_size, key_size, master_key):
        assert (block_size, key_size) in NUM_ROUNDS
        assert 0 <= master_key < (1 << key_size)
        self._block_size = block_size
        self._key_size = key_size
        self._word_size = block_size // 2
        self._num_rounds = NUM_ROUNDS[(block_size, key_size)]
        self._sequence = get_sequence(self._num_rounds)
        self._modulus = 1 << self._word_size
        self.change_key(master_key)

    def _LROT(self, x, r):
        assert 0 <= x < self._modulus
        res = (x << r) % self._modulus
        res |= x >> (self._word_size - r)
        return res

    def _round(self, round_key, left, right):
        assert 0 <= round_key < self._modulus
        assert 0 <= left < self._modulus
        assert 0 <= right < self._modulus
        temp = left
        left = right ^ (left & self._LROT(left, 5))             ^ self._LROT(left, 1) ^ round_key
        right = temp
        # print hex(round_key), hex(left), hex(right)
        return left, right

    def change_key(self, master_key):
        assert 0 <= master_key < (1 << self._key_size)
        states = []
        for i in range(self._key_size //self._word_size):
            states.append(master_key % self._modulus)
            master_key >>= self._word_size

        constant = self._modulus - 4
        round_keys = []
        for i in range(self._num_rounds):
            round_keys.append(states[0])
            left, right = states[1], states[0]
            left, right = self._round(constant ^ self._sequence[i],
                                      left, right)
            states.append(left)
            states.pop(0)
            states[0] = right

        self.__round_keys = tuple(round_keys)

    def encrypt(self, plaintext,num_rounds):
        assert 0 <= plaintext < (1 << self._block_size)
        left = plaintext >> self._word_size
        right = plaintext % self._modulus

        for idx in range(num_rounds):
            left, right = self._round(self.__round_keys[idx],
                                      left, right)

        ciphertext = (left << self._word_size) | right
        return ciphertext


def print_test_vector(block_size, key_size, key, plain, cipher):
    print ('Simeck', block_size, key_size)
    print ('key', hex(key)[2:].rstrip('L').zfill(key_size // 4))
    print ('plaintext', hex(plain)[2:].rstrip('L').zfill(block_size // 4))
    print ('ciphertext', hex(cipher)[2:].rstrip('L').zfill(block_size // 4))


# In[9]:


import math
key64 = 0x191811100908010f
ciphertext32_1,ciphertext32_2,ciphertext32_3,ciphertext32_4,ciphertext32_5,ciphertext32_6,ciphertext32_7,ciphertext32_8=0,0,0,0,0,0,0,0
simeck32 = Simeck(32, 64, key64)
for i in range(2**15):
    plaintext32  = i<<17 #^ 0x19181170
    ciphertext32_1 ^= simeck32.encrypt(plaintext32,1)
    ciphertext32_2 ^= simeck32.encrypt(plaintext32,2)
    ciphertext32_3 ^= simeck32.encrypt(plaintext32,3)
    ciphertext32_4 ^= simeck32.encrypt(plaintext32,4)
    ciphertext32_5 ^= simeck32.encrypt(plaintext32,5)
    ciphertext32_6 ^= simeck32.encrypt(plaintext32,6)
    ciphertext32_7 ^= simeck32.encrypt(plaintext32,7)
    ciphertext32_8 ^= simeck32.encrypt(plaintext32,8)
print(i,"  ",hex(ciphertext32_1),"\t",hex(ciphertext32_2),"\t",hex(ciphertext32_3),"\t",hex(ciphertext32_4),"\t",hex(ciphertext32_5),"\t",hex(ciphertext32_6),"\t",hex(ciphertext3


# In[ ]:
```

Code for `Simeck`'s Impossible differential in single key setting

```
#!/usr/bin/env python
# coding: utf-8

# In[45]:


# forward direction - encryption
input_left = "00000000000000000000000000000000"
input_right= "10000000000000000000000000000000"
rounds=7
p=32
for i in range(rounds):
    temp_right=[i for i in input_right]
    for j in range(p):
        if input_left[j] == '1':
            temp_right[j]="?"
            if temp_right[(j-1)%p]!='?' :
                if temp_right[(j-1)%p] == "1":
                    temp_right[(j-1)%p]="0"
                else:
                    temp_right[(j-1)%p]="1"
            temp_right[(j-5)%p]="?"
        if input_left[j] == '?':
            temp_right[j]="?"
            temp_right[(j-1)%p]="?"
            temp_right[(j-5)%p]="?"
    input_right=input_left
    input_left=""
    for j in temp_right:
        input_left+=j
    print(i+1,input_left,input_right)




# In[46]:


# backward direction - encryption
input_left = "00010000000000000000000000000000"
input_right= "00000000000000000000000000000000"
rounds=4
p=32
for i in range(rounds):
    temp_left=[i for i in input_left]
    for j in range(p):
        if input_right[j] == '1':
            temp_left[j]="?"
            if temp_left[(j-1)%p]!='?' :
                if input_left[(j-1)%p] == "1":
                    temp_left[(j-1)%p]="0"
                else:
                    temp_left[(j-1)%p]="1"
            temp_left[(j-5)%p]="?"
        if input_right[j] == '?':
            temp_left[j]="?"
            temp_left[(j-1)%p]="?"
            temp_left[(j-5)%p]="?"
    input_left=input_right
    input_right=""
    for j in temp_left:
        input_right+=j
    print(input_left,input_right)




# In[47]:


# new 13 round distinguisher
# forward direction - encryption
input_left = "00000000000000000000000100000000"
input_right= "00000000000000000000000000000000"
rounds=8
p=32
for i in range(rounds):
    temp_right=[i for i in input_right]
    for j in range(p):
        if input_left[j] == '1':
            temp_right[j]="?"
            if temp_right[(j-1)%p]!='?' :
                if temp_right[(j-1)%p] == "1":
                    temp_right[(j-1)%p]="0"
                else:
                    temp_right[(j-1)%p]="1"
            temp_right[(j-5)%p]="?"
        if input_left[j] == '?':
            temp_right[j]="?"
            temp_right[(j-1)%p]="?"
            temp_right[(j-5)%p]="?"
    input_right=input_left
    input_left=""
    for j in temp_right:
```

```
            input_left+=j
        print(i+1,input_left,input_right)




# In[48]:


# backward direction - encryption
input_left = "00010000000000000000000000000000"
input_right= "00000000000000000000000000000000"
rounds=5
p=32
for i in range(rounds):
    temp_left=[i for i in input_left]
    for j in range(p):
        if input_right[j] == '1':
            temp_left[j]="?"
            if temp_left[(j-1)%p]!='?' :
                if input_left[(j-1)%p] == "1":
                    temp_left[(j-1)%p]="0"
                else:
                    temp_left[(j-1)%p]="1"
            temp_left[(j-5)%p]="?"
        if input_right[j] == '?':
            temp_left[j]="?"
            temp_left[(j-1)%p]="?"
            temp_left[(j-5)%p]="?"
    input_left=input_right
    input_right=""
    for j in temp_left:
        input_right+=j
    print(input_left,input_right)




# In[ ]:
```

Code for `Simeck`'s Impossible differential in related key setting

```
#!/usr/bin/env python
# coding: utf-8

# In[12]:


# key schedule
input=["" for i in range(15)]
input[0] = "1000000000000000"
input[1] = "0000000000000000"
input[2] = "0000000000000000"
input[3] = "0000000000000000"
print(1,"\t",input[0])
print(2,"\t",input[1])
print(3,"\t",input[2])
print(4,"\t",input[3])
for i in range(4,15):
    input_left = input[i-3]
    input_right= input[i-4]
    temp_right=['0' for i in input_right]
    for j in range(16):
        if input_left[j] == '1' :
            temp_right[j]="?"
            if temp_right[(j-1)%16]!='?' :
                temp_right[(j-1)%16]="1"
            temp_right[(j-5)%16]="?"
        elif input_left[j] == '?':
            temp_right[j]="?"
            temp_right[(j-1)%16]="?"
            temp_right[(j-5)%16]="?"
    #print(temp_right)
    for j in range(16):
        if input_right[j]=='?':
            temp_right[j]='?'
        if temp_right[j]!= '?' and input_right[j]==temp_right[j]:
                temp_right[j]='0'
        elif temp_right[j]!= '?' and input_right[j]!=temp_right[j]:
                temp_right[j]='1'

    for j in temp_right:
        input[i]+=j
    print(i+1,"\t",input[i])


# In[13]:


# forward direction - encryption
```

```
input_left = "0000000000000000"
input_right= "1000000000000000"
rounds=10
print(0,input_left,input_right)
for i in range(rounds):
    temp_right=[i for i in input_right]
    for j in range(16):
        if input_left[j] == '1':
            temp_right[j]="?"
            if temp_right[(j-1)%16]!='?' :
                if temp_right[(j-1)%16] == "1":
                    temp_right[(j-1)%16]="0"
                else:
                    temp_right[(j-1)%16]="1"
            temp_right[(j-5)%16]="?"
        if input_left[j] == '?':
            temp_right[j]="?"
            temp_right[(j-1)%16]="?"
            temp_right[(j-5)%16]="?"
        if input[i][j]== '?':
            temp_right[j]="?"
        if input[i][j]== "1":
            if temp_right[j]!="?":
                if temp_right[j]=="1":
                    temp_right[j]="0"
                else:
                    temp_right[j]="1"
    input_right=input_left
    input_left=""
    for j in temp_right:
        input_left+=j

    print(i+1,input_left,input_right)



# In[14]:


# backward direction - encryption
input_left = "0000010000000000"
input_right= "0000000000000000"
rounds=5
for i in range(rounds):
    temp_left=[i for i in input_left]
    for j in range(16):
        if input_right[j] == '1':
            temp_left[j]="?"
            if temp_left[(j-1)%16]!='?' :
                if input_left[(j-1)%16] == "1":
                    temp_left[(j-1)%16]="0"
                else:
                    temp_left[(j-1)%16]="1"
            temp_left[(j-5)%16]="?"
        if input_right[j] == '?':
            temp_left[j]="?"
            temp_left[(j-1)%16]="?"
            temp_left[(j-5)%16]="?"
        if input[-i-1][j]== '?':
            temp_left[j]="?"
        if input[-i-1][j]== "1":
            if temp_left[j]!="?":
                if temp_left[j]=="1":
                    temp_left[j]="0"
                else:
                    temp_left[j]="1"
    input_left=input_right
    input_right=""
    for j in temp_left:
        input_right+=j
    print(input_left,input_right)



# In[ ]:
```

Code for DDT, LAT, BCT and BDT of TWINE sbox

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


S=[0xC,0,0xF,0xA,2,0xB,9,5,8,3,0xD,7,1,0xE,6,4]
Sinv=[S.index(x) for x in range(16)] #inverse s-box


# In[2]:


# Difference Distribution Table
```

```
DDT=[[0 for i in range(16)] for j in range(16)]
for i in range(16):
    for j in range(16):
            DDT[i^j][S[i]^S[j]]+=1
for i in DDT:
    print(i)

#1-1 DDT
for i in [8,4,2,1]:
    for j in [8,4,2,1]:
        print(DDT[i][j]," ",end="")
    print()


# In[3]:


#Linear Approximation Table

LAT=[[-8 for i in range(16)] for j in range(16)]
for i in range(16):
    for j in range(16):
        for x in range(16):
            if (bin(i&x).count("1") - bin(j&S[x]).count("1")) % 2 == 0:
                    LAT[i][j]+=1
for i in LAT:
    print(i)
#1-1 LAT
for i in [8,4,2,1]:
    for j in [8,4,2,1]:
        print(LAT[i][j]," ",end="")
    print()


# In[4]:


# Boomerang Connectivty Table
BCT=[[0 for i in range(16)] for j in range(16)]
for i in range(16):
    for j in range(16):
        for x in range(16):
            if i==Sinv[S[x]^j]^Sinv[S[x^i]^j]:
                BCT[i][j]+=1
for i in BCT:
    print(i)



# In[5]:


# Boomerang Difference Table
BDT=[[[0 for k in range(16)] for i in range(16)] for j in range(16)]
for i in range(16):
    for j in range(16):
        for k in range(16):
            for x in range(16):
                if i==Sinv[S[x]^k]^Sinv[S[x^i]^k] and S[x]^S[x^i]==j:
                    BDT[i][j][k]+=1
print("The table for (i,j,0) which is similar to DDT")
for i in range(16):
    for j in range(16):
        print( BDT[i][j][0]," ",end="")
    print()

print("complete BDT")
ctr=0
for i in BDT:
    for j in i:
        print(j)
    print("---------------")


# In[ ]:
```

Code SimeckMILP

```
#include <fstream>
#include <stdlib.h>
#include <iostream>
#include "gurobi_c++.h"

#define BLOCK_SIZE (16)
#define ROUNDS (15)
using namespace std;


static void XORR (GRBEnv env, GRBModel& model,int num_rounds, GRBVar A[][BLOCK_SIZE], int a, GRBVar B[][BLOCK_SIZE],GRBVar C[][BLOCK_SIZE], GRBVar D[][BLOCK_SIZE])
{

                for (int j=0;  j<BLOCK_SIZE; j++)
                {
                model.addConstr(A[a-1][j] + B[num_rounds-1][j] + C[num_rounds-1][j]>=2*D[num_rounds-1][j]);
                model.addConstr(D[num_rounds-1][j]>=A[a-1][j]);
                model.addConstr(D[num_rounds-1][j]>=B[num_rounds-1][j];
```

```
                model.addConstr(D[num_rounds-1][j]>=C[num_rounds-1][j]);
                model.addConstr(A[a-1][j] + B[num_rounds-1][j] + C[num_rounds-1][j]<=2);
                }

}

static void HREPRESENTATION (GRBEnv env, GRBModel& model,int num_rounds, GRBVar T1[][BLOCK_SIZE], GRBVar T3[][BLOCK_SIZE],
                                                        GRBVar L[][BLOCK_SIZE],int r)
{

                for (int j=0;  j<BLOCK_SIZE; j++)
                {

                        model.addConstr(L[r-1][(j+0)%BLOCK_SIZE]+ L[r-1][(j+5)%BLOCK_SIZE] - T1[num_rounds-1][j]>=0);
                        model.addConstr(L[r-1][(j+1)%BLOCK_SIZE]==T3[num_rounds-1][j]);

                }

}

static void
                inOputmask (GRBEnv env, GRBModel& model, GRBVar L[][BLOCK_SIZE],GRBVar R[][BLOCK_SIZE],GRBVar L1[][BLOCK_SIZE],
                GRBConstr m1[BLOCK_SIZE],GRBConstr m2[BLOCK_SIZE],GRBConstr c1[BLOCK_SIZE],GRBConstr c2[BLOCK_SIZE])
{
                        for (int i=0;i<BLOCK_SIZE;i++)
                {

                        m1[i] = model.addConstr(L[0][i]==0);

                        m2[i] = model.addConstr(R[0][i]==0);

                        c1[i] = model.addConstr(L1[ROUNDS-1][i]==0);

                    c2[i] =  model.addConstr(L1[ROUNDS-2][i]==0);
                }

}


int main(int argc, char *argv[])
{
  GRBVar *vars = 0;

                GRBEnv env = GRBEnv();
                GRBModel model = GRBModel(env);

                // Set variables
                GRBVar L[ROUNDS][BLOCK_SIZE] ;
                GRBVar R[ROUNDS][BLOCK_SIZE] ;
                GRBVar T1[ROUNDS][BLOCK_SIZE] ;
                GRBVar T2[ROUNDS][BLOCK_SIZE] ;
                GRBVar T3[ROUNDS][BLOCK_SIZE] ;
                GRBVar R1[ROUNDS][BLOCK_SIZE] ;
                GRBVar DT1[ROUNDS][BLOCK_SIZE] ;
                GRBVar DT3[ROUNDS][BLOCK_SIZE] ;
                GRBConstr m1[BLOCK_SIZE];
                GRBConstr m2[BLOCK_SIZE];
                GRBConstr c1[BLOCK_SIZE];
                GRBConstr c2[BLOCK_SIZE];

                for(int i=0;i<ROUNDS; i++)
                        for(int j=0;j<BLOCK_SIZE; j++)
                        {
                                L[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                        R[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                                T1[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                                T2[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                                T3[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                                R1[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                                DT1[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                                DT3[i][j]= model.addVar(0.0,1.0,0.0,GRB_BINARY);
                        }

                model.update();


                // ROUND 1
                XORR(env,model,1,R,1,T1,T2,DT1);
                XORR(env,model,1,T2,1,T3,R1,DT3);
                HREPRESENTATION(env,model,1,T1,T3,L,1);

                // ROUND 2
                if(ROUNDS!=1)
                {
                XORR(env,model,2,L,1,T1,T2,DT1);
                XORR(env,model,2,T2,2,T3,R1,DT3);
                HREPRESENTATION(env,model,2,T1,T3,R1,1);
                }
                //OTHER ROUNDS
                for(int i=3;i<=ROUNDS;i++)
                {
                XORR(env,model,i,R1,i-2,T1,T2,DT1);
                XORR(env,model,i,T2,i,T3,R1,DT3);
                HREPRESENTATION(env,model,i,T1,T3,R1,i-1);
                }

                inOputmask(env,model,L,R,R1,m1,m2,c1,c2);

                //objective function
                // GRBQuadExpr OBJfun1,OBJfun2,OBJfun;
                GRBLinExpr OBJfun1,OBJfun2,OBJfun;
                for(int j=0;j<BLOCK_SIZE;j++)
                        OBJfun1 += L[0][j];
```

```
                for(int i=2;i<=ROUNDS;i++)
                        for(int j=0;j<BLOCK_SIZE;j++)
                        OBJfun2+=R1[i-2][j];
                OBJfun=OBJfun1+OBJfun2;
                model.addConstr(OBJfun>=1);


                model.setObjective(OBJfun,GRB_MINIMIZE);
                model.write("model.lp");

int numvars = model.get(GRB_IntAttr_NumVars);
vars = model.getVars();


                model.optimize();

std::cout << "----------------------\n ";


        for(int i=0;i<BLOCK_SIZE;i++){
                                m2[i].set(GRB_DoubleAttr_RHS, 1.0);
                                for(int j=0;j<BLOCK_SIZE;j++){
                                        c1[j].set(GRB_DoubleAttr_RHS, 1.0);

                                model.update();

                model.optimize();
                        if (model.get(GRB_IntAttr_Status) == 3){
                        std::cout<< "model is infeasible"<<"\n";

                        cout <<"m2 "<< i <<"\t"<<"c1 "<< j <<"\n";
std::cout << "----------------------\n ";
        //                        }
                                c1[j].set(GRB_DoubleAttr_RHS, 0.0);
                                model.update();}

                                m2[i].set(GRB_DoubleAttr_RHS, 0.0);
                                model.update();
                        }


}

        return 0;
}
```