Fault Induced Zero-Sum Exploiting Division Trails: Applications on PRESENT and GIFT

Anonymous Submission

Contents

1 Introduction

Table 1: summary of fault attacks on PRESENT and GIFT

cipher	fault model	no. of faults	strategy	ref

1.1 Organization of the paper

2 Preliminaries and Background

2.1 Notations

For any $a \in \mathbb{F}_2^n$, the Hamming weight of is $wt(a) = \sum_{i=1}^{i=n} a_i$. For any vector $\boldsymbol{a} = (a_0, a_1, \cdots, a_{m-1}) \in \mathbb{F}_2^{l_0} \times \mathbb{F}_2^{l_1} \times \cdots \times \mathbb{F}_2^{l_{m-1}}$, the vectorial Hamming weight of $\mathbf{a}$ is $W(\boldsymbol{a}) = (wt(a_0), wt(a_1), \cdots, wt(a_{m-1})) \in \mathbb{Z}^m$. For any $\boldsymbol{k} \in \mathbb{Z}^m$ and $\boldsymbol{k'} \in \mathbb{Z}^m$, we define $\boldsymbol{k} \succeq \boldsymbol{k'}$ if $k_i \geq k'_i$ for all i. For any integer $k \in \{0, 1, .., n\}$ we define the set $\mathbb{S}_k^n = \{a \in \mathbb{F}_2^n : k \leq wt(a)\}$ and for any vector $\boldsymbol{k} \in (\{0, 1, .., n\})^m$ we define the set $\mathbb{S}_{\boldsymbol{k}}^{m,n} = \{\boldsymbol{a} = (a_1, a_2, ..., a_m) \in (\mathbb{F}_2^n)^m : \boldsymbol{k} \preceq W(\boldsymbol{a})\}$. For any vector $u \in \mathbb{F}_2^n$ and $x \in \mathbb{F}_2^n$, we define the *bit product* $\pi_u : \mathbb{F}_2^n \to \mathbb{F}_2$ as $\pi_u(x) = \prod_{i=1}^n {x_i}^{u_i}$ and for any vector $\boldsymbol{u} \in (\mathbb{F}_2^n)^m$ and $\boldsymbol{x} \in (\mathbb{F}_2^n)^m$, we define the *vectorial bit product* $\pi_{\boldsymbol{u}} : (\mathbb{F}_2^n)^m \to \mathbb{F}_2^n$ as $\pi_{\boldsymbol{u}}(\boldsymbol{x}) = \prod_{i=1}^m \pi_{u_i}(x_i) = \prod_{i=1}^m \left(\prod_{j=1}^n x_{i_j}^{u_{i_j}} \right)$. The Algebraic Normal Form (ANF) of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be defined as $f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u^f \pi_u(x)$ and the degree of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is d if d is the degree of the largest monomial in the ANF of f, i.e., $d = \max_{u \in \mathbb{F}_2^n, a_u^f \neq 0} wt(u)$.

We use bold lowercase letters to represent vectors in a binary field. For any n-bit vector $\mathbf{s} \in \mathbb{F}_2^n$, its i-th coordinate is denoted by s_i, thus we have $\mathbf{s} = (s_{n-1}, ..., s_0)$.

2.2 Fault Attack

A Fault Attack is an attack to break the cryptosystem by exploiting its hardware design. A successful fault attack consists of two things Fault Injection and Fault Propagation. Fault injection depends upon the hardware and the fault model. Fault propagation depends upon the property and the design structure of the cipher.

2.2.1 Differential Fault Attack

Differential Fault Attack is one of the basic fault attack techniques. Here attacker takes a message and computes its corresponding ciphertexts through the oracle. Then he takes the same message and injects a particular difference with the message at a fixed round. From the original and the faulty ciphertext, he tries to recover the key to break the cryptosystem.

2.2.2 Integral Fault Attack

In an integral fault attack, the attacker takes a message and computes its ciphertexts. He again takes the same message and modifies some particular nibble or byte at a particular round so that the nibble or byte takes all values at that particular position i.e. All property. Then he uses the propagation of the injected All property and from the ciphertexts by decrypting one or more rounds to recover the key of the cipher.

2.3 Division Property

Definition 1. *(Division Property. [Tod17])* A multi-set $X \subseteq \mathbb{F}_2^n$ is said to have the division property of order k, $\mathcal{D}_k^n$ for some $1 \leq k \leq n$, if the sum over all vectors $x \in X$ of the product $x^u = 0$, for all vectors u with hamming weight less than k, i.e.,

$$\bigoplus_{x \in X} \pi_u(x) = 0, \forall u \in \mathbb{F}_2^n \text{ with } wt(u) < k.$$

Definition 2. *(Vectorial Division Property. [Tod17])* A multi-set $X \subseteq \mathbb{F}_2^{l_0} \times \mathbb{F}_2^{l_1} \times \cdots \times \mathbb{F}_2^{l_{m-1}}$ is said to have the division property $\mathcal{D}_{\mathbb{K}^{l_0,l_1,\cdots,l_{m-1}}}$ for some set of m-dimensional vectors $\mathbb{K}$ whose i-th element takes a value between 0 to l_i , it fulfills the following conditions:

$$\bigoplus_{\boldsymbol{x} \in X} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{cases} \text{unknown}, & \text{if there is } \boldsymbol{k} \in \mathbb{K} \text{ s.t. } W(\boldsymbol{u}) \succeq \boldsymbol{k} \\ 0 & \text{otherwise} \end{cases}$$

Moreover, if each li is restricted to 1, we will say bit-based division property and we will denote it by $\mathcal{D}_{\mathbb{K}}^{1,n}$.

2.3.1 Division Trail

Suppose X with the division property $\mathcal{D}_{\boldsymbol{k}}^{n,m}$ is an input multiset of a iterated block cipher and let the round function is f_r. The r-round division property propagation

$$\{\boldsymbol{k}\} \stackrel{def}{=} \mathbb{K}_0 \xrightarrow{f_r} \mathbb{K}_1 \xrightarrow{f_r} \mathbb{K}_2 \xrightarrow{f_r} \cdots \xrightarrow{f_r} \mathbb{K}_r$$

. This shows the propagation through all r-round division trails from $\{\boldsymbol{k}\}$ to $\mathbb{K}_r$.

An Substitution Permutation Network structure consists of a substitution layer named SBox and a permutation layer. These two together make the round function of the cipher. Here a model can be made for each round function. The input variables, $a = (a_0, a_1, \cdots, a_n)$ denotes the input property and $b = (b_0, b_1, \cdots, b_n)$ denotes the output property of the cipher. Depending upon the choice of a and b, the property will differ through the rounds.

Input Division Property The attack that we show in this paper is a random nibble-based fault attak. As shown in [XZBL16] we make the MILP model of Division Trails of the cipher's SBox and see how the property propagates through the cipher the bits which we want to set as active at the initial round, we set the value of the bits 1 and keep other bits value to 0. Therefore $\mathbb{K}_0$ contains the vector whose only ith nibble position has value 1, where i can be any random nibble.

Output Division Property After giving the initial input division property to a MILP solver we have to specify where it will stop the search. Here we see the propagation of Division Property through the cipher by activating some particular bits i.e. in other words, giving All property in the bits. We use the propagation **??** as output Division Property to stop the search. So from Proposition 1, if $\mathbb{K}_r$ contains all the unit vectors then we terminate the search.

Proposition 1. *([XZBL16]) Let $\mathbb{X}$ be a multi-set with bit-based division property $\mathcal{D}_{\mathbb{K}}^{n}$, then $\mathbb{X}$ does not have integral property iff $\mathbb{K}$ contains all vectors of weight 1.*

If the output multiset $\mathbb{K}$ contains all the vectors with hamming weight 1, we stop the search.

2.4 Propagation of Division Property

In this paper we work on two lightweight ciphers `PRESENT` and `GIFT`. Both the ciphers design are based on SPN structure. In this section we see how the Division Property propagates through the cipher. Suppose we give a set as input to the cipher whose initial Division Property $\mathbb{D}_{K_0}^{1,n}$ and after the propagation of the Division Property, we get $\mathbb{Y}$ as output whose Divion Property is $\mathbb{D}_{K_0}^{1,n}$. Then the propagation $\mathbb{D}_{K_0}^{1,n}$ to $\mathbb{D}_{K_0}^{1,n}$ is called a valid Division Trail through the cipher.

Definition 3. Let fr denote the round function of an r round iterative primitive. Suppose the initial division property is $\mathcal{D}^{1,n}_{\{k_0\}}$ and after $(i-1)$-round propagation, the division property is $\mathcal{D}^{1,n}_{\mathbb{K}_i}$. Then we have the following chain of division

$$\{k_0\} := \mathbb{K}_0 \xrightarrow{f_0} \mathbb{K}_1 \rightarrow \xrightarrow{f_1} \mathbb{K}_1 \xrightarrow{f_2} \cdots$$

Moreover, for any vector $k_i \in \mathbb{K}_i (i \geq 1)$, there must exist a vector $k_{i-1} \in \mathbb{K}_{i-1}$ such that k_{i-1} can propagate to k_i by division property propagation rules. For $(k_0, k_1, \cdots, k_{r-1})$, if k_{i-1} can propagate to k_i for all $i \in \{1, 2, \cdots, r\}$, we call $(k_0, k_1, \cdots, k_{r-1})$ an r-round division trail.

In this work we are giving an initial Division Property $(a_0, a_1, \cdots, a_n)$ as input and after propagation of the Division Trail we get $(b_0, b_1, \cdots, b_n)$ as output. The bits where we want to give All property, we set the values 1 and keep other values to 0. After the propagation, we find the places which has stil the value 0. We get the balanced bits in those position.

2.4.1 Modeling SBox

As said earlier, we do our attack on PRESENT and GIFT. Both of them are SPN structures where the permutation layer is only bit permutation. So, after getting the output variables we just have to permute and give those as input in the next round. Hence the cipher totally depends on modelling the sbox. To model the sbox of PRESENT and GIFT we use the algo presented in the paper [XZBL16]. This take an input property vector as input variables and gives a set of vectors as output Division Property. The output set contains those vectors b_i such that from a to each b_i there is a possible Division Trail.

Algorithm 1 SBox Division Trail ($k = (k_0, k_1, \cdots, k_{n-1})$)

$\mathbb{S}_k = \{a | a \succeq k\}$
$\mathrm{F(X)} = \{\pi_a(x) | a \in \mathbb{S}_k\}$
$\mathbb{K} = \phi$
for $u \in \mathbb{F}_2^n$ **do**
  if $\pi_u(x) \cap F(X) \neq \phi$ **then**
    $Flag = True$
    $R = \phi$
    for $v \in \mathbb{K}$ **do**
      if $v \succeq u$ **then**
        $Flag = False$
      else if $u \succeq v$ **then**
        $R = R \cup \{v\}$
    if $Flag = True$ **then**
      $\mathbb{K} = \mathbb{K} \setminus R$
      $\mathbb{K} = \mathbb{K} \cup \{u\}$
return $\mathbb{K}$

2.5 PRESENT [BKL+07]

PRESENT is a SPN structure with 31 rounds. Each round consists of state size 64 bits & key size 80 and 128, depending upon the version 64 or 128 respectively. To reduce the hardware area, each round has SBox, bit permutation and key whitening.

addRoundKey. This function takes the state $b_{63} \cdots b_0$ and the corresponding round key $K_i = k^i_{63} \cdots k^i_0$ for $1 \leq i \leq 32$ and returns $b_j \oplus k^i_j$ for $0 \leq j \leq 63$.

***S*-box.** The S-box of PRESENT is a function $S : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ which is given in the table 2. It takes the state bits and divide it into 4-bit words. The S-box is applied in each of the words. The updated state value is then return as the output.

Table 2: PRESENT S-box

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S(x)	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

***p*-Layer.** PRESENT uses bit permutation in its permutation layer. It takes the i-th bit and moves to $p(i)$ in the next round. The bit permutation of PRESENT can be seen from [**?**]

The key schedule. Depending upon the version of PRESENT its key schedule differs. PRESENT-64 takes $k_{79}k_{78}\cdots k_{16}$ and PRESENT-128 takes $k_{127}k_{126}\cdots k_{64}$ respectively as its round key and updates the key register by bit rotation and round counter at each round .

2.6 GIFT [BPP+17]

GIFT-64 is a SPN structure with state size 64-bit (128-bit for GIFT-128) consists of 28 rounds (40 rounds for GIFT-128) and key size 128 bit.

Round function. Both the round functions of GIFT-64 and GIFT-128 have SubCells, Bit Permutation and Addition of round keys.

Initialization. GIFT-64 takes 64 bit (128 bit for GIFT-128) plaintext like $b_{64}b_{63}\cdots b_0$ and a master key of size 128-bit as $K = k_7||k_6||\cdots||k_0$, where k_i is a 16-bit word.

SubCells. The S-box of GIFT is a function $S : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ which is given in the table 3. It takes the state bits and divide it into 4-bit words. The S-box is applied in each of the words. The updated state value is then return as the output.

Table 3: GIFT SBox

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S(x)	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

PermBits. Like PRESENT, GIFT also uses bit permutation in its permutation layer. It takes the i-th bit and moves to $p(i)$ for the next round. The bit permutation for GIFT-64 and GIFT-128 can be seen from [BPP+17].

AddRoundKey. GIFT-64 takes a round key of size 32 (64 for GIFT-128) at each round. It is divided into two 16-bit words (32 for GIFT-128) $RK = U||V = u_{15}\cdots u_0||v_{15}\cdots v_0$ and XOR-ed with the last two bits of each nibble.

Key schedule and round constants. Both the key schedule and round constants are same for both GIFT versions. each round constant is a 6-bit number which is XOR-ed in each round at some particular position. The values of the constants for each round can be seen from [BPP+17].

2.7 Modeling Division Property

3 Fault Model

The Fault Model shows the type and nature of a fault. Depending upon the impact of the fault, a fault model can be of bit level or byte model or, word level. Also, it can be categorized by seeing whether its distribution is unform or random. The attack we present here falls under the random nibble level fault model. After choosing the nibble randomly we give fault to generate some ciphertexts and try to recover the key from there.

3.1 Fault Invarient from Division Property

Fault attacks are most useful in practical scenario. If an attacker gets access of the device, he can manipulate with the device and get the hidden information from there. Within the fault models, the most powerful is the random distribution one. In that case the attacker can fault on any random position and of that only he can exploit the cipher. In this paper, our attack is based upon random nibble fault model. Hence the distribution we are taking at the time of giving fault is random and the attack is on nibble level. Initially we choose a nibble randomly from the cipher and give faults to generate an input Division Property at that position. The property then propagates through the cipher and generates Division Trials. As described earlier if we get the output Division Trail variables as 0. We get balanced bits in those position. In this work, we show for these two SPN structures `PRESENT` and `GIFT` we always get the balanced bits at the same position irrespective of giving the initial input Division Property at any random nibble. This seems the propagation of the All property that we generally do in Integral Cryptanalysis but the main difference is that here we can give All at any random nibble and in every case we get balanced bits at same position. This is a fault invariant model for the SPN structures `PRESENT` and `GIFT` as after choosing the nibble randomly also the balanced bit remains at constant position.

3.2 Results using Division Property on `PRESENT` and `GIFT`

As described earlier, we use Division Property to find balanced position after some rounds in the cipher. For `PRESENT` we get a 5-round property of the cipher. We give a initial input Division Property at a nibble and depending upon the propagation of the Division Trail after 5 rounds we get balanced positionsin some particular bits. After the permutation layer of the 5th round, the balanced bits occurs at the first 4 nibbles. Hence we are getting total 16 bit balanced positions which remains invariant if we give input Division Property at random nibble. Like `PRESENT`, for `GIFT`-64 also we get the random nibble property for 5 rounds. Here the balanced bits are at different position than `PRESENT`. For `GIFT`-64 we get 16 bits which appears in the 0th bit of each nibble after the permutation layer of 5th round. Like the previous two ciphers, for `GIFT`-128 we get some balanced bit positions after 5 rounds. Interestingly for `GIFT`-128 we get two sets of balanced bits, eachconsists of 80 bit postion. The first set occurs if we give the input Division Property at nibbles 0 to 15. The other set occurs if we give the initial input Division Property at the nibble postions 16 to 31. The two sets have an intersection of .... bit positions that we get in **??**.

Table 4: balanced bit position after last p-layer of PRESENT-64

round	# active bits	#balanced bits	balanced bits position
5	4	4	0,1,2,3
5	3	0	
4	4	64	0(1)64
4	3*	16	0(1)16
3	2*	16	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60
3	2*	16	2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62
3	1	0	

Table 5: balanced bit position after last p-layer of GIFT–64

round	# active bits	#balanced bits	balanced bits position
5	4	16	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60∗
5	3	0	
4	4	64	0(1)64
4	3*	24	∗
4	2	0	
3	2*		∗
3	1	0	0

4 Balancedness in PRESENT cipher

In this section, we look into a new property of the cipher PRESENT. This is a 5 rounds random nibble property of the block cipher. Here we give All property at any random nibble in the input. As a result, we get balanced bits at fixed positions after 5 rounds. We illustrate the active bit propagation and the monomial view of the propagated bits by Figure 1. Suppose we give the All property at the input of the 0th nibble. Therefore whenever in the ANF (Algebraic Normal Form) representation of a bit, the product term $x_3x_2x_1x_0$ comes, that bit will be balanced. After the SBox of the 1st round, the anf representation of the bits becomes $y_3y_2y_1y_0$, where

$$
\begin{aligned}
y_0 &= x_0 \oplus x_2 \oplus x_3 \oplus x_1x_2 \\
y_1 &= x_1 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \\
y_2 &= 1 \oplus x_2 \oplus x_3 \oplus x_0x_1 \oplus x_0x_3 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \\
y_3 &= 1 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3
\end{aligned}
$$

After the permutation, the active bits go to the 0th bit of the nibbles 0,4,8,12 respectively. The other bits of that round are inactive. Hence at the output of the 2nd round, only the output bits of the nibble number 0,4,8,12 are active. At 3rd round, the ANF representation of the bits of 0th bit of each SBox is $c_i + c_jy_k$, where $c_i, c_j's$ are constant and k varies in range(3).

Hence at the output of the 2nd round, only 16 bits are active. Up to this point, the active bit's ANF representation does not contain the product term $x_3x_2x_1x_0$, so the constant bits as well as the active bits are balanced. Interestingly the 16 active bits go into the 0th input bit of 16 SBox s in the 3rd round. At the output of the 3rd round, all 64 bits become active. In the input of the 4th round, each SBox contains one bit dependent

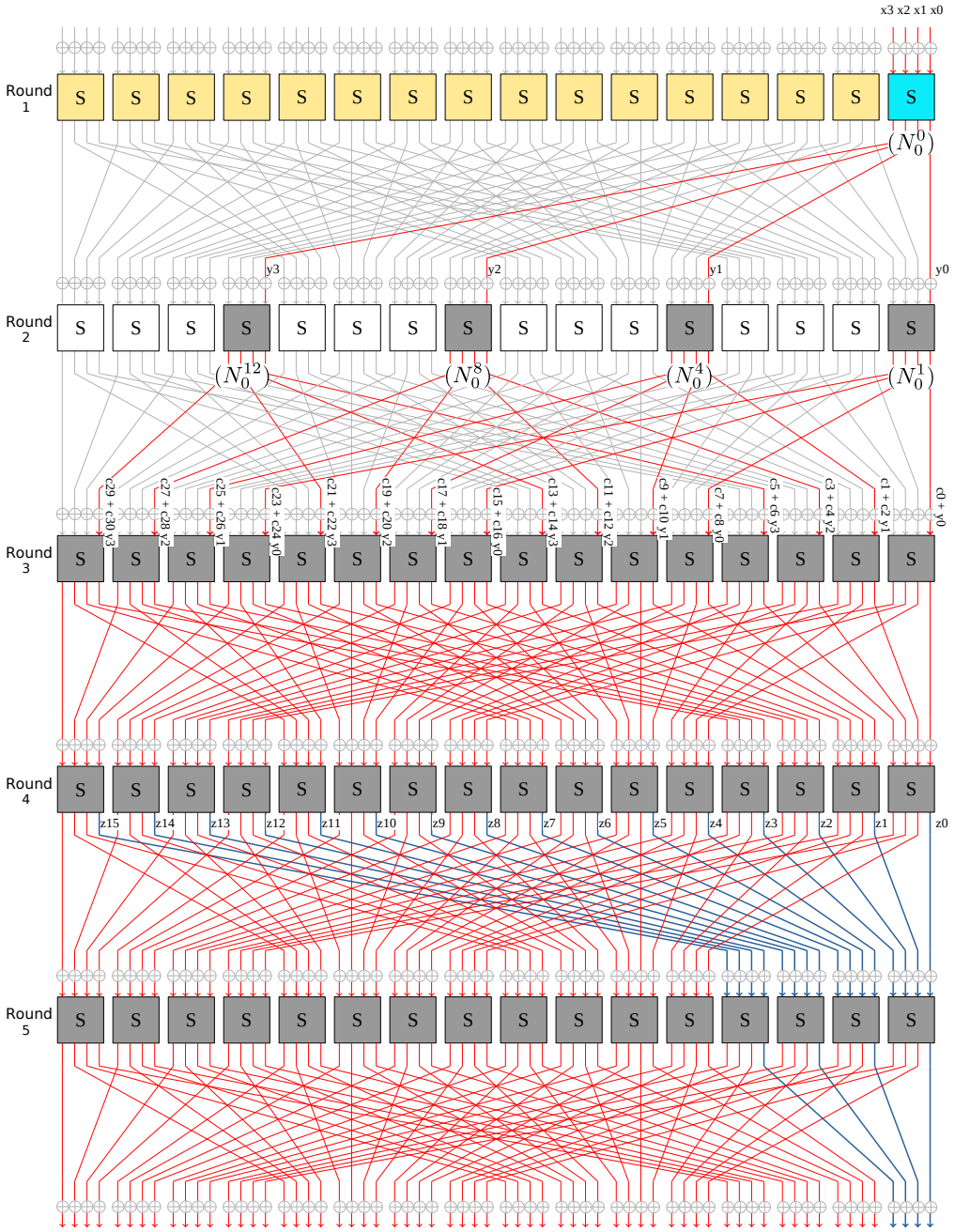

Figure 1: active bit propagation on PRESENT

upon y_3, y_2, y_1, y_0 respectively. Therefore at the output of the 4th round, there will be $z_i's$; where each z_i is a combination of $y_3y_2y_1y_0$. Though at this point, the ANF of each $z_i's$ contains $y_3y_2y_1y_0$ i.e. a combination of $x_3x_2x_1x_0$ but the product term $x_3x_2x_1x_0$ is not

there. At the output of round 4, we see only the 0th bit of each SBox. This is because only the ANF representation of the 0th bit contains a maximum 2-degree term. Whereas the other bits are of the 3-degree term. Therefore for other bits, their chance of producing the product term $x_3x_2x_1x_0$ is higher. The 0th bit of each SBox after the 4th round goes to the input of the initial 4 SBox s in the input of the 5th round. Again we see in the next round (i.e. 5th round), that for the 0th bit of SBox number 0,1,2,3 the chance of containing the term $x_3x_2x_1x_0$ is less than the other bits as in the previous round also they are coming from the 0th bits and in this round also they are the 0th bit of SBox 0,1,2,3. We implemented this in Sagemath and verify that the anf representation of 0,1,2,3 bits only (after 5 rounds permutation) does not contain the product term $x_3x_2x_1x_0$. The other bit's ANF representation contains it. Therefore after 5 rounds, the bits 0,1,2,3 becomes balanced.

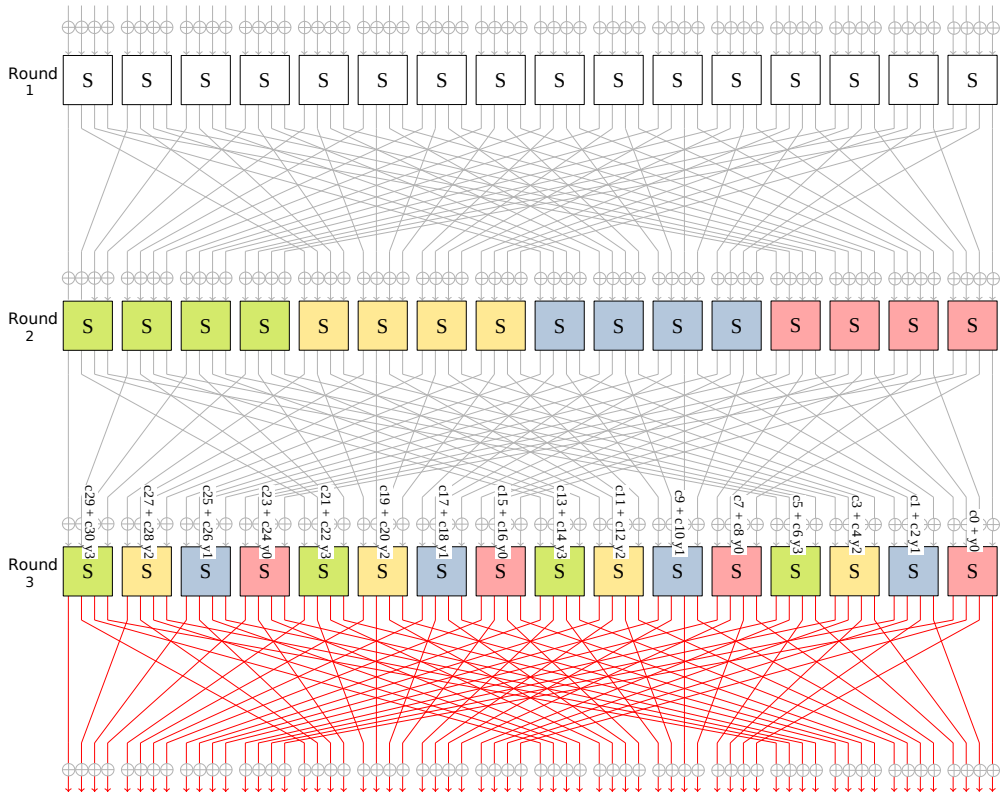

Figure 2: random nibble active bit propagation on `PRESENT`

We have shown the 4 bits become balanced after 5 rounds if we give the All property at the input of the 0th nibble. Now we show if we give the All property at the input of any SBox then also after 5 rounds the same 4 bits remain balanced. For this claim, we show that if we give All at any SBox then also at the output of the 3rd round the ANF representation of the bits are the same, just the constants vary. We illustrate this from Figure 2 . In Figure 2, we can see that irrespective of the nibble position of All property, due to present permutation 0th bit always goes to one of the red nibbles in round 1. Similarly, 1st bit, 2nd bit, and 3rd bit of the All property go to one of the blue, yellow, and green ones in the 2nd round. Hence because of the present permutation layer one of each colored SBox in Figure 2 becomes active in the 2nd round, where red, blue, yellow

and green one contains constant combination of y_0, y_1, y_2, y_3 respectively. From each red SBox, one bit goes to red ones in the 3rd round. The same thing happens for other colors also in round 2. Therefore in the input of 3rd round constant combination of y_0, y_1, y_2, y_3 goes to nibble $4i, 4i+1, 4i+2, 4i+3$ for i in range(3) respectively. Hence in the output of the 3rd round, the anf representation of bits will be similar (without constants) irrespective of where the All property has been given. So, after 5 rounds we always get the balanced bits at the same position despite of varying the All property position.

5 PRESENT key recovery attack

In this section, we describe our last round key recovery attack on PRESENT 64 using 5 round random nibble fault model.

5.1 1 round key recovery attack

5.1.1 Distinguisher model

To recover the last round key, we will use our 5 round random nibble fault model, which is described in Section 4. Our model is random nibble fault i.e. if we give All property at any random nibble then also the same 4 bits of 0th nibble will be balanced after 5 rounds. For this attack, we give 16 faults on a random nibble at input of 25th round. As our attack will work on any random nibble, so for the convention we assume we are giving fault to generate 16 faulty ciphertexts at 0th nibble we describe our key recovery attack in the next section.

5.1.2 Key Recovery attack

As described in the previous section, we are giving 16 faults to generate the All property at the 0th nibble in the input of the 25th round. We will get 4 balanced bits at the end of the 30th round. Using 16 faulty ciphertexts we recover 4 key bits of the last round. For this we choose those 4 bits which are coming from 0th nibble at 31-st round (N_0^{31} in Figure 3). We choose all possible 16 values of these 4 key bits. For each possible key value, we decrypt for one round of the ciphertexts. Then check for which key values, after inverting 1 round of 15 faulty ciphertexts and 1 ciphertext, the input 4 bits of N_0^{31} is 0. We will take those keys as possible keys of the last round selected 4 bits of fig. 3.

5.1.3 Complexity

In this 1 round attack of PRESENT, we recover 4 bits of the last round uniquely. Here for each key nibble, we have to decrypt 16 faulty ciphertexts. Therefore time complexity of this attack $= 2^8$. For this, we have to store 16 faulty ciphertexts. So the data complexity of this attack will be 2^4. In algorithm 2 we are storing the possible nibble values of the key to a key list. But instead of storing those nibble keys, we can print those values. Then we do not have to store any value. So, the memory complexity of the attack is negligible.

5.2 2 rounds key recovery attack

In this section, we discuss the 2 round attack on PRESENT 64. We use our 5-round random nibble fault model distinguisher to recover some bit information of the last and 2nd last round key.

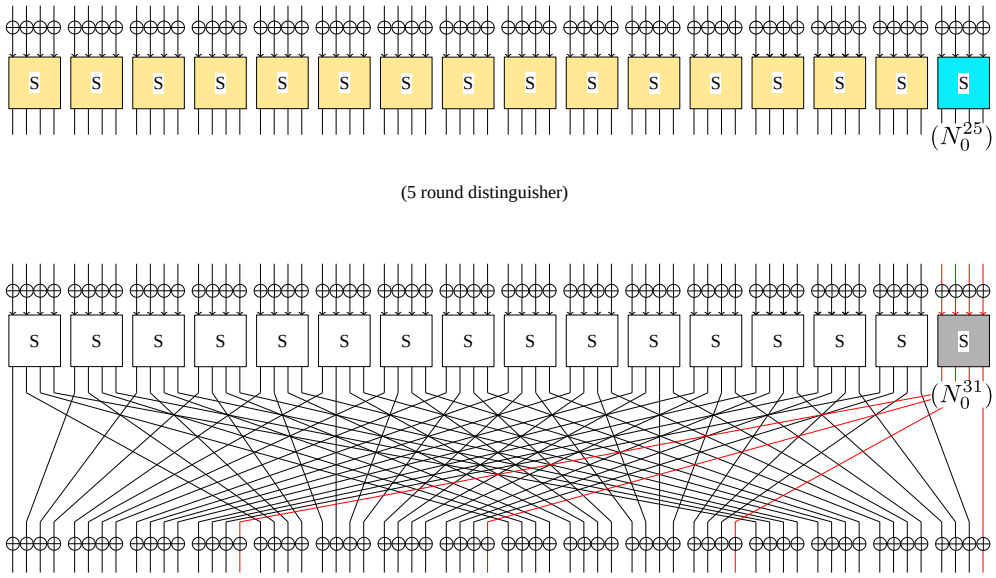

Figure 3: 1 round key recovery attack on PRESENT

Algorithm 2 KEY RECOVERY $(C'_1, \cdots, C'_{16})$

```
Initialize an array called key_list
for each key_nibble in range{0 - 16} do          ▷ output of SB at N_0^6 in fig. ??
    for each j in range {0 - 16} do
        dec_msg[j] ← SB^{-1}(key_nibble⊕ perm^{-1}(C'_j))
    xor_sum ← dec_msg[0] ⊕ ··· ⊕ dec_msg[15]
    if first 4 bits of xor_sum is 0 then          ▷ input of SB at N_0^6 in fig. ??
        store {key_nibble} in key_list
return key_list
```

5.2.1 Distinguisher model

We use the same distinguisher model as discussed in the section 4. Here we give 15 faults to generate 15 faulty ciphertexts and alongside generate the original ciphertext of the message. So we have in total 16 ciphertexts. In this case, we give All property in a nibble at the input of the 24th round as shown in Figure 4. As our attack works on any random nibble so here also for the convention we give the All at 0th nibble of input of 24th round i.e. input of N_0^{24}. We discuss our key recovery attack in the next part of this section.

5.2.2 Key Recovery attack

Here we give fault to generate 16 ciphertexts at input N_0^{24} in Figure 4. By our attack model, we get the 4 initial bits as balanced after 5 rounds. So we get the balanced bits at input of 0th nibble in the 30th round i.e. N_0^{30} in Figure 4. We guess the 16 key bits of the last round and 4 key bits of the 2nd last round. We maintain a counter, ctr to keep track of the possible key values. Initially, we put the ctr value to 1 i.e. we consider all the 20-bit key values [16 for the last round and 4 for the 2nd last] as possible keys. If the possible keys do not satisfy our condition then we put the ctr value of the corresponding key as 0. Now for the last 4 nibble of keys i.e. output of the SBox of $N_{31}^{12}, N_{31}^{8}, N_{31}^{4}, N_{31}^{0}$ in Figure 4 we guess all possible 2^{16} values of them and decrypt one round. Then we use

4 key bits of 30th round's 0th nibble i.e. output of N_0^{30} in Figure 4 and decrypt the 16 ciphertexts for one more round. We check whether at the input of N_0^{30} in Figure 4 the xor value of 16 decrypt ciphertexts becomes 0 or not. If it does not become 0 then we update the corresponding ctr value to 0. At last, we take those values of keys as possible keys for which the ctr value remains 1.

Algorithm 3 KEY RECOVERY $(C'_1, \cdots, C'_{16})$

```
Initialize an array called key_ctr of size 2^20
extract 16 bit last round keys and 4 bit 2nd last round keys
for each key_nibble in last_round_key do ▷ output of SB at N_0^31, N_4^31, N_8^31, N_12^31 in Figure 4
    for each j in range {0 - 16} do
        dec_msg[j] ← SB^-1(key_nibble_last_round⊕ perm^-1(C'_j))
    for each key_nibble in 2nd_last_round_key do          ▷ 2nd round decryption
        for each j in range {0 - 16} do
            dec_msg[j] ← -1(key_nibble_2nd_last_round⊕ perm^-1(dec_msg[j]))
    xor_sum ← dec_msg[0] ⊕ ··· ⊕ dec_msg[15]
    if first 4 bits of xor_sum is 0 then          ▷ input of SB at N_0^30 in Figure 4
        store {key_nibble} in key_list
return key_list
```

5.2.3 Complexity

In our two rounds key recovery attack for PRESENT, we use all possible keys of the nibble 0, 4, 8 and 12 to decrypt one round in the last layer. This can be done in parallel. As we have 16 messages, we need $2^{4+4} = 2^8$ computations to invert the last layer. In the 2nd layer, we have to do again 2^4 many messages which are 2^8. The xor sum and checking of 1ast 4 bits takes a constant amount of time. Therefore total time complexity = $\mathcal{O}(2^8)$. This reduces the key space of the last to ... . This attack needs a total of 16 ciphertexts. Hence data complexity = 2^4. The algorithm returns the possible key values that satisfy the xor-sum condition. So here we are not storing any value of the keys. Therefore the needed memory is negligible.

6 Balancedness in GIFT cipher

In this section we discuss about a to 5 round property of PRESENT which is discussed in the section Section 4. Both PRESENT and GIFT uses SPN structure where the premutation layer is only bit permutation. Due to different bit permutation and SBox of PRESENT and GIFT, for GIFT 64 we get in total 16 balanced bits after 5 roundds. This is also a random nibble property i.e. it is irrespective of in which nibble we give All property at the input. We always get balanced bits at those particular 16 bits.

Initially we give the All property at the input of 0th nibble (as shown in fig Figure 6). From the monomial representation of Division property we can say that, those bits will be unknown whose ANF representation contains $x_3x_2x_1x_0$. After 1st round the representation of bits of N_0^0 will be ..... After permutation layer, those 4 bits from N_0^0 goes to nibble 0, 4, 8, 12 SBox contains constant combination of $y_3y_2y_1y_0$ in the ANF representaiton of their bits. So, the input of 3rd round contains combination of $y_3y_2y_1y_0$ as shown in Figure 6. After the permutation of 3rd round we can see the 0th bits of the SBox es goes to nibble 0, 1, 2, 3 in the next round (input of green color SBox es in round 4). Similarly 1st, 2nd and 3rd bits of the SBox es afer permutation layer of 3rd round goes to red, blue and yellow SBox es respectively. The ANF of input bits of each green SBox es in the 4th

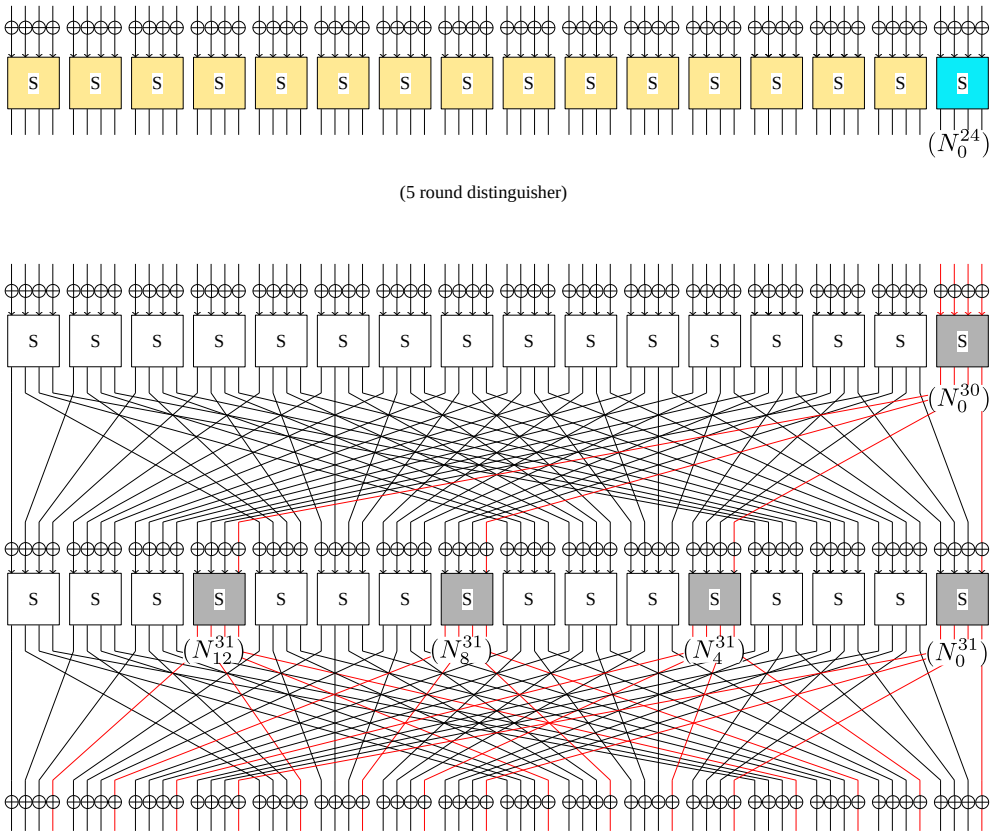

Figure 4: 2 rounds key recovery attack on PRESENT

round is $(c_0 + c_1y_0, c_2 + c_3y_1, c_4 + c_5y_2, c_6 + c_7y_3)$ respectively. At the output of 4th round, each bit contains combination of $y_3y_2y_1y_0$ in their ANF representaion. As the input bits of the nibble for different colored SBox es are same, their output also remains same without constants. Now in GIFT-64-128's SBox ANF representation, lowest degree term appears for 0th and 1st bits. But if we consider 2 rounds the possibility of not containing the product term $x_3x_2x_1x_0$ is higher for 0th bit. This is because the 2 degree term of 0th bit contains 0th bit and 1st bit from the last round and both of them are 2-degree term too. Whereas the 1st bit contains previous round's 0th and 2nd bit term in 2-degree but 2nd bit contain s a 3rd degree term in the last round. For this, after two rounds the chances of getting balancedness at 0th bit is higher than other bits. We implemented this in Sage and compute the ANF's of the bits after 5 rounds. We see that after permutation layer of 5th round, the 0th bit of each SBox does not contain the product term $x_3x_2x_1x_0$. The other bits ANF contains the term. Hence the 0th bits of each nibble becomes balanced after 5 rounds.

In the previoues paragraph we show the propagation after giving the All property at 0th nibble. Now we show that if we give All property at any random nibble then also, the same 16 bits becomes balanced. Suppose we give All at input of round 1 in nibble 0, 4, 8, 12. Then after permutation in the input of round 2, one of red nibble, blue yellow and green contains a constant combination of bit y_0, y_1, y_2, y_3 respectively. If we give All at nibble (1, 5, 9, 13) then input bit sequence at red, blue, yellow and green nibbles becomes (y_0, y_1, y_2, y_3), (y_1, y_2, y_3, y_0), (y_2, y_3, y_0, y_1), (y_3, y_0, y_1, y_2) respectively. So, we get a 4 bit

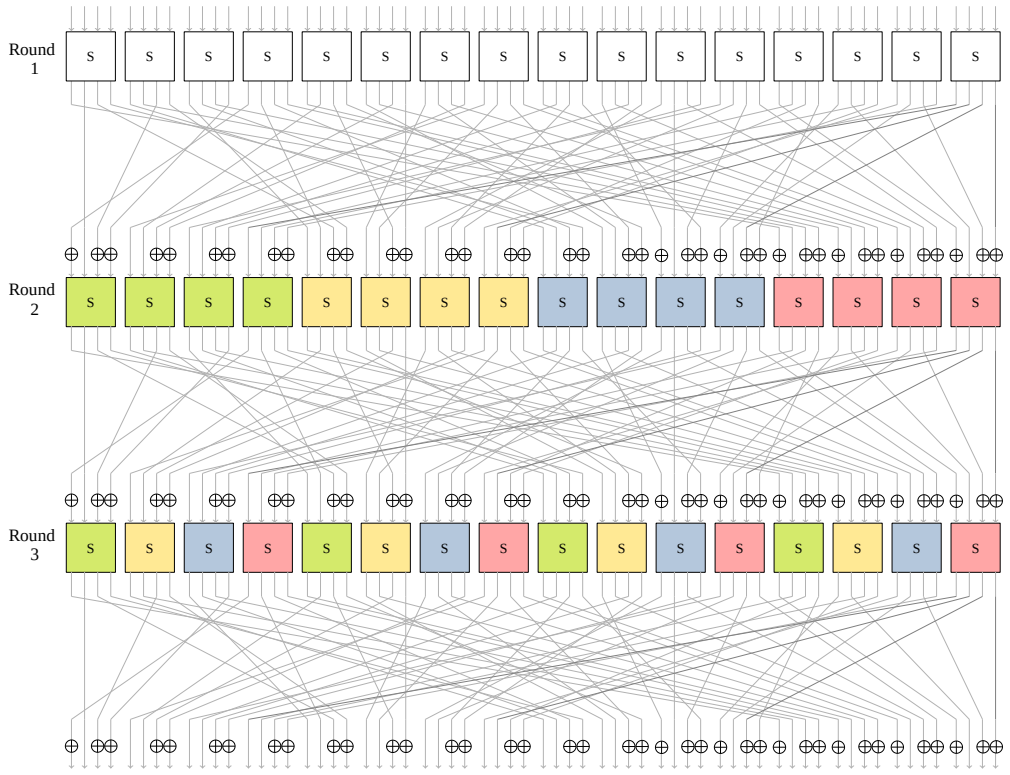

Figure 5: random nibble active bit propagation on GIFT

sequence for giving All at any nibble in round 1. In the output of round 3 for each bit in nibble 0, 1, 2, 3 we get the ANF as constant combination of one of (y_0, y_1, y_2, y_3), (y_1, y_2, y_3, y_0), (y_2, y_3, y_0, y_1), (y_3, y_0, y_1, y_2). So in the input of round 4, for nibble 0, 1, 2, 3 we get one of the ANF of the anf sequence which is shown in Figure 6. These 4 ANF sequences leads to the balancedness of 0th bit after 5th round. This is shown in the previous paragraph.

Algorithm 4 KEY RECOVERY $(C'_1, \cdots, C'_{16})$

Initialize an array called key_list
for each key_nibble in range{0 - 4} **do** ▷ output of SB at N_0^{31} in fig. **??**
  for each j in range {0 - 16} **do**
    dec_msg[j] $\leftarrow SB^{-1}$(key_nibble$\oplus$ perm$^{-1}(C'_j)$)
  xor_sum $\leftarrow$ dec_msg[0] $\oplus \cdots \oplus$ dec_msg[15]
  if 0th bit of xor_sum is 0 **then** ▷ input of SB at N_0^{31} in fig. **??**
    store {key_nibble} in key_list
return key_list

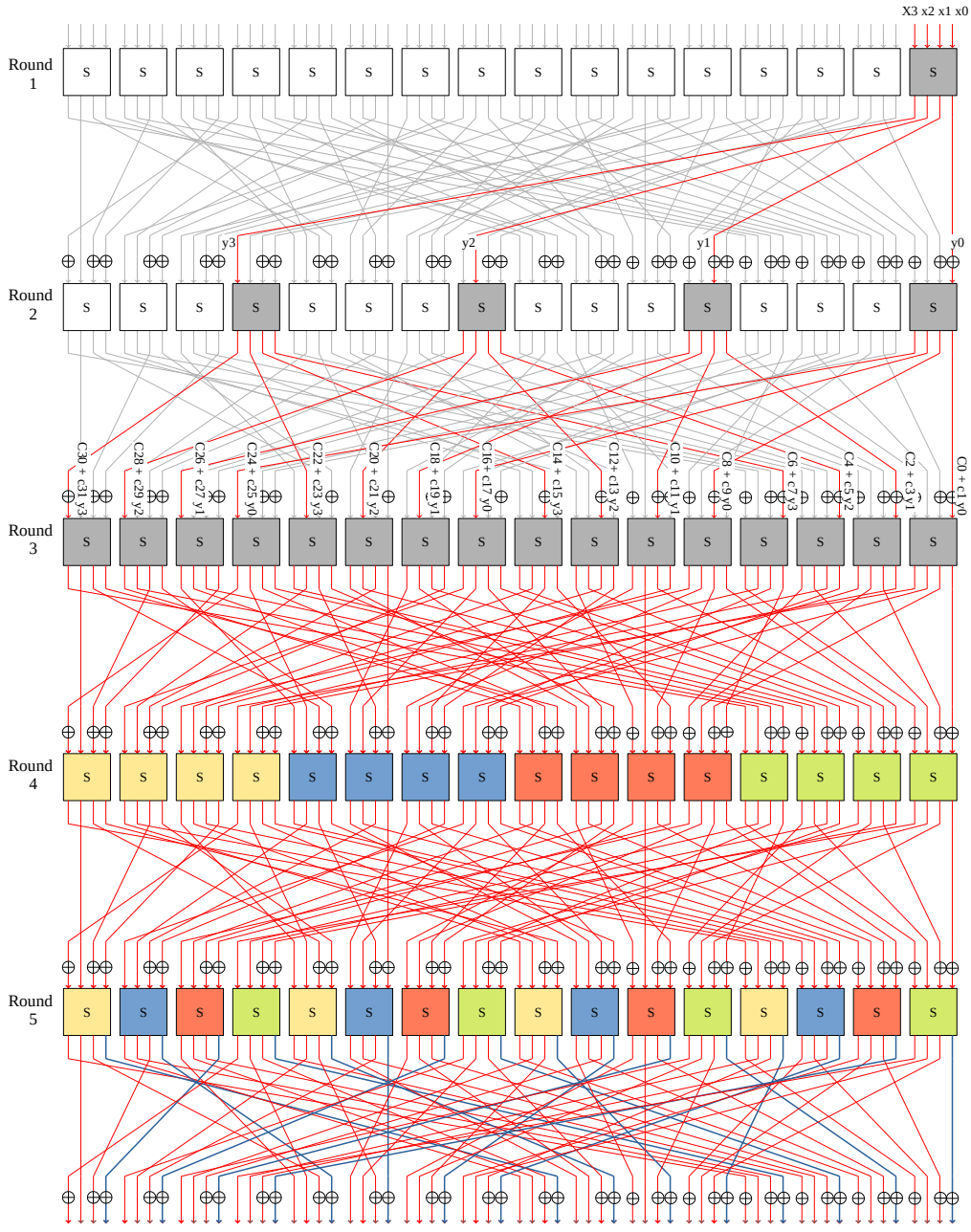

Figure 6: active bit propagation on `GIFT`

7 DTM of each models

References

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017.

[Lea10] Gregor Leander. Small scale variants of the block cipher PRESENT. *IACR Cryptol. ePrint Arch.*, page 143, 2010.

[Tod17] Yosuke Todo. Integral cryptanalysis on full MISTY1. *J. Cryptol.*, 30(3):920–959, 2017.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *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*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016.