Differential Fault Analysis of NORX

Amit Jana[1], Dhiman Saha[2], and Goutam Paul[1]

[1] Indian Statistical Institute, Kolkata
[2] Indian Institute of Technology, Bhilai

Abstract. In recent literature, there has been a particular interest in studying nonce based AE schemes in the light of fault based attacks as they seem to present an automatic protection against Differential Fault Attacks (DFA). In this work, we present the first DFA on nonce based CAESAR scheme NORX. We demonstrate a scenario when faults introduced in NORX in parallel mode can be used to collide the internal state to produce an *all-zero* state. We later show how this can be used to replay NORX despite being instantiated by different nonces, messages. Once replayed, we show how the key of NORX can be recovered using secondary faults and using the faulty tags. We use different fault models to showcase the versatility of the attack strategy. A detailed theoretical analysis of the expected number of faults required under various models is also furnished. Under the random bit flip model, around 1384 faults are to be induced to reduce the key space from 2^{128} to 2^{32} while the random byte flip model requires 136 faults to uniquely identify the key. To the best of our knowledge, this is the first fault attack that uses *both internal* and *classical differentials* to mount a DFA on a nonce based authenticated cipher which is otherwise believed to be immune to DFA.

1 Introduction

Fault analysis has always been one of the most popular physical attacks which is primarily attributed to the ease with which such attacks are mounted in practice and secondarily due to the difficulty in inhibiting them. The initiation of CAESAR competition [1] on authenticated encryption (AE) schemes, generated a lot of interest in the crypto community to analyze these ciphers which try to combine the goals of authenticity and confidentiality under a single unified primitive. Researchers also tried to look at authenticated ciphers from the fault analysis perspective. Authenticated ciphers submitted to CAESAR presented many interesting problems due to their diverse design strategies and paradigms as well as an array of useful features like Nonce-Misuse Resistance (NMR), Online Authenticated Encryption (OAE), Inverse Free, Release of Unverified Plaintexts (RUP), Parallelizable Encryption/Decryption and others (A good account of these is available in [2]). Eventually, it was found that some of these desirable features lead to previously nonexistent vulnerabilities with regards to fault attacks [22,23,25]. One of many ways that AE schemes can be classified in based on the use of *nonces* giving us two types of authenticated ciphers: one that prohibits reusing the nonce while the other that provides *some* security under nonce-reuse.

In CHES 2016 [23] and later in JCEN'17 [24] Saha and Roy Chowdhury using a demonstration on nonce-based authenticated cipher PAEQ, highlighted the importance of the *nonce-barrier* in the context of automatic prevention of Differential Fault Attacks (DFA) which is the most popular and well studied type of fault attack. The basic problem seems to be the fact that nonce based schemes inhibit replaying of the algorithm which is a premise to DFA thereby implicitly thwarting them. The authors show how parallelism in PAEQ could be exploited to mount a DFA by completely avoiding the nonce constraint. They also generalized the attack showcasing the threat it poses to parallelizable ciphers that employ the counter-mode.

In this work, we target another nonce-respecting CAESAR submission called NORX which survived up till the third round. NORX [5, 7, 8] is the family of sponge based authenticated encryption with associated data (AEAD) algorithms designed by Aumasson, Jovanovic and Neves. The original submission proposes versions of NORX with 32 and 64−bit words called `NORX32` and `NORX64` respectively. Subsequently two more versions were proposed with 8 and 16−bit words called respectively `NORX8` and `NORX16` in [6]. Our interest in NORX stems from the fact that it has a unique *parallel* architecture based on the *MonkeyDuplex* construction [11] and supports an arbitrary parallelism degree, based on LRX primitives. An intriguing feature of NORX is the way it instantiates parallelism due to the MonkeyDuplex mode. This deviates from the classical parallelizable ciphers as stated in [23], but NORX still uses a variant of the counter-mode to separate the branches. This is the reason that the ideas proposed in CHES'16 cannot be directly extended to get an attack on NORX.

Our Contribution. Our first contribution, comes in form of generating fault-based internal state collisions on NORX with level of parallelism = 2 leading to an *all-zero* state at the end of message processing. Once the collision is generated in *any*[3] NORX instantiation, if the trailing data is identical, then all such NORX instances will produce the same (faulty) tag. In other words, the tags would also collide which can be interpreted as a *faulty forgery*. This mimics the replay of the NORX which makes threat of classical DFA pertinent and devising the attack constitutes our second contribution. We find that if we are able to inject faults in the internal state of permutation in the last iteration of the round function, then based on the fault model adopted, one or multiple bits of internal state is revealed by the `XOR` of the tags. We perform extensive theoretical analysis using the well-known "Coupon Collector Problem" [21] to establish bounds on the expected number of faults required using *four* different fault models to reduce the key-space to practically acceptable limits. The summary of these results are furnished in Table 1.

Previous Cryptanalysis of NORX. In [4], Aumasson et al. thoroughly analyzed the differential and the rotational properties of the core permutation of NORX.

[3] We stress that this is true for any nonce and messages with consecutive identical blocks

Fault Model	Expected # Faults	Reduced Key-Space	Reference
Random Bit Flip	1384	2^{32}	Section 4.1
	1544	2^{16}	
Random Byte Flip	136	1	Section 4.2
Random Byte with Known Fault	332	2^{32}	Section 4.3
	372	2^{16}	
Random Consecutive Bit Flip	136	2^{32}	Section 4.4
	152	2^{12}	

Table 1: Summary of the attacks on NORX reported in this work

They gave upper bounds on the differential probability for the reduced permutation. More precisely, if an attacker can only modify the nonce during initialization, then any single round differential characteristic has probabilities of less than 2^{-60} (for NORX32) and 2^{-53} (for NORX64). Furthermore, they have found the best characteristics for four rounds with probabilities of 2^{-584} and 2^{-836} for NORX32 and NORX64, respectively. In [15], Das et al. describe statistical variants of zero-sum distinguishers that allow to distinguish the full-round permutation of NORX-64 and 3.5 rounds of the permutation of NORX-32 from random permutations. These results cover more rounds compared to the first order differential analysis provided in [4]. The used approach is similar to zero-sum distinguishers [9], but it is probabilistic rather than deterministic. Later in [10], Bagheri et al. showed a state/key recovery attack for both variants for a reduced versions of NORX v2.0 where the underlying permutation applies half the rounds (2 out of 4). After that, in [18] Dwivedi et al. analyzed the state-recovery resistance of several submitted CAESAR candidates, including NORX, using a SAT solver. They have also analyzed modified versions of these algorithms, including round-reduced variants. Later in [14], using non-random properties of the underlying permutation, they showed a ciphertext-only forgery with time and data complexity 2^{-66} (resp. 2^{-130}) for the variant of NORX v2.0 of 128-bit (resp. 256-bit) keys and broke the designer's claim of a 128-bit (resp. 256-bit) security. Furthermore, they showed that this forgery attack can be extended to a key-recovery attack on the full NORX v2.0 with the same time and data complexities. Also for NORX v3.0, the resulting attack enables an adversary to generate forgeries with data complexities $2^{2w}, w = 32, 64$ for 128, 256-bit keys respectively. It is interesting to see that despite a lot of cryptanalytic results reported in literature, there is no side-channel attack or physical attack developed on NORX. This forms one of the initial motivations of this work.

Outline. The rest of the paper is organized as follows: We begin by briefly outlining the description of NORX v3.0 in Section 2. We describe, how to create an internal collision on NORX with parallelism degree $p = 2$ using counter fault

in Section 3.1 and then give the attack scenario for the DFA on NORX ($p = 2$) in Section 3.2. A detailed theoretical analysis along with the expected number of faults required under various fault models is given in Section 4. A discussion is furnished in Section 5 while the concluding remarks are given in Section 6.

2 Specifications of NORX

In this section we will give a description of the NORX family of Authenticated Encryption with Associated Data (in short "AEAD") algorithms, mainly the description of NORX v3.0.

$$S = \left(\begin{array}{cccc} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ \hline s_{12} & s_{13} & s_{14} & s_{15} \end{array}\right) \qquad S^{init} = \left(\begin{array}{cccc} n_0 & n_1 & u_2 & u_3 \\ K^{(1)} & K^{(2)} & K^{(3)} & K^{(4)} \\ u_8 & u_9 & u_{10} & u_{11} \\ \hline u_{12} & u_{13} & u_{14} & u_{15} \end{array}\right)$$

(a) State representation (b) State initialization

Fig. 1: Matrix form of the state

A NORX instance is denoted by (w, l, p, t), where $w \in \{32, 64\}$ is the word size, $l(1 \leq l \leq 63)$ is the round number, $p(0 \leq p \leq 255)$ is a parallelism degree and $t(\leq 4w)$ is the tag size. The state S is viewed as concatenation of 16 words, i.e., $S = s_0||s_1||....||s_{15}$, where $s_0, ..., s_{11}$ are called the rate words (where data blocks are injected) and $s_{12}, ..., s_{15}$ are called the capacity words (which remain untouched). The state S can be viewed as a 4×4 matrix, is given in Figure 1a. More information on the constants can be found in [5]. The encryption algorithm takes as inputs a key K of k-bits, a nonce N of n-bits, a plaintext M and a associated data formed by a header A and a trailer Z. The header, plaintext and trailer are three optional strings. The encryption algorithm computes an authentication tag T of t-bits, and a ciphertext C of same bit-length as the plaintext M. Similarly, the decryption algorithm takes as inputs (K, N, A, C, Z, T) and returns either $\perp$ or M depending on the tag verification succeeds or not. Both encryption and decryption algorithms begin by an initialization phase that sets the internal state to S^{init}, consists of $4w$-bit key $K = K^{(1)}||K^{(2)}||K^{(3)}||K^{(4)}$, the $2w$-bit nonce $N = n_0||n_1$ and some initialization constants (u_i) in the internal state, is given in Figure 1b.

The basic building block of NORX is a permutation F, also called a round, and F^l is an l consecutive applications of F. The permutation F over the state S transform its columns with

$$G(s_0, s_4, s_8, s_{12}) \quad G(s_1, s_5, s_9, s_{13}) \quad G(s_2, s_6, s_{10}, s_{14}) \quad G(s_3, s_7, s_{11}, s_{15})$$

and then transform its diagonals with

$$G(s_0, s_5, s_{10}, s_{15}) \quad G(s_1, s_6, s_{11}, s_{12}) \quad G(s_2, s_7, s_8, s_{13}) \quad G(s_3, s_4, s_9, s_{14}).$$

Those two operations are denoted by $col(S)$ and $diag(S)$ respectively. The complete pseudo-code for the NORX core permutation F^l is given in Fig. 2. The G function uses cyclic rotations $\ggg$ and a non-linear operation H interchangeably to update its four input words a, b, c, d. The rotation offsets r_0, r_1, r_2, and r_3 for the cyclic rotations of 32- and 64-bit NORX are specified in Table 2. The designers proposed certain configurations of the mode to process the payload in parallel. The parallel mode is controlled by the parameter $0 \leq p \leq 255$. For $p = 1$, the design of NORX corresponds to the sequential duplex construction, is shown in Fig. 3. For $p > 1$, e.g., for $p = 2$, the design of NORX with parallelism degree 2 is shown in Fig. 4. The parameter combinations of the NORX variants are given in [13, Table 1].

w	r_0	r_1	r_2	r_3
32	8	11	16	31
64	8	19	40	61

Table 2: Rotation offsets for `NORX32` and `NORX64`

Algorithm 1: $col(S)$

c1. $(s_0, s_4, s_8, s_{12}) \leftarrow G(s_0, s_4, s_8, s_{12})$;
c2. $(s_1, s_5, s_9, s_{13}) \leftarrow G(s_1, s_5, s_9, s_{13})$;
c3. $(s_2, s_6, s_{10}, s_{14}) \leftarrow G(s_2, s_6, s_{10}, s_{14})$;
c4. $(s_3, s_7, s_{11}, s_{15}) \leftarrow G(s_3, s_7, s_{11}, s_{15})$;
c5. return S;

Algorithm 2: $diag(S)$

d1. $(s_0, s_5, s_{10}, s_{15}) \leftarrow G(s_0, s_5, s_{10}, s_{15})$;
d2. $(s_1, s_6, s_{11}, s_{12}) \leftarrow G(s_1, s_6, s_{11}, s_{12})$;
d3. $(s_2, s_7, s_8, s_{13}) \leftarrow G(s_2, s_7, s_8, s_{13})$;
d4. $(s_3, s_4, s_9, s_{14}) \leftarrow G(s_3, s_4, s_9, s_{14})$;
d5. return S;

Algorithm 3: $H(x, y)$

H1. return $(x \oplus y) \oplus ((x \wedge y) \ll 1)$;

Algorithm 4: $F^l(S)$

F1. **for** $i = 0$ *to* $l - 1$ **do**
  $S \leftarrow diag(col(S))$;
F2. return S;

Algorithm 5: $G(a, b, c, d)$

G1. $a \leftarrow H(a, b)$;
G2. $d \leftarrow (a \oplus d) \ggg r_0$;
G3. $c \leftarrow H(c, d)$;
G4. $b \leftarrow (b \oplus c) \ggg r_1$;
G5. $a \leftarrow H(a, b)$;
G6. $d \leftarrow (a \oplus d) \ggg r_2$;
G7. $c \leftarrow H(c, d)$;
G8. $b \leftarrow (b \oplus c) \ggg r_3$;
G9. return a, b, c, d;

Fig. 2: The NORX permutation F^l

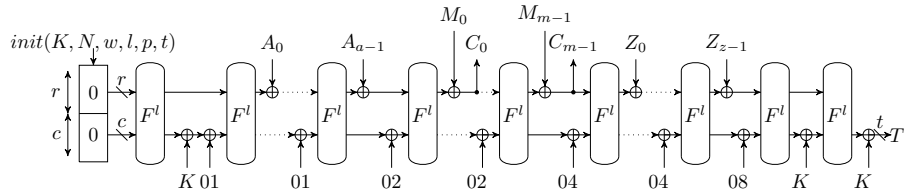


Fig. 3: Layout of standard NORX ($p = 1$)

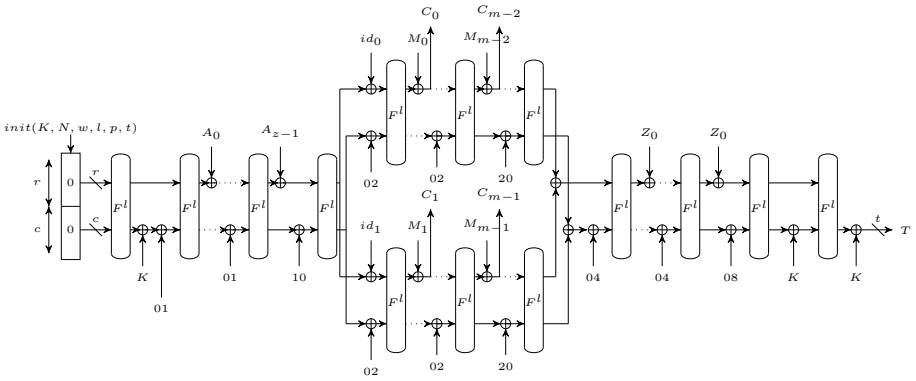


Fig. 4: Layout of NORX with parallel encryption ($p = 2$)

3 Attack Scenario

In this section, we illustrate the fault injections required to first create a replay in the nonce respecting scenario, i.e., create same states by a counter fault to get the all-zero state after merging, and then we will perform DFA. To perform differential analysis on NORX with parallelism degree $p = 2$, for each message query, the attacker will inject two faults where the first fault is a counter fault to create a replay and the second fault will be at the capacity words inside the last diag(S) call (i.e., just before the tag T is obtained). Thus it shows that the attacker moves from a single fault model to a double fault one. However, in this particular context, it is clear that performing the double fault is not too difficult compared to simple one as

1. the attacker can send a long enough message so that the second fault occurs far enough from the first one and
2. thus, we are able to observe the success of the first fault so that the attacker can tune the two faults settings independently.

3.1 Creating a Replay on NORX with Parallel Encryption (p = 2)

An important requirement of DFA is the ability to replay the execution of this cipher in order to exploit the difference between faulty and non-fault outputs. The

DFA in a cryptosystem, requires an attacker to be able to inject faults by replaying a previous fault-free run of the algorithm, which is known as the Replaying criterion of DFA. The definite structure of nonce-based encryption proposed by Rogaway in [20] expects the uniqueness of the nonce in every instantiation of the cipher and the security claims rely on this premise. Thus the use of a unique nonce contradicts the ability to replay a cipher and thereby resulting in an automatic protection from DFA. So it is quite clear that the DFA is not applicable for the standard NORX with $p = 1$. But for NORX with parallelism degree $p = 2$, we can make an exception by inducing counter fault and processing the messages in a specific way for each query. For each message query, we can produce a replay i.e., creating a fixed state $S^0 = \mathbf{0}$ after the merging phase (i.e., before processing the trailing data) by injecting a fault on the counter id_1 such that $id_1 = id_0 = 0$ and processing the message $M = M_0||M_1||\cdots||M_{m-2}||M_{m-1}$ such that $M_i = M_{i+1}, i = 0, 2, \cdots, m-2$. This description is outlined in Figure 5. In the rest of the paper, we will use this replay criterion to analyze the DFA on NORX ($p = 2$), which we will be denoting by NORX$^{p=2}$.

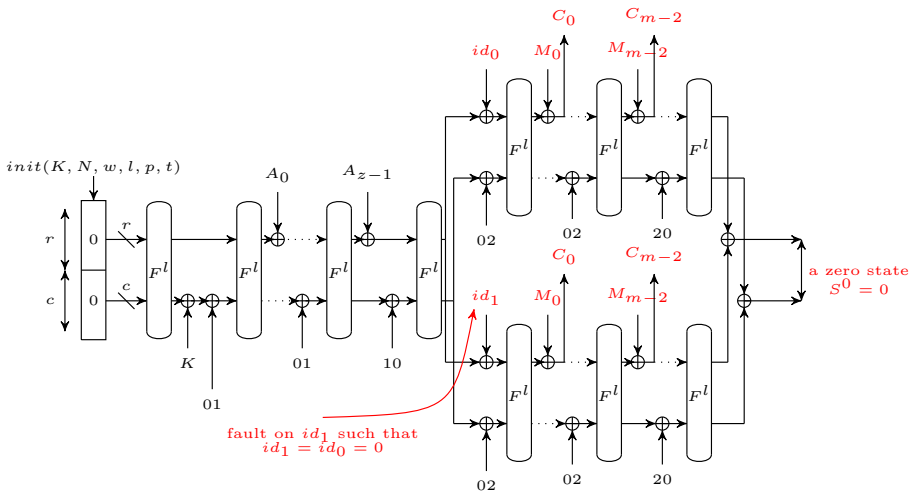


Fig. 5: Counter fault on NORX with parallel encryption ($p = 2$)

Feasibility of the Counter Fault. According to the branch algorithm in [8], for $p = 2$, the branching is done by `XOR`ing the variable $i(= 1)$ at 12 different positions in the state. To produce a replay, an attacker can inject a fault at the counter i in two different ways. The first one is by injecting a precise bit fault at the `lsb` of the variable i so that the state remain same after `XOR`ing with 12 different positions of the state and immediate after, induce a random single bit fault on i. Another one is by injecting a random bit fault at the variable i except

the `lsb` so that it will skip the for loop to produce branching and leaves both the states same after the branching phase.

3.2 Injecting Fault at the Last diag(S) Call

In NORX, an authentication tag T is generated by first injecting the domain separation constant `08`, then transforming the state S twice with the permutation F^l interleaved by two key additions to the capacity, and finally extracting the $t(\leq 4w)$ rightmost capacity bits from S, i.e., $T = F^l(F^l(S \oplus 08) \oplus K) \oplus K$. Thus we can think of the tag T as the `XOR`ing of the outputs of the last $diag(S)$ operation with the master key K. We denote the last $diag(S)$ operation to generate the tag T as $ldiag(S)$ and follow this notation in the rest of the paper. The tag T can be viewed as $T_1||T_2||T_3||T_4$ and the key K as $K^{(1)}||K^{(2)}||K^{(3)}||K^{(4)}$ respectively, where $|K^{(i)}| = |T_i| = w, i = 1, 2, 3, 4$. The core permutation F of NORX has a natural parallelism of 4 independent G applications. The four diagonal words $(s_0, s_5, s_{10}, s_{15})$, $(s_1, s_6, s_{11}, s_{12})$, (s_2, s_7, s_8, s_{13}) and (s_3, s_4, s_9, s_{14}) are fed into the G-function independently inside the $diag(S)$ call. So after the $ldiag(S)$ operation, the tag $T = T_1||T_2||T_3||T_4$ can be viewed as independent `XOR`ing of the outputs of 4 parallel executions of the G function with the keys $K^{(1)}, K^{(2)}, K^{(3)}, K^{(4)}$, depicted in Figure 6.

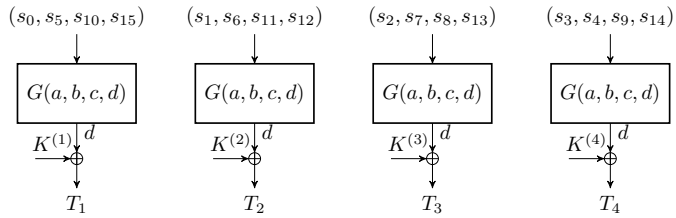


Fig. 6: Parallel execution of G-functions inside the last $diag(S)$ call to generate the tag T

If we take a look at the input (a, b, c, d) to the G application, it is clear that the word d is nothing but a word that belongs to the capacity part of the state. Inside one of the four independent G applications during $ldiag(S)$ operation, at step (G6) in Algorithm 5, the updated word d will produce the corresponding tag word $T_i, i = 1, 2, 3, 4$ by `XOR`-ing with the corresponding word of the key K. Let Δ^W denotes the faulty word due to faults injected on W at the time of execution among any steps from (G1) to (G5) inside one of the four independent G applications of the $ldiag(S)$ operation. In the rest of the paper, Δ^b means the faults are induced at step (G5), whereas Δ^c means at step (G3). Also let T be the fresh tag and T^{Δ^W} denotes the faulty tag due to Δ^W. Inside the G application, H is the only non-linear operation that helps to increase the degree of a boolean function. Our aim is to recover the bits of d by inducing faults

on a word and observe the XORed relation of the faulty and non-faulty tags[4]. Following are the steps to recover the bits of the word d that we also outlined in Figure 7. The overall fault induction process using primary (counter) faults and secondary ($ldiag(S)$) faults is furnished in Figure 8.

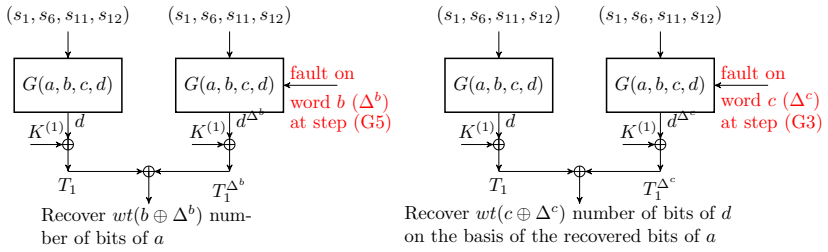


Fig. 7: Fault inducing steps to recover the capacity d of the state S inside the G-function

1. Recover the corresponding bits of a by injecting a fault Δ^b and repeat this step until we recover all the bits of a.
2. Based on the recovered bits of a, we can recover the bits of d by injecting a fault Δ^c and repeat this step until we recover all the bits of d.

4 DFA on NORX with Different Fault Models

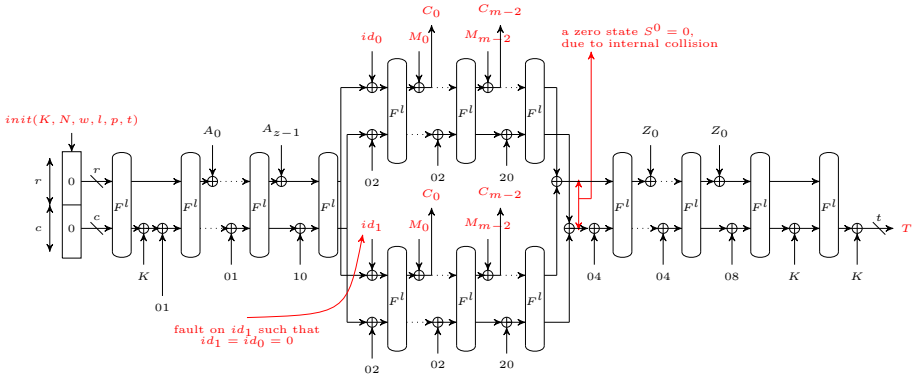
Before going to discuss several fault models, we need to build the bit relations of the words a, b, c, d from step (G5)-(G6) and step (G3)-(G6) inside the G application respectively.

Bit relations of step (G5)-(G6): Assuming the words a, b, c, d are unknown upto step (G4) inside the G application. Then the update of the words a, b, c, d from step (G5) to step (G6) inside the G application (more details are given in Appendix A.1), the capacity word d after step (G6) can be written in bit level relations as,

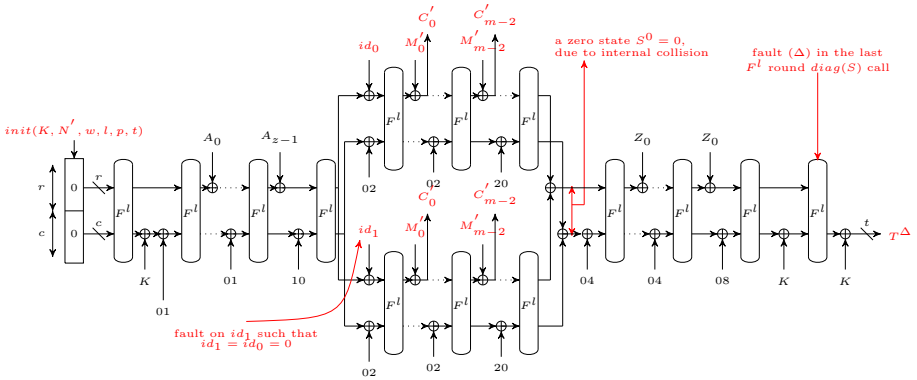
$$d_i = \begin{cases} d_{i+16} \oplus a_{i+16} \oplus b_{i+16} \oplus (a_{i+17} \wedge b_{i+17}), & \text{if } i \in \{1, 2, \cdots, 32\} \setminus \{15\}; \\ d_{31} \oplus a_{31} \oplus b_{31}, & \text{if } i = 15. \end{cases} \quad (1)$$

Bit relations of step (G3)-(G6): Assuming the words a, b, c, d are unknown upto step (G2) inside the G application. Update of the words a, b, c, d from

[4] Here non-faulty tag means, the tag where we don't induce any fault at the last $diag(S)$ call, after the internal collision in the state by giving a counter fault. But physically, all tags are faulty due to the prerequisite of fault based internal state collision.



(a) A non-faulty tag T using an internal collision



(b) A faulty tag T^Δ using an internal collision

Fig. 8: Replay on NORX ($p = 2$)

step (G3) to step (G6) inside the G application (more details are given in Appendix A.2), the capacity word d can be written as,

$$d_i = \begin{cases} d_{i+16} \oplus a_{i+16} \oplus b'_{i+16} \oplus (a_{i+17} \wedge b'_{i+17}), & \text{if } i \in \{1, 2, \cdots, 32\} \setminus \{15\}; \\ d_{31} \oplus a_{31} \oplus b'_{31}, & \text{if } i = 15. \end{cases} \quad (2)$$

where

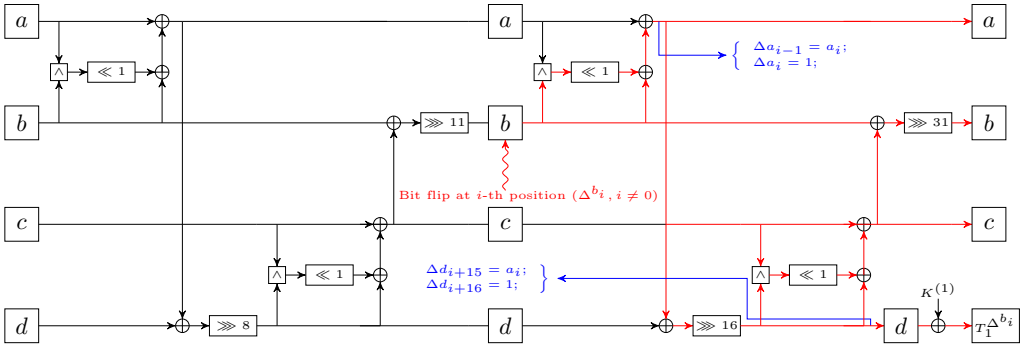
$$b'_i = \begin{cases} b_{i+21} \oplus c_{i+21} \oplus d_{i+21} \oplus (c_{i+22} \wedge d_{i+22}), & \text{if } i \in \{1, 2, \cdots, 32\} \setminus \{10\}; \\ b_{31} \oplus c_{31} \oplus d_{31}, & \text{if } i = 10. \end{cases} \quad (3)$$

For the DFA on NORX with $p = 2$, two faults are injected for each encryption queries. One corresponds to create internal state collision (replay) and another one corresponds to induce faults at the last $ldiag(S)$ operation. So for the DFA, analyzing the relation among the faulty and non-faulty tags by injecting faults on a word inside the $G(s_0, s_5, s_{10}, s_{15})$ will be sufficient to count the total number of faults required to recover the full key K, where $(s_0, s_5, s_{10}, s_{15})$ is the first diagonal input of the $ldiag(S)$ operation. Since at the $ldiag(S)$ operation, four capacity words are the inputs (correspond to four diagonal inputs) to the four independent G applications. Let n be the number of faults (except the counter faults) required, by analyzing the relation among the faulty and non-faulty tags of $G(s_0, s_5, s_{10}, s_{15})$. Thus, the total number of faults (except the counter faults) will be $4n$. In the rest of the paper that follows, *any fault Δ^b/Δ^c means that the fault induced on the word b/c at step (G5)/(G3) inside the $G(a, b, c, d)$ application with respect to first diagonal input* and the corresponding faulty tag will be $T_1^{\Delta^b}/T_1^{\Delta^c}$.

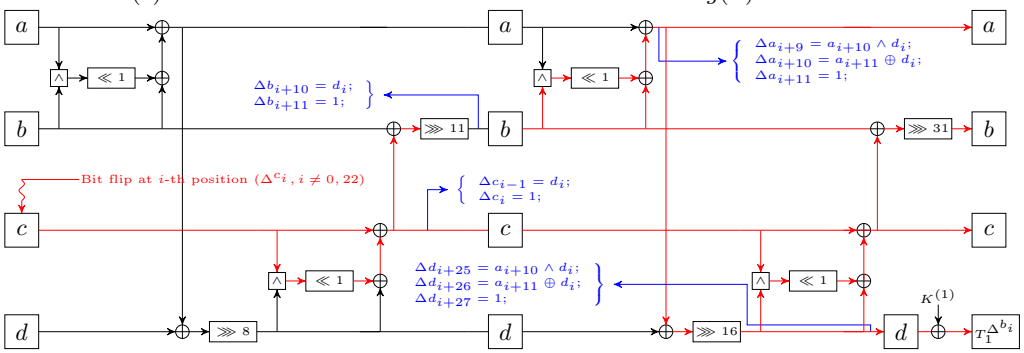
4.1 Random Bit-flip Fault Model

Here we will show some interesting relations among the words a, b, c, d by inducing a single bit random fault either on the word b or on c, by formalizing their respective `XOR`ed relations of the fresh tag and the faulty tag. We categorize this fault model into two cases depending on a fault is injected either in the word b or in c.

Case 1. Let $\Delta^{b_i}, 0 \leq i \leq 31$, denote a bit flip at the i-th position in the word b and $T_1^{\Delta^{b_i}} = d^{\Delta^{b_i}} \oplus K^{(1)}$ be the faulty tag corresponds to Δ^{b_i}. Due to $\Delta^{b^i}, i \neq 0$, b_i will be present at two consecutive positions in d as $i+15$ and $i+16$ according to the Equation (1). But for Δ^{b_0}, the bit b_0 will be present at one position in d as 16. Figure 9a illustrates the location of the bit fault injection to retrieve the corresponding bit of the word a. Thus for any Δ^{b^i}, the generalized form of the



(a) Bit fault Δ^{b_i} inside the G-function at the last $diag(S)$ call



(b) Bit fault Δ^{c_i} inside the G-function at the last $diag(S)$ call

Fig. 9: A bit flip scenarios in the G circuit. Here Δa_i represents the value of the difference of the word a and the faulty word a due to bit flip in a word, at the i-th position. Similarly, $\Delta b_i, \Delta c_i, \Delta d_i$ carry the same meaning.

XORed tag will look like as[5]

$$(T_1 \oplus T_1^{\Delta^{b_i}})[j] = \begin{cases} \begin{cases} a_i, \text{ if } j = i+15 \\ 1, \text{ if } j = i+16 \\ 0, \textit{ otherwise} \end{cases}, \text{if } i \in \{0, \cdots, 31\} \setminus \{0\}; \\ \begin{cases} 1, \text{if } j = 16 \\ 0, \textit{otherwise} \end{cases}, \quad \text{if } i = 0. \end{cases} \tag{4}$$

Rigorous analysis of the above relations are given in Appendix B.1. Therefore it is clear from Equation (4) that, for any Δ^{b_i}, we can find the corresponding faulty bit position in the word b by observing the non zero value 1 from the relation $T_1 \oplus T_1^{\Delta^{b_i}}$. So using this relation, we will retrieve all the bit values of the word a except a_0 by injecting random bit faults in the word b.

Case 2. Similarly, $\Delta^{c_i}, 0 \leq i \leq 31$, denote a bit flip at the i-th position in the word c. Let $T_1^{\Delta^{c_i}} = d^{\Delta^{c_i}} \oplus K$ be the faulty tag corresponding to Δ^{c_i}. According to the Equation 2, the bit c_i will be present at three consecutive positions in d as $i+25$, $i+26$, $i+27$ respectively due to $\Delta^{c^i}, i \neq 0, 22$. But for $\Delta^{c^i}, i = 0, 22$, the bit c_i will be present at two consecutive positions in d as $i+26, i+27$ respectively. Figure 9b illustrates the location of the bit fault injection to retrieve the corresponding bit of the word d. Thus for any Δ_{c_i}, the generalized form of the XORed tag will be,

$$(T_1 \oplus T_1^{\Delta^{c_i}})[j] = \begin{cases} \begin{cases} a_{i+10} \wedge d_i, \text{ if } j = i+25 \\ a_{i+11} \oplus d_i, \text{ if } j = i+26 \\ 1, \quad \text{if } j = i+27 \\ 0, \quad \textit{otherwise} \end{cases}, \text{if } i \in \{0, \cdots, 31\} \setminus \{0, 21, 22\}; \\ \begin{cases} a_{31} \wedge d_{21}, \text{if } j = 14 \\ d_{21}, \quad \text{if } j = 15 \\ 1, \quad \text{if } j = 16 \\ 0, \quad \textit{otherwise} \end{cases}, \quad \text{if } i = 21; \\ \begin{cases} a_1 \oplus d_{22}, \text{if } j = 16 \\ 1, \quad \text{if } j = 17 \\ 0, \quad \textit{otherwise} \end{cases}, \quad \text{if } i = 22; \\ \begin{cases} a_{11}, \text{if } j = 26 \\ 1, \quad \text{if } j = 27 \\ 0, \quad \textit{otherwise} \end{cases}, \quad \text{if } i = 0. \end{cases} \tag{5}$$

Rigorous analysis of the above relations are given in Appendix B.2. Based on the retrieved bits in the word a, we can recover all the bits in the word d, except

[5] In this paper, for any word W, both of the notations $W_i, W[i]$ carry the same meaning, i.e., both represent the i-th position of the word W.

d_0 from the Equation (5), by injecting random bit faults in c such that all bit positions covered at least once. If we can recover d (except d_0), then the secret key $K^{(1)}$ (except $K_0^{(1)}$) will be recovered by XORing the respective word d and the non-faulty tag. For the unknown key bit $K_0^{(1)}$, we will guess that bit and check the corresponding tag for a given query.

Before estimating the number of independent parallel faults, we will first estimate the number of single bit faults on a word such that each bit of that word will be flipped at least once. Now this problem is equivalent to the well known problem, called "Coupon Collector Problem [21]". The estimation of the number of bit flips is given by the following result.

Theorem 1. *For any word W of length w, let X be a discrete random variable that represents the number of random bit flips on W such that each bit position will be flipped at least once. Then the expected number of random bit flips required in order to hit each of the bit position of the word W will be $w \cdot H_w$, i.e., $E[X] = w \cdot H_w$, where $H_w = \sum_{i=1}^{w} \frac{1}{i}$ is the w^{th} harmonic number.*

Corollary 1. *For any word W of length w, let X be a discrete random variable that represents the number of random bit flips on W such that l different bit positions will be flipped at least once. Then the expected number of random bit flips required in order to hit at least l different bit positions of the word W will be $w \cdot (H_w - H_{w-l})$, i.e., $E[X] = w \cdot (H_w - H_{w-l})$, where $H_w = \sum_{i=1}^{w} \frac{1}{i}$ is the w^{th} harmonic number.*

Number of Recovered bits (r)	Expected Number of Δ^{b_i} (z_1)	Expected Number of Δ^{b_i} (z_2)	Total Number of faults ($= 2 \cdot (z_1 + z_2)$)	Brute-force complexity
16	520	88	1206	2^{64}
20	520	124	1288	2^{48}
24	520	172	1384	2^{32}
28	520	252	1544	2^{16}
30	520	328	1696	2^{8}
31	520	392	1824	2^{4}
32	520	520	2080	2^{0}

Table 3: Expected number of single bit random faults to recover the key K

4.2 Random Byte Flip Fault Model

Any word W of size $w \in \{32, 64\}$ can be viewed as four sequential bytes W^0, W^1, W^2, W^3 respectively, where $W^j = (W_{8j}, W_{8j+1}, W_{8j+2}, W_{8j+3}, W_{8j+4},$

W_{8j+5},W_{8j+6},W_{8j+7}), $j = 0, 1, 2, 3$. In this model, we will flip a byte randomly out of four bytes of the word and collect the corresponding faulty tag. So to recover $K^{(1)}$ such that $T_1 = d \oplus K^{(1)}$, we will first recover the word a using byte flips on the word b and based on the retrieved word a we can recover the word d by inducing byte flips on the word c.

Case 1. Byte Flipping on the word b: Let $\Delta^{b^i}, i = 0, 1, 2, 3$ represents that a byte b^i is flipped out of the word b. According to the Equation 1, the XORed relation of the two tags $T_1 \oplus T_1^{\Delta^{b^i}}, i = 0, 1, 2, 3$ is given in Figure 10, where $T_1, T_1^{\Delta^{b^i}}$ represents the non-faulty and the faulty tag due to Δ^{b^i} respectively.

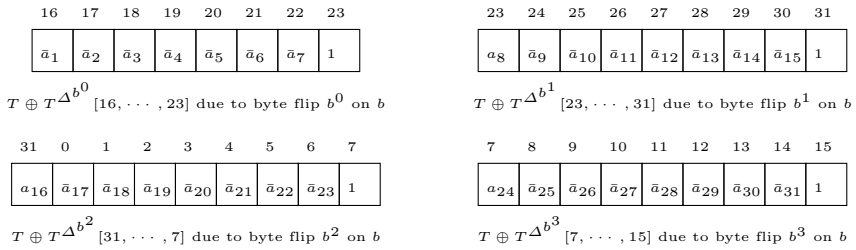


Fig. 10: Four different cases of byte flipping on the word b

Case 2. Byte Flipping on the word c: Similarly, $\Delta^{c^i}, i = 0, 1, 2, 3$ represents a byte c^i is flipped out of the word c. According to the Equation [2,3], the xored relation of the two tags $T_1 \oplus T_1^{\Delta^{c^i}}, i = 0, 1, 2, 3$ is given in Figure 11, where $T_1, T_1^{\Delta^{c^i}}$ represents the non-faulty and the faulty tag due to Δ^{c^i} respectively. From case 1, it is clear that we can recover all bits of a except a_0, by injecting some random byte flips in the word b such that all the four bytes of b will be flipped at least once. Similarly from case 2, we we can recover all the bits of d except d_0 by injecting some random byte flips on the word c. So we will recover the key $K^{(1)}$ except $K_0^{(1)}$ due to the unrecoverable bit d_0. To estimate the number of byte flips in a word so that all the 4 bytes will be flipped at least once, we will use Lemma 1. As the word contains four bytes, the expected number of byte flips needed to cover all the four bytes will be $4 \cdot H_4 = 8.3$. Hence for both of the cases, the expected number of byte flips will be $2 \cdot 4 \cdot H_4 \approx 17$. Since the capacity of the state has four words, so the total number of byte faults at the $ldiag(S)$ operation will be $4 \cdot 17 = 68$. Therefore, the total number of faults (including counter faults) will be $2 \cdot 68 = 136$.

4.3 Random Byte with Known Fault Model

In this Model, a random byte fault (Δ) will be injected on a word with a known fault value, i.e., Δ is known. If T_1, T_1^{Δ} represent the non-faulty and the faulty

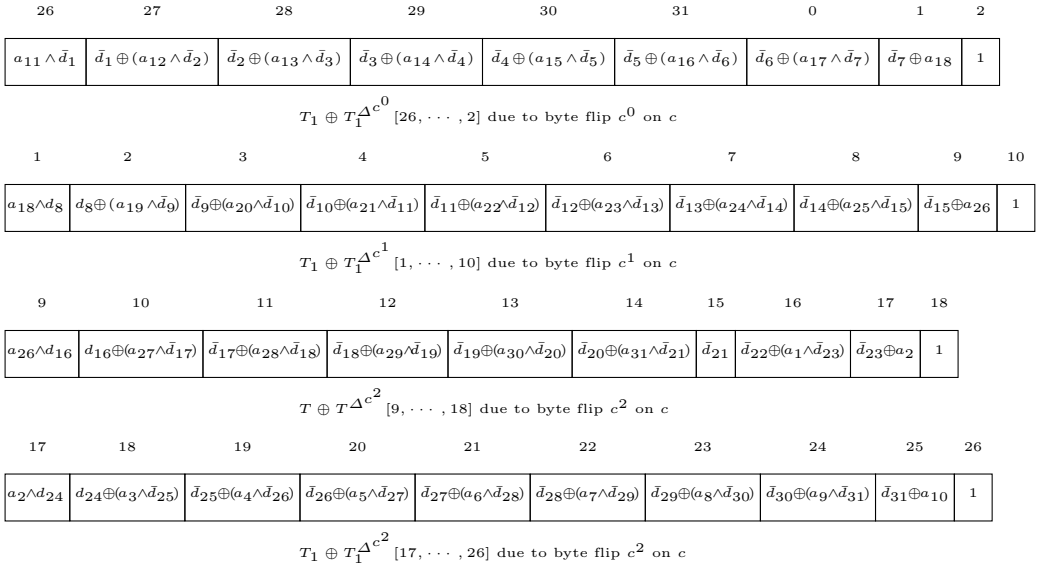


Fig. 11: Four different cases of byte flippings on the word c

tags due to a random byte with known fault Δ in a word respectively, then we can easily rectify the corresponding faulty byte of the word from the relation $T_1 \oplus T_1^{\Delta}$ by observing 1 on the corresponding byte position. However if we know Δ, then we can recover exactly $wt(\Delta)$ number of bits from their `XOR`ed tag relation, where $wt(\Delta)$ represents the hamming weight of Δ. But for first byte fault Δ such that the 0-th bit is flipped, then we can recover exactly $wt(\Delta) - 1$ number of bits from their `XOR`ed tag relation. This happens because the bit information $(x_0 \wedge y_0)$ will be lost due to one left shift inside the H-function of NORX core permutation. Let $\Delta^{W^i}, i = 0, 1, 2, 3$ represents the corresponding byte fault on the word W, where $W = (W_0, \cdots, W_{31})$ and $W^j = (W_{8j}, W_{8j+1}, W_{8j+2}, W_{8j+3}, W_{8j+4}, W_{8j+5}, W_{8j+6}, W_{8j+7})$, $j = 0, 1, 2, 3$.

Thus for any $\Delta^b (= \Delta^{b^0} || \Delta^{b^1} || \Delta^{b^2} || \Delta^{b^3})$, we will catch which $\Delta^{b^i}, i = 0, 1, 2, 3$ happens because the relation $T_1 \oplus T_1^{\Delta^{b^i}}$ will contain a 1 corresponding to that byte position and will be 0 for the remaining bytes. According to this model, as we know the value of Δ^{b^i}, we will extract the bits information of a^i corresponding to the non-zero bits of Δ^{b^i} from their `XOR`ed tag $T_1 \oplus T_1^{\Delta^{b^i}}$. Similarly we will recover the word d by injecting Δ^{c^i} and extract the bits information of d^i corresponding to the non-zero bits of Δ^{c^i} from $T_1 \oplus T_1^{\Delta^{c^i}}$. For better understanding, we give an example of two random byte difference with known value as $\Delta_1 =$ `0x9B000000`, $\Delta_2 =$ `0x005C0000` (in hexadecimal notation) on both the word b and c in Figure 12. So we have to estimate that how many random byte

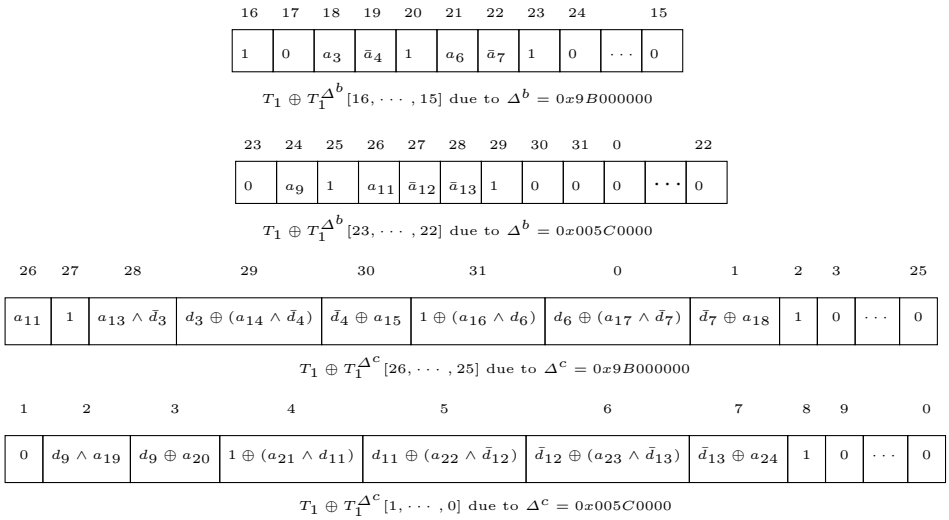


Fig. 12: Two different cases of byte flippings on the word b as well on c

faults are required such all the bits of the word will be flipped at least once.

By simulating the above problem for 10^5 times, the expected number of byte faults to cover the whole bit positions of the word is 29. So to recover the word a and then d, total number byte faults required will be approximately $2 \cdot 29 = 58$. But instead of recovering all the bits of the word d, we can recover r (≥ 16) number of bits of the secret key $K^{(1)}$ and the remaining bits will be recovered by guessing them. Our simulation result is given in Table 4.

No. of Recovered Bits (r)	Expected No. of Δ^{b^i} ($= Z_1$)	Expected No. of Δ^{c^i} ($= Z_2$)	Total No. of faults ($= Z_1 + Z_2$)	Brute-force complexity
16	126	20	292	2^{64}
20	126	32	316	2^{48}
24	126	40	332	2^{32}
28	126	60	372	2^{16}
30	126	76	404	2^{8}
31	126	92	436	2^{4}
32	126	126	504	2^{0}

Table 4: Expected number of random byte with known faults to recover the key K

4.4 Consecutive Bit Flip Fault Model

Here, we will discus the generalization of the xored relation of the non-faulty and the faulty tag, due to consecutive bit ($l, 1 \leq l \leq 32$) flip in a word. For $l = 1$, we already discuss it in Section 4.1. In this model, there are two scenarios.

1. For each replay, the adversary can flip a random but a fixed number of consecutive bits in a word, i.e., l is fixed for each replay, but the starting position of the consecutive bit flip injected by the adversary is random.
2. Another scenario is that, where both are random, i.e., both of l and the starting position of the consecutive bit flip are random.

Consecutive bits fault on b: Let $\Delta^{b_{[i,i+l-1]}}$ denotes any l consecutive bits flip in the word b starting from the bit position i and $T_1^{\Delta^{b_{[i,i+l-1]}}}$ be the corresponding faulty tag. According to the Equation (1), the bit b_0 will appear in one position in d as d_{16}, where as the remaining b_i's, $i \neq 0$ will appear at two consecutive positions in d as d_{i+15}, d_{i+16} respectively. Thus for any l consecutive bits flip, if 0^{th} bit of the word b is flipped, then there will be several cases to their `XOR`ed tag relation, otherwise if $0 \notin \{i, \cdots, i+l-1\}$, then it will be quite easy to follow the bit relation of $T_1 \oplus T_1^{\Delta^{b_{[i,i+l-1]}}}$. Rigorous analysis of the `XOR`ed tag relation for this case (i.e., $0 \notin \{i, \cdots, i+l-1\}$) as well as the other corner cases (i.e., when $0 \in \{i, \cdots, i+l-1\}$) are given in Appendix C.1. So for any $\Delta^{b_{[i,i+l-1]}}$ such that $0 \notin \{i, \cdots, i+l-1\}$, the generalized `XOR`ed tag relation can be written as,

$$(T_1 \oplus T_1^{\Delta^{b_{[i,i+l-1]}}})[j] = \begin{cases} a_i, & \text{if } j = i+15; \\ \bar{a}_{i+j}, & \text{for } j \in \{i+16, \cdots, i+14+l\}; \\ 1, & \text{if } j = i+15+l; \\ 0, & \textit{otherwise}. \end{cases} \tag{6}$$

From the Equation 6 or from the relations build by the corner cases, we can retrieve all the bits of a (except a_0) by inducing some random $\Delta^{b_{[i,i+l-1]}}$ in the word b so that all the bits in the word b will be flipped at least once.

Consecutive bits fault on c: Similarly, let $\Delta^{c_{[i,i+l-1]}}$ denotes any l consecutive bits flip starting from the bit position i in the word c and $T_1^{\Delta^{[c_{i,i+l-1}]}}$ be the corresponding faulty tag. According to the Equations (2) & (3), the c_0 will appear at two positions in d as d_{26}, d_{27} respectively, c_{21} will appear at three consecutive position in d as d_{14}, d_{15}, d_{16} respectively and c_{22} will appear at two consecutive position in d as d_{16}, d_{17} respectively. Whereas the remaining c_i's, $i \neq 0, 21, 22$ will appear at three consecutive positions in d as $d_{i+25}, d_{i+26}, d_{i+27}$ respectively. The generalized `XOR`ed tag relation for $\Delta^{c_{[i,i+l-1]}}$ such that $0, 21, 22 \notin \{i, \cdots, i+l-1\}$ is given below. Whereas the rigorous analysis of all the `XOR`ed tag relations

including the above corner cases, are given in Appendix C.2.

$$(T_1 \oplus T_1^{\Delta^{c[i,i+l-1]}})[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25; \\ d_i \oplus (\bar{d}_{i+1} \wedge a_{i+11}), & \text{if } j = i+26; \\ \bar{d}_{i+j-1} \oplus (\bar{d}_{i+j} \wedge a_{i+12}), & \text{for } j \in \{2 \cdots, l-1\}; \\ \bar{d}_{i+l-1} \oplus a_{i+10+l}, & \text{if } j = i+25+l; \\ 1, & \text{if } j = i+26+l; \\ 0, & \textit{otherwise}. \end{cases} \quad (7)$$

Based on the retrieved bits of a that for any $\Delta^{c_i^{i+l-1}}$, we can recover the all the bits of d except d_0 by following one of the above tag relations. Hence we can recover $K^{(1)}$ (except $K_0^{(1)}$). This would be done by inducing some $\Delta^{c_i^{i+l-1}}$ in the word c such that all the bits of c flip at least once. Instead of recovering all the bits of K^1, we can recover some of the bits of $K^{(1)}$ and use the brute-force approach for the remaining key bits. Our simulation is given in the Table 5. In the next section, we provide a discussion on the in-feasibility of Statistical Fault Analysis (SFA) on NORX.

No. of consecutive bits (l)	Recovered Bits (r)	Expected No. of $\Delta^{b[i,i+l-1]}$ (z_1)	Expected No. of $\Delta^{c[i,i+l-1]}$ (z_2)	Total No. of faults $2 \cdot (z_1 + z_2)$	Brute-force complexity
2	16	268	44	624	2^{64}
	24	268	84	704	2^{32}
	29	268	144	824	2^{12}
	31	268	200	936	2^{4}
3	16	196	28	448	2^{64}
	24	196	60	512	2^{32}
	29	196	100	592	2^{12}
	31	196	144	680	2^{4}
7	16	136	12	296	2^{64}
	24	136	28	328	2^{32}
	29	136	48	368	2^{12}
	31	136	88	448	2^{4}
19	16	128	4	264	2^{64}
	24	128	16	288	2^{32}
	29	128	32	320	2^{12}
	31	128	44	332	2^{4}
random	16	60	4	128	2^{64}
	24	60	8	136	2^{32}
	29	60	16	152	2^{12}
	31	60	32	184	2^{4}

Table 5: Expected number of consecutive faults for different l and r

5 Discussion

In the literature of physical attacks on cryptographic protocols, most powerful and effective fault analyses are DFA and SFA. In DFA [12], some input to be encrypted twice with a fault being induced in the last rounds of the second run, and then, the difference between the correct and faulty ciphertexts is used to retrieve the master key. On the other hand, SFA [19] requires only a collection of faulty ciphertexts to recover the correct key, but does not require correct and faulty ciphertext pairs. However, two conditions must need to satisfy for SFA: the inputs to the block cipher are different from each other, and the faulty ciphertexts are the direct outputs of the block cipher [16, 17]. More precisely, the attacker will collect biased faulty ciphertexts (distributed non uniformly), then compute backwards to the target byte corresponding to different key guesses and try to discard wrong key guesses that would lead to closely uniform measured distribution of the biased target byte. Thus, this type of attack will be very usefull where key whitening mechanism is used in any kind of iterated ciphers or in any kind of modes/AEAD schemes where underlying block cipher/permutation uses key whitening mechanism. But in NORX design, the master key only used in the initial phase and at the end of tag generation phase. Also the message is `XOR`ed with the rate part of the state to output the ciphertext and then apply the permutation on the state to process another message. So if we generate biased fault on a specific word/byte of a state just before the message processing at the last round of the permutaion F and collect the faulty ciphertexts by `XOR`ing the message with the rate part of the state, then there will be no way to guess the key such that after one round inverse we can check wheather that specific word/byte values behaves non uniform or not. Similarly, at the last diagonal round of the tag generation phase, collecting faulty tags by injecting biased faults on a specific byte/word inside the state does not help to apply SFA. Because, here we only know the capacity of state (tag) and the whole rate part is unknown to us. So by guessing the key byte/word corresponding to the faulty tag, we can not lead to the target byte to check its non uniform behavior by inverting one diagonal round. In our attack, we make use of Differential Fault Attacks (DFA) on NORX with $p = 2$, by creating a replay in the nonce respecting scenario. Also, we discuss several random fault models and recover the secret key bits by building a mathematical relation between the faulty and non-faulty tags. Moreover, in the precise control fault model, it will be quite easy to recover the secret key bits with a very few faults based on the mathematical relation between the faulty and non-faulty tags.

6 Conclusion

We show the first fault attack that uses both internal and classical differentials to mount a differential fault analysis on the nonce based authenticated cipher NORX. We show that faults introduced in NORX in parallel mode can be used to collide the internal state to produced an all-zero state and this can be used

to replay NORX despite being instantiated by different nonces and messages. Once replayed, we show how the internal state of NORX can be recovered using secondary faults. We use four different fault models. Under the random bit flip model, around 1384 faults are to be induced to reduce the key space from 2^{128} to 2^{32}. Whereas for the reduced key space of size 32, under random byte with known fault and random consecutive bit flip models, approximately 332 and 136 faults are to be induced respectively. Finally, for the random byte flip model, our analysis shows that, by injecting approximately 136 faults, we can recover the full key. Moreover, another drawback of this design is that with parallelism degree $p \geq 2$, we can always create a replay by inducing $(p-1)$ number of counter faults while having the same number of secondary faults. Hence our analysis can be directly applied to estimate the expected number of faults.

References

1. CAESAR Competition. `https://competitions.cr.yp.to/caesar.html`.
2. Farzaneh Abed, Christian Forler, and Stefan Lucks. General classification of the authenticated encryption schemes for the CAESAR competition. *Computer Science Review*, 22:13–26, 2016.
3. Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. 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*, pages 105–125, 2014.
4. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Analysis of NORX: investigating differential and rotational properties. In *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, pages 306–324, 2014.
5. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX: parallel and scalable AEAD. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, pages 19–36, 2014.
6. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX8 and NORX16: authenticated encryption for low-end systems. *IACR Cryptology ePrint Archive*, 2015:1154, 2015.
7. Jovanovic P. Neves S. Aumasson, J. NORX V1. `http://competitions.cr.yp.to/round1/norxv1.pdf`, 2015.
8. Jovanovic P. Neves S. Aumasson, J. NORX V2.0. `http://competitions.cr.yp.to/round2/norxv20.pdf`, 2016.
9. Meier W. Aumasson, J. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list, 2009. `http://aumasson.jp/data/papers/AM09.pdf`.
10. Nasour Bagheri, Tao Huang, Keting Jia, Florian Mendel, and Yu Sasaki. Cryptanalysis of reduced NORX. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 554–574, 2016.

11. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, pages 320–337, 2011.
12. Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, pages 487–496, 1992.
13. Alex Biryukov, Aleksei Udovenko, and Vesselin Velichkov. Analysis of the NORX core permutation. *IACR Cryptology ePrint Archive*, 2017:34, 2017.
14. Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard. Cryptanalysis of NORX v2.0. *IACR Trans. Symmetric Cryptol.*, 2017(1):156–174, 2017.
15. Sourav Das, Subhamoy Maitra, and Willi Meier. Higher order differential analysis of NORX. *IACR Cryptology ePrint Archive*, 2015:186, 2015.
16. Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Victor Lomné, and Florian Mendel. Statistical fault attacks on nonce-based authenticated encryption schemes. 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*, pages 369–395, 2016.
17. Christoph Dobraunig, Stefan Mangard, Florian Mendel, and Robert Primas. Fault attacks on nonce-based authenticated encryption: Application to keyak and ketje. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, pages 257–277, 2018.
18. Ashutosh Dhar Dwivedi, Milos Kloucek, Pawel Morawiecki, Ivica Nikolic, Josef Pieprzyk, and Sebastian Wójtowicz. Sat-based cryptanalysis of authenticated ciphers from the CAESAR competition. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, July 24-26, 2017.*, pages 237–246, 2017.
19. Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118, 2013.
20. Phillip Rogaway. Nonce-based symmetric encryption. In *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, pages 348–359, 2004.
21. Sheldon Ross. *A first course in probability.* Prentice Hall, New York,7th edition, 2005.
22. Dhiman Saha and Dipanwita Roy Chowdhury. Scope: On the side channel vulnerability of releasing unverified plaintexts. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 417–438, 2015.
23. Dhiman Saha and Dipanwita Roy Chowdhury. Encounter: On breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 581–601, 2016.
24. Dhiman Saha and Dipanwita Roy Chowdhury. Internal differential fault analysis of parallelizable ciphers in the counter-mode. *Journal of Cryptographic Engineering*, Nov 2017.

25. Dhiman Saha, Sukhendu Kuila, and Dipanwita Roy Chowdhury. Escape: Diagonal fault analysis of APE. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 197–216, 2014.

A Upadate of Words Inside the G Function

A.1 Bit relations of step (G5)-(G6)

Each update of the words a, b, c, d from step $(G5)$ to step $(G6)$ inside the G-function can be written in bit vectors as,

G5.

$$\begin{aligned}
&\Big(a^{'}_0, a^{'}_1, \cdots, a^{'}_{30}, a^{'}_{31}\Big)\\
&\leftarrow \Big((a_0 \oplus b_0, a_1 \oplus b_1, \cdots, a_{31} \oplus b_{31}) \oplus ((a_0 \wedge b_0, a_1 \wedge b_1, \cdots, a_{31} \wedge b_{31}) \ll 1)\Big)\\
&= \Big(a_0 \oplus b_0 \oplus (a_1 \wedge b_1), a_1 \oplus b_1 \oplus (a_2 \wedge b_2), \cdots, a_{30} \oplus b_{30} \oplus (a_{31} \wedge b_{31}), a_{31} \oplus b_{31}\Big).
\end{aligned}$$

G6.

$$\begin{aligned}
&\Big(d_0, \cdots, d_{14}, d_{15}, d_{16}, \cdots, d_{30}, d_{31}\Big)\\
&\leftarrow \Big((d_0 \oplus a^{'}_0, \cdots, d_{14} \oplus a^{'}_{14}, d_{15} \oplus a^{'}_{15}, d_{16} \oplus a^{'}_{16}, \cdots, d_{30} \oplus a^{'}_{30}, d_{31} \oplus a^{'}_{31}) \ggg 16\Big)\\
&= \Big((d_0 \oplus a_0 \oplus b_0 \oplus (a_1 \wedge b_1), \cdots, d_{14} \oplus a_{14} \oplus b_{14} \oplus (a_{15} \wedge b_{15})), d_{15} \oplus a_{15} \oplus b_{15} \oplus\\
&\quad (a_{16} \wedge b_{16}), d_{16} \oplus a_{16} \oplus b_{16} \oplus (a_{17} \wedge b_{17}), \cdots, d_{30} \oplus a_{30} \oplus b_{30} \oplus (a_{31} \wedge b_{31}), d_{31}\\
&\quad \oplus a_{31} \oplus b_{31}) \ggg 16\Big)\\
&= \Big(d_{16} \oplus a_{16} \oplus b_{16} \oplus (a_{17} \wedge b_{17}), \cdots, d_{30} \oplus a_{30} \oplus b_{30} \oplus (a_{31} \wedge b_{31}), d_{31} \oplus a_{31} \oplus b_{31},\\
&\quad d_0 \oplus a_0 \oplus b_0 \oplus (a_1 \wedge b_1), \cdots, d_{15} \oplus a_{15} \oplus b_{15} \oplus (a_{16} \wedge b_{16})\Big).
\end{aligned}$$

A.2 Bit relations of step (G3)-(G6)

Each update of the words a, b, c, d from step $(G3)$ to step $(G6)$ inside the G-function can be expressed in bit vectors as,

G3.

$$\begin{aligned}
&\Big(c^{'}_0, c^{'}_1, \cdots, c^{'}_{30}, c^{'}_{31}\Big)\\
&\leftarrow \Big(c_0 \oplus d_0, c_1 \oplus d_1, \cdots, c_{30} \oplus d_{30}, c_{31} \oplus d_{31}) \oplus ((c_0 \wedge d_0, c_1 \wedge d_1, \cdots, c_{30} \wedge d_{30},\\
&\quad c_{31} \wedge d_{31}) \ll 1\Big)\\
&= \Big(c_0 \oplus d_0, c_1 \oplus d_1, \cdots, c_{30} \oplus d_{30}, c_{31} \oplus d_{31}) \oplus (c_1 \wedge d_1, c_2 \wedge d_2, \cdots, c_{31} \wedge d_{31}, 0\Big)\\
&= \Big(c_0 \oplus d_0 \oplus (c_1 \wedge d_1), c_1 \oplus d_1 \oplus (c_2 \wedge d_2), \cdots, c_{30} \oplus d_{30} \oplus (c_{31} \wedge d_{31}), c_{31} \oplus d_{31}\Big).
\end{aligned}$$

G4.

$$\Big(b_0^{'}, \cdots, b_9^{'}, b_{10}^{'}, b_{11}^{'}, \cdots, b_{31}^{'}\Big)$$
$$\leftarrow \Big((b_0 \oplus c_0^{'}, \cdots, b_9 \oplus c_9^{'}, b_{10} \oplus c_{10}^{'}, b_{11} \oplus c_{11}^{'}, \cdots, b_{31} \oplus c_{31}^{'}) \ggg 11\Big)$$
$$= \Big(b_{21} \oplus c_{21} \oplus d_{21} \oplus (c_{22} \wedge d_{22}), \cdots, b_{30} \oplus c_{30} \oplus d_{30} \oplus (c_{31} \wedge d_{31}), b_{31} \oplus c_{31} \oplus d_{31},$$
$$b_0 \oplus c_0 \oplus d_0 \oplus (c_1 \wedge d_1), \cdots, b_{20} \oplus c_{20} \oplus d_{20} \oplus (c_{21} \wedge d_{21})\Big).$$

G5.

$$\Big(a_0^{'}, \cdots, a_9^{'}, a_{10}^{'}, a_{11}^{'}, \cdots, a_{30}^{'}, a_{31}^{'}\Big)$$
$$\leftarrow \Big(a_0 \oplus b_0^{'}, \cdots, a_9 \oplus b_9^{'}, a_{10} \oplus b_{10}^{'}, a_{11} \oplus b_{11}^{'} \cdots, a_{30} \oplus b_{30}^{'}, a_{31} \oplus b_{31}^{'}) \oplus$$
$$(a_0 \wedge b_0^{'}, \cdots, a_9 \wedge b_9^{'}, a_{10} \wedge b_{10}^{'}, a_{11} \wedge b_{11}^{'}, \cdots, a_{30} \wedge b_{30}^{'}, a_{31} \wedge b_{31}^{'}) \ll 1\Big)$$
$$= \Big(a_0 \oplus b_0^{'} \oplus (a_1 \wedge b_1^{'}), \cdots, a_9 \oplus b_9^{'} \oplus (a_{10} \wedge b_{10}^{'}), a_{10} \oplus b_{10}^{'} \oplus (a_{11} \wedge b_{11}^{'}),$$
$$a_{11} \oplus b_{11}^{'} \oplus (a_{12} \wedge b_{12}^{'}), \cdots, a_{30} \oplus b_{30}^{'} \oplus (a_{31} \wedge b_{31}^{'}), a_{31} \oplus b_{31}^{'}\Big).$$

G6.

$$\Big(d_0, \cdots, d_{14}, d_{15}, d_{16}, \cdots, d_{25}, d_{26}, d_{27}, \cdots, d_{31}\Big)$$
$$\leftarrow \Big((d_0 \oplus a_0^{'}, \cdots, d_{14} \oplus a_{14}^{'}, d_{15} \oplus a_{15}^{'}, d_{16} \oplus a_{16}^{'}, \cdots, d_{25} \oplus a_{25}^{'}, d_{26} \oplus a_{26}^{'},$$
$$d_{27} \oplus a_{27}^{'}, \cdots, d_{31} \oplus a_{31}^{'}) \ggg 16\Big)$$
$$= \Big(d_{16} \oplus a_{16}^{'}, \cdots, d_{30} \oplus a_{30}^{'}, d_{31} \oplus a_{31}^{'}, d_0 \oplus a_0^{'}, \cdots, d_9 \oplus a_9^{'}, d_{10} \oplus a_{10}^{'},$$
$$d_{11} \oplus a_{11}^{'}, \cdots, d_{15} \oplus a_{15}^{'}\Big)$$
$$= \Big(d_{16} \oplus a_{16} \oplus b_{16}^{'} \oplus (a_{17} \wedge b_{17}^{'}), \cdots, d_{30} \oplus a_{30} \oplus b_{30}^{'} \oplus (a_{31} \wedge b_{31}^{'}), d_{31} \oplus a_{31}$$
$$\oplus b_{31}^{'}, d_0 \oplus a_0 \oplus b_0^{'} \oplus (a_1 \wedge b_1^{'}), \cdots, d_9 \oplus a_9 \oplus b_9^{'} \oplus (a_{10} \wedge b_{10}^{'}), d_{10} \oplus a_{10} \oplus b_{10}^{'}$$
$$\oplus (a_{11} \wedge b_{11}^{'}), d_{11} \oplus a_{11} \oplus b_{11}^{'} \oplus (a_{12} \wedge b_{12}^{'}), \cdots, d_{15} \oplus a_{15} \oplus b_{15}^{'} \oplus (a_{16} \wedge b_{16}^{'})\Big).$$

B Random Bit Flip Fault Model

B.1 Bit Flip on Word b

Let $d^{\Delta^{b_i}}, 0 \leq i \leq 31$, denote the faulty values of the word d, due to Δ^{b_i}. Using Equation 1, the xor-ed relation of $d, d^{\Delta^{b_i}}$, including the corner cases are given as follows.

For any $\Delta^{b_i}, i \neq 0$, we have,

$$(d \oplus d^{\Delta^{b_i}})[j] = \begin{cases} a_i, \text{ if } j = i + 15; \\ 1, \text{ if } j = i + 16; \\ 0, \textit{ otherwise.} \end{cases}$$

For Δ^{b_0}, we have,

$$(d \oplus d^{\Delta^{b_0}})[j] = \begin{cases} 1, \text{ if } j = 16; \\ 0, \textit{ otherwise.} \end{cases}$$

B.2 Bit Flip on Word c

Let $d^{\Delta^{c_i}}, 0 \leq i \leq 31$, denote the faulty value of the word c, due to Δ^{c_i}. Using Equations 2 & 2, the xored relation of $d, d^{\Delta^{c_i}}$, including the corner cases are given as follows.

For $\Delta^{c_i}, i \neq 0, 21, 22$, we have,

$$(d \oplus d^{\Delta^{c_i}})[j] = \begin{cases} a_{i+10} \wedge d_i, \text{ if } j = i + 25; \\ a_{i+11} \oplus d_i, \text{ if } j = i + 26; \\ 1, \quad \text{if } j = i + 27; \\ 0, \quad \textit{otherwise.} \end{cases}$$

For $\Delta^{c_{21}}$, we have,

$$(d \oplus d^{\Delta^{c_{21}}})[j] = \begin{cases} a_{31} \wedge d_{21}, \text{ if } j = 14; \\ d_{21}, \quad \text{if } j = 15; \\ 1, \quad \text{if } j = 16; \\ 0, \quad \textit{otherwise.} \end{cases}$$

For $\Delta^{c_{22}}$, we have,

$$(d \oplus d^{\Delta^{c_{22}}})[j] = \begin{cases} a_1 \oplus d_{22}, \text{ if } j = 16; \\ 1, \quad \text{if } j = 17; \\ 0, \quad \textit{otherwise.} \end{cases}$$

For Δ^{c_0}, we have,

$$(d \oplus T_1^{\Delta^{c_0}})[j] = \begin{cases} a_{11}, \text{ if } j = 26 \\ 1, \quad \text{if } j = 27 \\ 0, \quad \textit{otherwise} \end{cases}$$

C Consecutive Bit Flip Fault Model

Here we will discus the generalization of the xored tags for any consecutive $l, 1 < l \geq 32$ bits flip in a word.

C.1 Consecutive Bits Flip on b

Let $\Delta^{b_{[i,i+l-1]}}$ denotes any l consecutive bits flip in the word b starting from the bit position i and $T^{\Delta^{b_{[i,i+l-1]}}}$ be the corresponding faulty tag. Also let $d^{\Delta^{b_{[i,i+l-1]}}}$ denotes the faulty value of the word d, due to $\Delta^{b_{[i,i+l-1]}}$. Using Equation 1, the xor-ed relation of $d, d^{\Delta^{b_{[i,i+l-1]}}}$, including all the corner cases, are given as follows.

Case 1: For any $\Delta^{b_{[i,i+l-1]}}$ such that $0 \notin \{i, \cdots, i+l-1\}$, we have,

$$d \oplus d\Delta^{b_{[i,i+l-1]}}[j] = \begin{cases} a_i, & \text{if } j = i+15; \\ \bar{a}_{i+j}, & \text{for } j \in \{i+16, \cdots, i+14+l\}; \\ 1, & \text{if } j = i+15+l; \\ 0, & \textit{otherwise}. \end{cases}$$

Case 2: $\Delta^{b_{[i,i+l-1]}}$ such that $0 \in \{i, \cdots, i+l-1\}$.

Sub-case 1: For $\Delta^{b_{[i,i+l-1]}}$, i.e., for $i = 0$, we have,

$$d \oplus d^{\Delta^{b_{[0,l-1]}}}[j] = \begin{cases} \bar{a}_j, & \text{for } j \in \{16, \cdots, 14+l\}; \\ 1, & \text{if } j = 15+l; \\ 0, & \textit{otherwise} \end{cases}$$

Sub-case 2: For any $\Delta^{b_{[i,i+l-1]}}$ such that $0 \in \{i+1, \cdots, i+l-2\}$, we have,

$$d \oplus d^{\Delta^{b_{[i,i+l-1]}}}[j] = \begin{cases} a_i, & \text{if } j = i+15; \\ \bar{a}_{i+12}, & \text{for } j \in \{i+16, \cdots, i+14+l\}/\{15\}; \\ 1, & \text{if } j = i+15+p = 15, 1 \le p \le l-1; \\ 1, & \text{if } j = i+15+l; \\ 0, & \textit{otherwise}. \end{cases}$$

C.2 Consecutive Bits Flip on c

Similarly, $\Delta^{c_{[i,i+l-1]}}$ denotes any l consecutive bits flip in the word c starting from the bit position i and $T^{\Delta^{c_{[i,i+l-1]}}}$ be the corresponding faulty tag. Also let $d^{\Delta^{c_{[i,i+l-1]}}}$ denotes the faulty value of the word d, due to $\Delta^{c_{[i,i+l-1]}}$. Using Equations 2 & 3, the xored relation of $d, d^{\Delta^{c_{[i,i+l-1]}}}$, including all the corner cases, are given as follows.

Case 1. For any $\Delta^{c_{[i,i+l-1]}}$ such that $0, 21, 22 \notin \{i, \cdots, i+l-1\}$, we have,

$$d \oplus d^{\Delta^{c_{[i,i+l-1]}}}[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25; \\ d_i \oplus (\bar{d}_{i+1} \wedge a_{i+11}), & \text{if } j = i+26; \\ \bar{d}_{i+j-1} \oplus (\bar{d}_{i+j} \wedge a_{i+12}), & \text{for } j \in \{2 \cdots, l-1\}; \\ \bar{d}_{i+l-1} \oplus a_{i+10+l}, & \text{if } j = i+25+l; \\ 1, & \text{if } j = i+26+l; \\ 0, & \textit{otherwise}. \end{cases}$$

Case 2. For any $\Delta^{c_{[i,i+l-1]}}$ such that $0 \in \{i, \cdots, i+l-1\}$ and $i = 0$, we have,

$$d \oplus d^{\Delta^{c_{[0,l-1]}}}[j] = \begin{cases} a_{11} \wedge \bar{d}_1, & \text{if } j = 26; \\ \bar{d}_j \oplus (a_{11+j} \wedge \bar{d}_{j+1}), & \text{for } j \in \{26+1, \cdots, 26+l-1\}; \\ \bar{d}_l \oplus a_{11+l}, & \text{if } j = 26+l; \\ 1, & \text{if } j = 27+l; \\ 0, & \textit{otherwise.} \end{cases}$$

Case 3. $\Delta^{c_{[i,i+l-1]}}$ such that $21, 22 \in \{i, \cdots, i+l-1\}$.

Sub-case 3.1. $21, 22 \in \{i, \cdots, i+l-1\}$ such that $i = 21, 22$.

If $i = 21$, then we have,

$$d \oplus d^{\Delta^{c_{[21,20+l]}}}[j] = \begin{cases} a_{31} \wedge d_{21}, & \text{if } j = 14; \\ d_{21}, & \text{if } j = 15; \\ \bar{d}_{21+j-1} \oplus (a_{31+j} \wedge d_{21+j}), & \text{for } j \in \{16, \cdots, 13+l\}; \\ \bar{d}_{21+l-1} \oplus a_{31+l}, & \text{if } j = 14+l; \\ 1, & \text{if } j = 15+l; \\ 0, & \textit{otherwise.} \end{cases}$$

If $i = 22$, then we have,

$$d \oplus d^{\Delta^{c_{[22,21+l]}}}[j] = \begin{cases} d_{22} \oplus (a_1 \wedge d_{23}), & \text{if } j = 16; \\ \bar{d}_{22+j} \oplus (a_{j+1} \wedge d_{23+j}), & \text{for } j \in \{17, \cdots, 15+l\}; \\ \bar{d}_{22+l} \oplus a_{l+1}, & \text{if } j = 16+l; \\ 1, & \text{if } j = 17+l; \\ 0, & \textit{otherwise.} \end{cases}$$

Sub-case 3.2. For $21, 22 \in \{i+1, \cdots, i+l-2\}$, we have,

$$d \oplus d^{\Delta^{c_{[i,i+l-1]}}}[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25; \\ d_i \oplus (a_{i+11} \wedge \bar{d}_{i+1}), & \text{if } j = i+26; \\ \bar{d}_{i+j-1} \oplus (a_{i+10+j} \wedge \bar{d}_{i+j}), & \text{for } j \in \{i+27, \cdots, i+24+l\}; \\ \bar{d}_{i+l-1} \oplus a_{i+10+l}, & \text{if } j = i+25+l; \\ \bar{d}_{21}, & \text{if } \exists s \in \{1, \cdots, l\} \ni j = i+25+s = 15; \\ 1, & \text{if } j = i+26+l; \\ 0, & \textit{otherwise.} \end{cases}$$

Sub-case 3.3. $21, 22 \in \{i, \cdots, i+l-1\}$ such that $i+l-1 = 21, 22$.

If $i+l-1 = 21$, then we have,

$$d \oplus d^{\Delta^{c_{[i,i+l-1]}}}[j] = \begin{cases} a_{i+10} \wedge d_i, & \text{if } j = i+25; \\ d_i \oplus (a_{i+11} \wedge \bar{d}_{i+1}), & \text{if } j = i+26; \\ \bar{d}_{i+j-1} \oplus (a_{i+10+j} \wedge \bar{d}_{i+j}), & \text{for } j \in \{i+27, \cdots, i+24+l\}; \\ d_{i+l-1}, & \text{if } j = i+25+l; \\ 1, & \text{if } j = i+26+l; \\ 0, & \textit{otherwise.} \end{cases}$$

If $i+l-1=22$, then we have,

$$d\oplus d^{\Delta^{c_{[i,i+l-1]}}}[j]=\begin{cases} a_{i+10}\wedge d_i, & \text{if } j=i+25;\\ d_i\oplus(a_{i+11}\wedge \bar{d}_{i+1}), & \text{if } j=i+26;\\ \bar{d}_{i+j-1}\oplus(a_{i+10+j}\wedge \bar{d}_{i+j}), & \text{for } j\in\{i+27,\cdots,i+23+l\};\\ \bar{d}_{i+l-2}, & \text{if } j=i+24+l=15;\\ \bar{d}_{i+l-1}\oplus a_{i+10+l}, & \text{if } j=i+25+l;\\ 1, & \text{if } j=i+26+l;\\ 0, & \textit{otherwise}. \end{cases}$$