Security Analysis of ASCON Cipher under Persistent Faults

Madhurima Das[1] and Bodhisatwa Mazumdar[2]

[1] Department of Mathematics, Indian Institute of Technology Indore, India
madhurimadasisme@gmail.com,
[2] Department of Computer Science and Engineering, Indian Institute of Technology Indore, India,
bodhisatwa@iiti.ac.in

[3]

Abstract. This work investigates persistent fault analysis on ASCON cipher that has been recently standardized by NIST USA for lightweight cryptography applications. In persistent fault, the fault once injected through RowHammer injection techniques, exists in the system during the entire encryption phase. In this work, we propose a model to mount persistent fault analysis (PFA) on ASCON cipher. In the finalization round of the ASCON cipher, we identify that the fault-injected S-Box operation in the permutation round, p^{12}, is vulnerable to leaking information about the secret key. The model can exist in two variants, a single instance of fault-injected S-Box out of 64 parallel S-Box invocations, and the same faulty S-Box iterated 64 times. The attack model demonstrates that any Spongent construction operating with authenticated encryption with associated data (AEAD) mode is vulnerable to persistent faults. In this work, we demonstrate the scenario of a single fault wherein the fault, once injected is persistent until the device is powered off. Using the proposed method, we successfully retrieve the 128-bit key in ASCON. Our experiments show that the minimum number and the maximum number of queries required are 63 plaintexts and 451 plaintexts, respectively. Moreover, we observe that the number of queries required to mount the attack depends on fault location in the S-box LUT as observed from the plots reporting the minimum number of queries and average number of queries for 100 key values.

1 Introduction

Cryptographic primitives are designed to be mathematically secure by establishing a number of cryptographic properties. Despite such properties, implementations of such primitives may leak information about the secret key, which greatly reduce the complexity of attacks. Attacks with such information leakage from physical implementations are referred to as *side-channel attacks*. In this class of attacks, fault attacks belong to a category of implementation attacks on embedded systems. Pertaining to the class of active attacks, fault attacks injects errors

[3] **Disclaimer**: A preliminary version of this work was presented as a work-in-progress paper at SPACE 2024. This work is supported from c3i Hub project from IIT Kanpur with Sanction number, IHUB-NTIHAC/2021/01/21.

in the operation of target device, it has been demonstrated on block ciphers, such as RSA, AES, LED, PRESENT, and PICCOLO. Fault analysis attacks are more often mounted in two phases, *fault injection* (FI) and *fault analysis*. The fault injection can be mounted through mechanisms, such as voltage glitching [1], clock glitching [2], focused laser beams [3], and electromagnetic pulses [4] during the execution of key-dependent operations in encryption algorithms. Remote faults have been mounted on graphic processing units (GPU) and other high-end processors using Rowhammer techniques and dynamic frequency voltage scaling techniques [5–7].

Fault attacks were first reported by Boneh et al. [8] on RSA-CRT implementations. In 1997, Biham and Shamir proposed differential fault analysis (DFA), demonstrating that DFA is applicable to any block cipher [9]. DFA being a transient fault model, has been shown to recover the full key in DES and AES ciphers. In transient fault, the fault appears for a very small time duration, typically within a clock cycle. The injected fault may corrupt few bits or bytes in the intermediate state. Depending on the fault injection mechanism, the fault distribution can be uniformly random or may have a statistically biased distribution. Algebraic fault analysis (AFA) exploit the algebraic structure of encryption algorithm in a differential setting of fault-free and faulty execution modes [10]. Some other fault analysis methods exploit statistical biases could be exploited in a differential case or with only faulty ciphertexts.

A large number of fault analysis methods were proposed in the transient fault model. However, some attacks consider permanent fault model arising from physical defects that damage the device permanently. In this model, almost all invocations of the target encryption algorithm will comprise this fault. In 2018, Zhang et al. [5] prposed another fault model that falls between transient and permanent fault model called persistent fault was proposed. This fault once injected in look-up table based implementation of a cryptographic primitive, such as Sbox, persists in the device for multiple encryptions until the device is rebooted. The persistent bit-flip fault in Sbox is injected using Rowhammer technique. Unlike DFA, the adversary in PFA, does not require to synchronize or time fault injection in multiple rounds during run time. In [5], once the fault is injected, encryption is performed under a given plaintext, the adversary captures the faulty ciphertext, and subsequently performs PFA to recover the key. However, in many variants of PFA, a prior knowledge of fault location and value is required. In the scenario of multiple faults, this analysis becomes computationally intensive.

In this work, to improve such criticality, we consider a chosen plaintext based PFA. As a case study to mount the proposed chosen plaintext based PFA, we consider ASCON cipher family which has been chosen by NIST USA as the lightweight cryptography standard in February 2023 [11,12]. The ASCON cipher family belongs to the class of authenticated encryption schemes, which protect the confidentiality and authentication of data by employing sponge construction. The sponge construction employs a duplex mode of operation, which absorbs the data and subsequently squeezes the data. As this construction does not

involve key scheduling, it performs well in terms of high-speed implementation and memory-constrained environment. To the best of our knowledge, this work is the first to address persistent fault analysis (PFA) on ASCON, which has recently become a NIST standard for lightweight cryptography. The contributions of the proposed work are as follows:

1. We propose Chosen Plaintext based Persistent Fault Analysis(CP-PFA) on ASCON [13].
2. We model the PFA attack on the finalization stage in ASCON cipher. The analysis demonstrates the 128-bit key recovery with 64 to 520 plaintext queries or more.
3. In the proposed CP-PFA, we mounted the attack on two models of the substitution layer. In the first model, only one of the 64 has been injected with a persistent fault. In the second model, multiple Sboxes in the substitution layer are injected with the same persistent fault.
4. In each of the above models, we consider two models of the adversary. In the strong adversary model, the adversary can inject fault at a targeted entry within the Sbox LUT. In the weaker model, the adversary injects the fault in the SBox LUT but does not have control over the entry location at which the fault is injected.

The organization of the paper is as follows. Section 2 introduces the notation that will be consistently used throughout the paper. We establish the background of ASCON Cipher and Persistent Fault Attack in section 3. Section 4 consists of a detailed discussion of the proposed attack. Section 5 concludes the paper.

2 Notations

The Table 1 lists the notations that are used throughout the paper.

3 Backround

In this section, we present the background for the proposed work.

3.1 ASCON Cipher

The ASCON cipher [11] belongs to a family of authenticated encryption algorithms for resource-constrained devices and high-end CPUs.

ASCON cipher suite has 4 variables namely ASCON-128, ASCON-128a, ASCON-80pq, ASCON-Hash, and ASCON-XOF. The first two differ in round numbers in the permutation. ASCON-80pq, which is designed for the post-quantum era, has increased key size. ASCON-HASH is hash function that builds upon the extendable output function ASCON-XOF.

Table 1: Notation Table

Symbol	**Description**
K	Secret key K of $k \leq 160$ bits.
K_0	First 64 bits of the key.
K_1	Last 64 bits of the key. $K = K_0 \| \| K_1$.
N, T, $T^{'}$	Nonce N, fault free tag T, faulty tag $T^{'}$ all of 128 bits
$T_{j,0}[i](T^{'}_{j,0}[i])$	The i^{th} bit of the first 64 bits of the fault-free (faulty) tag of the j^{th} query, $0 \leq j \leq 128$, $0 \leq i \leq 64$.
$T_{j,1}[i](T^{'}_{j,1}[i])$	The i^{th} bit of the last 64 bits of the fault-free (faulty) tag of the j^{th} query, $0 \leq j \leq 128$, $0 \leq i \leq 64$. $T = T_0 \| \| T_1 (T^{'} = T^{'}_0 \| \| T^{'}_1)$.
$K_0[i](K_1[i])$	The i^{th} bit of $K_0(K_1)$, $i \in \{0, 1, \ldots, 63\}$
P_i	64 bit plaintext block after padding
C_i	64 bit ciphertext block
A_i	64 bit block of associated data after padding
$\perp$	Error or verification of authenticated ciphertext failed
S	The 320-bit state S of the sponge function
S_r, S_c	The r-bit outer and c-bit inner part of the state S
p	Permutations
p_C, p_S, p_L	Constant-addition, substitution and linear layer of p
p_a, p_b	Permutations consisting of a, b rounds, respectively
$x \oplus y$	XOR of bitstrings x and y
$x_0, \ldots, x_4$	The five 64-bit words of the state S
$x_{j,i}, \ldots, x_{4,i}$	Bit i, $0 \leq i < 64$, of the word x_j, $0 \leq j \leq 4$, with $x_{j,0}$ the rightmost bit (LSB) of word x_j
$x \oplus y$	Bitwise XOR of 64-bit words, x and y
$x \odot y$	Bitwise AND of 64-bit words, x and y
$x \gg i$	Right-rotation (circular shift) by i bits of 64-bit word x
$SB_{j,i}(x_{m,i})$	The m^{th} bit of the output of the i^{th} S-Box among 64 S-Boxes in the j^{th} query. $0 \leq j \leq 128$, $0 \leq i \leq 64$, $0 \leq m \leq 4$.
$\sum_i x_i$	Linear function for state word x_i.
$\sum_i^{-1} x_i$	Inverse of linear function for state word x_i.
$\|\|,\|\|_{i=0}^{63}$	Concatenation, concatenating 64 bits.

Architecture of ASCON-128 The ASCON 128 cipher is an authenticated encryption with associated data (AEAD) algorithm, utilizing the MonkeyDuplex construction. It operates on a state of 320 bits. This state S is split into an outer part, S_r of $r = 64$ bits, and an inner part S_c of $c = 256$ bits, calculated as $c = 320 - r$. The 320-bit state S in ASCON is split into five 64-bit words x_i, $S = S_r \| S_c = x_0 \| x_1 \| x_2 \| x_3 \| x_4$. The initial 320-bit state of ASCON 128 is composed using the secret key K of k bits, nonce N of 128 bits, and IV of 64 bits.

$$IV_{k,r,a,b} \leftarrow k\|r\|a\|b\|0^{160-k} = 0x80400c0600000000, S \leftarrow IV_{k,r,a,b}\|K\|N \quad (1)$$

ASCON-128 is built around two 320-bit permutations denoted by $\mathrm{p^a}$ and $\mathrm{p^b}$ which primarily differ in their number of rounds, $a = 12$ and $b = 6$ as shown in Fig. 1. These permutations are composed of p_C, addition of constants layer;

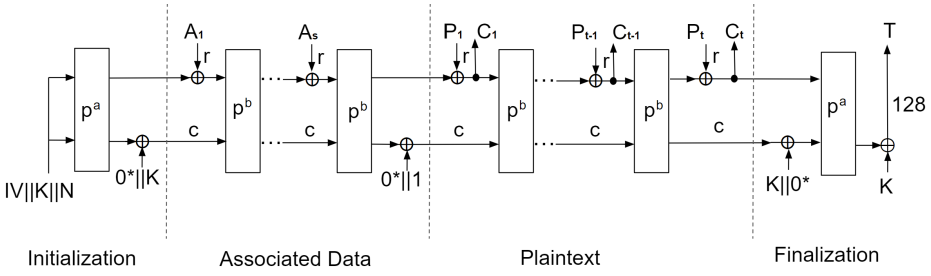

Fig. 1: Encryption Of ASCON-128

p_S, substitution layer; p_{L}, linear diffusion layer. The p_c adds a constant c_r to x_2 of the state S in each round of p_a and p_b, as $x_2 \leftarrow x_2 \oplus c_r$. The specific constants used in each round of the permutation are mentioned in [11]. In p_S, the state S is updated with 64 parallel instances of the 5-bit S-box $SB(x)$ to each bit-slice of the five registers, x_0, x_1, x_2, x_3, x_4. This operation is carried out using a bit-sliced approach, executing operations across entire 64-bit words. The lookup table of the S-Box is given in Table 2. The p_L layer diffuses each 64-bit register word x_0, x_1, x_2, x_3, x_4 with the linear functions for respective words, $\sum_i x_i$. The linear functions are defined as,

$$\sum_0(x_0) = x_0 \oplus (x_0 \gg 19) \oplus (x_0 \gg 28), \sum_1(x_1) = x_1 \oplus (x_1 \gg 61) \oplus (x_1 \gg 39)$$

$$\sum_2(x_2) = x_2 \oplus (x_2 \gg 1) \oplus (x_2 \gg 6), \sum_3(x_3) = x_3 \oplus (x_3 \gg 10) \oplus (x_3 \gg 17)$$

$$\sum_4(x_4) = x_4 \oplus (x_4 \gg 7) \oplus (x_4 \gg 41)$$

Table 2: Lookup Table for S-Box

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
$S(x)$	4	b	1f	14	1a	15	9	2	1b	5	8	12	1d	3	6	1c	1e	13	7	e	0	d	11	18	10	c	1	19	16	a	f	17

The encryption consists of four steps: Initialization, processing the associated data, processing the plaintext, and finalization. In the initialization phase, operation p^a is applied to the initial state, followed by a XOR with the secret key K.

$$S \leftarrow p^a(S) \oplus \left(0^{320-k} \| K\right) \tag{2}$$

The associated data A is processed in blocks of 64 bits. It appends a one-bit 1 and the smallest number of zeros required to make A a multiple of r bits, then split it into s number of blocks of r bits, $A_1 \| \ldots \| A_s$. Each block is XORed to S_r. Subsequently, p^b operation is applied to S.

$$S \leftarrow p^b\left(\left(S_r \oplus A_i\right) \| S_c\right), \quad 1 \leq i \leq s \tag{3}$$

A one-bit domain separation constant is then XORed with S_c. The plaintext P is processed into blocks of 64 bits. It appends a single 1 and the smallest number of 0s to the plaintext P so that the length of the plaintext is a multiple of r bits. The resulting plaintext is split into t blocks, each of 64 bits, $P_1 \| \ldots \| P_t$. In encryption, a padded plaintext block P_i, where $i \in \{1, 2, \ldots, t\}$, is XORed to the internal state S_r to generate the ciphertext block C_i. For every block except the endmost, the whole internal state S is transformed by permutation p^b with b rounds. The last ciphertext block C_t is then truncated to the length of the unpadded last plaintext block fragment, and the total length of the ciphertext C is exactly the same as for the original plaintext P. In the stage of finalization, the secret key K is XORed to the internal state S_c, and the state undergoes permutation p_a operation with a rounds.

$$S \leftarrow p^a\left(S \oplus \left(0^r \| K \| 0^{c-k}\right)\right) T \leftarrow \lceil S \rceil^{128} \oplus \lceil K \rceil^{128}$$

The tag T is formed by XORing the last 128 bits of the state and the 128 bits of the key K. The encryption algorithm outputs the tag T and the ciphertext, i.e., $\mathcal{E}_{k,r,a,b}(K, N, A, P) = (C, T)$. The decryption and verification algorithm, $\mathcal{D}_{k,r,a,b}(K, N, A, C, T) \in \{P, \perp\}$. is similar to $\mathcal{E}_{k,r,a,b}(K, N, A, P) = (C, T)$ with the plaintext blocks as input and ciphertext blocks are the output.

3.2 Persistent Fault Attack

Fault attack is an active attack which involves two distinct phases. In the first phase, the attacker disrupts the operation of the target device. This process is known as fault injection. In the second phase, the attacker analyzes the resulting faulty ciphertexts to extract the cryptographic key. This process is called fault analysis. In Persistent Fault Analysis (PFA), the fault will persist until the device reboots. Such a fault can be injected in an algorithmic constant stored in memory, such as an entry in S-Box LUT. Until the memory is refreshed, the fault remains present for subsequent encryptions.

The fault is injected in round computation and it persists across multiple computations. In the context of block ciphers, which are the primary target of this attack, these computations pertain to the round function. An encryption algorithm involves several invocations of the round functions. The injected fault

remains over several encryptions (and thus multiple round function calls), but the faulty value may not always be accessed. For instance, if the fault occurs in an S-box element, the round computation is affected only if the faulty S-box element is accessed. Otherwise, the injected fault does not influence that round computation. If the faulty value is not accessed during encryption, the resulting ciphertext will be correct, otherwise, it will be incorrect. The attacker analyzes the correct and incorrect ciphertexts to retrieve information about the key [5,14]. The attacker aims to minimize the number of plaintext queries while mounting PFA [13]. The complete Persistent Fault Attack comprises three stages:

1. **Fault Injection Stage**: The persistent fault is introduced before the first encryption.
2. **Encryption Stage**: The adversary waits for the victim to initiate the encryption process. The adversary then observes the produced ciphertexts, some of which are correct while others are incorrect due to the persistent fault.
3. **Fault Analysis Stage**: The adversary analyzes the correct and faulty ciphertexts to recover the secret key.

PFA techniques are employed to retrieve deeper round keys in SPN ciphers, where the final round key by itself is insufficient to deduce the entire master key [15].

4 Proposed Work

In this attack, two distinct fault models are considered. In both the models, a persistent fault is injected at a single location within the S-Box lookup table (LUT). The first model involves injecting the fault in only one of the 64 S-Boxes within p_S during the final round of the finalization phase in ASCON. The second model applies the fault across all 64 S-Boxes in p_S during the p^{12} permutation in the finalization phase.

4.1 Inverse of ASCON-128's LINEAR LAYER

Theorem (Rivest [16])
If n is a power of 2, v is an n-bit word, and r_1, r_2,. . ., r_k are distinct fixed integers modulo n, then the function $R(v)$ is invertible if and only if k is odd, where $R(v)$ is $R(v) = R(v; r_1, r_2, .., r_k) = (v_n <<< r_1) \oplus (v_n <<< r_2) \oplus ... \oplus (v_n <<< r_k)$.

The linear layer consists of XOR of right rotations of the 64-bit words, x_0, x_1, x_2, x_3, and x_4. As $k = 3$ for all five transformations, the linear layer of ASCON-128 is invertible. The rotations for the inverse of the linear layer are given in Table 3.

4.2 Fault Model

The adversary can inject the persistent fault by flipping bits in the S-Box LUT implementation through RowHammer injection techniques. The fault may vary depending on the adversarial setting. For instance, the $0x01$ entry of fault-free

Table 3: Rotations for Inverse Linear Layer

Permutation	Rotations	Size
Σ_0^{-1}	0 3 6 9 11 12 14 15 17 18 19 21 22 24 25 27 30 33 36 38 39 41 42 44 45 47 50 53 57 60 63	31
Σ_1^{-1}	0 1 2 3 4 8 11 13 14 16 19 21 23 24 25 27 28 29 30 35 39 43 44 45 47 48 51 53 54 55 57 60 61	33
Σ_2^{-1}	0 2 4 6 7 10 11 13 14 15 17 18 20 23 26 27 28 32 34 35 36 37 40 42 46 47 52 58 59 60 61 62 63	33
Σ_3^{-1}	1 2 4 6 7 9 12 17 18 21 22 23 24 26 27 28 29 31 32 33 35 36 37 40 42 44 47 48 49 53 58 61 63	33
Σ_4^{-1}	0 1 2 3 4 5 9 10 11 13 16 20 21 22 24 25 28 29 30 31 35 36 40 41 44 45 46 47 48 50 53 55 60 61 63	35

ASCON S-Box maps to $0x0b$, the bit must be flipped so that the $0x01$ entry in the faulty S-Box maps to $0x08$ as shown in Fig. 2. The lookup tables for both the fault-free S-Box, $S(x)$, and faulty S-Box, $S'(x)$ are depicted in Fig. 3. The injected fault satisfies the condition $S(x) \oplus S^{'}(x) = 0x03$. The flipping of last two S-Box output bits corrupts the state words x_3 and x_4, which aids our PFA method. The induced fault while satisfying this condition ensures that

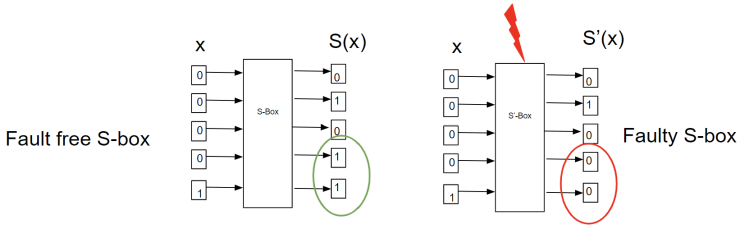

Fig. 2: Fault injection at $0x01$ such that $S(x) \oplus S^{'}(x) = 0x03$.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	$1a$	$1b$	$1c$	$1d$	$1e$	$1f$
$S(x)$	4	b	$1f$	14	$1a$	15	9	2	$1b$	5	8	12	$1d$	3	6	$1c$	$1e$	13	7	e	0	d	11	18	10	c	1	19	16	a	f	17

Lookup Table for fault free S-Box

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	$1a$	$1b$	$1c$	$1d$	$1e$	$1f$
$S(x)$	4	8	$1f$	14	$1a$	15	9	2	$1b$	5	8	12	$1d$	3	6	$1c$	$1e$	13	7	e	0	d	11	18	10	c	1	19	16	a	f	17

Lookup Table for faulty S-Box

Fig. 3: The lookup tables for the fault-free and faulty S-Box.

the faulty tag is generated by XOR operation of the last 128 bits of the state (which correspond to the state words, x_3 and x_4) with the 128 bits of the key K. So, the induced fault constraints the fault-free and faulty S-Box differential, $S(x) \oplus S'(x) \in \{0x03, 0x07, 0x0b, 0x0f, 0x13, 0x17, 0x1b, 0x1f\}$.

Our key recovery result comes under a strong caveat. In this work, our proposed CP-PFA assumes significant nonce-misuse, a feature for which ASCON designers have not proposed any security claim. Due to usage of 128-bit key

in initialization and finalization phase, it is not clear how a state recovery can result in a future key recovery or forgery attack.

4.3 Proposed CP-PFA on ASCON

In the proposed chosen plaintext based PFA when the attacker is granted access to the encryption oracle, it chooses a random plaintext P_1 of 64 bits, and creates a set of 64 plaintexts $P_{1,i}$, where $i \in \{0, 1, 2, \ldots, 63\}$. Each $P_{1,i}$ differs from P_1 in the i^{th} bit. This will ensure that every possible five bit word appears as input to all the 64 SBoxes in p_S in round 12 of permutation in the finalization phase. If these 64 queries do not retrieve the key then the number of queries is increased in a similar fashion. Plaintext sets, P_2 and P_3 can be randomly formed, each set creating 64 plaintexts. In total, there are 192 plaintexts, P_j, $j \in \{1, 2, \ldots, 192\}$ are input to the ASCON encryption. The partition of plaintext set, P_j is as follows.

$$P_j = \begin{cases} P_{1,j}, & \text{if } j < 65 \\ P_{2,(j-65)}, & \text{if } j < 129 \\ P_{3,(j-129)}, & \text{otherwise} \end{cases}$$

However, depending on the key value, there exist fault locations in S-Box for which the number of queries required is more than 192. With access to the ASCON encryption, the attacker inputs the plaintext and obtains the corresponding tag values, $T_j = T_{j,0}||T_{j,1}$, $j \in \{1, 2, \ldots, 192\}$. After capturing the set of fault-free tag values, the adversary injects the persistent bit-flip fault in the S-Box LUT implementation in substitution layer, p_S of p^{12} operation in the finalization phase.

In Fig. 4, consider the state word x_4 obtained after p^{12} in finalization phase marked as $tgin_1$. In j^{th} query, $j \in \{1, 2, \ldots, 192\}$, applying the inverse linear layer to x_4 yields $SB_{j,i}(x_{4,i})$ where $i \in \{0, \ldots, 63\}$ in the round 12 of permutation in the finalization step, i.e, $\sum_4^{-1} tgin_1 = ||_{i=0}^{63} SB_{j,i}(x_{4,i})$ where j denotes query number. From Fig. 4, we observe $tgin_1 = T_{j,1} \oplus K_1$ for the j^{th} query and hence we get,

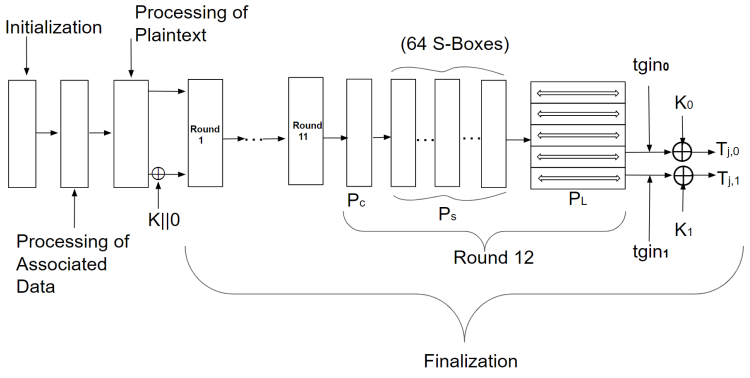

Fig. 4: ASCON encryption with the magnified last round of p^{12} in finalization phase.

$$\sum_{4}^{-1} tgin_1 = \sum_{4}^{-1}(T_{j,1} \oplus K_1) \Rightarrow ||_{i=0}^{63} SB_{j,i}(x_{4,i}) = \sum_{4}^{-1}(T_{j,1} \oplus K_1)$$

$$\Rightarrow SB_{j,i}(x_{4,i}) = \sum_{4}^{-1} T_{j,1}[i] \oplus \sum_{4}^{-1} K_1[i],\ \forall i \in \{0, \ldots, 63\}$$

$$\Rightarrow \sum_{4}^{-1} K_1[i] = \sum_{4}^{-1} T_{j,1}[i] \oplus SB_{j,i}(x_{4,i}),\ \forall i \in \{0, \ldots, 63\} \quad (4)$$

Similarly, applying the inverse linear operation to state word x_3 obtained after the 12^{th} round permutation in the finalization phase (denoted as $tgin_0$ in Fig. 4), yields $SB_{j,i}(x_{3,i})$ where $i \in \{0, \ldots, 63\}$, i.e, $\sum_{3}^{-1} tgin_0 = ||_{i=0}^{63} SB_{j,i}(x_{3,i})$, where j denotes the query number. Hence,

$$\sum_{3}^{-1} tgin_0 = \sum_{3}^{-1}(T_{j,0} \oplus K_0) \Rightarrow ||_{i=0}^{63} SB_{j,i}(x_{3,i}) = \sum_{3}^{-1}(T_{j,0} \oplus K_0)$$

$$\Rightarrow SB_{j,i}(x_{3,i}) = \sum_{3}^{-1} T_{j,0}[i] \oplus \sum_{3}^{-1} K_0[i],\ \forall i \in \{0, \ldots, 63\}$$

$$\Rightarrow \sum_{3}^{-1} K_0[i] = \sum_{3}^{-1} T_{j,0}[i] \oplus SB_{j,i}(x_{3,i}),\ \forall i \in \{0, \ldots, 63\} \quad (5)$$

Given that the tag T_j is captured, $\sum_{3}^{-1} K_0[i]$ and $\sum_{4}^{-1} K_1[i]$ can be computed if $SB_{j,i}(x_{3,i})$ and $SB_{j,i}(x_{4,i})$ are retrieved, respectively, for each $i \in \{0, 1, 2, \ldots, 63\}$. These values correspond to the third and fourth bits of the S-Box output for all the 64 S-Boxes in p_S. After $\sum_{3}^{-1} K_0[i]$ and $\sum_{4}^{-1} K_1[i]$ have been computed for each $i \in \{0, 1, 2, \ldots, 63\}$, the linear operation is applied to $\sum_{3}^{-1} K_0$ and $\sum_{4}^{-1} K_1$, i.e., $\sum_3(\sum_{3}^{-1} K_0)$ and $\sum_4(\sum_{4}^{-1} K_1)$, which yields K_0 and K_1, respectively.

To retrieve the keys K_0 and K_1, faults are injected into the S-Box LUT immediately before round 12 in the finalization phase. In the last round of p^{12} of finalization phase, the faulty S-Box is invoked in p_S as shown in Fig. 5. As a result, the adversary flips the last two bits of the S-Box output. In this single fault model, whenever the faulty S-Box is invoked, the output value, $SB'_{j,i}(x_{3,i}) \,||\, SB'_{j,i}(x_{4,i})$ is constrained to one of the four possible combinations, 00, 01, 10, or 11. The set of plaintexts, P_j, $j \in \{1, 2, \ldots, 192\}$ is input to the encryption function comprising the faulty S-Box during p^{12} in the finalization phase. This process yields the respective faulty tag, $T'_j = T'_{j,0} \,||\, T'_{j,1}$. Fig. 5 illustrates that out of the 64 S-Boxes in the p_S step of the p^{12} permutation during the finalization step of the encryption query, when the faulty S-Box LUT implementation is invoked at the i^{th} S-Box, the relation $\sum_{3}^{-1} tgin'_0 = ||_{i=0}^{63} SB'_{j,i}(x_{3,i})$ is obtained, where j represents the query number. Consequently, the following relation for K_0 is established.

$$\sum_{3}^{-1} tgin_0^{'} = \sum_{3}^{-1}(T_{j,0}^{'} \oplus K_0) \Rightarrow ||_{i=0}^{63} SB_{j,i}^{'}(x_{3,i}) = \sum_{3}^{-1}(T_{j,0}^{'} \oplus K_0)$$

$$\Rightarrow SB_{j,i}^{'}(x_{3,i}) = \sum_{3}^{-1} T_{j,0}^{'}[i] \oplus \sum_{3}^{-1} K_0[i] \ , \forall i \in \{0, \ldots, 63\}$$

$$\Rightarrow \sum_{3}^{-1} K_0[i] = \sum_{3}^{-1} T_{j,0}^{'}[i] \oplus SB_{j,i}^{'}(x_{3,i}) \ , \forall i \in \{0, \ldots, 63\} \tag{6}$$

Similarly, the following relation for K_1 is established:

$$\sum_{4}^{-1} tgin_1^{'} = \sum_{4}^{-1}(T_{j,1}^{'} \oplus K_1) \Rightarrow ||_{i=0}^{63} SB_{j,i}^{'}(x_{4,i}) = \sum_{4}^{-1}(T_{j,1}^{'} \oplus K_1)$$

$$\Rightarrow SB_{j,i}^{'}(x_{4,i}) = \sum_{4}^{-1} T_{j,1}^{'}[i] \oplus \sum_{4}^{-1} K_1[i] \ , \forall i \in \{0, \ldots, 63\}$$

$$\Rightarrow \sum_{4}^{-1} K_1[i] = \sum_{4}^{-1} T_{j,1}^{'}[i] \oplus SB_{j,i}^{'}(x_{4,i}) \ , \forall i \in \{0, \ldots, 63\} \tag{7}$$

The set of plaintexts considered by randomly selecting three plaintexts, and then creating 64 sets of plaintexts from each of them as discussed earlier, ensures that each possible entry of the faulty S-Box can be invoked in the p_S step of round p^{12} of permutation in the the finalization phase. So the fault value in the S-Box ensures that at least one of $T_{j,0}(T_{j,1})$ will differ from $T_{j,0}^{'}(T_{j,1}^{'})$, $j \in \{1, 2, \ldots, 192\}$.

4.4 Fault Model I

This fault model considers 64 parallel implementations of S-Box LUTs in the p_S step of round p^{12} in the finalization phase. Only one of the 64 S-Boxes is injected with persistent fault at a specific entry in the S-Box LUT.

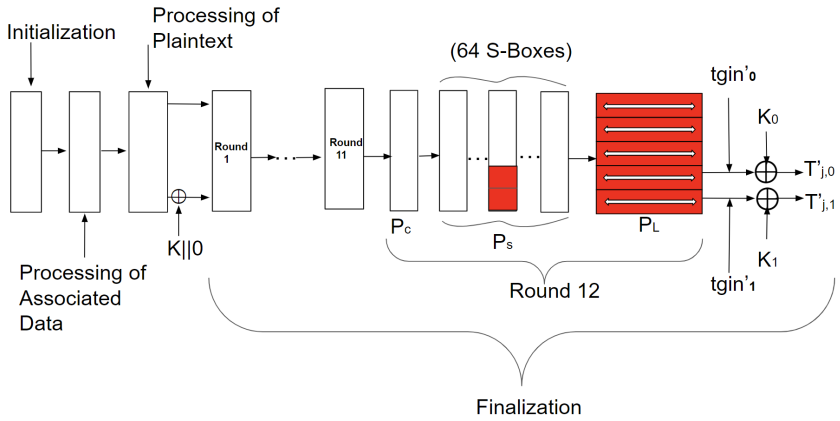

Fig. 5: Faulty S-Box LUT implementation invoked at the i^{th} S-Box in the last round of finalization stage during encryption.

To obtain the i^{th} bit of $\sum_3^{-1} K_0$ and $\sum_4^{-1} K_1$, $i \in \{0, 1, 2, \ldots, 63\}$, persistent fault is injected in the i^{th} S-Box out of the 64 parallel S-Boxes. Thereafter, we analyze the faulty tag values, $T'_j = T'_{j,0}||T'_{j,1}$ that differ from the fault-free tag values, $T_j = T_{j,0}||T_{j,1}$, $j \in \{1, 2, \ldots, 192\}$. Upon identifying such j values for both T_0 and T_1, it can be inferred that for that j^{th} query, the entry in the i^{th} S-Box corresponds to the LUT entry that produces the faulty output.
Claim: In the given model, the following equivalence holds:

(i) $T_{j,0} \neq T'_{j,0} \Leftrightarrow SB_{j,i}(x_{3,i}) \neq SB'_{j,i}(x_{3,i})$ (ii) $T_{j,1} \neq T'_{j,1} \Leftrightarrow SB_{j,i}(x_{4,i}) \neq SB'_{j,i}(x_{4,i})$.

Proof: In this model $T_{j,0} \neq T'_{j,0}$ implies $T_{j,0}[i] \neq T^{'}_{j,0}[i]$ as only the i^{th} S-Box out of all the 64 S-Boxes is faulty. Let us assume

$$SB_{j,i}(x_{3,i}) = SB'_{j,i}(x_{3,i})$$

Applying linear function $\sum_3$ on both sides we get,

$$SB_{j,i}(x_{3,i}) \oplus SB_{j,(i-10)mod64}(x_{3,(i-10)mod64}) \oplus SB_{j,(i-17)mod64}(x_{3,(i-17)mod64})$$
$$= SB^{'}_{j,i}(x_{3,i}) \oplus SB_{j,(i-10)mod64}(x_{3,(i-10)mod64}) \oplus SB_{j,(i-17)mod64}(x_{3,(i-17)mod64})$$

XORing K_0 on both sides,

$$SB_{j,i}(x_{3,i}) \oplus SB_{j,(i-10)mod64}(x_{3,(i-10)mod64}) \oplus SB_{j,(i-17)mod64}(x_{3,(i-17)mod64}) \oplus K_0[i]$$
$$= SB^{'}_{j,i}(x_{3,i}) \oplus SB_{j,(i-10)mod64}(x_{3,(i-10)mod64}) \oplus SB_{j,(i-17)mod64}(x_{3,(i-17)mod64}) \oplus K_0[i]$$
$$\Leftrightarrow T_{j,0}[i] = T^{'}_{j,0}$$

So, it is proved that $SB_{j,i}(x_{3,i}) = SB'_{j,i}(x_{3,i}) \Leftrightarrow T_{j,0} = T'_{j,0}$. It follows $SB_{j,i}(x_{3,i}) \neq SB'_{j,i}(x_{3,i}) \Leftrightarrow T_{j,0} \neq T'_{j,0}$. Hence proved. A similar approach can establish Claim (ii). Depending on the control of the adversary in injecting persistent fault in the S-Box LUT, we consider the following two cases.

Case I This analysis considers a strong adversary fault model, where the adversary can inject fault at a specific entry within the S-Box LUT. Since the location of the fault injection is controlled by the adversary, the outputs $SB^{'}_{j,i}(x_{4,i})$ and $SB^{'}_{j,i}(x_{3,i})$, corresponding to the last two bits of the faulty S-Box, are known. Given that $SB^{'}_{j,i}(x_{4,i})$ and $SB^{'}_{j,i}(x_{3,i})$ are known, equation 6 and equation 7 are used to retrieve $\sum_4^{-1} K_1[i]$ and $\sum_3^{-1} K_0[i]$ for each $i \in \{0, 1, 2, \ldots, 63\}$. Once $\sum_3^{-1} K_0$ and $\sum_4^{-1} K_1$ have been computed, the linear layer can be applied to these values, i.e., $\sum_3(\sum_3^{-1} K_0)$ and $\sum_4(\sum_4^{-1} K_1)$ which give K_0 and K_1, respectively. The steps are summarized in Algorithm 1.

Case II In this scenario, a weaker model is considered, where the adversary injects a fault into the S-Box LUT but does not have control over the specific entry location where the fault is introduced. The output values, $SB'_{j,i}(x_{3,i})||SB'_{j,i}(x_{4,i})$, are limited to one of four possible combinations: $0||0$, $0||1$, $1||0$, or $1||1$, regardless

Algorithm 1: Recovering 128 bits of Key by injecting persistent fault in only one location of S-Box LUT and it is called for only once among all the 64 S-Boxes in the Substitution layer in last round of p^{12} in finalization when the fault location is known.

Inputs: P_j , where $j \in \{1, 2, \ldots, 192\}$
Output: 128-bits of Key, $K = K_0||K_1$.
for $j = 0$ **to** 191 **do**
    Encryption queries using P_j and get the tags, $(T_j, C_j) \leftarrow \mathcal{E}_{k,r,a,b}(K, N, A, P_j)$.
end for
Inject persistent fault in one location of S-Box LUT by flipping the last two bits of the original S-Box output for that location.
for $bit = 0$ **to** 63 **do**
    for $j = 0$ **to** 191 **do**
        In finalization in the last round the p_S calls the faulty S-Box LUT for the bit^{th} S-Box among all the 64 S-Boxes, i.e
        $SB_{j,bit}(x_{3,bit})||SB_{j,bit}(x_{4,bit}) \neq SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit})$
        Make encryption queries using the same P_j and get the corresponding faulty $T'_j = T'_{j,0}||T_{j,1}$, ie. $(T'_j, C_j) \leftarrow \mathcal{E}'_{k,r,a,b}(K, N, A, P_j)$
        if $T'_{j,0} \neq T_{j,0}$ **then**
            $j_{0,bit} = j$. //capturing the value of j
        end if
        if $T'_{j,1} \neq T_{j,1}$ **then**
            $j_{1,bit} = j$. //capturing the value of j
        end if
        Compute $\sum_3^{-1} T'_{j_{0,bit},0}$ and $\sum_4^{-1} T'_{j_{1,bit},1}$.
        Compute $\sum_3^{-1} K_0[bit] = \sum_3^{-1} T'_{j_{0,bit},0}[bit] \oplus SB'_{j_{0,bit},bit}(x_{3,bit})$.
        Compute $\sum_4^{-1} K_1[bit] = \sum_4^{-1} T'_{j_{1,bit},1}[bit] \oplus SB'_{j_{1,bit},bit}(x_{4,bit})$.
    end for
end for
for $bit = 0$ **to** 63 **do**
    $\sum_3^{-1} K_0 = \sum_3^{-1} K_0 \oplus ((\sum_3^{-1} K_0[bit] \odot 1) << (64 - (bit + 1)))$
    $\sum_4^{-1} K_1 = \sum_4^{-1} K_1 \oplus ((\sum_4^{-1} K_1[bit] \odot 1) << (64 - (bit + 1)))$
end for
Compute $\sum_3(\sum_3^{-1} K_0)$ and $\sum_4(\sum_4^{-1} K_1)$
Return K_0 and K_1 and hence $K = K_0||K_1$

of the LUT entry location where fault is injected. Each of these four combinations is processed following the approach outlined in *Case I*, using Equation 6 and Equation 7, i.e., each possible combination is an instance of *Case I* which reduces the key search space to four key values. Once the four probable key values are obtained, the encryption algorithm is queried using each key value with the plaintexts, P_j, where $j \in \{1, 2, \ldots, 192\}$, and the corresponding tag values are received. The correct key is identified as the one for which all the newly received tag values match the fault-free tag values generated previously. The steps of this case are mentioned in Algorithm 2.

4.5 Fault Model II

In this model, multiple S-boxes in the substitution layer are injected with the same persistent fault. This fault model considers one S-Box LUT implementation, invoked 64 times in the substitution layer in p^{12} in finalization. So, when the fault is injected in the S-Box LUT in a specific entry that faulty S-Box is invoked for all the 64 S-Boxes. As a result, unlike Fault Model I, in Fault Model

Algorithm 2: Recovering 128 bits of Key by injecting persistent fault in only one location of S-Box LUT and it is called for only once among all the 64 S-Boxes in the Substitution layer in last round of p^{12} in finalization when the fault location is unknown.

Inputs: P_j , where $j \in \{1, 2, \ldots, 192\}$.
Output: 128-bits of Key, $K = K_0||K_1$.
Initialize Keytemp_{00}=$\text{Keytemp}_{00,0}||\text{Keytemp}_{00,1}$, Keytemp_{01}=$\text{Keytemp}_{01,0}||\text{Keytemp}_{01,1}$, Keytemp_{10}=$\text{Keytemp}_{10,0}||\text{Keytemp}_{10,1}$, Keytemp_{11}=$\text{Keytemp}_{11,0}||\text{Keytemp}_{11,1}$
Follow Algorithm 1 to get Keytemp_{00} if $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 0||0$.
Follow Algorithm 1 to get Keytemp_{01} if $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 0||1$.
Follow Algorithm 1 to get Keytemp_{10} if $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 1||0$.
Follow Algorithm 1 to get Keytemp_{11} if $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 1||1$.
for $j = 0$ **to** 191 **do**
    Make encryption queries using the plaintexts and get the corresponding tags.
    $(T_j^{00}, C_j^{00}) \leftarrow \mathcal{E}_{k,r,a,b}(\text{Ktemp}_{00}, N, A, P_j)$
    $(T_j^{01}, C_j^{01}) \leftarrow \mathcal{E}_{k,r,a,b}(\text{Ktemp}_{01}, N, A, P_j)$
    $(T_j^{10}, C_j^{10}) \leftarrow \mathcal{E}_{k,r,a,b}(\text{Ktemp}_{10}, N, A, P_j)$
    $(T_j^{11}, C_j^{11}) \leftarrow \mathcal{E}_{k,r,a,b}(\text{Ktemp}_{11}, N, A, P_j)$
end for
for $j = 0$ **to** 191 **do**
    if $T_j = T_j^{00}$ **then**
        count00++
    end if
    if $T_j = T_j^{01}$ **then**
        count01++
    end if
    if $T_j = T_j^{10}$ **then**
        count10++
    end if
    if $T_j = T_j^{11}$ **then**
        count11++
    end if
end for
if count00=192 **then**
    K=Keytemp_{00}
end if
if count01=192 **then**
    K=Keytemp_{01}
end if
if count10=192 **then**
    K=Keytemp_{10}
end if
if count11=192 **then**
    K=Keytemp_{11}
end if
Return $K = K_0||K_1$

II, more than one S-Box out of all the 64 S-Boxes can be faulty. From Fig. 4, for any j^{th} query, we have the following equations $\forall i \in \{1, 2, \ldots, 64\}$,

$$T_{j,1}[i] = K_1 \oplus SB_{j,i}(x_{4,i}) \oplus SB_{j,i}(x_{4,(i-7)mod64}) \oplus SB_{j,i}(x_{4,(i-41)mod64}), \quad (8)$$

$$T_{j,0}[i] = K_0 \oplus SB_{j,i}(x_{0,i}) \oplus SB_{j,i}(x_{0,(i-10)mod64}) \oplus SB_{j,i}(x_{0,(i-17)mod64}) \quad (9)$$

From Fig. 5, for any j^{th} query, we have the following equations $\forall i \in \{1, 2, \ldots, 64\}$,

$$T^{'}_{j,1}[i] = K_1 \oplus SB^{'}_{j,i}(x_{4,i}) \oplus SB^{'}_{j,i}(x_{4,(i-7)mod64}) \oplus SB^{'}_{j,i}(x_{4,(i-41)mod64}), \quad (10)$$

$$T^{;}_{j,0}[i] = K_0 \oplus SB^{'}_{j,i}(x_{0,i}) \oplus SB^{'}_{j,i}(x_{0,(i-10)mod64}) \oplus SB^{'}_{j,i}(x_{0,(i-17)mod64}) \quad (11)$$

Computing XOR of equation 8 and equation 10 for any j^{th} query and for any $i \in \{1, 2, \ldots, 64\}$, if $T_{j,1}[i] \neq T^{'}_{j,1}[i]$ then we have,

$$1 = (SB_{j,i}(x_{4,i}) \oplus SB^{'}_{j,i}(x_{4,i})) \oplus (SB_{j,i}(x_{4,(i-7) \mod 64}) \oplus SB^{'}_{j,i}(x_{4,(i-7) \mod 64}))$$
$$\oplus (SB_{j,i}(x_{4,(i-41) \mod 64}) \oplus SB^{'}_{j,i}(x_{4,(i-41) \mod 64})) \quad (12)$$

Similarly, computing XOR of equation 9 and equation 11 for any j^{th} query and for any $i \in \{1, 2, \ldots, 64\}$, if $T_{j,0}[i] \neq T^{'}_{j,0}[i]$ then we have,

$$1 = (SB_{j,i}(x_{3,i}) \oplus SB^{'}_{j,i}(x_{3,i})) \oplus (SB_{j,i}(x_{3,(i-10) \mod 64}) \oplus SB^{'}_{j,i}(x_{3,(i-10) \mod 64}))$$
$$\oplus (SB_{j,i}(x_{3,(i-17) \mod 64}) \oplus SB^{'}_{j,i}(x_{3,(i-17) \mod 64})) \quad (13)$$

From equation 12 we have two cases,

(i) $SB_{j,i}(x_{4,i}) \neq SB^{'}_{j,i}(x_{4,i})$ or $SB_{j,i}(x_{4,(i-7) \mod 64} \neq SB^{'}_{j,i}(x_{4,(i-7) \mod 64}$ or $SB_{j,i}(x_{4,(i-41) \mod 64}) \neq SB^{'}_{j,i}(x_{4,(i-41) \mod 64})$

(ii) $SB_{j,i}(x_{4,i}) \neq SB^{'}_{j,i}(x_{4,i})$ and $SB_{j,i}(x_{4,(i-7) \mod 64} \neq SB^{'}_{j,i}(x_{4,(i-7) \mod 64}$ and $SB_{j,i}(x_{4,(i-41) \mod 64}) \neq SB^{'}_{j,i}(x_{4,(i-41) \mod 64})$

Similarly, from equation 13 we have two cases,

(i) $SB_{j,i}(x_{3,i}) \neq SB^{'}_{j,i}(x_{3,i})$ or $SB_{j,i}(x_{3,(i-10) \mod 64} \neq SB^{'}_{j,i}(x_{3,(i-10) \mod 64}$ or $SB_{j,i}(x_{3,(i-17) \mod 64}) \neq SB^{'}_{j,i}(x_{3,(i-17) \mod 64})$

(ii) $SB_{j,i}(x_{3,i}) \neq SB^{'}_{j,i}(x_{3,i})$ and $SB_{j,i}(x_{3,(i-10) \mod 64} \neq SB^{'}_{j,i}(x_{3,(i-10) \mod 64}$ and $SB_{j,i}(x_{3,(i-17) \mod 64}) \neq SB^{'}_{j,i}(x_{3,(i-17) \mod 64})$

Thus more than one S-Box out of 64 S-Boxes in the substitution layer in p^{12} in finalization stage can be affected by the fault injected in the LUT of SBox.

To recover $\sum_4^{-1} K_1$ and $\sum_3^{-1} K_0$, the analysis focuses on comparing the faulty tag values, $T'_j = T'_{j,0}||T'_{j,1}$ with the corresponding fault-free tag values, $T_j = T_{j,0}||T_{j,1}$, where $j \in \{1, 2, \ldots, 192\}$. By identifying the discrepancies between these tags, it is inferred that for the j^{th} query, at least one of the 64 S-Boxes in the p_S of the finalization phase in round 12 has been affected by the fault. Subsequently, for each value of i, where $i \in \{1, 2, \ldots, 64\}$, the indices for which $\sum_3^{-1} T'_0[i] \neq \sum_3^{-1} T_0[i]$ and $\sum_4^{-1} T'_1[i] \neq \sum_4^{-1} T_1[i]$ satisfy, are determined. Using these identified indices, and from equation 6 and equation 7, the values of $\sum_3^{-1} K_0[i]$ and $\sum_4^{-1} K_1[i]$ are recovered for the respective values of i. We continue this process until we get $\sum_3^{-1} K_0[i]$ and $\sum_4^{-1} K_1[i]$, $\forall i \in \{1, 2, \ldots, 64\}$. Thereafter we compute $\sum_3(\sum_3^{-1} K_0)$ and $\sum_3(\sum_3^{-1} K_0)$ and thereby retrieve the entire key, $K = K_0||K_1$. There are two adversary cases. First, when the adversary is strong and the location of the fault injection is known, and second when

the fault location is not known. For each identified value of i, if the fault location is known, it corresponds to *Case I* of *Fault Model I*. On the contrary, if the fault location is not known to the adversary, it corresponds to the *Case II* of *Fault Model I*. Thus we retrieve the the key $K = K_0||K_1$. The steps are summarized in Algorithm 3 and Algorithm 4.

Algorithm 3: Recovering 128 bits of Key by injecting persistent fault in only one location of S-Box LUT and it is called for all the 64 S-Boxes in the Substitution layer in last round of p^{12} in finalization when the fault location is known.

Inputs: P_j , where $j \in \{1, 2, \ldots, 192\}$
Output: 128-bits of Key, $K = K_0||K_1$.
for $j = 0$ **to** 191 **do**
    Make encryption queries using the plaintexts and get the corresponding tags $T_j = T_{j,0}||T_{j,1}$, i.e. $(T_j, C_j) \leftarrow \mathcal{E}_{k,r,a,b}(K, N, A, P_j)$
end for
Inject fault in one location of S-Box LUT by flipping the last two bits of the original S-Box output for that location.
for $j = 0$ **to** 191 **do**
    In finalization in last round, p_S calls the faulty S-Box LUT for the bit^{th} S-Box among all the 64 S-Boxes, i.e $SB_{j,bit}(x_{3,bit})||SB_{j,bit}(x_{4,bit}) \neq SB^{'}_{j,bit}(x_{3,bit})||SB^{'}_{j,bit}(x_{4,bit})$
    Make encryption queries using the same P_j and get the corresponding faulty $T^{'}_j = T^{'}_{j,0}||T_{j,1}$, i.e. $(T^{'}_j, C_j) \leftarrow \mathcal{E}^{'}_{k,r,a,b}(K, N, A, P_j)$
end for
for $j = 0$ **to** 191 **do**
    if $T^{'}_{j,0} \neq T_{j,0}$ **then**
        for $bit = 0$ **to** 63 **do**
            if $\sum_3^{-1} T^{'}_{j,0}[bit] \neq \sum_3^{-1} T_{j,0}[bit]$ **then**
                Compute $\sum_3^{-1} K_0[bit] = \sum_3^{-1} T^{'}_{j,0}[bit] \oplus SB^{'}_{j,bit}(x_{3,bit})$.
                Compute $\sum_4^{-1} K_1[bit] = \sum_4^{-1} T^{'}_{j,1}[bit] \oplus SB^{'}_{j,bit}(x_{4,bit})$.
            end if
        end for
    end if
end for
for $bit = 0$ **to** 63 **do**
    $\sum_3^{-1} K_0 = \sum_3^{-1} K_0 \oplus ((\sum_3^{-1} K_0[bit] \odot 1) << (64 - (bit + 1)))$
    $\sum_4^{-1} K_1 = \sum_4^{-1} K_1 \oplus ((\sum_4^{-1} K_1[bit] \odot 1) << (64 - (bit + 1)))$
end for
Compute $\sum_3(\sum_3^{-1} K_0)$ and $\sum_4(\sum_4^{-1} K_1)$
Return K_0 and K_1 and hence $K = K_0||K_1$

5 Experimental Results

In the experimental results, we report the number of plaintext queries required to recover $K_0 \,||\, K_1$ after injecting persistent faults into the S-Box under different fault models. The experiments were conducted on a system with 12th Gen Intel Core i5-12500H processor operating at 2.50 GHz, and 16.0 GB of RAM (15.6 GB usable). The simulations for the proposed CP-PFA algorithms were conducted using a custom implementation of the ASCON-128 cipher written in C programming language. The implementation comprise modules for persistent fault injection at the S-Box level, fault analysis, and the key recovery.

Algorithm 4: Recovering 128 bits of Key by injecting persistent fault in only one location of S-Box LUT, and it is invoked for all the 64 S-Boxes in the Substitution layer in last round of p^{12} in finalization when the fault location is unknown.

1: **Inputs:** $P[i]$, where $i \in \{1, 2, \ldots, 192\}$
2: **Output:** 128-bits of Key.
3: **Initialize** Keytemp_{00}=$\text{Keytemp}_{00,0}||\text{Keytemp}_{00,1}$, Keytemp_{01}=$\text{Keytemp}_{01,0}||\text{Keytemp}_{01,1}$, Keytemp_{10}=$\text{Keytemp}_{10,0}||\text{Keytemp}_{10,1}$, Keytemp_{11}=$\text{Keytemp}_{11,0}||\text{Keytemp}_{11,1}$
4: Follow Algorithm 3 to receive Keytemp_{00} assuming $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 0||0$.
5: Follow Algorithm 3 to receive Keytemp_{01} assuming $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 0||1$.
6: Follow Algorithm 3 to receive Keytemp_{10} assuming $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 1||0$.
7: Follow Algorithm 3 to receive Keytemp_{11} assuming $SB'_{j,bit}(x_{3,bit})||SB'_{j,bit}(x_{4,bit}) = 1||1$.
8: Follow Algorithm 2 from STEP 8
9: **Return** K_0 and K_1 and hence $K = K_0||K_1$

The experiment started with a randomly selected 64-bit plaintext, P_1, from which a set of 64 additional plaintexts, $P_{1,i}$, $i \in \{0, 1, 2, \ldots, 63\}$, was generated. However, using only these 65 plaintexts was not sufficient to retrieve the entire key. As a result, we progressively increased the number of queries while following the same process to get the complete 128-bit key. The plots provided in Fig. 6 to Fig. 7 denote the number of queries required to retrieve the corresponding keys for all 32 possible fault locations in the S-Box LUT. We find that the number of queries required is the same for Algorithm 1 and Algorithm 2. Similarly, the number of queries required is the same for Algorithm 3 and Algorithm 4. All the experiments are performed for 100 random key values. However, each plot in Fig. 6 to Fig. 7 has five random keys and the corresponding number of queries to recover each of them. Algorithm 4 is used to generate the data for the plots. From the plots, we can see that there is no specific pattern for the number of queries. The key values of $K_0 \,||\, K_1$ that require less than 65 queries to get recover for a one fault location can be considered to be the weak keys. Fig. 8 has five random key vales and the corresponding number of queries to recover each of them when the fault model follows Algorithm 2. Fig. 7 shows that the maximum number of queries required to retrieve a key value is 451 and from Fig. 6 we get the least number of queries required which in this case is 63. From Fig. 9, which shows the average number of queries to mount PFA over 100 key samples for each fault location, we observe that fault location $0x1f$ can be the most vulnerable location for injecting the fault in general. Fig. 10 denotes the minimum number of queries against each fault location for 100 randomly chosen key values. From Fig. 10, we observe that fault location $0x01$ when injected with the persistent fault, has the maximum count of keys that requires minimum number of queries to recover the key. Thus S-Box LUT location, $0x01$ has increased vulnerability as compared to other locations.

6 Conclusion

The comprehensive analysis conducted on the ASCON-128 authenticated encryption scheme has revealed significant vulnerabilities, particularly in the con-

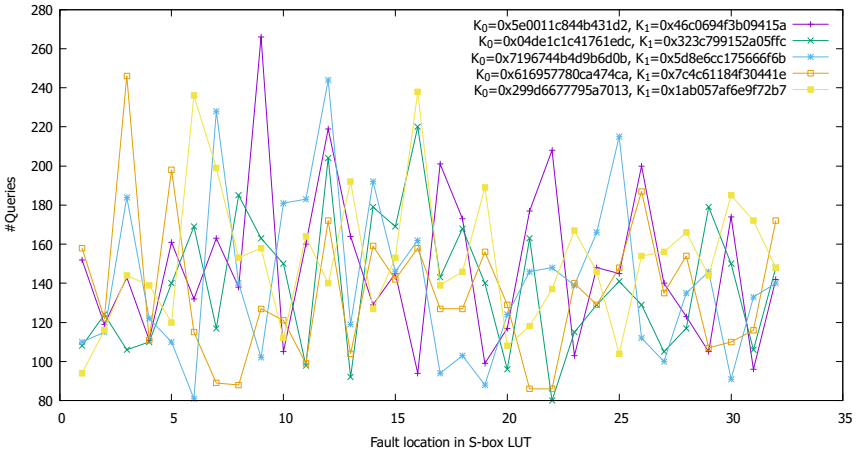

Fig. 6: Number of queries for a fault location at one of 32 locations of S-box LUT following Algorithm 4.

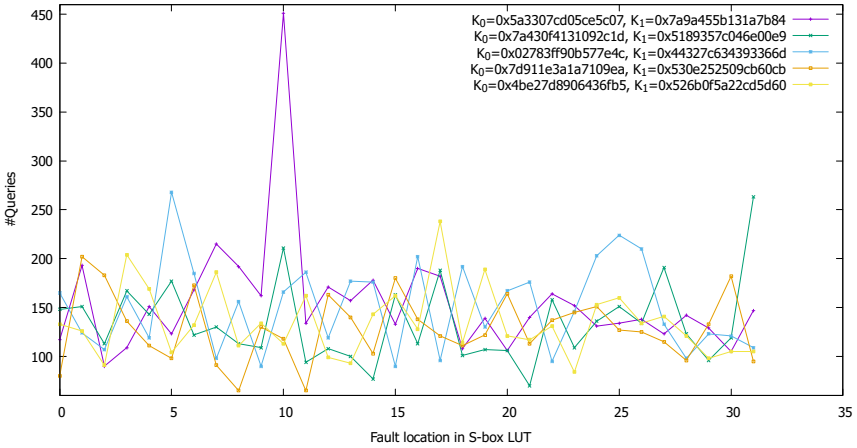

Fig. 7: Number of queries for a fault location at one of 32 locations of S-box LUT following Algorithm 4.

text of fault attacks. From the architecture it can be observed that the ASCON-128 design mixes key bits into the state via the S-box in each round, making it challenging for attackers to track differences between related inputs.

The experimental analysis focused on the behavior of the S-Box under fault injection attacks. By employing Persistence Fault Analysis, faults were strategically introduced into the S-Box during the finalization stage of the encryption process. This involved inducing flipping the last two bits in the S-Box entry in LUT in the last round, altering the output in a way that allowed the extraction of key bits.

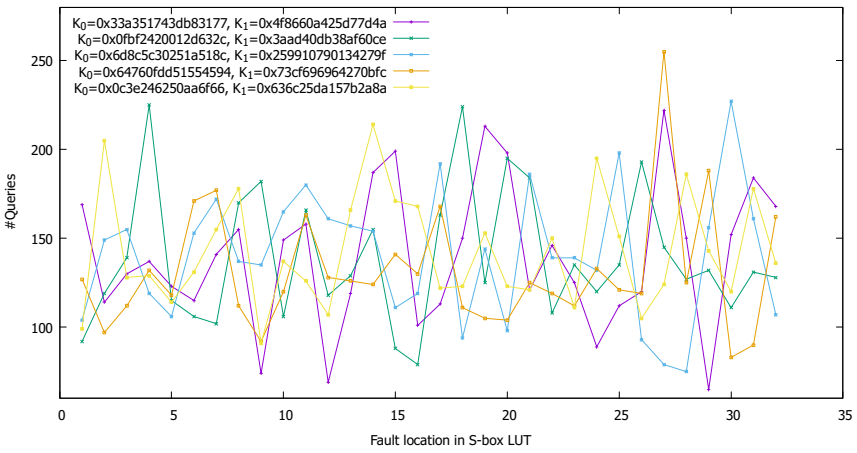

Fig. 8: Number of queries for a fault location at one of 32 locations of S-box LUT following Algorithm 2.

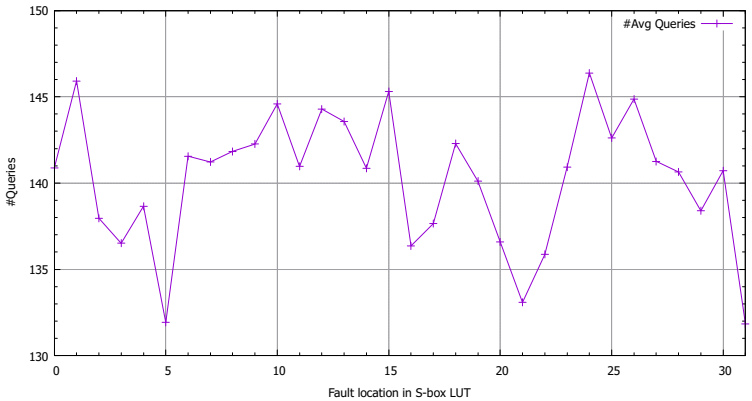

Fig. 9: Average number of queries to mount PFA over 100 random key samples for each fault location.

In the experimental setup, plaintexts were carefully selected and modified to ensure that every possible entry of the S-Box could be tested under fault conditions. A set of plaintexts ranging from 60-451 was used to query the encryption oracle and the resulting tag values were analyzed. By comparing the faulty and non-faulty tag values, the specific positions of the faults were identified, allowing for the recovery of the key bits. This approach enabled the successful extraction of the complete 128-bit key, demonstrating the feasibility and effectiveness of the fault attack. These vulnerabilities show the critical need for robust security measures in lightweight cryptographic solutions like ASCON-128. These attacks can be mounted on ASCON 128a and ASCON 80pq to extract the last 128 bits of the key as the only difference is in round number and key size respectively. Moreover, we believe that this work can be extended to the case of multiple

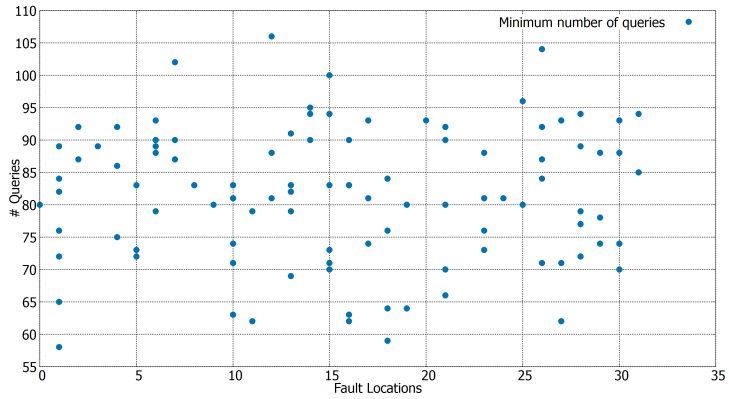

Fig. 10: Minimum number of queries required to mount PFA over 100 random key samples and the corresponding fault location in the S-Box LUT.

persistent faults in unknown locations of S-Box LUT, such as Statistical Persistent Fault Analysis (SPFA) and Practical Multiple Persistent Faults Analysis (PMPFA).

References

1. N. Selmane, S. Guilley, and J.-L. Danger, "Practical setup time violation attacks on aes," in *2008 Seventh European Dependable Computing Conference*, 2008, pp. 91–96.
2. M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria, "When clocks fail: on critical paths and clock faults," in *Proceedings of the 9th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Application*, ser. CARDIS'10. Berlin, Heidelberg: Springer-Verlag, 2010, p. 182–193. [Online]. Available: https://doi.org/10.1007/978-3-642-12510-2_13
3. S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 2–12.
4. A. Beckers, M. Kinugawa, Y. Hayashi, D. Fujimoto, J. Balasch, B. Gierlichs, and I. Verbauwhede, "Design considerations for EM pulse fault injection," in *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, ser. Lecture Notes in Computer Science, S. Belaïd and T. Güneysu, Eds., vol. 11833. Springer, 2019, pp. 176–192. [Online]. Available: https://doi.org/10.1007/978-3-030-42068-0_11
5. F. Zhang, X. Lou, X. Zhao, S. Bhasin, W. He, R. Ding, S. Qureshi, and K. Ren, "Persistent fault analysis on block ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 150–172, 2018.
6. K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1466–1482.
7. M. Sabbagh, Y. Fei, and D. Kaeli, "A novel gpu overdrive fault attack," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

8. D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1997, pp. 37–51.
9. E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Annual international cryptology conference*. Springer, 1997, pp. 513–525.
10. N. T. Courtois, K. Jackson, and D. Ware, "Fault-algebraic attacks on inner rounds of des," in *E-Smart'10 Proceedings: The Future of Digital Security Technologies*. Strategies Telecom and Multimedia, 2010.
11. C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2: Lightweight authenticated encryption and hashing," *Journal of Cryptology*, vol. 34, pp. 1–42, 2021.
12. M. S. Turan, M. S. Turan, K. McKay, D. Chang, L. E. Bassham, J. Kang, N. D. Waller, J. M. Kelsey, and D. Hong, *Status report on the final round of the NIST lightweight cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology, 2023.
13. F. Zhang, R. Huang, T. Feng, X. Gong, Y. Tao, K. Ren, X. Zhao, and S. Guo, "Efficient persistent fault analysis with small number of chosen plaintexts," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 519–542, 2023.
14. F. Zhang, Y. Zhang, H. Jiang, X. Zhu, S. Bhasin, X. Zhao, Z. Liu, D. Gu, and K. Ren, "Persistent fault attack in practice," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 172–195, 2020.
15. P. Joshi and B. Mazumdar, "Deep round key recovery attacks and countermeasure in persistent fault model: a case study on gift and klein," *Journal of Cryptographic Engineering*, pp. 1–23, 2024.
16. R. L. Rivest, "The invertibility of the xor of rotations of a binary word," *International Journal of Computer Mathematics*, vol. 88, no. 2, pp. 281–284, 2011.