Partial Differential Fault Analysis on Ascon

Yang Gao[1,2]

[1] Beijing National Research Center for Information Science and Technology (BNRist), School of Integrated Circuits, Tsinghua University, Beijing, China.
[2] Key Laboratory of Network Cryptography Technology of Henan, Information Engineering University, Zhengzhou, China.
gaoyang_1279@outlook.com

Abstract. Authenticated Encryption with Associated Data (AEAD) is a trend in applied cryptography because it combine confidentiality, integrity, and authentication into one algorithm and is more efficient than using block ciphers and hash functions separately. The Ascon algorithm, as the winner in both the CAESAR competition and the NIST LwC competition, will soon become the AEAD standard for protecting the Internet of Things and micro devices with limited computing resources. We propose a partial differential fault analysis (PDFA) technology for the Ascon algorithm, using stuck-at fault and random-nibble fault models respectively. Theoretically, after 9.9 full-round fault injections or 263 single nibble fault injections, 128-bit key can be completely recovered. In addition, we conducted the first discussion of this analysis method under different nonce configurations. In the Nonce-respect case, an average of 130 additional Tag queries are required to complete the guessing of the faulty tag, afterwards equating this case with the Nonce-misuse case. Subsequent experimental results proved the correctness of the theoretical model. Finally we discuss some countermeasures against proposed attacks, and we propose a new S-box that can be used to replace the existing S-box in ASCON to render PDFA ineffective.

Keywords: Authenticated Encryption · Ascon · CAESAR · NIST LWC competition · Differential fault analysis

1 Introduction

Due to the rapid evolution of emerging fields like the Internet of Things, wireless sensor networks, healthcare, and distributed control systems, interactive connections and wireless communications among highly constrained devices have become increasingly prevalent. However, conventional encryption algorithms, originally designed for PCs or server environments, are no longer suitable for devices with extremely limited computing resources. Consequently, there is an urgent demand for the development of ciphers that can operate within the constrained resources of simple electronic devices. To tackle this challenge, researchers are actively engaged in tailoring cryptographic primitives for these resource-constrained devices, forming a specialized area of cryptography known as lightweight ciphers (LWC).

In 2013, the National Institute of Standards and Technology (NIST) initiated a standardization research project on LWC, with the goal of identifying lightweight encryption algorithms suitable for restricted environments. These algorithms aim to significantly reduce implementation costs while ensuring the security of the system. The LWC competition [MBSTM16] commenced in July 2015, lasting for a total of 6 years. The project outlined seven key considerations in the design of LWCs: security strength, flexibility, low overhead for multiple functions, ciphertext expansion, susceptibility to side channel and fault attacks, constraints on the number of plaintext-ciphertext pairs, and resilience

against related-key attacks. During the initial release and evaluation stage of the algorithm, researchers conducted comprehensive analyses and attacks on candidate algorithms. This included differential analysis, impossible differential analysis, truncated differential analysis, cube analysis, algebraic cube analysis, side channel analysis, and more. In March 2021, the LWC competition announced the selection of 10 algorithms for the final standard algorithm set: Ascon, Elephant, GIFT-COFB, Grain128-AEAD, ISAP, Photon-Beetle, Romulus, Sparkle, TinyJambu, Xoodyak. After comprehensive consideration, NIST officially announced the selection of the Ascon family as the winning algorithm for the LWC competition in February 2023. Following this decision, NIST opted to standardize the Ascon family algorithms [NIS23a], aiming to enhance data security for IoT and micro devices with constrained computing resources [NIS23b]. It is anticipated that Ascon will soon be integrated into millions of authentication chips, RFID tags, and radio-controlled devices.

Ascon made its debut in the Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) competition [Ber16], submitted by Dobraunig et al. [DEMS14] in 2014. Subsequently, it underwent revisions and was released as Ascon v1.2 [DEMS21] in 2016. In the initial submission, the designers delineated the cipher's design principles and presented preliminary cryptanalysis results, notably focusing on the differential properties of the Ascon permutation [DEMS15]. Its exceptional performance in areas such as energy efficiency and security led to its recognition as a finalist in the CAESAR competition in March 2018. Moreover, the CAESAR committee endorsed it as the preferred choice for lightweight cryptographic applications in February 2019. Over the past decade since the introduction of the Ascon algorithm, extensive traditional cryptanalysis [Tez16, LZWW17, LDW17, YLW+23] and security assessments [CDN23, NSS23] have been conducted, contributing to a comprehensive understanding of its strengths and vulnerabilities. Simultaneously, researchers have made strides in exploring side-channel attacks against Ascon, yielding notable findings [GWDE17, RAD20b, RAD+20a, YKSH23].

Fault analysis, a crucial aspect of side-channel analysis, is a powerful and efficient method. It entails inducing faults in a device through physical manipulation (using methods such as clock glitches, voltage spikes, and laser beams) to generate computational errors [GT04]. These errors, along with valid outputs, are then utilized to compromise the security of systems, allowing for exploits like key recovery, e-wallet balance manipulation, unauthorized signature acceptance, and PIN code retrieval. Initially proposed by Boneh et al. [BDL97], fault analysis emerged as a means to analyze the RSA signature algorithm in CRT mode. Over time, researchers have developed and refined various fault analysis techniques, including differential fault analysis [BS97], collision fault analysis [Hem04], algebraic fault analysis [CJW10], statistical fault analysis [FJLT13], etc. These advancements have been applied to evaluate different AEAD cryptographic algorithms, including GIFT-COFB [LGH22], Grain-128AEAD [SOX+21], PHOTON-Beetle [JP22], TinyJambu [SAM24], among others, all of which were selected in the final round of LWC competition. However, the fault analysis against Ascon has yielded limited success. The proposed methods either fail to recover the complete key, rely on unrealistic attack assumptions, or necessitate extensive data collection and analysis. In summary, the complexity of existing fault analysis methods hinders their efficacy against Ascon.

In this study, we propose a novel differential fault analysis method called Partial Differential Fault Analysis (PDFA) for Ascon. Our analysis achieves complete key recovery by leveraging fault injection under two different fault models. In the first fault model, the attacker fixes one bit in the S-box and extracts the 4-bit part of the 5-bit S-box, transforming it into a 4-bit S-box commonly used in traditional LWCs for analysis. Subsequently, the complete 128-bit key is recovered through a classic random-nibble fault in the second fault model. A noteworthy advantage of our analysis lies in the minimal attention the attacker needs to pay to round permutations, focusing solely on the round where the tag

is generated. This significantly mitigates the challenges associated with fault injection and algorithmic analysis. Simultaneously, we incorporate the DFA statistical analysis strategy from traditional LWC to comprehensively quantify the attack complexity. This allows for a thorough evaluation of the efficiency and effectiveness of our proposed PDFA method in both theoretical and practical contexts.

Our contributions:

(1). We propose a novel analysis named Partial Differential Fault Analysis, which facilitates the complete recovery of the S-box intermediate state in the final permutation during Finalization. The lower complexity of this approach can be accurately calculated, leading to the full recovery of Key K.

(2). We make a detailed distinction and explanation of the analysis methods in the two cases of Nonce-respect and Nonce-misuse. The results show that only additional limited processing steps are needed to treat the two cases as equivalent.

(3). We analyze the differential properties of S-box in Ascon in detail, and distinguish the weak position, the moderate position and the strong position within 5 bits S-box.

(4). The partial differential distribution table was utilized to determine the instances where the S-box input could or could not be recovered after introducing N faults. A series of theorems were then employed to calculate the success rate of fault analysis and the expected number of fault injections necessary to recover the key successfully.

(5). We have successfully simulated 200,000 PDFA experiments on the computer, with the experimental results aligning closely with the theoretical analysis.

(6). As a countermeasure, we propose an alternative S-box that can preserve the original Ascon S-box's cryptographic properties.

Outline: In Section 2, we provide a detailed description of Authenticated Encryption with Associated Data (AEAD), an in-depth introduction to the Ascon cipher, and a consistent notation used throughout the paper. Section 3 introduces the idea of analyzing the Ascon algorithm, along with the two fault models used by PDFA. Section 4 discusses the characteristics of fault analysis in two scenarios: Nonce-misuse and Nonce-respect. We then outline the specific fault analysis implementation processes under two different nonce configurations. Section 5 delves into the probability of successful key recovery and the expected number of required fault injections under the two fault models. In Section 6, we validate the theoretical analysis through specific experiments. Section 7 compares PDFA with existing Ascon fault analysis methods, discusses the fault model's applicability, and proposes countermeasures or mitigation measures for attacks. Finally, Section 8 provides a concluding summary.

2 Background

2.1 Introduction to AEAD and Ascon algorithm

2.1.1 AEAD algorithm

Authenticated encryption (AE) combines symmetric encryption and authentication primitives into a single algorithm that are traditionally separated. Furthermore, in AEAD, associated data (AD) is authenticated but does not participate in encryption [Rog02]. This is particularly useful for communication protocols where header information must be authenticated but does not need to be encrypted. If AD or message authentication fails, a failure Tag is output, otherwise plaintext is released, which prevents chosen ciphertext attacks.

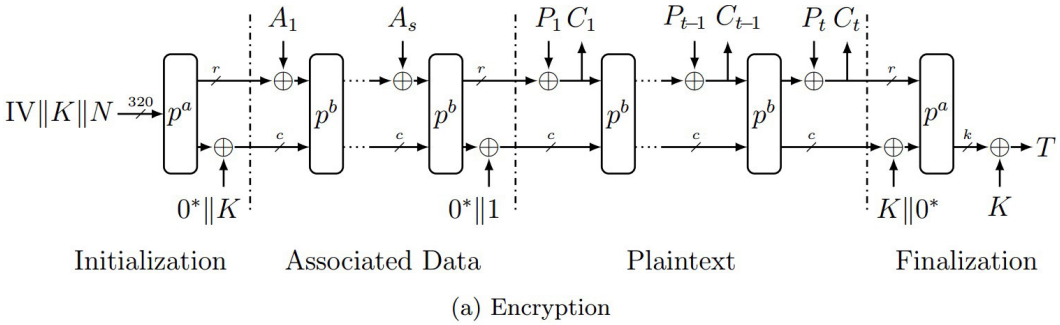
$$\mathcal{E} : \{0,1\}^k \times \{0,1\}^v \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^t$$

$$\mathcal{D} : \{0,1\}^k \times \{0,1\}^v \times \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^t \to \{0,1\}^* \cup \perp$$

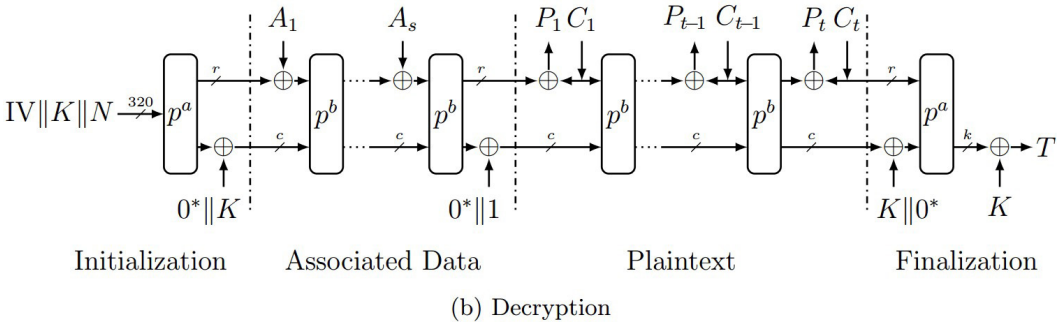
The above shows the input and output of AEAD encryption and decryption. Formally, set $k, v, t \geq 1$, let $K \in \{0,1\}^k$ denote Key, let $N \in \{0,1\}^v$ denote Nonce, $A \in \{0,1\}^*$ denote Associated data, $P \in \{0,1\}^*$ denote Plaintext, $T \in \{0,1\}^t$ denote Verification tag, $C \in \{0,1\}^*$ denote Ciphertext. AEAD algorithm is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, with a key generation algorithm $\mathcal{K}$ that returns a random K, encryption algorithm $\mathcal{E}_K(N, A, P)$ and decryption algorithm $\mathcal{D}_K(N, A, C, T)$, Where $\mathcal{E}$ outputs a (C, T) pair and $\mathcal{D}$ outputs plaintext P or an invalid symbol $\perp$ (if the label is invalid).

2.1.2 Ascon Specification

Ascon cipher is an AEAD algorithm based on sponge duplex structure [GJMG11]. In Ascon, the size of state S is 320 bits, in which rate bit $r = 64$ bits and capacity $c = 256$ bits. The initial state of Ascon consists of a 64-bit constant (called IV), followed by a 128-bit key K and a 128-bit random number N. In encryption, the 320-bit sponge state is divided into five 64-bit words, X_0, X_1, X_2, X_3 and X_4. The state S is defined as: $S = S_r||S_c = X_0||X_1||X_2||X_3||X_4$.



(a) Encryption



(b) Decryption

Figure 1: Structure of authenticated encryption in Ascon.

As shown in Figure 1, encryption process is divided into four stages: Initialization, Associated data processing, Plaintext processing, and Finalization. Ascon uses two permutation functions p^a and p^b, p^a for the Initialization and Finalization, and p^b for the Associated data and Plaintext processing. They iteratively apply the permutation p based on a Substitution-Permutation Network (SPN) structure, differing only in the number of rounds. Permutation p is the core element of Ascon and consists of three sub-transformations p_C, p_S and p_L, i.e. $p = p_L \circ p_S \circ p_C$.

p_C involves the xor operation of the 8-bit constant c_r with the word X_2, i.e. $X_2 = X_2 \oplus c_r$. The constant depends only on the index of the current permutation, so it is easy to calculate.

p_S is a nonlinear operation that represents a substitution layer. The layer consists of 64 5-bit S-boxes. In other words, the input 5 bits of any S(x) are taken from the 5 64-bit words X_0,...,X_4, that is, one bit is extracted from each word, where one bit from X_0

acts as the Most Significant Bit(MSB) for the S-box input, and one bit from X_4 acts as the Least Significant Bit(LSB) for the S-box input. The S-box used in Ascon is an affine equivalent of Keccak Sbox [KJA+18], and the specific values are shown in Table 1.

Table 1: Ascon 5-bit S-box.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(x)	4	11	31	20	26	21	9	2	27	5	8	18	29	3	6	28
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S(x)	30	19	7	14	0	13	17	24	16	12	1	25	22	10	15	23

p_L is a set of linear functions, Σ_i, that separately provide diffusion within each register word X_i. For each $i \in \{0, \ldots, 4\}$, $\Sigma_i(X_i)$ is built as an XOR of X_i and its two cyclic shift results, and the cyclic shift index is fixed and depends only on i. The function Σ_i is defined as follows.

$$\Sigma_0(X_0) = X_0 \oplus (X_0 >>> 19) \oplus (X_0 >>> 28)$$
$$\Sigma_1(X_1) = X_1 \oplus (X_1 >>> 61) \oplus (X_1 >>> 39)$$
$$\Sigma_2(X_2) = X_2 \oplus (X_2 >>> 1) \oplus (X_2 >>> 6)$$
$$\Sigma_3(X_3) = X_3 \oplus (X_3 >>> 10) \oplus (X_3 >>> 17)$$
$$\Sigma_4(X_4) = X_4 \oplus (X_4 >>> 7) \oplus (X_4 >>> 41)$$

The linear mapping of diffusion can be expressed in matrix form as:

$$\Sigma_i(X_i) = (L_i X_i) \mod 2, i = 0, 1, 2, 3, 4$$

Where $\oplus$ and $>>>$ denote the XOR and cyclic rightward operations, respectively. The vector representation of the state word is denoted as X_i, while L_i represents a 64×64 sparse matrix. Matrix multiplication is computed modulo 2. The initial line of L_3 signifies the presence of a non-zero element at the position corresponding to the cyclic shift value of the associated diffusion function. For instance, the first row of L_3 is a vector with elements 0, 10, and 17 set to 1, and the remaining elements set to 0. Similarly, the non-zero elements of the first row in L_4 are positioned at 0, 7, and 41. Each subsequent row of the matrix cyclically shifts the entire preceding row to the right by one unit.

Ascon introduced two versions: Ascon-128 and Ascon-96 [DEMS15], featuring distinct security levels and parameters. In this work, we focus on fault analysis in Ascon-128. The designers of Ascon mentioned the level of security achievable with these two versions and provided preliminary cryptanalysis results in their proposal submitted to the NIST LWC competition. Ascon-128 claims a security level of 128 bits in terms of plaintext confidentiality and plaintext/data/nonce integrity under three assumptions: Single use of each nonce, outputting plaintext only if tag is correct, and blocks encrypted with the same key are less than 2^{64}.

2.2 Notations

The important symbols and notations used in this paper are shown in Table 2.

3 Fault model used in PDFA for Ascon

DFA is an excellent strategy for analyzing traditional LWCs such as LBlock, Present, TWINE, etc. This efficacy stems from two primary factors. Firstly, owing to the prevalence of lightweight cryptographic devices, attackers can conveniently procure multiple ciphertexts

Table 2: Symbols and notations.

$\|V\|$	The number of elements contained in set V
K and T	128-bit key and tag of Ascon
K_0 and K_1	Two 64-bit words divided by K
T_0 and T_1(T_0^* and T_1^*)	Two 64-bit words divided by T after stuck-at fault before (after) random-nibble fault
Y_3 and Y_4(Y_3^* and Y_4^*)	Two 64-bit words output by Finalization after stuck-at fault before (after) random-nibble fault
X_3 and X_4(X_3^* and X_4^*)	The second least significant and least significant 64-bit words output by S-box layer after stuck-at fault before (after) random-nibble fault
S^*	4-bit S-box after stuck-at fault
α_{i_n}	4-bit fault value of the i^{th} S^* in n^{th} fault injection
β_{i_n}	4-bit output difference of the i^{th} S^* in n^{th} fault injection
β'	Highest 2 bits of 4-bit output difference in S^*
m_i	4-bit input value of the i^{th} S^*
Q_{i_n}	4-bit possible input set of the i^{th} S^* in n^{th} fault injection
$\lfloor x \rfloor_k$	Bitstring x truncated to the first (most significant) k bits
$\lceil x \rceil_k$	Bitstring x truncated to the last (least significant) k bits
G_n and H_n	The number of fault trails that can obtain a unique solution after n^{th} fault injection when α takes elements in G and H
T_n	The number of fault trails that obtain a unique solution after n^{th} fault injection
p_n ($\tilde{p}_n$)	The probability that a single S-box has a unique solution after (exactly) the n^{th} fault injection
E (E')	Expectation of the number of fault injections required to recover a single (every) S-box input value

for identical (plaintext, key) pairs. Conversely, traditional LWCs often utilize 4-bit S-boxes to strike a balance between efficiency and security. Nonetheless, compared to their non-lightweight counterparts (e.g., the 8-bit S-box in AES), these lightweight S-boxes may exhibit some shortcomings in terms of differential uniformity and nonlinearity. Specifically, for DFA, the differential distribution table of 4-bit S-boxes harbors more statistical properties, significantly facilitating the analysis of LWC algorithms.

However, since the advent of Ascon, DFA methods for it have been scant. To date, only one study has explored this avenue, employing a bit-flip fault model. This scarcity can be attributed to the divergence of Ascon from the aforementioned advantageous conditions for DFA. Firstly, Ascon adopts a 5-bit S-box, markedly elevating the complexity of differential distribution analysis. Moreover, unlike traditional LWCs, the AEAD algorithm necessitates a nonce as part of its input. Across various encryption requests, this nonce is typically reset and randomized. Consequently, even if an attacker targets identical plaintexts, keys, and Associated Data (AD), injecting the same fault at the same position during algorithm execution may not yield same faulty ciphertexts and tags in both encryption processes. Notably, in practical scenarios, where nonce configurations can vary between Nonce-respect and Nonce-misuse, cryptanalysis typically needs to consider both scenarios.

3.1 Key recovering in Ascon

Our objective is to analysis the entire Ascon-128 algorithm to obtain knowledge of K. Throughout the Ascon process, K participates in both Initialization and Finalization stages. Initially, we explore the potential for conducting DFA during Initialization. If a fault is injected during the initialization process to induce a difference, Ascon must undergo all the procedures of absorbing AD and generating ciphertext. Consequently, the fault diffusion obtained at each ciphertext output tends to be relatively uniform, rendering it challenging to clearly distinguish the characteristics correlated with K.

Hence, we contemplate performing DFA during Finalization. As indicated in Subsubsection 2.1.2, the output of p^a during Finalization consists of two 64-bit words, Y_3 and Y_4. We divide the 128-bit K into two 64-bit key words, denoted as $K = K_0||K_1$. Subsequently, Y_3 and Y_4 are XORed with two key words and then combined to derive the verification tag T, resulting in:

$$T_0 = K_0 \oplus Y_3, T_1 = K_1 \oplus Y_4, T = T_0 \oplus T_1$$

It is evident from the above equations that K can be directly recovered upon knowledge of the 64-bit words Y_3 and Y_4. Referring again to Subsubsection 2.1.2, we know that Y_3 and Y_4 can be directly obtained from the 64-bit words X_3 and X_4 output by S-box layer through the linear transformation layer p_L. Consequently, our goal shifts to recovering the 64-bit words X_3 and X_4.

Traditional DFA methods often commence from the final S-box layer involved in the algorithm and proceed with reverse-order analysis. Likewise, we explore injecting faults at the input of the S-box in the last round of p^a during the Finalization phase of the Ascon encryption process. As this represents the concluding nonlinear transformation of the entire cipher, such injections will introduce an unknown difference at the S-box output. According to the criterion of differential diffusion in cryptographic algorithms, nonlinear components alter difference values, whereas linear components primarily propagate and replicate difference values [Hey02]. Consequently, these differences maintain linearity after traversing the linear layer p_L and undergoing XOR operations with the key, ultimately manifesting as differences in the output tag T of the entire Ascon algorithm. In essence, following DFA implementation, we obtain several faulty tags, and the discrepancies between these tags can be translated into output differences of the S-box through inverse linear transformation. Assuming correct execution of the Ascon encryption process, the resulting tag is represented as:

$$T = T_0||T_1 = (K_0 \oplus Y_3)||(K_1 \oplus Y_4) = (K_0 \oplus \Sigma_3(X_3))||(K_1 \oplus \Sigma_4(X_4)).$$

The tag obtained after injecting a fault at the S-box input of the last round of permutation p in Finalization is

$$T^* = T_0^*||T_1^* = (K_0 \oplus Y_3^*)||(K_1 \oplus Y_4^*) = (K_0 \oplus \Sigma_3(X_3^*))||(K_1 \oplus \Sigma_4(X_4^*)),$$

we observe that $X_3 \oplus X_3^*$ and $X_4 \oplus X_4^*$ are the lowest and second-lowest 64-bit words. More precisely, we can discern the Least Significant Bit (LSB) and second LSB of the 5-bit XOR value of the correct and faulty outputs of each of the 64 S-boxes before and after fault injection. This information, coupled with an appropriate fault model, is adequate for deducing the input of the S-box layer and subsequently obtaining the 64-bit words X_3 and X_4.

3.2 Fault models in PDFA

Prior to embarking on the specific fault analysis of Ascon, we first introduce the two fault models utilized in the proposed PDFA method.

3.2.1 Stuck-at fault model

By naturally analyzing the differential distribution table of the 5-bit S-box (see in Appendix), we found that there are some properties. However, these properties alone proved insufficient for recovering the 128-bit key. To address this limitation, we explored the possibility of transforming the 5-bit S-box into a 4-bit S-box through a strategic manipulation. We hypothesized that such a transformation might reveal differential distribution properties analogous to those observed in other classic LWCs like MIBS [GWY+19b]. This forms the crux of the proposed PDFA method.

To address the first question regarding the optimal method for processing the 5-bit S-box to achieve the most effective analysis, we turned to the original Ascon's S-box. Specifically, we explored the idea of fixing the LSB of the S-box and studying the relationship between the remaining 4 bits and the output difference concerning the S-box input. Without loss of generality, we introduced a stuck-at-0 fault into the S-box, resulting in the reduced S-box denoted as S^*, as illustrated Table 3.

Table 3: S-box S^* after stuck-at-0 fault injected in LSB.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	4	31	26	9	27	8	29	6	30	7	0	17	16	1	22	15

In alignment with the characteristics of the Ascon algorithm, only the last two bits of the output difference β' could be obtained after fault injections during the final round of the S-box in the Finalization stage. Consequently, the classic differential equation $S^*(m) \oplus S^*(m \oplus \alpha) = \beta$ had to be transformed into the form $S^*(m) \oplus S^*(m \oplus \alpha) = \beta'$, where $\beta' = \beta \mod 4$. In simpler terms, only four possibilities existed for β', namely {00, 01, 10, 11}. The resulting differential distribution table of S^* after introducing a stuck-at-0 fault in the LSB is presented in Table 4.

Our objective is to determine the input value of S^* with as few fault injections as possible. Ideally, the best-case scenario is that the unique S^* input m can be obtained after just two fault injections. To formalize this objective, we introduce Definition 1 [GWY+19a].

Definition 1. Let $S^*(x)$ denote a "stuck-at" S-box in Ascon. For any fixed input difference $\alpha \in \mathbb{F}_2^4$ and output difference $\beta' \in \mathbb{F}_2^2$, we difine

$$\langle \alpha | \beta' \rangle = \left\{ m \in \mathbb{F}_2^4 : S^*(m) \oplus S^*(m \oplus \alpha) = \beta' \right\}.$$

We try to find two groups of $\langle \alpha_1 | \beta_1' \rangle$ and $\langle \alpha_2 | \beta_2' \rangle$ corresponding to the same S^* input m in differential distribution table, such that $|\langle \alpha_1 | \beta_1' \rangle \cap \langle \alpha_2 | \beta_2' \rangle| = 1$. At this time, it has $\langle \alpha_1 | \beta_1' \rangle \cap \langle \alpha_2 | \beta_2' \rangle = m$. However, upon examination of Table 4, it becomes evident that no such $\langle \alpha_1 | \beta_1' \rangle$ and $\langle \alpha_2 | \beta_2' \rangle$. In other words, we observe $\left| \langle \alpha_i | \beta_i' \rangle \cap \langle \alpha_j | \beta_j' \rangle \right| \geq 2$ for any i, j. Consequently, obtaining a unique S^* input m after two fault injections becomes unfeasible.

Given the input m of S^*, a fault value α must correspond to an output difference β'. For the sake of simplifying subsequent discussions, we introduce the following Definition 2 is given:

Definition 2. Let $\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n$ represent the sequence of fault injections, where the first injection is denoted as α_1, the second as α_2, and so forth, until the nth injection, α_n. This sequence is termed the fault trail from α_1 to α_n. The set $\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \rangle$ denotes the possible input values of S^* under this fault trail, defined as

$$\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \rangle = \langle \alpha_1 | \beta_1' \rangle \cap \langle \alpha_2 | \beta_2' \rangle \cap \cdots \cap \langle \alpha_n | \beta_n' \rangle.$$

Example 1. Consider the scenario where the input value of S^* is $m = 2$, we randomly inject the fault values α_1 and α_2. In this case, $\langle \alpha_1 \to \alpha_2 \rangle = \{0, 2\}$.

Table 4: Differential distribution table of S* after stuck-at-0 fault injected in LSB.

α	$\beta' = 00$	$\beta' = 01$	$\beta' = 10$	$\beta' = 11$
0	0,1,2,3,4,5, 6,7,8,9,10,11, 12,13,14,15	-	-	-
1	-	8,9,10,11, 12,13,14,15	-	0,1,2,3, 4,5,6,7
2	-	-	0,1,2,3,4,5, 6,7,8,9,10,11, 12,13,14,15	-
3	-	0,1,2,3, 4,5,6,7	-	8,9,10,11, 12,13,14,15
4	-	-	8,9,10,11, 12,13,14,15	0,1,2,3, 4,5,6,7
5	0,1,2,3, 4,5,6,7	-	-	8,9,10,11, 12,13,14,15
6	8,9,10,11, 12,13,14,15	0,1,2,3, 4,5,6,7	-	-
7	-	8,9,10,11, 12,13,14,15	0,1,2,3, 4,5,6,7	-
8	1,3,9,11	5,7,3,15	0,2,8,10	4,6,12,14
9	5,7,12,14	1,3,8,10	4,6,13,15	0,2,9,11
10	0,2,8,10	4,6,12,14	1,3,9,11	5,7,13,15
11	4,6,13,15	0,2,9,11	5,7,12,14	1,3,8,10
12	0,2,12,14	4,6,8,10	1,3,13,15	5,7,9,11
13	4,6,9,11	0,2,13,15	5,7,8,10	1,3,12,14
14	1,3,13,15	5,7,9,11	0,2,12,14	4,6,8,10
15	5,7,8,10	1,3,12,14	4,6,9,11	0,2,13,15

The aforementioned Example 1 highlights a straightforward yet important observation: when the actual input value of S* is 2, it becomes impossible to distinguish whether the input value is $m = 2$ or $m = 0$ through only two fault injections. Additional fault injections become necessary to resolve this ambiguity. From the perspective of an attacker, this situation is undesirable as they strive to minimize the number of injections while maximizing their understanding of the intermediate state of cryptographic systems. To address this issue, we must explore altering the positions of fixed bits in the original S-box in Ascon. Such changes will lead to a complete transformation of the stuck-at S-box. Moreover, for the same fixed position, "stuck-at-0" and "stuck-at-1" will also yield entirely different alterations to S*. Consequently, there exist $2 \times 5 = 10$ types of stuck-at S-boxes S*, as listed below:

Upon examining Table 5, it becomes evident that the significance of the five bit positions within Ascon's S-box for conducting a successful differential analysis varies notably. As per the outcomes of our program executions, for the complete 5-bit S-box, there is $\min\{|\langle\alpha_i \to \alpha_j\rangle|\} = 2, 0 \le a_i, a_j \le 31$(see in Appendix). From an attacker's perspective, it's apparent that the MSB x_0 and the second MSB x_1 are obviously bad positions. Because after installing the stuck-at fault here, $\min\{|\langle\alpha_i \to \alpha_j\rangle|\}$ will be increased, which means two fault injections reduce 16 possible S-box input values to 4 at most. On the other hand, the LSB x_4 and the second LSB x_3 are moderate positions. After inducting the stuck-at fault, $\min\{|\langle\alpha_i \to \alpha_j\rangle|\} = 2$. The effect is the same as not installing the stuck-at fault and this is obviously unacceptable. In essence, there is only one possible good position that we can determine, which is the middle (3rd) bit of the S-box, x_2. At this time,

Table 5: S* under different stuck-at models.

stuck-at model	The corresponding output when the remaining 4 bit input of S-box are 0 to 15	$\min\{\lvert\langle\alpha_i \to \alpha_j\rangle\rvert\}$
stuck-at-0 at x_4	4,31,26,9,27,8,29,6,30,7,0,17,16,1,22,15	2
stuck-at-1 at x_4	11,20,21,2,5,18,3,28,19,14,13,24,12,25,10,23	2
stuck-at-0 at x_3	4,11,26,21,27,5,29,3,30,19,0,13,16,12,22,10	2
stuck-at-1 at x_3	31,20,9,2,8,18,6,28,7,14,17,24,1,25,15,23	2
stuck-at-0 at x_2	4,11,31,20,27,5,8,18,30,19,7,14,16,12,1,25	1
stuck-at-1 at x_2	26,21,9,2,29,3,6,28,0,13,17,24,22,10,15,23	4
stuck-at-0 at x_1	4,11,31,20,26,21,9,2,30,19,7,14,0,13,17,24	4
stuck-at-1 at x_1	27,5,8,18,29,3,6,28,16,12,1,25,22,10,15,23	4
stuck-at-0 at x_0	4,11,31,20,26,21,9,2,27,5,8,18,29,3,6,28	4
stuck-at-1 at x_0	30,19,7,14,0,13,17,24,16,12,1,25,22,10,15,23	4

$\min\{\lvert\langle\alpha_i \to \alpha_j\rangle\rvert\} = 1$, suggesting that the S-box input can be conclusively determined by at least two fault injections. This observation aligns with the design specifics of Ascon. This observation aligns with the design specifics of Ascon. Notably, the Finalization output results from a linear transformation amalgamating the LSB and the second LSB of the 64 S-box outputs into respective words X_4 and X_3 respectively. This shows that for algorithm designers, two positions that are inconsequential for differential analysis are allowed to expose, while safeguarding the most vulnerable bit against potential attackers.

3.2.2 Random-nibble fault model

After determining the stuck-at position, it is necessary to decide whether to employ the stuck-at-0 or stuck-at-1 model. Both models partition all 32 outputs of the original 5-bit S-box into two distinct halves, and the values of two resultant S* boxes have no overlap. However, upon examination of Table 5, it became evident that $\min\{\lvert\langle\alpha_i \to \alpha_j\rangle\rvert\} = 1$ is in both cases. Consequently, we hypothesize that the differential characteristics of the two S* are identical. Subsequent verification through programming confirmed this conjecture, with the difference distribution tables of both models being identical, as illustrated in Table 6.

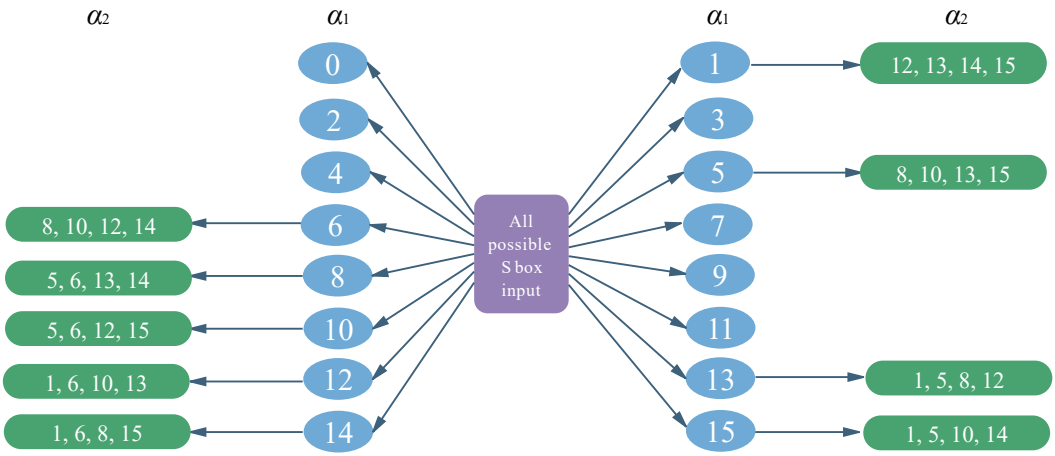


Figure 2: Trails to obtain unique S* input after 2 fault injections.

Once the stuck-at position is established, the 5-bit S-box can be transformed into a 4-bit S* for further analysis. In classic LWCs, the random-nibble fault model commonly

employed in PRESENT, TWINE, and LBlock can be readily applied. It's noteworthy to delineate that the random-nibble fault model manifests in two forms: fault injection in single S-box and fault injection in all S-boxes within a round simultaneously (named "full-round" S-box). With a total of 64 S-boxes operating in parallel during the last round of p^a in Finalization, injecting 64 random-nibble faults concurrently minimizes the number of injections required, albeit at the expense of increased complexity compared to the former model. In order to comprehensively gauge the attacker's capabilities across varying scenarios, this paper will explore the random-nibble fault model in both aforementioned situations.

Table 6: Differential distribution table of S* after stuck-at fault injected in x_2.

α	$\beta' = 00$	$\beta' = 01$	$\beta' = 10$	$\beta' = 11$
0	0,1,2,3,4,5, 6,7,8,9,10,11, 12,13,14,15	-	-	-
1	12,13,14,15	8,9,10,11	4,5,6,7	0,1,2,3
2	-	8,9,10,11, 12,13,14,15	-	0,1,2,3, 4,5,6,7
3	0,1,2,3, 8,9,10,11	4,5,6,7, 12,13,14,15	-	-
4	-	-	1,3,5,7, 8,10,12,14	0,2,4,6, 9,11,13,15
5	1,3,4,6	0,2,5,7	8,10,13,15	9,11,12,14
6	0,2,4,6	1,3,5,7	9,11,13,15	8,10,12,14
7	-	-	0,2,5,7, 9,11,12,14	1,3,4,6, 8,10,13,15
8	1,2,9,10	5,6,13,14	0,3,8,11	4,7,12,15
9	-	1,2,5,6, 8,11,12,15	-	0,3,4,7, 9,10,13,14
10	5,6,12,15	1,2,8,11	4,7,13,14	0,3,9,10
11	1,2,5,6, 9,10,13,14	-	0,3,4,7, 8,11,12,15	-
12	0,7,11,12	3,4,8,15	2,5,9,14	1,6,10,13
13	0,4,9,13	3,7,10,14	2,6,11,15	1,5,8,12
14	3,4,10,13	0,7,9,14	1,6,8,15	2,5,11,12
15	3,7,8,12	0,4,11,15	1,5,10,14	2,6,9,13

Firstly, we focus on examining the set of possible input values of S* after two fault injections, that is, the fault trail $\alpha_1 \to \alpha_2$. Employing programming tools, we systematically explore all fault trails outlined in Table 6, revealing a consistent outcome: regardless of the input value m of S*, the fault trails satisfying $\langle \alpha_1 \to \alpha_2 \rangle = m$ remain constant, which are illustrated in Figure 2.

Evidently, when $\alpha_1 = 0$, it is equivalent to an invalid empty fault. For the rest of the effective fault injection, as depicted in the preceding Figure 2, reveals that when $|\langle \alpha_1 \to \alpha_2 \rangle| = 1$, it implies $\alpha_1 \in G$, where $G = \{1, 5, 6, 8, 10, 12, 13, 14, 15\}$. Conversely, considering $H = \{2, 3, 4, 7, 9, 11\}$, if $\alpha_1 \in H$, irrespective of the value α_2 takes, $|\langle \alpha_1 \to \alpha_2 \rangle| \geq 2$. In such cases, obtaining the sole possible input of S* becomes unattainable through only two fault injections. Remarkably, upon revisiting Table 6, a correlation emerges: precisely when $\alpha_1 \in G$, there is $|\langle \alpha_1 \rangle| = 4$; for $\alpha_1 \in H$, $|\langle \alpha_1 \rangle| = 8$; and when $\alpha_1 = 0$, $|\langle \alpha_1 \rangle| = 16$. Consequently, it can be inferred that elements in G exhibit a robust ability to filter the input of S*, those in h demonstrate a weaker filtering capacity, while $\alpha_1 = 0$ (empty fault) lacks any ability to filter the input of S*. These characteristics form the basis of the

discussion on complexity.

For a single S^* box, there exist $16 \times 16 = 256$ situations in which two faults are injected. Referring to Figure 2, it becomes evident that 36 situations satisfy $|\langle \alpha_1 \to \alpha_2 \rangle| = 1$. Consequently, the probability of attaining the unique input value of S^* box after two fault injections is calculated as $\frac{9 \times 4}{256} = \frac{9}{64}$. For full-round S^* boxes, 64 random-nibble faults must be simultaneously injected. Since the 64 S^* boxes operate independently during the Finalization, it is apparent that the probability of acquiring the input value for all S^* by injecting two faults is $\left(\frac{9}{64}\right)^{64} \approx 2.99 \times 10^{-55}$. This minute value implies that the likelihood of uniquely determining the intermediate state of S^* boxes after two fault injections is exceedingly small, verging on negligible. However, this doesn't render simultaneous random-nibble fault injections in full-round S^* boxes futile. As the number of fault injections increases, more intriguing conclusions emerge. Subsequent to the two-time fault injection discussion, our focus will delve into a detailed analysis in Section 5.

4 PDFA process on Ascon

4.1 Discussion of different nonce configurations

As discussed in Subsubsection 2.1.1, the AEAD algorithm implemented by Ascon differs from traditional LWCs in that it requires a nonce as input. Typically, the nonce is reset and randomized for each encryption request, a practice known as Nonce-respect. However, in real-world scenarios, there exists the possibility of reusing the (key, nonce) pair, referred to as Nonce-misuse or Nonce-reuse. In such cases, where multiple plaintexts are encrypted using the same (key, nonce) pair, the state after Initialization remains constant throughout. This section delves into the operation of PDFA under these two nonce configurations.

It's important to note that regardless of whether Nonce-misuse or Nonce-respect is employed, a stuck-at fault must be injected at the S^* box input of the last round of p^a during the Finalization stage of the encryption process. The only variation lies in the timing of the injection of the random-nibble fault. Thus, for the ensuing discussion, the stuck-at fault is assumed to have been injected at the S^* box of the final round of p^a during Finalization, whether in encryption or decryption processes.

4.1.1 Nonce-misuse

In this scenario, the attacker is assured of obtaining both the ciphertext and tag for the same plaintext, key, nonce, and AD. Put differently, prior to tag generation in Finalization, the internal state of Ascon remains identical across consecutive runs. This satisfies the prerequisites for executing PDFA. The attacker's task then involves injecting the fault models outlined in Subsection 3.2 into the encryption process during repeated executions of the cryptosystem and collecting multiple faulty tags. Drawing from the pertinent details in Subsection 3.1, the S^* box input in the last round of permutation p^a during Finalization can be obtained, thereby facilitating the recovery of K.

4.1.2 Nonce-respect

Performing DFA in this scenario poses significant challenges due to the inability of the attacker to observe both correct and faulty outputs for the same plaintext. Nevertheless, the AEAD algorithm necessitates the use of a nonce during the decryption process of a given encryption request. It is imperative to ensure that this nonce matches the one used in the corresponding encryption request; otherwise, the tag verification will fail. Consequently, a pair of encryption and decryption operations can be viewed as a natural replay with a fixed nonce. Hence, we propose injecting random-nibble faults into the

decryption process, which constitutes the fundamental condition for completing DFA in a Nonce-respect scenario.

As elucidated in Subsection 3.1, the primary objective of our proposed PDFA is to ascertain the input value of the final round of the S^* and subsequently recover the key. These input values are determined by solving a set of differential equations. The initial information required by the attacker includes the input and output differences. In the context of the random-nibble fault model, the fault value represents the input difference. Thus, solving the differential equations hinges upon acquiring knowledge of the output difference.

Injecting random-nibble faults in Finalization of decryption leads to a change in the resulting tag, denoted as T^*. If T^* deviates from the tag T obtained during the encryption process without fault injection, the verification process fails. Our purpose is to manipulate the decryption input to $K \times N \times AD \times C \times T^*$ to successfully pass the verification and deduce the output difference $\beta' = T^* \oplus T$. We focus on the scenario of injecting nibble faults into a single S^* box, resulting in four possible output differences: $\beta' = \{00, 01, 10, 11\}$. Here, $K = K_0||K_1$ represents the 128-bit encryption key, $T = T_0||T_1$ denotes the 128-bit tag output during encryption, and $T^* = T_0^*||T_1^*$ signifies the 128-bit label computed during decryption. And $T_0(T_1)$ represents the XOR result of the 64-bit word $Y_3(Y_4)$ output by permutation p^a and the 64-bit key $K_0(K_1)$ in Finalization. The detailed results of the discussion are outlined below.

1. $\beta' = \mathbf{00}$. The output difference caused by injecting a fault at S^* box has last two bits as 0, implying no variation in the components involved in tag calculation. It implies $T^* = T$, signifying, $T_0^* = T_0$, $T_1^* = T_1$.

2. $\beta' = \mathbf{01}$. The second LSB in output difference of S^* remains unchanged, while LSB is 1. This suggests a change in the 64-bit word X_3 output by S^*, with X_4 remaining unaffected. Without loss of generality, let's assume fault injection into the first S^* box, S_0^*. This leads to

$$\Delta X_3 = X_3 \oplus X_3^* = 100\ldots00.$$

Utilizing the properties of the linear transformation layer p_L in permutation p, we derive

$$\Delta Y_3 = Y_3 \oplus Y_3^* = \Sigma_3(X_3) \oplus \Sigma_3(X_3^*) = \Sigma_3(X_3 \oplus X_3^*) = L_3 \cdot (X_3 \oplus X_3^*) \mod 2$$

Consequently, the difference in the 64-bit word Y_3 after the linear transformation is $\Delta Y_3 = 100\ldots100\ldots100\ldots0$($1st$, $11th$ and $18th$ bits are 1, others are 0). As Y_3 is directly XORed with the key word K_0 to obtain T_0, i.e., $T_0^* = K_0 \oplus Y_3^*$ and $T_0 = K_0 \oplus Y_3$, it follows that

$$T_0^* = T_0 \oplus Y_3 \oplus Y_3^* = T_0 \oplus 100\ldots100\ldots100\ldots0,\ T_1^* = T_1$$

3. $\beta' = \mathbf{10}$. The LSB in output difference of S^* remains unchanged, while the second LSB is 1. This indicates a change in the 64-bit word X_4 output by S^*, with X_3 remaining unchanged. Without loss of generality, assuming fault injection into the first S^* box, S_0^*, we have

$$\Delta X_4 = X_4 \oplus X_4^* = 100\ldots00.$$

Following the properties of the linear transformation layer p_L in permutation p, we obtain

$$\Delta Y_4 = Y_4 \oplus Y_4^* = \Sigma_4(X_4) \oplus \Sigma_4(X_4^*) = \Sigma_4(X_4 \oplus X_4^*) = L_4 \cdot (X_4 \oplus X_4^*) \mod 2$$

Hence, the difference in the 64-bit word Y_4 following the linear transformation can be represented as $\Delta Y_4 = 100\ldots100\ldots100\ldots0$($1st$, $8th$ and $42th$ bits are 1, others are 0). As Y_4 is directly XORed with the key word K_1 to yield T_1, denoted as $T_1^* = K_1 \oplus Y_4^*$, $T_1 = K_1 \oplus Y_4$. Consequently,

$$T_1^* = T_1 \oplus Y_4 \oplus Y_4^* = T_1 \oplus 100\ldots100\ldots100\ldots0,\ T_0^* = T_0$$

Table 7: The probability of guessing the correct β' under different fault values in S^*

fault value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\beta' = 00$	1	1/4	-	1/2	-	1/4	1/4	-	1/4	-	1/4	1/2	1/4	1/4	1/4	1/4
$\beta' = 01$	-	1/4	1/2	1/2	-	1/4	1/4	-	1/4	1/2	1/4	-	1/4	1/4	1/4	1/4
$\beta' = 10$	-	1/4	-	-	1/2	1/4	1/4	1/2	1/4	-	1/4	1/2	1/4	1/4	1/4	1/4
$\beta' = 11$	-	1/4	1/2	-	1/2	1/4	1/4	1/2	1/4	1/2	1/4	-	1/4	1/4	1/4	1/4

4. $\beta' = \mathbf{11}$. Both the second LSB and LSB in output difference of S^* being 1. From the aforementioned analysis, it follows that

$$T_0^* = T_0 \oplus Y_3 \oplus Y_3^* = T_0 \oplus 100\ldots100\ldots100\ldots0 (1st, 11th \text{ and } 18th \text{ bits are 1}),$$

$$T_1^* = T_1 \oplus Y_4 \oplus Y_4^* = T_1 \oplus 100\ldots100\ldots100\ldots0 (1st, 8th \text{ and } 42th \text{ bits are 1}).$$

During the decryption process, when inputting $K \times N \times AD \times C \times T^*$, only one of the four aforementioned scenarios can successfully pass the tag verification. By repetitively injecting nibble faults into the same S^* box and guessing T^*, multiple faulty tags capable of passing verification can be collected. At this juncture, Nonce-respect can transition into a Nonce-misuse scenario. Notably, the probabilities of the aforementioned four scenarios occurring are not uniform. According to Table 6, we derive the following properties.

Proposition 1. *If S^* box input value $m \in \{0, 1, 2, 3, 4, 6, 9, 10, 12, 13, 14\}$, the probabilities in above four situations are $\frac{5}{16}, \frac{3}{16}, \frac{3}{16}, \frac{5}{16}$ respectively; If $m \in \{5, 7, 8, 11, 14\}$, the probabilities of four situations are $\frac{3}{16}, \frac{5}{16}, \frac{5}{16}, \frac{3}{16}$ respectively.*

The concept introduced in Proposition 1 deviates somewhat from mere subconscious intuition. Conventionally, it is posited that the likelihood of encountering each of the four aforementioned situations is 1/4. Consequently, when attempting to guess β' without prior knowledge of the fault value, it's common practice to initially lean towards guessing $\beta' = 00$ or $\beta' = 11$, thus increasing the probability of correctly identifying β' that would allow $T^* = T \oplus \beta'$ to pass verification sooner.

During the analysis employing the proposed fault models, the value of the random-nibble fault is known to us. Consequently, a row in Table 6 can be directly locked. For example, if the fault value is 4, β' cannot be 00 or 01. Then the probability of accurately guessing β' rises to 1/2, as demonstrated in Table 7 shows.

Subsequently, based on the data presented in Table 7, we further derive Theorem 1.

Theorem 1. *In the Nonce-respect scenario, the expectation number of faulty tags T^* required for correctly guessing β' in a single S^* box is 2.03125.*

Proof. Observing above Table 7, it becomes evident that when $\alpha \in G = \{1, 5, 6, 8, 10, 12, 13, 14, 15\}$, the probability of determining the correct β' is 1/4. Conversely, when $\alpha \in H = \{2, 3, 4, 7, 9, 11\}$, this probability increases to 1/2. Additionally, when $\alpha = 0$, the probability of ascertaining the correct β' is 1 (as an empty fault does not alter the tag).

Furthermore, when $\alpha \in H$, the expectation of determining the correct β' is calculated as

$$E_H = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} \cdot 1 = 1.5 \text{ (Sampling without replacement)};$$

when $\alpha \in G$, the expectation becomes β' is

$$E_G = 1 \cdot \frac{1}{4} + 2 \cdot \frac{3}{4} \cdot \frac{1}{3} + 3 \cdot \frac{3}{4} \cdot \frac{2}{3} \cdot \frac{1}{2} + 4 \cdot \frac{3}{4} \cdot \frac{2}{3} \cdot 1 = 2.5.$$

Since each fault value in the random-nibble model carries the same probability, we deduce that

$$E_{guess} = \frac{1 \cdot 1 + 6 \cdot E_H + 9 \cdot E_G}{16} = 2.03125,$$

which marginally exceeds 2. □

The analysis in Theorem 1 elucidates that a single S* box permits successful tag authentication even in the presence of up to 2.03125 additional guesses of faulty tags during decryption. This signifies a relaxation of the nonce reuse restriction inherent in the Nonce-respect scenario, aligning it with the proposed PDFA's Nonce-misuse scenario.

4.2 Specific PDFA process

4.2.1 Single S-box

The steps for recovering the input of a single S-box are delineated below:

Step1 : Begin by selecting any key, nonce, associated data, and plaintext for Ascon encryption, denoted as $E_K(N, AD, P)$.

Step2 : During the algorithm's execution, inject a stuck-at-0(1) fault into the middle (3rd) bit of the ith S-box when reaching the final round of permutation p^a in the Finalization stage, yielding the tag T.

Step3(**Nonce − misuse scenario**): Re-execute the encryption algorithm with the same parameters. At the final round of permutation p^a in Finalization, inject stuck-at-0(1) fault into the middle (3rd) bit of the ith S-box. Subsequently, inject a random-nibble fault α_{i_1} into the remaining 4 bits of the ith S-box, generating the faulty tag $T^* = T_0^*||T_1^*$.

Step3*(**Nonce − respect scenario**): Utilize the same parameters to execute the decryption algorithm. Guess the tag $T^* = (T_0 \oplus \Delta Y_3)||(T_1 \oplus \Delta Y_4)$ computed during the decryption process at this point and employ T^* as the input for decryption. Upon reaching the final round of permutation p^a in Finalization, inject a stuck-at-0(1) fault into the middle (3rd) bit of the ith S-box. Then inject a random-nibble fault into the remaining 4 bits of the same S-box, and verify whether the validation succeeds. If not, continue to guess the tag T^* until validation is achieved.

Step4: Calculate the difference value $\Delta T = T \oplus T^* = \Delta T_0||\Delta T_1$ between the correct and faulty tags, where $\Delta T_0 = T_0 \oplus T_0^*$, $\Delta T_1 = T_1 \oplus T_1^*$.

Step5: Represent the 64-bit words ΔT_0 and ΔT_1 in vector form, calculate $\Delta X_3 = L_3^{-1}(\Delta T_0)$ and $\Delta X_4 = L_4^{-1}(\Delta T_1)$. Then, obtain the results $\Delta X_3 = (0, \ldots 0, \Delta x_{3,i}, \ldots, 0, \ldots, 0)$, $\Delta X_4 = (0, \ldots 0, \Delta x_{4,i}, \ldots, 0, \ldots, 0)$.

Step6: For the ith S-box S_i, the last two bits of the output difference β_i' can be expressed as $\beta_i' = \Delta x_{3,i} \oplus \Delta x_{4,i}$.

Step7: Refer to Table 6 to solve the differential equation $S^*(m) \oplus S^*(m \oplus \alpha_i) = \beta_i'$, and save all possible input m in S* box input candidate set Q_i.

Step8: Repeat the above steps from **Step3** to **Step7** n times. For S_i^*, n input candidate sets $Q_{i_1}, Q_{i_2}, \ldots Q_{i_n}$ after n fault injections can be obtained. When $|Q_{i_1} \cap Q_{i_2} \cap \cdots \cap Q_{i_n}| = 1$ is satisfied, the input value of S_i^* is recovered, which is $m_i = Q_{i_1} \cap Q_{i_2} \cap \cdots \cap Q_{i_n}$.

Step9: Repeat the above steps from **Step1** to **Step8** 64 times. Find the input values $m_i (i = 0, \ldots, 63)$ of all 64 S* boxes.

Extract the second LSB bit $x_{3,i}$ and LSB bit $x_{4,i}$ of all 64 S-boxes output $S(\lfloor m_i \rfloor_2 ||0|| \lceil m_i \rceil_2)$ respectively and concatenate them into 64-bit intermediate state words X_3 and X_4, then K can be expressed as

$$K = K_0||K_1 = (\Sigma_3(X_3) \oplus T_0)||(\Sigma_4(X_4) \oplus T_1).$$

4.2.2 All S-boxes in a round

Steps of recovering input of all S-boxes in a round at a time are listed below:

Step1 : Begin by selecting any key, nonce, associated data, and plaintext for Ascon encryption, denoted as $E_K(N, AD, P)$.

Step2 : At the final round of the permutation p^a during Finalization, simultaneously introduce stuck-at-0(1) faults into the middle (3rd) bits of all 64 S-boxes to generate the tag T.

Step3(**Nonce − misuse scenario**): Re-execute the encryption algorithm with identical parameters. During the final round of permutation p^a in Finalization, inject stuck-at-0(1) faults into the middle (3rd) bits of all 64 S-boxes. Subsequently, inject 64 random-nibble faults $\alpha_i (i = 0, 1, \ldots, 63)$ into the remaining 4 bits of all S-boxes, resulting in the faulty tag $T^* = T_0^* || T_1^*$.

Step3*(**Nonce − respect scenario**): Proceed with the decryption algorithm using the same parameters and guess the tag $T^* = (T_0 \oplus \Delta Y_3) || (T_1 \oplus \Delta Y_4)$ computed during decryption at this stage, and utilize T^* as the input for decryption. During the final round of permutation p^a in Finalization, inject stuck-at-0(1) faults into the middle (3rd) bits of all 64 S-boxes. Then inject 64 random-nibble faults $\alpha_i (i = 0, 1, \ldots, 63)$ into the remaining 4 bits of all S-boxes. Observe the verification process; if it fails, continue refining the guessed tag T^* until validation succeeds.

Step4: Compute the difference $\Delta T = T \oplus T^* = \Delta T_0 || \Delta T_1$ between the correct and faulty tags, where $\Delta T_0 = T_0 \oplus T_0^*$, $\Delta T_1 = T_1 \oplus T_1^*$.

Step5: Convert the 64-bit words ΔT_0 and ΔT_1 into vector form. Compute $\Delta X_3 = L_3^{-1}(\Delta T_0)$ and $\Delta X_4 = L_4^{-1}(\Delta T_1)$. This yields $\Delta X_3 = (\Delta x_{3,0}, \Delta x_{3,1}, \ldots, \Delta x_{3,63})$ and $\Delta X_4 = (\Delta x_{4,0}, \Delta x_{4,1}, \ldots, \Delta x_{4,63})$.

Step6: For all 64 S-boxes $\mathrm{S}_i (i = 0, 1, \ldots 63)$, the last two bits of the output difference β_i' can be expressed as $\beta_i' = \Delta x_{3,i} \oplus \Delta x_{4,i}$.

Step7: Refer to Table 6 to solve the differential equation $S^*(m) \oplus S^*(m \oplus \alpha_i) = \beta_i'$ for each S^* box, and save all possible input m in S_i^* box input candidate set Q_i.

Step8: Repeat the above steps from **Step3** to **Step7** n times. For S_i^*, n input candidate sets $Q_{i_1}, Q_{i_2}, \ldots Q_{i_n}$ after n fault injections can be obtained. When $|Q_{i_1} \cap Q_{i_2} \cap \cdots \cap Q_{i_n}| = 1$ is satisfied for all $0 \leq i \leq 63$, then the SDFA process is completed. At this stage, the input value of S_i^* is $m_i = Q_{i_1} \cap Q_{i_2} \cap \cdots \cap Q_{i_n}$.

The representation of K remains consistent with that in Subsubsection 4.2.1. Notably, the computational complexity of guessing the faulty tag reaches $4^{64} = 2^{128}$ in **Step3***. Consequently, during practical analysis, random-nibble fault injections into full-round S-boxes is only viable for Nonce-misuse scenarios.

5 Complexity analysis

In Section 4, we provided a detailed process of PDFA under different random-nibble fault models and nonce configurations. The subsequent problem is the complexity of analysis. In Subsubsection 4.1.2, we discuss the success rate of guessing the faulty tag T^* within the Nonce-respect scenario. This section now shifts focus to another critical issue highlighted at the end of Subsubsection 3.2.2. We aim to discuss the probability of successfully recovering K and the expectation of required number of fault injections under two distinct random-nibble fault models.

As highlighted in Subsubsection 3.2.2, following two random-nibble fault injections, the S_i^* box input may not necessarily converge to a unique value. Uniqueness in determination occurs only when the fault trail from α_1 to α_2 assumes specific values, denoted as $|\langle \alpha_1 \to \alpha_2 \rangle| = 1$. Specifically, under these conditions, the first fault α_1 takes the value $G = \{1, 5, 6, 8, 10, 12, 13, 14, 15\}$; Conversely, when α_1 assumes the value $H = \{2, 3, 4, 7, 9, 11\}$, it becomes impossible to uniquely determine the S_i^* box input, resulting in $|\langle \alpha_1 \to \alpha_2 \rangle| \geq 2$. Notably, when $\alpha_1 = 0$(empty fault), it is also impossible to uniquely determine the S_i^* input. Unlike the case in H, this fault offers no assistance in filtering possible inputs of S_i^*, hence possessing unique statistical properties warranting separate discussion. Next, we generalize all the above properties to the case of multiple fault injections.

Theorem 2. *Let G_n represents the number of fault trails that can obtain a unique input after n fault injections in S^* when α_1 takes elements in G, then $G_n = 2^{2n-4} \cdot (2^n - 2)^2$.*

Proof. Given that $\alpha_1 \in G$, as established in Subsubsection 3.2.2, we can deduce that $|\langle\alpha_1\rangle| = 4$. Consequently, following the nth fault injection, all scenarios involving the screening of input from the S_i^* box can be categorized into three cases:

A. $|\langle\alpha_1 \to \alpha_n\rangle| = 4$;
B. $|\langle\alpha_1 \to \alpha_n\rangle| = 2$;
C. $|\langle\alpha_1 \to \alpha_n\rangle| = 1$.

When α_n falls into cases A, B, and C, we denote the number of fault trails resulting in a unique input as g_{n_a}, g_{n_b}, and g_{n_c} , respectively. Since the elements in G share the same properties of fault trails, let's consider the scenario with $\alpha_1 = 1$ without loss of generality.

Case A: There are 4 values for $\alpha_n \in \{0, 1, 2, 3\}$ satisfying A. Due to $|\langle\alpha_1 \to \alpha_n\rangle| = 4$ and $|\langle\alpha_1\rangle| = 4$, it follows that $\langle\alpha_1\rangle \subseteq \langle\alpha_n\rangle$, implying that the nth fault injection does not contribute to filtering the input of the S_i^* box. As $|\langle\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n\rangle| = 1$, it must be the case that $|\langle\alpha_1 \to \alpha_2 \to \cdots \to \alpha_{n-1}\rangle| = 1$. Consequently, when the penultimate fault α_{n-1} is injected, the unique input of S_i^* has been obtained. Thus, $g_{n_a} = G_{n-1}$, and the total number of trails in Case A is $4 \cdot G_{n-1}$.

Case B: There are 8 values for $\alpha_n \in \{4, 5, 6, 7, 8, 9, 10, 11\}$ satisfying B in total. Due to the conditions of $|\langle\alpha_1 \to \alpha_n\rangle| = 2$ is more complicated, we continue adopting classification discussion:

B1. $|\langle\alpha_1 \to \alpha_{n-1} \to \alpha_n\rangle| = 1$;
B2. $|\langle\alpha_1 \to \alpha_{n-1} \to \alpha_n\rangle| = 2$.

When α_{n-1} falls into cases B1 and B2, we denote the number of fault trails resulting in a unique input as $g_{n_{b1}}$ and $g_{n_{b2}}$. Similarly, since all α_n in Case B share the same properties of fault trails, let's consider the scenario with $\alpha_n = 4$ without loss of generality.

Case B1: There are 8 values for $\alpha_{n-1} \in \{8, 9, 10, 11, 12, 13, 14, 15\}$ satisfying B1 in total. At this time there is $|\langle\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n\rangle| = |\langle\alpha_1 \to \alpha_{n-1} \to \alpha_n\rangle| = 1$. Regardless of the values of $\alpha_2, \alpha_3, \ldots \alpha_{n-2}$, the unique input of S^* box can be identified by α_1, α_{n-1} and α_n. Hence, $g_{n_{b1}} = 16^{n-3}$, and the total number of trails in Case B1 is $8 \cdot 16^{n-3}$.

Case B2: There are 8 values for $\alpha_{n-1} \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ satisfying B2 in total. In this case, the last two fault injections can be combined into once. Let the combined fault $\alpha' = \alpha_{n-1} \to \alpha_n$, which implies $\langle\alpha'\rangle = \langle\alpha_{n-1} \to \alpha_n\rangle$. When $|\langle\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n\rangle| = 1$, it is equivalent to $|\langle\alpha_1 \to \alpha_2 \to \cdots \to \alpha_{n-2} \to \alpha'\rangle| = 1$. Similarly, when $|\langle\alpha_1 \to \alpha_{n-1} \to \alpha_n\rangle| = 2$, it is equivalent to $|\langle\alpha_1 \to \alpha'\rangle| = 2$. Hence, Case B2 after n fault injections is equivalent to the Case B after $n-1$ fault injections. Therefore, $g_{n_{b2}} = g_{n-1_b}$, and the total number of trails in Case B2 is $8 \cdot g_{n-1_b}$.

Now, let's consider the entire Case B, where $g_{n_b} = 8 \cdot 16^{n-3} + 8 \cdot g_{n-1_b}$. Since there are 8 values for α_n satisfying B, the total number of trails in Case B is $8 \cdot g_{n_b} = 8 \cdot (8 \cdot 16^{n-3} + 8 \cdot g_{n-1_b})$.

Case C: There are 4 values for $\alpha_n \in \{12, 13, 14, 15\}$ satisfying C. At this point, there is $|\langle\alpha_1 \to \alpha_2 \to \cdots \to \alpha_n\rangle| = |\langle\alpha_1 \to \alpha_n\rangle| = 1$. In other words, regardless of the values of $\alpha_2, \alpha_3, \ldots \alpha_{n-1}$, the unique input of S^* box can be determined by α_1 and α_n. Thus, $g_{n_c} = 16^{n-2}$, and the total number of trails in Case C is $4 \cdot 16^{n-2}$.

To summarize, when α_1 takes elements from G, the number of fault trails that can obtain a unique input after n fault injections is:

$$\begin{aligned} G_n &= 4 \cdot g_{n_a} + 8 \cdot g_{n_b} + 4 \cdot g_{n_c} \\ &= 4 \cdot G_{n-1} + 8 \cdot (8 \cdot 16^{n-3} + 8 \cdot g_{n-1_b}) + 4 \cdot 16^{n-2} \\ &= 4 \cdot G_{n-1} + 8 \cdot 16^{n-2} + 64 \cdot g_{n-1_b} \end{aligned} \tag{1}$$

Equation 1 is a recurrence relation. Combining it with another recurrence relation

$$g_{n_b} = 8 \cdot 16^{n-3} + 8 \cdot g_{n-1_b} \tag{2}$$

We can derive the general formula for G_n.

First, let's find the general term of g_{n_b} in Equation 2. It is easy to know that when α_2 falls into Case B, there is $|\langle \alpha_1 \to \alpha_2 \rangle| = 2$. Consequently, the initial value is $g_{2_b} = 0$. Utilizing Mathematica components, we can derive the general term of g_{n_b}, which is

$$g_{n_b} = 2^{3n-8} \cdot (2^n - 4), n \geq 2$$

Thus, the general term of G_n is given by $G_{n+1} = 4 \cdot G_n + 8 \cdot 16^{n-1} + 2^{3n-2} \cdot (2^n - 4)$. From Figure 2, we know that when $\alpha_1 \in G$, there is $G_2 = 4$. Therefore, the general term of G_n is found to be:

$$G_n = 2^{2n-4} \cdot (2^n - 2)^2, n \geq 1$$

□

Next, we continue to discuss the number of fault trails when the first fault $\alpha_1 \in H = \{2, 3, 4, 7, 9, 11\}$.

Theorem 3. *Let H_n represents the number of fault trails that can obtain a unique input after n fault injections in S^* when α_1 takes elements in H, then $H_n = 2^{n-4} \cdot (2^n - 4) \cdot (2^n - 2)^2$.*

Proof. Given that $\alpha_1 \in H$, as established in Subsubsection 3.2.2, we can deduce that $|\langle \alpha_1 \rangle| = 8$. No matter what value α_2 takes, there is $|\langle \alpha_1 \to \alpha_n \rangle| \geq 2$. Consequently, following the *nth* fault injection, all scenarios involving the screening of input from the S_i^* box can be categorized into three cases:

A. $|\langle \alpha_1 \to \alpha_n \rangle| = 8$;
B. $|\langle \alpha_1 \to \alpha_n \rangle| = 4$;
C. $|\langle \alpha_1 \to \alpha_n \rangle| = 2$.

Since the elements in H share the same properties of fault trails, let's consider the scenario with $\alpha_1 = 2$ without loss of generality.

Case A: There are 2 values for $\alpha_n \in \{0, 2\}$ satisfying A. As $|\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \rangle| = 1$, it must be the case that $|\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_{n-1} \rangle| = 1$. Consequently, when the penultimate fault α_{n-1} is injected, the unique input of S_i^* has been obtained. When α_n falls into cases A, we denote the number of fault trails resulting in a unique input as h_{n_a}. Thus, $h_{n_a} = H_{n-1}$, and the total number of trails in Case A is $2 \cdot H_{n-1}$.

Case B: There are 8 values for $\alpha_n \in \{4, 5, 6, 7, 8, 9, 10, 11\}$ satisfying B in total. Due to the conditions of $|\langle \alpha_1 \to \alpha_n \rangle| = 4$ is more complicated, we continue adopting classification discussion.

case i. When $\alpha_n \in \{1, 3, 4, 5, 6, 7\}$, we denote the number of fault trails resulting in a unique input after n fault injections as h_{n_b}. All cases can be divided into the following three categories.

B1. $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 1$;
B2. $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 2$.
B3. $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 4$.

When α_{n-1} falls into cases B1, B2 and B3, we denote the number of fault trails resulting in a unique input as $h_{n_{b1}}$, $h_{n_{b2}}$ and $h_{n_{b3}}$ respectively. Similarly, since all α_n in case i share the same properties of fault trails, let's consider the scenario with $\alpha_n = 1$ without loss of generality.

Case B1: There are 4 values for $\alpha_{n-1} \in \{12, 13, 14, 15\}$ satisfying B1 in total. At this time there is $|\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \rangle| = |\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 1$. Regardless of the

values of $\alpha_2, \alpha_3, \ldots \alpha_{n-2}$, the unique input of S* box can be identified by α_1, α_{n-1} and α_n. Hence, $h_{n_{b1}} = 16^{n-3}$, and the total number of trails in Case B1 in case i is $4 \cdot 16^{n-3}$.

Case B2: There are 8 values for $\alpha_{n-1} \in \{4, 5, 6, 7, 8, 9, 10, 11\}$ satisfying B2 in total. In this case, the last two fault injections can be combined into once. Let the combined fault $\alpha' = \alpha_{n-1} \to \alpha_n$, which implies $\langle \alpha' \rangle = \langle \alpha_{n-1} \to \alpha_n \rangle$. When $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 2$, it is equivalent to $|\langle \alpha_1 \to \alpha' \rangle| = 2$. Hence, Case B2 after n fault injections in case i is equivalent to the Case C after $n-1$ fault injections. Therefore, $h_{n_{b2}} = h_{n-1_c}$, and the total number of trails in Case B2 in case i is $8 \cdot h_{n-1_c}$.

Case B3: There are 4 values for $\alpha_{n-1} \in \{0, 1, 2, 3\}$ satisfying B3 in total. Similarly, let the combined fault $\alpha' = \alpha_{n-1} \to \alpha_n$. When $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 4$, it is equivalent to $|\langle \alpha_1 \to \alpha' \rangle| = 4$. Hence, Case B3 after n fault injections in case i is equivalent to the Case i in case B after $n-1$ fault injections. Therefore, $h_{n_{b3}} = h_{n-1_b}$, and the total number of trails in Case B3 in case i is $4 \cdot h_{n-1_b}$.

Now, let's consider the entire case i in Case B, where $h_{n_b} = 4 \cdot 16^{n-3} + 8 \cdot h_{n-1_c} + 4 \cdot h_{n-1_b}$. Since there are 6 values for α_n satisfying case i, the total number of trails in Case i is $6 \cdot h_{n_b}$.

case ii. When $\alpha_n \in \{9, 11\}$, we denote the number of fault trails resulting in a unique input after n fault injections as h'_{n_b}. All cases can be divided into the only two categories.

B2. $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 2$.

B3. $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 4$.

When α_{n-1} falls into cases B2 and B3, we denote the number of fault trails resulting in a unique input as $h'_{n_{b2}}$ and $h'_{n_{b3}}$ respectively. Similarly, since all α_n in case ii share the same properties of fault trails, let's consider the scenario with $\alpha_n = 9$ without loss of generality.

Case B2: There are 12 values for $\alpha_{n-1} \in \{1, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15\}$ satisfying B2 in total. In this case, the last two fault injections can be combined into once. Let the combined fault $\alpha' = \alpha_{n-1} \to \alpha_n$. When $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 2$, it is equivalent to $|\langle \alpha_1 \to \alpha' \rangle| = 2$. Hence, Case B2 after n fault injections in case ii is equivalent to the Case C after $n-1$ fault injections. Therefore, $h'_{n_{b2}} = h_{n_{b2}} = h_{n-1_c}$, and the total number of trails in Case B2 in case ii is $12 \cdot h_{n-1_c}$.

Case B3: There are 4 values for $\alpha_{n-1} \in \{0, 2, 9, 11\}$ satisfying B3 in total. Similarly, let the combined fault $\alpha' = \alpha_{n-1} \to \alpha_n$. When $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 4$, it is equivalent to $|\langle \alpha_1 \to \alpha' \rangle| = 4$. Particularly, B3 in case ii is slightly different from that in case i. Case B3 after n fault injections in case ii is equivalent to the Case ii (Not Case i) in case B after $n-1$ fault injections. Therefore, $h'_{n_{b3}} = h'_{n-1_b}$, and the total number of trails in Case B3 in case ii is $4 \cdot h'_{n-1_b}$.

Now, let's consider the entire case ii in Case B, where $h'_{n_b} = 12 \cdot h_{n-1_c} + 4 \cdot h'_{n-1_b}$. Since there are 2 values for α_n satisfying case ii, the total number of trails in Case ii is $2 \cdot h'_{n_b}$. Furthermore, combining two cases of i and ii, the total number of trails in Case B can be obtained as $6 \cdot h_{n_b} + 2 \cdot h'_{n_b}$

Case C: There are 6 values for $\alpha_n \in \{8, 10, 12, 13, 14, 15\}$ satisfying C in total. Due to the conditions of $|\langle \alpha_1 \to \alpha_n \rangle| = 2$ is more complicated, we continue adopting classification discussion:

C1. $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 1$;

C2. $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 2$.

When α_{n-1} falls into cases C1 and C2, we denote the number of fault trails resulting in a unique input as $h_{n_{c1}}$ and $h_{n_{c2}}$. Similarly, since all α_n in Case C share the same properties of fault trails, let's consider the scenario with $\alpha_n = 8$ without loss of generality.

Case C1: There are 8 values for $\alpha_{n-1} \in \{4, 5, 6, 7, 12, 13, 14, 15\}$ satisfying C1 in total. At this time there is $|\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \rangle| = |\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 1$. Regardless of the values of $\alpha_2, \alpha_3, \ldots \alpha_{n-2}$, the unique input of S* box can be identified by α_1, α_{n-1}

and α_n. Hence, $h_{n_{c1}} = 16^{n-3}$, and the total number of trails in Case C1 is $8 \cdot 16^{n-3}$.

Case C2: There are 8 values for $\alpha_{n-1} \in \{0, 1, 2, 3, 8, 9, 10, 11\}$ satisfying C2 in total. In this case, the last two fault injections can be combined into once. Let the combined fault $\alpha' = \alpha_{n-1} \to \alpha_n$, which implies $\langle \alpha' \rangle = \langle \alpha_{n-1} \to \alpha_n \rangle$. When $|\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n \rangle| = 1$, it is equivalent to $|\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_{n-2} \to \alpha' \rangle| = 1$. Similarly, when $|\langle \alpha_1 \to \alpha_{n-1} \to \alpha_n \rangle| = 2$, it is equivalent to $|\langle \alpha_1 \to \alpha' \rangle| = 2$. Hence, Case C2 after n fault injections is equivalent to the Case C after $n-1$ fault injections. Therefore, $h_{n_{c2}} = h_{n-1_c}$, and the total number of trails in Case C2 is $8 \cdot h_{n-1_c}$.

Now, let's consider the entire Case C, where $h_{n_c} = 8 \cdot 16^{n-3} + 8 \cdot h_{n-1_c}$. Since there are 6 values for α_n satisfying C, the total number of trails in Case C is $6 \cdot h_{n_c} = 6 \cdot (8 \cdot 16^{n-3} + 8 \cdot h_{n-1_c})$.

To summarize, when α_1 takes elements from H, the number of fault trails that can obtain a unique input after n fault injections is:

$$\begin{aligned} H_n &= 2 \cdot h_{n_a} + 6 \cdot h_{n_b} + 2 \cdot h'_{n_b} + 6 \cdot h_{n_c} \\ &= 2 \cdot H_{n-1} + 6 \cdot (h_{n_{b1}} + h_{n_{b2}} + h_{n_{b3}}) + 2 \cdot (h'_{n_{b2}} + h'_{n_{b3}}) + 6 \cdot (h_{n_{c1}} + h_{n_{c2}}) \\ &= 2 \cdot H_{n-1} + 6 \cdot (4 \cdot 16^{n-3} + 4 \cdot h_{n-1_b} + 8 \cdot h_{n-1_c}) + 2 \cdot (12 \cdot h_{n-1_c} + 4 \cdot h'_{n-1_b}) \\ &\quad + 6 \cdot (8 \cdot 16^{n-3} + 8 \cdot h_{n-1_c}) \\ &= 2 \cdot H_{n-1} + 72 \cdot 16^{n-3} + 24 \cdot h_{n-1_b} + 8 \cdot h'_{n-1_b} + 120 \cdot h_{n-1_c} \end{aligned} \tag{3}$$

Equation 3 is a recurrence relation. Combining it with another three recurrence relations

$$h_{n_c} = 8 \cdot 16^{n-3} + 8 \cdot h_{n-1_c} \tag{4}$$

$$h'_{n_b} = 12 \cdot h_{n-1_c} + 4 \cdot h'_{n-1_b} \tag{5}$$

$$h_{n_b} = 4 \cdot 16^{n-3} + 8 \cdot h_{n-1_c} + 4 \cdot h_{n-1_b} \tag{6}$$

We can derive the general formula for H_n.

First, let's find the general term of h_{n_c} in Equation 4. It is easy to know that the initial value is $h_{2_c} = 0$. Utilizing Mathematica components, we can derive the general term of h_{n_c}, which is

$$h_{n_c} = 2^{3n-8} \cdot (2^n - 4), n \geq 2$$

Then Equation 5 and Equation 6 can be expressed as $h'_{n+1_b} = 3 \cdot 2^{3n-6} \cdot (2^n - 4) + 4 \cdot h'_{n_b}$ and $h_{n+1_b} = 4 \cdot 16^{n-2} + 2^{3n-5} \cdot (2^n - 4) + 4 \cdot h_{n_b}$. Combined with the initial value $h'_{2_b} = h_{2_b} = 0$, the general term of h'_{n_b} and h_{n_b} can be obtained as:

$$h'_{n_b} = 4^{n-4} \cdot (2^n - 4) \cdot (2^n - 8), n \geq 2$$

$$h_{n_b} = 2^{2n-8} \cdot (2^n - 4)^2, n \geq 2$$

Thus, the general term of H_n is given by

$$\begin{aligned} H_{n+1} &= 2 \cdot H_n + 72 \cdot 16^{n-2} + 24 \cdot h_{n_b} + 8 \cdot h'_{n_b} + 8 + 120 \cdot h_{n_c} \\ &= 2 \cdot H_n + 72 \cdot 16^{n-2} + 3 \cdot 2^{2n-5} \cdot (2^n - 4)^2 + 2^{2n-5} \cdot (2^n - 4) \cdot (2^n - 8) \\ &\quad + 15 \cdot 2^{3n-5} \cdot (2^n - 4) \end{aligned}$$

From Figure 2, we know that when $\alpha_1 \in H$, there is $H_2 = 0$. Therefore, the general term of H_n is found to be:

$$H_n = 2^{n-4} \cdot (2^n - 4) \cdot (2^n - 2)^2, n \geq 1$$

□

Combining Theorem 2 and Theorem 3, the following Theorem 4 can be obtained, which is the core theorem of this section.

Theorem 4. *Let T_n represents the number of fault trails that can obtain a unique input after n fault injections in S^*, then $T_n = (2 - 3 \cdot 2^n + 2^{2n})^2$.*

Proof. According to Theorem 2 and Theorem 3, when $\alpha_1 \in G = \{1, 5, 6, 8, 10, 12, 13, 14, 15\}$, we have $G_n = 2^{2n-4} \cdot (2^n - 2)^2$; When $\alpha_1 \in H = \{2, 3, 4, 7, 9, 11\}$, we have $H_n = 2^{n-4} \cdot (2^n - 4) \cdot (2^n - 2)^2$. In particular, when $\alpha_1 = 0$, it is an empty fault, and thus we have

$$|\langle \alpha_1 \to \alpha_2 \to \cdots \to \alpha_{n-1} \rangle| = |\langle \alpha_2 \to \cdots \to \alpha_{n-1} \rangle| = 1.$$

This is equivalent to the case where a unique solution is obtained after n fault injections, so the number of fault trails is T_{n-}. Therefore, Equation 7 can be obtained as follows:

$$\begin{aligned} T_n &= T_{n-1} + 9 \cdot G_n + 6 \cdot H_n \\ &= T_{n-1} + 9 \cdot 2^{2n-4} \cdot (2^n - 2)^2 + 6 \cdot 2^{n-4} \cdot (2^n - 4) \cdot (2^n - 2)^2 \end{aligned} \tag{7}$$

Thus, the general term of T_n is given by

$$T_n = (2 - 3 \cdot 2^n + 2^{2n})^2, n \geq 1$$

□

After calculating the number of fault trails that obtain a unique solution after n fault injections, we can calculate the probability of key recovery after (exactly) the nth fault injection. Moreover, the expectation of the number of fault injections required to successfully recover K under two random-nibble fault models can be calculated, as stated in Theorem 5.

Theorem 5. *The probability that a single S-box has a unique solution after n fault injections is $p_n = \left(\frac{(2^n-1)\cdot(2^n-2)}{4^n}\right)^2, n \geq 1$, The expectation of the number of fault injections required to recover the input value of a single S-box is $E \approx 4.114$, and the expectation of the number of fault injections required to recover the input value of all S-boxes in a round is $E' \approx 9.912$.*

Proof. According to Theorem 4, the number of fault trails that can yield a unique solution after n fault injections is given by $T_n = (2 - 3 \cdot 2^n + 2^{2n})^2$. It is evident that the total number of fault trails after n fault injections is 16^n. Thus, the probability of achieving a unique solution after n fault injections is:

$$p_n = \frac{T_n}{16^n} = \frac{(2 - 3 \cdot 2^n + 2^{2n})^2}{16^n} = \left(\frac{2 - 3 \cdot 2^n + 2^{2n}}{4^n}\right)^2 = \left(\frac{(2^n - 1) \cdot (2^n - 2)}{4^n}\right)^2.$$

Further, let $\tilde{p}_n$ denote the probability of obtaining a unique solution after exactly n fault injections. We establish that $\tilde{p}_1 = p_1 = 0$, and for $n \geq 2, \tilde{p}_n = p_n - p_{n-1}$. Now, let's examine the scenario of injecting 64 random-nibble faults simultaneously to recover the full-round S-box. Given the independence of the 64 S-boxes, the probability of attaining a unique solution after n fault injections is p_n^{64}. Refer to Table 8 for the values of p_n, $\tilde{p}_n$ and p_n^{64}.

Table 8: Probability ($\tilde{p}_n$) p_n that there is a unique input of S^* after (exactly) 1 to 20 fault injections

n	1	2	3	4	5	6	7
p_n	0	14.0625%	43.0664%	67.2913%	82.4833%	90.9378%	95.3913%
$\tilde{p}_n$	0	14.0625%	29.0039%	24.2249%	15.192%	8.4545%	4.4535%
p_n^{64}	0	2.99×10^{-55}	3.84×10^{-24}	9.75×10^{-12}	4.44×10^{-6}	2.29×10^{-3}	0.0488

n	8	9	10	11	12	13	14
p_n	97.676%	98.833%	99.4153%	99.7073%	99.8536%	99.9268%	99.9634%
$\tilde{p}_n$	2.285%	1.157%	0.5823%	0.2920%	0.1463%	0.0732%	0.0366%
p_n^{64}	0.222	0.4718	0.6871	0.829	0.9105	0.9542	0.9768

n	15	16	17	18	19	20	.....
p_n	99.9817%	99.9908%	99.9954%	99.9977%	99.9989%	99.9994%	.....
$\tilde{p}_n$	0.0183%	0.0091%	0.0046%	0.0023%	0.0012%	0.0005%	.....
p_n^{64}	0.9883	0.9942	0.9971	0.9985	0.9993	0.9996	.....

Observing Table 8 reveals that for $n > 20$, $\tilde{p}_n$ approaches 0 infinitely, rendering their contributions to the expectation of fault injections negligible. Consequently, the expectation of fault nibbles required to recover the input value of a single S^* box is determined by $E = \sum_{n=1}^{20} n \cdot \tilde{p}_n = \sum_{n=2}^{20} n \cdot (p_n - p_{n-1}) \approx 4.114$. Similarly, the expectation of injecting 64 random-nibble faults simultaneously in full-round S^* boxes is denoted as is $E' = \sum_{n=2}^{20} n \cdot (p_n^{64} - p_{n-1}^{64}) \approx 9.913$.

□

6 Simulation of PDFA experiments

6.1 Experimental environment

Regarding hardware specifications, the system comprises a PC equipped with an Intel Core i7-1260P 2.1GHz CPU, running a 64-bit operating system with 16GB of memory. The programming environment utilized is Visual C++ for Microsoft Visual Studio 2022, along with Wolfram Mathematica version 12.1.

6.2 Experimental results

In reference to Subsection 4.2, after recovering the 64 S^* box inputs $m_i (i = 0, 1, \ldots, 63)$, the second LSB $x_{3,i}$ and LSB $x_{4,i}$ of the 64 $S(\lfloor m_i \rfloor_2 || 0 || \lceil m_i \rceil_2)$ (or $S(\lfloor m_i \rfloor_2 || 1 || \lceil m_i \rceil_2)$) are individually extracted. These bits are then amalgamated into 64-bit intermediate state words denoted as $X_{3,i}$ and $X_{4,i}$. Consequently, the key K is represented as $K = K_0 || K_1 = (\Sigma_3(X_3) \oplus T_0) \,||\, (\Sigma_4(X_4) \oplus T_1)$, thus concluding the analysis of the Ascon algorithm. The complexity associated with recovering the S-box input under the Nonce-misuse scenario aligns with the complexity of analyzing Ascon itself. In the Nonce-respect scenario, additional steps involve guessing faulty tags during the decryption process, leading to an average guessing complexity of $2.03125 \times 64 = 130$ tag queries.

To empirically evaluate the Ascon algorithm, we conduct 10 sets of experiments under both the Stuck-at-0 and Stuck-at-1 fault models, encompassing 10,000 PDFA trials in each set. In each trial, we meticulously document the number of fault injections required to recover the input of full-round S-boxes, as well as the average number needed to recover the input value of a single S-box. The culmination of these experimental results is presented in Table 9 below.

Table 9: Experimental results under 10 groups of stuck-at-0 and 10 groups of stuck-at-1 fault models

Experiment serial number	Stuck-at-0		Stuck-at-1	
	Average fault injections to recover all S-boxes	Average fault injections to recover single S-box	Average fault injections to recover all S-boxes	Average fault injections to recover single S-box
1	9.9190	4.11304	9.9271	4.11424
2	9.9164	4.11165	9.9197	4.11374
3	9.9365	4.11296	9.9156	4.11465
4	9.9137	4.11396	9.9261	4.11455
5	9.9327	4.11556	9.9130	4.11442
6	9.9026	4.11070	9.9053	4.10930
7	9.9158	4.11654	9.9079	4.11238
8	9.8951	4.1136	9.9254	4.11369
9	9.9162	4.11351	9.9106	4.11799
10	9.9049	4.11945	9.9215	4.11536

The experimental results in Table 9 are close to the theoretical results in Section 5, which illustrates the practicality and accuracy of the PDFA model.

7 Discussion

7.1 Comparison with existing analysis on Ascon

Ascon has undergone extensive scrutiny by researchers, leading to its selection as the lightweight use case portfolio in CAESAR competition and winner in NIST-LwC competition. The original submission by its designers proposed a zero-sum distinguisher for full 12-rounds permutation with complexity 2^{130}, alongside two key recovery attacks [DEMS15]. Subsequently, Jovanovic et al. [JLM14] validated the security assertions regarding Ascon's mode of operation. Tezcan [Tez16] introduced truncated, impossible, and improbable differential attacks targeting 5 of ASCON's 12 rounds during the initialization phase, with complexities ranging from 2^{109} to 2^{256}. Li et al. [LDW17] presented a 7-round key recovery attack, with a time complexity of $2^{103.9}$ for key retrieval. However, the full 12-round Ascon configuration remains impervious to such attacks. Additionally, Samwel et al. [SD17] conducted the first side-channel analysis of Ascon, employing a differential power analysis attack and correlation power analysis on a toy-sized ASCON implementation. Their findings suggest that attacking a full-sized and full-protected Ascon implementation is infeasible.

Ascon exhibits robust cryptographic properties, demonstrating resistance against various linear and differential cryptanalysis methods. However, vulnerabilities in Ascon's implementation become apparent in the face of side-channel and fault analysis targeting key recovery. In this context, Ramezanpour et al. [RAD19b] introduced a Statistical Ineffective Fault Analysis (SIFA) on full-round Ascon. Their approach relies on the assumption of an uneven or biased fault probability distribution. The complexity of key search depends on the number of experiments required to find statistics. Each experiment necessitates between 12.5 and 2500 correct tag values, individually subjected to invalid faults, with distributions ranging from highly biased to more even fault distributions. The minimum achievable key search space is 2^{124}. Moreover, the success of the attack hinges on a strong assumption that the attacker can collect pairs of faulty and fault-free tag values from the same input, which is infeasible in the Nonce-respect scenario. In the same year, Ramezanpour et al. [RAD19a] proposed Fault Intensity Map Analysis (FIMA) to enhance the previous technology reliant solely on error deviation. This advancement reduces the number of required fault injections by 50%, reaching a minimum of 250 and 305 for different

Table 10: Comparison between PDFA and existing fault attacks

Attack method	Fault model	Residual keyspace	Nonce configuration	Complexity (number of tags or faults)	Year
SIFA	random-AND	2^{124}	Nonce-misuse	12.5–2500	2019
FIMA	random-AND	2^{124}	Nonce-misuse	$p \in [0, 0.3]$, 250 $p \in [0, 0.2]$, 305	2019
SSFA	bit-reset multi-byte	Between 1 to 2^{64}	Nonce-respect	1 word, 85 1 Byte, 8×85 1 bit, 64×85	2019
DFA	bit-flip bit-set	-	Nonce-respect	1024	2023
PDFA	stuck-at random-nibble	-	Nonce-respect	Single S-box random nibble, 64×4.11=263 Full-round S-box random nibble, 9.91	2024
			Nonce-misuse	extra 130 faulty tag guesses	

data sizes. Joshi et al. [JM19] introduced a SubSet Fault Analysis (SSFA) attack on Ascon, exploiting various fault models with different granularities to uniquely recover keys. For the SSFA attack, they determined that 85, 680, and 5440 faults are needed to reduce the key space to values between 1 and 2^{64} at different bit-reset fault granularities of 1 bit, byte, and word, respectively. Furthermore, Surya et al. [SMS20] proposed a local clock glitch fault injection attack on Ascon-128. Jana [Jan23] introduced a Differential Fault Analysis (DFA) method tailored for ASCON, employing a two-stage fault model to facilitate error forgery. By leveraging a bit-flip fault in the first stage followed by a bit-set fault in the subsequent stage, the attacker can effectively retrieve the key through 1024 erroneous queries. This process necessitates an additional 576 bytes of memory and demonstrates linear time complexity. However, it's worth noting that the article lacks confirmation of both actual experimental results and theoretical values. Moreover, the practicality of this approach is hindered by the relatively large number of tag queries required. A detailed comparison between our proposed analysis method and existing fault analysis techniques is presented in Table 10.

7.2 Feasibility and practicality of the fault models

This part delves into critical considerations regarding the fault models addressed in the proposed analysis, emphasizing the feasibility and implementation challenges associated with each.

The first fault model examined is the stuck-at fault, with the underlying assumption that attackers can inject such faults. Practical experiments by Roscian et al. [RSDT13] on the RAM memory of a microcontroller revealed successful induction of stuck-at faults using a laser beam. Piscitelli et al. [PBR15] further noted a higher occurrence rate of stuck-at faults compared to bit-flip faults. While the laser-induced malfunction requires costly equipment, Skorobogatov [Sko10] demonstrated that stuck-at faults can be precisely injected in terms of both location and timing [BBKN12].

Moving on to random-nibble faults, the assumption is made that faults can be injected at nibble granularity. Agoyan et al. [ADM+10b] and Dutertre et al [DMNT10]. reported that, in laser-induced faults, injecting nibble faults is notably easier than injecting single-bit faults due to the diameter and spot size of the laser beam. This process demands more control and precision.

Regarding reproducibility, the fault model posits that the same fault can be accurately reproduced multiple times through laser beam injection. Dutertre et al. [DMNT10]

showcased reproducibility of byte errors on smart cards with $0.35\mu m$ microcontrollers and SOSSE operating systems at a clock frequency of 16MHz. Agoyan et al. [ADM+10a] conducted similar experiments, demonstrating the repeatability of injecting single-bit and nibble faults using a laser beam.

In conclusion, the fault models discussed in this article are shown to be reproducible and hold significant relevance across various practical applications.

7.3 Countermeasures

Countermeasures against PDFA can be categorized into three main groups: 1) Error detection, 2) Error randomization, and 3) Algorithm enhancement.

In the first category, error detection techniques leverage spatial/temporal redundancy to identify faults in cryptographic operations and implement measures to mitigate erroneous values. Examples of such redundant algorithms for detecting errors in cryptographic operations encompass those designed for 8-bit S-boxes [KJA+18], lightweight components [KTAS14], and utilizing SIMD to exploit software redundancy [LCFS18]. We propose an additional approach for PDFA, involving the sacrifice of part of the storage to complete the verification of calculation results from the final round of permutation p^a in S-boxes. For instance, in Ascon, the intermediate state S11 after the penultimate round ($11th$ round) in Finalization stage can be stored in memory. Subsequently, the calculation is completed, and the state is saved as S12. S12 is then decrypted to obtain the input of the previous round, denoted as S′11. Finally, the obtained S′11 is compared with the saved state S11. If a mismatch is detected, cipher execution is halted, and the system generates no output. If no mismatch occurs, execution is deemed complete.

The second category of countermeasures, error randomization, involves introducing supplementary operations on the cipher to randomize errors induced by fault injection, thus preventing the leakage of valuable information about the data-dependent distribution of faults to potential attackers. An illustrative example is the AES infection countermeasure proposed in [TBM14] and evaluated in [PCM17]. In this method, the state of cipher is cross-checked with a redundant state, and if any disparity arises due to an error, the state is substituted with a dummy state. Consequently, what becomes visible to the attacker is not an faulty state but rather a randomized state. To thwart attackers from injecting identical faults into both cryptographic and redundant states, a fault space transformation [PCMC17] is suggested, wherein cryptographic and redundant operations are executed using distinct encodings. Consequently, fault spaces in cryptographic operations are mapped to dissimilar fault spaces in redundant operations. The selection of this mapping ensures that the distribution of error values in the cryptographic state differs from that in the redundant state. For PDFA, the approach of randomizing error tags can be directly employed through fault space transformation.

The third category of countermeasures, specifically algorithm enhancement, which is exemplified by the cipher DEFAULT [BBB+21]. This method incorporates a meticulously designed DEFAULT-CORE component (lacking security against DFA) positioned between two DEFAULT-LAYERs (providing robust DFA security). Ideally, this arrangement includes a sophisticated cipher interposed between the two DEFAULT-LAYER blocks, resulting in an enhanced overall performance compared to previous configurations. While this design inherently curtails the information accessible to potential attackers, there remains a vulnerability wherein attackers can leverage the properties of the round function to identify standardized keys, characterizing large classes of equivalent keys. Through strategic fault placement optimization, the equivalent key fault complexity for recovering DEFAULT-LAYER can be diminished to fewer than 100 fault computations. The work underscores the effectiveness of password-level protection but emphasizes the necessity for more profound consideration beyond linear structures [NDE22]. Consequently, the paper draws inspiration from Ascon's 5-bit S-box, aiming to modify it in a way that preserves all

critical cryptographic properties of the original S-box, such as algebraic degree, differential uniformity, nonlinearity, balance, linear and differential branch numbers. The objective is to maximize the number of trails obtaining unique S-box input. For instance, after introducing a stuck-at fault in any of the 5-bit positions of the S-box [JM19] as outlined in Table 11, regardless of how many random-nibble faults are injected, a trail obtaining unique S-box input values will never emerge.

Table 11: Improved 5-bit S-box with PDFA resistance in Ascon

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(x)	4	25	31	6	26	7	9	16	27	5	8	18	15	17	20	14
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S(x)	30	19	21	28	0	13	3	10	2	12	1	11	22	24	29	23

8 Conclusion

The widespread adoption of AEAD algorithms in constrained devices has sparked increased interest in physical attacks like DFA, which forms the primary focus of this study. Moreover, the LwC competition outlined in [1] specifically highlights "side-channel and fault attacks" as essential criteria, further motivating our research endeavors. This paper delves into examining the susceptibility of the Ascon algorithm to DFA and proposes a PDFA analysis method capable of fully recovering the 128-bit key.

We introduce two fault models in this work: the stuck-at model and the random-nibble model. The former effectively condenses the 5-bit S-box into a 4-bit S-box, elucidating its distinctive differential properties. The latter encompasses two variations: single S-box fault injection and full-round S-box fault injection. We determine the average number of fault injections necessary to retrieve the S-box input under both models. Our experimental and theoretical findings corroborate each other, validating the accuracy of the proposed models.Furthermore, we present a rigorous mathematical framework for analyzing fault injections within the Ascon algorithm's S-box. We offer recursive relationships and general formulas to compute the number of fault trails in various scenarios. We assess the success rate of recovering the key K of Ascon under diverse fault models and derive mathematical expressions to quantify this metric. Additionally, we introduce two distinct nonce configurations. In the Nonce-misuse scenario, both single S-box and full-round S-box fault models are applicable. Conversely, in the Nonce-respect scenario, only fault injection in a single S-box is feasible, necessitating additional computational complexity to guess the faulty tags. On average, 130 tag queries are required to complete the guessing process. Overall, our proposed PDFA method enables comprehensive analysis of the Ascon algorithm. Through specific fault analysis, we demonstrate that the 128-bit key can be fully recovered after either 9.9 full-round fault injections or 263 single-nibble fault injections (calculated as 4.1×64). To conclude, the article addresses potential countermeasures to mitigate the proposeed analysis. One notable suggestion is the introduction of a new S-box to replace the existing one in Ascon, offering enhanced resistance against differential fault analysis.

References

[ADM+10a] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to flip a bit? In *16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece*, pages 235–239. IEEE Computer Society, 2010.

[ADM+10b] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. Single-bit dfa using multiple-byte laser fault injection. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*, pages 113–119. IEEE, 2010.

[BBB+21] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Mustafa Khairallah, Thomas Peyrin, Sumanta Sarkar, and Siang Meng Sim. DEFAULT: Cipher level resistance against differential fault attack. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 124–156. Springer, Heidelberg, December 2021.

[BBKN12] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proc. IEEE*, 100(11):3056–3076, 2012.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 37–51. Springer, Heidelberg, May 1997.

[Ber16] Bernstein. (2016) Cryptographic competitions. [Online]. Website, 2016. https://competitions.cr.yp.to/caesar.html.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, Heidelberg, August 1997.

[CDN23] Bishwajit Chakraborty, Chandranan Dhar, and Mridul Nandi. Exact security analysis of ASCON. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part III*, volume 14440 of *Lecture Notes in Computer Science*, pages 346–369. Springer, 2023.

[CJW10] Nicolas T Courtois, Keith Jackson, and David 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.

[DEMS14] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon. *Submission to the CAESAR competition: http://ascon. iaik. tugraz. at*, 2014.

[DEMS15] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Cryptanalysis of Ascon. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 371–387. Springer, Heidelberg, April 2015.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, July 2021.

[DMNT10] Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, and Assia Triaz. Reproducible single-byte laser fault injection. In *6th Conference on Ph. D. Research in Microelectronics & Electronics*, pages 1–4. IEEE, 2010.

[FJLT13] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance*

in Cryptography, Los Alamitos, CA, USA, August 20, 2013, pages 108–118. IEEE Computer Society, 2013.

[GJMG11] Bertoni Guido, Daemen Joan, P Michaël, and VA Gilles. Cryptographic sponge functions, 2011.

[GT04] Christophe Giraud and Hugues Thiebeauld. A survey on fault attacks. In Jean-Jacques Quisquater, Pierre Paradinas, Yves Deswarte, and Anas Abou El Kalam, editors, *Smart Card Research and Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France*, volume 153 of *IFIP*, pages 159–176. Kluwer/Springer, 2004.

[GWDE17] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Ascon hardware implementations and side-channel evaluation. *Microprocess. Microsystems*, 52:470–479, 2017.

[GWY+19a] Yang Gao, Yongjuan Wang, Qing-jun Yuan, Tao Wang, and Xiangbin Wang. Improvement of differential fault attack based on lightweight ciphers with GFN structure. In Xingming Sun, Zhaoqing Pan, and Elisa Bertino, editors, *Artificial Intelligence and Security - 5th International Conference, ICAIS 2019, New York, NY, USA, July 26-28, 2019, Proceedings, Part II*, volume 11633 of *Lecture Notes in Computer Science*, pages 549–560. Springer, 2019.

[GWY+19b] Yang Gao, Yongjuan Wang, Qing-jun Yuan, Tao Wang, and Xiangbin Wang. Probabilistic analysis of differential fault attack on MIBS. *IEICE Trans. Inf. Syst.*, 102-D(2):299–306, 2019.

[Hem04] Ludger Hemme. A differential fault attack against early rounds of (triple-)DES. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 254–267. Springer, Heidelberg, August 2004.

[Hey02] Howard M. Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.

[Jan23] Amit Jana. Differential fault attack on ascon cipher. *IACR Cryptol. ePrint Arch.*, page 1923, 2023.

[JLM14] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 85–104. Springer, Heidelberg, December 2014.

[JM19] Priyanka Joshi and Bodhisatwa Mazumdar. A subset fault analysis of ASCON. Cryptology ePrint Archive, Report 2019/1370, 2019. https://eprint.iacr.org/2019/1370.

[JP22] Amit Jana and Goutam Paul. Differential fault attack on photon-beetle. In Chip-Hong Chang, Ulrich Rührmair, Debdeep Mukhopadhyay, and Domenic Forte, editors, *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, ASHES 2022, Los Angeles, CA, USA, 11 November 2022*, pages 25–34. ACM, 2022.

[KJA+18] Mehran Mozaffari Kermani, Amir Jalali, Reza Azarderakhsh, Jiafeng Xie, and Kim-Kwang Raymond Choo. Reliable inversion in gf(2^8) with redundant arithmetic for secure error detection of cryptographic architectures. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37(3):696–704, 2018.

[KTAS14] Mehran Mozaffari Kermani, Kai Tian, Reza Azarderakhsh, and Siavash Bayat Sarmadi. Fault-resilient lightweight cryptographic block ciphers for secure embedded systems. *IEEE Embed. Syst. Lett.*, 6(4):89–92, 2014.

[LCFS18] Benjamin Lac, Anne Canteaut, Jacques J. A. Fournier, and Renaud Sirdey. Thwarting fault attacks against lightweight cryptography using SIMD instructions. In *IEEE International Symposium on Circuits and Systems, ISCAS 2018, 27-30 May 2018, Florence, Italy*, pages 1–5. IEEE, 2018.

[LDW17] Zheng Li, Xiaoyang Dong, and Xiaoyun Wang. Conditional cube attack on round-reduced ASCON. *IACR Trans. Symm. Cryptol.*, 2017(1):175–202, 2017.

[LGH22] Shuai Liu, Jie Guan, and Bin Hu. Fault attacks on authenticated encryption modes for GIFT. *IET Inf. Secur.*, 16(1):51–63, 2022.

[LZWW17] Yanbin Li, Guoyan Zhang, Wei Wang, and Meiqin Wang. Cryptanalysis of round-reduced ASCON. *Sci. China Inf. Sci.*, 60(3):38102, 2017.

[MBSTM16] Kerry McKay, Lawrence Bassham, Meltem Sönmez Turan, and Nicky Mouha. Report on lightweight cryptography. Technical report, National Institute of Standards and Technology, 2016.

[NDE22] Marcel Nageler, Christoph Dobraunig, and Maria Eichlseder. Information-combining differential fault attacks on DEFAULT. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 168–191. Springer, Heidelberg, May / June 2022.

[NIS23a] NIST. Lightweight Cryptography Standardization Process: NIST Selects Ascon. Website, 2023. https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon.

[NIS23b] NIST. NIST Selects 'Lightweight Cryptography' Algorithms to Protect Small Devices. Website, 2023. https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptographyalgorithms-protect-small-devices.

[NSS23] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Committing security of ascon: Cryptanalysis on primitive and proof on mode. *IACR Trans. Symmetric Cryptol.*, 2023(4):420–451, 2023.

[PBR15] Roberta Piscitelli, Shivam Bhasin, and Francesco Regazzoni. Fault attacks, injection techniques and tools for simulation. In *10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2015, Napoli, Italy, April 21-23, 2015*, pages 1–6. IEEE, 2015.

[PCM17] Sikhar Patranabis, Abhishek Chakraborty, and Debdeep Mukhopadhyay. Fault tolerant infective countermeasure for AES. *J. Hardw. Syst. Secur.*, 1(1):3–17, 2017.

[PCMC17] Sikhar Patranabis, Abhishek Chakraborty, Debdeep Mukhopadhyay, and Partha Pratim Chakrabarti. Fault space transformation: A generic approach to counter differential fault analysis and differential fault intensity analysis on aes-like block ciphers. *IEEE Trans. Inf. Forensics Secur.*, 12(5):1092–1102, 2017.

[RAD19a] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. FIMA: Fault intensity map analysis. In Ilia Polian and Marc Stöttinger, editors, *COSADE 2019*, volume 11421 of *LNCS*, pages 63–79. Springer, Heidelberg, April 2019.

[RAD19b] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. A statistical fault analysis methodology for the ascon authenticated cipher. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019*, pages 41–50. IEEE, 2019.

[RAD+20a] Keyvan Ramezanpour, Abubakr Abdulgadir, William Diehl, Jens-Peter Kaps, and Paul Ampadu. Active and passive side-channel key recovery attacks on ascon. In *Proc. NIST Lightweight Cryptogr. Workshop*, pages 1–27, 2020.

[RAD20b] Keyvan Ramezanpour, Paul Ampadu, and William Diehl. SCARL: side-channel analysis with reinforcement learning on the ascon authenticated cipher. *CoRR*, abs/2006.03995, 2020.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002.

[RSDT13] Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault model analysis of laser-induced faults in SRAM memory cells. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 89–98. IEEE Computer Society, 2013.

[SAM24] Iftekhar Salam, Janaka Alawatugoda, and Hasindu Madushan. Statistical fault analysis of tinyjambu. *Discover Applied Sciences*, 6(2):1–13, 2024.

[SD17] Niels Samwel and Joan Daemen. DPA on hardware implementations of ascon and keyak. In *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*, pages 415–424. ACM, 2017.

[Sko10] Sergei Skorobogatov. Optical fault masking attacks. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010*, pages 23–29. IEEE Computer Society, 2010.

[SMS20] G Surya, Paolo Maistri, and Sriram Sankaran. Local clock glitching fault injection with application to the ascon cipher. In *2020 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS)*, pages 271–276. IEEE, 2020.

[SOX+21] Md. Iftekhar Salam, Thian Hooi Ooi, Luxin Xue, Wei-Chuen Yau, Josef Pieprzyk, and Raphaël C.-W. Phan. Random differential fault attacks on the lightweight authenticated encryption stream cipher grain-128aead. *IEEE Access*, 9:72568–72586, 2021.

[TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying fault invariant with randomization - A countermeasure for AES against differential fault attacks. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 93–111. Springer, Heidelberg, September 2014.

[Tez16] Cihangir Tezcan. Truncated, impossible, and improbable differential analysis of ascon. Cryptology ePrint Archive, Report 2016/490, 2016. https://eprint.iacr.org/2016/490.

[YKSH23] Shih-Chun You, Markus G. Kuhn, Sumanta Sarkar, and Feng Hao. Low trace-count template attacks on 32-bit implementations of ASCON AEAD. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):344–366, 2023.

[YLW+23] Xiaorui Yu, Fukang Liu, Gaoli Wang, Siwei Sun, and Willi Meier. A closer look at the s-box: Deeper analysis of round-reduced ASCON-HASH. *IACR Cryptol. ePrint Arch.*, page 865, 2023.

Appendices

(The values in Table 12 are expressed in hexadecimal format, with the binary representations provided in brackets.)

Table 12: Differential distribution table of 5-bit S-box in Ascon

α	Correspondence between the output difference and the input value when the input difference (fault value) is fixed							
0x01	β	9(01001)	b(01011)	d(01101)	f(01111)	18(11000)	1a(11010)	1c(11100)	1e(11110)
	m	12,13,16,17	2,3,6,7	10,11,14,15	0,1,4,5	1a,1b,1e,1f	a,b,e,f	18,19,1c,1d	8,9,c,d
0x02	β	11(10001)	13(10011)	15(10101)	17(10111)	19(11001)	1b(11011)	1d(11101)	1f(11111)
	m	14,16,18,1a	4,6,8,a	15,17,19,1b	5,7,9,b	10,12,1c,1e	0,2,c,e	11,13,1d,1f	1,3,d,f
0x03	β	1(00001)	5(00101)	9(01001)	d(01101)	10(10000)	14(10100)	18(11000)	1c(11100)
	m	c,f,1c,1f	d,e,1d,1e	8,b,18,1b	9,a,19,1a	0,3,10,13	1,2,11,12	4,7,14,17	5,6,15,16
0x04	β	6(10110)	e(01110)	16(10110)	1e(11110)				
	m	8,9,c,d, 18,19,1c,1d	a,b,e,f, 1a,1b,1e,1f	2,3,6,7, 12,13,16,17	0,1,4,5, 10,11,14,15				
0x05	β	11(10001)	13(10011)	14(10100)	16(10110)	18(11000)	1a(11010)	1d(11101)	1f(11111)
	m	0,1,4,5	10,11,14,15	a,b,e,f	1a,1b,1e,1f	8,9,c,d	18,19,1c,1d	2,3,6,7	12,13,16,17
0x06	β	1(00001)	3(00011)	5(00101)	7(00111)	9(01001)	b(01011)	d(01101)	f(01111)
	m	3,5	13,15	2,4	12,14	1,7	11,17	0,6	10,16
	β	11(10001)	13(10011)	15(10101)	17(10111)	19(11001)	1b(11011)	1d(11101)	1f(11111)
	m	b,d	1b,1d	a,c	1a,1c	9,f	19,1f	8,e	18,1e
0x07	β	2(00010)	3(00011)	6(00110)	7(00111)	a(01010)	b(01011)	e(01110)	f(01111)
	m	1,6,11,16	9,e,19,1e	0,7,10,17	8,f,18,1f	2,5,12,15	a,d,1a,1d	3,4,13,14	b,c,1b,1c
0x08	β	6(00110)	7(00111)	e(01110)	f(01111)	16(10110)	17(10111)	1e(11110)	1f(11111)
	m	3,b,12,1a	4,c,15,1d	1,9,10,18	6,e,17,1f	5,d,14,1c	2,a,13,1b	7,f,16,1e	0,8,11,19
0x09	β	1(00001)	3(00011)	4(00100)	6(00110)	8(01000)	a(01010)	d(01101)	f(01111)
	m	0,9	11,18	7,e	16,1f	5,c	14,1d	2,b	13,1a
	β	10(10000)	12(10010)	15(10101)	17(10111)	19(11001)	1b(11011)	1c(11100)	1e(11110)
	m	1,8	10,19	6,f	17,1e	4,d	15,1c	3,a	12,1b
0x0a	β	1(00001)	2(00010)	4(00100)	7(00111)	9(01001)	a(01010)	c(01100)	f(01111)
	m	7,d	13,19	2,8	16,1c	5,f	11,1b	0,a	14,1e
	β	11(10001)	12(10010)	14(10100)	17(10111)	19(11001)	1a(11010)	1c(11100)	1f(11111)
	m	3,9	17,1d	6,c	12,18	1,b	15,1f	4,e	10,1a
0x0b	β	2(00010)	3(00011)	6(00110)	7(00111)	a(01010)	b(01011)	e(01110)	f(01111)
	m	15,1e	1,a	4,f	10,1b	6,d	12,19	17,1c	3,8
	β	12(10010)	13(10011)	16(10110)	17(10111)	1a(11010)	1b(11011)	1e(11110)	1f(11111)
	m	11,1a	5,e	0,b	14,1f	2,9	16,1d	13,18	7,c
0x0c	β	1(00001)	8(01000)	10(10000)	19(11001)				
	m	4,6,8,a, 15,17,19,1b	1,3,d,f, 10,12,1c,1e	5,7,9,b, 14,16,18,1a	0,2,c,e, 11,1d,13,1f				
0x0d	β	1(00001)	3(00011)	5(00101)	7(00111)	8(01000)	a(01010)	c(01100)	e(01110)
	m	13,1e	2,f	11,1c	0,d	16,1b	7,a	14,19	5,8
	β	10(10000)	12(10010)	14(10100)	16(10110)	19(11001)	1b(11011)	1d(11101)	1f(11111)
	m	12,1f	3,e	10,1d	1,c	17,1a	6,b	15,18	4,9
0x0e	β	1(00001)	2(00010)	4(00100)	7(00111)	11(10001)	12(10010)	14(10100)	17(10111)
	m	14,16,18,1a	0,2,c,e	11,13,1d,1f	5,7,9,b	10,12,1c,1e	4,6,8,a	15,17,19,1b	1,3,d,f
0x0f	β	8(01000)	9(01001)	c(01100)	d(01101)	18(11000)	19(11001)	1c(11100)	1d(11101)
	m	4,b,17,18	3,c,10,1f	6,9,15,1a	1,e,12,1d	0,f,13,1c	7,8,14,1b	2,d,11,1e	5,a,16,19
0x10	β	9(01001)	b(01011)	18(11000)	1a(11010)				
	m	9,a,d,e, 19,1a,1d,1e	8,b,c,f, 18,1b,1c,1f	1,2,5,6, 11,12,15,16	0,3,4,7, 10,13,14,17				
0x11	β	11(10001)	13(10011)	15(10101)	17(10111)				
	m	2,6,a,e, 13,17,1b,1f	3,7,b,f, 12,16,1a,1e	1,5,9,d, 10,14,18,1c	0,4,8,c, 11,15,19,1d				
0x12	β	1(00001)	3(00011)	5(00101)	7(00111)	9(01001)	b(01011)	d(01101)	f(01111)
	m	2,10	0,12	1,13	3,11	6,14	4,16	5,17	7,15
	β	10(10000)	12(10010)	14(10100)	16(10110)	18(11000)	1a(11010)	1c(11100)	1e(11110)
	m	e,1c	c,1e	d,1f	f,1d	a,18	8,1a	9,1b	b,19
0x13	β	2(00010)	4(00100)	a(01010)	c(01100)				

	m	4,7,8,b, 14,17,18,1b	5,6,9,a, 15,16,19,1a	0,3,c,f, 10,13,1c,1f	1,2,d,e, 11,12,1d,1e				
0x14	β	4(00100)	5(00101)	6(00110)	7(00111)	c(01100)	d(01101)	e(01110)	f(01111)
	m	0,4,10,14	b,f,1b,1f	1,5,11,15	a,e,1a,1e	3,7,13,17	8,c,18,1c	2,6,12,16	9,d,19,1d
0x15	β	5(00101)	7(00111)	9(01001)	b(01011)	11(10001)	13(10011)	1d(11101)	1f(11111)
	m	3,7,12,16	2,6,13,17	0,4,11,15	1,5,10,14	8,c,19,1d	9,d,18,1c	b,f,1a,1e	a,e,1b,1f
0x16	β	10(10000)	11(10001)	12(10010)	13(10011)	14(10100)	15(10101)	16(10110)	17(10111)
	m	f,19	7,11	9,1f	1,17	8,1e	0,16	e,18	6,10
	β	18(11000)	19(11001)	1a(11010)	1b(11011)	1c(11100)	1d(11101)	1e(11110)	1f(11111)
	m	b,1d	3,15	d,1b	5,13	c,1a	4,12	a,1c	2,14
0x17	β	2(00010)	4(00100)	a(01010)	c(01100)	12(10010)	14(10100)	1a(11010)	1c(11100)
	m	a,d,1a,1d	b,c,1b,1c	9,e,19,1e	8,f,18,1f	2,5,12,15	3,4,13,14	1,6,11,16	0,7,10,17
0x18	β	4(00100)	5(00101)	6(00110)	7(00111)	c(01100)	d(01101)	e(01110)	f(01111)
	m	f,17	8,10	6,1e	1,19	4,1c	3,1b	d,15	a,12
	β	14(10100)	15(10101)	16(10110)	17(10111)	1c(11100)	1d(11101)	1e(11110)	1f(11111)
	m	0,18	7,1f	9,11	e,16	b,13	c,14	2,1a	5,1d
0x19	β	3(00011)	6(00110)	8(01000)	d(01101)	10(10000)	15(10101)	1b(11011)	1e(11110)
	m	5,d,14,1c	2,a,13,1b	0,8,11,19	7,f,16,1e	4,c,15,1d	3,b,12,1a	1,9,10,18	6,e,17,1f
0x1a	β	1(00001)	2(00010)	5(00101)	6(00110)	8(01000)	b(01011)	c(01100)	f(01111)
	m	b,11	5,1f	0,1a	e,14	7,1d	9,13	c,16	2,18
	β	11(10001)	12(10010)	15(10101)	16(10110)	18(11000)	1b(11011)	1c(11100)	1f(11111)
	m	f,15	1,1b	4,1e	a,10	3,19	d,17	8,12	6,1c
0x1b	β	2(00010)	3(00011)	4(00100)	5(00101)	a(01010)	b(01011)	c(01100)	d(01101)
	m	9,12	6,1d	3,18	c,17	1,1a	e,15	b,10	4,1f
	β	12(10010)	13(10011)	14(10100)	15(10101)	1a(11010)	1b(11011)	1c(11100)	1d(11101)
	m	d,16	2,19	7,1c	8,13	5,1e	a,11	f,14	0,1b
0x1c	β	1(00001)	3(00011)	8(01000)	a(01010)	10(10000)	12(10010)	19(11001)	1b(11011)
	m	1,e,12,1d	3,c,10,1f	6,9,15,1a	4,b,17,18	2,d,11,1e	0,f,13,1c	5,a,16,19	7,8,14,1b
0x1d	β	3(00011)	5(00101)	8(01000)	e(01110)	10(10000)	16(10110)	1b(11011)	1d(11101)
	m	7,b,16,1a	5,9,14,18	2,e,13,1f	0,c,11,1d	6,a,17,1b	4,8,15,19	3,f,12,1e	1,d,10,1c
0x1e	β	8(01000)	9(01001)	a(01010)	b(01011)	c(01100)	d(01101)	e(01110)	f(01111)
	m	a,14	2,1c	8,16	0,1e	5,1b	d,13	7,19	f,11
	β	18(11000)	19(11001)	1a(11010)	1b(11011)	1c(11100)	1d(11101)	1e(11110)	1f(11111)
	m	e,10	6,18	c,12	4,1a	1,1f	9,17	3,1d	b,15
0x1f	β	2(00010)	3(00011)	4(00100)	5(00101)	12(10010)	13(10011)	14(10100)	15(10101)
	m	3,f,10,1c	4,8,17,1b	1,d,12,1e	6,a,15,19	7,b,14,18	0,c,13,1f	5,9,16,1a	2,e,11,1d