Differential Fault Attack on Ascon Cipher

Amit Jana

Cryptology & Security Research Unit,
R. C. Bose Centre for Cryptology & Security,
Indian Statistical Institute, Kolkata, India,
janaamit001@gmail.com

Abstract. This work investigates the security of the Ascon authenticated encryption scheme in the context of fault attacks, with a specific focus on Differential Fault Analysis (DFA). Motivated by the growing significance of lightweight cryptographic solutions, particularly Ascon, we explore potential vulnerabilities in its design using DFA. By employing a novel approach that combines faulty forgery in the decryption query under two distinct fault models, leveraging bit-flip faults in the first phase and bit-set faults in the second, we successfully recover the complete Ascon key. This study sheds light on the impact of key whitening in the final permutation call and discusses potential threats when this safeguard is absent. Additionally, we consider the implications of injecting multiple bit-flip faults at the S-box input, suggesting alternative strategies for compromising the state space. Our findings contribute valuable insights into the gray-box security landscape of Ascon, emphasizing the need for robust defenses to ensure the integrity and resilience of lightweight cryptographic primitives against diverse fault attacks.

Keywords: CAESAR · NIST· LwC· Authenticated Encryption· Side-channel Attacks· Differential Fault Attack· Faulty Forgery· Ascon

1 Introduction

The Internet of Things (IoT) [1] represents the modern evolution of the Internet, establishing a network of small objects to connect myriad devices across various platforms. Within IoT technology, key components like Radio Frequency Identification (RFID) and Wireless Sensor Networks (WSN) find applications in areas such as traffic control, environmental surveillance, and home automation. Nevertheless, conventional cryptographic methods such as AES (SPN-based block cipher) and SHA (secure hashing algorithm) encounter difficulties in IoT environments due to resource limitations. Lightweight cryptography (LwC) methods have emerged as a solution, addressing challenges related to physical size, processing capacity, memory constraints, and power consumption in embedded systems and sensor networks.

As the Internet of Things (IoT) and cloud computing continue to advance, the prevalence of small-scale devices has increased. These devices often engage in cryptographic operations, making them susceptible to device-specific attacks.

For instance, sensor nodes in Wireless Sensor Networks (WSN) deployed in unattended and potentially hostile environments face exposure to various circumstances. In the context of IoT, this highlights the transition from the traditional black-box attack model, relying on classical cryptanalysis, to the gray-box model, where attackers gain access to additional side-channel information. The Fault Attack (FA) has emerged as a potent threat in the gray-box model, with the Differential Fault Attack (DFA) being a widely employed technique introduced in 1996 by Boneh et al. [2] and demonstrated by Biham and Shamir in [3]. DFA leverages computational errors to extract cryptographic keys. *The NIST Lightweight Cryptography (LwC) Competition, emphasizing side-channel and fault attacks, motivates the study of submissions in light of physical attacks like DFA, forming the main focus of this work. Additionally, the LwC design considerations in [4] stipulate "Side-channel and fault attacks" as a specific requirement, further motivating our research efforts.*

In recent times, there has been a surge in the development of lightweight cryptographic primitives tailored for resource-constrained devices. These encompass a spectrum of algorithms such as Stream ciphers, Block ciphers, Hash functions, Message Authentication Codes (MAC), and Authenticated ciphers. Noteworthy instances among lightweight block ciphers include PRESENT [5], PRINCE [6], LED [7], SKINNY [8], KATAN & KTANTAN [9], and GIFT [10]. Comprehensive analyses and categorizations of lightweight cryptography are provided in [11] and [12], respectively. Moreover, both national (NIST) and international (ISO/IEC) bodies offer methodologies for lightweight cryptography applicable to IoT and RFID devices [4]. In 2013, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was established by a consortium of international cryptologic researchers to promote the development of authenticated encryption schemes. In 2019, Ascon [13] was selected as the preferred choice for lightweight applications in the final portfolio. Later, NIST has initiate a standardization for lightweight cryptography, commencing in 2019, featured 56 accepted submissions in Round-1, whittling down to 32 candidates in Round-2 and culminating in the announcement of 10 finalists, with Ascon emerging as the recent winner.

The cryptographic properties of Ascon render it secure against various linear and differential cryptanalysis methods. However, the Ascon implementation is susceptible to side-channel and fault attacks aimed at key recovery. In this context, Joshi et al. [14] introduced a subset fault analysis (SSFA) attack on Ascon. The authors utilized various fault models with different granularities to uniquely recover the key. For the SSFA attack, they found that it required 85, 680, and 5440 faults to reduce the key space to values between 1 and 2^{64} under different bit-reset fault granularities of 1 bit, byte, and word, respectively. In 2019, Ramezanpour et al. [15] presented a statistical fault attack on Ascon using double-fault injections at a selected pair of S-boxes. In experiments on a software implementation of Ascon, the author demonstrated that between 12.5 to 2500 correct tag values (i.e., ineffective faults) were sufficient for key recovery for fault distributions ranging from highly biased to more uniform.

Jacob et al. [16] demonstrated how to modify ASCON-128a by exploiting the pseudo-random properties of cellular automata to prevent statistical ineffective fault attacks (SIFA) and subset fault analysis (SSFA). Additionally, Surya et al. [17] proposed a local clock glitching fault injection attack on ASCON-128. Furthermore, in [18,19], side-channel attacks were proposed on ASCON to recover its key using 24K power traces on the Artix-7 FPGA.

Performing Differential Fault Analysis (DFA) is highly challenging due to the inability of an attacker to observe both the correct and faulty output for the same input. Some cryptographic schemes assert a certain level of resilience even in misuse scenarios, such as repeated nonces or the release of unverified plaintext. However, unlike nonce-based encryption schemes, the decryption procedure (with a fixed nonce) remains susceptible to DFA. In the case of a sponge authenticated encryption (AE) scheme, conducting DFA necessitates creating a replay in each encryption query. Yet, the use of a unique nonce in each encryption makes generating a faulty state collision challenging to satisfy the replay criterion. However, the impact of this unique nonce becomes irrelevant in the decryption query. Additionally, to orchestrate a replay of the sponge AE in the decryption query, one might need to fabricate a faulty forgery. Here, "faulty forgery" denotes an attacker repetitively making queries by selecting a new tag T' and then injecting faults in the intermediate rounds of the state during the last permutation call until the attacker successfully obtains at least one tag forgery. Several works [20–22] pose a significant risk to nonce-based designs by performing DFA using faulty forgery in the decryption query.

In this study, we present an innovative approach by proposing a differential fault attack on the ASCON AE scheme through faulty forgery in the decryption query. Our attack achieves comprehensive key recovery by leveraging faulty forgery under two distinct fault models. In the first fault model, the attacker successfully retrieves the 64-bit key, and subsequently, the remaining 64-bit key is obtained through the second fault model. The interesting part of this attack is that the attacker does not need to invert any round function operations from the fresh and faulty tag pairs. Instead, a pre-computed table of forgery differences and input patterns from the S-box is constructed, allowing the attacker to recover keys through pattern matching from these tables. This distinctive approach enhances the efficiency and effectiveness of the attack.

Our Contributions. In this paper, in alignment with the NIST LwC requirements, we investigate the vulnerability of the ASCON AEAD primitive to a differential fault attack (DFA). ASCON emerged victorious in both the lightweight cryptography competitions, CAESAR and the NIST-organized competition. Our primary contribution lies in outlining a forging strategy under two different fault models, where we strategically employ faulty forgery in the decryption query. Unlike conventional DFA-based key recovery, where attackers guess the key and perform inverse computations, we introduce an innovative approach. Instead of relying on inverse computations, we pre-compute tables of forgery differences and input patterns from the S-box. This approach allows the attacker to recover keys through efficient pattern matching from these tables. Our combined approach, utilizing both bit-flip faults in the first phase and bit-set faults in the second phase, enables the attacker to uniquely retrieve the key with 1024 faulty queries, requiring 576 bytes of extra memory and exhibiting linear time complexity. This represents a significant advancement in the field of differential fault attacks on lightweight cryptographic primitives.

Paper Outline. The remaining sections of the paper are structured as follows. Section 2 introduces the notation that will be consistently used throughout the paper. In Section 3, we provide a concise overview of the Ascon AE scheme. Sections 4 delve into the specifics of the differential fault attacks, exploring their complexities on Ascon under two distinct fault models. Furthermore, Section 4 includes a detailed discussion on the attack, elucidating the reasons behind selecting the second phase attack under bit-set faults, and highlighting how it facilitates the faulty forgery process and enhances key recovery. The paper concludes with Section 5, summarizing the key findings and contributions.

2 Notations

The following list of notations are used throughout the paper.

- K, N, T, P, C, and A represent the secret key of k bits, nonce of 128 bits, tag of 128 bits, plaintext, ciphertext, and associated data, respectively.
- S, S_r, S_c denote the sponge state of b bits, the r-bit rate part of S, and the c-bit capacity part of S, respectively.
- π, π^a, π^b denote the single-round, a-round, and b-round Ascon permutations, respectively.
- $\pi = \pi_C \circ \pi_{SB} \circ \pi_L$ denotes the composition of three operations: constant addition, substitution, and linear mixing, respectively.
- $x \in \{0,1\}^n$ denotes the bitstring of length n.
- $x_0, x_1, \ldots, x_4$ denote the five 64-bit words of the state S with x_0 and x_4 being the most and least significant words, respectively.
- $x_{i,j}, 0 \leq i \leq 4, 0 \leq j \leq 63$ denotes the j^{th} bit of the word x_i.
- For any input $(x_{0,j}, x_{1,j}, x_{2,j}, x_{3,j}, x_{4,j})$, where $0 \leq j \leq 63$, to the Ascon S-box, $x_{0,j}$ and $x_{4,j}$ denote the most and least significant bits, respectively.
- $(\Delta x_{0j}, \Delta x_{1j}, \ldots, \Delta x_{4,j}) \xrightarrow{\text{S-box}} (\Delta y_{0j}, \Delta y_{1j}, \ldots, \Delta y_{4j})$ denotes the input and output difference of S-box respectively.
- $\oplus$ denotes the bitwise XOR operation.
- $x \ggg i$ denotes the right rotation of the the n-bit word x by i bits.

3 Specification

Ascon cipher, designed by Dobraunig et al. [13] is an authenticated encryption with associated data (AEAD), based on MonkeyDuplex construction [23] with stronger keyed initialization and finalization function. The authenticated encryption (AE) designs of Ascon are parametrized by the key length $k \leq 160$ bits, the rate (data block size) r and internal round numbers a and b. The recommended instances for ASCON authenticated encryption and their parameters are listed in Table 1. The encryption procedure $\mathcal{E}_{k,r,a,b}$ takes as inputs a secret key K with k bits, a nonce N with 128 bits, associated data A of arbitrary length and a plaintext P of arbitrary length. It authenticates both the associated data and the message to produce an output consisting of the authenticated ciphertext C of exactly the same length as the plaintext P as well as an authentication tag T of size 128 bits:

$$\mathcal{E}_{k,r,a,b}(K, N, A, P) = (C, T).$$

The decryption and verification procedure $\mathcal{D}_{k,r,a,b}$ takes as input the key K, nonce N, associated data A, ciphertext C and tag T, and outputs either the plaintext P if the verification of the tag is correct or an error $\perp$ if the verification of the tag fails:

$$\mathcal{D}_{k,r,a,b}(K, N, A, C, T) \in \{P, \perp\}.$$

The ASCON cipher operate on a state S of b bits, where $b = 320$. The state is updated with permutations π^a and π^b. The 320-bit state S is divided into an outer part S_r of r bits and an inner part S_c of c bits, where $r + c = 320$. The main components of the ASCON design are the two 320-bit permutations π^a and π^b. The permutations iteratively apply an SPN-based round transformation π which in turn consists of three operations: *Addition of Constants*, *Substitution Layer*, and *Linear Diffusion Layer*. The ASCON state with S-box operation and the linear diffusion operation is described in Figure 2. The constant addition step π_C adds a round constant to register word x_2 of the state S in each round. The substitution layer π_{SB} updates the state S with 64 parallel applications of the 5-bit S-box $SB(x)$, where $x = (x_{4,j}, x_{3,j}, x_{2,j}, x_{1,j}, x_{0,j})$, $0 \le j \le 63$. The lookup table of SB is given in Table 2. The linear diffusion layer π_L provides diffusion within each 64-bit register word $x_i, i = 0, 1, \ldots, 4$. The row diffusion in

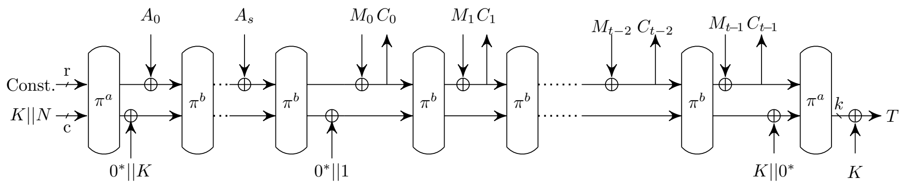


Fig. 1: The Design of ASCON Cipher

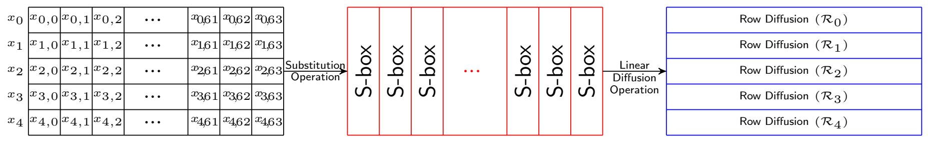

x_0
x_1
x_2
x_3
x_4
$x_{0,0}$ $x_{0,1}$ $x_{0,2}$ ⋯ $x_{0,61}$ $x_{0,62}$ $x_{0,63}$
$x_{4,0}$ $x_{4,1}$ $x_{4,2}$ ⋯ $x_{4,61}$ $x_{4,62}$ $x_{4,63}$
Substitution Operation
S-box
Linear Diffusion Operation
Row Diffusion ($\mathcal{R}_0$)
Row Diffusion ($\mathcal{R}_1$)
Row Diffusion ($\mathcal{R}_2$)
Row Diffusion ($\mathcal{R}_3$)
Row Diffusion ($\mathcal{R}_4$)

Fig. 2: The S-box and Linear operation of ASCON Permutation π

each word are defined as follows:

$$\begin{cases} \mathcal{R}_0: & x_0 = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28), \\ \mathcal{R}_1: & x_1 = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39), \\ \mathcal{R}_2: & x_2 = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6), \\ \mathcal{R}_3: & x_3 = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17), \\ \mathcal{R}_4: & x_4 = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41). \end{cases}$$

For authenticated encryption of Ascon design, the encryption and decryption operations are illustrated in Figure 1. The encryption is partitioned into four phases: initialization, processing of associated data, processing of the plaintext, and finalization. In the "Initialization" phase, the 320-bit initial state of Ascon is formed by the concatenation of an Initial Vector (IV), the secret key K and nonce N. At the Initialization, π^a is applied to the state, and the 128-bit secret key is XORed with the last 128 capacity bits. The "Associated Data" stage absorbs r-bit blocks of associated data into the sponge by XORing them with the rate bits, followed by π^b after each block absorption. In the "Plaintext" stage, plaintext blocks are absorbed, and ciphertext blocks are squeezed out. After processing all plaintext blocks, Ascon enters the "Finalization" stage to generate the authentication tag T. Here, the secret key is XORed with 128 capacity bits (most significant) of the state, and π^a is performed. The tag T consists of the last 128 capacit bits (least significant) of the state XORed with 128 bits of the key K.

Table 1: Recomended Parameters of Ascon AE Schemes

Schemes	Size in Bits				# Rounds	
	Key	Nonce	Tag	Data Blocks	π^a	π^b
Ascon-128	128	128	128	64	12	6
Ascon-128a	128	128	128	128	12	8

4 Differential Fault Attack on Ascon

In this comprehensive attack, our objective is to recover the key of the Ascon cipher by employing two distinct fault setups. To achieve this, we explore two different fault models in this attack.

In the initial phase of the attack, we elucidate how an attacker can generate forgeries (faulty forgery) of Ascon in the decryption query using bit-flip faults. Leveraging the collected forgeries, we successfully reduce the internal state space to 2^{64} and subsequently recover its 64-bit key.

Table 2: The Ascon S-box

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

Moving on to the second phase of the attack, we opt for generating faulty forgery under bit-set faults. This strategic choice enables us to uniquely recover the state, thereby obtaining the key in a unique manner. This multi-phase approach highlights the sophistication of the attack strategy, leveraging different fault models to compromise the security of the Ascon cipher and ultimately retrieve its key.

4.1 Fault Model

In this attack, we consider two different kind of fault models. The first fault model we consider in this attack is to induce precise bit-flip faults in the state. For example, an attacker can induce a precise bit-flip fault to change one bit (at the input of S-box operation) in the internal state during the tag verification of Ascon decryption oracle call. This can be practically achieved because the attacker can use a Laser beam to induce such faults [24–26] with high accuracy (space and time). Further, EM is also a good way of injecting such precise bit-flip faults which does not require any chip de-packaging. Practically, precise bit-level fault injections can be achieved by EM fault injection setup and is being shown in [27].

In the second phase of our attack strategy, we consider the scenario where the attacker can introduce bit-set faults at the input of the S-box operation during the finalization stage's last round in the decryption oracle call. A bit-set fault, when applied to a data bit, causes a change in its value from zero to one, thereby introducing a computation error. Conversely, if the logical value of the data bit was already one before the fault injection, it remains unaltered. Practical experiments conducted on a microcontroller, as documented in [28,29], have successfully demonstrated the induction of bit-set faults using laser beams. Research findings highlight that the occurrence rate of bit-set/reset faults is notably higher than that of bit-flip faults. While the implementation of laser-induced faults requires expensive equipment, it allows for the injection of faults with exceptional precision in both target location and timing, as illustrated in [30].

4.2 Faulty Forgery on Ascon

During the decryption query of the Ascon AEAD, we initiate a faulty forgery by deliberately introducing repeated bit-flip faults at the final Ascon permu-

Table 3: Difference Distribution Table (DDT) of S-box Corresponding to Input Bit Differences

In/out	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
1	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	4	-	4	-	4	-	4	-	4	-	4	-	4
4	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	8	-
8	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4	-	-	-	-	-	-	4	4
1f	-	-	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	4	4	4	4	-		-	-	-	-	-	-	-	-

tation call, precisely before the tag evaluation. Specifically, these bit-flip faults are induced at the last round (before the S-box operation) of the final Ascon permutation denoted as π^a in the decryption query.

To implement this, we randomly select $T^{'} = T \oplus \Delta$ based on the input bit difference in the state. In the context of the round function of the permutation π, if a bit difference is injected (prior to the S-box operation) at any bit position in the j-th column, the resulting output differences from the S-box operation spread to two bit positions, namely $(3, j)$ and $(4, j)$ in the state words x_3 and x_4. In our approach, we specifically focus on the differences in the bit positions $(3, j)$ and $(4, j)$ within the column. This targeted selection is made since differences in other bit positions within the column are deemed irrelevant and do not impact the tag computation.

Let $\delta_{i,j}$ denote the bit differences of the state after the S-box operation, where $0 \leq i \leq 4$ and $0 \leq j \leq 63$. It is evident that the two bit differences at positions $(3, j)$ and $(4, j)$ are $\delta_{3,j} = b_1$ and $\delta_{4,j} = b_2$, where $b_1, b_2 \in \{0, 1\}$. The two 64-bit word differences δ_3 and δ_4 take the form $\delta_3 = 00 \ldots \delta_{3,j} \ldots 0$ and $\delta_4 = 00 \ldots \delta_{4,j} \ldots 0$. Therefore, for each bit-flip fault, we select the Δ difference as $\Delta = \mathcal{R}_3(\delta_3)||\mathcal{R}_4(\delta_4)$. This scenario is visually depicted in Figure 3. Additionally, we summarize the steps to obtain the faulty forgery in Algorithm 1. We estimate the number of required faulty decryption queries to get a successful forgery when an attacker inject a fault (bit-flip) at the specific bit position. The following propositions give the answer and finally, show that 4 faulty queries are sufficient to get one successful tag forgery.

Theorem 1. *For* Ascon *AEAD, let χ denotes the number of faulty queries to collect a successful forgery by injecting falts at the $(i, j)^{th}$ position (before S-box operation) in the last round of the final permutation. Then, $E(\chi) = 2^2$.*

Proof. To make a valid forgery, we have to try for all $b_1||b_2 \in \{00, 01, 10, 11\}$ and make the corresponding $T^{'}$ as $T^{'} = T \oplus \mathcal{R}_3(\delta_3)||\mathcal{R}_4(\delta_4)$. Thus, the probability (p) to get one successful forgery will be $\frac{1}{2^2}$.

Let χ denotes the number of trials to get a success, i.e., a valid forgery. So, for each trial, $\Pr[success] = p$ and $Pr[failure] = q = 1 - p$. Therefore, $\Pr[\chi = j] = q^{j-1} \cdot p$. It shows that χ follows a geometric distribution with probability p. Hence, $E(\chi) = \frac{1}{p} = 2^2$.

Algorithm 1 FAULTY FORGERY BIT DIFFERENCE PATTERN OF ASCON

```
    Input: Associated data $A$, Plaintext $P$, and the target bit position $(i, j)$,    $0 \leq i \leq 4, 0 \leq j \leq 63$ in the state
    Output: The output bit difference patterns $b_2||b_1$ for which faulty forgery    happens
1: Initialize a list $\mathcal{L}[3] = [0x01, 0x02, 0x03]$
2: Make an encryption query $(N, A, P)$ and get $(C, T)$.
3: Choose the last round function $\pi$ at the finalization phase where faults will be injected before the S-box operation
4: for $p = 0$ to $2$ do                                   ▷ In the decryption query
5:     Inject bit-flip fault at the bit position $(i, j)$ before the S-box operation
6:     Compute $\Delta x_3 = (((\mathcal{L}[p]\&0x02) >> 1) << 63 - j)$ and $\Delta x_4 = ((\mathcal{L}[p]\&0x01) << 63 - j)$
7:     Choose the tag difference $\Delta T = \mathcal{R}_3(\Delta x_3)||\mathcal{R}_4(\Delta x_4)$
8:     Make the decryption query as $(N, A, C, T^{'})$, where $T^{'} = T \oplus \Delta T$ and check whether $T^{'}$ is a valid forgery or not? If happens, then return the bit difference pattern $(\mathcal{L}[p]\&0x1)||((\mathcal{L}[p]\&0x2) >> 1)$.
9: return $0||0$.
```

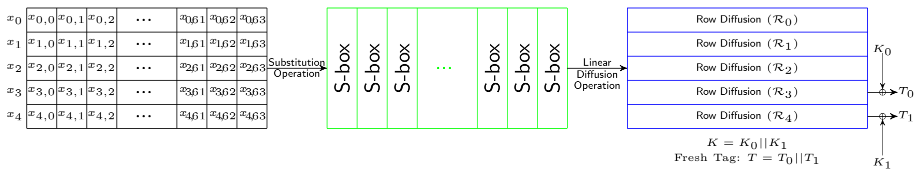
For any bit difference at the input to S-box, there are four possible output difference patterns denoted as $b_2||b_1$: 00, 01, 10, or 11. Another perspective is to consider the differential distribution table (DDT) of ASCON S-box, where a given input difference leads to various output differences. The Table 3 provides the DDT corresponding to input bit differences for the S-box. Analyzing these output differences for a fixed input difference, the initial two LSB bit differences may appear in the forms of different strings. It is important to note that the decryption query does not provide any response for the output difference pattern $b_1||b_2 = 00$, where no difference reflects to the tag part ($T^{'} = T$). This behavior stems from the property of the S-box DDT.

Consequently, the faulty forgery is executed based on the output differences $b_1||b_2 \in \{01, 10, 11\}$, and the Δ difference is crafted accordingly. Therefore, if no successful forgeries occur (corresponding to bit-flip faults) for $b_1||b_2 \in \{01, 10, 11\}$, it is inferred that a forgery must indeed occur for $b_1||b_2 = 00$. This assumption is grounded in the understanding of the differential distribution table (DDT) properties of the ASCON S-box, where the absence of differences in the $b_1||b_2 = 00$ pattern indicates a unique behavior in the decryption response.

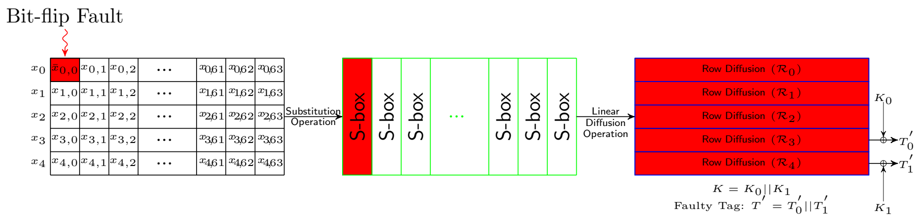
4.3 The First Phase Attack

Notably, when bit faults are injected at a given bit position in the input to S-box for faulty forgery, the attacker gains knowledge of the first two LSB bits of the output difference $b_1||b_2$ after the S-box operation. This implies that the attacker possesses complete information on the S-box input difference while having only partial information on the S-box output difference. To filter out the S-box difference, two approaches are considered.

The first approach involves computing backward from the S-box by guessing the input values of the inverse S-box. This iterative process is repeated for all 64 S-box operations, leading to a reduction in the state space. Following the reduction, the focus is on identifying the state words x_3 and x_4. If either one or both of these words are uniquely determined, the corresponding 64-bit keys can be unequivocally retrieved. However, this approach may prove intricate.


Substitution Operation
S-box
Linear Diffusion Operation
Row Diffusion ($\mathcal{R}_0$)
Row Diffusion ($\mathcal{R}_1$)
Row Diffusion ($\mathcal{R}_2$)
Row Diffusion ($\mathcal{R}_3$)
Row Diffusion ($\mathcal{R}_4$)
K_0
K_1
T_0
T_1
$K = K_0||K_1$
Fresh Tag: $T = T_0||T_1$

(a) Fresh Tag in Encryption Query


Bit-flip Fault
Substitution Operation
S-box
Linear Diffusion Operation
Row Diffusion ($\mathcal{R}_0$)
Row Diffusion ($\mathcal{R}_1$)
Row Diffusion ($\mathcal{R}_2$)
Row Diffusion ($\mathcal{R}_3$)
Row Diffusion ($\mathcal{R}_4$)
K_0
K_1
T_0'
T_1'
$K = K_0||K_1$
Faulty Tag: $T' = T_0'||T_1'$

(b) Faulty Tag in Decryption Query

Fig. 3: An example of Faulty Forgery of Ascon

Hence, the second approach is preferred, offering an alternative strategy for efficiently achieving the same goal. In this approach we filter out the inputs of the S-box in the forward direction and store them in a table. The approach is as follows. Consider the input/output difference of $j^{th}, 0 \leq j \leq 63$ S-box as

$$(\Delta x_{0,j},\Delta x_{1,j},\Delta x_{2,j},\Delta x_{3,j},\Delta x_{4,j}) \xrightarrow{\text{S-box}} (*,*,*,\Delta y_{3,j},\Delta y_{4,j}),$$

where $\Delta_{in} = \Delta x_{0,j}||\Delta x_{1,j}||\Delta x_{2,j}||\Delta x_{3,j}||\Delta x_{4,j}$ is a one bit input difference and $\Delta_{out} = *||*||*||\Delta x_{3,j}||\Delta x_{4,j}$ is the output difference, with $b_1 = \Delta y_{3,j}, b_2 = \Delta y_{4,j}$ being the two LSB output bit differences in the state. Thus, for all inputs $x \in \{0,1\}^5$, we have to compute the output difference $b_2||b_1$ corresponding to each input differences $\Delta_{in} = (0x1 << i), i \in \{0,1,2,3,4\}$ and store them in a table. The faulty forgery scenario at the targeted S-box (j^{th}) is depicted in Figure 4. The evaluation of the table is given in Algorithm 2. Also, for each S-box input value, the first two LSB bit difference pattern $b_2||b_1$ for each 5 different input bit differences are presented in Table 4.

Upon closer examination of Table 4, it becomes evident that each output bit difference pattern of S-box corresponds to exactly two different input values of S-box for each set of five faulty forgery at the S-box input. For instance, the difference pattern 01 01 01 11 11 occurs for two different input values, namely 3 and 7. The pairs of S-box input values sharing the same output bit difference patterns follow the pattern $(z, z+4)$, where $z = 8 \cdot a + b$ and $a, b \in \{0,1,2,3\}$. This observation indicates that any output bit difference pattern corresponding to faulty forgery for each fault at the S-box input effectively reduces the S-box input space to 2. Consequently, for each faulty forgery at the 64 different S-box, the state space is directly reduced from 2^{128} to 2^{64}. This reduction in state space, in turn, diminishes the key space to 2^{64}, rendering it impractical for any type of fault attacks.

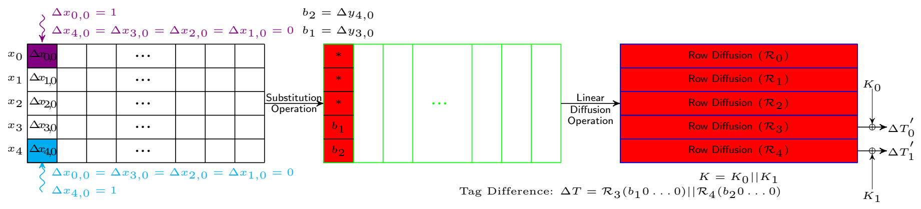

$\Delta x_{0,0} = 1$
$\Delta x_{4,0} = \Delta x_{3,0} = \Delta x_{2,0} = \Delta x_{1,0} = 0$
$b_2 = \Delta y_{4,0}$
$b_1 = \Delta y_{3,0}$
x_0
x_1
x_2
x_3
x_4
$\Delta x_{0,0}$
$\Delta x_{1,0}$
$\Delta x_{2,0}$
$\Delta x_{3,0}$
$\Delta x_{4,0}$
Substitution Operation
$*$
b_1
b_2
Linear Diffusion Operation
Row Diffusion ($\mathcal{R}_0$)
Row Diffusion ($\mathcal{R}_1$)
Row Diffusion ($\mathcal{R}_2$)
Row Diffusion ($\mathcal{R}_3$)
Row Diffusion ($\mathcal{R}_4$)
K_0
$\Delta T_0'$
$\Delta T_1'$
K_1
$\Delta x_{0,0} = \Delta x_{3,0} = \Delta x_{2,0} = \Delta x_{1,0} = 0$
$\Delta x_{4,0} = 1$
$K = K_0 || K_1$
Tag Difference: $\Delta T = \mathcal{R}_3(b_1 0 \dots 0) || \mathcal{R}_4(b_2 0 \dots 0)$

Fig. 4: Example of Faulty Forgeries at 0th Column in the Last Round Finalization Phase

$(0,4)\xrightarrow{\text{S-box}}(4,26)$ 00100 11010 ✗✗✗✗✓	$(1,5)\xrightarrow{\text{S-box}}(11,21)$ 01011 10101 ✗✗✗✗✓	$(2,6)\xrightarrow{\text{S-box}}(31,9)$ 11111 01001 ✗✓✗✗✓	$(3,7)\xrightarrow{\text{S-box}}(20,2)$ 10100 00010 ✗✓✗✗✓
$(8,12)\xrightarrow{\text{S-box}}(27,29)$ 11011 11101 ✓✓✗✗✓	$(9,13)\xrightarrow{\text{S-box}}(5,3)$ 00101 00011 ✗✗✗✗✓	$(10,14)\xrightarrow{\text{S-box}}(8,6)$ 01000 00110 ✗✗✗✗✓	$(11,15)\xrightarrow{\text{S-box}}(18,28)$ 10010 11100 ✗✗✗✗✓
$(16,20)\xrightarrow{\text{S-box}}(30,0)$ 11110 00000 ✗✗✗✗✓	$(17,21)\xrightarrow{\text{S-box}}(19,13)$ 10001 01101 ✗✗✗✓✓	$(18,22)\xrightarrow{\text{S-box}}(7,17)$ 00111 10001 ✗✓✗✓✓	$(19,23)\xrightarrow{\text{S-box}}(11,24)$ 01110 11000 ✗✓✗✗✓
$(24,28)\xrightarrow{\text{S-box}}(16,22)$ 10000 10110 ✓✓✗✗✓	$(25,29)\xrightarrow{\text{S-box}}(12,10)$ 01100 01010 ✓✓✗✗✓	$(26,30)\xrightarrow{\text{S-box}}(1,15)$ 00001 01111 ✓✗✗✗✓	$(27,31)\xrightarrow{\text{S-box}}(25,23)$ 11001 10111 ✓✗✗✗✓

Fig. 5: Ascon S-box Output Pairs Correspond to Input Pairs Which Generates the Same faulty forgery Difference Pattern in Table 4

To accurately recover the 64-bit keys, further exploration is needed based on this reduced 2^{64} state space. For all these pairs of inputs, it must be checked whether the LSB bits of S-box's output are the same or not. We list all these input pairs and their corresponding outputs in Figure 5. From Figure 5, it is evident that the LSB bits of each of the 64 S-box can be uniquely retrieved, meaning that the word x_4 after the S-box operation is uniquely recovered. This final step facilitates the direct retrieval of the 64-bit key by XORing the fresh tag T with the word $\mathcal{R}_4(x_4)$.

The complete process for recovering the 64-bit keys can be summarized as follows. Fix the column ($j^{th}, 0 \leq j \leq 63$ S-box) position in a state and collect the five output bit difference patterns $b_2||b_1$ by performing faulty forgery through injecting bit-flip faults on five input bits to the S-box input. Based on the five output bit difference patterns, search for the input values of the S-box from Table 4, and then retrieve the LSB bit according to Figure 5, which directly provides the actual bit value (at the j^{th} column position) of the word x_4 after the S-box operation. Repeat this procedure for each of the 64 S-boxes to directly retrieve the word x_4 after the S-box operation. Then, apply the Row Mixing operation to x_4 and XORing it with the fresh tag T to uniquely obtain the 64-bit key. This entire process is detailed in Algorithm 3.

Table 4: The Output Bit Difference Pattern of Ascon S-box When Faulty Forgery Happens under Bit-flip Faults

Input	Output Bits Difference Pattern $b_2\|\|b_1$ of each input bit difference					Input	Output Bit Difference Pattern $b_2\|\|b_1$ of each input bit difference				
	$\Delta_{in} = 0x10$	$\Delta_{in} = 0x08$	$\Delta_{in} = 0x04$	$\Delta_{in} = 0x02$	$\Delta_{in} = 0x01$		$\Delta_{in} = 0x10$	$\Delta_{in} = 0x08$	$\Delta_{in} = 0x04$	$\Delta_{in} = 0x02$	$\Delta_{in} = 0x01$
0	01	11	01	11	11	16	01	01	01	10	10
1	00	01	01	11	11	17	00	11	01	10	10
2	00	11	01	11	11	18	00	01	01	10	10
3	01	01	01	11	11	19	01	11	01	10	10
4	01	11	01	11	11	20	01	01	01	10	10
5	00	01	01	11	11	21	00	11	01	10	10
6	00	11	01	11	11	22	00	01	01	10	10
7	01	01	01	11	11	23	01	11	01	10	10
8	11	11	01	11	01	24	11	01	01	10	00
9	10	01	01	11	01	25	10	11	01	10	00
10	10	11	01	11	01	26	10	01	01	10	00
11	11	01	01	11	01	27	11	11	01	10	00
12	11	11	01	11	01	28	11	01	01	10	00
13	10	01	01	11	01	29	10	11	01	10	00
14	10	11	01	11	01	30	10	01	01	10	00
15	11	01	01	11	01	31	11	11	01	10	00

Algorithm 2 Computing the Pre-defined Table of faulty forgery Bit Difference Pattern of Ascon S-box

```
    Input: Ascon S-box $S$
    Output: A Table consisting the output difference patterns $b_2||b_1$ for each input bit differences
 1: Initialize a two dimentional table $\mathcal{T}[32][5]$
 2: for $x = 0$ to 31 do
 3:     for $i = 0$ to 4 do
 4:         $\Delta_{in} = (0x1 << i)$
 5:         $\Delta_{out} = S(x) \oplus S(x \oplus \Delta_{in})$
 6:         $b_1 = ((\Delta_{out}\ \&\ 0x2) >> 1)$
 7:         $b_2 = (\Delta_{out}\ \&\ 0x1)$
 8:         $\mathcal{T}[x][i] = b_2||b_1$
 9: return the table $\mathcal{T}$
```

Algorithm 3 The Last 64-bit Key Recovery of Ascon using Bit-flip faulty forgery

```
    Input: Associated Data $A$ and Plaintext $P$
    Output: 64-bit key of Ascon using faulty forgery
 1: Compute the table $\mathcal{T}$ according to Algoritm 2                      ▷ See Table 4
 2: Make an encryption query $(N, A, P)$ and get $(C, T)$
 3: Choose the last round function $\pi$ at the finalization phase where faults will be injected before
    the S-box operation
 4: Initialize three variables $i = j = l = 0$          ▷ $i \leftarrow$ Row Position, $j \leftarrow$ Column Position
 5: Initialize a 64-bit register $x_4 = 0x0$ and a 64-bit key $K^{'}$       ▷ The LSB word after the S-box
    operation
 6: Initilize a table $\mathcal{G}[64][5]$
 7: for $j = 0$ to 63 do
 8:     $l = 0$
 9:     for $i = 0$ to 4 do
10:         Fix the bit position $(i, j)$ where bit-flip faults will be injected.
11:         Call the Algorithm 1 and store the output bit difference pattern $b_2||b_1$     in the table
    $\mathcal{G}[j][l] = b_2||b_1$
12:         $l = l + 1$
13:     Search for the one input value $x$ of S-box from the table $\mathcal{T}$ according to the output difference
    patterns from the $j^{th}$ row of $\mathcal{G}$                                  ▷ See Table 4
14:     Compute the LSB bit $z$ ($(4, j)^{th}$ bit) from S-box($x$), i.e., $z$ = S-box($x$)&$0x1$
15:     $x_4 = x_4 \oplus (z << 63 - j)$
16: $K^{'} = T_1 \oplus \mathcal{R}_4(x_4)$                             ▷ $T = T_0||T_1, K = K_0||K_1$
17: return $K^{'}$
```

4.4 The Second Phase Attack

In the initial phase of the attack, we demonstrated the retrieval of the 64-bit key through faulty forgery using bit-flip faults. The natural question arises: Can the attacker recover the remaining 64-bit key? Notably, a bit difference pattern obtained through faulty forgery, induced by bit faults at each bit in the S-box, directly reduces the S-box input space to 2. Furthermore, these two values share the third LSB bit in the S-box input (see Figure 5). Therefore, for each S-box in the finalization round, recovering the third LSB input bit of S-box directly reduces the state space from 2^{64} to a unique value. Consequently, the remaining 64-bit key can be directly retrieved. The straightforward approach involves performing faulty forgery by injecting bit-set faults, instead of bit-flip faults, at the third bit position (LSB) before the S-box transformation. The steps to uniquely recover the key are outlined in Algorithm 4.

Algorithm 4 The Initial 64-bit Key Recovery of Ascon using Bit-set faulty forgery

```
    Input: Associated Data $A$ and Plaintext $P$ and the list $\mathcal{G}$ from Algorithm 3
    Output: 64-bit key of Ascon using faulty forgery
 1: Consider the same $(C, T)$ pair of an encryption query $(N, A, P)$ according to Algorithm 3.
 2: Choose the last round function $\pi$ at the finalization phase where faults will be injected before
    the S-box operation during decryption
 3: Initialize three variables $j = l = 0$ and $i = 2$          ▷ $i \leftarrow$ Row Position, $j \leftarrow$ Column Position
 4: Initialize a 64-bit register $x_3 = 0x0$ and a 64-bit key $K^{''}$      ▷ Third LSB word after the S-box
    operation
 5: Initialize a list $\mathcal{B}[64]$
 6: for $j = 0$ to 63 do
 7:     $l = 0$
 8:     Fix the bit position $(i, j)$ where bit-set faults will be injected.
 9:     Compute $\Delta x_3 = (((\mathcal{G}[j][i]\ \&\ 0x02) >> 1) << 63 - j)$ and $\Delta x_4 = ((\mathcal{G}[j][i]\ \&\ 0x01) <<
    63 - j)$
10:     Choose the tag difference $\Delta T = \mathcal{R}_3(\Delta x_3)||\mathcal{R}_4(\Delta x_4)$
11:     Make the decryption query as $(N, A, C, T^{'})$, where $T^{'} = T \oplus \Delta T$
12:     if $T^{'}$ is a valid forgery then
13:         $\mathcal{B}[j] = 0$
14:     else
15:         $\mathcal{B}[j] = 1$
16:     Search for the input value $x$ of S-box from the table $TB$ so that the third LSB bit value
    equals to $\mathcal{B}[j]$                                                ▷ See Table 4
17:     Compute the second LSB bit $z$ ($(3, j)^{th}$ bit) from S-box$(x)$, i.e., $z = ((\text{S-box}(x)\ \&\ 0x2) >> 1)$
18:     $x_3 = x_3 \oplus (z << 63 - j)$
19: $K^{''} = T_0 \oplus \mathcal{R}_3(x_3)$                               ▷ $T = T_0||T_1, K = K_0||K_1$
20: return $K^{''}$
```

For better comprehension, the described algorithm operates as follows. First, target the $j^{th}, 0 \leq j \leq 63$ S-box and select the actual output bit difference pattern from the table $\mathcal{G}$ (see Algorithm 3). Compute the tag difference ΔT by applying linear transformations to the state words x_3, x_4. Execute faulty forgery in the decryption and check whether forgery occurs. If forgery occurs, the third LSB input bit of S-box will be 0; otherwise, it will be 1. This bit value enables the unique recovery of the S-box input. Repeat this procedure to uniquely retrieve all S-box inputs. This ultimately reduces the state space of size 2^{64} to a unique value. Consequently, the first 64-bit key is uniquely recovered.

4.5 Attack Complexity

In the first phase of the attack, the two for loops at step 7 and step 9 in Algorithm 3 are utilized to fix the bit position in the state. For each chosen bit position, the attacker performs faulty forgery (using Algorithm 1) by trying three different bit difference patterns. Consequently, to recover the last 64-key, a total of $64 \cdot 5 \cdot 3 = 960$ bit-flip faults are needed. As for memory usage by Algorithm 3, it requires two 64-bit words, and the tables $\mathcal{T}$, $\mathcal{G}$ take 160 and 320 bytes of memory, respectively. Overall, 496 bytes of memory are needed to implement Algorithm 3.

Similarly, in the second phase of the attack, the attacker must perform 64 faulty forgery operations (using Algorithm 4) to uniquely recover the other 64-key. Additionally, it requires two 64-bit words and a list of 64 bytes to implement

Algorithm 4. Thus, for both attacks combined, the total number of faulty queries will be $1024 = 2^{10}$, and the space complexity will be 576 bytes of extra memory. Furthermore, the time complexity of both algorithms can run in linear time.

4.6 Software Implementation

Practically, we have implemented both our proposed algorithms (Algorithm 3, Algorithm 4) to make faulty forgery and then, retrieved the key uniquely. The implementation was performed on a Intel(R) Core(TM) i5-8250U CPU @1.60GHz machine and the secret key was successfully recovered approximately with the above-mentioned complexities. The source code of this attack is available in [31].

5 Discussion

In this attack, we have demonstrated successful key recovery under the DFA attack for the Ascon design. It is worth noting that, in the Ascon design, the key undergoes XORing both before and after the last permutation call to produce the final tag. This XORing of the key after the last permutation call facilitates direct key recovery. A pertinent question arises: what if the key whitening is not employed in the final permutation call before tag release?

If the key whitening is omitted, the attacker can still recover the key by executing the DFA attack at the last round in the second-to-last permutation call. Whether the XORing of the key occurs separately before or after the permutation call, the recovery of the key using the DFA attack remains feasible. However, it's crucial to consider scenarios where no other key is XORed before or after the permutation call. In such cases, unique state recovery through DFA at the last round in the final permutation call directly leads to key recovery by inverting the permutations accordingly. This poses a potential threat to AE schemes of this type when subjected to DFA attacks in the decryption query.

Moreover, in our DFA attack, we opted for faulty forgery under a bit-set fault scenario in the second phase attack to minimize the number of fault injections. It is conceivable that other strategies, such as injecting multiple bit-flip faults at the S-box input, might offer alternative ways to further reduce the state space and, consequently, diminish the key space. This opens avenues for further exploration on the potential vulnerabilities of AE schemes under diverse fault scenarios.

6 Conclusion

In conclusion, this work has presented a novel and effective approach to conduct a Differential Fault Analysis (DFA) attack on the Ascon authenticated encryption scheme. By incorporating faulty forgery in the decryption query under two distinct fault models, utilizing bit-flip faults in the first phase and bit-set faults in the second, we have successfully demonstrated the complete recovery of the

Ascon key. Notably, the attack leverages the XORing of the key before and after the last permutation call, showcasing its vulnerability in the absence of key whitening in this final stage.

The discussion has highlighted potential threats to AE schemes like Ascon when key whitening is not employed in the final permutation call. Furthermore, considerations about injecting multiple bit-flip faults at the S-box input suggest alternative strategies for reducing the state space and potentially compromising the security of such schemes.

This work contributes insights into the security landscape of Ascon, particularly in the context of fault attacks, and underscores the importance of comprehensive evaluations to fortify cryptographic designs against potential vulnerabilities. Given the rising popularity of Ascon as a lightweight cryptographic solution, our proposed DFA underscores the importance of strengthening its defenses against various types of attacks. This is crucial to maintain the reliability and security of Ascon as a promising cryptographic tool.

References

1. "Internet of things global standards initiative," https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx.
2. D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults (extended abstract)," in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, ser. Lecture Notes in Computer Science, W. Fumy, Ed., vol. 1233. Springer, 1997, pp. 37–51. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_4
3. E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, ser. Lecture Notes in Computer Science, B. S. K. Jr., Ed., vol. 1294. Springer, 1997, pp. 513–525. [Online]. Available: https://doi.org/10.1007/BFb0052259
4. K. A. McKay, L. E. Bassham, M. S. Turan, and N. W. Mouha, "Report on lightweight cryptography," *NIST Interagency/Internal Report (NISTIR),National Institute of Standards and Technology, Gaithersburg, MD, [online]*, 2017. [Online]. Available: https://doi.org/10.6028/NIST.IR.8114
5. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: an ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, 2007, pp. 450–466. [Online]. Available: https://doi.org/10.1007/978-3-540-74735-2_31
6. J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract," in *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December*

2-6, 2012. Proceedings, ser. Lecture Notes in Computer Science, X. Wang and K. Sako, Eds., vol. 7658. Springer, 2012, pp. 208–225. [Online]. Available: https://doi.org/10.1007/978-3-642-34961-4_14

7. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw, "The LED block cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, 2011, pp. 326–341. [Online]. Available: https://doi.org/10.1007/978-3-642-23951-9_22
8. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Robshaw and J. Katz, Eds., vol. 9815. Springer, 2016, pp. 123–153. [Online]. Available: https://doi.org/10.1007/978-3-662-53008-5_5
9. C. D. Cannière, O. Dunkelman, and M. Knezevic, "KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers," in *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds., vol. 5747. Springer, 2009, pp. 272–288. [Online]. Available: https://doi.org/10.1007/978-3-642-04138-9_20
10. S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A small present - towards reaching the limit of lightweight encryption," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, ser. Lecture Notes in Computer Science, W. Fischer and N. Homma, Eds., vol. 10529. Springer, 2017, pp. 321–345. [Online]. Available: https://doi.org/10.1007/978-3-319-66787-4_16
11. G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, and C. Manifavas, "A review of lightweight block ciphers," *J. Cryptogr. Eng.*, vol. 8, no. 2, pp. 141–184, 2018. [Online]. Available: https://doi.org/10.1007/s13389-017-0160-y
12. A. Biryukov and L. Perrin, "State of the art in lightweight symmetric cryptography," *IACR Cryptol. ePrint Arch.*, p. 511, 2017. [Online]. Available: http://eprint.iacr.org/2017/511
13. C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight authenticated encryption and hashing," *J. Cryptol.*, vol. 34, no. 3, p. 33, 2021. [Online]. Available: https://doi.org/10.1007/s00145-021-09398-9
14. P. Joshi and B. Mazumdar, "Ssfa: Subset fault analysis of ascon-128 authenticated cipher," *Microelectronics Reliability*, vol. 123, p. 114155, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271421001219
15. K. Ramezanpour, P. Ampadu, and W. 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.* IEEE, 2019, pp. 41–50. [Online]. Available: https://doi.org/10.1109/HST.2019.8741029
16. J. Jacob, J. Joseph, M. K. Abinshad, A. K. N, and J. Jose, "Prevention of fault attacks in ASCON authenticated cipher using cellular automata," in *Cellular Automata - 14th International Conference on Cellular Automata for Research and Industry, ACRI 2020, Lodz, Poland, December 2-4, 2020, Proceedings*, ser. Lecture Notes in Computer Science, T. M. Gwizdalla, L. Manzoni, G. C. Sirakoulis,

S. Bandini, and K. Podlaski, Eds., vol. 12599. Springer, 2020, pp. 18–25. [Online]. Available: https://doi.org/10.1007/978-3-030-69480-7_3

17. G. Surya, P. Maistri, and S. Sankaran, "Local clock glitching fault injection with application to the ASCON cipher," in *IEEE International Symposium on Smart Electronic Systems, iSES 2020 (Formerly iNiS), Chennai, India, December 14-16, 2020*. IEEE, 2020, pp. 271–276. [Online]. Available: https://doi.org/10.1109/iSES50453.2020.00067
18. K. Ramezanpour, A. Abdulgadir, W. Diehl, J.-P. Kaps, and P. Ampadu, "Active and passive side-channel key recovery attacks on ascon," *NIST Lightweight Cryptography Workshop*, 2020. [Online]. Available: https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/active-passive-recovery-attacks-ascon-lwc2020.pdf
19. K. Ramezanpour, P. Ampadu, and W. Diehl, "SCARL: side-channel analysis with reinforcement learning on the ascon authenticated cipher," *CoRR*, vol. abs/2006.03995, 2020. [Online]. Available: https://arxiv.org/abs/2006.03995
20. D. Saha and D. R. Chowdhury, "Scope: On the side channel vulnerability of releasing unverified plaintexts," in *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, O. Dunkelman and L. Keliher, Eds., vol. 9566. Springer, 2015, pp. 417–438. [Online]. Available: https://doi.org/10.1007/978-3-319-31301-6_24
21. A. Jana, "Differential fault attack on feistel-based sponge AE schemes," *J. Hardw. Syst. Secur.*, vol. 6, no. 1, 2022.
22. A. Jana and G. Paul, "Differential fault attack on photon-beetle," in *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, ASHES 2022, Los Angeles, CA, USA, 11 November 2022*, C. Chang, U. Rührmair, D. Mukhopadhyay, and D. Forte, Eds. ACM, 2022, pp. 25–34. [Online]. Available: https://doi.org/10.1145/3560834.3563824
23. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Duplexing the sponge: Single-pass authenticated encryption and other applications," in *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. Miri and S. Vaudenay, Eds., vol. 7118. Springer, 2011, pp. 320–337. [Online]. Available: https://doi.org/10.1007/978-3-642-28496-0_19
24. M. Agoyan, J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, and A. Tria, "How to flip a bit?" in *16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece*. IEEE Computer Society, 2010, pp. 235–239. [Online]. Available: https://doi.org/10.1109/IOLTS.2010.5560194
25. J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, A. Tria, and T. Vaschalde, "Fault round modification analysis of the advanced encryption standard," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012*. IEEE Computer Society, 2012, pp. 140–145. [Online]. Available: https://doi.org/10.1109/HST.2012.6224334
26. B. Selmke, S. Brummer, J. Heyszl, and G. Sigl, "Precise laser fault injections into 90 nm and 45 nm sram-cells," in *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, ser. Lecture Notes in Computer Science, N. Homma and M. Medwed, Eds., vol. 9514. Springer, 2015, pp. 193–205. [Online]. Available: https://doi.org/10.1007/978-3-319-31271-2_12

27. S. Saha, R. S. Chakraborty, S. S. Nuthakki, Anshul, and D. Mukhopadhyay, "Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability," in *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, ser. Lecture Notes in Computer Science, T. Güneysu and H. Handschuh, Eds., vol. 9293. Springer, 2015, pp. 577–596. [Online]. Available: https://doi.org/10.1007/978-3-662-48324-4_29
28. C. Roscian, A. Sarafianos, J. Dutertre, and A. Tria, "Fault model analysis of laser-induced faults in SRAM memory cells," in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, W. Fischer and J. Schmidt, Eds. IEEE Computer Society, 2013, pp. 89–98. [Online]. Available: https://doi.org/10.1109/FDTC.2013.17
29. A. Menu, J. Dutertre, J. Rigaud, B. Colombier, P. Moëllic, and J. Danger, "Single-bit laser fault model in NOR flash memories: Analysis and exploitation," in *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*. IEEE, 2020, pp. 41–48. [Online]. Available: https://doi.org/10.1109/FDTC51366.2020.00013
30. S. Skorobogatov, "Optical fault masking attacks," in *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010*, L. Breveglieri, M. Joye, I. Koren, D. Naccache, and I. Verbauwhede, Eds. IEEE Computer Society, 2010, pp. 23–29. [Online]. Available: https://doi.org/10.1109/FDTC.2010.18
31. "Unoptimized c-implementations of differential fault attack on ascon cipher," https://github.com/janaamit001/DFA_on_ASCON.git, 2023.