SDDT: An Operation Skip Attack Framework for Bitslice Ciphers—Validated on PIPO

Dongwoo Kang[1], Hanbeom Shin[1], DongHyeon Kim[1], Seokhie Hong[2], and HeeSeok Kim[3]*

[1] School of Cybersecurity, Korea University, Seoul 02841, South Korea
{redr1102, newonetiger, 1s0m0rph1sm}@korea.ac.kr
[2] SmartM2M, Busan 48058, South Korea
shhong@smartm2m.co.kr
[3] Department of Cyber Security, College of Science and Technology, Korea University, Sejong-si 30019, South Korea
80khs@korea.ac.kr

Abstract. Bitslice implementations are widely adopted in lightweight cryptography (LWC) due to their efficiency and inherent resilience to side-channel attacks. However, this paper reveals that their decomposition of the S-box exposes critical vulnerabilities to the operation skip fault model. Unlike data corruption faults suffering from high-entropy diffusion, we identify that skipping bitwise operations induces strongly restricted differential patterns. To exploit this characteristic, we propose the *Skip-induced Difference Distribution Table (SDDT)*, a framework mapping operation omissions to output differences. We validate this approach on the block cipher PIPO through practical experiments, successfully recovering the master key from deeper rounds with fewer faults than previously possible. Our findings underscore the fragility of bitslice designs against precise operation skip faults.

Keywords: Differential Fault Analysis · Operation Skip Fault · Bitslice Cipher · PIPO · Skip-induced Difference Distribution Table.

1 Introduction

With the rapid proliferation of the Internet of Things (IoT), the demand for secure communication in resource-constrained environments has increased significantly [25,18]. To address this need, various lightweight cryptography (LWC) algorithms—such as Serpent [7], RECTANGLE [35], GIFT [3], PIPO [20], and Ascon [12]—have been widely implemented and optimized in bitslice form. This implementation paradigm offers significant advantages: it not only maximizes software throughput via parallel block processing but also inherently mitigates cache timing attacks by ensuring constant-time execution [21]. Furthermore, bitslice structures facilitate the efficient integration of masking countermeasures against Side-Channel Analysis (SCA) [14,22]. Consequently, while

* ✉ Corresponding author

bitslicing strengthens resistance against timing and side-channel attacks, evaluating its resilience against fault injection (FI) [10] attacks has become an active and critical area of research.

Differential Fault Analysis (DFA) [9] stands as the most prominent and widely studied technique among fault injection attacks. However, conventional DFA approaches predominantly rely on specific fault models, such as bit-flips or random byte faults, which present inherent limitations when targeting deeper rounds. In a typical scenario using these models, the injected fault passes through the non-linear operation (S-box). Due to the avalanche effect, a single bit-flip or random fault at the input generates a wide variety of output differences. As the fault propagates through subsequent rounds, this variety increases rapidly, leading to full diffusion where the difference becomes indistinguishable from random data. This phenomenon makes it mathematically intractable to trace the fault back to the source, thereby restricting conventional DFA to the final few rounds of the cipher [33,36,1,15].

In contrast, this paper introduces a novel perspective by exploiting the operation skip fault model [4], specifically targeting the structural nature of bitslice implementations. Unlike Look-Up Table (LUT) based implementations, bitslice designs decompose the S-box into a sequence of bitwise logical operations (e.g., `AND, XOR`). Crucially, we identify that skipping a single instruction is not a random error but a specific logical omission, which causes the S-box to generate strongly restricted output differences. Unlike standard fault models that induce high-entropy diffusion, the operation skip fault model limits the possible output differences to a very small and specific subset, significantly reducing the entropy of the fault propagation.

Leveraging this restricted differential characteristic, we demonstrate that an adversary can trace faults through deeper rounds, overcoming the fast diffusion that hinders standard DFA. To systematically analyze this behavior, we propose the *Skip-induced Difference Distribution Table (SDDT)*, a novel analytical framework that maps skipped logical operations to their corresponding differential properties. We applied this framework to the block cipher PIPO, successfully recovering the master key by injecting fewer faults into deeper rounds than those accessible by previous methods, as summarized in Table 1. Our experimental results confirm that the SDDT-based approach is not only theoretically sound but also practically feasible in real-world environments.

Our main contributions are summarized as follows:

- **Proposal of the Skip-induced Difference Distribution Table (SDDT):** We define the SDDT, a novel analytical tool that systematically maps the structural omission of bitwise logical operations to restricted output difference patterns. Unlike standard DDTs based on input differences, the SDDT characterizes the deterministic nature of operation skips, enabling enhanced differential analysis.
- **Establishment of an SDDT-based DFA Framework:** We propose a comprehensive DFA framework that exploits the low-entropy characteristics identified by the SDDT to efficiently filter wrong key candidates. This

methodology generalizes the attack process, enabling adversaries to trace faults through the diffusion layer where traditional models often fail due to high entropy.

– **Experimental Validation of Improved DFA on PIPO:** Leveraging the proposed SDDT, we successfully recovered the master key by injecting fewer faults into the round preceding the target, achieving a deeper attack depth than the previous studies listed in Table 1. This capability is significant as targeting deeper rounds substantially increases the cost and overhead of standard redundancy-based countermeasures [17,4].

Table 1. Comparison of DFA attacks on PIPO.

Target Cipher	Reference	Fault Model	Fault Loc.	# Faults
PIPO-64/128	[24]	Bit-flip	Round 12 and 13	64
PIPO-64/256	[24]	Bit-flip	Round 16 and 17	128
PIPO-64/128	[23]	Random Byte	Round 12 and 13	≤ 32
PIPO-64/128	[34]	Random Byte	Round 13 and Key schedule	$4^{\dagger}$
PIPO-64/128	**Ours**	**Operation Skip**	**Round 11 and 12**	$\mathbf{\leq 10}$
PIPO-64/256	**Ours**	**Operation Skip**	**Round 15 and 16**	$\mathbf{\leq 20}$

† This attack reduces the key space to 2^{14}, whereas others identify the unique key.
* PIPO-64/128 employs 13 rounds, whereas PIPO-64/256 employs 17 rounds.
** Fault location denotes the injection round required to recover the last round key.
*** The required number of faults is evaluated based solely on valid correct-faulty ciphertext pairs. We demonstrated the feasibility of our attack through real-world experiments, achieving a fault injection success rate of 98.85%. Under these conditions, the master key of PIPO-64/128 was successfully recovered using ≤ 12 faults.

The remainder of this paper is organized as follows. Section 2 provides the preliminaries on fault injection attacks, bitslice implementations, the DDT, and the target cipher PIPO. Section 3 details the proposed SDDT and the analysis methodology based on SDDT. Section 4 describes the full attack scenario and key recovery algorithm on PIPO. Section 5 demonstrates the feasibility of the proposed attack via simulation. Section 6 presents the experimental results, and Section 7 concludes the paper.

2 Preliminaries

2.1 Fault Injection Attacks

Fault Injection (FI) attacks are a prominent class of active physical attacks that intentionally induce malfunctions in cryptographic devices to compromise their security [31,4]. Unlike passive side-channel attacks that merely observe physical leakages, FI attacks apply external physical stress—such as manipulating the voltage, clock signal, or targeting the chip with EM pulses or laser beams—to the device. These disturbances cause the cryptographic algorithm to produce faulty results, which can be analyzed to extract secret information, such as the private key. Injection techniques are generally categorized based on the level of physical access required: non-invasive methods include clock glitching, which violates flip-flop setup times, and voltage glitching, which lowers the power supply; semi-invasive methods include EM pulse injection, which induces transient currents and bit flips, and laser injection, which targets specific transistors to disrupt logic. For systematic security analysis, these effects are abstracted into logical fault models that define how disturbances manifest within the algorithm's logic.

- **Bit-flip Model** [9,32]. This is the most standard model, assuming that a specific bit in an intermediate state is flipped (i.e., $0 \rightarrow 1$ or $1 \rightarrow 0$). This model typically represents errors caused by laser or moderate EM injection.
- **Random Byte Model** [30,5,24]. When an external disturbance, such as an electromagnetic pulse, affects a wider data path (e.g., an 8-bit bus), the value of a specific byte is assumed to be corrupted to an unknown random value. This model is frequently adopted when precise bit-level control is infeasible.
- **Stuck-at Model** [9,11,16]. Often observed in EMFI or optical attacks, this model assumes that a specific bit or a set of bits is fixed to a constant value (0 or 1), regardless of the original data. This is also referred to as a unidirectional fault.
- **Operation Skip Model** [2,26,27,29]. This model assumes that the processor bypasses the execution of a specific assembly instruction (acting as a `NOP`). In our analysis, this physical fault is interpreted as the omission of the corresponding logical operation within the cryptographic algorithm.
- **Operation Replacement Model** [13]. Instead of skipping an instruction, this model assumes that the processor executes a different, valid assembly instruction. This disturbance is modeled as the intended logical operation being replaced by a different operation. This model is complex but realistic in environments where instruction fetch or decode stages are targeted.

Among various analysis techniques, Differential Fault Analysis (DFA) [9] is one of the most prominent approaches for exploiting these induced errors. DFA relies on comparing the correct ciphertext with the faulty ciphertext. By analyzing the difference between these two outputs, the adversary can recover the secret key with a small number of faults.

2.2 Bitslice Implementation

The concept of bitslice implementation was introduced by Eli Biham in 1997 [6] as a novel software technique to optimize the performance of the Data Encryption Standard (DES) [28]. The primary motivation behind its proposal was to address significant performance bottlenecks in conventional software implementations of block ciphers on general-purpose processors. These bottlenecks stemmed from inefficient bit-level permutations and slow memory accesses required for S-box Look-Up Tables (LUTs), which were ill-suited for standard processor architectures.

Bitslice implementations offer compelling advantages in terms of both performance and security. Architecturally, bitslicing exploits data-level parallelism to achieve high throughput. By utilizing wide registers available in modern CPUs—such as standard 32-bit/64-bit registers or specialized SIMD (Single Instruction, Multiple Data) vectors—a bitslice implementation can process multiple independent data blocks simultaneously.

Furthermore, from a security perspective, bitslicing provides an effective countermeasure against cache-timing side-channel attacks, while significantly reducing the overhead of masking countermeasures. Conventional implementations relying on LUTs exhibit data-dependent memory access patterns, leading to variations in execution time that can leak secret information through the CPU cache. In contrast, bitslice implementations rely solely on sequences of boolean operations, ensuring that the execution time remains independent of the secret data processed [21]. Moreover, this boolean decomposition significantly reduces the overhead of high-order masking [14]. Unlike LUT-based approaches that require computationally expensive re-computations to secure S-boxes, bitslice designs allow linear operations (e.g., `XOR`) to be masked trivially, requiring complex masking logic only for the few non-linear gates.

The fundamental design principle of bitslicing involves a bit-level orthogonal representation of data. Conventionally, data blocks are stored within processor registers in their standard, contiguous format. In contrast, in a bitslice representation, as illustrated in Fig. 1, these data blocks are "sliced," and their constituent bits are decomposed and distributed across multiple registers.

For instance, on an architecture with a register size of W, the bitslice implementation processes W data blocks in parallel. In this organization, the register R_i contains the i-th bit from all data blocks. Accordingly, complex cryptographic non-linear operations are reformulated as sequences of fundamental bitwise logical operations (`AND`, `OR`, `XOR`, `NOT`). A single bitwise operation among the registers operates on W independent data bits simultaneously.

2.3 Difference Distribution Table

The Difference Distribution Table (DDT) is a fundamental statistical tool used to analyze the properties of an S-box in the context of differential cryptanalysis [8]. It quantifies the probability of an input difference propagating to a specific output difference through the non-linear substitution layer.

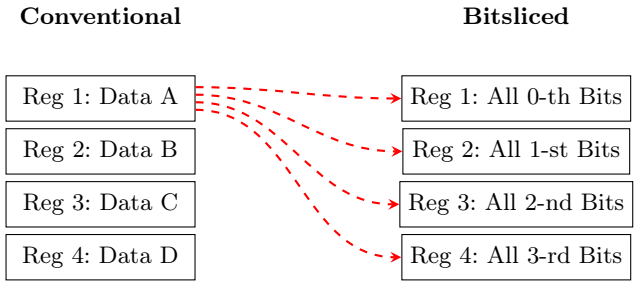

Fig. 1. A conceptual comparison between conventional data representation and bitslice representation. For clarity, the distribution of bits from Data B, C, and D is omitted; only the transposition of Data A is illustrated. In the conventional approach, the S-box is implemented utilizing a LUT for each data block. In contrast, in the bitslice representation, each register stores the i-th bit of the data blocks. Consequently, the S-box is computed as a sequence of logical operations applied across these registers.

Definition and Generation of DDT For an n-bit S-box $S(x)$, the DDT is a $2^n \times 2^n$ matrix where each entry $\mathrm{DDT}[\Delta x][\Delta y]$ characterizes the differential behavior of the S-box. Specifically, it represents the number of input pairs with an input difference Δx that result in an output difference Δy, defined as:

$$\mathrm{DDT}[\Delta x][\Delta y] = \#\{x \in \{0,1\}^n \mid S(x) \oplus S(x \oplus \Delta x) = \Delta y\} \tag{1}$$

where $\Delta x \in \{0,1\}^n$ is the input difference and $\Delta y \in \{0,1\}^n$ is the output difference. A higher value in the table indicates a higher probability that the difference Δx propagates to Δy. To generate this matrix, one iterates over all possible inputs x and input differences Δx, computes the resulting output difference Δy, and increments the corresponding counter in the table. The detailed construction of the DDT is described in **Algorithm 1**.

Algorithm 1 Generation of DDT

Require: An n-bit S-box S
Ensure: DDT of size $2^n \times 2^n$
$\mathrm{DDT}[2^n][2^n] \leftarrow 0$ ▷ Initialize matrix with zeros
for $\Delta x = 0$ to $2^n - 1$ **do** ▷ Iterate over all possible Δx
  for $x = 0$ to $2^n - 1$ **do** ▷ Iterate over all possible x
    $\Delta y \leftarrow S(x) \oplus S(x \oplus \Delta x)$
    $\mathrm{DDT}[\Delta x][\Delta y] \leftarrow \mathrm{DDT}[\Delta x][\Delta y] + 1$ ▷ Increment the counter for Δy
  end for
end for
return DDT

Intermediate Value Filtering based on DDT property In the context of DFA, the differential property plays a crucial role in filtering the internal state. When the adversary induces a fault resulting in an input difference Δx to the S-box and observes an output difference Δy, the possible values of the intermediate state x such that satisfy Eq. (2) are limited. This reduction is critical because the round key can be derived from x and the known ciphertext.

$$\Delta y = S(x) \oplus S(x \oplus \Delta x). \tag{2}$$

In our case on PIPO, input difference Δx is not uniquely determined but restricted to a possible input difference set ΔX. We generalize the filtering process as formulated in Eq. (3) to accommodate this set. Let $\mathcal{X}_{\Delta X, \Delta y}$ be the set of possible input candidates for x given ΔX and Δy:

$$\mathcal{X}_{\Delta X, \Delta y} = \{x \in \{0,1\}^n \mid \exists \Delta x \in \Delta X \text{ s.t. } S(x) \oplus S(x \oplus \Delta x) = \Delta y\}. \tag{3}$$

The procedure to filter the candidate $\mathcal{X}_{\Delta X, \Delta y}$ is outlined in **Algorithm 2**.

Algorithm 2 Finding Candidate Values based on DDT property

Require: An n-bit S-box $S(x)$
Require: Set of input differences ΔX
Require: Observed output difference $\Delta y \in \{0,1\}^n$
Ensure: Set of candidate inputs $\mathcal{X}_{\Delta X, \Delta y}$

```
X_{ΔX,Δy} ← ∅                                   ▷ Initialize empty set
for each guess x ∈ {0,1}^n do                   ▷ Iterate over all possible x
    for all Δx ∈ ΔX do                          ▷ Iterate over all possible Δx
        δ ← S(x) ⊕ S(x ⊕ Δx)
        if δ == Δy then                         ▷ Filter x s.t. satisfies the given differential
            X_{ΔX,Δy} ← X_{ΔX,Δy} ∪ {x}
            break
        end if
    end for
end for
return X_{ΔX,Δy}
```

2.4 Bitslice Block Cipher PIPO

PIPO is a lightweight block cipher proposed by Kim et al. at ICISC 2020 [20]. It is specifically designed to provide optimal performance on 8-bit AVR microcontrollers. The cipher employs a bitslice-oriented structure, allowing the non-linear substitution layer to be executed using only bitwise logical operations without LUTs.

Table 2. Example of intermediate value filtering for PIPO.

Δy	Candidate Values $\mathcal{X}_{\Delta X,\Delta y}$
0x01	0x02, 0x0C, 0x2A, 0x34, 0x42, 0x4C, 0x52, 0x5C, 0x6A, 0x74, 0x84, 0x8A
0x02	0x41, 0x47, 0x49, 0x4F, 0x51, 0x57, 0x59, 0x5F, 0x81, 0x87, 0x89, 0x8F, 0x91, 0x97, 0x99, 0x9F
0x03	0x26, 0x38, 0x90, 0x9E, 0xA0, 0xBE, 0xC0, 0xDE, 0xE0, 0xEE, 0xF0, 0xFE
0x05	0x7B, 0x7D, 0xAB, 0xAD
0x06	0x60, 0x66, 0x68, 0x6E, 0x70, 0x76, 0x78, 0x7E, 0xA0, 0xA6, 0xA8, 0xAE, 0xB0, 0xB6, 0xB8, 0xBE
⋮	

The results are derived from the specific input difference set $\Delta X = \{$0x46, 0x5E, 0xC6, 0xDE, 0x4E, 0x56, 0xCE, 0xD6$\}$. Due to space constraints, only a subset of output differences is displayed. The average cardinality of the candidate set $|\mathcal{X}_{\Delta X,\Delta y}|$ is approximately 8.33. These findings form the basis of the attack proposed in the subsequent sections.

Design Specification PIPO supports 64-bit block and key sizes of 128 and 256 bits, employing 13 and 17 rounds, respectively. The internal state is represented as eight 8-bit registers, denoted as $(R[0], R[1], \ldots, R[7])$, where $R[0]$ corresponds to the least significant byte. The round function consists of three operations: S-layer, R-layer, and Key Addition.

- **S-layer:** A specific 8-bit S-box is applied to each vertical column, providing inter-register confusion and non-linearity.
- **R-layer:** Intra-register diffusion is performed via rotations of the registers.
- **Key Addition:** The round key RK_r is XORed to the state.

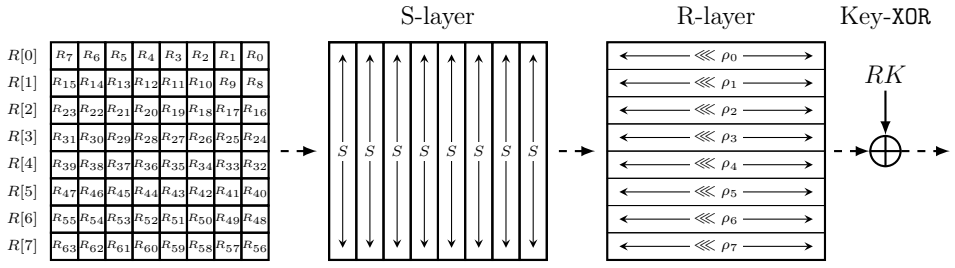

Fig. 2. Structure of the PIPO block cipher round function.

Detailed Specification The bitslice implementation of PIPO is characterized by its S-box design and rotation constants, as defined in the specification; the structural layout is depicted in Fig. 2.

S-layer The PIPO S-box maps 8 input bits to 8 output bits using a sequence of logic gates (`AND`, `OR`, `XOR`, `NOT`). This structure allows the S-box to be computed in parallel for all 8-bit positions within the registers. The reference C implementation using bitslice logic is provided in Fig. 3.

```
void sbox(u8 *X)
{
        u8 T[3] = { 0, };
        //(MSB: x[7], LSB: x[0])
        // Input: x[7], x[6], x[5], x[4], x[3], x[2], x[1], x[0]
        //S5_1
        X[5] ^= (X[7] & X[6]); (1)
        X[4] ^= (X[3] & X[5]); (2)
        X[7] ^= X[4]; (3)
        X[6] ^= X[3]; (4)
        X[3] ^= (X[4] | X[5]); (5)
        X[5] ^= X[7]; (6)
        X[4] ^= (X[5] & X[6]); (7)
        //S3
        X[2] ^= X[1] & X[0]; (8)
        X[0] ^= X[2] | X[1]; (9)
        X[1] ^= X[2] | X[0]; (10)
        X[2] = ~X[2]; (11)
        // Extend XOR
        X[7] ^= X[1]; (12) X[3] ^= X[2]; (13) X[4] ^= X[0]; (14)
        //S5_2
        T[0] = X[7]; (15) T[1] = X[3]; (16) T[2] = X[4]; (17)
        X[6] ^= (T[0] & X[5]); (18)
        T[0] ^= X[6]; (19)
        X[6] ^= (T[2] | T[1]); (20)
        T[1] ^= X[5]; (21)
        X[5] ^= (X[6] | T[2]); (22)
        T[2] ^= (T[1] & T[0]); (23)
        // Truncate XOR and bit change
        X[2] ^= T[0]; T[0] = X[1] ^ T[2]; X[1] = X[0]^T[1];
        X[0] = X[7]; X[7] = T[0];
        T[1] = X[3]; X[3] = X[6]; X[6] = T[1];
        T[2] = X[4]; X[4] = X[5]; X[5] = T[2];
        // Output: x[7], x[6], x[5], x[4], x[3], x[2], x[1], x[0]
}
```

Fig. 3. Reference C implementation of the PIPO S-box using bitslice logic. The indices in red parentheses denote the target operation units for the skip fault model.

For our analysis, we adopt an abstraction where each statement separated by a semicolon (;) in the code is treated as a single skip unit. While a single high-level statement may be compiled into multiple assembly instructions, this approach facilitates a systematic analysis of logical operation skips. By defining the skip unit at this logical level, we significantly reduce the complexity of the fault space compared to an instruction-level analysis, while still capturing the core vulnerabilities inherent to the bitslice structure. Moreover, the operations within the `Truncate XOR and bit change` section are excluded from the analysis scope. We assume that these operations are likely to be executed simultaneously or are too tightly coupled to be skipped individually in a realistic attack scenario.

R-layer The linear layer mixes the bits within each register using simple left rotations. In a bitslice architecture, standard bit permutations across different S-boxes translate directly to word-wise cyclic shifts. This design choice guarantees that the linear diffusion layer incurs minimal computational overhead, as circular rotations are highly optimized and often execute in a single clock cycle on most modern microprocessors. For each register R_i $(i = 0 \dots 7)$, the update function is defined as:

$$R_i \leftarrow (R_i \lll \rho_i), \qquad \text{where } (\rho_0, \dots, \rho_7) = (0, 7, 4, 3, 6, 5, 1, 2). \tag{4}$$

Key Schedule The key schedule for PIPO is designed to be extremely lightweight. For PIPO-64/128, the 128-bit master key is divided into two 64-bit halves, $K = K^1 \,||\, K^0$. The round key RK_r for the r-th round is derived as:

$$RK_r = K^{r \pmod 2} \oplus c_r \tag{5}$$

where $c_r = r$ is the round constant.

Similarly, for PIPO-64/256, the 256-bit master key is divided into four 64-bit parts, denoted as $K = K^3 \,||\, K^2 \,||\, K^1 \,||\, K^0$. The round key is generated using a modulo-4 operation:

$$RK_r = K^{r \pmod 4} \oplus c_r. \tag{6}$$

This simple generation strategy avoids the overhead of complex key expansion algorithms in both variants. From a cryptanalytic perspective, this straightforward relationship implies that recovering a few round keys is directly equivalent to recovering parts of the master key, greatly simplifying the final key extraction phase of our attack.

2.5 Notations

Table 3 summarizes the notations used throughout this paper. To facilitate readability, the notations used in Fig. 4 are defined in Section 4.2 in detail.

Table 3: Summary of notations used in this paper.

Symbol	Description
$\oplus$	Bitwise XOR operation
$\lll \rho$	Circular left shift (rotation) by ρ bits
$x \in \{0,1\}^n$	An n-bit vector x
K	Master key (128-bit or 256-bit)
RK_r	Round key for the r-th round (64-bit)
R_i	The i-th 8-bit register of the state ($0 \leq i \leq 7$)
$S(x)$	S-box function
SL	S-layer of PIPO
RL	R-layer of PIPO
Δx	Input difference of the S-box ($\Delta x = x \oplus x'$)
Δy	Output difference of the S-box ($\Delta y = S(x) \oplus S(x')$)
DDT	Difference Distribution Table
E_i	Fault model representing the skip of the i-th operation
$S_{E_i}(x)$	Faulty S-box function with the i-th operation skipped
SL_{E_i}	Faulty S-layer with the i-th operation skipped
SDDT	Skip-induced Difference Distribution Table
C	Correct ciphertext
C_{E_i}	Faulty ciphertext induced by fault E_i
$\mathcal{A}_r, \mathcal{B}_r, \mathcal{C}_r$	64-bit state differences in round r
$\mathbb{D}_i$	Set of possible 8-bit output differences from E_i
$\mathbb{D}_i^{RL}$	Expanded set of possible 8-bit differences through RL
$\mathbb{D}_{\text{impossible}}$	Set of impossible output differences

In this paper, we distinguish between *instructions* and *operations* for clarity. An instruction refers to a single machine-level instruction defined by the target instruction set architecture (ISA), while an operation denotes a logical computation unit at the algorithmic level. An operation may consist of one or more instructions. Since the physical instruction skip can be modeled to a logical operation skip, we interprete the fault behavior at the algorithmic level.

3 Skip-induced Difference Distribution Table

In this section, we propose a novel attack method tailored for the operation skip fault model. This model is reasonable since the S-box of the bitslice cipher is

implemented as a sequence of bitwise logical operations. We introduce the *Skip-induced Difference Distribution Table* (SDDT), which characterizes the behavior of the S-box under operation skip faults. Furthermore, we describe how to utilize this table to filter candidate values for key recovery.

3.1 Definition and Generation of SDDT

The conventional DDT analyzes the propagation of input differences to output differences. However, in our fault model, the difference is not caused by the input difference but by the omission of a specific operation during the execution. To capture this feature, we define a new statistical table.

Let $S(x)$ be the original S-box function for an input $x \in \{0,1\}^n$. We define E_i as the fault event where the i-th operation of the S-box sequence is skipped. The resulting faulty S-box function is denoted as $S_{E_i}(x)$. The output difference Δy, induced by the fault E_i, can be expressed as:

$$\Delta y = S(x) \oplus S_{E_i}(x). \tag{7}$$

We define the SDDT as a table that records the frequency of occurrence of each output difference Δy for a given fault E_i and for all possible x. Formally, the entry SDDT$[i][\Delta y]$ is defined as:

$$\text{SDDT}[i][\Delta y] = \#\{x \in \{0,1\}^n \mid S(x) \oplus S_{E_i}(x) = \Delta y\}, \tag{8}$$

where # denotes the cardinality of the set. Unlike the standard DDT, the rows of the SDDT correspond to the operation skip index i; the specific operations corresponding to these indices are illustrated in Fig. 3 for the case on PIPO. The procedure to construct the SDDT is described in **Algorithm 3**.

Algorithm 3 Generation of SDDT

Require: An n-bit S-box $S(x)$
Require: Set of skip faults $\{E_1, \ldots, E_N\}$
Ensure: SDDT of size $N \times 2^n$ ▷ N denotes the number of operations of S-box
1: SDDT$[N][2^n] \leftarrow 0$ ▷ Initialize matrix with zeros
2: **for** each operation index i from 1 to N **do** ▷ Iterate over all operation indices
3: **for** $x = 0$ to $2^n - 1$ **do** ▷ Iterate over all possible x
4: $\Delta y \leftarrow S(x) \oplus S_{E_i}(x)$
5: SDDT$[i][\Delta y] \leftarrow$ SDDT$[i][\Delta y] + 1$ ▷ Increment the counter for Δy
6: **end for**
7: **end for**
8: **return** SDDT

The complete SDDT for the PIPO S-box is provided in Table 4. We observe that except for the 11-th operation (`NOT`), all operation skips can induce a zero output difference. This is due to the inherent property of the `NOT` operation, where the output is always different from its input.

Op #	Output Differences ΔY_i (Count)
1	0x00(192), 0x22(4), 0x25(8), 0x26(2), 0x36(2), 0x3A(4), 0x64(2), 0x6C(2), 0xAE(2), 0xBD(8), 0xBE(2), 0xE0(4), 0xE3(4), 0xEB(4), 0xF3(4), 0xF4(2), 0xF8(4), 0xFB(4), 0xFC(2)
2	0x00(192), 0x17(8), 0x37(8), 0x8B(8), 0x97(8), 0x9B(8), 0xA3(4), 0xAB(4), 0xAF(8), 0xB3(4), 0xBB(4)
3	0x00(128), 0x0B(16), 0x17(16), 0x1B(16), 0x23(8), 0x2B(8), 0x33(8), 0x3B(8), 0x3F(16), 0x97(16), 0xA7(16)
4	0x00(128), 0x0C(16), 0x1C(16), 0x24(16), 0x34(16), 0x8C(16), 0x9C(16), 0xAC(16), 0xBC(16)
5	0x00(64), 0x42(48), 0x5A(48), 0xC2(48), 0xDA(48)
6	0x00(128), 0x0E(8), 0x12(32), 0x1E(8), 0x22(16), 0x26(8), 0x36(8), 0x3A(16), 0x8E(8), 0x9E(8), 0xAE(8), 0xBE(8)
7	0x00(192), 0xA0(16), 0xA8(16), 0xB0(16), 0xB8(16)
8	0x00(192), 0x43(8), 0x47(8), 0x4F(8), 0x5B(8), 0xC3(8), 0xC7(8), 0xCF(8), 0xDB(8)
9	0x00(64), 0x23(8), 0x2B(8), 0x2F(8), 0x33(8), 0x3B(8), 0x3F(8), 0xA2(32), 0xA7(8), 0xAA(32), 0xB2(32), 0xB7(8), 0xBA(32)
10	0x00(64), 0x05(48), 0x85(48), 0x89(48), 0x99(48)
11	0x46(64), 0x5E(64), 0xC6(64), 0xDE(64)
12	0x00(128), 0x05(32), 0x09(32), 0x19(32), 0x85(32)
13	0x00(128), 0x42(32), 0x5A(32), 0xC2(32), 0xDA(32)
14	0x00(128), 0xA0(32), 0xA8(32), 0xB0(32), 0xB8(32)
15	0x00(128), 0x04(32), 0x08(32), 0x18(32), 0x84(32)
16	0x00(128), 0x02(32), 0x1A(32), 0x82(32), 0x9A(32)
17	0x00(128), 0x80(32), 0x88(32), 0x90(32), 0x98(32)
18	0x00(192), 0x0C(16), 0x1C(16), 0x8C(16), 0x9C(16)
19	0x00(128), 0x04(64), 0x84(64)
20	0x00(64), 0x08(128), 0x18(64)
21	0x00(128), 0x02(64), 0x82(64)
22	0x00(64), 0x10(192)
23	0x00(192), 0x80(64)

Table 4: Full SDDT of PIPO S-box

3.2 Intermediate Value Filtering based on SDDT property

The SDDT provides a mechanism for key recovery by allowing the adversary to filter the possible values of the intermediate state x. Suppose the adversary

induces a skip fault at the chosen i-th operation (E_i) and observes an S-box output difference Δy. According to the definition of SDDT, the actual intermediate value x must satisfy Eq. (7).

Let $\mathcal{X}_{i,\Delta y}$ be the set of possible candidates for x given i and Δy:

$$\mathcal{X}_{i,\Delta y} = \{x \in \{0,1\}^n \mid S(x) \oplus S_{E_i}(x) = \Delta y\}. \tag{9}$$

If SDDT$[i][\Delta y] = 0$, the occurence of Δy is theoretically impossible under the assumed fault model, indicating an experimental error or an invalid assumption. If SDDT$[i][\Delta y] = a$, the search space for x is reduced from 2^n to a. The procedure to filter the candidate set $\mathcal{X}_{i,\Delta y}$ is outlined in **Algorithm 4**. Based on $\mathcal{X}_{i,\Delta y}$, the candidates for the round key RK can be filtered by leveraging the known ciphertext. Since PIPO uses an 8-bit S-box ($n = 8$), if the average size of $\mathcal{X}_{i,\Delta y}$ is smaller than 2^8, the candidate space for the key is effectively reduced.

Algorithm 4 Finding Candidate States based on SDDT property

Require: An n-bit S-box $S(x)$
Require: Skipped operation index i
Require: Observed output difference $\Delta y \in \{0,1\}^n$
Ensure: Set of candidate inputs $\mathcal{X}_{i,\Delta y}$
1: $\mathcal{X}_{i,\Delta y} \leftarrow \emptyset$ ▷ Initialize empty set
2: **for** each guess $x \in \{0,1\}^n$ **do** ▷ Iterate over all possible x
3: $\delta \leftarrow S(x) \oplus S_{E_i}(x)$
4: **if** $\delta = \Delta y$ **then** ▷ Filter x s.t. satisfies Eq. (7)
5: $\mathcal{X}_{i,\Delta y} \leftarrow \mathcal{X}_{i,\Delta y} \cup \{x\}$
6: **end if**
7: **end for**
8: **return** $\mathcal{X}_{i,\Delta y}$

4 Attack Scenario on PIPO

The goal of our attack on PIPO-64/128 is to recover the 128-bit master key K. To achieve this, the adversary exploits the lightweight key scheduling structure of PIPO. The 128-bit master key K is divided into two 64-bit subkeys, K^0 and K^1, such that $K = K^1 \parallel K^0$. The round key RK_r for the r-th round is derived by alternately using these subkeys:

$$RK_r = K^{r \pmod 2} \oplus c_r \tag{10}$$

where $c_r = r$.

Since PIPO-64/128 consists of 13 rounds, the round keys for the last two rounds correspond to $RK_{12} = K^0 \oplus \texttt{0xC}$ and $RK_{13} = K^1 \oplus \texttt{0xD}$. Consequently, recovering the last round key RK_{13} and the penultimate round key RK_{12} allows for the immediate reconstruction of the entire master key. Based on this

observation, our attack strategy focuses on recovering these two specific round keys.

Similarly, this key recovery attack is applicable to PIPO-64/256. In the case of PIPO-64/256, the 256-bit master key is partitioned into four 64-bit subkeys. By sequentially recovering these round keys using the method described in this and subsequent section, the full 256-bit master key can be reconstructed.

4.1 Fault Model

We adopt the operation skip fault model, which is highly practical in the context of embedded devices executing software implementations of bitslice ciphers. The assumptions and capabilities of the adversary in this model are defined as follows:

- **Fault Location:** The adversary can inject faults during the execution of the S-box operations (S-layer). Additionally, the adversary targets a specific round in the encryption process.
- **Fault Type:** The adversary is capable of skipping a single operation at a specific index i within the S-box sequence. Based on the definitions in Section 3, this fault is denoted as E_i.
- **Data Collection:** The adversary first injects the fault E_i to obtain the faulty ciphertext C_{E_i}. Correspondingly, the adversary acquires the correct ciphertext C by encrypting the same plaintext under normal operating conditions.

Under these assumptions, the adversary utilizes the difference between C and C_{E_i} to perform the key recovery attack described in the following sections.

4.2 Fault Propagation

This section analyzes the fault propagation induced by an operation skip fault. We formulate a general propagation model parameterized by the skip index i, which remains valid prior to the identification of the specific fault location. Based on this analysis, we determine the optimal operation index i to be targeted.

State Definitions for Propagation Analysis Each of the six boxes indicates an $8 \times 8 = 64$-bit state. This representation is the same as in Fig. 2. The last round of the cipher is denoted as R.

- The script uppercase letters $\mathcal{A}_r$, $\mathcal{B}_r$, and $\mathcal{C}_r$ represent the 64-bit differences for the r-th round SL input, r-th round SL output, and r-th round RL output, respectively, when a fault E_i is injected on $R-1$ round SL. Note the distinction between a single ciphertext C with the 64-bit difference $\mathcal{C}_r$.
- The j-th columns of the state are denoted as $\mathcal{A}_r^{(j)}$, $\mathcal{B}_r^{(j)}$, and $\mathcal{C}_r^{(j)}$. Since our approach is applied to each S-box individually, this column-specific notation is necessary to track the differential behavior.

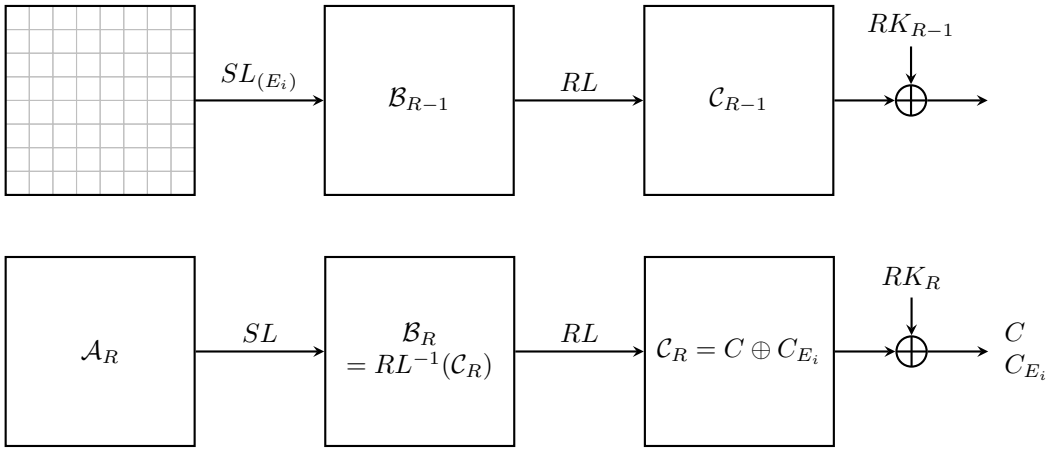

Fig. 4. Propagation of faults and differences in the last two rounds of PIPO.

Based on the analysis in Section 3.1, the 8-bit difference for each $\mathcal{B}_{R-1}^{(j)}$ is restricted by the SDDT. We define the set of possible 8-bit output differences from E_i as $\mathbb{D}_i$. This set $\mathbb{D}_i$ is derived directly from the SDDT. Since the linear rotation layer (RL) diffuses these differences, the set of possible 8-bit differences at $\mathcal{C}_{R-1}^{(j)}$ is expanded; we denote the possible 8-bit difference set $\mathbb{D}_i^{RL}$. Because key addition is a linear operation, $\mathcal{C}_{R-1} = \mathcal{A}_R$, which means the specific difference remains invariant. The 64-bit difference $\mathcal{A}_R$ induces the difference $\mathcal{B}_R$, and is transformed into $\mathcal{C}_R$ through RL. Again, since key addition is linear, $\mathcal{C}_R = C \oplus C_{E_i}$.

The adversary exploits this fault propagation through two distinct mechanisms.

1. **Final Round Filtering (DDT):** They apply intermediate value filtering to the last round's substitution layer (SL) using standard DDT properties, as described in Section 2.3. Given the set of possible input differences $\Delta X = \mathbb{D}_i^{RL}$ and the observed output difference $\mathcal{B}_R^{(j)}$, this filtering process directly reduces the candidate space for the final round key RK_R.
2. **Penultimate Round Filtering (SDDT):** Analogous analysis is conducted on the penultimate round using the SDDT properties detailed in Section 3.2. Similar to the first phase, this intermediate value filtering enables the recovery of the preceding round key RK_{R-1}.

Before applying our framework to PIPO, we define a selection criterion for identifying the optimal operation to skip, thereby maximizing the effectiveness of the SDDT. In the structure of PIPO, the output difference generated by an operation skip propagates through the linear rotation layer (RL) to become the input difference for the next round. If the input difference for a specific S-box

is zero, the differential analysis described in Section 2.3 cannot be applied. To ensure a reliable attack, it is crucial to guarantee that the operation skip itself generates a non-zero output difference. Therefore, based on the properties shown in Table 4, we selected the operation at index $i = 11$ as our target, as it is the only operation that never yields a zero output difference. Unless specified, all operation skip faults discussed hereafter target the operation $i = 11$.

With SDDT in Table 4, set of possible output difference $\mathbb{D} = \{\texttt{0x46}, \texttt{0x5E}, \texttt{0xC6}, \texttt{0xDE}\}$ when $i = 11$ is fixed. Through bitwise rotation RL, $\mathbb{D}$ could be expanded to $\mathbb{D}^{RL}$. The procedure to compute $\mathbb{D}^{RL} = \{\texttt{0x46}, \texttt{0x5E}, \texttt{0xC6}, \texttt{0xDE}, \texttt{0x4E}, \texttt{0x56}, \texttt{0xCE}, \texttt{0xD6}\}$ is shown in Table 5. Notably, 5 bits across all elements in $\mathbb{D}$ share fixed values; these constant bit positions are preserved (merely shifted) after the rotation. The fact that only 3 bits remain variable results in a set cardinality of $\#\mathbb{D}^{RL} = 2^3 = 8$.

Table 5. Bitwise representation of possible input differences and the resulting output difference of RL.

Possible Input Differences				RL	**Output difference**
0x46	**0x5E**	**0xC6**	**0xDE**		
0	0	1	1	→	**?**
1	1	1	1	→	1
0	0	0	0	→	0
0	1	0	1	→	**?**
0	1	0	1	→	**?**
1	1	1	1	→	1
1	1	1	1	→	1
0	0	0	0	→	0

4.3 Key Recovery Strategy

In this section, we present a strategy to recover the master key K by exploiting the operation skip fault. As demonstrated in Sections 2.3 and 3.2, both the DDT and the SDDT allow for the filtering of intermediate values at the granularity of individual columns (i.e., per S-box). This property enables us to avoid an exhaustive search over the entire 64-bit state. Instead, we adopt a divide-and-conquer approach: the adversary independently recovers the partial equivalent key $\hat{k}$ of RK_r associated with each 8-bit column and subsequently reconstructs the full 64-bit round key.

By leveraging the DDT or SDDT and a single correct-faulty ciphertext pair, the adversary can obtain a set of candidates for each 8-bit partial equivalent key. To uniquely determine the correct key, the adversary repeats the fault injection

process and employs an *intersection scheme*. This procedure involves intersecting the candidate sets generated from multiple fault injections to filter out wrong candidates and isolate the unique, deterministic key.

Recovery Algorithm for RK_R with DDT property For a given pair of correct and faulty ciphertexts C and C_E, the 64-bit difference $\mathcal{B}_R = RL^{-1}(C \oplus C_E)$ is uniquely determined. This implies that the output difference of the substitution layer (SL) is fixed for each column. Furthermore, based on the assumption that the adversary can invoke a precise operation skip fault, the set of possible input differences $\mathbb{D}^{RL} = \{\texttt{0x46}, \texttt{0x5E}, \texttt{0xC6}, \texttt{0xDE}, \texttt{0x4E}, \texttt{0x56}, \texttt{0xCE}, \texttt{0xD6}\}$ is known.

Consequently, the prerequisites for **Algorithm 2** are satisfied, with the set of input differences defined as $\Delta X = \mathbb{D}^{RL}$ and the observed output difference as $\Delta y = \mathcal{B}_R^{(j)}$. This enables the adversary to filter the set of possible 8-bit inputs for the j-th column of the final SL for all $0 \leq j \leq 7$. For each 8-bit input candidate x, the corresponding 8-bit partial round key is uniquely determined using the correct ciphertext C, which can be expressed as

$$\hat{k} = S(x) \oplus \left(RL^{-1}(C)\right)^{(j)}, \tag{11}$$

where $\hat{k}$ denotes an 8-bit partial equivalent key associated with the j-th column prior to the linear diffusion RL. Because the round key addition is performed after RL, $\hat{k}$ corresponds to a subset of round-key bits that appear in non-adjacent positions in the post-RL representation.

By employing the intersection scheme and multiple pairs of ciphertexts, the adversary can uniquely identify the correct round key RK_R.

Recovery Algorithm for RK_{R-1} with SDDT Having successfully recovered the final round key RK_R, the adversary proceeds to retrieve the preceding round key RK_{R-1}. The overall procedure leverages the obtained RK_R to partially decrypt the ciphertext pairs and applies a filtering logic described in Section 3.2.

The analysis employing the SDDT mirrors the DDT-based mechanism, with the SDDT replacing the DDT as the filtering tool. By partially decrypting the correct and faulty ciphertexts (C and C_E) for one round using RK_R and computing their XOR sum, the adversary derives the difference $\mathcal{C}_{R-1}$:

$$\mathcal{C}_{R-1} = SL^{-1} \circ RL^{-1}(C \oplus RK_R) \oplus SL^{-1} \circ RL^{-1}(C_E \oplus RK_R). \tag{12}$$

Subsequently, applying the inverse linear layer operation RL^{-1} yields the difference $\mathcal{B}_{R-1}$. Based on the fault E_{11}, the difference in each column $\mathcal{B}_{R-1}^{(j)}$ is guaranteed to be one of the four possible values—`0x46`, `0x5E`, `0xC6`, or `0xDE`. By identifying exactly which of these four differences has occurred, the adversary can narrow down the candidates. Since every entry in the SDDT[11] typically holds a uniform count of 64, a single filtering step reduces the number of round

key candidates to 64 per column. Iterating this process allows for the construction of a reduced candidate set for RK_{R-1}. For each input candidate x, the corresponding 8-bit partial equivalent key $\hat{k}$ is determined as:

$$\hat{k} = S(x) \oplus RL^{-1} \circ SL^{-1} \circ RL^{-1}(C \oplus RK_R)^{(j)}. \tag{13}$$

4.4 Achieving Robustness against Invalid Fault Injections

In a real-world experimental environment, achieving perfect temporal and spatial precision for fault injection is challenging. Consequently, a fault might be induced at an unintended operation, or unexpected random noise may corrupt the attack. Such invalid faults can lead to incorrect key recovery if interpreted as valid data. To ensure the success of the attack even under these imperfect conditions, we propose two mitigation strategies: *Fault Profiling* and a *Voting Scheme*.

1. **Fault Profiling**
   As demonstrated in the previous section, the input difference to the last round S-box under the target fault model is restricted to one of 8 specific patterns. By exhaustively testing all 256 possible values of x against the 8 specific input difference patterns, we identified that only 246 unique output differences are reachable. This implies that the remaining 10 output difference values are *impossible* under the assumed fault model. We define the set of impossible differences $\mathbb{D}_{\text{impossible}}$:

   $$\begin{aligned}\mathbb{D}_{\text{impossible}} = \{&\texttt{0x00}, \texttt{0x04}, \texttt{0x09}, \texttt{0x0D}, \texttt{0x19},\\ &\texttt{0x1D}, \texttt{0x82}, \texttt{0x86}, \texttt{0xCA}, \texttt{0xFC}\}.\end{aligned} \tag{14}$$

   This set can serve as a filter. For a PIPO structure processing 8 S-boxes in parallel, if the output difference of any S-box belongs to $\mathbb{D}_{\text{impossible}}$, the fault injection is deemed invalid, and the ciphertext pair is discarded. The probability P_{filter} of detecting and filtering a random invalid ciphertext pair using this method is calculated as follows:

   $$P_{filter} = 1 - \left(\frac{246}{256}\right)^8 \approx 27.3\%. \tag{15}$$

   Although this filtering mechanism removes a portion of invalid faults, it is not exhaustive.
2. **Voting Scheme**
   Since the fault profiling step cannot filter out all invalid faults, relying on a strict set intersection method to recover the key is risky. In an intersection approach, processing a single invalid fault can result in the elimination of the correct key from the candidate set, leading to attack failure.
   To overcome this limitation, we adopt a *voting scheme* (or frequency-based approach) instead of set intersection. The procedure is as follows:
   - For each pair of correct ciphertext and faulty ciphertext, derive a set of candidate keys.

- Instead of intersecting these sets, increment a counter (vote) for every key present in the candidate set.
- After processing all collected data, the key with the highest number of votes is selected as the correct secret key.

This method ensures that the correct key accumulates the most votes from valid fault injections, while candidates derived from invalid faults are scattered randomly, thus maintaining the robustness of the attack.

5 Simulation Analysis

In this section, we validate the theoretical soundness of our attack through software simulation under an ideal fault model (where only valid E_{11} faults are injected). Given this ideal setting, we utilized the intersection scheme, as the voting scheme is unnecessary in the absence of noise.

The simulations were performed specifically on PIPO-64/128. However, since PIPO-64/256 shares an identical round function structure—differing only in the total number of rounds and the key scheduling—the proposed analysis is equally applicable to the 256-bit key variant. Note that for the 13-round PIPO-64/128, the final key is RK_{13}.

5.1 Attack Evaluation Metrics

To evaluate the efficiency of the attack, we employ two primary metrics:

- **Candidate Key Space Size:** We measure how the size of the candidate key space decreases under the intersection scheme, as the number of available correct and faulty ciphertext pairs increases.
- **Data Complexity:** The minimum number of pairs required to uniquely identify the secret key (i.e., reduce the candidate space size to 1).

5.2 Simulation Results

We performed simulations to evaluate the efficiency of the proposed attack. The simulation environment was set up as follows:

- **Target:** We used the reference C implementation [19] of the block cipher PIPO, with minimal modifications to simulate an operation skip on the i-th operation of the S-box.
- **Configuration:** Random master key and random plaintexts were generated for each iteration.
- **Iterations:** To ensure statistical reliability, we conducted 100 independent trials for each of 100 randomly generated master keys, resulting in a total of 10,000 trials. The average number of surviving key candidates was calculated for each number of injected faults, denoted as N.

Recovering RK_{13} with DDT property Figure 5 illustrates the reduction in the equivalent key search space as N increases. The results demonstrate a substantial reduction in the number of candidate keys. These results confirm that the intersection-based filtering is highly effective in an ideal fault injection scenario.

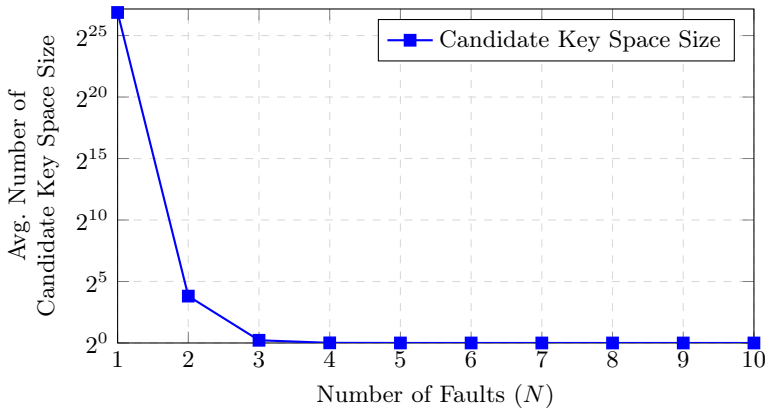

Fig. 5. Convergence of the size of candidate key space of RK_{13} with respect to the number of faults. The key space reduces drastically at $N = 2$ and is uniquely identified at $N = 5$.

- At $N = 1$, the average size of key space is approximately 2^{27}. While this significantly reduces the key space from 2^{64} to 2^{27}, a unique key cannot be determined. Based on Table 2, the expected number of candidates is 8.33 per S-box, suggesting a total search space of $8.33^8 \approx 2^{24.47}$. We attribute the minor gap between the simulation and this estimation to the statistical variance inherent in averaging the candidates per S-box versus averaging the final key space size.
- A sharp convergence is observed at $N = 2$, where the candidate space diminishes to approximately 14.04. This indicates that utilizing just two ciphertext pairs is sufficient to narrow down the candidates to a computationally negligible number.
- For $N \geq 5$, the average size of key space reaches exactly 1.0000. This implies that with 5 faults injected, the proposed attack can uniquely identify the 64-bit round key RK_{13} with a probability of practically 100%.

Recovering RK_{12} with SDDT property Figure 6 illustrates the convergence of the surviving candidate space for the 8-bit partial equivalent key of RK_{12} as N increases. These empirical results confirm that the SDDT enables effective key filtering, albeit with a theoretical bound imposed by the fault model.

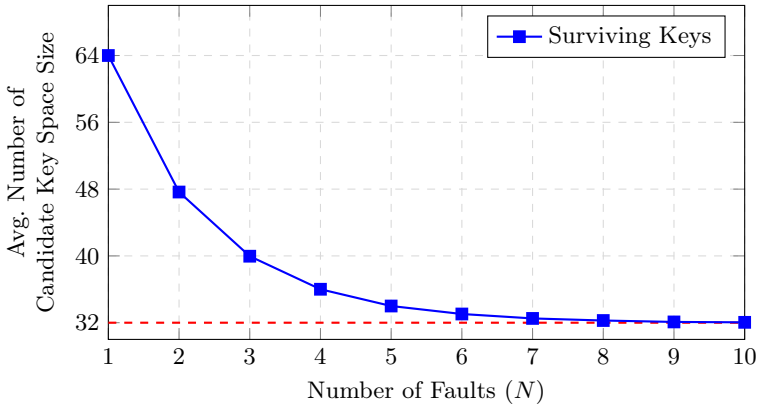

Fig. 6. Convergence of the size of candidate key space of RK_{12}. The search space starts at 64 and effectively reduces to 32 as N increases.

- At $N = 1$, the number of 8-bit partial key candidates is reduced to exactly 64. This aligns with the theoretical expectation discussed in Section 4.3, where the differential properties associated with the fault E_{11} result in 64 possible intermediate values for each possible output difference.
- As N approaches 10, the candidate reduction saturates at a lower bound of 32 rather than converging to a unique solution. This limitation stemmed from the intrinsic collision in the differential patterns. As demonstrated in Table 5, all four possible output differences derived from the fault E_{11} share identical values across 5 of the 8 bit-positions. Consequently, these specific bits remain invariant and indistinguishable via differential analysis. Since the filtering process relies on unique output difference patterns to discard invalid keys, this lack of entropy in the 5-bit field prevents the unique identification of the corresponding key bits.

To resolve this ambiguity and recover the full 128-bit Master Key, we extend the attack strategy to the preceding round. The adversary injects the same type of fault (skipping the operation $i = 11$) during round 11. By combining the candidate set obtained in this section with the analysis of a fault injected in Round 11, we can uniquely identify RK_{12}. Furthermore, this fault allows us to filter RK_{11} using the SDDT property in Section 3.2. Since RK_{11} is related to RK_{13} via the PIPO key schedule, this process also serves to validate the correctness of the previously recovered RK_{13}. Finally, utilizing the invertible key schedule, the Master Key K is fully recovered.

6 Experiments

In this section, we present our experimental results. Unlike ideal fault models where faults are injected precisely as intended, practical environments cannot

guarantee that the targeted operation is always accurately skipped. To address this, we implemented the fault profiling and the voting scheme introduced in Section 4.4.

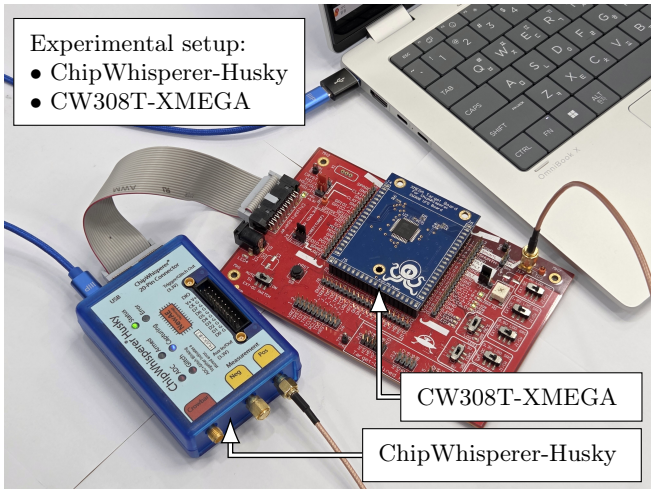

Fig. 7. Experimental setup. The ChipWhisperer-Husky is connected to the host PC for data acquisition, while the target board is mounted on the CW308 UFO board.

6.1 Experimental Setup

To verify the practical feasibility of the proposed attack, we conducted experiments on a real-world 8-bit microcontroller environment. The detailed specifications of our experimental setup are as follows:

- **Target Device:** Since PIPO is designed to operate efficiently on 8-bit registers, we selected the ATXmega128D4-AU, an 8-bit AVR microcontroller, as our target board.
- **Fault Injection Platform:** We employed the ChipWhisperer-Husky, which functions as both the clock glitch injector and the serial communication interface with the host PC.
- **Software Implementation:** Consistent with the simulation described in Section 5, we utilized the reference C implementation of PIPO-64/128 [19], compiled with the `-O3` optimization flag.

To successfully induce operation-level skip faults, precise understanding over the assembly instruction sequence is required. Therefore, we first analyzed the assembly segment corresponding to the 11th operation in the generated listing file `.lss` file. The disassembled code is presented in Fig. 8.

```
4fc: 79 2f  mov r23, r25
4fe: 75 2b  or r23, r21
500: 7c 27  eor r23, r28
502: 71 83  std Z+1, r23
504: 90 95  com r25 ← Target to Skip
506: 1c 97  sbiw r26, 0x0c ; 12
508: a1 f4  brne .+40  ; 0x532 <sbox+0x84>
```

Fig. 8. Disassembled code snippet showing the target operation implemented with the `com` instruction.

The target operation, $X[2] \leftarrow \sim X[2]$, is compiled into a single-cycle `com` instruction operating directly on general-purpose registers. While this structural simplicity makes it a suitable candidate for operation-skip fault injection, the operation is embedded in a dense sequence of single-cycle ALU instructions. This layout significantly constrains the temporal margin for isolating a single-instruction skip. In practice, imprecise clock glitching may suppress not only the target instruction but also adjacent instructions, or potentially corrupt the register write-back stage.

To overcome this challenge and achieve successful fault injection, we employed a high-resolution parameter sweep strategy. We heuristically varied the glitch delay and width to identify the precise timing window that suppresses the target `com` instruction while preserving the integrity of the surrounding instructions. The detailed parameters we used are explained at Table 6.

Table 6. Experimental parameters for fault injection.

Parameter	Value
`ext_offset`	13
`offset`	50
`width`	150
`repeat`	1

6.2 Experimental Results

For the experimental evaluation, we adopted the metrics defined in Section 5.1. In contrast to the intersection scheme where candidates merely *survive*, our approach accumulates votes for all candidate keys. Consequently, we evaluated the performance based on the number of candidates tied for the highest vote

count. Note that invalid ciphertext pairs identified via the fault profiling method described in Section 4.4 were discarded; these excluded instances are not counted in the number of faults N.

By using test vectors with the parameters in Table 6, we achieved a 98.85% success rate for the instruction skip. Trials resulting in device resets or system crashes were excluded from the statistics; we considered only the cases where a ciphertext was successfully returned. While the targeted instruction was successfully skipped in 98.85% of the cases, the remaining failures are attributed to the skipping of incorrect instructions, multiple instructions being skipped, or other unexpected faults.

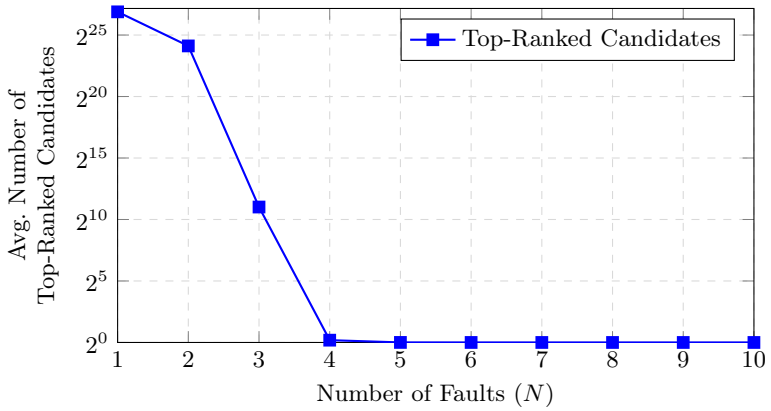

Fig. 9. Experimental convergence of the number of surviving key candidates of RK_{13} with respect to the number of faults. We observe a steady decline in the number of candidates up to $N = 4$, with the correct key being uniquely determined at $N = 6$.

Figure 9 presents the results of the attack performed on the actual device. The experiments were repeated 10,000 times for each number of faults, identical to Section 5. As shown in the figure, the correct master key is uniquely identified (candidate size reaches 1) at approximately $N = 6$.

Figure 10 illustrates the voting score distribution for a representative 8-bit partial key, obtained from the real-world experiment using 6 pairs of correct and faulty ciphertexts. The horizontal axis corresponds to the 256 possible key candidates, while the vertical axis indicates the accumulated vote count for each candidate.

As evident from the graph, the correct partial key accumulated the maximum possible score of 6. In contrast, wrong candidates—arising because a single fault injection yields multiple candidates rather than a unique key, or due to experimental noise—failed to accumulate significant votes, typically remaining at a count of 1 or 2 (depicted in gray).

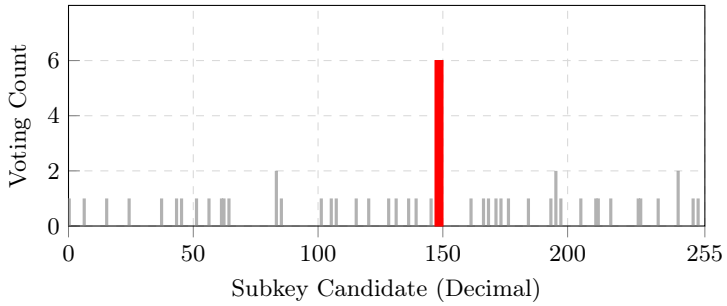

Fig. 10. Voting score distribution for the 8-bit partial equivalent key candidates at $N = 6$. The correct key candidate is uniquely identified with the maximum score of 6.

Figure 11 illustrates the candidate reduction for RK_{12}. While the trend resembles the simulation results in Fig. 6, the average number of surviving keys is slightly higher due to the interference of invalid faults.

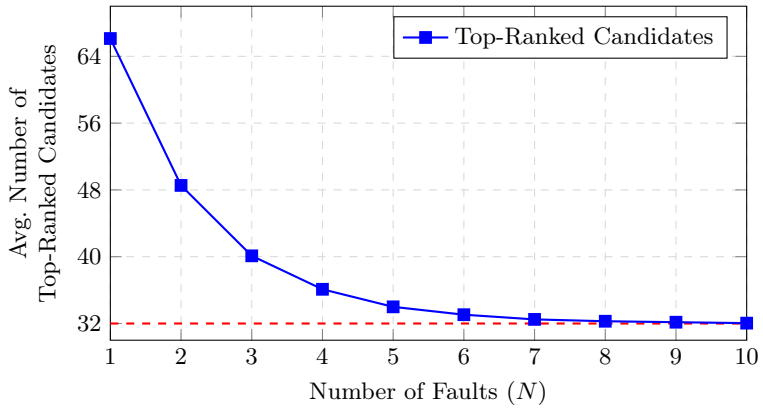

Fig. 11. Experimental convergence of the number of surviving 8-bit partial equivalent keys of RK_{12}.

The experimental results demonstrate the practical feasibility of our attack. The adversary successfully recovered the final round key RK_{13} using approximately 6 faults and reduced the candidate space for each 8-bit partial key of RK_{12} to 32. Consequently, by extending this analysis—injecting operation skip faults into Round 11—the full master key can be recovered.

7 Conclusion

In this paper, we have re-evaluated the security of bitslice implementations, which are widely favored in modern lightweight cryptography for their efficiency and intrinsic resistance to timing attacks. We demonstrated that the very structural characteristic of bitslice designs—the decomposition of the S-box into bitwise operations—creates a unique attack surface vulnerable to the operation skip fault model. Unlike traditional Look-Up Table (LUT) based implementations where such faults are infeasible, bitslice implementations allow adversaries to induce deterministic functional omissions.

Our primary contribution lies in the identification of the strongly restricted differential characteristics caused by these operation skips. While standard fault models (e.g., bit-flips) suffer from rapid diffusion and high entropy, forcing attacks to be limited to the final few rounds, our research reveals that operation skips maintain low entropy even after propagating through the diffusion layer. To exploit this property, we proposed the **Skip-induced Difference Distribution Table (SDDT)**, a systematic framework that maps specific instruction skips to their output difference patterns.

By applying the SDDT framework to the block cipher PIPO, we successfully performed a key recovery attack. Most notably, we achieved this by injecting faults into deeper rounds than those accessible by previous DFA studies, while simultaneously requiring fewer faults. This proves that the operation skip fault model poses a more severe threat in terms of both attack depth and fault complexity.

Several avenues for future research remain to fully assess the scope of this vulnerability. First, since this weakness stems from the fundamental nature of bitslice logic rather than the specific design of PIPO, we intend to generalize the SDDT framework to other prominent bitslice ciphers, such as Serpent, RECTANGLE, GIFT, and Ascon. Second, we plan to extend our analysis to more complex scenarios, including multiple or random instruction skips, to validate the attack's feasibility in noisier environments. Consequently, developing cost-effective countermeasures that can protect bitslice implementations against such faults—without compromising their performance benefits—stands as an imperative task for future cryptographic engineering.

Acknowledgments. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(RS-2024-00348138) and by Samsung Electronics Co., Ltd (IO251216-14690-01).

References

1. Bagheri, N., Ebrahimpour, R., Ghaedi, N.: New differential fault analysis on PRESENT. EURASIP Journal on Advances in Signal Processing **2013**(1), 145 (2013)

2. Balasch, J., Gierlichs, B., Verbauwhede, I.: An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus. In: Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 105–114. IEEE (2011)
3. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. Lecture Notes in Computer Science, vol. 10529, pp. 321–345. Springer, Cham, Switzerland, Taipei, Taiwan (Sep 25–28, 2017). https://doi.org/10.1007/978-3-319-66787-4_16
4. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P.: Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE **100**(11), 3056–3076 (2012)
5. Bartlett, H., et al.: Random fault attacks on a class of stream ciphers. Security and Communication Networks **2019**(1), 1680263 (2019)
6. Biham, E.: A fast new DES implementation in software. In: Biham, E. (ed.) Fast Software Encryption – FSE'97. Lecture Notes in Computer Science, vol. 1267, pp. 260–272. Springer Berlin Heidelberg, Germany, Haifa, Israel (Jan 20–22, 1997). https://doi.org/10.1007/BFb0052352
7. Biham, E., Anderson, R.J., Knudsen, L.R.: Serpent: A new block cipher proposal. In: Vaudenay, S. (ed.) Fast Software Encryption – FSE'98. Lecture Notes in Computer Science, vol. 1372, pp. 222–238. Springer Berlin Heidelberg, Germany, Paris, France (Mar 23–25, 1998). https://doi.org/10.1007/3-540-69710-1_15
8. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology **4**(1), 3–72 (Jan 1991). https://doi.org/10.1007/BF00630563
9. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski, Jr., B.S. (ed.) Advances in Cryptology – CRYPTO'97. Lecture Notes in Computer Science, vol. 1294, pp. 513–525. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 1997). https://doi.org/10.1007/BFb0052259
10. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: Fumy, W. (ed.) Advances in Cryptology – EUROCRYPT'97. Lecture Notes in Computer Science, vol. 1233, pp. 37–51. Springer Berlin Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997). https://doi.org/10.1007/3-540-69053-0_4
11. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: Exploiting ineffective fault inductions on symmetric cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 547–572 (2018). https://doi.org/10.13154/tches.v2018.i3.547-572, https://tches.iacr.org/index.php/TCHES/article/view/7286
12. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Submission to NIST. NIST Lightweight Cryptography Round 2 Specification (PDF) (Sep 2019), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf
13. Gong, X., Zhang, X., Wu, Q., Zhang, F., Xu, J., Shen, Q., Zhang, Z.: Practical opcode-based fault attack on AES-NI. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(3), 693–716 (2025). https://doi.org/10.46586/tches.v2025.i3.693-716
14. Grosso, V., Leurent, G., Standaert, F.X., Varici, K.: LS-designs: Bitslice encryption for efficient masked software implementations. In: Cid, C., Rechberger, C.

(eds.) Fast Software Encryption – FSE 2014. Lecture Notes in Computer Science, vol. 8540, pp. 18–37. Springer Berlin Heidelberg, Germany, London, UK (Mar 3–5, 2015). https://doi.org/10.1007/978-3-662-46706-0_2

15. Jana, A.: Differential fault attack on ascon cipher. In: Mukhopadhyay, S., Stănică, P. (eds.) Progress in Cryptology - INDOCRYPT 2024: 25th International Conference in Cryptology in India, Part II. Lecture Notes in Computer Science, vol. 15496, pp. 53–72. Springer, Cham, Switzerland, Chennai, India (Dec 18–21, 2024). https://doi.org/10.1007/978-3-031-80311-6_3
16. Joshi, P., Mazumdar, B.: SPSA: Semi-permanent stuck-at fault analysis of AES rijndael SBox. Journal of Cryptographic Engineering **13**(2), 201–222 (2023)
17. Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes for fault based side-channel cryptanalysis of symmetric block ciphers. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **21**(12), 1509–1517 (2002)
18. Katagi, M., Moriai, S.: Lightweight cryptography for the internet of things. Tech. rep., Sony Corporation (2008)
19. Kim, H., Jeon, Y., Kim, G., Kim, J., Sim, B., Han, D., Seo, H., Kim, S., Hong, S., Sung, J., Hong, D.: PIPO Reference Implementation (Bitsliced C). https://github.com/PIPO-Blockcipher/PIPO-Blockcipher/blob/master/PIPO_reference_bitslice.c (2020), accessed: 2026-01-24
20. Kim, H., Jeon, Y., Kim, G., Kim, J., Sim, B.Y., Han, D.G., Seo, H., Kim, S., Hong, S., Sung, J., Hong, D.: PIPO: A lightweight block cipher with efficient higher-order masking software implementations. In: Hong, D. (ed.) ICISC 20: 23rd International Conference on Information Security and Cryptology. Lecture Notes in Computer Science, vol. 12593, pp. 99–122. Springer, Cham, Switzerland, Seoul, Korea (Dec 2–4, 2020). https://doi.org/10.1007/978-3-030-68890-5_6
21. Könighofer, R.: A fast and cache-timing resistant implementation of the AES. In: Malkin, T. (ed.) Topics in Cryptology – CT-RSA 2008. Lecture Notes in Computer Science, vol. 4964, pp. 187–202. Springer Berlin Heidelberg, Germany, San Francisco, CA, USA (Apr 7–11, 2008). https://doi.org/10.1007/978-3-540-79263-5_12
22. Le, T.H., Canovas, C., Clédière, J.: An overview of side channel analysis attacks. In: Abe, M., Gligor, V. (eds.) ASIACCS 08: 3rd ACM Symposium on Information, Computer and Communications Security. pp. 33–43. ACM Press, Tokyo, Japan (Mar 18–20, 2008). https://doi.org/10.1145/1368310.1368319
23. Lim, S., Han, J., Han, D.G.: Single-byte error-based practical differential fault attack on bit-sliced lightweight block cipher PIPO. IEEE Access **10**, 67802–67813 (2022). https://doi.org/10.1109/ACCESS.2022.3185799
24. Lim, S., Han, J., Lee, T., Han, D.G.: Differential fault attack on lightweight block cipher PIPO. In: Park, J.H., Seo, S.H. (eds.) ICISC 21: 24th International Conference on Information Security and Cryptology. Lecture Notes in Computer Science, vol. 13218, pp. 296–307. Springer, Cham, Switzerland, Seoul, Korea (Dec 1–3, 2021). https://doi.org/10.1007/978-3-031-08896-4_15
25. McKay, K.A., Bassham, L.E., Turan, M.S., Mouha, N.: Status report on the first round of the NIST lightweight cryptography standardization process. Tech. Rep. NISTIR 8114, National Institute of Standards and Technology (Sep 2017)
26. Mella, S., Melzani, F., Visconti, A.: Differential fault attacks against AES tampering with the instruction flow. In: 2014 11th International Conference on Security and Cryptography (SECRYPT). pp. 1–8. IEEE (2014)

27. Menu, A., Renauld, M., Ordas, A., Dutertre, J.M., Tria, A.: Experimental analysis of the electromagnetic instruction skip fault model. In: Proceedings of the 15th Design and Technology of Integrated Systems in Nanoscale Era (DTIS). pp. 1–7. IEEE (2020)
28. National Bureau of Standards: Data encryption standard (des). Federal Information Processing Standards Publication 46 (1977), available at https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub46.pdf
29. Nguyen, V.S., Grosso, V., Cayrel, P.L.: Practical persistent fault attacks on AES with instruction skip. IACR Communications in Cryptology (CiC) **2**(1), 40 (2025). https://doi.org/10.62056/a60l5wol7
30. Piret, G., Quisquater, J.J.: A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: Walter, C.D., Koç, Çetin Kaya., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2003. Lecture Notes in Computer Science, vol. 2779, pp. 77–88. Springer Berlin Heidelberg, Germany, Cologne, Germany (Sep 8–10, 2003). https://doi.org/10.1007/978-3-540-45238-6_7
31. Shuvo, A.M., Zhang, T., Farahmandi, F., Tehranipoor, M.: A comprehensive survey on non-invasive fault injection attacks. Cryptology ePrint Archive, Report 2023/1769 (2023), https://eprint.iacr.org/2023/1769
32. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski, Jr., B.S., Koç, Çetin Kaya., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2002. Lecture Notes in Computer Science, vol. 2523, pp. 2–12. Springer Berlin Heidelberg, Germany, Redwood Shores, CA, USA (Aug 13–15, 2003). https://doi.org/10.1007/3-540-36400-5_2
33. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. In: Information Security Theory and Practices. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2011)
34. Xiao, H., Wang, L.: Differential fault analysis on the lightweight block cipher plug-in plug-out. Security and Privacy **6**(3), e286 (2023). https://doi.org/10.1002/spy2.286
35. Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. Science China Information Sciences **58**(12), 1–15 (2015)
36. Zhao, X., Guo, S., Zhang, F., Wang, T., Shi, Z., Ji, K.: Algebraic differential fault attacks on LED using a single fault injection. Cryptology ePrint Archive, Report 2012/347 (2012), https://eprint.iacr.org/2012/347