SoK : On DFA Vulnerabilities of Substitution-Permutation Networks

Mustafa Khairallah
School of Physical and Mathematical Sciences, NTU, Singapore
mustafam001@e.ntu.edu.sg

Xiaolu Hou
Acronis, Singapore
ho0001lu@e.ntu.edu.sg

Zakaria Najm
Temasek Labs at NTU, Singapore
TU Delft, The Netherlands
zakaria.najm@ntu.edu.sg

Jakub Breier
Underwriters Laboratories, Singapore
jbreier@jbreier.com

Shivam Bhasin
Temasek Labs at NTU
Singapore
sbhasin@ntu.edu.sg

Thomas Peyrin
School of Physical and Mathematical Sciences, NTU, Singapore
thomas.peyrin@ntu.edu.sg

ABSTRACT

Recently, the NIST launched a competition for lightweight cryptography and a large number of ciphers are expected to be studied and analyzed under this competition. Apart from the classical security, the candidates are desired to be analyzed against physical attacks. Differential Fault Analysis (DFA) is an invasive physical attack method for recovering key information from cipher implementations. Up to date, almost all the block ciphers have been shown to be vulnerable against DFA, while following similar attack patterns. However, so far researchers mostly focused on particular ciphers rather than cipher families, resulting in works that reuse the same idea for different ciphers.

In this article, we aim at bridging this gap, by providing a generic DFA attack method targeting Substitution-Permutation Network (SPN) based families of symmetric block ciphers. We provide the overview of the state-of-the-art of the fault attacks on SPNs, followed by generalized conditions that hold on all the ciphers of this design family. We show that for any SPN, as long as the fault mask injected before a non-linear layer in the last round follows a non-uniform distribution, the key search space can always be reduced. This shows that it is not possible to design an SPN-based cipher that is completely secure against DFA, without randomization. Furthermore, we propose a novel approach to find good fault masks that can leak the key with a small number of instances. We then developed a tool, called *Joint Difference Distribution Table (JDDT)* for pre-computing the solutions for the fault equations, which allows us to recover the last round key with a very small number of pairs of faulty and non-faulty ciphertexts. We evaluate our methodology on various block ciphers, including `PRESENT-80`, `PRESENT-128`, `GIFT-64`, `GIFT-128`, `AES-128`, `LED-64`, `LED-128`, `SKINNY-64-64`, `SKINNY-128-128`, `PRIDE` and `PRINCE`. The developed technique would allow automated DFA analysis of several candidates in the NIST competition.

KEYWORDS

differential fault analysis, substitution-permutation network, fault attack

ACM Reference Format:
Mustafa Khairallah, Xiaolu Hou, Zakaria Najm, Jakub Breier, Shivam Bhasin, and Thomas Peyrin. 2018. SoK : On DFA Vulnerabilities of Substitution-Permutation Networks. In . ACM, New York, NY, USA, 17 pages.

1 INTRODUCTION

Substitution-Permutation Network (SPN) is a fundamental design strategy for block ciphers, with many primitives using SPN either for a part of their design or as the main design concept. An SPN consists of one or more (usually many) iterations of the following three operations:

(1) Substitution (confusion): The state of the network is divided into words and a non-linear substitution is applied to each of them. The substitution function can be the same for all of them, or different functions can be used for different words.
(2) Permutation (diffusion): A state-wise permutation is applied. This step is responsible for propagating the information between the internal state words as fast as possible.
(3) Key mixing: A secret key is mixed with the state, usually using an XOR operation.

The maximum security level possible for a block cipher is measured by its resistance against brute force attacks, hence by the bit-size of the master key used to generate the round keys. However, there are many cryptanalytic techniques that try to push this boundary, by studying the specific properties of the building blocks of the cipher. Most of these techniques fall into one of two categories:

(1) Linear Cryptanalysis: the attacker tries to approximate the cipher as a group of linear equations between the input, output and key bits, with high probability.
(2) Differential Cryptanalysis: the attacker tries to leak information about the key bits by observing the differences between different input/output pairs.

Over the years, many design techniques and studies have been established in order to build ciphers that are secure against these two types of attacks. The idea in case of SPN is that the substitution layer provides highly non-linear relations between bits of internal words, with low maximum difference probability, while the permutation layer mixes these relations together, increasing the complexity. By repeating these two operations many times (rounds), mixing with (random) key bits in every iteration, the plaintext/ciphertext pair should be practically indistinguishable from two uniformly random vectors. However, if the number of iterations is not enough, the previous statement cannot be true. Thus, usually cryptanalysts start by analyzing reduced-round versions of the SPNs in question, while the cipher designers try to increase the number of rounds beyond the maximum number of rounds with non-ideal properties.

Surprisingly, the reduced-round properties of SPNs have been useful beyond the theoretical analysis and/or defining the minimum required number of rounds for a cipher. With the emergence of fault attacks as a rising domain in the field of hardware security, the attacker can change some of the internal bits of the cipher in the last few rounds and use the properties of only these rounds to leak information about the key. For example, there are practical fault attacks against AES that use the differential properties of 1, 2, 3 or 4 rounds, while classical attacks require the properties of 10 full rounds. However, in case of fault analysis such as Differential Fault Analysis (DFA), the attacks generally either rely on heuristic analysis, empirical data or on general ideas that do not take the specifics of the cipher into consideration. Similarly, the countermeasures for these attacks are generally either at the implementation level [20, 24, 25, 31] or at the the protocol/encryption mode level [4, 14, 15, 29].

The systemization of DFA on SPNs started by the work of Piret and Quisquater [32], where they proposed a somewhat general DFA analysis of AES-like ciphers. Although their attack has been enhanced in several subsequent works, it was still exclusive to AES-like ciphers. Besides, the complexity of the attack is high and the complexity analysis is approximate (Section 2). Moreover, the attack did not discuss how to identify the optimal faults thoroughly. On the other hand, in [36], the authors provided a discussion on the optimality of different DFA attacks, which we revisit in our paper and identify the advantages and shortcomings of their approach. Hence, we identified a need for the systemization of DFA attacks on SPNs, in order to have a general methodology for analyzing SPNs, as opposed to analyzing each cipher independently. This helps not only to study and compare the available SPNs in literature, but to analyze future SPNs, as well.

The goal of this paper is to understand why different SPNs behave differently against DFA, and what properties make an SPN stronger or weaker in this context, achieving a better understanding of how the two layers of an SPN interact with each other in the context of DFA. We study some of the available DFA attacks against SPNs, identifying the weak points of the SPN design strategy against DFA. We also find general vulnerabilities of SPNs against these attacks, enabling us to find new attacks against modern SPN ciphers. The main idea is to find a new approach to identify a good location for fault injection and quantify the corresponding expected amount of information leakage. For most SPNs, the attack of choice is a single-word fault injected in round $r-2$ for an SPN with r rounds, using a 2-round distinguisher. We identify what are the weaknesses that are common for all SPN ciphers and what are the differences between them. In the process, we also propose a method for efficiently performing the attacks using a Time-Memory trade-off, which allows to pre-compute a big part of the analysis, with the ability to reuse it for any attack instance. We believe the proposed methodology will serve as a useful tool in analyzing a plethora of ciphers expected to be submitted for NIST lightweight cryptography competition.

Our Contributions.

(1) We revisit the information theoretic approach from [36], providing new insights on how the differential properties of the function attacked and the fault distribution affect the information leakage, showing that, for any *deterministic non-linear* bijective function of the form $S(x) \oplus K$, where x and K are unknown, the entropy of K can be reduced by calculating $S(x \oplus \Delta x) \oplus K$, as long as Δx is non-uniform. Hence, *we show that it is not possible to design an SPN-based cipher that is inherently secure against DFA* (Sections 3 and 4).

(2) We *formalize* the complexity of retrieving the master key by injecting a fault into the last round of an SPN. Next, we propose an approach to accelerate such attack by reducing the number of faults and maximizing the information leakage per fault (Section 4).

(3) Once a good fault location is identified, we propose a new tool for pre-computing the solutions of the fault equations, called the *Joint Difference Distribution Table (JDDT)*, based on the fault model, and the properties of the substitution and permutation layers (Section 4.2).

(4) We describe a class of SPNs that share a similar 2-round distinguisher and provide a new method to analyze the differential properties of related attack and use it to find key candidates for the last round key by observing a single pair of faulty and non-faulty ciphertexts (Section 5). We validate this analysis on modern ciphers, such as PRESENT-80, PRESENT-128, GIFT-64, GIFT-128, AES-128, SKINNY-64-64, SKINNY-128-128, PRINCE, LED-64 and LED-128 (Section 6). We report three different kinds of results:

   M *Match optimal results* of well-studied ciphers, like AES

   O *Find optimal attacks* for less studied ciphers, superseding previous known best attacks like PRESENT, LED, PRIDE

   N *Find new attacks* on recently proposed ciphers with no (or little) public fault analysis like SKINNY, GIFT

(5) We discuss some general intuitions or good practices which can help cipher designers to improve security of their cipher against DFA.

Table 1 shows the ciphers analyzed in this paper using our technique, which is also illustrated in Figure 1. It shows that some of the lightweight ciphers are harder to break. However, none of these ciphers can be considered secure, since the key can be uniquely identified using at most 12 faulty and non-faulty ciphertext pairs.

2 BACKGROUND

Differential Fault Analysis (DFA) is the oldest and most popular fault analysis method targeting symmetric cryptography. Since its inception in 1997 [7], almost all the symmetric block ciphers have been shown vulnerable against it.

The working principle of DFA is as follows. The attacker first runs an encryption procedure on plaintext P with a secret key K without disturbing the computation. Then, she repeats the encryption with the same inputs, but injects a fault, normally during the last few rounds of the cipher. She compares the faulty ciphertext

Table 1: Comparison between the different ciphers analyzed by our analysis technique. The location of the fault is considered to be known. N denotes new results, O denotes optimal results compared to previous results.

			Remaining Brute-force Complexity					
Cipher	Fault Model	Implementation	1 pair	2 pairs	3 pairs	4 pairs	16 pairs	Attack Type
AES-128	Random Byte	Any	$2^{8.06}$	1	1	1	1	M[38]
AES-128	1-Bit Flip	Any	$2^{0.15}$	1	1	1	1	O
LED-64	Random Nibble	Any	$2^{10.4}$	1	1	1	1	M[23]
LED-64	1-Bit Flip	Any	$2^{9.5}$	1	1	1	1	O
LED-128	Random Nibble	Any	$2^{74.4}$	$2^{20.8}$	$2^{10.4}$	1	1	M[23]
LED-128	1-Bit Flip	Any	$2^{73.5}$	2^{19}	$2^{9.5}$	1	1	O
PRESENT-80	4-Bit Flip	Any	2^{41}	2^{2}	1	1	1	O
PRESENT-80	Random Nibble	Bit-Sliced	2^{41}	2^{2}	1	1	1	O
PRESENT-128	4-Bit Flip	Any	2^{95}	2^{67}	2^{34}	2^{6}	1	O
GIFT-64	4-Bit Flip	Any	$2^{111.175}$	$2^{100.2}$	$2^{86.3}$	$2^{72.4}$	1	N
GIFT-128	4-Bit Flip	Any	$2^{111.175}$	$2^{100.2}$	$2^{86.3}$	$2^{72.4}$	1	N
PRIDE	4-Bit Flip	Any	$2^{104.6}$	$2^{87.2}$	$2^{72.4}$	2^{64}	1	O
SKINNY-64-64	Random Nibble	Any	2^{51}	$2^{40.4}$	$2^{34.8}$	$2^{5.6}$	1	M[40]
SKINNY-128-128	Random Byte	Any	$2^{102.4}$	$2^{81.28}$	$2^{69.76}$	$2^{11.52}$	1	M [40]
PRINCE	Random Nibble	Any	$2^{93.2}$	2^{48}	$2^{13.2}$	1	1	M[37]
PRINCE	4-Bit Flip	Any	$2^{84.16}$	2^{48}	$2^{4.16}$	1	1	O
Other Differential Fault Analyses in Literature								
PRESENT-80 [9]	16-bit Flip	Any	2^{40}	2^{16}	2^{6}	1	1	
PRESENT-80 [30]	1-bit Flip + Side Channel	Any	2^{80}	2^{76}	2^{66}	2^{64}	1	
PRIDE [26]	16-bit Flip	Any	$2^{86.4}$	2^{64}	$2^{22.4}$	1	1	

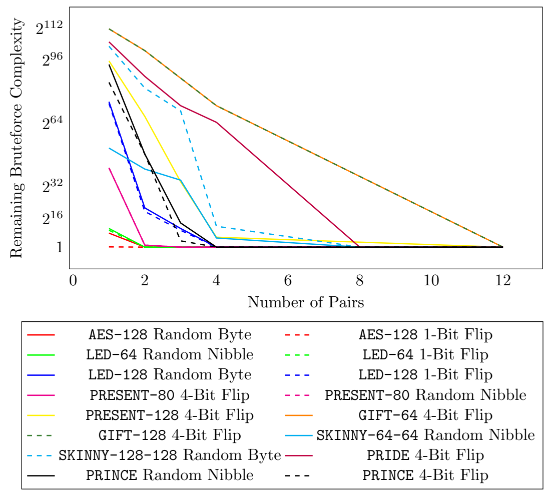

Figure 1: The number of remaining key candidates vs. the number of faulty and non-faulty ciphertext pairs for the ciphers in Table 1

with the correct one and gets information about one of the round keys. Depending on attacker model and the cipher structure, she repeats the fault injection several times until the guessing complexity of the key is low enough to get K.

The trend in analyzing block ciphers with DFA usually follows the same pattern for different encryption algorithms – first, an intuitive approach appears, requiring more faults but lower brute-force complexity. Later, researchers tend to develop more sophisticated techniques that can reveal the secret key with either single or very low number of faults, while increasing the complexity of the analysis step. As an example, one can take DFA on AES, that improved from the early approaches requiring 35-250 faulty encryptions [16, 17], to more recent one that needs just a single fault [38]. Similarly, first DFA of PRESENT required 65 faults [41], later decreased to 2 [13].

A complementary approach to this is automated analysis that is focusing either on DFA [11] or Algebraic Fault Analysis (AFA) [43].

Related work about DFA on SPNs. The work presented in our paper is closely related and inspired by the work of Piret and Quisquater [32], which was later extended and optimized by Tunstall et al. [38]. In this paper, we extend this line of work in three directions:

(1) Instead of performing an approximate analysis of the attack complexity, assuming ideal primitives (Sbox and diffusion layers), we incorporate the details of the cipher in question into the analysis. While this leads to similar results in case of AES, since the Sbox of AES is well designed (almost ideal) and the diffusion layer uses an MDS matrix, the results concerning lightweight ciphers are different, due to the non-ideal primitives used. This analysis enables us to compare ciphers with respect to DFA security, beyond the simple bit security, showing that SPN ciphers with the same block and key sizes are not necessarily the same when it comes to security against DFA.
(2) The analysis from [32] was targeting AES-like SPNs. We extend our analysis to a wider class of SPNs to include also bit-permutation based ciphers, such as PRESENT and GIFT, and SPNs with a diffusion layer that depends on an almost-MDS matrix, such as SKINNY.

(3) We provide a framework for efficiently implementing the computational part of the attack, showing that a huge part of the attack can be pre-computed only once per cipher and reused to attack as many instances as required, as opposed to the random search approach used in previous works.

Multiple-Fault Attacks on SPN. While the definition of whether a fault is considered a single fault or multiple fault is vague and generally implementation-dependent, it is usually the case that a single fault is either a single-bit fault or a fault that is limited to b bits, which is the input size of 1 Sbox. In our analysis, we consider any fault distribution over more than b bits to be a multiple fault attack and out of the scope of our paper. However, recently a similar analysis to ours has been presented by Lac et al. [27] which discusses a multiple-fault model. Their analysis was described for the LS family of block ciphers, then extended to AES-like cipher, e.g. AES and LED. The idea of their analysis is to accelerate the attack in Section 4 by injecting a multiple-word fault to the input of the diffusion layer in the second last round, such that all the Sboxes in the last round are active, with known input differences. However, the multiple-word known fault model is less practical compared to our work and is not suitable for some implementations. Moreover, in [27] only a special class of Sboxes was considered. Hence, in Section 5, we propose a single fault DFA for a wide class of SPNs.

Multiple-fault attacks were also used to attack PRESENT [9] and PRIDE [26], with each of these attacks requiring to flip 16 bits simultaneously.

Definitions.

Throughout this paper, we use some definitions for fault models that are listed below:

- Single fault: a fault whose maximum width is $\leq b$, where b is the number of input bits to the Sbox used in the cipher in question.
- Uniform/Random fault: a fault that can have any value between 1 and $2^b - 1$, where all the values are equiprobable.
- Constant/Known fault: a fault that has a specific value defined by the attacker.

For all the faults used in this paper, the location and timing of the faults are assumed to be known to the attacker. In practice, if the attacker is uncertain about the timing or location of the fault, he needs to repeat the analysis for every possibility.

Practical Fault Models. Three fault models are used throughout this paper:

(1) Random fault model: A random byte/nibble is added to an internal byte/nibble of the cipher. This is the most practical fault model used in this paper and it has been used in several practical attacks [23, 37, 38].
(2) Single bit flip: A specific internal bit of the cipher is flipped. In practice, it is more complex than the random fault, but it has been shown to be practical in several papers/attacks [1, 3, 21, 39].
(3) Four-bit flip: 4 adjacent internal bits are flipped together. While this requirement can be challenging, it was shown in [9] that such fault is practically possible. Moreover, there are a few tricks the attacker can use to get around this requirement. In Section B.1.2, we show some tricks that work around the requirements of this model, showing that for some implementations, the attacker can achieve the required fault with only a random fault injection. The idea is that with some knowledge on the nature of the implementation, the attacker can use a random fault that can only lead to the required fault value.

Table 2: Notation used in the rest of the paper.

X_i	input to the function $S(x)$ at a certain invocation
Y_i	output of the function $S(x)$ at a certain invocation
K	secret key
Z_i	$Y_i \oplus K$
ΔX	$X_1 \oplus X_2$
ΔY	$Y_1 \oplus Y_2$
ΔZ	$Z_1 \oplus Z_2$

3 INFORMATION THEORETIC DFA MODEL: TOWARDS A THEORETICAL SECURITY METRIC FOR DFA

In an effort to find a theoretical metric for studying the security of ciphers against DFA, the authors of [36] introduced an information theoretic approach for evaluating whether a DFA is optimal or not. Given a fault model at the input of a function $S(x) + k$, the authors provided an information theoretic equation that can be used to calculate the maximum amount of information leakage under that fault model. First, we present the equation and then we present some results based on that equation, which contradicts with one of the inferential conclusions the authors made, due to the ambiguity of the mathematical definition of the fault model in the original paper.

Notation. $X_i, Y_i, Z_i, K, \Delta X, \Delta Y$ and ΔZ are random variables defined in Table 2.

$$H(K|Z_1Z_2) = H(\Delta X|\Delta Y) + H(X_1|\Delta X\Delta Y) \qquad (1)$$

Equation 1 can be used to calculate the entropy of the Secret Key, knowing a single pair of faulty and correct ciphertexts [36] (or, generally, any pair of ciphertexts). The first thing to notice is that $\Delta Y = \Delta Z = Z_1 \oplus Z_2$, which is public. However, the equation does not show how to calculate X_1 and ΔX. These two variables implicitly hold the information about the function $S(x)$ and the assumption about the fault model. Assuming a uniform fault model and no knowledge about $S(x)$, it is straightforward to deduce that no information about the key is leaked and hence $H(K|Z_1Z_2) = n$. In [36], the authors also calculate the entropy of the key $H_{\mathsf{AES}}(K|Z_1Z_2)$ assuming a uniform fault model and analyze the AES Sbox. Since $H_{\mathsf{AES}}(K|Z_1Z_2) = n$, the authors infer that AES Sbox is a good cryptographic function. While we are not challenging this conclusion, we can show that this analysis is not conclusive, as this result is achieved due to the uniformity of the fault model and not because of the properties of the AES Sbox. *In addition, we show that it is not possible to design any SPN that is inherently secure against DFA without randomization, where the target security level is considered to be at least the brute force complexity of searching for x, i.e.* $O(2^{|x|})$.

First, we define $s_{\Delta x,\Delta y}$ as the number of solutions of Equation 2. Hence, $H(X_1|\Delta X = \Delta x, \Delta Y = \Delta y) = log(s_{\Delta x,\Delta y})$. Additionally, since $S(x)$ is a bijective function, $Pr(\Delta X = \Delta x, \Delta Y = \Delta y) = \frac{s_{\Delta x,\Delta y}}{2^n}$. Finally, $Pr(\Delta X = \Delta x|\Delta Y = \Delta y) = Pr(\Delta X = \Delta x, \Delta Y = \Delta y)$, since

ΔY is public and $Pr(\Delta Y) = 1$.

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

Theorem D.1 is used to find the amount of information leakage when the input difference follows a known distribution. The proofs of required for this section are available in Appendix D.

Theorem 3.1. *If ΔX is sampled from $\mathcal{S}$, such that $|\mathcal{S}| = z$ and $P(\Delta X) = p_x$.then the expected number of leaked bits of K, when ΔY is observed is*

$$n - \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x, \Delta y}}{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j, \Delta y} p_{x_j}} log(\frac{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j, \Delta y} p_{x_j}}{p_x}) \tag{3}$$

Corollary 3.2. *Given a pair of faulty and correct ciphertexts Z_1 and Z_2, if $\Delta X \in \{0,1\}^n$ is a uniform random variable, then $H(K|Z_1Z_2) = n$, regardless of the properties of the function $S(x)$.*

From Corollary D.2, we can see that if the fault model is unbiased and unrestricted, the key space is not reduced, regardless of the cryptographic properties of $S(x)$. Hence, any fault model used in DFA must be non-uniform, with respect to $S(x)$.

Corollary 3.3. *If $\Delta X = \Delta x$ (constant), then using one pair (Z_1, Z_2), the key space can be reduced from 2^n to $s_{\Delta x, \Delta y}$.*

Corollary 3.4. *Only linear (affine) Boolean functions achieves the theoretical security bound $H(K|Z_1Z_2) = n \forall \Delta x$, regardless of the distribution of ΔX.*

Despite that linear/affine functions achieve the required bound in terms of differential cryptanalysis, they are not helpful as they can be analyzed using linear cryptanalysis. For example, a function that looks like $z = L(x) + k$, where $L(x)$ is affine and z is known, can be analyzed as $L^{-1}(z) = x + L^{-1}(k)$ and the cipher is then attacked neglecting this function, with the target to find $L^{-1}(k)$ instead of k, since we can easily derive one from the other. Theorem D.5 can be used to compare the quality of different fault values and to design attacks that maximize the information leakage.

Theorem 3.5. *If $\Delta X = \Delta x$ (constant), then the expected number of leaked bits of K is $n - \sum_{\Delta y \in \{0,1\}^n} log(s_{\Delta x, \Delta y}) P(\Delta Y = \Delta y | \Delta X = \Delta x)$.*

4 DFA AGAINST THE LAST ROUND OF SPN

Theorem D.3 and Corollary D.4 can be used to provide a generic attack against any Substitution-Permutation-Network (SPN) that follows the description in Section 1. An important observation is that any linear function at the end of the last round can be effectively neglected. In other words, if the last step of the SPN is of the form $C = L(x) \oplus K_r$, where $L(x)$ is a linear function and K_r is the last round key, it can be converted into $L^{-1}(C) = x \oplus L^{-1}(K_r)$ and attack the cipher for the effective key $K_{eff} = L^{-1}(K_r)$. Then, the real key is calculated as $K_r = L(K_{eff})$. Hence, in the generic attack we consider this a standard step of no additional cost.

Using this structure, the space of last round key K_r of any SPN can be reduced using the following procedure. We assume the cipher has w words per state and the state size is n. Each word has size $b = \frac{n}{w}$.

1. For each substitution function in the last round, the Difference Distribution Table (DDT) is calculated and the minimum entry value is located. An input difference with such value in the corresponding row is selected for each function.
2. If the cipher includes a linear function $L(x)$ before the addition of K_r, we concatenate the function L^{-1} to the cipher, such that the output is $L^{-1}(C)$.
3. Iteratively, we inject a fault by flipping the bits of one word according to the corresponding input difference. By observing the output difference and applying Theorem D.3, the space of the last round key K_{eff} bits XORed with this word is reduced to $s_{\Delta x, \Delta y}$, which is the number of solutions of the DDT for the input/output difference pair $(\Delta x, \Delta y)$.
4. By repeating this for every word, the overall space of K_{eff} is reduced from 2^n to $\prod_{i=0}^{w-1} s^{w_i}_{\Delta x, \Delta y}$, where $s^{w_i}_{\Delta x, \Delta y}$ is the number of solutions for word w_i of the last round, based on the corresponding input/output difference pair. The number of faults is equal to w.

For widely used ciphers, such as AES, LED, PRESENT and GIFT, the value of $s_{\Delta x, \Delta y} = 2$ for most of the possible input/output difference pairs, and $w = 16$. Hence, they require at most 16 faults and the resulting space for the last round key is 2^{16}. This is the simplest DFA attack against SPN. However, depending on the underlying cipher, optimizations can be found. For example, for the 4 mentioned ciphers, we can double the number of faults by repeating the fault injection operation for every word. While every DFA gives 2 candidates for a key word, the overlap between these candidates is the right key word, i.e. we get the actual key value. Another optimization is to use more sophisticated/smart fault injection mechanism to trigger many words simultaneously [9, 35]. However, we think that these optimizations are cipher-dependent and still follow the outline of the generic attack. Another aspect that is cipher dependent is the relation between the last round key K_r and the master key. If the master key cannot be uniquely determined from the last round key, the attacker can brute force the undetermined bits, which means that the number of candidates of the master key is the number of candidates for the last round key multiplied by the exponent of difference between the sizes of the two keys, $2^{|K_m| - |K_r|}$, where, K_m is the master key and $|X|$ is the bit length of X. Another solution, is to use the last round key candidates to decrypt the last round and use the same DFA attack to get candidates for the second-to-last round keys and try to compute the master key using the two sets of candidates, this can be repeated until enough round keys are obtained.

Since this is a generic attack, we can characterize the cipher by $2^{|K_m| - |K_r|} \cdot min(\prod_{i=0}^{n-1} s^{w_i}_{xy}) = 2^{|K_m| - |K_r|} \cdot \prod_{i=0}^{\frac{w}{-}1} min_{(\Delta x, \Delta y)}(s^{w_i}_{xy})$. This shows the complexity of retrieving the master key using an instantiation of the attack.

4.1 Reduction of the number of faults

In this section, we describe a framework for accelerating the key recovery using lesser number of faults than what is described in Section 4. The goal of the attacker is to recover the key with the minimum number of faults possible, while keeping the differential analysis simple. For example, injecting the difference in the input block will maximize the number of active Sboxes, but the analysis of the faulty ciphertexts requires the full differential cryptanalysis of the cipher to be feasible, which contradicts the security of the cipher. In order to achieve that, he has to trigger more than one Sbox at once, while trying to maintain that the difference propagation remains simple. While it is hard to find the exact minimum number of faults required, we describe a heuristic approach to approximate this minimum value. First, we define two properties of the Sboxes involved in the SPN.

Definition 4.1. Interacting Sboxes two Sboxes are interacting if the outputs of these two Sboxes are combined together in the Linear Diffusion Layer.

Definition 4.2. Active Sbox is an Sbox for which the input difference is not equal to 0 when a certain fault is injected in the SPN.

The goal of the attacker is to find the fault value/location that maximizes the number of active Sboxes while keeping the number of interacting Sboxes minimal. For example, for an attacker who can inject a known-byte fault at the input/output of any of the Sboxes of AES, his goal to maximize the number of active Sboxes in the final few rounds, while the number of interacting Sboxes has to be 0.

4.1.1 Example 1: Application to LS SPNs. The LS family of SPNs was proposed by Grosso et al. in 2014 [18]. The target of this family is to be able to have very efficient bit-sliced implementations. One round of the cipher is described as follows: a block of $n \times m$ bits is organized into an array:

$$\begin{bmatrix} b_0^0 & b_1^0 & \cdots & b_{m-1}^0 \\ b_0^1 & b_1^1 & \cdots & b_{m-1}^1 \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ b_0^{n-1} & b_1^{n-1} & \cdots & b_{m-1}^{n-1} \end{bmatrix} \tag{4}$$

Then, a non-linear substitution operation is performed on each column using an n-bit Sbox, followed by a linear diffusion operation applied on each row, using an m-bit linear function. Finally, the round key is XORed. The previous steps are repeated for r rounds. In [27], the authors proposed a DFA on this family, where a fault is injected in one of the rows before the diffusion layer in round $r-1$, such that the input to r has exactly one row where every bit is flipped. This activates every Sbox in round r with the same input difference Δx and output difference Δy_i. Consequently, the space of K_r is reduced to $\prod_{i=0}^{n} s_{\Delta x, \Delta y_i}$. However, since the DFA from [27] usually requires a fault over a large word of width 8 bits or more, depending on the row length, it is considered to be a multiple fault attack, if $m > n$. An interesting question would be if it is possible to find a set of low Hamming weight differences, such that the bits whose value is 1 are located near each other, i.e. the set bits are spread over $\leq n$ bits, such that when these differences are applied to the input of the linear diffusion layer, the output difference has a high Hamming weight value. In other words, we trigger as much Sboxes as possible with a low Hamming weight fault. For example, we describe a toy-cipher that is constructed in an LS structure, using the AES Sbox and MixColumn operations. The state consists of an 8×32 bit array, as follows:

$$\begin{bmatrix} b_0^0 & b_1^0 & \cdots & b_{31}^0 \\ b_0^1 & b_1^1 & \cdots & b_{311}^1 \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ b_0^7 & b_1^7 & \cdots & b_{31}^7 \end{bmatrix} \tag{5}$$

Each round of the cipher consists of 32 column-wise Sbox operations, followed by 8 row-wise MixColumn operations. If the attacker can flip any 8 adjacent bits, there are 130 32-bit fault values that are spread over only 8 bits and trigger at least 24 Sboxes. However, the highest number of Sboxes that can be triggered with such model is 29 out of 32 Sboxes, with only a single value achieving this bound. Hence, for our toy cipher, the maximum number of active Sboxes, while maintaining the number of interacting Sboxes at 0, is 29. It is also noticeable that it is impossible for this toy cipher to have exploitable faults in any round except the last round where the number of active Sboxes is larger and the number of interacting Sboxes is 0.

In case of SCREAM [19], the state is a 16×16 bit array. Hence, a single fault according to our definition can be up to 16 adjacent bits. However, since 16-bit Sboxes are not common in practice, and since the bit-sliced implementation provided by the designers targets 8-bit microcontrollers, we search for fault masks that are spread over at most 8-bits. We consider two cases, the first case is when the 8 bits are any 8 adjacent bits at the input of the linear layer, while the second is when the 8 bits are exactly either the top half or the bottom half of the input. In the first case, the maximum number of active Sboxes is 14, while in the second case, it is 13. This means that in order to attack the last round of the cipher and activate more than 14 Sboxes, the fault has to be spread over 9 or more bits.

4.1.2 Example 2: Application to SKINNY*.* SKINNY is an AES-like block cipher presented in CRYPTO 2016 by Beierle et al. [6]. It is targeted for lightweight applications and uses the Tweakey framework to provide the possibility of using it as a tweakable block cipher [22]. It has 6 versions, 3 of which use 64-bit blocks and 4-bit nibbles/Sboxes, while the other 3 use 128-bit blocks and 8-bit nibbles/Sboxes. The master key sizes are 64, 128, 192, 128, 256, 384, for each of them, respectively. Generally, a version tagged SKINNY-n-m uses n-bit blocks and m-bit tweakeys. Being an AES-like cipher, it follows our description nicely, except that the mixing layer uses a non-MDS MixColumn matrix given below:

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{6}$$

Since 2 columns have 3 non-zero elements and 2 columns have 1 non-zero element, injecting a fault in round $r-2$ activates at most 3 Sboxes only in round $r-1$, and if the fault is injected in the proper nibble/byte, it can activate at most 7 Sboxes in round r, which is less than half the number of Sboxes, distributed as follows: 2 columns with 3 Sboxes, 1 column with 1 Sbox and 1 column with 0 Sboxes. If a nibble/byte fault is injected in round $r-3$, the number of interacting Sboxes cannot be equal to 0.

4.2 Joint Difference Distribution Table (JDDT)

We can define a single round of an SPN as a group of non-linear functions $S(x)$, where $S(x)$ consists of a linear part (diffusion layer) and a non-linear part (Sbox). The main idea of the JDDT is that when a single word difference Δ is injected into the input of the diffusion layer in round j, the inputs to n corresponding Sboxes in round $j+1$ are not independent, but are $\{l_1(\Delta), l_2(\Delta), ...l_n(\Delta)\}$, where $l_i(x)$ is a linear function from 1 word to 1 word, which corresponds to the difference propagation through the diffusion layer. Hence, instead of analyzing each of the Sboxes in round $j+1$ independently, we develop a Joint Difference Distribution Table (JDDT) for the n Sboxes, which is actually a portion of the DDT of the function $[A_1, A_2, ...A_n] \leftarrow S(\{a_1, a_2, ...a_n\})$. The JDDT of n Sboxes consists of exactly $2^{-(n-1)b}$ rows of the overall DDT of $S(x)$. The purpose of the JDDT is to provide candidates for the output value of $S(x)$, given Δ and the output difference.

We compute the JDDT as follows: we consider all the 2^{4b} possible output differences and divide them into four b-bit differences. We use each of these values to access the corresponding DDT and find all the possible input differences corresponding to this value. Typically, for good Sboxes, these would be four lists of around 2^{b-1} values each, which means $2^{4(b-1)}$ possible values for the difference at the output of the diffusion layer. However, only a subset of these values satisfies the relation $\{l_1(\Delta), l_2(\Delta), l_3(\Delta), l_4(\Delta)\}$. Hence, they are tested and only the solutions corresponding to valid differences are stored into the JDDT.

5 THREE ROUND DFA ATTACK ON SPNS

In this section, we describe a single fault DFA attack against a family of SPN-based block ciphers. This family includes majority of the widely used SPNs, such as AES, LED, PRESENT and GIFT. The advantage of the attack described in this section is that it uses a single-fault injection and a single pair of faulty and correct ciphertexts. It can be used as a security metric for an SPN against DFA attacks. The family of SPNs we consider has the following properties:

(1) The state of the cipher consists of w words, each of b bits divided into g groups, where $w = 4g$ and $g \bmod 4 \equiv 0$.
(2) Each round consists of a substitution layer that operates on every word individually, using a non-linear Sbox, followed by a diffusion layer that consists of two parts: shuffling and mixing.
(3) The shuffling step generates new groups, such that every group consists of 4 words from 4 different groups in the previous round.
(4) The mixing step performs a linear operation on the words of every group, such that every output word depends on the 4 input words. In other words, the mixing step operates on each group independently.
(5) Every word at the output of the substitution layer of the current round affects exactly one group of the next round.
(6) Every 4 groups before shuffling are mapped into 4 groups after shuffling.
(7) There exists at east one word difference value δ_o, such that when exactly one word is active in round j, with difference δ_o at the input of the diffusion layer, 4 words are active at the beginning of round $j+1$ and 4 groups are active at the beginning of round $j+2$.

These steps are depicted in Figure 2.

In case of AES, and LED, the last property is satisfied by all word differences, hence the attack can be launched with a random difference. On the other hand, in case of PRESENT or GIFT, this property is only satisfied when all the bits of the chosen word are inverted, i.e. $\delta = [1111]_2$. A random fault in the case of AES is not to be confused with the uniform fault model in Theorem D.2, since $S(x)$ in this attack will be a 4 word to 4 word function consisting of the mixing layer followed by 4 Sboxes. Hence, the fault model used for AES is a restricted fault, picked from only 256 possible input values out of 2^{32} values.

The attack consists of an offline phase and an online phase. In the offline phase, first, we perform a slight modification to the SPN during the analysis. Instead of each round consisting of Substitution, Shuffling, Mixing and Addition of the round key K_r, we consider it to consist of Substitution, Shuffling, Addition of the effective round key $L^{-1}(K_r)$ and finally, Mixing. In this view, we merge Mixing and Substitution into one function $S(x)$. Then, we generate the extended DDT of the Sboxes in this function. Finally, we compute the JDDT for this function.

In the online phase of the attack, once the output difference of a group is obtained, the JDDT is accessed, and based on the assumptions on the input difference values (fault model, Sbox/Mixing differential properties, etc), a set of potential output values is required. These values are XORed with the ciphertext to get a set of key-words candidates of this group. The time complexity of the attack is $O(2^{4b+2})$ $S(x)$ operations. Since a full encryption consists of around $g \times r$ $S(x)$ operations, the time complexity can be represented as $O(\frac{2^{4b+2}}{g \times r})$ encryptions. The space complexity is the space required to store the JDDT, $O(2^{5b})$. The number of key candidates is $|K_s| \times 2^{4b(g-4)+|K_m|-|K_r|}$, where $|K_s|$ is the size of the key space for the four attacked groups after the analysis and is a characteristic of the cipher.

The number of key candidates can be further reduced by generating all the candidates for the outputs of round $r-1$ and performing the online phase again for each, then keeping only the keys that satisfy a specific relation between K_r and K_{r-1}, such that both round keys represent a valid key schedule from the same master key. In round $r-1$ only one group is active, hence we can solve only for 4 key words. For every K_r candidate, we get $|K_s^{'}| \leq 2^{4b}$ candidates for these four words. Eventually, the key space size temporarily grows to $|K_s^{'}| \times |K_s| \times 2^{4b(g-4)+|K_m|-|K_r|}$. However, given the key scheduling algorithm of the cipher, which is used to generate round keys, we consider the relationship between the 4 words of round $r-1$ and the 16 words of round r. $P_{ks} \leq 1$ is the probability that such relation holds, assuming uniformly random bit assignment. Hence, the final key space size is $P_{ks} \times |K_s^{'}| \times |K_s| \times 2^{4b(g-4)+|K_m|-|K_r|}$ and the overall complexity of the online phase of the attack is bounded by $O(\frac{|KS| \times 2^{4b(g-4)}}{r})$, which is the brute force cost of the second step of the attack. It is to be noted that the second step may not always lead to better results, as it depends on the key schedule of the cipher. For the attacker to gain from this step, the condition $P_{ks} \times |K_s^{'}| \ll 1$ should be satisfied.

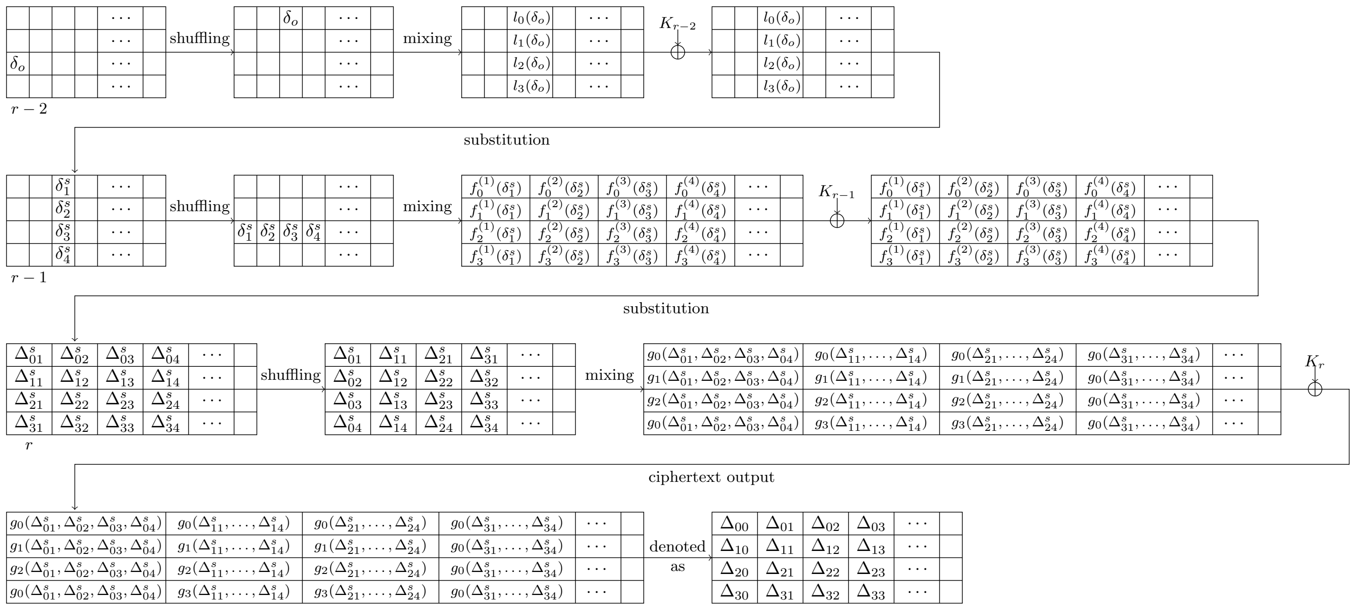

Figure 2: 3 Round DFA on SPNs

6 SINGLE FAULT ATTACKS AGAINST REAL WORLD SPNS

In this section, we first study the DFA on PRESENT-80, an SPN based on bit permutations. To the best of our knowledge, no single fault attack with a small number of pairs have been reported so far against it. We also discuss applying our technique to AES-128, showing it matches the best known attack against AES. Besides, we apply our technique to SKINNY, since is has some unique properties that can be exploited to achieve more efficient attacks. We provide more case studies in Appendix B.

6.1 PRESENT-80/128: Finding Optimal Attack

PRESENT [8] is a lightweight block cipher proposed by Bogdanov et al. in CHES 2007. It targets applications such as RFID tags and sensor networks. It has been extensively studied over the years. [42] provides an analysis of the differential properties of PRESENT. In [9], the authors use a multiple fault model to attack it, flipping all the bits of 4 specific words at the same time. Since the fault model is specific and hard to achieve, they rely on a hardware Trajan to inject it. In [30], the authors presented a single fault attack on PRESENT. However, the attack requires side-channel assistance (namely power measurements) in order for the analysis to work, which is not required in this paper. At the end of this section we compare the attack from [30] to our attack.

PRESENT consists of 31 rounds, each round operates on a 64-bit block and contains three operations: addRoundKey, sBoxLayer and pLayer. addRoundKey is described as follows

$$s_j \leftarrow s_j \oplus k_j \tag{7}$$

where j is the round index and s_j, k_j are the cipher state and round key at round j, respectively. The next step is to divide the state s_j into 16 4-bit nibbles $b_j^{(i)}$, i.e. $s_j = b_j^{(0)} b_j^{(1)} \dots b_j^{(15)}$. Each nibble is replaced using the Sbox function $sb(x)$, defined using Table 3. The 16 parallel Sboxes represent the sBoxLayer.

Table 3: The 4-bit Sbox used in the PRESENT cipher

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$sb(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Table 4: The bit permutation used in PRESENT

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Finally, the pLayer is described as a bit permutation $P(i)$ over 64 bits, where i refers to the bit index. The permutation is described in Table 4. For the purpose of the analysis in this paper, we describe the permutation using a different, yet equivalent, representation. The new representation consists of the shuffling and mixing operations described in Section 5. The shuffling operation starts after the sBoxLayer, by grouping the 16 $b_j^{(i)}$ into four groups, where each group is a column in the following matrix.

$$M = \begin{bmatrix} b_j^{(0)} & b_j^{(1)} & b_j^{(2)} & b_j^{(3)} \\ b_j^{(4)} & b_j^{(5)} & b_j^{(6)} & b_j^{(7)} \\ b_j^{(8)} & b_j^{(9)} & b_j^{(10)} & b_j^{(11)} \\ b_j^{(12)} & b_j^{(13)} & b_j^{(14)} & b_j^{(15)} \end{bmatrix} \tag{8}$$

Then, the shuffling operation is defined as the matrix transpose operation M^t. Hence, the mixing operation can be described as 4

Table 5: The mixing operation in PRESENT

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P^{'}(i)$	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

parallel instances of the permutation in Table 5, where a bit i is i-th bit of the nibble $g + \lfloor i/4 \rfloor \times 4$, and $g \in \{0, 1, 2, 3\}$ is the group number. A somewhat similar representation of PRESENT was used for efficient software implementations in [34].

This representation makes some of the properties of the pLayer more clear:

(1) The bits of nibbles $b_j^{(g)}, b_j^{(g+4)}, b_j^{(g+8)}, b_j^{(g+12)}$ (before shuffling) depend completely on the bits of nibbles $b_j^{(4g)}, b_j^{(4g+1)}, b_j^{(4g+2)}, b_j^{(4g+3)}$.
(2) If only one nibble in group g is active at the input of the mixing operation with difference δ, the number of active nibbles after mixing is equal to the Hamming weight of δ.
(3) If only one nibble $0 \leq i \leq 3$ in group g is active at the input of the mixing operation, then all the active nibbles after mixing have the same difference $\delta^{'} = 2^i$.

Property 2 can be used to show that if one nibble is active at the input of the mixing operation, then only $\delta = [1111]_2$ leads to 4 active nibbles at the output. Property 3 can be used to find the value of the output differences from $\{1, 2, 4, 8\}$. Hence, by injecting a fault δ in nibble i in the output of the sBoxLayer in round 29, a difference equal to $2^{i \ mod \ 4}$ is injected in the 4 nibbles of group $\lfloor i/4 \rfloor$. In round 30, the sBoxLayer changes these differences into $\{\delta_0, \delta_1, \delta_2, \delta_3\}$, then the shuffling operation distributes these four differences into 4 different groups. By using property 2, we can show that the number of active Sboxes in round 31 (the final round), is the Hamming weight of the vector $v = [\delta_0 \ \delta_1 \ \delta_2 \ \delta_3]$. In order to study the properties of the vector v, we need to study the differential properties of the Sbox. The analysis here refers to the differential cryptanalysis of PRESENT performed by Wang [42]. Depending on where the fault $[1111]_2$ is injected in round 29, four Sboxes in round 30 are triggered with difference $2^{i \ mod \ 4} \in \{1, 2, 4, 8\}$. Table 6 represents the part of the DDT of the Sbox used in PRESENT corresponding to these values. The second-to-last column, μ_{HW}, is the average Hamming weight of the output difference, corresponding to the given input difference and a random value. It is clear that the maximum average number of active Sboxes in round 31, given a fault of $[1111]_2$ in round 29, is 12 and it is achieved when $i \equiv 3 \ mod \ 4$. In other words, the best locations, in terms of diffusion, to inject the fault are nibbles 3, 7, 11, or 15. Similarly, the value of the differences at the input of the active Sboxes in round 31 depends on the fault locations: $[0001]_2$ for $i = 3$, $[0010]_2$ for $i = 7$, $[0100]_2$ for $i = 11$, and $[1000]_2$ for $i = 15$. The last column of Table 6 shows the expected number of solutions of the Sbox output value, assuming the given input difference and a random value, calculated using Theorem D.5. It shows that the information leakage is maximized when $i \in \{7, 11\}$. Analyzing the last round using the attack described in Section 5, the number of candidates of the last round key $|K_s|$ is given by

$$|K_s| = 2^{1.25 \times 12} \times 16^4 = 2^{31} \tag{9}$$

which shows a leakage of 34 bits by applying. In order to perform the second step of the attack, we need to look at the key scheduling algorithm of PRESENT and study the relation between the active bits of rounds 30 and 31.

Table 6: Part of the DDT of the Sbox used in PRESENT

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	μ_{HW}	EXP(SOLs)
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0	2.5	2^2
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0	2.25	$2^{1.25}$
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0	2.25	$2^{1.25}$
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4	3	$2^{1.5}$

6.1.1 PRESENT-80.

The key scheduling algorithm for PRESENT-80 works as follows:

(1) The master key K_m is stored in an 80-bit register as: $[\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_0]$.
(2) The current round key K_i is given by $[\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_{16}]$.
(3) $[\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_0] \leftarrow [\mathsf{K}_{18}\mathsf{K}_{17}...\mathsf{K}_0\mathsf{K}_{79}\mathsf{K}_{78}...\mathsf{K}_{19}]$
(4) $[\mathsf{K}_{79}\mathsf{K}_{78}\mathsf{K}_{77}\mathsf{K}_{76}] \leftarrow sb([\mathsf{K}_{79}\mathsf{K}_{78}\mathsf{K}_{77}\mathsf{K}_{76}])$.
(5) $[\mathsf{K}_{19}\mathsf{K}_{18}\mathsf{K}_{17}\mathsf{K}_{16}\mathsf{K}_{15}] \leftarrow [\mathsf{K}_{19}\mathsf{K}_{18}\mathsf{K}_{17}\mathsf{K}_{16}\mathsf{K}_{15}] \oplus \mathtt{round\_counter}$.

Assuming the contents of the register at round 31 are $[\kappa_{79}\kappa_{78}...\kappa_0]$, then the contents at round 30 are

$$[\kappa_{60}\kappa_{59}...\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\tilde{\kappa_{16}}\tilde{\kappa_{15}}...\kappa_0]sb^{-1}([\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76}])[\kappa_{75}\kappa_{74}...\kappa_{61}] \tag{10}$$

Besides, the round keys are given by

$$K_{31} = [\kappa_{79}\kappa_{78}...\kappa_{16}] \tag{11}$$

$$K_{30} = [\kappa_{60}\kappa_{59}...\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\tilde{\kappa_{16}}\tilde{\kappa_{15}}...\kappa_0]\mathtt{truncate}(sb^{-1}([\kappa_{79}\kappa_{78}\kappa_{77}\kappa_{76}], 3)) \tag{12}$$

where $\mathtt{truncate}(\mathtt{x}, 3)$ return the 3 most significant bits of x.

If the fault is injected in round 29 in nibble 7, then nibbles 1, 5, 9 and 12 are active in round 30, with an Sbox input difference equal $[1000]_2$. The correct solutions for these nibbles of K_{30} must satisfy

$$[\kappa_{52}\kappa_{51}\kappa_{50}\kappa_{49}\kappa_{36}\kappa_{35}\kappa_{34}\kappa_{33}\kappa_{20}\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\kappa_4\kappa_3\kappa_2\kappa_1] \tag{13}$$

Since $\kappa_4, \kappa_3, \kappa_2$, and κ_1 are not used in K_{31}, the probability of satisfying the condition $P_{ks} = 2^{-12}$. Besides, since the 4 nibbles have an input difference of $[1000]_2$, and using Table 6, the number of solutions $|K_s^{-1}|$ for the 4 active nibbles of K_{30} is 2^6. Hence, combining steps one and two of the attack, the number of candidates of K_{31} becomes 2^{25}, and the number of master key candidates becomes 2^{41}, with complexity 2^{31}.

On the other hand, if the fault is injected in round 29 in nibble 11, then nibbles 1, 5, 9 and 12 are active in round 30, with an Sbox input difference equal $[1000]_2$. The correct solutions for these nibbles of K_{30} must satisfy

$$[\kappa_{56}\kappa_{55}\kappa_{54}\kappa_{53}\kappa_{40}\kappa_{39}\kappa_{38}\kappa_{37}\kappa_{24}\kappa_{23}\kappa_{22}\kappa_{21}\kappa_8\kappa_7\kappa_6\kappa_5] \tag{14}$$

Similar to the previous case, 12 conditions must be satisfied. Hence, the analysis is the same. If the attack is applied twice (2 fault injections), the second time leads to a new set of 2^{41} keys is calculated. However, the probability of one or more of the wrong key candidates overlapping the first set, i.e. the probability that the size of intersection between the two sets ≥ 1, can be computed using the Binomial distribution. The experiment is defined as selecting a uniform random key 80-bit vector, and it is successful if the selected key is one of the 2^{41} keys calculated during the first attack. Hence, $p = 2^{-39}$ and $Pr(X \leq k) = \sum_i^k Pr(X = i)$, which is also the probability of the key space size to k with 2 faults. Table 7 shows

Table 7: The probability of having a key space of size k after 2 fault injections

k	0	1	2	3	4	5	6	7	8	$Pr(x \leq 8)$
$Pr(x = k)$	0.018	0.073	0.147	0.195	0.195	0.156	0.104	0.060	0.030	0.979

that the key space is reduced to at most 8 keys (77-bit leakage) after 2 fault injections, with probability 97.9%.

6.2 AES-128: Matching Best Known DFA Attack

AES [33] is considered as the standard block cipher for most applications. It was selected in 2001 by NIST through a public international competition. The design details can be found in [12]. It follows the description in Section 5 directly, with the ShiftRows operation representing shuffling and the MixColumns operation representing mixing. Moreover, since the MixColumns operation is designed to achieve the maximum branching number of 5, it follows immediately that if exactly a single byte at the input of MixColumns is active, all the four output bytes must be active. Hence, a uniform random byte fault model will serve the purpose of the attack described in Section 5. However, as shown in the attack in PRESENT, sometimes it is easier to achieve a single-bit fault, e.g. bit-sliced implementation. We are going to study both cases in this section. We refer to [33] for a full description of AES.

Similar to PRESENT, the number of solutions of Equation 2 where $S(x)$ is the Sbox function, $s_{\Delta x, \Delta y}$, is either 0, 2 or 4 for any given pair $(\Delta x, \Delta y)$ except (0,0). However, as the AES Sbox is an 8-bit function, this means that the probability of any of these pairs where the number of solutions is not zero is much lower. Moreover, the DDT of the AES Sbox is more structured, such that for any given non-zero Δx, the number of values Δy that correspond to 0, 2 and 4 solutions $s_{\Delta x, \Delta y}$ are 129, 126 and 1, respectively. Hence, for any input fault difference and a random input value, the expected number of solutions for the output value of that Sbox is $2^{1.0156}$ solutions, according to Theorem D.5. On the other hand, unlike the case of PRESENT, the diffusion in AES does not depend on any properties of the output value of the Sbox.

In order to asses the attack in Section 5, we need to study not just the properties of the DDT, but also the properties of the JDDT described in the attack. By injecting a single byte fault Δ in the input of the MixColumns operation, the four byte differences at the input of the Sboxes are $\{\delta, \delta, 2 \cdot \delta, 3 \cdot \delta\}$, or a rotation of this set, depending on which byte in the input is faulted. Hence, the number of solutions in the JDDT for an input difference δ and output difference $\{\Delta_1, \Delta_2, \Delta_3, \Delta_4\}$, is the number of solutions $\{y_1, y_2, y_3, y_4\}$, such that $\delta_1 = 2 \cdot \delta, \delta_2 = \delta, \delta_3 = 2\delta$, and $\delta_4 = 3 \cdot \delta$. This number is equal to $s_{2 \cdot \delta, \Delta_1} \times s_{\delta, \Delta_2} \times s_{\delta, \Delta_3} \times s_{3 \cdot \delta, \Delta_4}$. We can apply Theorem D.1 on the 32-bit function constructed by performing the MixColumns operation followed by 4 parallel Sboxes for two cases:

(1) $\delta \in \{0,1\}^8 - \{0\}^8$ is a uniformly random variable ($p_x = \frac{1}{255}$): This case is valid when the attack in Section 5 with a uniformly random fault. Given the output difference value, $\{\delta_1, \delta_2, \delta_3, \delta_4\}$ represent a 32-bit random vector, selected from 127^4 possible values. However, 24 conditions are imposed by the MixColumns equations. This limits the number of possibilities to $\frac{127^4}{2^{24}} = 2^{3.95}$. First, for simplicity, we assume that all the possibilities are equiprobable. Using Theorem D.5, the expected number of solutions for any given 32-bit difference at the output of the mixing operation is $2^{4 \times 1.0156} = 2^{4.0624}$ solutions, then the overall expected number of solutions is $2^{8.01}$. Since the AES state 4 columns (groups), step 1 of the attack in Section 5 reduce the last round key space from 2^{128} values to $|KS| = 2^{32.05}$. Applying step 2 of the attack, we get $|KS^{'}| = 2^{8.01}$. For AES-128, $|K_m| = |K_{10}|$ and a single round key is enough to deduce the master key and, hence, all the other round keys. Therefore, every candidate for K_{10} imposes 32 conditions on the 32 active bits of K_9, leading to $P_{ks} = 2^{-32}$. To sum up, after applying the two steps of the attack, the expected number of key candidates is $2^{8.06}$, with complexity $O(2^{32.05})$. To assess the expected number of key candidates after applying the attack twice, we use a similar binomial distribution to the one used to assess PRESENT. Since, $Pr(X = 0) = 1 - 2^{-111.88} \approx 1$, therefore, it is expected that the attacker can uniquely identify the key using two faults with overwhelming probability. A similar result has been achieved by Tunstall et al. in [38], where they concluded that when a single fault is injected in round 8 at the input of the MixColumns operation, the key space can be reduced to 2^8 candidates. The analysis was specific to AES-128 and it used an approximation regarding the expected number of solution for each Sbox. Hence, the analysis in our paper is more generic and more conservative, showing that the number of key candidates after one fault is expected to be $\approx$ 4.2% more than what they originally estimated.

(2) δ is the output difference of an AES Sbox triggered by a constant input difference: This case is valid the attack in Section 5 with a constant fault. As discussed earlier, for some implementations, e.g. bit-sliced software implementations, it might be easier to inject a fault in a specific bit of the state. In such cases we use a stronger fault model, where $\delta = \delta_o$, such that δ_o is a constant known to the attacker and has a Hamming weight of 1. For this model, the first part of the analysis is similar. However, every difference δ_i at the input of the Sbox is restricted to 127 out of 255 values (as a known difference Δx for the AES Sbox can lead to 127 possible values for Δy). Hence, the number of key candidates after step 1 is reduced by a factor of $\frac{127}{255}^4$, leading to a total number of key candidates $|KS| = 2^{28.02}$. In step 2, $|KS^{'}| = 2^{4.0624}$, since we need to solve only for δ_o. Since P_{ks} depends on the conditions derived from the key scheduling algorithm and not the fault model, it remains the same. Consequently, the overall number of key candidates for this fault model is $2^{0.09}$.

In order to verify the previous results, we have implemented the pre-computation algorithm described in Section 5 for each case. We calculated the average number of leaked bits for every possible output difference (255^4 possibilities), according to Theorem D.1. The computation was performed on an AMD Opteron 6378 quad-core processor. The average number of solutions for case 1 is $\approx 2^8$, and computing the JDDT (the pre-computation phase) takes around 14 CPU-Hours. For case 2, we need two JDDTs. The first one is for the first step of the attack, where δ can take 1 value with probability

2^{-6} and 126 values with probability 2^{-7}. The second table is for the second step of the attack, considering a constant input difference. The first table is computed in 30 CPU-Hours on the same machine and leads to $2^{7.03}$ solutions per group on average. The reason of the time overhead compared to the random case is to calculate the conditional probability distribution used in Theorem D.1. The second table is computed in 3 CPU-Minutes and leads to an average of $2^{4.034}$ solutions, for $\delta = 1$. The resulting key space has a size of $2^{0.154}$ or 1.1 candidates. Most of the time the key can be identified with a single fault.

6.3 SKINNY: Matching Best Known DFA Attack

A similar analysis to the analysis of SKINNY in this section was recently independtly reported by Vafaei et al. [40]. We briefly discussed the diffusion of SKINNY in section 4.1. While SKINNY does not exactly fall in the family of SPNs we are considering in this section, since the MixColumns matrix is not an MDS matrix, most of the analysis still applies to it. If the injected fault follows a uniform distribution, then according to Theorem D.2 and the fact that there are no Mix-Column equations to reduce the space of this fault, the column with 1 Sbox cannot be used to reduce the key space, but can only be used when the input difference follows a non-uniform distribution. For example, the JDDT of the function constructed by 3 of the SKINNY 4 bit Sbox, such that all the input difference have to be equal shows an expected reduction from 2^{12} to $2^{4.36029}$ solutions for a uniform fault model. By analyzing the MixColumns matrix of SKINNY, we can find the best location for injecting the fault in round $r - 2$ is any nibble/byte of the third row, i.e. b_8, b_9, b_{10} or b_{11} at the input of the MixColumns operations. The activity pattern of the last round Sboxes is as follows (up to a cyclic rotation of the columns):

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{15}$$

However, since the round key is added only to the first two rows, the key recovery method is slightly different from other AES-like cipher. The output value of the Sboxes in rows 2 and 3 in the last round is visible to the attacker. Hence, the attacker can use the invert those Sboxes and find the input differences to them. Since the input difference to all active Sboxes in the same column is the same, then the attacker knows the input difference to 3 of the Keyed Sboxes. Using this knowledge, he can reduce the key nibbles/bytes of these Sboxes to an expected value is $2^{1.4}$ key candidates for SKINNY-64-64 and $2^{2.88}$ for SKINNY-128-128, using Theorem D.5, reducing the size of the key space of the last round key to $\approx 2^{24.2}$ for SKINNY-64-64 and $\approx 2^{48.64}$ for SKINNY-128-128, while the size of the master key space is reduced to $2^{56.2}$ and $2^{112.64}$ respectively. Since there are four potential good fault locations, in Table 8 we sum up the vulnerable Sboxes in the last round corresponding to each of these faults. We can observe that after 4 pairs of fault and non-fault ciphertexts, all the Sboxes in the last round have been activated, with b_0, b_1, b_2 and b_3 activated twice. This leads to reducing the number of candidates of the last round to between an average of $2^{5.6}$ and $2^{11.52}$ candidates for SKINNY-64-64 and SKINNY-128-128 respectively. Overall, The last round key can be uniquely identified with high probability using 4 more pairs. By repeating the attack for the second to last round, we can reduce master key space to an expected value of $2^{11.2}$ and 2^{23}, for SKINNY-64-64 and SKINNY-128-128 respectively, after 8 pairs and uniquely identify the key with high probability of after 16 pairs.

Table 8: The active Sboxes in the last round of SKINNY for every fault location

Fault location in round $r - 2$	Vulnerable Sboxes in round r
b_8	b_0, b_2, b_4
b_9	b_1, b_3, b_5
b_{10}	b_2, b_0, b_6
b_{11}	b_3, b_1, b_7

With this result, SKINNY seems to be the most immune cipher against the three round attack we describe in this paper. However, we show next that the reason it is immune against this attack is the same reason it is vulnerable to a more efficient 4-round attack, unlike other ciphers.

6.3.1 *Four Round Attack on* SKINNY.

In this section we study the attack when the fault is injected in round $r - 3$ instead of $r - 2$. The reason this attack is possible is due to the fact that the diffusion in SKINNY is very slow. While it may not be possible to keep the number of interacting active Sboxes to zero after 4 rounds, we show that a value of only 2 interacting Sboxes can be achieved after 4 rounds. We consider the faults after 3 rounds (after applying ShiftRows and MixColumns respectively):

$$\begin{bmatrix} a & 0 & c & d \\ 0 & e & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & p & 0 & m \end{bmatrix} \tag{16}$$

$$\begin{bmatrix} a \oplus i & p & c & d \oplus m \\ a & 0 & c & d \\ i & e & 0 & 0 \\ a \oplus i & 0 & c & d \end{bmatrix} \tag{17}$$

It is worth mentioning the same property of SKINNY was used by Liu et al. [28], but in a different context and with less details. Since the last round key is added only to the top two rows, our goal is to find the output values of the top 8 Sboxes, given the information about the bottom 8. Hence, the attacker already knows the values of a, i, e, c and d and can reduce the key space of the corresponding key nibbles/bytes. In Table 9, we show the active Sboxes in the last round corresponding to each of the four good fault locations. After one fault the number of candidates of the last round key is expected to be 2^{19} and $2^{38.4}$ for both versions of SKINNY, by applying Theorem D.5. We can also observe that if we use 4 pairs, every Sbox in the last round is activated at least twice. While we can repeat the same attack one round earlier to get the full master key, we notice that the four pairs we have are already 4 valid pairs for the 3-round attack described in the previous section. Hence, we can use the same pairs to reduce the space of the round key at round $r - 1$. Combining the two attacks together, the number of candidates for the master key after only 4 pairs is expected to be $2^{5.6}$ for

Table 9: The active Sboxes in the last round of SKINNY for every fault location

Fault location in round $r-3$	Vulnerable Sboxes in round r
b_8	b_0, b_2, b_4, b_6, b_7
b_9	b_1, b_3, b_5, b_7, b_4
b_{10}	b_2, b_0, b_6, b_4, b_5
b_{11}	b_3, b_1, b_7, b_5, b_6

SKINNY-64-64 and $2^{11.52}$ for SKINNY-128-128. With 8 pairs, the master key can be uniquely identified with high probability.

The DFA attacks against SKINNY apply directly to any SKINNY-n-m version, as long as only *n* bits represent the fixed master key and the rest can be controlled by the attacker.

7 CONCLUSION

In this paper, we presented a generic DFA on SPNs. More specifically, we formulated an information-theoretic model of fault attack in the last round of an SPN and showed that for any non-uniform fault, the key space can always be reduced. We proposed an attack method which injects a single fault in the second last round of a class of SPNs and by using a novel tool, called *Joint Difference Distribution Table (JDDT)*, the master key of the cipher can be recovered. Our method was evaluated on various block ciphers, including PRESENT-80, PRESENT-128, GIFT-64, GIFT-128, AES-128, LED-64, and LED-128.

For the future work, we would like to extend our method to other block cipher designs. Also, we note that our work can serve as a basis for designing fault resistant block ciphers. Besides, as the automation of fault analysis is currently gaining attention of the community [10], the JDDT method for analysis of SPNs can potentially be fully automated, which can be integrated with some cryptographic design and analysis software tools.

REFERENCES

[1] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. 2010. How to flip a bit?. In *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International.* IEEE, 235–239.

[2] Martin R Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçın. 2014. Block ciphers–focus on the linear layer (feat. PRIDE). In *International Cryptology Conference.* Springer, 57–76.

[3] Nasour Bagheri, Reza Ebrahimpour, and Navid Ghaedi. 2013. New differential fault analysis on PRESENT. *EURASIP Journal on Advances in Signal Processing* 2013, 1 (2013), 145.

[4] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Mustafa Khairallah, and Thomas Peyrin. 2018. Protecting block ciphers against differential fault attacks without re-keying. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST).* IEEE, 191–194.

[5] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. 2017. GIFT: a small PRESENT. In *International Conference on Cryptographic Hardware and Embedded Systems.* Springer, 321–345.

[6] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. 2016. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Annual Cryptology Conference.* Springer, 123–153.

[7] Eli Biham and Adi Shamir. 1997. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference.* Springer, 513–525.

[8] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. 2007. PRESENT: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 450–466.

[9] Jakub Breier and Wei He. 2015. Multiple fault attack on present with a hardware trojan implementation in FPGA. In *Secure Internet of Things (SIoT), 2015 International Workshop on.* IEEE, 58–64.

[10] Jakub Breier, Xiaolu Hou, and Shivam Bhasin (Eds.). 2019. *Automated Methods in Cryptographic Fault Analysis* (1st ed.). Springer.

[11] Jakub Breier, Xiaolu Hou, and Yang Liu. 2018. Fault Attacks Made Easy: Differential Fault Analysis Automation on Assembly Code. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 2 (2018), 96–122.

[12] Joan Daemen and Vincent Rijmen. 2013. *The design of Rijndael: AES-the advanced encryption standard.* Springer Science & Business Media.

[13] Fabrizio De Santis, Oscar M Guillen, Ermin Sakic, and Georg Sigl. 2014. Ciphertext-only fault attacks on PRESENT. In *International Workshop on Lightweight Cryptography for Security and Privacy.* Springer, 85–108.

[14] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, and Florian Mendel. 2014. On the security of fresh re-keying to counteract side-channel and fault attacks. In *International Conference on Smart Card Research and Advanced Applications.* Springer, 233–244.

[15] Christoph Dobraunig, François Koeune, Stefan Mangard, Florian Mendel, and François-Xavier Standaert. 2015. Towards fresh and hybrid re-keying schemes with beyond birthday security. In *International Conference on Smart Card Research and Advanced Applications.* Springer, 225–241.

[16] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. 2003. Differential fault analysis on AES. In *International Conference on Applied Cryptography and Network Security.* Springer, 293–306.

[17] Christophe Giraud. 2004. DFA on AES. In *International Conference on Advanced Encryption Standard.* Springer, 27–41.

[18] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varıcı. 2014. LS-designs: Bitslice encryption for efficient masked software implementations. In *International Workshop on Fast Software Encryption.* Springer, 18–37.

[19] Shai Halevi, Don Coppersmith, and Charanjit Jutla. 2002. SCREAM: A software-efficient stream cipher. In *International Workshop on Fast Software Encryption.* Springer, 195–209.

[20] Wei He, Jakub Breier, and Shivam Bhasin. 2016. Cheap and cheerful: A low-cost digital sensor for detecting laser fault injection attacks. In *International Conference on Security, Privacy, and Applied Cryptography Engineering.* Springer, 27–46.

[21] Ludger Hemme. 2004. A differential fault attack against early rounds of (triple-) DES. In *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 254–267.

[22] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. 2014. Tweaks and keys for block ciphers: the TWEAKEY framework. In *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 274–288.

[23] Kitae Jeong and Changhoon Lee. 2012. Differential fault analysis on block cipher LED-64. In *Future Information Technology, Application, and Service.* Springer, 747–755.

[24] Mustafa Khairallah, Jakub Breier, Shivam Bhasin, and Anupam Chattopadhyay. 2019. Differential Fault Attack Resistant Hardware Design Automation. In *Automated Methods in Cryptographic Fault Analysis.* Springer, 209–219.

[25] Mustafa Khairallah, Rajat Sadhukhan, Radhamanjari Samanta, Jakub Breier, Shivam Bhasin, Rajat Subhra Chakraborty, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. 2018. DFARPA: Differential fault attack resistant physical design automation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018.* IEEE, 1171–1174.

[26] Benjamin Lac, Marc Beunardeau, Anne Canteaut, Jacques JA Fournier, and Renaud Sirdey. 2016. A First DFA on PRIDE: from Theory to Practice. In *International Conference on Risks and Security of Internet and Systems.* Springer, 214–238.

[27] Benjamin Lac, Anne Canteaut, Jacques Fournier, and Renaud Sirdey. 2017. DFA on LS-Designs with a Practical Implementation on SCREAM. In *International Workshop on Constructive Side-Channel Analysis and Secure Design.* Springer, 223–247.

[28] Guozhen Liu, Mohona Ghosh, and Ling Song. 2017. Security analysis of SKINNY under related-tweakey settings. *IACR Transactions on Symmetric Cryptology* 2017, 3 (2017), 37–72.

[29] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. 2010. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In *International Conference on Cryptology in Africa.* Springer, 279–296.

[30] Sikhar Patranabis, Jakub Breier, Debdeep Mukhopadhyay, and Shivam Bhasin. 2017. One plus one is more than two: a practical combination of power and fault analysis attacks on PRESENT and PRESENT-like block ciphers. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2017 Workshop on.* IEEE, 25–32.

[31] Conor Patrick, Bilgiday Yuce, Nahid Farhady Ghalaty, and Patrick Schaumont. 2016. Lightweight fault attack resistance in software using intra-instruction redundancy. In *International Conference on Selected Areas in Cryptography.* Springer, 231–244.

[32] Gilles Piret and Jean-Jacques Quisquater. 2003. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In *International workshop on cryptographic hardware and embedded systems.* Springer, 77–88.

[33] NIST FIPS Pub. 2001. 197: Advanced encryption standard (AES). *Federal information processing standards publication* 197, 441 (2001), 0311.

[34] Tiago Reis, Diego Aranha, and Julio López. 2017. PRESENT Runs Fast: Efficient and Secure Implementation in Software. In *Conference on Cryptographic Hardware and Embedded Systems (CHES'17)*.

[35] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. 2009. A Diagonal Fault Attack on the Advanced Encryption Standard. *IACR Cryptology ePrint Archive* 2009, 581 (2009).

[36] Kazuo Sakiyama, Yang Li, Mitsugu Iwamoto, and Kazuo Ohta. 2012. Information-theoretic approach to optimal differential fault analysis. *IEEE Transactions on Information Forensics and Security* 7, 1 (2012), 109–120.

[37] Ling Song and Lei Hu. 2013. Differential fault attack on the PRINCE block cipher. In *International Workshop on Lightweight Cryptography for Security and Privacy*. Springer, 43–54.

[38] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. 2011. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP international workshop on information security theory and practices*. Springer, 224–233.

[39] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. 2014. Differential fault analysis on the families of SIMON and SPECK ciphers. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 40–48.

[40] Navid Vafaei, Nasour Bagheri, Sayandeep Saha, and Debdeep Mukhopadhyay. 2018. Differential Fault Attack on SKINNY Block Cipher. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 177–197.

[41] Gaoli Wang and Shaohui Wang. 2010. Differential Fault Analysis on PRESENT Key Schedule. In *Computational Intelligence and Security (CIS), 2010 International Conference on*. 362–366. https://doi.org/10.1109/CIS.2010.84

[42] Meiqin Wang. 2008. Differential cryptanalysis of reduced-round PRESENT. In *International Conference on Cryptology in Africa*. Springer, 40–49.

[43] Fan Zhang, Shize Guo, Xinjie Zhao, Tao Wang, Jian Yang, Francois-Xavier Standaert, and Dawu Gu. 2016. A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers. *IEEE Transactions on Information Forensics and Security* 11, 5 (2016), 1039–1054.

A SPN VS DFA: GOOD DESIGN PRACTICES

While we have shown that it is almost impossible to design an SPN that provides the same level of security against DFA as the classical security, in the presence of non-uniform fault models, the results also show that not all the SPNs with the same classical security level provide the same level of security against DFA. For example, `GIFT-64` and `GIFT-128` required 12 pairs of faulty and non-faulty ciphertexts with very specific fault models, located in different locations, while `AES-128` requires only 2 pairs, with the fault injected in the same location every time and can be uniformly random. Here, we list some of the techniques we believe make DFA harder to perform and help `GIFT` be harder to break than the other ciphers considered in this paper:

(1) Using Sboxes with irregular DDTs and higher linearity means that, on average, the number of solutions when injecting faults is higher.
(2) Using smaller Sboxes means that the gap between the number of possible solutions for each Sbox before and after applying DFA is smaller, hence, the information leakage is lower.
(3) The lower the branching number of the diffusion layer, the harder it is to accelerate the attack and reduce the number of faults required.
(4) The smaller the ratio between the round key size and the master key size, the lesser the information leaked during applying DFA on the final rounds.
(5) Using a non-symmetric diffusion layer (e.g. `GIFT`) makes the JDDT more complex, as opposed to a symmetric diffusion layer (e.g. `PRESENT`).
(6) While it is a good practice to design the round key to be smaller than the master key (e.g. `LED-128`) and the block size (e.g. `GIFT`), it is not a good practice to skip adding the round key to some of the final Sboxes altogether, as this may leak vital information about other Sboxes in the same round (e.g. `SKINNY`).

Table 10: The probability of having a key space of size k for the last round key K_{31} after 2 fault injections

k	0	1	2	3	4	5	6	7	8
$Pr(x = k)$	0.779	0.195	0.0.024	0.002	1×10^{-4}	6×10^{-6}	2×10^{-7}	9×10^{-9}	2×10^{-10}

In general, as discussed in Section 4.1, the goal of the designer should be to minimize the number of active Sboxes before they start interacting with each other. This can be achieved by having a very slow diffusion when the number of active Sboxes in a given round is very small (1 or 2), and very fast diffusion, otherwise. For example, `SKINNY` has very slow diffusion, but even when 7 out of 16 Sboxes are active, the 4-round DFA attack we describe in Section 6.3.1 can be still mounted. However, even if the designer manages to come up with a design that satisfies all of these guidelines, it is still impossible to prevent DFA against SPNs completely. So the ultimate goal should be to increase the number of faults required and/or use other constructions that may prevent DFA.

B MORE CASE STUDIES TO OUR TECHNIQUES

B.1 `PRESENT-128` and Practical Implementations of `PRESENT`: Finding Optimal DFA Attack

B.1.1 `PRESENT-128`. The key scheduling algorithm for `PRESENT-128` is similar to present `PRESENT-80` (refer to [8], Appendix II, for full description), where

$$K_{31} = [\kappa_{127}\kappa_{126}\ldots\kappa_{64}] \tag{18}$$

$$K_{30} = [\kappa_{60}\kappa_{59}\ldots\tilde{\kappa_{19}}\tilde{\kappa_{18}}\tilde{\kappa_{17}}\tilde{\kappa_{16}}\tilde{\kappa_{15}}\ldots\kappa_{0}]\mathtt{truncate}(sb^{-1}([\kappa_{127}\kappa_{126}\kappa_{125}\kappa_{124}],3)) \tag{19}$$

We note that only 3 bits overlap between K_{31} and K_{30}: $\kappa127, \kappa126$, and $\kappa125$. However, since none of these bits is active in round 30, for both the fault locations we are considering, we conclude that there is no gain in performing step 2 while attacking `PRESENT-128`. Hence, the number of key candidates for K_{31} and K_m are 2^{31} and 2^{95}, respectively. An interesting result is that the increased security of `PRESENT-128` is not just due to the increased Key size, but due to the good design of the larger key scheduling algorithm, which leads to $P_{ks} = 1$. However, Table 10 shows that by applying the attack twice, the last round key K_{31} can be identified as 1 of 8 candidates with probability almost 1. Afterwards, the attack can be applied to a round-reduced version of `PRESENT-128` for each of these candidates, targeting K_{30} as the last round key, resulting in 2^{34} or 2^{6} master key candidates after applying one or two more faults in round 28, respectively.

B.1.2 Practical Implementations. The attacks described in this section require the injection of a fault $\delta = [1111]_2$ in the output of the sBoxLayer of round 29, in either nibble 7 or nibble 11. While this requirement can be challenging, it was shown in [9] that such fault is practically possible. Moreover, there are a couple of tricks that the attacker can use to get around this requirement. First, we notice that if instead of injecting $\delta = [1111]_2$ at the output of the Sbox, we inject $\delta' = [1000]_2$ at the corresponding input, there is a 0.25 chance that the correct fault is triggered at the output. We can observe the occurrence of the correct value by the number of active groups in round 31. Besides, if we inject 4 of such faults, there is a very high probability that the required fault occurs, and the other three pairs can be used to further decrease the key space size. The same applies for $\delta' = [1111]_2$, $\delta' = [0110]_2$, and $\delta' = [0111]_2$. Hence, depending on the precision of the equipment the attacker has, he can target 1-, 2- or 3-bit flips with probability 0.25 of getting the required difference, or 4-bit flips (with probability 1).

Moreover, for some specific implementations, the attack can be performed using even a single uniform random fault. In case of a software, bit-sliced implementation of width w, a uniform random fault is injected in the most significant bit of the input of the Sbox at nibble 7, round 29. Since each instance is triggered with probability 0.5, and the instances that are triggered have probability 0.25 of getting the required difference at the output of the Sbox at round 29, then it is expected that $\frac{w}{8}$ instances will have the required value and $\frac{w}{2}$ will be active in general, potentially allowing to even uniquely identify the key using only a single fault. This expectation can be achieved even using architecture as small as 8-bit micro-controllers.

B.2 GIFT-64: New Results

GIFT cipher was proposed in CHES 2017 by Banik et al. [5] as a more lightweight version of PRESENT. It has two versions: GIFT-64 and GIFT-128. Both these versions have a 128-bit master key, but they differ in the block size, 64 bits and 128 bits, respectively. In this section we analyze only the first version, noting that the techniques from our paper can bet extended with slight modifications to attack GIFT-128, as well. The differences between GIFT-64 and PRESENT-128 are as follows:

(1) It runs for 28 rounds only.
(2) The Sbox and mixing layer are different.
(3) The Key Scheduling algorithm is also different, where 32 bits are extracted from the key state every round, followed by a linear state update. The property that is of interest to our analysis is that every 4 consecutive round keys are independent and uniquely identify the master key.
(4) Every round, half of the bits (32 bits), only, are mixed with key bits.

From the structure of GIFT-64, we conclude that the best single-nibble fault value in round 26 is $[1111]_2$ at the output of one of the Sboxes. Similar to to PRESENT, that fault value can be achieved by injecting a fault $[1000]_2$ at the input of the Sbox, with probability 25% (refer to [5], Appendix C.2, for the DDT of GIFT Sbox). However, due to the properties of the mixing layer, regardless of which nibble the fault is injected in, such fault will always trigger the four groups of round 27 by the same four different input differences $\{[0001]_2, [0010]_2, [0100]_2, [1000]_2\}$, up to a cyclic rotation operation. Table 11 shows the corresponding rows of the DDT of GIFT Sbox. Instead of selecting the difference with the largest average Hamming weight, as in the case of PRESENT, in order to find the average number of active Sboxes in the last round, in the case of GIFT-64 the average number of Sboxes in the last round is always

Table 11: Part of the DDT of the Sbox used in GIFT

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	μ_{HW}
1	0	0	0	0	0	2	2	0	2	2	2	2	2	0	0	2	2.25
2	0	0	0	0	0	4	4	0	0	2	2	0	0	2	2	0	2.25
4	0	0	0	2	0	4	0	6	0	2	0	0	0	2	0	0	2.5
8	0	0	0	4	0	0	0	4	0	0	0	4	0	0	0	4	3

Table 12: Part of the auxiliary DDT of the Sbox used in GIFT with respect to the active key bits

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	EXP(SOLs)
1	0	0	0	0	0	2	2	0	1	2	2	2	1	0	0	2	$2^{0.75}$
2	0	0	0	0	0	4	4	0	0	2	2	0	0	2	2	0	$2^{1.5}$
4	0	0	0	2	0	4	0	4	0	2	0	0	0	2	0	0	$2^{1.625}$
8	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	2	2^{1}

Table 13: The probability distribution of different output bit differences when a single input bit is active for GIFT Sbox

	0	1	2	3	EXP(Sols)
1	0.5	0.5	0.5	0.75	$2^{5.452}$
2	0.5	0.5	0.75	0.5	$2^{6.204}$
4	1	0.5	0.75	0.5	$2^{3.384}$
8	1	1	0.5	0.5	$2^{3.059}$

10. Since only two key bits are added to the output of each Sbox, we need to compute an auxiliary DDT, which corresponds to the number of solutions of the active key bits, shown in Table 12. Since, in round 28, each group is triggered with a different input difference, in Table 13 we calculate the different probabilities for each Sbox in round 28 to be active. Combined with the number of solutions in Table 12, for every possible input difference, the expected number of candidates for each group in the last round is computed, which leads to an overall number of $2^{18.1}$ key candidates for K_{28}. In order to identify K_{28} uniquely, we need 3 repetitions of the whole attack, respectively, which reduces the master key space to 2^{96} keys. The attack then needs to be repeated four times for the 4 last round keys, required 12 fault injections, on average.

B.3 GIFT-128: New Results

Any two consecutive rounds of GIFT-128 can be viewed as two parallel independent instances of GIFT-64. We use this representation for rounds 38 and 39, while round 40 is the final round and, as explained in Section 5, the shuffling and mixing operations in the last round have no effect on our analysis. Hence, the same analysis used for GIFT-64 can be used for GIFT-128. First, 3 faults are used to recover half the bits of K_{40}, then another three faults are used to recover the other half. The attack is then repeated for K_{39}.

GIFT-128 is very similar to GIFT-64 except for the following differences:

(1) The block size is 128 bits, divided into 8 groups of 4 nibbles, each.
(2) 64-bit round keys are used.
(3) Only every two successive round keys are independent, as opposed to 4 rounds in the case of GIFT-64.
(4) The description of the shuffling operation is slightly different. It can be viewed as two steps; matrix transposition, followed by interleaving.

From this description, any two successive rounds of GIFT-128 can be viewed as a pair of parallel and independent two rounds of GIFT-64. Since in the case of GIFT-128, we need only two round

Table 14: The active Sboxes in the last round depending on the fault location

Faulty Location	Active Sboxes
1	0,2,3,6,7,8,11,12,15
5	0,1,4,5,6,9,10,13,14
13	0,2,3,6,7,8,11,12,15

keys, the number of faults and complexity of the attack is exactly the same as in the case of GIFT-64.

B.4 PRIDE: Finding Optimal DFA Attack

PRIDE [2] is an SPN-based cipher that can be considered as from the same family of ciphers as PRESENT and GIFT, but targeted towards low latency application. Hence, it uses a more complicated diffusion layer to achieve faster diffusion. In [26], the authors showed that by flipping 16 adjacent bits at the input of the linear diffusion layer in the second to last round, all the Sboxes in the last round can be activated with known input difference. However, since flipping 16 bits is not an easy task and requires high precision, we analyze the last round limiting the number of bits to be flipped to 4 (single-fault model). The location of the fault is the same as [26], round $r-1$. Injecting faults in earlier rounds does not help due to the fast diffusion of PRIDE which will increase the number of active interacting Sboxes. This limits the possible fault locations to 16 nibbles. We tested all the 16 possibilities and found out that the nibbles that maximize the number of active Sboxes in the last round are nibbles: 1, 5 and 13, with a maximum of 9 active Sboxes in the last round. The active Sboxes are shown in Table 14. We notice that if we use all these three pairs, each Sbox has appeared at least once. Hence, we conclude that 4 pairs of faulty and non-faulty ciphertexts are required on average to uniquely identify the last round key, and then the attack can be repeated for the previous round. This is double the number of faults required in [26], but using a simpler and less demanding fault model. The exact complexities are given in Section 1.

C MORE DETAILS ON THREE ROUND DFA ATTACK ON SPNS

Here we state the assumptions of the cipher we consider:

- Any intermediate value of the cipher is referred to as the *state*.
- The state consists of w words, each of $b-$bits.
- The state can be divided into g groups and each group consists of 4 words.
- Thus, we represent the state as a $4 \times g$ array.
- Each column of the array is a group. We refer to the $(i, j)-$entry $(0 \leq i \leq 4, 0 \leq j \leq g)$ of the array as the ith word of group j.
- The cipher consists of r rounds, each round consists of the following operations in the exact order: substitution, shuffling, mixing and key xor.

Figure 2 describes the following process:

(1) If a word is changed due to the fault injection, we say that this word is *active*.

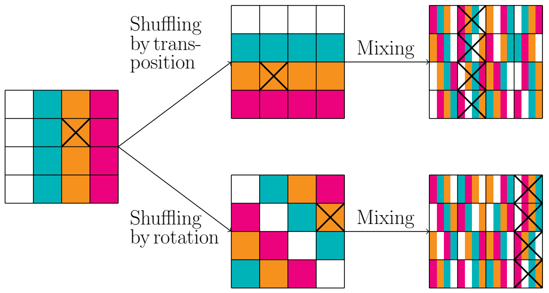

Figure 3: Fault propagation pattern for different diffusion functions in SPNs.

(2) We assume the shuffling takes input a 4×4 array and outputs its transpose.
(3) We inject a fault with fault mask δ_o at the input of shuffling in round $r-2$. Let us assume the fault is injected in the 2nd word of group 0. It works similarly for a fault injected in a different word.
(4) Since shuffling in round $r-2$ changes the locations of each word, the fault mask is relocated to a word in a different group.
(5) Mixing in round $r-2$ then produces a state which contains exactly one group whose 4 words are all active. As mixing is a linear operation, we denote the resulting differences as $\ell_0(\delta_o), \ell_1(\delta_o), \ell_2(\delta_o), \ell_3(\delta_o)$ for the 4 active words. In our figure, this particular group is group 2.
(6) After the substitution operations in round $r-1$, the 4 active words remain active and the other words remain non-active. We denote those 4 new differences as $\delta_1^s, \delta_2^s, \delta_3^s, \delta_4^s$ respectively.
(7) The shuffling in round $r-1$ distribute the 4 differences into 4 different groups.
(8) The mixing in round $r-1$ then makes all the 16 words in those 4 groups active. The differences in groups 0, 1, 2, 3 are linear functions of $\delta_1^s, \delta_2^s, \delta_3^s, \delta_4^s$ respectively. For the group affected by δ_j^s, we denote $f_i^{(j)}(\delta_j^s)$ for the difference of its ith word as a linear function of δ_j^s.
(9) After the substitution in round r, the same 16 words will remain active. We denote the difference corresponding to $f_i^{(j)}(\delta_j^s)$ as Δ_{ij}^s.
(10) Shuffling in round r again changes the locations of each difference.
(11) Mixing in round r mixes the differences with linear operations. In the figure, g_0, g_1, g_2, g_3 are linear functions.
(12) The differences produced after mixing in round r are what can be observed by attacker from the ciphertext, where we denote the difference in the ith word of group j as Δ_{ij}.

The JDDT gives us the following information:

(1) Given δ_o, what are the possible values of $\delta_1^s, \delta_2^s, \delta_3^s, \delta_4^s$.
(2) Given $\Delta_{00}, \Delta_{10}, \Delta_{20}, \Delta_{30}$, what are the possible values of δ_1^s.
(3) Given $\Delta_{01}, \Delta_{11}, \Delta_{21}, \Delta_{31}$, what are the possible values of δ_2^s.
(4) Given $\Delta_{02}, \Delta_{12}, \Delta_{22}, \Delta_{32}$, what are the possible values of δ_3^s.
(5) Given $\Delta_{03}, \Delta_{13}, \Delta_{23}, \Delta_{33}$, what are the possible values of δ_4^s.

D PROOFS FOR SECTION 3

Theorem D.1. *If ΔX is sampled from $\mathcal{S}$, such that $|\mathcal{S}| = z$ and $P(\Delta X) = p_x$.then the expected number of leaked bits of K, when ΔY is observed is*

$$n - \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x,\Delta y}}{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}} log(\frac{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}}{p_x}) \qquad (20)$$

Proof. Since, using Bayes' law, we have

$$\begin{aligned}
&P(\Delta X = \Delta x | \Delta Y = \Delta y)\\
=&\frac{P(\Delta Y = \Delta y|\Delta X = \Delta x)P(\Delta X = \Delta x)}{P(\Delta Y = \Delta y)}\\
=&\frac{P(\Delta Y = \Delta y|\Delta X = \Delta x)P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} P(\Delta Y = \Delta y|\Delta X = \Delta x_j)P(\Delta X = \Delta x_j)}\\
=&\frac{\frac{s_{\Delta x,\Delta y}}{2^n} P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} P(\Delta X = \Delta x_j)}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&H(K|Z_1, Z_2)\\
&= \sum_{\Delta X \in \mathcal{S}} P(\Delta X = \Delta x|\Delta Y = \Delta y)\times\\
&(H(X|\Delta X = \Delta x, \Delta Y = \Delta y) - log(P(\Delta X = \Delta x|\Delta Y = \Delta y)))\\
&= \sum_{\Delta X \in \mathcal{S}} \frac{\frac{s_{\Delta x,\Delta y}}{2^n} P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} P(\Delta X = \Delta x_j)} \times\\
&\left(log(s_{\Delta x,\Delta y}) - log\left(\frac{\frac{s_{\Delta x,\Delta y}}{2^n} P(\Delta X = \Delta x)}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} P(\Delta X = \Delta x_j)} \right)\right)\\
&= \sum_{\Delta X \in \mathcal{S}} \frac{\frac{s_{\Delta x,\Delta y}}{2^n} p_x}{\sum_{\Delta x_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} p_{x_j}} log(\frac{2^n \sum_{\Delta X_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} p_{x_j}}{p_x})\\
&= \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x,\Delta y}}{2^n \sum_{\Delta X_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} p_{x_j}} (n + log(\frac{\sum_{\Delta X_j \in \mathcal{S}} \frac{s_{\Delta x_j,\Delta y}}{2^n} p_{x_j}}{p_x}))\\
&= \sum_{\Delta X \in \mathcal{S}} \frac{p_x s_{\Delta x,\Delta y}}{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}} log(\frac{\sum_{\Delta X_j \in \mathcal{S}} s_{\Delta x_j,\Delta y} p_{x_j}}{p_x})
\end{aligned}$$

□

Corollary D.2. *Given a pair of faulty and correct ciphertexts Z_1 and Z_2, if $\Delta X \in \{0,1\}^n$ is a uniform random variable, then $H(K|Z_1 Z_2) = n$, regardless of the properties of the function $S(x)$.*

Proof. Since

$$\begin{aligned}
&H(\Delta X|\Delta Y = \Delta y)\\
&= - \sum_{\Delta x \in \{0,1\}^n} Pr(\Delta X = \Delta x|\Delta Y = \Delta y) log(Pr(\Delta X = \Delta x|\Delta Y = \Delta y))\\
&= - \sum_{\Delta x \in \{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} log(\frac{s_{\Delta x,\Delta y}}{2^n}),
\end{aligned}$$

and

$$\begin{aligned}
&H(X_1|\Delta X \Delta Y)\\
&= \sum_{\Delta x \in \{0,1\}^n} H(X_1|\Delta X = \Delta x, \Delta Y = \Delta y) Pr(\Delta X = \Delta x, \Delta Y = \Delta y)\\
&= \sum_{\Delta x \in \{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} log(s_{\Delta x,\Delta y}),
\end{aligned}$$

from Equation (1), we have

$$\begin{aligned}
&H(K|Z_1 Z_2)\\
&= \sum_{\Delta x \in \{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} log(s_{\Delta x,\Delta y}) - \frac{s_{\Delta x,\Delta y}}{2^n} log(\frac{s_{\Delta x,\Delta y}}{2^n})\\
&= \sum_{\Delta x \in \{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} log(\frac{2^n}{s_{\Delta x,\Delta y}} s_{\Delta x,\Delta y}) = n \sum_{\Delta x \in \{0,1\}^n} \frac{s_{\Delta x,\Delta y}}{2^n} = n
\end{aligned}$$

□

Corollary D.3. *If $\Delta X = \Delta x$ (constant), then using one pair (Z_1, Z_2), the key space can be reduced from 2^n to $s_{\Delta x,\Delta y}$.*

Proof. If ΔX is constant, then $H(\Delta X|\Delta Y) = 0$ and $H(X_1|\Delta X \Delta Y) = H(X_1|\Delta X = \Delta x, \Delta Y = \Delta y) = log(s_{\Delta x,\Delta y})$. Hence, $H(K|Z_1 Z_2) = log(s_{\Delta x,\Delta y})$. □

Corollary D.4. *Only linear (affine) Boolean functions achieves the theoretical security bound $H(K|Z_1 Z_2) = n \forall \Delta x$, regardless of the distribution of ΔX.*

Proof. In order for a function $S(x)$ to achieve $H(K|Z_1 Z_2) = n \forall \Delta x$ for any distribution, $s_{\Delta x,\Delta y}$ must be equal to 2^n or $0 \ \forall \ \Delta x, \Delta y \in \{0,1\}^n$, which is achieved only by linear and affine functions. □

Theorem D.5. *If $\Delta X = \Delta x$ (constant), then the expected number of leaked bits of K is $n - \sum_{\Delta y \in \{0,1\}^n} log(s_{\Delta x,\Delta y}) P(\Delta Y = \Delta y|\Delta X = \Delta x)$.*

Proof. We have

$$\begin{aligned}
&H(K|\Delta X)\\
&= \sum_{\Delta y \in \{0,1\}^n} H(X|\Delta X, \Delta y) P(\Delta Y = \Delta y|\Delta X = \Delta x)\\
&= \sum_{\Delta y \in \{0,1\}^n} log(s_{\Delta x,\Delta y}) Pr(\Delta Y = \Delta y|\Delta X = \Delta x).
\end{aligned}$$

□

E THE DETAILS OF THE GIFT-128 ANALYSIS

GIFT-128 is very similar to GIFT-64 except for the following differences:

(1) The block size is 128 bits, divided into 8 groups of 4 nibbles, each.
(2) 64-bit round keys are used.
(3) Only every two successive round keys are independent, as opposed to 4 rounds in the case of GIFT-64.
(4) The description of the shuffling operation is slightly different. It can be viewed as two steps; matrix transposition, followed by interleaving, as shown in Equation 21.

$$\begin{bmatrix} b_j^{(0)} & b_j^{(1)} & b_j^{(2)} & b_j^{(3)} & b_j^{(4)} & b_j^{(5)} & b_j^{(6)} & b_j^{(7)} \\ b_j^{(8)} & b_j^{(9)} & b_j^{(10)} & b_j^{(11)} & b_j^{(12)} & b_j^{(13)} & b_j^{(14)} & b_j^{(15)} \\ b_j^{(16)} & b_j^{(17)} & b_j^{(18)} & b_j^{(19)} & b_j^{(20)} & b_j^{(21)} & b_j^{(22)} & b_j^{(23)} \\ b_j^{(24)} & b_j^{(25)} & b_j^{(26)} & b_j^{(27)} & b_j^{(28)} & b_j^{(29)} & b_j^{(30)} & b_j^{(31)} \end{bmatrix} \rightarrow \tag{21}$$

$$\begin{bmatrix} b_j^{(0)} & b_j^{(4)} & b_j^{(8)} & b_j^{(12)} & b_j^{(16)} & b_j^{(20)} & b_j^{(24)} & b_j^{(28)} \\ b_j^{(1)} & b_j^{(5)} & b_j^{(9)} & b_j^{(13)} & b_j^{(17)} & b_j^{(21)} & b_j^{(25)} & b_j^{(29)} \\ b_j^{(2)} & b_j^{(6)} & b_j^{(10)} & b_j^{(14)} & b_j^{(18)} & b_j^{(22)} & b_j^{(26)} & b_j^{(30)} \\ b_j^{(3)} & b_j^{(7)} & b_j^{(11)} & b_j^{(15)} & b_j^{(19)} & b_j^{(23)} & b_j^{(27)} & b_j^{(31)} \end{bmatrix} \tag{22}$$

From this description, any two successive rounds of `GIFT-128` can be viewed as a pair of parallel and independent two rounds of `GIFT-64`. One instance takes the following nibbles as an input:

$$\begin{bmatrix} b_j^{(0)} & b_j^{(1)} & b_j^{(2)} & b_j^{(3)} \\ b_j^{(8)} & b_j^{(9)} & b_j^{(10)} & b_j^{(11)} \\ b_j^{(16)} & b_j^{(17)} & b_j^{(18)} & b_j^{(19)} \\ b_j^{(24)} & b_j^{(25)} & b_j^{(26)} & b_j^{(27)} \end{bmatrix} \tag{23}$$

while the other one takes the following input:

$$\begin{bmatrix} b_j^{(4)} & b_j^{(5)} & b_j^{(6)} & b_j^{(7)} \\ b_j^{(12)} & b_j^{(13)} & b_j^{(14)} & b_j^{(15)} \\ b_j^{(20)} & b_j^{(21)} & b_j^{(22)} & b_j^{(23)} \\ b_j^{(28)} & b_j^{(29)} & b_j^{(30)} & b_j^{(31)} \end{bmatrix} \tag{24}$$