Multi-Stage Fault Attacks on Block Ciphers

Philipp Jovanovic, Martin Kreuzer, Ilia Polian
Fakultät für Informatik und Mathematik
Universität Passau
94032 Passau, Germany
{*Philipp.Jovanovic, Martin.Kreuzer, Ilia.Polian*}*@uni-passau.de*

***Abstract*—This paper introduces Multi-Stage Fault Attacks, which allow Differential Fault Analysis of block ciphers having independent subkeys. Besides the specification of an algorithm implementing the technique, we show concrete applications to LED-128 and PRINCE and demonstrate that in both cases approximately 3 to 4 fault-injections are enough to reconstruct the full 128-bit key.**

***Keywords*-cryptanalysis; Differential Fault Analysis; LED-128; lightweight block cipher; Multi-Stage Fault Attack; PRINCE;**

I. Introduction

Small mobile and embedded devices, like RFID chips and nodes of sensor networks, increasingly find their way into our everyday life. In many cases they are utilised to process sensitive (personal) data, for example in the form of financial and medical information, which requires protection against unauthorised access. The design of cryptographic primitives that provide acceptable security and can be implemented on devices with strictly limited resources is very challenging and has raised significant interest in the last few years. Numerous new algorithms have been proposed to address those problems including lightweight block ciphers such as PRESENT [3], PRINCE [5], and LED [7]

On the other hand, with the ubiquitous proliferation and easy accessibility of such devices, new types of attacks, known as *side-channel analysis*, examine the implementation of cryptographic primitives. *Differential Fault Analysis* [2], which combines fault attacks [4] with differential cryptanalysis [1], is particularly effective and was employed for successful attacks on a variety of ciphers, including Trivium [8], AES [11] and LED-64 [9]

Symmetric ciphers encrypt information using a *secret key* k. Classical ciphers, such as DES or AES, incorporate a *key schedule* which generates a number of *subkeys* from k using certain transformations. The subkeys are used for individual steps, or *rounds*, of the encryption algorithm. Several state-of-the-art ciphers have no key schedule: their subkeys are simply independent parts of k. For example, LED-128 uses a 128-bit key $k = k_0 \parallel k_1$ which consists of two 64-bit subkeys k_0 and k_1 that are used during encryption. One argument for avoiding the key schedule is the lower cost of hardware implementations. Furthermore, several attacks utilise the dependencies between different rounds of the algorithm introduced by the key schedule; such dependencies are avoided when subkeys are independent.

In this paper, we propose a generic differential fault analysis technique for ciphers with s independent subkeys. The attack is mounted in s stages, where each stage produces a set of candidates for one subkey. The number of fault injections per stage is variable; in general, performing more fault injections will narrow down the set of subkey candidates. We introduce an algorithm which minimises the overall number of required fault injections while producing a number of key candidates that can be analysed by brute-force search. It can be regarded as a generalisation of the "peeling-off" approach [6], where the last subkey is derived and the cipher is reduced by partial decryption using that subkey. In contrast to [6], our algorithm does not require the definite knowledge of the last subkeys but works with a (comparatively small) set of subkey candidates that include the actual (correct) subkeys together with a number of wrong guesses. The algorithm uses special *threshold variables* τ_i to adaptively distribute the fault injections between stages such as to keep the overall effort feasible.

We report the application of the algorithm to complete attacks against two recent ciphers: LED-128 and PRINCE. The attack on LED-128 is the first one based purely on fault injections. The attack mentioned in [9] assumed the capability of the adversary to temporarily set one of the 64-bit subkeys to 0, in addition to applying controlled fault injections to the hardware. The attack on PRINCE has been developed independently and simultaneously by a different group and published online [10]. Note that these attacks on particular ciphers serve as illustrations of the generic technique. We verified our algorithm by simulating 10,000 attacks, and we were able to derive key candidate spaces that were sufficiently small for brute-force search using roughly 3 fault injections for LED-128 and between 3 and 4 fault injections for PRINCE on average.

The work is structured as follows: Section II introduces the Multi-Stage Fault Attack algorithm in its most general form, tailored to attack block ciphers using independent subkeys, i.e. block ciphers having no key schedule. Sections III and IV first recall the designs of LED and PRINCE, respectively, then show how to cryptanalyse the ciphers using Differential Fault Analysis, and finally give some experimental results on the attacks. Section V concludes the paper and presents an outlook on future work.

II. The Multi-Stage Fault Attack Framework

Let F be a block cipher with block size 2^n ($n \in \mathbb{N}$). We assume that the secret key k is written as a concatenation of independent subkeys $k = k_0 \parallel k_1 \parallel \cdots \parallel k_{s-1}$ such that each subkey is of size 2^n. Further, we suppose that the encryption algorithm works on 2^m-bit sized parts (for $m = 2$ those are called *nibbles*) of the 2^n-bit state. For the 64-bit ciphers LED-128 and PRINCE two subkeys are used and we have $n = 6$, $s = 2$, and $m = 2$. The proposed Multi-Stage Fault Attack is a known-plaintext attack and proceeds in s *stages* (one stage per subkey).

The attack starts by encrypting the known plaintext and recording the obtained ciphertext c. Each stage $i \in \{0, 1, \ldots, s-1\}$ consists of one or several fault injections. The particular parameters of a fault injection (location and time) depend on the cipher under analysis and on the capabilities of the attacker. For example, ciphers based on

Substitution-Permutation Networks typically require fault injection two rounds before termination for successful differential fault analysis. Let the fault induced during the j-th fault injection of stage i be denoted by f_{ij}, and let c_{ij} be the ciphertext obtained by the encryption affected by this fault. Note that c_{ij} is observable by the attacker, who is assumed to have physical access to the hardware into which she injects faults. However, she may or may not know which fault f_{ij} was injected, as many physical fault-injection techniques do not allow perfect control of the bits that flip as a result of the disturbance. The assumptions on the fault-injection capabilities must be formalised in a *fault model*.

In this work, we employ two fault models: *random and known fault model* (RKF) and *random and unknown fault model* (RUF). Both models assume that a fault perturbs one nibble (2^m-bit sized part) of the state of the cipher while leaving its other $(2^n - 2^m)$ bits unaffected. We represent f_{ij} as a bit string with values 1 on the positions where the state is flipped, i.e., the fault injection is described by an addition (bitwise XOR) of f_{ij} to the state. Consequently, there are at most $2^{n-m} \cdot (2^{(2^m)} - 1)$ different faults in a particular stage (which amounts to 240 in case of LED-128 and PRINCE). The RKF model assumes that the attacker can target a specific nibble, e.g., the very first one that includes state bit positions 0 through $2^m - 1$. The RUF model assumes that the fault injection will perturb a randomly selected nibble, and it cannot be observed which nibble was affected. The RUF model is weaker and therefore easier to match by practical fault-injection equipment, but it requires more complex mathematical analysis.

We denote by $\mathsf{Analyse}(c, c_{ij})$ a cipher-dependent procedure that performs differential fault analysis and yields a set K_{ij} of candidates for the i-th part k_i of the secret key. We will introduce two instances of the procedure $\mathsf{Analyse}(c, c_{ij})$ for the ciphers LED-128 in Section III and PRINCE in Section IV. Performing multiple fault injections during the same stage results in multiple invocations of $\mathsf{Analyse}(c, c_{ij})$ for different c_{ij} and therefore results in different sets of subkey candidates K_{ij}. Since the correct subkey k_i must be contained in all K_{ij}, it must also be contained in their intersection $K_i = \bigcap K_{ij}$, which is frequently much smaller than the individual sets K_{ij}. Consequently, multiple fault injections reduce the size of the subkey candidate set. Note that no reduction occurs if the same fault is injected multiple times, resulting in identical c_{ij}; in that case, the fault injection must be repeated.

After all stages have been performed, the final candidate set can be obtained by computing the Cartesian product $K = K_0 \times \cdots \times K_{s-1}$, where K_i are subkey candidate sets calculated during the individual stages. As it will become apparent, it is possible to further reduce this set and therefore the complexity of the subsequent brute-force search.

The Multi-Stage Fault Attack algorithm incorporates a mechanism to balance between the available computational power and the number of fault injections necessary in order to successfully execute the attack. As was observed above, more fault injections in stage i will reduce the set of subkey candidates K_i. Let T be an estimate of time complexity of one invocation of procedure $\mathsf{Analyse}(c, c_{ij})$ (either actual run-time in milliseconds or number of operations). We define a sequence of *threshold values* $\tau_0, \ldots, \tau_{s-1}$ which have the same unit as T. The value of τ_i roughly represents the amount of computational power allocated to stage i; it will be used to set an upper bound for the number of invocations of procedure $\mathsf{Analyse}(c, c_{ij})$ in stage i. In stage 0 at the beginning of the algorithm, fault injections are continued until subkey candidate set K_0 becomes so small that $T \cdot \#K_0 < \tau_0$ holds. In stage i, subkey candidate sets $K_0, \ldots, K_{i-1}$ have been calculated already. Each subkey combination $(k_0, \ldots, k_{i-1}) \in K_0 \times \cdots \times K_{i-1}$ is used to partially decrypt c and c_{ij}, and procedure $\mathsf{Analyse}$ is applied to the obtained intermediate states. The worst-case number of procedure invocations is $\#(K_0 \times \cdots \times K_{i-1})$, and the number of fault injections is set such that K_i is sufficiently small to fulfill $T \cdot \#(K_0 \times \cdots \times K_i) < \tau_i$.

The formal description of the attack algorithm is as follows.

1) The Multi-Stage Fault Attack Algorithm.: The algorithm iteratively computes $K = K_0 \times \cdots \times K_{s-1}$. In the beginning, $i \longleftarrow 0$, $j \longleftarrow 0$, and $K \longleftarrow \emptyset$.

(1) Let $K_{ij} \longleftarrow \emptyset$. Inject fault f_{ij} according to the fault model and obtain a faulty ciphertext c_{ij}.
(2) For every $x \in K$, partially decrypt c and c_{ij} and obtain intermediate correct and faulty states v_x and v'_x. For stage 0 we have $K = \emptyset$ and $(v_x, v'_x) := (c, c_{ij})$.
(3) Apply $\mathsf{Analyse}(v_x, v'_x)$ and obtain a key candidate set $K_{ij}(x)$ depending on x.
(4) If $K_{ij}(x) \neq \emptyset$, append this set as a new element to K_{ij}. Otherwise, discard x from K. If there is an untried $x \in K$, continue with the next x in step (2).
(5) If $j > 0$, set $K_{ij} \longleftarrow K_i \cap K_{ij}$. This intersection is computed pairwise as the elements of K_i and K_{ij} are sets.
(6) Set $K_i \longleftarrow K_{ij}$. If $T \cdot \#(K \times K_i) < \tau_i$, set $K \longleftarrow K \times K_i$ and go to next stage ($i \longleftarrow i + 1$, $j \longleftarrow 0$), otherwise inject an additional fault ($j \longleftarrow j + 1$).
(7) If $i = s$ then stop and return K. Otherwise, start again from step (1).

In step (6) we can leverage the knowledge which x led to which key candidate set $K_{ij}(x)$ by computing $\{x\} \times K_{ij}(x)$ instead of the complete Cartesian product.

III. Application to LED-128

We first recall the design of the LED family of lightweight block ciphers [7] and then show how to cryptanalyse LED-128 using a Multi-Stage Fault Attack.

A. Specification of LED

The LED family of lightweight block ciphers features a 64-bit block size and has two instances, LED-64 and LED-128, with keysizes of 64 and 128 bit. The design of LED is heavily inspired by AES. Its basic layout is a Substitution-Permutation Network which is composed of the operations AddKey (AK), AddConstants (AC), the SBox-layer SubCells (SC), which re-uses the PRESENT SBox [3], and has a linear layer. The latter consists of the two operations MixColumnsSerial (MCS), where the state (represented by a 4×4 matrix over $\mathbb{F}_{16}$) is multiplied by a MDS matrix, and ShiftRows (SR), which cyclically shifts the rows of the state. One of the prominent features of LED is that it has no key schedule. Instead, the key material is applied directly for input-/output-whitening and each time four rounds of encryption have been executed. In the case of LED-64, the secret key is used as given, whereas in LED-128 the key is decomposed into two 64-bit subkeys which are applied alternatingly.

For more details we refer the reader to the original specification [7] of LED. Next we continue with the cryptanalysis of LED-128 using the Multi-Stage Fault Attack algorithm.

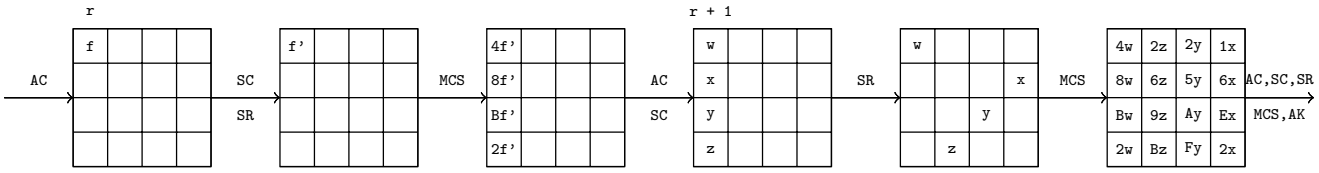

Figure 1. Fault propagation in LED over two rounds.

B. A Multi-Stage Fault Attack on LED-128

The attack on LED-128 is based on the Differential Fault Analysis of LED-64 [9], which employs one fault injection and reduces the number of key candidates to $2^{19} - 2^{26}$, a number that can be handled by exhaustive search. Applying this attack to LED-128 would shrink the key space for the last applied subkey k_0, but considering all combinations of up to 2^{26} candidates for k_0 and 2^{64} possibilities for the second subkey k_1 is clearly infeasible. The Multi-Stage Fault Attack algorithm solves this problem in two stages $i \in \{0, 1\}$, one for the reconstruction of each 64-bit subkey $k_i = k_{i,0} \parallel \cdots \parallel k_{i,15}$ with nibbles $k_{i,0}, \ldots, k_{i,15}$. Before we outline the details of the attack on LED-128, we first introduce notation.

Let w, x, y, z, u_0, u_1, u_2, u_3, v_j, v'_j and p_j be variables over $\mathbb{F}_{16}$ for $j \in \{0, \ldots, 15\}$. The variables v_j and v'_j represent nibbles of the correct and faulty cipher state and the variables p_j represent nibbles of the current key.

The Analyse procedure takes v_j and v'_j as input, uses *fault equations* to analyse them and finally generates a set of key candidates K_i for the current subkey k_i. The fault equations are obviously the central building block of Analyse. Following the techniques of [9], we construct *generalised fault equations* $D_s(l)$, with $s, l \in \{0, \ldots, 15\}$, as follows: First we fix a fault location $l \in \{0, \ldots, 15\}$, i.e. one of the 16 state nibbles, in round r_i. Then we track the fault propagation over two rounds as illustrated in Figure 1.

Finally, beginning from the end of round $r_i + 2$ and using v_j and v'_j as input, we invert the LED-128 encryption steps up to the point before the SBox is applied for the last time in round $r_i + 1$. The generalised fault equations $D_s(l)$ can be written as illustrated in Figure 2.

$$
\begin{aligned}
w &= (d_0)^{-1} \cdot g_0((v_{i_0}), (v'_{i_0})) & y &= (d_2)^{-1} \cdot g_0((v_{i_2}), (v'_{i_2})) \\
w &= (d_4)^{-1} \cdot g_1((v_{i_3}), (v'_{i_3})) & y &= (d_6)^{-1} \cdot g_1((v_{i_1}), (v'_{i_1})) \\
w &= (d_8)^{-1} \cdot g_2((v_{i_2}), (v'_{i_2})) & y &= (d_{10})^{-1} \cdot g_2((v_{i_0}), (v'_{i_0})) \\
w &= (d_{12})^{-1} \cdot g_3((v_{i_1}), (v'_{i_1})) & y &= (d_{14})^{-1} \cdot g_3((v_{i_3}), (v'_{i_3})) \\
x &= (d_3)^{-1} \cdot g_0((v_{i_3}), (v'_{i_3})) & z &= (d_1)^{-1} \cdot g_0((v_{i_1}), (v'_{i_1})) \\
x &= (d_7)^{-1} \cdot g_1((v_{i_2}), (v'_{i_2})) & z &= (d_5)^{-1} \cdot g_1((v_{i_0}), (v'_{i_0})) \\
x &= (d_{11})^{-1} \cdot g_2((v_{i_1}), (v'_{i_1})) & z &= (d_9)^{-1} \cdot g_2((v_{i_3}), (v'_{i_3})) \\
x &= (d_{15})^{-1} \cdot g_3((v_{i_0}), (v'_{i_0})) & z &= (d_{13})^{-1} \cdot g_3((v_{i_2}), (v'_{i_2}))
\end{aligned}
$$

$$
i_0 \in \{0, 4, 8, 12\}, i_1 \in \{1, 5, 9, 13\}, i_2 \in \{2, 6, 10, 12\}, i_3 \in \{3, 7, 11, 15\}
$$

$$
(d_0, \ldots, d_{15}) = \begin{cases} (4, 2, 2, 1, 8, 6, 5, 6, \mathtt{B}, 9, \mathtt{A}, \mathtt{E}, 2, \mathtt{B}, \mathtt{F}, 2), & \text{if } l \in \{0, 5, 10, 15\} \\ (1, 4, 2, 2, 6, 8, 6, 5, \mathtt{E}, \mathtt{B}, 9, \mathtt{A}, 2, 2, \mathtt{B}, \mathtt{F}), & \text{if } l \in \{1, 6, 11, 12\} \\ (2, 1, 4, 2, 5, 6, 8, 6, \mathtt{A}, \mathtt{E}, \mathtt{B}, 9, \mathtt{F}, 2, 2, \mathtt{B}), & \text{if } l \in \{2, 7, 8, 13\} \\ (2, 2, 1, 4, 6, 5, 6, 8, 9, \mathtt{A}, \mathtt{E}, \mathtt{B}, \mathtt{B}, \mathtt{F}, 2, 2), & \text{if } l \in \{3, 4, 9, 14\} \end{cases}
$$

Figure 2. The generalised fault equations for LED.

The functions $g_0, \ldots, g_3$ correspond to the right-hand sides of the fault equations as given in [9] and are illustrated in the Appendix A. Note that the dependency of $D_s(l)$ on l is introduced by the values d_s as their inverses over $\mathbb{F}_{16}$ are multiplied to the right-hand sides of the $D_s(l)$.

Additionally we can define the so-called *generalised key set fault equations* $E_0(l), \ldots, E_3(l)$, see Appendix A, by starting from four unknown state elements $u_0, \ldots, u_3 \in \mathbb{F}_{16}$ in round $r_i + 1$ and inverting the encryption steps of one more round. This can be derived again from Figure 1. Those equations share the variables w, x, y and z with $D_s(l)$ and define conditions on complete sets of key candidates, and thus improve the filtering procedure.

By evaluating all equations D_s for all possible key nibbles of the subkey k_i and discarding nibbles that do not fulfil the equations we effectively shrink the keyspace K_i. All key nibbles that pass the first filtering step generate valid assignments of the variables w, x, y and z, which are then double-checked by the equations $E_0(l), \ldots, E_3(l)$. If there is no solution for at least one of the assignments of $u_0, \ldots, u_3$ then the values assigned to w, x, y and z were incorrect and the entire candidate set computed through the equations D_s can be discarded. More details on the complete filtering process can be found in [9]. Note that coefficients for the equations D_s and E_t do not depend on the same fault locations l. This influences the time complexity when comparing the two fault models RKF and RUF with each other.

Finally, to run the Multi-Stage Fault Attack on LED-128, we first produce faulty ciphertexts c'_0 in stage 0, by injecting faults in round $r_0 = 46$. Then we apply Analyse to the pair of ciphertexts c and c'_0 which generates a set of key candidates K_0. Recall, as described in Section II, that multiple fault injections and thus multiple calls to Analyse might be necessary until the set K_0 is smaller than the threshold τ_0. In stage 1 we inject faults in round $r_1 = 42$ to get faulty ciphertexts c'_1. Then, for each $x \in K_0$, we partially decrypt c and c'_1 by 4 rounds and apply Analyse to each of those pairs which results in a key candidate set $K_1(x)$. The union of the sets $K_1(x)$ forms the candidate set K_1 of the subkey k_1. As in stage 0, multiple fault injections might be necessary to shrink K_1 until its size is smaller than τ_1. The final step of the attack determines the correct key by a brute-force search on $K_0 \times K_1$.

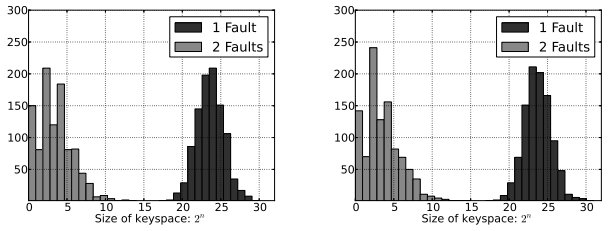

Figure 3. Num. of candidates in stage 0 / 1 (left / right) for LED-128.

1) Complexity analysis of the attack.: Executing one run of the Analyse method under the RKF model requires 2^{20} evaluations of single fault equations, as each one of the 16 equations D_s depends on four key nibbles. Compared

Table I
STATISTICS FOR THE NUMBER OF CANDIDATES OF $k_i, i \in \{0,1\}$.

	after stage 0		after stage 1	
# fault injections	1	2	1	2
min	$2^{17.00}$	1	$2^{17.00}$	1
max	$2^{30.00}$	$2^{14.00}$	$2^{31.00}$	$2^{15.00}$
avg	$2^{23.64}$	$2^{3.26}$	$2^{23.71}$	$2^{3.32}$
median	$2^{24.50}$	$2^{8.00}$	$2^{25.00}$	$2^{8.50}$

to that, the evaluation of the equations E_t has negligible time complexity and hence is not considered further. The complexity for the RUF model is even higher with 2^{24} evaluations. Under that model, the location of the fault injection is unknown and an attacker has to try all 16 possible combinations of the values of the variables d_s and e_t. This results in a complexity of about $T \cdot 2^{20}$ (for RKF) respectively $T \cdot 2^{24}$ (for RUF) for stage 1 where T is the number of key candidates for k_0.

C. Experimental Results

The results of the Multi-Stage Fault Attack on LED-128 were obtained from 10,000 runs of the attack using the RUF model. Figure 3 shows the numbers of remaining key candidates after one and two fault injections for stage $i \in \{0,1\}$. The number of times a particular count of key candidates appeared during our experiments is displayed on the Y-axis. Table I shows the numbers of key candidates if 1 and 2 fault injections are performed in stage 0 and 1 of the attack. If the attacker injects just a single fault in stage 0 she has to execute, on average, $2^{24} \cdot 2^{23.64} = 2^{47.64}$ evaluations of single fault equations in stage 1. Even in a rare best case where stage 0 yields 2^{17} key candidates, the complete attack requires 2^{41} evaluations, which is still beyond feasibility. But as soon as we inject a second fault in stage 0, the number of key candidates for k_0 drops very rapidly to $2^{3.26}$. Thus, three fault injections are required to break LED-128 on average: two in stage 0 and one in stage 1.

IV. APPLICATION TO PRINCE

As in the previous section we first describe the block cipher PRINCE, as specified in [5], and then show how to cryptanalyse it using the Multi-Stage Fault Attack algorithm from Section II. For the rest of the paper, let $\gg$ and $\ggg$ denote a non-cyclic resp. cyclic shift of a bitstring to the right, and let $\oplus$ denote addition using bitwise XOR.

A. Specification of PRINCE

PRINCE is a 64-bit block cipher with a 128-bit key. Before an encryption (or decryption) is executed, a 64-bit subkey k_2 is derived from the user supplied 128-bit key $k_0 \parallel k_1$ via $k_2 = (k_0 \ggg 1) \oplus (k_0 \gg 63)$. The subkeys k_0 and k_2 are used for input- and output-whitening. The core of PRINCE is a 12-round block cipher which solely uses k_1 as subkey. Figure 4 gives an overview of the cipher.

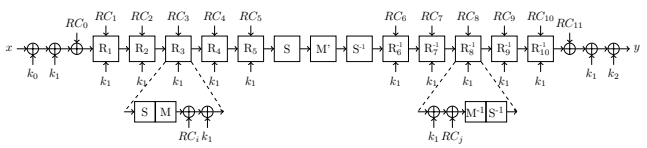

Figure 4. Layout of PRINCE.

Each round R_i and R_j^{-1} with $i \in \{1,\ldots,5\}$ and $j \in \{6,\ldots,10\}$ consists of a key addition, an S-layer, a linear layer which multiplies the state (represented by a 64 bit row vector) by a matrix, and the addition of a round constant (see Figure 4). The particular operations are described as follows.

1) S-Layer.: PRINCE uses the following 4-bit SBox:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	B	F	3	2	A	C	9	1	6	7	8	0	E	5	D	4

2) M-/M'-Layer.: The 64-bit state is multiplied by a 64×64 matrix M or M'. We refer to the original specification [5] for the exact definitions of M and M'. It is important to observe that M' is an involutive matrix and $M = SR \circ M'$, where SR is the ShiftRows operation as used in AES. Note that the multiplication by M is not an involution anymore, as ShiftRows is non-involutive.

3) RC_i-add.: This operation adds the round constant RC_i to the state using bitwise XOR. The values of RC_i are defined in the table below.

i	RC_i
0 – 2	0000000000000000, 13198a2e03707344, a4093822299f31d0,
3 – 5	082efa98ec4e6c89, 452821e638d01377, be5466cf34e90c6c,
6 – 8	7ef84f78fd955cb1, 85840851f1ac43aa, c882d32f25323c54,
9 – 11	64a51195e0e3610d, d3b5a399ca0c2399, c0ac29b7c97c50dd

Those constants have a special property: For all $i \in \{1,\ldots,10\}$ the equality $RC_i \oplus RC_{11-i} =$ `c0ac29b7c97c50dd` $(= \alpha)$ holds. Together with the fact that M' is involutive, this makes it possible to perform encryption and decryption using basically the same circuit (or implementation) and is referred to as the *α-reflection property*. For details, see the specification of PRINCE [5].

B. A Multi-Stage Fault Attack on PRINCE

The fault attack on PRINCE requires two stages, as k_0 can be easily derived as soon as k_2 is known. As before we now discuss the functionality of the Analyse method. To produce faulty ciphertexts for stage 0, we inject faults between the application of the S-Layer in round R_8^{-1} and the multiplication with the matrix M' in round R_9^{-1}. For stage 1, we inject faults exactly one round earlier. Figure 5 shows the fault propagation of a fault in PRINCE over two R^{-1} rounds.

In order to construct the fault equations used for key candidate filtering, we start, as in the case of LED, with the correct and faulty ciphertexts $c = c_0 \parallel \cdots \parallel c_{15}$ and $c' = c'_0 \parallel \cdots \parallel c'_{15}$ (or with the respective intermediate states in the second stage) and work backward through the encryption, inverting each of the steps, up to the point before the application of the last SBox. This corresponds to the last "matrix" in Figure 5. Let us start by fixing notation.

Let $i \in \{0,\ldots,15\}$, and let v_i and v'_i be variables representing the i-th nibble of the correct and the faulty ciphertext (or the i-th correct and faulty state nibble in the case of the second stage of the attack) respectively. Moreover, the variables p_i represent key nibbles and q_i round constants. In stage 0 we substitute the nibbles of RC_{11} for q_i and in stage 1 the nibbles of RC_{10} for q_i.

Let us point out a particular feature of the attack on PRINCE: an adversary cannot reconstruct one of the secret keys directly during stage 0 of the attack. The keys k_1 and k_2 are applied immediately in succession during one run of the encryption, and thus we can only reconstruct the XOR value $k_1 \oplus k_2$. This fact presents no drawback for the feasibility of the attack. In stage 1 we will directly reconstruct candidates for k_1.

Let $a = b_0 \parallel b_1 \parallel b_2 \parallel b_3$ be a 4-bit value and $j \in \{0,\ldots,3\}$. Then we define $\varphi_j(a)$ as the value equal to a

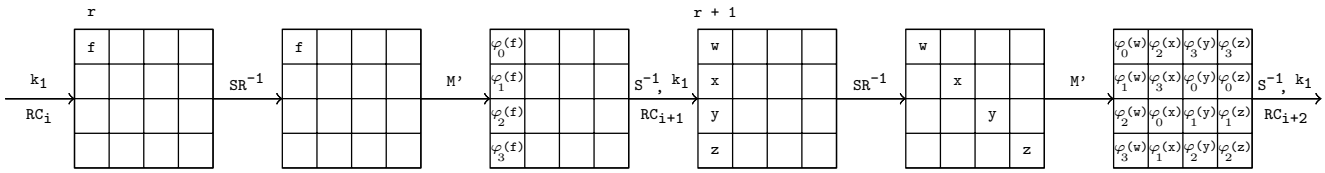

Figure 5. Fault propagation in PRINCE over two R^{-1} rounds.

except for the j-th bit b_j which is set to 0. So, for example, for $j = 2$, we get $\varphi_2(a) = b_0 \parallel b_1 \parallel 0 \parallel b_3$. Let w, x, y and z be variables and let $j_i \in \{0, \ldots, 3\}$ then we can describe the fault equations E_i of PRINCE in the following way:

$$S(v_i \oplus p_i \oplus q_i) \oplus S(v_i' \oplus p_i \oplus q_i) = \begin{cases} \varphi_{j_i}(w), & i \in \{0, \ldots, 3\} \\ \varphi_{j_i}(x), & i \in \{4, \ldots, 7\} \\ \varphi_{j_i}(y), & i \in \{8, \ldots, 11\} \\ \varphi_{j_i}(z), & i \in \{12, \ldots, 15\} \end{cases}$$

The values of the indices j_i from the variables on the right-hand sides of the equations E_i are derived from the multiplication of the state with the matrix M' (see Figure 5) and depend on the location l of the injected fault. They assume the following values:

$$(j_0, \ldots, j_{15}) = \begin{cases} (0,1,2,3,2,3,0,1,3,0,1,2,3,0,1,2), & \text{if } l \in \{0,7,10,13\} \\ (3,0,1,2,1,2,3,0,2,3,0,1,2,3,0,1), & \text{if } l \in \{1,4,11,14\} \\ (2,3,0,1,0,1,2,3,1,2,3,0,1,2,3,0), & \text{if } l \in \{2,5,8,15\} \\ (1,2,3,0,3,0,1,2,0,1,2,3,0,1,2,3), & \text{if } l \in \{3,6,8,12\} \end{cases} \quad (*)$$

A 4-bit value t is called *valid with respect to pattern* φ_{j_i} if the binary representation of t has a 0 at the bit position j_i. In the following we use bit pattern matching to construct inductively a set S_i which will ultimately contain candidates for the nibble p_i of the subkey. Key candidate filtering is done in three steps: *Evaluation*, *Inner Filtering* and *Outer Filtering*, which are described next. Note that the first two steps could be executed together, but for better comprehensibility we describe them separately.

1) Evaluation.: Each equation E_i is evaluated for all possible 4-bit values u of nibble candidates associated to the variable p_i. When the result of an evaluation $t = E_i(u)$ has been computed, the tuple (t, u) is appended to the set S_i.

2) Inner Filtering.: In this step we check for all tuples $(t, u) \in S_i$ if the entry t is valid with respect to the bit pattern φ_{j_i}. Those tuples that do not have a valid entry t are discarded, all others are kept.

For example, we take fault equation E_0 and assume that a fault was injected in nibble $l = 0$. From the definition of j_i above we see that the 0-th entry of j_0 is 0. Moreover, we assume that the tuple $(t, u) = (\texttt{0x7}, \texttt{0x3})$ is an element of S_0 and observe immediately that 0x7 matches the bit pattern $\varphi_{j_0} = 0 \parallel s_1 \parallel s_2 \parallel s_3$. Thus $(\texttt{0x7}, \texttt{0x3})$ is a valid tuple and 0x3 a potential candidate for the nibble associated to p_0.

3) Outer Filtering.: The idea in the final filtering step is to exploit the fact that the elements of the sets $S_{4 \cdot m}, \ldots, S_{4 \cdot m+3}$ are related to each other for a fixed $m \in \{0, \ldots, 3\}$. This is due to the fact that the right-hand sides of the equations $E_{4 \cdot m}, \ldots, E_{4 \cdot m+3}$ are derived from a common pre-image. This can be utilized to build conditions for filtering candidates of the nibbles associated to $p_{4 \cdot m}, \ldots, p_{4 \cdot m+3}$. First we fix $m \in \{0, \ldots, 3\}$ and order the tuples $(t_{4 \cdot m+n}, u_{4 \cdot m+n}) \in S_{4 \cdot m+n}$ lexicographically for all $n \in \{0, \ldots, 3\}$. Then we compute the sets $P_{4 \cdot m+n}$ containing the pre-images of all the values $t_{4 \cdot m+n}$. This is done as follows. After the Inner Filtering, all values $t_{4 \cdot m+n}$ match the bit pattern derived from $\varphi_{4 \cdot m+n}$. But we do not know if the $j_{4 \cdot m+n}$-th bit of $t_{4 \cdot m+n}$ had value 0 or 1 before it was fixed to 0. Hence we obviously have two possible values for the pre-images of $t_{4 \cdot m+n}$. One is $t_{4 \cdot m+n}$ itself, and the other has a 1 at bit position $j_{4 \cdot m+n}$. Then we intersect the pre-image sets $P_{4 \cdot m}, \ldots, P_{4 \cdot m+3}$ with each other and obtain a set G_m of pre-image candidates. After that we check for each $g_m \in G_m$ if, for every $n \in \{0, \ldots, 3\}$, there is at least one tuple in $S_{4 \cdot m+n}$ which has the value $\varphi_{i_{4 \cdot m+n}}(g_m)$ in its first component. If so, g_m is a valid pre-image. When all pre-images have been processed, all tuples are deleted from the sets $S_{4 \cdot m}, \ldots, S_{4 \cdot m+3}$, except those where the first entry has a valid pre-image g_m. This finishes the filtering stage.

Finally, after projecting the sets S_i to their second components u_i, the Cartesian product over those projections is computed to get the key candidates for $k_1 \oplus k_2$. If the number of candidates for $k_1 \oplus k_2$ is small enough, stage 0 of the attack ends. Otherwise the procedure above is repeated as described in the Multi-Stage Fault Attack algorithm in Section II.

In the second stage of the attack, candidates for k_1 are computed using the previously described configurations for the fault injections. This is repeated until the number of candidates for k_1 falls below the specified threshold value τ_1. As soon as this is the case, the candidates for k_1 and $k_1 \oplus k_2$ are used to derive candidates for the subkeys k_2 and k_0. Finally, a brute-force search on the Cartesian product $K_0 \times K_1$ is performed to find the actual key $k_0 \parallel k_1$.

4) Complexity analysis of the attack.: The complexity of the Analyse method in the case of PRINCE is very low. The computationally most expensive part is the evaluation of the fault equations. For each of the 16 equations we have to compute 16 evaluations, since there are 16 possible values for the key nibbles. Altogether this results in $2^8 = 256$ evaluations. As already mentioned in Section II, the number of evaluations is multiplied by a constant factor when using the RUF fault model. In the case of PRINCE, this factor has the value 4, as there are four different bit patterns for key candidate filtering and the attacker does know which pattern is the correct one, due to the unknown location of the fault injection. Therefore all four patterns j_i (see $(*)$) have to be tried, which gives $2^{10} = 1024$ evaluations for one run of Analyse. In stage 1 the Analyse method may be executed up to T times in the worst case, where T is the number of $k_1 \oplus k_2$ candidates. This results in a complexity of about $2^8 \cdot T$ or $2^{10} \cdot T$ evaluations, depending on the fault model which has been used.

C. Experimental Results

In this section we will describe the results of the Multi-Stage Fault Attack on PRINCE. All results were obtained from 10,000 runs of the attack. For the computation of the results, the weaker RUF fault model was used. We observed that the differences in the sizes of the candidate

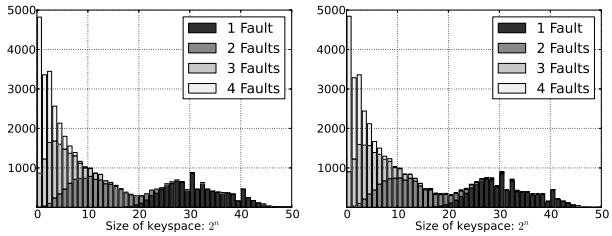

Figure 6. Num. of candidates in stage 0 / 1 (left / right) for PRINCE.

sets between the fault models RKF and RUF were effectively non-existent. This can be explained by the observation that in almost all cases the candidate sets are empty when a wrong bit pattern $(j_i)_{i=0,\ldots,15}$ is used for filtering. Figure 6 gives an overview (with stacked bars) on the two stages of the attack for multiple fault injections. Table II summarizes the results of the attack on PRINCE.

Table II
STATISTICS FOR THE NUMBER OF CANDIDATES OF $k_1 \oplus k_2$ AND k_1.

	after stage 0				after stage 1			
# keys / # faults	1	2	3	4	1	2	3	4
min	$2^{17.00}$	1	1	1	$2^{16.00}$	1	1	1
max	$2^{50.00}$	$2^{38.00}$	$2^{24.00}$	$2^{12.00}$	$2^{49.00}$	$2^{44.00}$	$2^{40.00}$	$2^{43.00}$
avg	$2^{30.89}$	$2^{11.44}$	$2^{4.12}$	$2^{1.47}$	$2^{30.41}$	$2^{11.64}$	$2^{4.44}$	$2^{1.82}$
median	$2^{34.50}$	$2^{19.50}$	$2^{12.50}$	$2^{7.00}$	$2^{33.50}$	$2^{21.50}$	$2^{21.00}$	$2^{21.00}$

For stage 0, we get on average $2^{30.89}$ candidates when injecting a single fault. This results in an overall complexity of $2^{40.89}$ evaluations of single fault equations for stage 1 under the RUF model. This is not feasible on common hardware. But with a second fault injection in stage 0, the complexity drops to $2^{11.44}$, which is easily doable. Thus, on average, we need about two fault injections in stage 0 to be able to finish stage 1. Furthermore as expected the numbers for stage 0 do not differ significantly from those of stage 1, except for the median, which surprisingly stays rather constant after the first fault injection. But as we only have to search through the generated key candidate sets, the average of $2^{30.41}$ is feasible for brute-force. Nevertheless, note that the maximal sizes of the key candidate sets are still quite high. Thus there might be cases where more than one fault injection is required for stage 1. In summary, our experiments show that on average 3 to 4 faults are required to reconstruct the complete 128-bit key of PRINCE with a Multi-Stage Fault Attack on common hardware.

V. CONCLUSION

State-of-the-art ciphers increasingly employ keys that consist of independent subkeys. We introduced the generic concept of Multi-Stage Fault Attacks which target individual subkeys by multiple fault injections. One stage consists of several fault injections followed by mathematical analysis that yields a set of candidates for a subkey. We presented an algorithm that balances the number of fault injections allocated to different stages, in order to keep the sizes of final candidate sets sufficiently small for brute-force search. The generic algorithm estimates the expected effort for each stage and decides the number of fault injections to be performed based on user-specified threshold variables while taking interaction between subkeys into account. We illustrated the successful application of the general principle on two recently introduced ciphers which we were able to break with 3 to 4 fault injections on average.

REFERENCES

[1] E. Biham and A. Shamir, Differential Cryptanalysis of DES-like Cryptosystems, In: *CRYPTO* 1990, LNCS, vol. **537**, Springer, Heidelberg 1990, pp. 2–21.

[2] E. Biham and A. Shamir, Differential Fault Analysis of Secret Key Cryptosystems, In: *CRYPTO* 1997, LNCS, vol. **1294**, Springer, Heidelberg 1997, pp. 513–525.

[3] A. Bogdanov et al., PRESENT: An Ultra-Lightweight Block Cipher, In: *CHES* 2007, LNCS, vol. **4727**, Springer, Heidelberg 2007, pp. 450–466.

[4] D. Boneh et al., On the Importance of Elimination Errors in Cryptographic Computations, J. Cryptology **14** (2001), pp. 101–119.

[5] J. Borghoff et al., PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications, In: *ASIACRYPT* 2012, LNCS, vol. **7658**, Springer Heidelberg 2012, pp. 208–225.

[6] G. Piret and J.-J. Quisquater, A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD, In: *CHES* 2003, LNCS, vol. **2779**, Springer, Heidelberg 2003, pp. 77–88.

[7] J. Guo, T. Peyrin, A. Poschmann and M. Robshaw, The LED Block Cipher, In: *CHES* 2011, LNCS, vol. **6917**, Springer, Heidelberg 2011, pp. 326–341.

[8] M. Hojsik and B. Rudolf, Differential Fault Analysis of Trivium, In: *FSE* 2008, LNCS, vol. **5086**, Springer, Heidelberg 2008, pp. 158–172.

[9] P. Jovanovic, M. Kreuzer and I. Polian, A Fault Attack on the LED Block Cipher, In: *COSADE* 2012, LNCS, vol. 7275, Springer, Heidelberg 2012, pp. 120–134.

[10] L. Song and L. Hu, Differential Fault Attack on the PRINCE Block Cipher, In *IACR Cryptology ePrint Archive*, Report 2013/043, 2013.

[11] M. Tunstall et al., Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault, In: *WISTP* 2011, LNCS, vol. **6633**, Springer, Heidelberg 2011, pp. 224–233.

APPENDIX

A. The functions $g_0, \ldots, g_3$ of the LED attack

Let c_j, c'_j and k_j be variables over $\mathbb{F}_{16}$, for $j \in \{0, \ldots, 15\}$, where c_j and c'_j represent nibbles of the correct and the faulty cipher state and k_j nibbles of the key. Furthermore let S^{-1} denote the inverse SBox of LED. Then we define $g_0, \ldots, g_3$ as:

$$
\begin{aligned}
g_0 &= S^{-1}(\mathtt{C}\cdot(c_0+k_0)+\mathtt{C}\cdot(c_4+k_4)+\mathtt{D}\cdot(c_8+k_8)+\mathtt{4}\cdot(c_{12}+c_{12}))+\\
&\quad S^{-1}(\mathtt{C}\cdot(c'_0+k_0)+\mathtt{C}\cdot(c'_4+k_4)+\mathtt{D}\cdot(c'_8+k_8)+\mathtt{4}\cdot(c'_{12}+k_{12}))\\
g_1 &= S^{-1}(\mathtt{3}\cdot(c_3+k_3)+\mathtt{8}\cdot(c_7+k_7)+\mathtt{4}\cdot(c_{11}+k_{11})+\mathtt{5}\cdot(c_{15}+k_{15}))+\\
&\quad S^{-1}(\mathtt{3}\cdot(c'_3+k_3)+\mathtt{8}\cdot(c'_7+k_7)+\mathtt{4}\cdot(c'_{11}+k_{11})+\mathtt{5}\cdot(c'_{15}+k_{15}))\\
g_2 &= S^{-1}(\mathtt{7}\cdot(c_2+k_2)+\mathtt{6}\cdot(c_6+k_6)+\mathtt{2}\cdot(c_{10}+k_{10})+\mathtt{E}\cdot(c_{14}+k_{14}))+\\
&\quad S^{-1}(\mathtt{7}\cdot(c'_2+k_2)+\mathtt{6}\cdot(c'_6+k_6)+\mathtt{2}\cdot(c'_{10}+k_{10})+\mathtt{E}\cdot(c'_{14}+k_{14}))\\
g_3 &= S^{-1}(\mathtt{D}\cdot(c_1+k_1)+\mathtt{9}\cdot(c_5+k_5)+\mathtt{9}\cdot(c_9+k_9)+\mathtt{D}\cdot(c_{13}+k_{13}))+\\
&\quad S^{-1}(\mathtt{D}\cdot(c'_1+k_1)+\mathtt{9}\cdot(c'_5+k_5)+\mathtt{9}\cdot(c'_9+k_9)+\mathtt{D}\cdot(c'_{13}+k_{13}))
\end{aligned}
$$

B. The generalised key set fault equations $E_0(l),\ldots,E_3(l)$ of the LED attack

Let $u_0, \ldots, u_3$, w, x, y and z denote variables over $\mathbb{F}_{16}$ and let S^{-1} be the inverse SBox of LED. The second set of LED fault equations is then defined as

$$
\begin{aligned}
f' &= (e_0)^{-1}\cdot(S^{-1}(u_0)+S^{-1}(u_0+w))\\
f' &= (e_1)^{-1}\cdot(S^{-1}(u_1)+S^{-1}(u_1+x))\\
f' &= (e_2)^{-1}\cdot(S^{-1}(u_2)+S^{-1}(u_2+y))\\
f' &= (e_3)^{-1}\cdot(S^{-1}(u_3)+S^{-1}(u_3+z))
\end{aligned}
$$

with values $e_0, \ldots, e_3$, which depend on the fault location l, as written below:

$$
(e_0,\ldots,e_3) = \begin{cases}
(\mathtt{4},\mathtt{1},\mathtt{2},\mathtt{2}), & \text{if } l \in \{0,1,2,3\}\\
(\mathtt{8},\mathtt{6},\mathtt{5},\mathtt{6}), & \text{if } l \in \{4,5,6,7\}\\
(\mathtt{B},\mathtt{E},\mathtt{A},\mathtt{9}), & \text{if } l \in \{8,9,10,11\}\\
(\mathtt{2},\mathtt{2},\mathtt{F},\mathtt{B}), & \text{if } l \in \{12,13,14,15\}
\end{cases}
$$