Protecting Block Ciphers against Differential Fault Attacks without Re-keying

– EXTENDED VERSION –

Anubhab Baksi[1], Shivam Bhasin[2], Jakub Breier[2], Mustafa Khairallah[3], Thomas Peyrin[1,2,3]
[1]*School of Computer Science and Engineering*
[2]*Temasek Laboratories*
[3]*School of Physical and Mathematical Sciences*
Nanyang Technological University, Singapore
anubhab001@e.ntu.edu.sg, sbhasin@ntu.edu.sg, jbreier@ntu.edu.sg,
mustafam001@e.ntu.edu.sg, thomas.peyrin@ntu.edu.sg

***Abstract*—In this article, we propose a new method to protect block cipher implementations against Differential Fault Attacks (DFA). Our strategy, so-called "Tweak-in-Plaintext", ensures that an uncontrolled value ('tweak-in') is inserted into some part of the block cipher plaintext, thus effectively rendering DFA much harder to perform. Our method is extremely simple yet presents many advantages when compared to previous solutions proposed at AFRICACRYPT 2010 or CARDIS 2015. Firstly, we do not need any Tweakable block cipher, nor any related-key security assumption (we do not perform any re-keying). Moreover, performance for lightweight applications is improved, and we do not need to send any extra data. Finally, our scheme can be directly used with standard block ciphers such as AES or PRESENT. Experimental results show that the throughput overheads, for incorporating our scheme into AES-128, range between $\approx 5\%$ to $\approx 26.9\%$ for software, and between $\approx 3.1\%$ to $\approx 25\%$ for hardware implementations; depending on the tweak-in size.**

I. Introduction

Physical attacks on cryptographic implementations are a relatively new paradigm of research for the last two decades. These attacks have achieved a huge popularity, mainly due to their effectiveness, wide applicability and possibility of exploiting real life vulnerabilities. Unlike classical attacks, which rely on finding mathematical/statistical weaknesses of a cipher, these attacks target a device performing a cryptographic operation.

One type of physical attack, referred to as the Fault Attack or Fault Analysis (FA) assumes more power for the attacker. Here, Eve can tamper with the device using methods; such as voltage glitch, LASER beam etc. This can effectively lead the device to malfunction, thereby disturbing the execution of the cipher. The resulting faulty output is later analyzed, which often reveals information regarding the secret key of the cipher.

This is an extended version of the paper with same title accepted in IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2018. This work is partly supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

Differential Fault Attack/Analysis (DFA) is usually used against symmetric key ciphers. It was introduced by Biham and Shamir at CRYPTO'97 [5], and the basic idea is depicted in Figure 1. The attacker normally injects a disturbance (fault) near the end of a cipher execution. Then, she analyzes the XOR difference between a non-faulty and the corresponding faulty output, which are produced by keeping all cipher inputs fixed. Depending on the context, she may need a few such pairs, so she injects multiple faults while keeping the inputs fixed. The faults are usually composed of flipping one/a few particular bits in the cipher. Because of its immense popularity among the research community, one can find works exploiting DFA aimed at all major ciphers.

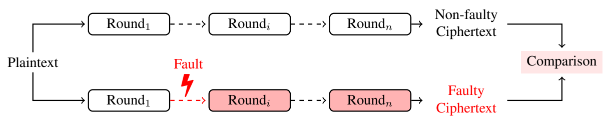


Fig. 1: Schematic view of Differential Fault Analysis

With the rapid advancement of fault injection techniques and methodologies, fault attacks are considered a serious threat to sensitive applications. This threat has motivated a chain of research for countermeasures. However, there is no DFA resistant cipher in existence so far; which means that the security of the cryptosystem remains dependent on the physical implementation of the cipher. The DFA countermeasures are generally classified into two categories: hardware-based and software-based. Hardware-based countermeasures can be either detection (e.g., sensors to detect any physical stress which could potentially result in fault injection [16], [18], [40]) or prevention (e.g., usage of a shield that blocks outside disturbance [2]). Software-based countermeasures can be classified into detection, infection, prevention and re-keying methods; which are described next.

a) Detection

Fault detection aims at detecting anomalies and then launching a defined recovery procedure. Initially, detection countermeasures came from communication theory; based on concurrent error detection, such as linear parity [24], non-linear $[n, k]$ codes [27], etc. Such codes exploit information redundancy and are easy to implement, requiring limited overhead. However, the protection is limited to very specific fault models and can be broken by other easily reproducible fault models [17]. Other detection countermeasures are based on operation level redundancy [23]. A typical example of such protection is duplicated computation, followed by a comparison. The duplication can be in either space or time and thus results in at least 100% overhead in either area or performance.

b) Infection

A typical problem with detection-based countermeasure is that the output comparison part can be subjected to a fault attack as well. In this case, regardless of whether they match or not, the attacker can bypass the comparison step, forcing the device to produce the faulty output. To overcome this situation, infective countermeasures have been proposed in [39] for public key cryptosystems. Several works follow this paradigm in both public and private key systems, e.g., [13], [14], [26], [36], [38]. The main principle of the fault infection involves diffusion of faulty value in the cipher state to prevent fault analysis. Use of randomization has also been proposed to boost the security of fault detection countermeasures [30]. In summary, this method takes the difference between two computations (Δ) of the same cipher. This difference is passed to another function, τ, such that $\tau(0) = 0$ and $\tau(\Delta)$ is a random value if $\Delta \neq 0$. The output from τ is then used to mask one output. Thus, if the outputs are identical, it is communicated. If not, the faulty output is XORed by a random mask, so that the attacker cannot get any information for the fault injection. . However, recent studies [1], [3], [37] show that this approach is not mature yet.

c) Prevention

Preventive countermeasures require strict design modifications to resist injected faults. They can be used to prevent instruction skip attacks in microcontrollers. They rely on inserting idempotent instructions, so that skipping one instruction does not alter the normal flow of operation [32].

d) Re-keying

In this protocol level approach, it is assumed that a portion of the device can be protected, whereas the rest of it is still vulnerable to fault attack. This assumption is practical, if the protected portion takes up a relatively small area. This concept was first presented by Medwed *et al.* at AFRICACRYPT'10 [31]. The authors construct a function, g (called, *re-keying* function), which takes the (fixed) master key K and a random nonce (for every session) as input and outputs a fresh session key K_i^* for each session i. This session key is later used in a block cipher E with plaintext $\mathsf{P_i}$ as: $\mathsf{C_i} = \mathsf{E}_{\mathsf{K}_i^*}(\mathsf{P_i})$. This g does not take any additional material, is stateless and has a low hardware footprint. The authors claim that even if g is not cryptographically secure, this scheme is capable of thwarting DFA and side-channel attacks.

However, this scheme has one important issue – it can only provide birthday-bound security, being prone to collision-based key recovery attack due to the frequent re-keying. This was pointed out by Dobraunig *et al.* at CARDIS'14 [10] (discussed in Section II-A). As an improvement, they later proposed to replace the re-keying function with a tweakable block cipher, with a block counter acting as a tweakable input [11] at CARDIS'15. This work extends the security of fresh re-keying scheme beyond birthday-bound security against side-channel attacks. In another work, it is independently demonstrated that a tweakable block cipher with a random (unknown) tweak can prevent side-channel and fault attacks [33]. However, as we point out in Section II-B tweakable block cipher based techniques have a few drawbacks.

Our Contributions

We present a new scheme that provides a safeguard against DFA for any block cipher (including already proposed ones) at the expense of under utilization of bandwidth. We show that for linear reduction of bandwidth, we gain a (sub-)exponential safeguard against DFA. As opposed to tweakable block ciphers, no synchronization is required between the sender (Alice) and the recipient (Bob). Also, this construction can be used for error detection at the recipient's (Bob) side. Finally, it does not incur much overhead to existing platforms (software and hardware), and is suitable for lightweight applications. Our scheme does not suffer from birthday bound on key size, if the tweak-in generation functions proposed in this paper, m-sequences, are used. To utilize the birthday bound on key size, one (with high probability) has to repeat encryption/computation multiplied by a sub-(exponential) factor on tweak-in size.

The rest of the document is organized as follows. Section II gives the overall context of our scheme. Then, Section III formally describes the scheme; followed by Section IV where the we analyze it in details. Finally, Section V concludes the paper.

II. Background

A. Birthday Bound for Fresh Re-keying Schemes

In this part, we briefly describe the birthday bound for fresh re-keying scheme [31] which is reported in CARDIS'14 [10].

The attack is done in two phases. In the first (off-line) phase, Eve builds a table with the same plaintext P, and stores the pairs

$$(\mathsf{C_i} = \mathsf{E}_{\mathsf{K}_i^*}(\mathsf{P}), \mathsf{K}_i^*)$$

in a list L, where K_i^* was chosen by her. Once enough (ciphertext, plaintext) pairs have been stored; she moves to the next (on-line) phase. In this phase, she queries multiple encryptions of the same plaintext P. As the session key K_i^* changes at every invocation, the response C_i' from the block cipher will be different each time with a high probability.

Now, if she can find one C'_i matching one C_i in L; then she recovers the corresponding session key K^*_i. Because of the birthday paradox, one can expect to find such a collision with a significant probability if L contains $2^{k/2}$ entries with the number of on-line queries are $2^{k/2}$; (the attack complexity being $2 \cdot 2^{k/2}$) where $k =$ size of K^*_i.

This attack is generic, and will anyway work, disregarding the choice of the re-keying function g. If g is not judiciously chosen, it may further be possible to find out the master key from K^*_i with a high probability (which was the case for [31]).

B. Tweakable Block Cipher Based Solution

The CARDIS'15 paper by Dobraunig *et al.* [11] attempts to achieve beyond birthday bound security by proposing a tweakable block cipher-based solution. The security claim by this work depends on the existence of an ideal tweakable block cipher.

There is a recent trend in developing tweakable block cipher in the community (such as the XTS construction [35] or the TWEAKEY framework [21]). The concept of tweakable block cipher is not so old [29] and therefore, there is no tweakable block cipher standard at the moment, which brings up the question of which cipher to choose to simulate an ideal primitive. Moreover, a tweakable block cipher is in general less efficient than a classical block cipher (XTS mode involves multiple encryptions; DEOXYS [22] ad-hoc tweakable block cipher follows the TWEAKEY framework and requires more rounds than its closest cipher AES).

Finally, a tweakable block cipher will likely require more resources than a classical block cipher due to the extra tweak input. Thus, a tweakable block cipher might not be a perfect choice for tightly constrained devices like Wireless Sensor Networks (WSNs) which are indeed vulnerable to side-channel and fault attacks. Therefore, if possible, it would be more interesting to have a protection from a classical and well studied block cipher only. In other words, we would like to have a protection that works for worldwide standards such as AES or PRESENT.

C. Our Approach: Tweaking the Plaintext

The work from Dobraunig *et al.* [11] only deals with resistance against side-channel attacks, although they speculate that their method can "probably be used to rule out" DFA. Our scheme can be thought of as a modification for any conventional block cipher as an external add-on that can provide a safeguard against existing DFA attacks. This scheme is easily portable to almost any existing hardware and software with minimal alteration. Although it could be interesting; we keep implication of our method with respect to side-channel analysis out of scope for this work.

Our technique, which we will refer to as the *Tweak-in-Plaintext* scheme from now on, incorporates ideas from detection (Section I). A basic requirement of DFA is to obtain faulty and non-faulty ciphertext, corresponding to a fixed plaintext. The plaintext is not required to be known or chosen, but it should be kept unchanged for two encryption calls. Tweakable block cipher based countermeasures, so far, thwart this by modifying the key with a different tweak, each time. Instead of doing that, we use the idea of tweaking the plaintext (by using a 'tweak-in'). We use the term 'tweak-in' instead of the common term, 'tweak', so as to distinguish the use cases.

All the hardware and software detection countermeasures rely on redundancy in one of the following parameters: area (e.g., circuit is duplicated), power (sensors are used to detect presence of an active fault injecting device), throughput (one circuit is used twice). In contrast to all these existing detection techniques, we use part of the bandwidth as a protection against DFA, which was not studied before. We show that, with a linear decrease in bandwidth utilization, our scheme offers safeguard of (sub-)exponential order against DFA, depending on the tweak-in generation mechanism.

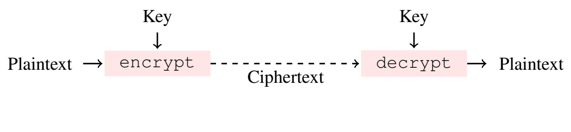
The security of our scheme against DFA depends on two assumptions:

(i) the underlying block cipher is an ideal block cipher,
(ii) the attacker is unable to control the tweak-in value, even by resetting the device.

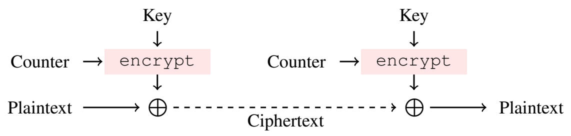
The tweak generation circuit, $\mathcal{T}$, has to be small, as it was the case for g in the re-keying scheme [31].

D. Block Cipher Modes

The employment strategy as well as the effectiveness of our scheme depends on the corresponding block cipher mode of operation. From our perspective, the block cipher modes of operations can be divided into two classes, depending on how the plaintext is processed by the sender (Alice) and recipient (Bob). We conveniently call them non-symmetric modes and symmetric modes respectively. The basic concepts are shown in Figure 2: Figure 2(A) for non-symmetric modes and Figure 2(B) for symmetric modes.



(A) Operation in Non-symmetric modes



(B) Operation in Symmetric modes

FIG. 2: Two types of block cipher modes

1) Non-symmetric Modes

For the modes under this class, the plaintext is used as an input to the actual block cipher; e.g., Electronic Codebook (ECB), Cipher Block Chaining (CBC). Alice and Bob use different functions: Alice uses the encrypt function for the plaintext, and Bob uses decrypt function for the ciphertext. Additionally, PMAC-like modes [8] also fall under this type, although the same function is used at both ends.

2) Symmetric Modes

The block cipher modes where plaintext-ciphertext are used like stream cipher belong to this class, where both Alice and Bob use same function (both use the block cipher `encrypt` function). Here the plaintext is XORed with the output of the block cipher `encrypt` function. The actual input to the block cipher is either a predetermined value or a value that depends on plaintext history (previous blocks). Examples include Counter (`CTR`), Output Feedback (`OFB`) and Cipher Feedback (`CFB`) modes. As pointed out in [28, Section 7.1], these modes posses inherent properties that can thwart DFA.

The scheme proposed in our paper is particularly useful when used with non-symmetric modes, as described in Section IV-A — IV-E. For symmetric modes, we discuss the similarities and differences between them and the proposed scheme in Section IV-F.

III. Description of Tweak-in-Plaintext Scheme

As already mentioned in Section II-C, we use the concept of tweaking the plaintext. The security claim against DFA here depends on the assumption that a part of the plaintext (which will be encrypted with a block cipher) is not controllable by the attacker. Thus a tweak-in, together with the actual plaintext (to be communicated) is used as the 'plaintext' input for the underlying block cipher. The k-bit key K used in the block cipher is not affected by the tweak-in. The basic work-flow is shown in Figure 3. Next, we formally describe the terms.

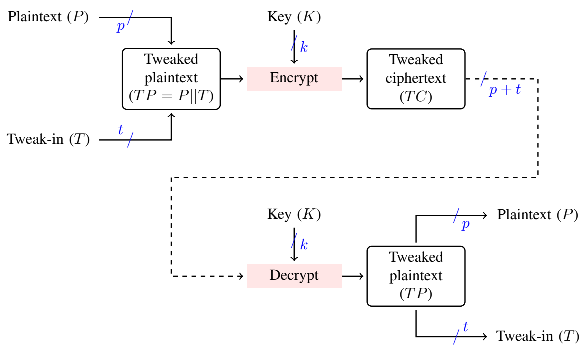


Fig. 3: Work-flow for Tweak-in-Plaintext scheme

- **Tweak-in.** A t-bit tweak-in T is generated from a circuit $\mathcal{T}$ independently to actual plaintext to be communicated. The attacker cannot control $\mathcal{T}$.
  The tweak-in value will change from one invocation of the encryption function to another. This will guarantee that the inputs to the block cipher are (very likely) different for two consecutive invocations, thus making DFA impossible to mount.
  The size t of the tweak-in is public, it can be fixed depending on the desired level of security against DFA. Note that choosing a very large value for t would be impractical as it would result in an inefficient communication bandwidth.
- **Plaintext.** The p-bit 'plaintext', denoted by P, is the actual message to be communicated (that Alice wants to send to Bob).
- **Tweaked plaintext.** Our scheme requires the plaintext to be concatenated with the tweak-in. This concatenated plaintext is referred to as tweaked plaintext and is denoted by TP $(= P||T)$. The size of TP $(= t + p)$ is equal to that of the plaintext input size for the underlying block cipher. The tweaked plaintext is then used as the plaintext input for the underlying block cipher, and encrypted with a secret key. For example, if a 16-bit tweak-in is chosen for `AES-128`; then $t = 16, |TP| = t + p = 128$.
- **Tweaked ciphertext.** The corresponding ciphertext obtained from the underlying block cipher after encrypting TP is termed as the tweaked ciphertext and is denoted by TC $(|TC| = t + p)$. Tweaked ciphertext is then communicated to Bob. Upon receiving TC, Bob decrypts it; and learns both the 'plaintext' P and the tweak-in T.

One similar, but different concept is proposed in [15, Algorithm 1], which masks the 'plaintext' with a random number r, encrypts it, and communicates r along with the ciphertext to Bob. So, this scheme keeps the plaintext bandwidth intact ($1\times$ utilization) but reduces communication bandwidth ($\frac{1}{2}\times$ utilization); in contrast, our scheme typically utilizes $\frac{31}{32}\times$ to $\frac{3}{4}\times$ of plaintext bandwidth (see Section IV-C) and $1\times$ communication bandwidth.

A. Advantages

Our scheme comes with many advantages when compared to existing DFA countermeasures:

1) No re-keying is needed

Our proposed countermeasure does not depend on the key. So, the re-keying concept is no longer required. Based on that feature:

a) No need for related-key security of the internal cipher

The internal block cipher is used as is, without re-keying. Thus, we do not need the internal cipher to resist attacks in much stronger security models such as the related-key model. This is particularly important as many ciphers have shown weaknesses in the related-key model (for example, `AES-192` and `AES-256` are known to have weakness [6], [7]).

b) No need to recompute the key schedule every time

Our scheme offers faster performance compared to re-keying based solution. With every re-keying, the key schedule must be recomputed, which can have non-negligible cost. For example, in the case of `AES`, the cost of the key schedule represents about 30% of the total encryption cost [4] in software.

c) Allows implementations with wired-in key

Our scheme works fine with a fixed key and thus can be directly used with a wired-in key. In other schemes [11], [31], [33], a part of the key has to be changed in every encryption, and thus a wired-in key can not be used in this situation.

d) No need for a tweakable block cipher

As already explained (Section II-B), tweakable block cipher might not be the perfect choice for devices. In practice, these

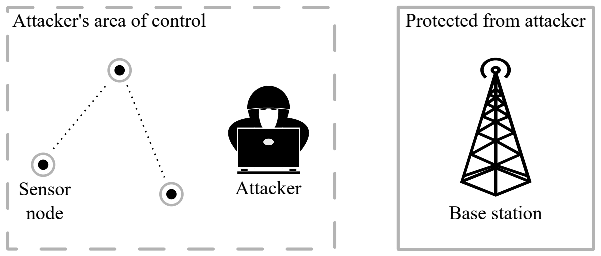


FIG. 4: Typical practical use case for Tweak-in-Plaintext scheme

devices actually stand vulnerable to hardware implementation attacks. In contrast, our method is applicable to any block cipher; thereby giving a choice of using a common block cipher which is well studied and standardized (like AES).

2) No need to send any extra data/ No synchronization is required

The tweak-in is sent to the decryption party as an inherent part of the ciphertext. As a result, we do not need to send any other data. This is in contrast with tweak based designs, where the tweak is to be communicated separately/synchronized.

3) Almost readily portable in existing devices

Unlike other schemes, this proposal needs minimal alterations to an existing architecture from a cipher designer's point of view. So, as a design, it can be ported in almost all devices without much difficulty. However, as the bandwidth utilization is reduced, some modification at the protocol level will likely be required.

4) Suitable for lightweight applications

Implementation of our scheme can be done with very small overhead. Therefore, it is well suitable in lightweight applications.

5) Can be (optionally) used for error detection

It is not needed for the tweak-in to be synchronized for our scheme to work. However, the tweak-in can be used for error detection at the recipient's side, by monitoring its values and ensuring they are generated correctly. Consider a situation, where the recipient (Bob) gets a tweak-in T from the sender (Alice). Now, upon receiving the ciphertext from Alice, Bob decrypts and finds out the tweak-in T^* embedded within the plaintext. If $T \neq T^*$, then Bob detects an error in the communication, which could be caused by an attacker. This is further elaborated in Section IV-D.

B. Limitations

There are, however, a few limitations with our scheme. These are presented below.

1) Only encryption can be protected

Our scheme can only protect the encryption circuit. Once the ciphertext is received by the recipient, it is not within the purview of our scheme to ascertain any protection. In other words, we assume that the sender (Alice) only is susceptible to fault attack, whereas the attacker (Eve) has no access to recipient's (Bob) end.

At a first glance this might seem to be a major drawback for our scheme, but as we explain here, this configuration seems to be the case in most real life scenarios. Let's consider, for example, the case of wireless sensor networks: a bunch of sensors are deployed in remote areas (which she is able to get hold of) from where they are communicating with the base station. However, the base station is located in a secure premises, which is off limit to the attacker. This means, she is not able to perform any attack at Bob's end. Refer to Figure 4 for a depiction of this scenario.

It is to be noted that our scheme can still be used with repeaters placed in between the sensor and the base station. Most of the repeaters are only responsible for restoring the digital signal from its physical modulated signal and do not perform decryption, as the routing addresses are not encrypted. However, even if the decryption-re-encryption function is performed, to have a successful attack, the attacker needs access to the output of the decryption function, which is not possible in this model. Otherwise, attacking the decryption circuit is similar to a classical differential attack on the encryption circuit.

2) Ciphertext from the block cipher is bigger than actual plaintext under communication

Here we are deviating a little from the classical notion of encryption-and-decryption, where the plaintext (to be communicated) and ciphertext (which is actually communicated) are of equal size. Under this mode, plaintext (before feeding to the block cipher) size is smaller than that of ciphertext.

3) Not the entire bandwidth is used

A part of the information communicated is the tweak-in. This leads to under utilization of the bandwidth. Yet, we argue that this little performance penalty is largely compensated by other performance (no need to compute the key schedule every encryption for example), security, and simplicity gains.

4) Limited to DFA protection only

The proposed scheme, in its current form, addresses DFA only. Other advanced fault attacks like biased fault attacks [9] or ciphertext-only fault attacks [12] which do not require an unchanged plaintext for faulty and non-faulty cipher execution, cannot be protected by this scheme.

IV. Analysis of the Scheme

A. Fault Complexity Analysis

The proposed mode provides a safeguard against DFA. In other words, it increases the number of encryptions and comparisons required by multiplicative factor $G_\mu(t)$ and $H_\mu(t)$, respectively, where t is the tweak-in size, μ is the number of faults required for the DFA and $G_\mu(\cdot)$ and $H_\mu(\cdot)$ depend on how the tweak-in is generated and on the encryption mode used. To analyze different tweak-in generation methods, we assume that the attacker tries to perform a conventional DFA attack on the underlying block cipher, and to find a collision on the tweak-in value. We also assume that the attacker can insert faults during the block cipher execution but not during the tweak-in generation and is limited to at most one fault per-execution (single-fault model). The attacker finds a collision on the tweak-in value by checking the ciphertext value and finding a pair that is partially identical, except for the faulty bits.

For non-symmetric modes, there are two possible models for generating the tweak-in and the corresponding factors. In order to attack any of them, the goal is to find μ collisions over the tweak, which is the motivation behind the attack on each of them.

1) **Pseudo-random tweak-ins.** The tweak-ins are generated using a Pseudo-Random Function (PRF). In this model, the user chooses a random t-bit string as the tweak-in, every time the encryption algorithm is invoked. The attacker's goal is to generate a pair of faulty and non-faulty ciphertexts, with the same tweak-in and plaintext. Due to the birthday paradox, the average number of collisions after n trials is approximately $\frac{n^2}{2^t}$. Therefore, after $\mu^{\frac{1}{2}}2^{\frac{t}{2}}$ encryptions, it is expected that μ collisions have occurred. If one half of the encryptions are faulty, then, with probability $2^{-\mu}$, these collisions lead to meaningful pairs of faulty and non-faulty ciphertexts. Overall, $G_\mu(t) = \mu^{\frac{1}{2}}2^{\mu+\frac{t}{2}}$ encryptions are required, on average, for the attack. For the analysis part of the attack, each non-faulty block needs to be compared to $\mu^{\frac{1}{2}}2^{\mu+\frac{t}{2}-1}$ faulty blocks, leading to an overall $H_\mu(t) = \mu 2^{2\mu+t-2}$ comparisons. To sum up, the attack complexity is $\mathcal{O}\left(\mu^{\frac{1}{2}}2^{\mu+t/2}\right)$ encryptions and $\mathcal{O}\left(\mu 2^{2\mu+t}\right)$ comparisons.
2) **Pseudo-random sequence of tweak-ins.** The tweak-ins are generated using a Pseudo-Random Sequence (PRS). In this model, the attack procedure is simpler to describe. The attacker runs 2^t non-faulty encryptions to cover the whole tweak-in space. Afterwards, the attacker runs μ faulty encryptions that collide with the first μ non-faulty encryptions. Overall, $G_\mu(t) = 2^t + \mu$ and $H_\mu(t) = \mu$. Additionally, as long as the tweak-in does not cycle, the attacker in this model does not learn any information.

The 1st model provides a sub-exponential safeguard in terms of encryptions and an exponential safe-guard in terms of comparisons; as opposed to exponential and constant safeguards in case of the 2nd model, respectively. Hence, depending on the underlying cipher and implementation, there is a trade-off to be made between the first two models. It seems that the 2nd model requires more encryptions. However, the 1st model requires higher number of comparisons and sup-exponentially more faults.

Table I summarizes the security gains achieved in both types with different tweak-in generation models.

Security against multiple fault attacks

Independent two stage attacks, which perform DFA against the tweak-in generation function, followed by DFA against the underlying block cipher are not applicable. The reason is that to launch such attacks the attacker need access to the output of the tweak-in generation function in the first stage. However, what the attacker sees is an encrypted version of the tweak-in generation function output, after it passes through the block cipher.

TABLE I: Comparison between the Security Gains for Different Types of Encryption Modes and Tweak-in Generation Models

Tweak-in generation model	Required # of faults	Required # of encryptions	Analysis complexity
PRF (1)	$\mathcal{O}\left(\mu^{\frac{1}{2}}2^{\mu+t/2}\right)$	$\mathcal{O}\left(\mu^{\frac{1}{2}}2^{\mu+t/2}\right)$	$\mathcal{O}\left(\mu 2^{2\mu+t}\right)$
PRS (2)	μ	$\mathcal{O}\left(2^t+\mu\right)$	μ

On the other hand, a more interesting attack is a modified DFA attack on the block cipher itself, where instead of generating a pair of faulty/non-faulty ciphertexts, the attacker generates two faulty ciphertexts as follows:

1) When the tweak value is t_0, the attacker injects a fault δ in the first round of the block cipher, such that $t_0 \oplus \delta = t_1$, where t_1 is the next tweak value. She observes the output $E_K(p|t_1)$.
2) The attacker then generates the next ciphertext $E_K(p|t_1)$ while performing the conventional fault injection required for DFA against the block cipher without tweak-in. We call that $E_K^f(p|t_1)$.
3) Using the pair $(E_K(p|t_1), E_K^f(p|t_1))$, DFA can be performed.

The success probability of this attack depends on the success probability of the attacker to predict δ when t_0 is secret. In case the tweak generation function is a PRF (stream-cipher), the probability should be 2^{-t}, where t is the tweak-in bit size. This means that such attack has no advantage over the collision attack described in Section IV-A.

B. Tweak-ins Generation

It is to be noted that the generation of tweak-ins is the only overhead of the proposed differential fault resistance scheme. To keep the scheme lightweight, it is recommended to use lightweight solutions for tweak-in generation.

Recall that the 1st model requires a PRF as a tweak-in generator. Here, we propose several possible candidates in this context. Generally, stream ciphers are the default choice when it comes to lightweight PRF. Naturally, we propose a few stream ciphers which are suitable for our scheme. Depending on the type of platform (hardware/software), the particular stream cipher can be chosen. For example, as the finalists of eSTREAM project [34], `HC-128`, `RABBIT`, `SALSA20/12` and `SOSEMANUK` are recommended for software platforms; as these stream ciphers are well-suited for software. On the other hand; `GRAINv1`, `MICKEY2.0` and `TRIVIUM` are more suited for hardware targets. Further, on-chip random number generators can be also used for tweak-in generation.

As for the 2nd model, the requirements are slightly different. This model requires the tweak-in to be derived from a pseudo-random sequence. To be exact, the properties needed are:

1) Non-recurrence

All the possible values have to be generated before values are reused, which is the only property used in the analysis in Section IV-A.

2) Balanced Differences

The differences between every two consecutive values have to be somewhat balanced. In other words, if the attacker does not know the current value, she should only be able to make a guess on the difference between the current value and next value with probability less than $\epsilon \cdot 2^{-t}$, where ϵ is small. This property is required in order to prevent multiple fault attacks on the tweak-in and block cipher, simultaneously, as will be shown later. This property is also inherently fulfilled in case of a PRF. This property can also be defined as the two-level auto-correlation of the PRS at a phase difference of 1.

However, using a stream cipher (PRF) to generate tweak-ins can be costly, let alone using a PRP to generate the PRS. The good news is that there exists a simpler family of functions that satisfies both the non-recurrence (so no birthday bound) and balanced differences properties at a very low cost compared to a strem cipher, at the expense of having a sequence length of $2^t - 1$ as opposed to 2^t. This family is the Maximal Length LFSRs, or m-sequences [20]. m-sequences satisfy Golombs Randomness Postulates [19] for PRS. Among these properties, the two-level autocorrelation is the most important.

To conclude, there is a security trade-off in the tweak-in generation. Using a stream cipher to generate the tweak-in provides higher DFA computational cost, but suffers from the birthday bound over the tweak-in size and the costly stream cipher computation. On the other hand, the birthday bound can be removed by using a maximal length LFSR, which has much lower cost, but the DFA computational cost will be smaller. The last problem can be removed by reseeding the LFSR once every $2^t - 1$ blocks.

C. Performance Analysis and Implementation Results

1) Hardware Implementation

In order to demonstrate the performance/area vs. security trade-off in hardware implementations, an FPGA implementation of `AES-128` with the proposed countermeasure has been implemented, for various values of t. The `AES` circuit used is the round-based multi-stream circuit described in [25]. Eight values of t has been selected; $4, 8, 12, 16, 20, 24, 28, 32$. The tweak-in generation function is selected to be an m-sequence, in order to minimize the implementation overhead and remove the birthday bound on security. Hence, for every value of t, a maximal length LFSR with minimum number of taps is used. The design is synthesized for Virtex 6 FPGA families. The synthesis constraints have been adjusted towards low area implementations. The post-synthesis results are provided in Table II. All the LFSRs used can be implemented using only $1 \sim 3$ XOR gates. Hence, they have no effect on the operating frequency and the area overhead is mainly due to the Flip-Flops. The results show that the area overhead is 1 LUT for the combinational part of the LFSR and t Flip Flops for storage. The area overhead increases linearly with the tweak-in size. Table II shows the throughput/area overhead vs. the tweak-in size. The efficiency drop is linear with respect to the tweak-in size and the slope is not steep. On the other hand, the number of blocks that can be encrypted between two reseeding operations is $2^t - 1$. Compared to the exponential security gain, the proposed countermeasure is considered a lightweight countermeasure.

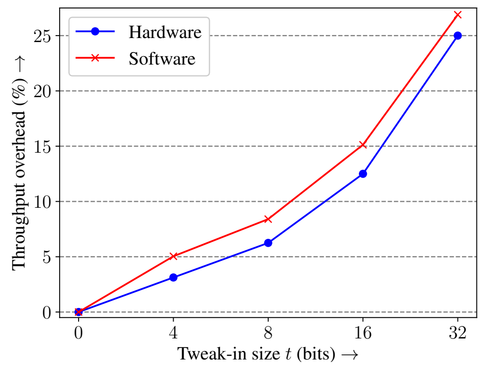


FIG. 5: Throughput overhead percentage vs. Tweak-in size (t) based on maximal length LFSRs with minimum number of taps for hardware and software

2) Software Implementation

The scheme has been tested on an `AES-128` encryption module running on an Atmel ATmega328P microcontroller at operating frequency of 16 MHz, with 1 KB EEPROM, 32 KB Flash, and 2 KB SRAM. It was used to encrypt a 7 Gbit file, using different sizes for the tweak-in: $4, 8, 16, 32$. The performance results are shown in Table III. A speed-optimized unprotected implementation takes $\approx 100.9 \times 10^9$ clock cycles for the encryption. This includes 756 clock cycles for the initial key expansion and 1718 per block encryption. When protected using the 2nd, i.e., non-recurring pseudo-random sequence, encrypting the same block takes between 106.1×10^9 and 138.3×10^9 clock cycles (the respective overhead is $\approx 5\%$ to $\approx 26.9\%$), for tweak-in sizes between 4 and 32 bits, respectively.

Figure 5 shows a comparison between the overhead of the proposed scheme for both hardware and software. The overhead in case of software is slightly higher, as the LFSR operation is more costly and time consuming, while in case of hardware, it is computed in parallel to the encryption itself.

D. Fault Detection

While the proposed construction does not require the knowledge of the tweak-in at the recipient side, the value is sent implicitly in an encrypted form. If the sender (Alice) and recipient (Bob) parties share the knowledge about the tweak-in generation method, the recipient (Bob) can compare the two values associated with each two consecutive blocks and detect whether these values conform to the pre-determined generation method. For example, if the tweak-in is generated using an

TABLE II: FPGA Performance Results

Tweak-in size t (bits)		0	4	8	12	16	20	24	28	32
# of blocks before reseeding		—	15	63	4095	$2^{16}-1$	$2^{20}-1$	$2^{24}-1$	$2^{28}-1$	$2^{32}-1$
Virtex 6	LUT-FF pairs	589	595	599	603	607	611	615	619	623
	Throughput (Gbps)	5.312	5.146	4.98	4.814	4.648	4.482	4.316	4.15	3.984
	Throughput/Area (Mbps/LUT)	9.01	8.65	8.31	7.90	7.66	7.34	7.02	6.70	6.40

TABLE III: Software Performance Results

t (bits)	# of encryptions	# of cycles	Throughput (Mbps)
0	58.7×10^6	100.9×10^9	1.19
4	60.6×10^6	106.1×10^9	1.13
8	62.6×10^6	110.6×10^9	1.09
16	67.1×10^6	118.5×10^9	1.01
32	78.2×10^6	138.3×10^9	0.87

LFSR, $\forall$ blocks i the associated tweak-in is T_i, and T_{i+1} must conform to $\texttt{lfsr}(T_i)$. It increases the work required by the attacker (Eve) during fault insertion, as she has to block the communication channel between the sender and the recipient. Otherwise, the recipient, e.g., the base station in case of a WSN, can easily detect the anomaly and perform necessary countermeasures. These countermeasures may include, for example, fixing the faulty unit or take it out of the system and update the secret keys.

E. Security Analysis

In this section, we discuss the security of the proposed construction regarding classical attacks, as opposed to DFAs discussed earlier. The security relies heavily on the security of the underlying block cipher. We discuss three security notions below. We assume the tweak-in is known by the attacker, but not controllable.

a) Known Plaintext Attacks (KPA)

Assuming the attacker knows the tweak-in value, a KPA reduces to a KPA of the underlying block cipher, as the attacker knows the whole plaintext block.

b) Chosen Plaintext Attacks (CPA)

Assuming the attacker knows the tweak-in value, a CPA gives the attacker a partial control over the plaintext block. This is stronger than the KPA and weaker than the CPA against the underlying block cipher. Therefore, the security is at least the security of the underlying cipher against CPA.

c) Chosen Ciphertext Attacks (CCA)

The CCA does not provide meaningful information regarding the encryption side. Hence, the security against CCA relies entirely on the security of the underlying block cipher.

F. Comparison to `CTR` and Other Symmetric Modes

At first, our proposed construction seems similar to the conventional counter (`CTR`) mode. In this section we explain the differences between the two constructions, which are also applicable to symmetric modes. Despite these differences, we acknowledge that the proposed construction is more suited towards non-symmetric modes, where both the `encrypt` and `decrypt` functions are used.

1) The symmetric modes require synchronization of the initialization vector (IV) between the sender and recipient sides, while in our construction, the tweak-in value is mixed with the plaintext and sent to the recipient side in an encrypted form.
2) Most of the protocols that use symmetric mode require the IV/counter value not to be repeated for the same key. If such requirement is fulfilled, they behave in the same way as our construction with respect to DFA. However, this criterion depends on the security of the synchronization process. The attacker can try to deceive the sender into using the same (K, IV) pair, generating a pair of faulty and non-faulty ciphertexts. In our construction, since the synchronization step is removed, the attacker has to rely on the collision attacks described in Section IV-A.

V. Conclusion

In this paper, we propose a new scheme which acts as a safeguard against Differential Fault Attacks (DFA). We change a part of the plaintext ('tweak-in') input to the underlying block cipher at any two consecutive encryption calls. This scheme is very easy to implement on a common block cipher (in both hardware and software implementation) and lightweight. We analyze this scheme in details under two different models for generating the tweak-in. An interesting follow-up work could be to observe how this kind of scheme can be used as a protection against side-channel attacks.

References

[1] Banik, S., Bogdanov, A.: Cryptanalysis of two fault countermeasure schemes. In: Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings. (2015) 241–252

[2] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. IACR Cryptology ePrint Archive **2004** (2004) 100

[3] Battistello, A., Giraud, C.: A note on the security of CHES 2014 symmetric infective countermeasure. In: Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers. (2016) 144–159

[4] Bertoni, G., Breveglieri, L., Fragneto, P., Macchetti, M., Marchesin, S.: Efficient software implementation of aes on 32-bit platforms. In Kaliski, B.S., Koç, ç.K., Paar, C., eds.: Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers, Berlin, Heidelberg, Springer Berlin Heidelberg (2003) 159–171

[5] Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In Kaliski, BurtonS., J., ed.: Advances in Cryptology - CRYPTO '97. Volume 1294 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1997) 513–525

[6] Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings. (2009) 1–18

[7] Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. (2009) 231–249

[8] Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings. (2002) 384–397

[9] Dobraunig, C., Eichlseder, M., Korak, T., Lomné, V., Mendel, F.: Statistical fault attacks on nonce-based authenticated encryption schemes. In: Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22, Springer (2016) 369–395

[10] Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F.: On the security of fresh re-keying to counteract side-channel and fault attacks. In: Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers. (2014) 233–244

[11] Dobraunig, C., Koeune, F., Mangard, S., Mendel, F., Standaert, F.: Towards fresh and hybrid re-keying schemes with beyond birthday security. In: Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers. (2015) 225–241

[12] Fuhr, T., Jaulmes, E., Lomné, V., Thillard, A.: Fault attacks on aes with faulty ciphertexts only. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on, IEEE (2013) 108–118

[13] Ghosh, S., Saha, D., Sengupta, A., Chowdhury, D.R.: Preventing fault attacks using fault randomization with a case study on AES. In: Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings. (2015) 343–355

[14] Gierlichs, B., Schmidt, J., Tunstall, M.: Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In: Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings. (2012) 305–321

[15] Guilley, S., Sauvage, L., Danger, J., Selmane, N.: Fault injection resilience. In: 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010. (2010) 51–65

[16] He, W., Breier, J., Bhasin, S.: Cheap and cheerful: A low-cost digital sensor for detecting laser fault injection attacks. In: Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. (2016) 27–46

[17] He, W., Breier, J., Bhasin, S., Chattopadhyay, A.: Bypassing parity protected cryptography using laser fault injection in cyber-physical system. In: Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security, ACM (2016) 15–21

[18] He, W., Breier, J., Bhasin, S., Miura, N., Nagata, M.: Ring oscillator under laser: Potential of pll-based countermeasure against laser fault injection. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on, IEEE (2016) 102–113

[19] Helleseth, T.: Golombs randomness postulates. In: Encyclopedia of Cryptography and Security. Springer (2011) 516–517

[20] Helleseth, T.: Maximal-length sequences. In: Encyclopedia of Cryptography and Security. Springer (2011) 763–766

[21] Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II. (2014) 274–288

[22] Jean, J., Nikolic, I., Peyrin, T., Seurin, Y.: Deoxys v1.4 Submission to CAESAR.

[23] Joye, M., Manet, P., Rigaud, J.B.: Strengthening hardware aes implementations against fault attacks. IET Information Security **1**(3) (2007) 106

[24] Karri, R., Kuznetsov, G., Gössel, M.: Parity-based concurrent error detection of substitution-permutation network block ciphers. In: Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings. (2003) 113–124

[25] Khairallah, M., Chattopadhyay, A., Peyrin, T.: Looting the LUTs : FPGA Optimization of AES and AES-like Ciphers for Authenticated Encryption. The 18th International Conference on Cryptology in India - IndoCrypt (2017)

[26] Kim, C.H., Quisquater, J.: Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures. In: Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, First IFIP TC6 / WG 8.8 / WG 11.2 International Workshop, WISTP 2007, Heraklion, Crete, Greece, May 9-11, 2007, Proceedings. (2007) 215–228

[27] Kulikowski, K., Karpovsky, M., Taubin, A.: Robust codes for fault attack resistant cryptographic hardware. In: Fault Diagnosis and Tolerance in Cryptography, 2nd International Workshop, Citeseer (2005) 1–12

[28] Lac, B., Canteaut, A., Fournier, J., Sirdey, R. In: DFA on LS-Designs with a Practical Implementation on SCREAM. Springer International Publishing, Cham (2017) 223–247

[29] Liskov, M., Rivest, R.L., Wagner, D.A.: Tweakable block ciphers. In: Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. (2002) 31–46

[30] Lomné, V., Roche, T., Thillard, A.: On the need of randomness in fault attack countermeasures - application to AES. In: 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012. (2012) 85–94

[31] Medwed, M., Standaert, F., Großschädl, J., Regazzoni, F.: Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In: Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010. Proceedings. (2010) 279–296

[32] Moro, N., Heydemann, K., Encrenaz, E., Robisson, B.: Formal verification of a software countermeasure against instruction skip attacks. J. Cryptographic Engineering **4**(3) (2014) 145–156

[33] Patranabis, S., Roy, D.B., Mukhopadhyay, D.: Using tweaks to design fault resistant ciphers. In: VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), 2016 29th International Conference on, IEEE (2016) 585–586

[34] Robshaw, M., Billet, O.: New stream cipher designs: the eSTREAM finalists. Volume 4986. Springer (2008)

[35] Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings. (2004) 16–31

[36] Selmane, N., Bhasin, S., Guilley, S., Graba, T., Danger, J.: WDDL is protected against setup time violation attacks. In: Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009. (2009) 73–83

[37] Selmke, B., Heyszl, J., Sigl, G.: Attack on a DFA protected AES by simultaneous laser fault injections. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016. (2016) 36–46

[38] Tupsamudre, H., Bisht, S., Mukhopadhyay, D.: Destroying fault invariant with randomization. In: International Workshop on Cryptographic Hardware and Embedded Systems, Springer (2014) 93–111

[39] Yen, S., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Trans. Computers **49**(9) (2000) 967–970

[40] Zussa, L., Dehbaoui, A., Tobich, K., Dutertre, J.M., Maurine, P., Guillaume-Sage, L., Clediere, J., Tria, A.: Efficiency of a glitch detector against electromagnetic fault injection. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, IEEE (2014) 1–6