Protecting block ciphers against differential fault attacks without re-keying

Baksi, Anubhab; Bhasin, Shivam; Breier, Jakub; Khairallah, Mustafa; Peyrin, Thomas

2018

Baksi, A., Bhasin, S., Breier, J., Khairallah, M., & Peyrin, T. (2018). Protecting block ciphers against differential fault attacks without re-keying. 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). doi:10.1109/HST.2018.8383913

<https://hdl.handle.net/10356/88761>

<https://doi.org/10.1109/HST.2018.8383913>

© 2018 Institute of Electrical and Electronics Engineers (IEEE). All rights reserved. This paper was published in 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) and is made available with permission of Institute of Electrical and Electronics Engineers (IEEE).

Downloaded on 26 Apr 2023 22:34:57 SGT

Protecting Block Ciphers against Differential Fault Attacks without Re-keying

Anubhab Baks¹, Shivam Bhasin², Jakub Breier², Mustafa Khairallah³, 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.

Differential Fault Attack (DFA), a common type of physical attack, is usually used against symmetric key ciphers. Because of its immense popularity among the research community, one can find works exploiting DFA aimed at all major ciphers.

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. Refer to [1] for a quick overview of the countermeasures proposed.

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.

II. BACKGROUND

A. Birthday Bound for Fresh Re-keying Schemes

Here we recall that the birthday bound for fresh re-keying scheme [5] suffers from the birthday bound. This result is reported in CARDIS’14 [2].

The 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 [5]).

B. Tweakable Block Cipher Based Solution

The CARDIS’15 paper by Dobraunig *et al.* [3] 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. In this regard, one may notice that there is a recent trend in developing tweakable block cipher in the community. However, as pointed out in [1, Section II-B], tweakable block cipher has its own setbacks.

C. Our Approach: Tweaking the Plaintext

Our technique will be referred to as the *Tweak-in-Plaintext* scheme from now on. A basic requirement of DFA is to obtain faulty and non-faulty ciphertext, corresponding to a fixed plaintext. 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’). This results in under

utilization of the bandwidth, which falls into the detection category. We use the term ‘tweak-in’ instead of the common term, ‘tweak’, so as to distinguish the use cases.

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 [5].

D. Non-Symmetric Block Cipher 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.

III. DESCRIPTION OF TWEAK-IN-PLAINTEXT SCHEME

Tweak-in. 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.

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 .

Next, we describe the potential advantages and limitations of our scheme in a nutshell. For a detailed discussion, interested readers may refer to [1].

A. Advantages

- 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*
- b) *No need to recompute the key schedule every time*
- c) *Allows implementations with wired-in key*
- d) *No need for a tweakable block cipher*
- 2) *No need to send any extra data/ No synchronization is required*
- 3) *Almost readily portable in existing devices*
- 4) *Suitable for lightweight applications*
- 5) *Error detection is possible if tweak is synchronized (optional)*

B. Limitations

- 1) *Only encryption can be protected*

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.

- 2) *Ciphertext from the block cipher is bigger than actual plaintext under communication*
- 3) *Not the entire bandwidth is used*

This is because, a part of the information communicated is the tweak-in.

- 4) *Limited to DFA protection only*

The applicability of other type of faults attacks are out of scope of the current work.

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

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 $\mathcal{O}(\mu^{\frac{1}{2}} 2^{\mu+t/2})$ encryptions and $\mathcal{O}(\mu 2^{2\mu+t})$ 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 **IV-A1** 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 **IV-A2** 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 **IV-A2** model requires more encryptions. However, the **IV-A1** 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.

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 (IV-A1)	$\mathcal{O}(\mu^{\frac{1}{2}} 2^{\mu+t/2})$	$\mathcal{O}(\mu^{\frac{1}{2}} 2^{\mu+t/2})$	$\mathcal{O}(\mu 2^{2\mu+t})$
PRS (IV-A2)	μ	$\mathcal{O}(2^t + \mu)$	μ

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.

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

Recall that the **IV-A1** 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, 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 **IV-A2** 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 stream cipher, at the expense of having a sequence length of $2^t - 1$ as opposed to 2^t .

C. Performance Analysis and Implementation Results

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

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 [4]. 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 ~ 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.

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

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 IV-A2, 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.

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] Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T.: Protecting block ciphers against differential fault attacks without re-keying (extended version). Cryptology ePrint Archive, Report 2018/085 (2018) <https://eprint.iacr.org/2018/085>.
- [2] 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
- [3] 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
- [4] 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)
- [5] 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