Persistent Fault Analysis on Block Ciphers

Fan Zhang[1,2], Xiaoxuan Lou[1,2], Xinjie Zhao[4], Shivam Bhasin[5], Wei He[6], Ruyi Ding[1,7], Samiya Qureshi[1] and Kui Ren[3]

[1] College of Information Science and Electronic Engineering, Zhejiang University, China
[2] Institute of Cyber Security Research, Zhejiang University, China
[3] College of Computer Science and Technology, Zhejiang University, China
[4] The Institute of North Electronic Equipment, China
[5] Nanyang Technological University, Singapore
[6] Shield Laboratory, Huawei International Pte. Ltd., Singapore
[7] Georgia Institute of Technology, GA, United States
fanzhang@zju.edu.cn

Abstract. Persistence is an intrinsic nature for many errors yet has not been caught enough attractions for years. In this paper, the feature of persistence is applied to fault attacks, and the *persistent fault attack* is proposed. Different from traditional fault attacks, adversaries can prepare the fault injection stage before the encryption stage, which relaxes the constraint of the tight-coupled time synchronization. The *persistent fault analysis* (PFA) is elaborated on different implementations of AES-128, specially fault hardened implementations based on Dual Modular Redundancy (DMR). Our experimental results show that PFA is quite simple and efficient in breaking these typical implementations. To show the feasibility and practicability of our attack, a case study is illustrated on the shared library Libgcrypt with rowhammer technique. Approximately 8200 ciphertexts are enough to extract the master key of AES-128 when PFA is applied to Libgcrypt1.6.3 with redundant encryption based DMR. This work puts forward a new direction of fault attacks and can be extended to attack other implementations under more interesting scenarios.

Keywords: Fault Analysis · PFA · DMR · AES · Libgcrypt · Rowhammer.

1 Introduction

Fault attack (FA) is a class of implementation level attacks on embedded systems [Joy12], which is usually used to attack different ciphers such as RSA, AES, PRESENT[BKL+07], LED[GPPR11], Piccolo[SIH+11]. FA is an active attack that disturbs the operation of target device. The disturbance is realized by forcing the device in a non-nominal operating condition. Common methods include changing the power supply voltage, changing the frequency of the external clock, varying the temperature or exposing the circuits to lasers during the key-dependent computations such as encryptions [BECN+06, ACS+07, TJ09, Insa, Insb]. The idea of fault attack was first reported on RSA-CRT by Boneh et al. in 1996 [DDL97]. Later, Biham and Shamir proposed the *differential fault analysis* (DFA) attack on the block cipher DES, which combines a fault attack with differential cryptanalysis [BS06]. Since then, DFA has been used to break various block ciphers [BS97, DLV03, TMA11]. Apart from breaking cryptographic systems, FA is also used for other attacks, like bypassing security checks in smartcards [BECN+06].

DFA operates in a differential setting, *i.e.*, exploiting the difference of correct and faulty ciphertexts for a fixed input. Later, other fault analysis techniques were introduced. Some analysis techniques exploited the algebraic structure of the algorithm with fault injection

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925, Vol. 2018, No. 3, pp. 150–172
DOI:10.13154/tches.v2018.i3.150-172

(AFA [CWJ10]) in a differential setting. Other analysis methods exploited statistical biases introduced due to fault injection [Riv09, FJLT13]. These biases could be either exploited in a differential setting or with faulty ciphertexts only.

Most, if not all, of the proposed fault analysis are developed with a transient fault assumption. This means that the injected fault does not persist from one encryption to another. A fault is injected during a target computation, while all other computations remain unaffected. Fault models, such as bit flips, random byte, etc., are often used in a transient fault setting. Alternately, some fault analysis techniques assume permanent faults. Permanent faults are equivalent to device defects which stay during the lifetime. As the fault is fixed, the bias introduced is exploited by statistical means. Stuck-at fault model is a common example of this kind.

In this paper, we develop fault analysis for a third kind of fault model, called as *persistent fault.* This fault falls between transient and permanent. While the fault persists from one encryption to another, it disappears when the target device reboots. We propose a statistical technique to exploit such faults, called as *Persistent Fault Analysis* (PFA).

For aforementioned analysis techniques to work, it is desirable that the fault is injected in the last few rounds of the cipher. If the fault is injected much deeper into the cipher (middle rounds), the analysis becomes too complex and does not give much advantage over a simple brute force search. Moreover, majority of the known attacks can only handle a single fault injection. This requirement puts several restrictions on the attacker's capability, as he is expected to inject single faults in later rounds only.

On the contrary, PFA assumes that a persistent fault might be always present. A common example is when the fault is injected into an algorithm constant stored in memory (ROM) like one element in S-box. Unless the ROM is refreshed, the fault will persist for all the subsequent encryptions, however only the rounds accessing that particular faulty S-box element will be affected. A DFA (or other aforementioned techniques) cannot be applied in this setting for two reasons. Firstly, fault can be multiple and present in earlier rounds. Secondly, it is not possible to acquire correct and faulty ciphertext in presence of persistent fault for a given plaintext, which prevents any differential analysis. The proposed PFA is developed to exploit such cases, where a fault is persistent and can affect multiple rounds. From a practical aspect, PFA also relaxes some constraints enforced on the adversary. The adversary does not need to synchronise the fault injection with later rounds in run time, and also does not require re-encryption for correct/faulty ciphertext pairs. The attack target can be perturbed before hand, by injecting the fault which persists and exploited later. When the victim encrypts on the plaintext, the adversary observes the resultant ciphertext and performs PFA to retrieve the secret key.

PFA is also capable of compromising some of the widely used fault countermeasures under its basic fault model. We target dual module redundancy (DMR), which performs redundant operations followed by comparison to detect faults. This countermeasure is also used for reliability verification, and thus widely adopted in commercial products. It is believed to be provably secure against single fault injection. As we show later, such countermeasures can be easily broken by PFA. While some versions of DMR are broken by design, others can be broken with higher number of available ciphertexts. For example, in AES, the attack roughly needs $10\times$ more ciphertexts than an unprotected design.

The main contributions of this work are summarized as follows:

- We target a new category of injected faults, called persistent faults.
- Based on persistent fault, we develop a fault analysis technique called *persistent fault analysis* (PFA) and explain its working mechanism. Unlike common fault analysis technique, PFA is not differential and it uses statistical means for key recovery. Thus, it is a faulty ciphertext only attack.
- We extend PFA to work in a multiple-faults setting.

- PFA is first validated on S-box and T-box based AES-128 on Virtex-5 FPGA. Xilinx data2mem software is used for emulation of persistent faults.
- PFA is then shown to break fault countermeasures based on Dual Modular Redundancy (DMR). Different variants of the countermeasure were broken with $2 - 10\times$ extra ciphertexts as compared to unprotected designs.
- PFA is then practically validated on a server under a shared library setting. By practical rowhammer attacks, on the fault hardened shared cryptographic library Libgcrypt, PFA is shown to successfully recover the secret key with 8200 ciphertexts.

The rest of the paper is organised as follows. Section 2 introduces the related work. Section 3 highlights the core idea, the process of PFA, the complexity and the comparison with other fault attacks. Section 4 extends PFA with multiple faults. Section 5 gives the background of AES implementations. Section 6 evaluates the PFA on different scenarios, especially on those with fault attack countermeasures. Section 7 shows a case study of PFA with the injection technique of rowhammering. Section 8 concludes the paper.

2 Related Works

Fault Duration: Faults in electronic circuits can be either *permanent* or *transient*. A *permanent fault* is caused by intentional or unintentional defects in the chip [Sko10], which permanently modifies its functionality. In contrast, a *transient fault* [BECN$^+$06] only influences the device for a very short time. A common application of transient faults is corrupting a single execution of an encryption. In this work, we are more interested in the third category called as *persistent fault*. The term "persistent" refers to the characteristic of a new type of faults whose duration may not be permanent and typically can last for several encryptions, for example, a few minutes or up to a few hours. Sometimes it might be persistent till the device is reset. An example of such fault is a modification of a stored constant, like an S-box entry, using rowhammer injection techniques [KDKF14]. Rowhammer injection techniques are used in some previous works to attack ciphers [BM16, RGB$^+$16, XZZT16]. With such faults, all the rounds of all encryptions will be affected. A reboot or refresh of the affected memory will restore the original functionality.

Fault Analysis: Most fault analysis techniques are differential in nature. They require a correct and faulty computation with same inputs, to exploit the difference of outputs for key recovery. Typical techniques include differential fault analysis(DFA) [BS06], algebraic fault analysis (AFA) [CWJ10], fault rate analysis (FRA) [WCWW13] and more. Other techniques are statistical in nature and sometimes exploit faulty ciphertexts only. Common examples are statistical fault analysis (SFA) [Riv09, FJLT13] and fault sensitivity analysis (FSA) [LSG$^+$10]. In practical fault attacks, adversaries usually face a ciphertext-only scenario, with no or limited control on inputs. In addition, some countermeasures restrict multiple encryptions with the same inputs by using techniques like random value padding [BBB$^+$18]. In this paper, we are interested in exploiting fault attacks that can be conducted under ciphertext-only attack scenario. A through comparison of PFA against other common analysis techniques is drawn in Section 3.5.

Persistent Fault Attack: The notion of *persistent fault attack* is not new. In [SHP10], a ultraviolet light was used to erase the contents of a microcontroller, particularly lookup tables. However, the precision of the attack was limited. Also the offline analysis in [SHP10] was differential and not developed particularly to exploit persistent faults. In [ASSS16], the AES lookup table implemented on the Xilinx FPGA using embedded block memories (BRAMs) was tampered, where the persistent attack is firstly mounted on the hardware implementation. However, the attack model was too strong and corresponding offline persistent analysis in [ASSS16] was very simple due to their assumption that the entire AES table was set as all zeroes and the last round key can be directly output as the ciphertext.

3 Persistent Fault Attack

This section provides details about the proposed *persistent fault analysis* methodology.

3.1 Fault model

The assumed fault model is as follows:

- The adversary can inject faults before the encryption of a block cipher. Typically, these faults alter a stored algorithm constant.
- The injected faults are persistent, *i.e.*, the affected constant stays faulty unless refreshed. Thus all iterations are computed with the faulty constant.
- The adversary is capable of collecting multiple ciphertext outputs. Thus, a watchdog counter on detected faults is considered out of scope.

In this section, we first show the analysis with single fault injection. Exploitation of multiple fault injections is discussed in next section.

3.2 Core idea

As stated in the fault model, the fault persists over several computations. In case of block ciphers, the prime target of this attack, these computations refer to the round function. Each encryption is composed of several calls of a round function. The injected fault persists over several encryptions (thus round function calls), but the faulty value may not be accessed. For example, if the fault exists in an S-box element, the round computation is only faulty if the faulty S-box element is accessed. Otherwise, the injected fault does not impact this round computation. If the faulty value is not accessed during an encryption, the resultant ciphertext will be correct, otherwise incorrect. We further exploit the statistical distribution of correct and incorrect ciphertexts to reveal key-dependant information. We call this newly developed fault analysis technique as ***Persistent Fault Analysis*** (**PFA**). The corresponding attack is called as ***Persistent Fault Attack***.

The complete persistent fault attack is composed of three stages. In the fault injection stage, the persistent fault is injected before the first encryption. Unlike traditional DFA, identification of exact round timing or precise location is not required. In the encryption stage, the adversary (denoted as $\mathcal{A}$) then waits for the victim (denoted as $\mathcal{V}$) to start the encryptions. $\mathcal{A}$ can then observe the produced ciphertexts, few of which are correct while others incorrect due to the persistent fault. In the fault analysis stage, $\mathcal{A}$ analyses the mixture of correct and faulty ciphertexts with PFA to recover the secret key. As shown later, PFA can be applied on unprotected implementation as well as some state of the art fault hardened implementations.

3.3 Persistent Fault Analysis (PFA)

In this part, we further detail the analysis technique of a persistent fault attack. While the analysis technique remains generic, for sake of simplicity, we start to explain with an example of SPN block cipher targeted for last round key recovery.

Let us assume a typical SPN construction with L words of b bits. A b-bit input is processed by a substitution component (typically, S-box), followed by linear permutation layer (LP) and a round key addition. Next, we take PFA on the last round of SPN block cipher as an example to describe the technique. As LP is linear, we remove it for a simpler analysis. Let x_j and y_j denote the j^{th} word of last round, at input and output of S-box respectively. y_j when mixed with j^{th} word of last round key, produces j^{th} word of ciphertext c_j. Then, it satisfies $y_j \oplus k_j = c_j$ which is equivalent to the following:

$$k_j = y_j \oplus c_j \tag{1}$$

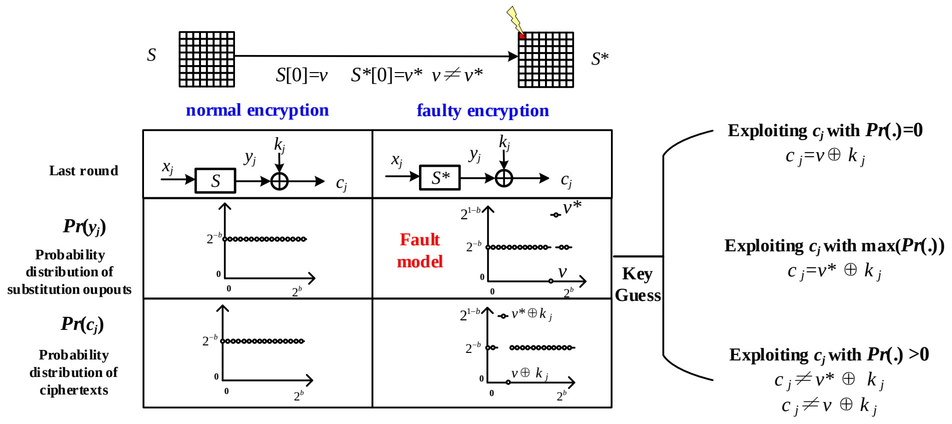


Figure 1: Overview of persistent fault analysis (PFA).

Let $Pr(y_j)$ and $Pr(c_j)$ denote the distribution probability of y_j and c_j respectively. As for the correct encryption, due to the avalanche effect, for each candidate of y_j, $Pr(y_j)$ is close to 2^{-b}. Let us assume that the fault is injected in the first S-box element, where correct value $S[0] = v$ is altered to $S'[0] = v^*$ and $v \neq v^*$. The same is illustrated in Fig. 1. The fault injection makes $Pr(y_j = v)$ as zero. As the number of observed encryption increases, $Pr(y_j = v^*)$ approaches to 2^{1-b}. For all other values of y_j, $Pr(y_j)$ converges to 2^{-b}. This difference in probabilities can be statistically distinguished, thus requiring multiple ciphertexts for conducting PFA. As k_j is fixed, the probability distribution of y_j also relates to distribution of c_j. With the collected ciphertext, the adversary can build the distribution of c_j, to retrieve information on y_j, eventually allowing the recovery of the key k_j. This analysis can be formally written as:

$$Pr(c_j) = Pr(y_j \oplus k_j) \tag{2}$$

For the j^{th} word of ciphertext, each possible value of c_j is denoted as $t, 0 \leq t < 2^b$. The adversary can collect N ciphertexts and calculate the appearance of t denoted as $Counts(t)$. $Counts(t)$ is a function to count the number of ciphertexts where $c_j = t$. Suppose $arg\_mincounts(t)$ and $arg\_maxcounts(t)$ are two respective functions to find the value of t with the minimal and maximal number of counts. The corresponding value of t is denoted as t_{min} and t_{max}, respectively, which can be computed as followings:

$$\begin{aligned} t_{min} &= arg\_mincounts(t) \triangleq \{t|\forall s : Counts(t) \leq Counts(s)\} \\ t_{max} &= arg\_maxcounts(t) \triangleq \{t|\forall s : Counts(t) \geq Counts(s)\} \end{aligned} \tag{3}$$

Then, three cryptanalysis strategies can be applied to recover the secret key.

Strategy 1: Exploiting t_{min}. Since the adversary can calculate the statistic distribution of each element in ciphertexts, he is aware of the value of t_{min}. He also knows v which is publicly known as the original value of the element in the S-box. If N is large enough, k_j can be directly deduced as:

$$k_j = v \oplus t_{min} \tag{4}$$

Strategy 2: Exploiting impossible values for $t \neq t_{min}$. For the other values t of c_j, where $t \neq t_{min}$, the adversary can use these values to eliminate impossible candidates of k_j, which can be denoted as

$$k_j \neq v \oplus t \tag{5}$$

Strategy 3: Exploiting t_{max}. The adversary can also try to find the value of t_{max} whose frequency is approaching 2^{1-b}. If the adversary knows v^*, *i.e.*, the faulty value of the element with the persistent fault, k_j can be easily computed as:

$$k_j = v^* \oplus t_{max} \tag{6}$$

All three strategies can be used in exploiting the fault for key recovery. However, based on the application scenario, one of the strategy might be better suited. Strategy 1 and 2 are accurate analyses. As long as the probability of t is nonzero, the value of $v \oplus t$ is the impossible candidate for k_j and can be eliminated. Both strategies require the value of v to be known.

Strategy 3 is a statistical analysis. Only when the total number of ciphertexts N is increased to a representative value, the probability of t_{max} (approaching to 2^{1-b}) can be obviously distinguished from other cases. As a result, $k_j = v^* \oplus t_{max}$ can be recovered. Strategy 3 requires the additional knowledge that the adversary should also know v^*, the value of the faulty element in the lookup table.

Algorithm 1 describes the pseudo code of persistent fault analysis on the last round key. The attack eliminates the impossible candidates for each key element until it identifies all L words of the last round key. In the attack, a two-dimensional array, $Counts[u][t], 0 \leq u \leq L$, $0 \leq t \leq 2^b$ is initialized to all zeroes. Then, the value of specific element of $Counts[u][t]$ is updated by counting the appearance of each ciphertext element. If the total number of counts for $Counts[u][t]$ is non-zero, then $(t \oplus v)$ can be discarded as an impossible candidate for k_u. Otherwise $(t \oplus v)$ can be kept as a possible candidate for k_u. In the end, only one non-zero value remains for one of the $Counts[u]$ at indice t', which reveals the value of $k_u = t' \oplus v$. Note that this analysis reveals all key elements in one enumeration of all N ciphertexts, which is quite simple and efficient.

Algorithm 1: Pseudo code of PFA on the last round of a general block cipher.

```
for $u = 0$; $u < L$; $u{+}{+}$ do
    for $t = 0$; $t < 2^b$; $t{+}{+}$ do
        Counts $[u][t]$=0;
    end
end
for $u = 0$; $u < L$; $u{+}{+}$ do
    for $n = 0$; $n < N$; $n{+}{+}$ do
        Counts $[u][c_{u,n}]$++ ;                    // $c_{u,n}$ is $c_u$ in the $n$-th ciphertext
    end
end
for $u = 0$; $u < L$; $u{+}{+}$ do
    for $t = 0$; $t < 2^b$; $t{+}{+}$ do
        if Counts $[u][t] > 0$ then
            Discard candidate $k_u = t \oplus v$;
        end
    end
end
```

3.4 Complexity analysis

Recall N is the total number of ciphertexts that are available and n is the number of ciphertexts already used for analysis. For any c_j, let θ_n denote the "average" number of different outputs of table lookups using n ciphertexts. When n is small, the lookup outputs might be pairwisely different, so $\theta_n = n$. When n is large, θ_n will converge to the value $\eta = 2^b - 1$, assuming single fault is injected. The adjective "average" means that the analysis of θ_n is conducted in a probabilistic way, which will give the estimated value of N for finding out the only impossible value v for c_j.

Let $\Delta\theta_n = \theta_n - \theta_{n-1}$. When $n = 0$, $\theta_0 = 0$. When $n = 1$, $\theta_1 = 1$ and $\Delta\theta_1 = 1$. For $n = 2$, we have $\Delta\theta_2 = (\eta - \theta_1)/\eta$, assuming that the remaining possible values of the lookup output θ_1 will satisfy the uniform distribution. Then, we can easily deduce the following:

$$\begin{aligned} &n = 2, \Delta\theta_2 = (\eta - \theta_1)/\eta, \theta_2 = \theta_1 + \Delta\theta_2 = 1 + (\eta - \theta_1)/\eta \\ &n = 3, \Delta\theta_3 = (\eta - \theta_2)/\eta, \theta_3 = \theta_2 + \Delta\theta_3 = 1 + (\eta - \theta_1)/\eta + ((\eta - \theta_1)/\eta)^2 \\ &\dots \end{aligned} \tag{7}$$

According to the observations in Eq.(7), we can infer the formula of θ_n by using the mathematical induction of geometric progression. Thus to compute θ_n, we have:

$$\theta_n = \frac{1 - q^n}{1 - q}, \quad \text{where} \quad q = \frac{\eta - 1}{\eta} \tag{8}$$

The value of v can never be output in the encryption. Let ϵ_n denote the average number of the remaining possible key candidates associated with c_j using n ciphertexts. We have $\epsilon_n = 2^b - \theta_n = 2^b - \frac{1-q^n}{1-q}$. When n increases and θ_n is approaching $\eta = 2^b - 1$, η impossible candidates of k_j, that is, $v \oplus t$, can be gradually eliminated. Finally, only the correct candidate of k_j, *i.e.*, $v \oplus t_{min}$, remains.

Fig. 2 describes the relationship between $\log_2 \epsilon_n$ and n where $b = 8$ (for AES). We can see that $\log_2 \epsilon_n$ decreases with the increase of n. When n is greater than 1400, $\log_2 \epsilon_n$ can be reduced to 1, which implies that two possible key candidate exist. In fact, when $n \geq 2000$, $\log_2 \epsilon_n$ can be reduced to 0, which gives the unique key candidate. Note that Fig.2 only gives a theoretical estimation of $\log_2 \epsilon_n$ for better understanding on the limit of N. In practice, the adversary needs more ciphertexts to statistically get the correct key.

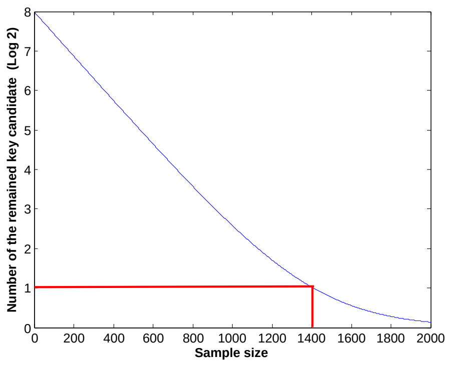


Figure 2: Relationship between $\log_2 \epsilon_n$ and n for one element of master key where $b = 8$.

The estimation of n (when $\log_2 \epsilon_n$ is reduce to 1) is popularly known as the coupon collector's problem [BHS94] where it needs $255 * (1/1 + 1/2 + ... + 1/255) \simeq 1561$ tries to collect 255 coupons. Similarly, it roughly requires 1561 trials to see all 255 occurring ciphertexts at least once.If the cipher is composed of smaller S-boxes (like $b = 4$), the analysis needs fewer ciphertext ($n \simeq 49$). PFA can be applied to DES like ciphers where S-boxes are not bijective. However, if the S-boxes are not identical, only a part of key bits will be recovered with one fault. PFA is currently applied on the last round, which in case of AES, extracts 16 key bytes for S-box implementation with one injection. PFA could be on the last but one round at the cost of complex analysis as fault might affect both last two rounds.

3.5 Comparison with other fault analysis

Most fault analyses, like DFA, AFA, etc., are differential in nature. They exploit difference between correct and faulty ciphertexts, produced from a fixed plaintext.

Other fault analysis like SFA [Riv09, FJLT13] or FSA [LSG+10], require extra requirements like biased fault or side-channel information. SFA needs multiple biased or constant faults, while PFA needs a single random fault (in ROM). Depending on attackers' capability, one of PFA/SFA might be preferred. The advantages and disadvantage of PFA against other fault analysis techniques can be summarized as follows:

3.5.1 Advantages

- The attack is not differential in nature and thus the control over the plaintext is not required. It exploits the statistical properties, which can be built directly upon faulty ciphertexts.
- The adversary does not necessarily need live synchronisation, to inject a fault at a sensitive moment. The adversary can inject a persistent fault beforehand and wait for the victim to start encryption. Such setting implies that remote, powerful but slow injection techniques like rowhammer can be well suited.
- The fault model remains relaxed compared to statistical attacks like SFA and FSA. While SFA assumes biased fault injection, FSA needs side-channel information for the analysis. PFA is built upon ciphertexts with a random fault model only without any biases. No side-channel information is required.
- As shown later, PFA can also be applied in multiple fault setting.
- By its nature, PFA can bypass some redundancy based countermeasures.
- Some circuits deploy fault injection sensors to detect injection attempts. For energy saving, these sensors are only operated in the so-called sensitive mode. An adversary can always inject the persistent fault before the victim is switched to the sensitive mode, rendering the protection ineffective.

3.5.2 Disadvantages

- As the analysis technique is statistical, it needs much higher number of ciphertexts as compared to DFA, which in some cases can be as low as 1 or 2 ciphertext pair.
- Persistent faults can be detected by some built-in health test mechanism.

4 PFA with Multiple Faults

Unlike other fault analysis techniques, PFA can also be applied in a multiple fault injection setting. By multiple faults, we mean the adversary modifies several elements in a persistent manner. If we take the previous example, the adversary modifies several elements of the

S-box. Analysis of multiple injection is becoming more realistic when the technology node is shrinking much faster than the fault injection capability. Thus, the adversary is likely to affect a larger area, like multiple memory locations [SHP10].

Let us assume that the adversary injects faults into λ elements of the S-box. The fault model remains similar to the previous case. There are at least $(2^b - \lambda)$ possible output values from S-box. For simplicity, we do not consider the linear layer in our equations. Thus, we have $(2^b - \lambda)$ candidate values for each ciphertext word c_j. Suppose S_v denotes the set of the corrupted S-box output elements $(v_0, v_1, \cdots v_{\lambda-1})$. For each v_i in S_v,

$$Counts(t) = 0, \quad k_j = v_i \oplus t, \quad (0 \leq \ i < \lambda, \quad 0 \leq j < L) \tag{9}$$

In the attack, for each possible value t in each ciphertext element, we calculate $Counts(t)$ for 2^b candidates of t. If $Counts(t) = 0$, we can keep $v_i \oplus t$ as candidates of k_j. If $Counts(t) \neq 0$, we can discard $v_i \oplus t$. With enough number of ciphertexts, at most λ candidates of k_j can be kept after PFA on each element of ciphertext. Then, the maximal residual entropy of the last round key can be calculated as $L \times \log_2 \lambda$ where L is the number of ciphertext words, *i.e.*, for AES $L = 16$, 16 words of one byte ($b = 8$) each.

Fig. 3 shows the relationship between the average residual key entropy corresponding to the sample size N and the number of multiple faults λ. We set $1 \leq \lambda \leq 16$ (no. of persistent faults) and $1 \leq N \leq 5000$ (no. of ciphertexts). For each value of λ, we repeat PFA 1000 times on different datasets, and the final result is averaged over all experiments. We increase the number of samples one by one and calculate the residual entropy of the last round key as a function on N.

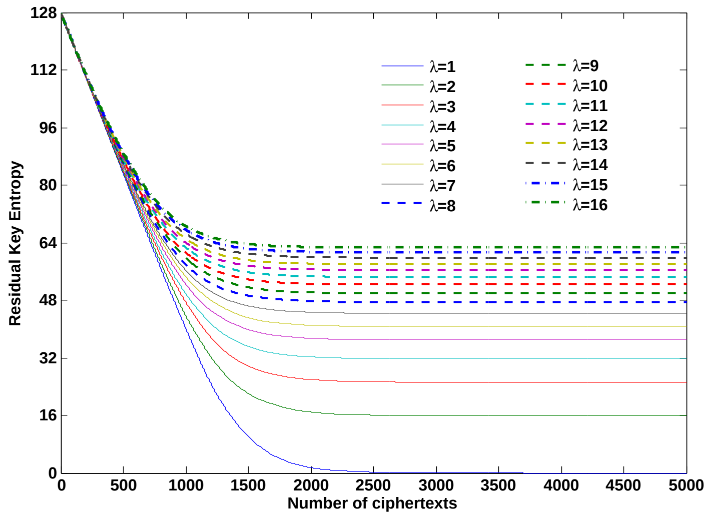


Figure 3: Relationship among the residual key entropy, N and λ.

From Fig. 3, we can see that, for each value of λ, the residual key entropy of the block cipher with $b = 8$ can be at most reduced to $16 \times \log_2 \lambda$ after PFA on the last round. Also, at $N = 2000$ the attack saturates and the key entropy cannot be further reduced without extra information. Normally, an adversary can try brute force of the remaining candidates. If the key entropy is beyond brute force search, the adversary can extend PFA to the last but one round and further reduce the key entropy.

5 Validation of PFA on AES-128

5.1 AES implementation

In this section, we check validation of our attack on AES-128. AES-128 [AES] is a symmetric block cipher algorithm with a block size of 128 bits. Suppose the i-th round is denoted as R_i. The initial nine rounds $R_i, 1 \leq i \leq 9$ comprise of four major operations: SubByte(SB), ShiftRow(SR), MixColumn(MC), and AddroundKey(AK). There is an additional key whitening before R_1 and no MixColumn operation in R_{10}. The SB function is the only non-linear operation of the block cipher consisting of an substitution box (S-box) lookup applied to the state. AES-128 uses an 8×8 S-box, *i.e.*, 256 bytes. Moreover, the operations in AES-128 are byte oriented which makes $b = 8$ or $\eta = 2^b - 1 = 255$.

Three exemplary AES-128 implementations are considered. All implementation are based on look-up tables which are stored in the memory. An adversary can inject faults in the memory to corrupt a particular entry of the look-up table, through various available injection means. The details of the tested AES-128 implementations are:

I1: S-box Implementation has an S-box lookup table in the encryption and an inverse table S^{-1} in the decryption. There are 256 elements in both S and S^{-1} tables and are used in all ten rounds.

I2: T-table Implementation optimises performance by merging SB, SR and MC operations in one T-table lookup. There are four T tables in the encryption, denoted as T_0, T_1, T_2, T_3. Each T_i is an 8×32 lookup *i.e.* 256 elements of 32 bits. In each round $R_i (1 \leq i \leq 10)$, each table of T_0, T_1, T_2, T_3 is accessed for four times. In the decryption, the four inverse lookup tables are denoted as $T_0^{-1}, T_1^{-1}, T_2^{-1}, T_3^{-1}$. Since R_{10} has no MC operation, the required output is extracted from the same T-tables to optimise memory requirements.

I3: T-table Implementation is similar to I2. Each of T_0, T_1, T_2, T_3 is accessed for four times in the first nine rounds. Four additional tables T'_0, T'_1, T'_2, T'_3 are used in R_{10}. Each element of T'_i also has 32 bits. However three bytes of the element in T'_i are just zeroes, while the only non-zero byte has the same value as that in S.

Table 1 summarises the types of implementations that are offered to apply the AES using S-box and T-box with the lookups in each round and table size. Note that I3 is also a realistic implementation whose assembly code could be found in the file `rijndeal-amd64.S` of the shared library `Libgcrypt 1.6.3`. Similar to Openssl, Libgcrypt is also a well-known cryptographic library which provides numerous cryptographic building blocks.

Table 1: Different implementations of AES-128 encryptions.

Type	Lookups in each round	Table size	Notes
I1	R_{1-10}: S	S:256B	Typical S-box implementation
I2	R_{1-10}: T_0, T_1, T_2, T_3	T_i:1KB	Typical T-box implementation
I3	R_{1-9}: T_0, T_1, T_2, T_3 R_{10}: T'_0, T'_1, T'_2, T'_3	T_i:1KB T'_i:1KB	Code can be found in rijndeal-amd64.S in the library Libgcrypt 1.6.3

I3 is analyzed in the next section using rowhammer attacks on modern processors. We test I1 and I2 on Virtex-5 FPGA (xc5vlx50). The area results of I1 and I2 are summarised in Table 2. Since there are 16 S-box and 16 S^{-1}-box consumption for each complete AES, each BRAM (RAMB36) can accommodate 4 S-box or S^{-1}-box. The BRAM cost for S-box based AES is 8. By contrast, the BRAM cost for T-box based AES is 4.

For the validation of the attack, we use *data2mem* tool [dat09] provided by Xilinx. It can update the BRAM content without the need of re-flashing a new bitstream. Thus, it allows us to emulate the persistent fault injection.

Table 2: Cost of S-box and T-box AES implementations.

AES style	RAMB36	Slice LUTs	Slice Registers	Occupied Slices
S-box	8	2630	2469	2131
T-box	4	4526	2492	1370

5.2 PFA on vulnerable S-box implementation (I1)

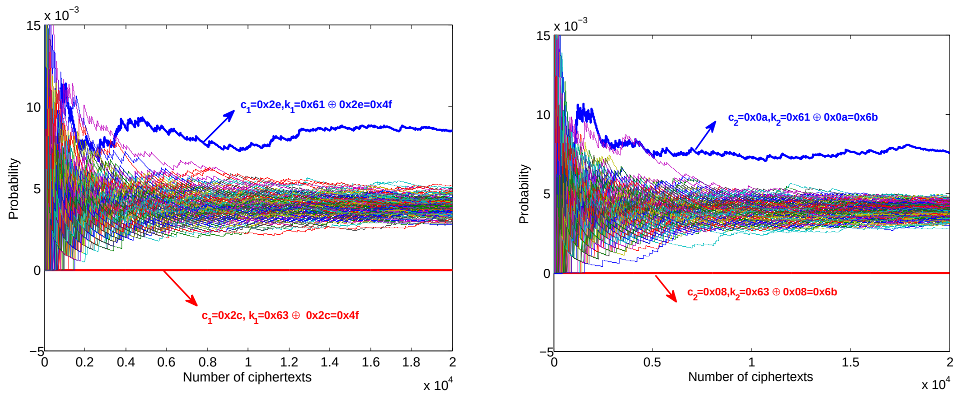
It is assumed that an adversary effects AES S-box with a fault injected on one element to collide with another existing element, resulting in only 255 distinct elements. The correct value victim v of the e^{th} element does not appear instead the effected faulty value v^* appears twice. All other elements in the S-box remain correct and appear with a probability of $\frac{1}{256}$. When only one fault is injected to the table, the discrete probability distribution function of y can be described as follows:

$$Pr(y = v) = 0; \quad Pr(y = v^*) = \frac{2}{256}; \quad Pr(y \neq v \wedge y \neq v^*) = \frac{1}{256} \tag{10}$$

The goal for adversary $\mathcal{A}$ is to extract the last round key K^{10}. Here Algorithm 1 can be directly applied to handle this situation where $b = 8$. However, the number of N samples needs to be carefully chosen in order to successfully launch the attack based on the theoretical estimation.

5.2.1 Attack result

Fig.4 shows the result of an exemplary PFA on AES S-box implementation. In the attack, the first element of S-box is injected with the persistent fault, $v = 0x63$. After the fault injection, only the 7-th bit of v is flipped from one to zero, $v^* = 0x61$. In Fig.4, 2×10^4 ciphertexts are collected. For each ciphertext byte, for example, c_1 in Fig.4a and c_2 in Fig.4b, the probability for each value is plotted as one curve, which is calculated as the counts of appearances for that specific value divided by the number of ciphertexts already used in the analysis. In both subfigures, two curves are obviously distinct from the rest. The red curve at the bottom is for t_{min}, which never appears in c_j. The blue curve on the top is for t_{max}, whose probability is converged to $\frac{2}{256}$ and significantly larger than that of other values. Since $c_1 = 0\text{x}2e$, k_1 can be computed as $k_1 = 0\text{x}61 \oplus 0\text{x}2e = 0\text{x}4f$. Similarly, $k_2 = 0\text{x}61 \oplus 0\text{x}0a = 0\text{x}6b$. The attack values of k_1 and k_2 are the same as the one in K^{10}.



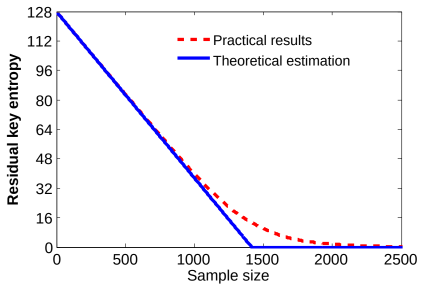
(a) Extract k_1 using the distribution of c_1 (b) Extract k_2 using the distribution of c_2

Figure 4: Exemplary PFA on AES-128 using distributions of ciphertext values.

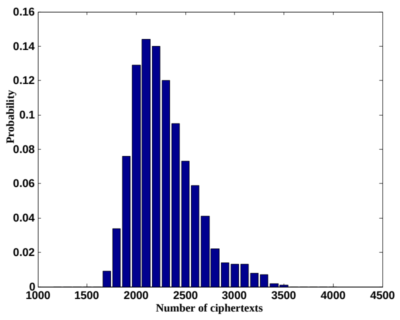
5.2.2 Residual key entropy for different sample size

Let $\phi_t(n)$ denote the theoretical estimation of the residual key entropy for K^{10} when n ciphertexts are used for analysis. Note that the similar analysis can be applied to all sixteen key bytes simultaneously. $\phi_t(n)$ can be approximately calculated as $16 \times \log_2 \epsilon_n$, where $\epsilon_n = 256 - \frac{1-q^n}{1-q}$ and $q = \frac{254}{255}$.

Let $\phi(n)$ denote the actual residual key entropy when n ciphertexts are used for analysis. The value of $\phi(n)$ can be calculated according to Algorithm 1, assuming the attacks on each byte k_j are equivalent. Fig. 5a shows how $\phi(n)$ and $\phi_t(n)$ decrease when n is increased. In Fig. 5a, $\phi(n)$ is very close to the theoretical value of $\phi_t(n)$. When $n \geq 1240$, $\phi_t(n) \leq 16$. When $n \geq 1360$, $\phi(n) \leq 16$, which implies that there are at most 2^{16} candidates of K^{10}. When $n \geq 1405$, $\phi_t(n) \leq 1$. And when $n \geq 2148$, $\phi(n) \leq 1$, which implies that there are at most two candidates for the full K^{10}.



(a) $\phi(n)$ v.s. $\phi_t(n)$.



(b) Distributions of N_f

Figure 5: Attack result of PFA on unprotected AES.

5.2.3 Sample size distributions for full key recovery

In order to guarantee the success rate of our PFA method, the fault attacks are conducted again ξ times with the random plaintext. For each attack, we increase the number of ciphertexts n in the analysis, until all the sixteen key bytes are recovered. Suppose N_f denotes the number of ciphertexts that is required when the adversary $\mathcal{A}$ can successfully extract all the sixteen bytes of AES for the first time in one specific attack.

Fig.5b describes the distributions of N_f for $\xi = 1000$ attacks. It is clear that $1678 \leq N_f \leq 3504$, it depicts that at least 1678 and at most 3504 samples are required to recover the full master key. The average value of N_f is 2281.

6 Defeating Fault Attack Countermeasures with PFA

In this section, we enhance the PFA to AES-128 protected against fault attacks.

6.1 Countermeasures against fault attacks

Dual Modular Redundancy (DMR) is a mechanism of using two redundant modules to prevent fault attacks [Joy12]. DMR has characteristics of reliability, security and also offers robustness to the systems in order to detect the error. This countermeasure is readily adopted in commercial solutions due to its properties thus enhances the reliability.

As shown in Fig.6, there are two modules in the DMR scheme: Module 1 and Module 2. If both modules are performing encryption, the countermeasure is named as *redundant* *encryption based DMR* (REDMR) where Module 2 is functionally equivalent to Module 1. Provided that the resultant ciphertexts of two modules are the same ($C' = C''$), the security check for REDMR is passed. Thus, the ciphertext is considered true and can be sent to display the output. REDMR passes the security check and is considered secure against single fault injection. In order to defeat this countermeasure, one has to inject either same fault in both the modules (two fault injection are required), or inject a fault in one module and bypass the security check.

An alternative strategy is adopted to make the same fault in two modules harder. If Module 1 is an encryption module and Module 2 is decrypting the ciphertext from Module 1, the corresponding countermeasure is named as *inversive decryption based DMR* (IDDMR). If the decrypted plaintext from the module is the same as the original plaintext ($P' = P$), the ciphertext of Module 1 is considered as true and can be sent to display the output. Since both the modules are performing different operations and have different architectures, injecting complementary fault is harder. Next, we focus on IDDMR, the stronger of the two countermeasures. Same analysis also applies to REDMR.

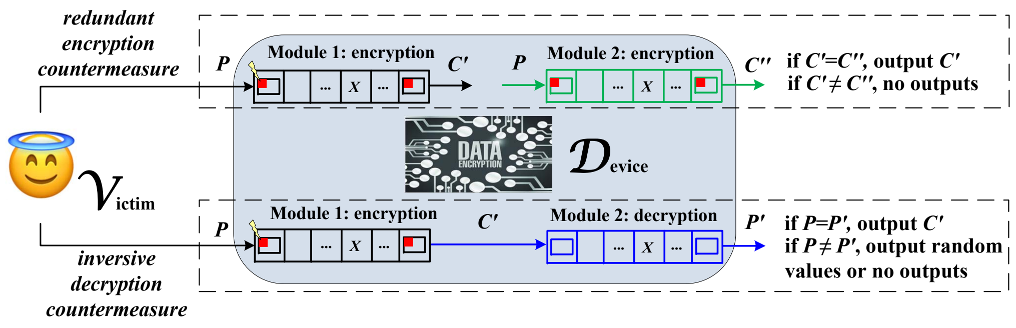


Figure 6: Countermeasures against fault attacks: REDMR and IDDMR.

Based on the reaction to failed security check, three countermeasure can be classified.

C1: No ciphertext output (NCO) . If $P' \neq P$ is detected, the victim $\mathcal{V}$ will not display the incorrect ciphertext C'.

C2: Zero value output(ZVO) . If $P' \neq P$ is detected, the victim $\mathcal{V}$ will display the output that is a ciphertext $C' = 0$.

C3: Random ciphertext output(RCO) . A ciphertext with total random values will appear at output when $P' \neq P$ is detected. The major benefit of this scheme is to embed the incorrect ciphertexts into a large pool of randomized ciphertexts, resulting in the adversary's difficulty of directly differentiating the fault leakages. Different from NCO and ZCO, $\mathcal{A}$ can not distinguish the correct encryption from faulty encryption.

Note that for REDMR, if both the modules use shared memory *i.e.* common look-up tables, all three countermeasures will fail to detect a fault in lookup tables, that is target of PFA. In the following, we do not consider this case but a stronger implementation where each module has independent memory, for instance, IDDMR.

6.2 PFA on S-box (I1) with NCO and ZCO

With NCO and ZCO, only a fraction of the N ciphertexts are available while the rest of ciphertexts are suppressed by either no ciphertext output (NCO) or by all zero values

(ZCO). This is because the injected faults in the lookup table are used in some intermediate computations in the encryption, and the incorrect ciphertext is used in the decryption, which further leads in the faulty output P'. Thus, the adversary $\mathcal{A}$ has to perform further more encryptions in order to have significant number of ciphertexts which are not suppressed. However, the analysing methodology remains exactly the same as in the unprotected case, once enough number of ciphertexts are available. Each AES encryption involves 160 S-box calls (16 in each of 10 rounds). Note that the 40 lookups in the key schedule are not considered. The probability p that one plaintext can bypass IDDMR, *i.e.*, the probability that all 160 S-box lookups do not access the faulty element in the S-box table, can be calculated as

$$p = (1 - \frac{1}{256})^{160} \approx 0.5346 \tag{11}$$

Thus, only $p \times N$ ciphertexts can be used for the attack. In this case, the attacker would need around $N/p \approx 1.8706 \times N$ encryptions, instead of N encryptions to perform full key recovery.

To investigate the case of PFA on NCO/ZCO-based IDDMR, we repeat the attack for $\xi = 1000$ times. Fig.7 describes the distribution of N_f. The statistic shows that $3042 \leq N_f \leq 7141$, which means that at least 3042, at most 7141 ciphertexts are required to recover the full master key. The average value of N_f is 4234. In the attack, if we set $n > 7200$, the success rate is 100%.

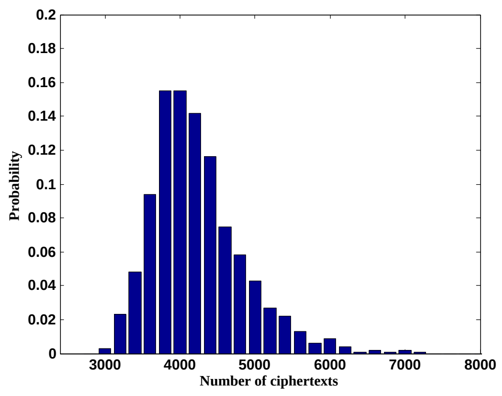


Figure 7: Distribution of N_f for PFA on AES with NCO/ZCO.

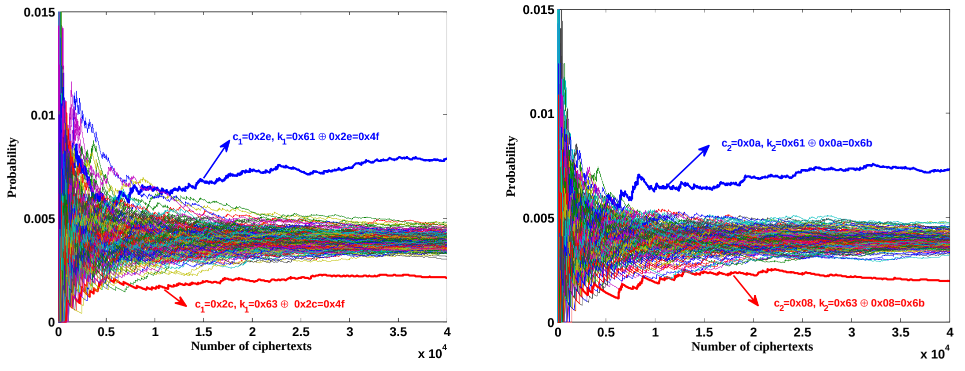
6.3 PFA on S-box (I1) with RCO

In contrary to Section 6.2, in the presence of RCO, the adversary cannot distinguish between correct ciphertext and random ciphertext. Each ciphertext byte can take all possible 256 values. Strategy 2, which depends on the impossible value of ciphertexts, can not defeat this countermeasure scheme. Whether Strategy 1 and 3 (exploiting t_{min} or t_{max}) can be applied or not depends upon the probability distribution for different values of the ciphertext byte. Since all the ciphertexts are generated either by a correct output when $P' = P$ or by a random output when $P' \neq P$, the discrete probability distribution function for the ciphertext byte y can be computed in two parts accordingly. One is the output of the S-box whose distribution is already described in Eq.(2). However, the constraint that the ciphertext is correct should be added, whose probability is given in Eq.(11). The other is the random output which satisfies the uniform distribution. The specific distribution is listed in Eq.(12) where p is the probability that one encryption does not access to any faulty element in S-box.

$$\begin{aligned} Pr(y = v) &= 0 \times p + \frac{1}{256} \times (1-p) = \frac{0.4654}{256} \\ Pr(y = v^*) &= \frac{2}{256} \times p + \frac{1}{256} \times (1-p) = \frac{1.5346}{256} \\ Pr(y \neq v \wedge y \neq v^*) &= \frac{1}{256} \times p + \frac{1}{256} \times (1-p) = \frac{1}{256} \end{aligned} \tag{12}$$

From Eq.(12), $y = v$ is still with the minimal probability and $y = v^*$ is with the maximal probability. In this case, both Strategy 1 and Strategy 3 can be adopted to extract the secret key, with updated specific probability value $Pr(y = v)$ and $Pr(y = v^*)$. Fig. 8 clearly demonstrates the probability distribution of two illustrative ciphertext bytes, c_1 in Fig.8a and c_2 in Fig.8b, which are the first and the second byte of AES ciphertext for S-box implementation protected with RCO. For each ciphertext byte, the probability for each value in corresponding to the increase of N is plotted as one curve, which is calculated as the counts of appearances for that specific value divided by the number of ciphertexts already used in the analysis.

In Fig. 8, 4×10^4 ciphertexts under the IDDMR are collected. The red curve at the bottom is for t_{min}, whose probability is converged to $\frac{0.4654}{256}$ and significantly smaller than that of other values. The blue curve on the top is for t_{max}, whose probability is converged to $\frac{1.5346}{256}$ and obviously larger than that of other values. In both subfigures, two curves are obviously distinctive from the rest.



(a) Extract k_1 using the distribution of c_1 (b) Extract k_2 using the distribution of c_2

Figure 8: Attack result on AES-128 S-box implementation with RCO.

Algorithm 2 describes the analysis on AES-128 (Implementation I1) which extracts K^{10} when IDDMR with RCO is applied. The attack sets two threshold values τ_1 and τ_2 to filter the correct key candidates. The choice of τ_1 and τ_2 is based on the empirical experience, in order to distinguish the two biased bytes from other 254 unbiased bytes. For the sake of simplicity, τ_1 is set as $0.9 \times \frac{1.5346}{256}$ and τ_2 is set as $1.1 \times \frac{0.4654}{256}$, in order for identifying the key candidate with maximal and minimal probability, respectively.

Similar to Algorithm 1, the two-dimensional $Counts[u][t]$ is updated by counting the appearance of each ciphertext byte. For Strategy 1, if the total number of counts for $Counts[u][t]$ divided by N is smaller than τ_1, $(t + v)$ can be kept as the possible candidate for k_u and inserted into a candidate set $\Phi_{u,1}$. For Strategy 2, If $Counts[u][t]$ divided by N is larger than τ_2, $(t + v^*)$ can be kept and inserted into a candidate set $\Phi_{u,2}$.

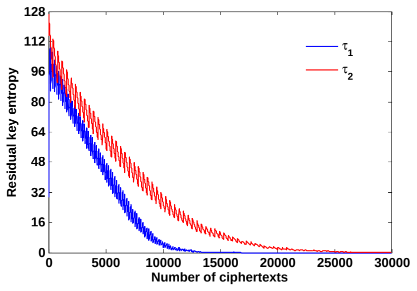
According to Algorithm 2, we conduct two experiments to estimate $\phi(n)$, the residual key entropy, by using τ_1 and τ_2. Let $\phi_{u,1}$, $\phi_{u,2}$ denote the residual key entropy ($\log_2$ based)

Algorithm 2: Attack on AES-128 to extract the round key in R_{10} under IDDMR.

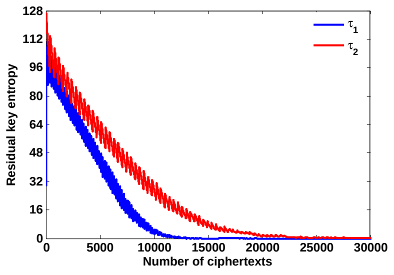
```
for $u = 0; u < 16; u{+}{+}$ do
    for $t = 0; t < 256; t{+}{+}$ do
        Counts $[u][t]$=0;
    end
end
for $u = 0; u < 16; u{+}{+}$ do
    for $n = 0; n < N; n{+}{+}$ do
        Counts $[u][c_{u,n}]$++; ;                    // $c_{u,n}$ is $c_u$ in the $n$-th ciphertext
    end
end
for $u = 0; u < 16; u{+}{+}$ do
    for $t = 0; t < 256; t{+}{+}$ do
        if $\frac{\text{Counts}[u][t]}{N} < \tau_1$ then
            Insert $t + v$ into $\Phi_{u,1}$; ;       // Strategy 1
        end
        if $\frac{\text{Counts}[u][t]}{N} > \tau_2$ then
            Insert $t + v^*$ into $\Phi_{u,2}$; ;     // Strategy 2
        end
    end
end
```

of $\Phi_{u,1}$, $\Phi_{u,2}$ respectively. $\Phi_{u,1}$ and $\Phi_{u,2}$ are the residual key search space when analyzing the u-th byte of the last subkey in R_{10} merely with only τ_1 or τ_2, respectively.

In the first experiment, the residual key entropy $\phi_1(n)$ is calculated as the sum of all $\phi_{u,1}$. In the second experiment, $\phi_2(n)$ is the sum of all $\phi_{u,2}$. In each experiment, 1000 attacks are performed. $\phi_1(n)$ and $\phi_2(n)$ are depicted in Fig.9a respectively, showing how the residual key entropies change with the number of ciphertexts used in analysis. We can see that $\phi_1(n)$ in PFA using τ_1 is much less than that using τ_2. On average, the required sample size is about 9280, 12660 for using different thresholds (τ_1, τ_2), respectively, when the residual key entropy is less than 16.



(a) S-box implementation with RCO



(b) T-box implementation with RCO

Figure 9: $\phi_1(n)$, $\phi_2(n)$ v.s. the number of samples, when different thresholds (τ_1, τ_2) are used in PFA on IDDMR countermeasures. $\phi_1(n) = \sum_{u=0}^{15} \phi_{u,1}$, $\phi_2(n) = \sum_{u=0}^{15} \phi_{u,2}$.

6.4 PFA on T-tables (I2) with RCO

In this section, we enhance our fault attack to T-tables based implementation (I2) protected with additional RCO. Even though the use of T-tables based implementation is limited and discouraged due to huge memory requirement and vulnerability to cache attacks, it is still used in some libraries. In comparison to the S-box implementation, in this type of implementation, the attack requires more effort in fault injection. For T-tables (I2), each table T_i is accessed for 4 times in each AES round and 40 times in one encryption. Let k_j

denote the j^{th} byte of the last round key K^{10}, $0 \leq j \leq 15$.

In case of a straight forward attack, the adversary is required to modify four entries, one in each T-table to make sure the correspondence of single fault injection in S-box implementation. For T-tables stored in memory, four byte faults are injected into T_0, T_1, T_2, T_3 simultaneously. For instance, the adversary can modify the following four elements: the first element of T_0 (the third byte, $0xc66363a5 \rightarrow 0xc66361a5$), the first element of T_1 (the fourth byte, $0xa5c66363 \rightarrow 0xa5c66361$), the first element of T_2 (the first byte, $0x63a5c663 \rightarrow 0x61a5c663$), the first element of T_3 (the second byte, $0x6363a5c6 \rightarrow 0x6361a5c6$). Then, p'', the probability that one plaintext can bypass IDDMR, is still $(1 - \frac{1}{256})^{160} \approx 0.5346$.

In our attack, we set $\tau_1 = 0.9 \times \frac{1.5346}{256}$ and $\tau_2 = 1.1 \times \frac{0.5346}{256}$. Fig. 9b shows the relationship between $\phi(n)$ and the sample size n. It is observed that in this scenario, $\phi(n)$ in PFA using τ_1 is much less than using τ_2. To reduce the residual key entropy to be less than 16, the average sample size is 8840 and 12870 for τ_1 and τ_2, respectively.

6.5 Discussion

It is important to mention that the 40 S-box lookups in the key schedule of AES do not affect our analysis that much. In the scenario of REDMR or IDDMR, the probability of outputting a correct ciphertext will be calculated as $(1 - \frac{1}{256})^{200}$. The analysis will remain as the same and quite straightforward. In addition, considering the multiple faults on IDDMR, it is also possible that two faults are injected to S and S^{-1} separately. Intuitively, the fault in S^{-1} might cancel some faulty propagation during the reversible decryption process, resulting more number of ineffective ciphertexts for analysis. However, our preliminary analysis shows the second fault to S^{-1} does not improve our PFA. It will require the same number of ciphertexts as in PFA on the normal IDDMR. It should be noted that, to further reduce the data complexity and the key search space, our PFA can be extended to more rounds besides the last round, *e.g.*, the 9-th round of AES-128.

7 Case Study: Rowhammer-based PFA on T-box (I3)

Cryptographic libraries are vital to provide the security to the applications. Cryptographic primitives frequently used are hardcoded in the *shared libraries*, which are assembled among multiple processes. In addition to that, it simultaneously prevents the performance downgrading. Cryptographic libraries such as Libgcrypt [lib], OpenSSL, PolarSSL etc. are some common examples. In this section, we experiment persistent bit fault that is injected into the T-tables of the binary file of Libgcrypt(Implementation I3), which is compiled and deployed as a shared library.

7.1 Attack overview

The overview of our attack on shared libraries is illustrated in Fig.10. The fault is injected in a setting where two processes running on the same computer that access the Libgcrypt in the memory. The process $\mathcal{A}$ is the malicious adversary and the other process $\mathcal{V}$ is the victim. Both are user privilege processes at the security level of ring-3. In our experiments, we further hardened `aes_encrypt` by REDMR. The choice for REDMR is motivated by use of the `encrypt` function only. In fact, the adversary $\mathcal{A}$ can prepare an attacking environment before $\mathcal{V}$'s encryption. $\mathcal{A}$ can search for the entire T-table (marked as green in Fig.10) in his own virtual address space (marked as blue). The size of entire T-table is 8KB which includes all T_i and T'_i ($0 \leq i \leq 3$) and crosses three physical pages. Note that T_i and T'_i may not be aligned to the starting address of the physical page that they reside. The offset of T'_i to the resident page can be inferred from the layout of the *.so* binary file.

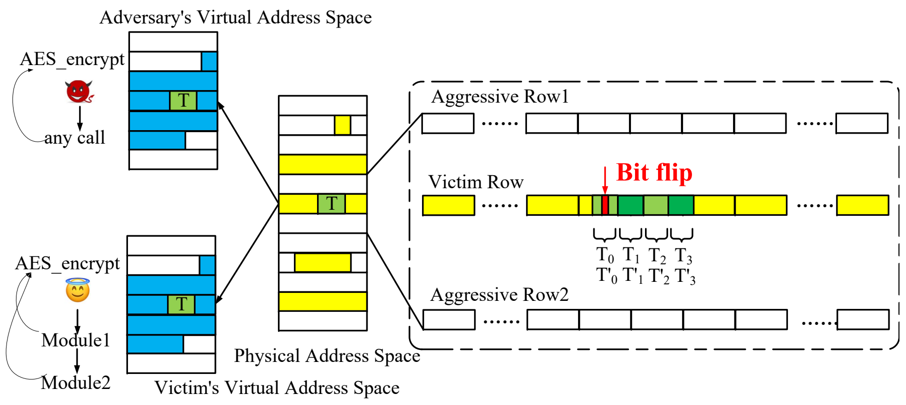


Figure 10: Overview of the rowhammer-based persistent fault attacks on shared libraries.

$\mathcal{A}$ can flip one bit in one of the T-tables (illustrated as T'_0 in Fig.10) in Libgcrypt 1.6.3 with a very simple hammering technique.

7.2 Attack procedure

For fault injection by rowhammer technique, the adversary $\mathcal{A}$ performs four distinct steps: profiling, allocation, positioning and hammering, which are described as below:

Step1: Profiling The adversary profiles the DRAM when the victim process is not running. Profiling helps in determining the vulnerable rows. The outcome of the profiling step is a dictionary of bit flips information for each physical row, for example, the rows which may contain bit flips after the hammering, and the location of those flips in the specific row. Note that this dictionary is a hardware profile for the physical memory, and credible to resist any alterations for the software changes.

Step2: Allocation In this step, the adversary aims to allocate T'_i into the vulnerable rows. Once the shared library is loaded, the OS allocates it to a random location in the page cache based on current memory usage. So in order to steer the victim towards targeted vulnerable location, several attempts are made to try different locations, as the shared library is reloaded to a new address after being evicted from memory.

Step3: Positioning In the positioning step, the adversary $\mathcal{A}$ tries to position virtual pages that are mapped to adjacent aggressive rows for next hammering. $\mathcal{A}$ first determines the row in which the victim library is located using certain highest bits (*i.e*, the row ID) of its physical address. Then he calculates the highest bits of adjacent rows and obtain the range of physical addresses that are located on adjacent rows. After that, he keeps spraying blank pages to the memory with the function *mmap* and checks if the physical address of a specific blank page is within in the address range of adjacent rows or not. If it is, he can keep the blank page, otherwise just drop it. After the number of blank pages in the adjacent rows are reached up to a threshold, the spraying is halted and those virtual addresses of blank pages in two adjacent rows are pushed into two sets VA_1 and VA_2, respectively. These two virtual address sets are used for hammering in the next step.

Step4: Hammering In this step, the adversary can access pages that are saved in VA_1 and VA_2 with high frequency. Once it results the bit flips in one of T'_i, hammering is interrupted. $\mathcal{A}$ then waits for $\mathcal{V}$ to start the encryption and observe the ciphertext. If there are no flips, the attack should be started from Step 2 again.

```
000d6710h: C6 63 63 A5 63 00 00 00 F8 7C 7C 84 7C 00 00 00 ;
000d6720h: EE 77 77 99 77 00 00 00 F6 7B 7B 8D 7B 00 00 00 ;
000d6730h: FF F2 F2 0D F2 00 00 00 D6 6B 6B BD 6B 00 00 00 ;
000d6740h: DE 6F 6F B1 6F 00 00 00 91 C5 C5 54 C5 00 00 00 ;
000d6750h: 60 30 30 50 30 00 00 00 02 01 01 03 01 00 00 00 ;
000d6760h: CE 67 67 A9 67 00 00 00 56 2B 2B 7D 2B 00 00 00 ;
000d6770h: E7 FE FE 19 FE 00 00 00 B5 D7 D7 62 D7 00 00 00 ;
000d6780h: 4D AB AB E6 AB 00 00 00 EC 76 76 9A 76 00 00 00 ;
```

Figure 11: The the first 16 elements of T_0 and T_0' in the binary file of Libgcrypt-1.6.3.

The adversary might have to repeat the above process several times to achieve desirable faults in T_i' which are accessed by the victim. Once the victim $\mathcal{V}$ starts the encryption, the adversary collects N ciphertexts. Each time the adversary $\mathcal{A}$ can only inject one bit to one table, therefore only four keys bytes can be extracted. The fault injection should be repeated for different tables T_0', T_1', T_2', T_3'.

7.3 Experiment results

7.3.1 Setup

Our attack is conducted on the Lenovo ThinkPad x230 laptop. The CPU is Intel(R) Core(TM) i5-3320M at 2.60GHz. It contains two Samsung DDR3 modules, each with 2GB at 1333MHz . The Linux OS is Ubuntu 12.04 LTS with the kernel version of 3.2.0-79 generic. The target is Libgcrypt v1.6.3. The compiler for building the shared library is gcc 4.6.3 with the default optimization setting.

In Libgcrypt 1.6.3, the size of the binary file is 2853 KB where the first element of AES T-table T_0 starts at the offset 000d6710h. The partial layout of the first 16 elements of T_0 and T_0' is shown in Fig.11. Every element (4 bytes) of T_0 is followed by the corresponding element of T_0' (also 4 bytes, where only one byte is non-zero). Note that T_i' is actually a copy of the compact S-box and will only be accessed during the last round R_{10}.

7.3.2 Result of hammering

The attack result for injecting one bit flip to any one of the four T-tables can be depicted in the Table 3. The second column records the attack time that it takes to successfully inject one bit to any of T_0', T_1', T_2', T_3', ranging from 3 up to 230 minutes for the first 20 experiments facilitated with profiling information. The injection time on average is about 64.1 minutes. The exact location of the detected bit flip, the correct value of that bit in standard T-tables, and the faulty value after the hammering are also listed in the last three columns of Table 3. Statistically, injections occur 5,4,6,5 times to T_0', T_1', T_2', T_3', in 90.80, 57.75, 49.83, 59.6 minutes respectively. The numbers of injections are evenly distributed among all four tables with certain small variance in the injection time. In contrast, two experiments are conducted without the profiling information, which takes about 461 and 1367 minutes, respectively.

7.3.3 Result of fault analysis

According to experimental results in Table 3, the adversary $\mathcal{A}$ can inject one bit flip into one of the four tables. Since REDMR is vulnerable to PFA, the faulty ciphertexts are available to $\mathcal{A}$. Bytes of K^{10} are recovered row wise. A flip is injected in $T_2'[100]$ from $0x43$ to $0x41$, and 4000 ciphertext are collected. This allows the recovery of k_2, k_6, k_{10}, k_{14}. Similarly, flip in $T_0'[235]$ from $0xe9$ to $0xa9$ and $T_3'[67]$ from $0x70$ to $0x50$, helps recover k_0, k_4, k_8, k_{12} and k_3, k_7, k_{11}, k_{15}. Rest of the key can be recovered by brute force or another flip in $T_1'[208]$ from $0x70$ to $0x50$. The number of ciphertexts required for recovering each column of the key is shown in Fig. 12. Roughly, 8200 ciphertexts are required *i.e.* 2050 per row, to recover the full last round key.

Table 3: The attack result for injecting one bit flip to any of T'_i with or w/o profiling.

ID	Attack time(min)	Location of flip	Data before injection	Data after injection
1	30	$T'_0[235]$	e900 0000	a900 0000
2	38	$T'_1[208]$	0070 0000	0050 0000
3	53	$T'_2[100]$	0000 4300	0000 4100
4	81	$T'_3[67]$	0000 001a	0000 0018
5	230	$T'_0[18]$	c900 0000	c800 0000
6	102	$T'_1[131]$	00ec 0000	00cc 0000
7	77	$T'_2[172]$	0000 9100	0000 9000
8	3	$T'_3[34]$	0000 0093	0000 0091
9	104	$T'_0[230]$	8e00 0000	8600 0000
10	49	$T'_2[126]$	0000 f300	0000 7300
11	86	$T'_3[101]$	0000 004d	0000 004c
12	75	$T'_3[55]$	0000 009a	0000 001a
13	17	$T'_2[221]$	0000 c100	0000 8100
14	44	$T'_1[67]$	001a 0000	0018 0000
15	53	$T'_3[147]$	0000 00dc	0000 00d8
16	5	$T'_0[108]$	0000 0050	0000 0010
17	41	$T'_2[252]$	0000 0f00	0000 0b00
18	62	$T'_2[140]$	0000 6400	0000 4400
19	47	$T'_1[13]$	00d7 0000	0097 0000
20	85	$T'_0[168]$	c200 0000	8200 0000
1	461(w/o profiling)	$T'_3[75]$	0000 00b3	0000 00f3
2	1367(w/o profiling)	$T'_1[163]$	000a 0000	0002 0000

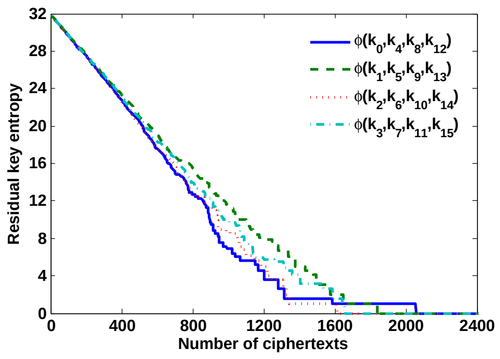


Figure 12: The residual key entropy v.s. the sample size. The PFA attack is conducted on Libgcrypt 1.6.3 with REDMR.

8 Conclusion

In this paper, we propose persistent fault analysis, a novel fault attack based on persistent fault model. The power of the proposed PFA is that it can even attack general block ciphers hardened with certain countermeasures against fault attacks. The attack is first validated in an FPGA environment on AES-128 hardened with DMR countermeasures, to recover the last round key. Further, using rowhammer based fault injection, the attack is practically conducted in a shared library setting to target AES-128 in cryptographic library Libgcrypt. The proposed attack opens an alternate analysis technique which motivates to break various fault countermeasures under this threat model. As a countermeasure, built-in health test with fault counters can be integrated to verify the functionality of the algorithm before performing block encryptions and limit the number of faulty ciphertexts. This can be easily performed, for instance, at S-box level but remains difficult to implement for more complex algorithms. Moreover, it motivates to research new configurations of DMR countermeasures and other novel countermeasures to resist PFA.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant 61472357 and 61571063, and in part by the Zhejiang University Fundamental Research Funds for the Central Universities under Grant 2018QNA5005.

References

[ACS+07] M. Alderighi, F. Casini, D S., S. Pastore, G. R. Sechi, and R. Weigand. Evaluation of single event upset mitigation schemes for sram based fpgas using the flipper fault injection platform. In *IEEE International Symposium on Defect and Fault-Tolerance in Vlsi Systems*, pages 105–113, 2007.

[AES] http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[ASSS16] Alejandro Cabrera Aldaya, Alejandro J. Cabrera Sarmiento, and Santiago SÍćnchez-Solano. Aes t-box tampering attack. *Journal of Cryptographic Engineering*, 6(1):31–48, 2016.

[BBB+18] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Mustafa Khairallah, and Thomas Peyrin. Protecting block ciphers against differential fault attacks without re-keying. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 191–194. IEEE, 2018.

[BECN+06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[BHS94] Gunnar Blom, Lars Holst, and Dennis Sandell. *Problems and Snapshots from the World of Probability*. Springer Verlag, 1994.

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. *Lecture Notes in Computer Science*, 4727:450–466, 2007.

[BM16] Sarani Bhattacharya and Debdeep Mukhopadhyay. *Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis*. Springer Berlin Heidelberg, 2016.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. *Lncs*, 1294:513–525, 1997.

[BS06] Eli Biham and Adi Shamir. Differential cryptanalysis of the data encryption standard. *Crystal Research & Technology*, 17(1):79–89, 2006.

[CWJ10] Nicolas T Courtois, David Ware, and Keith M Jackson. Fault-algebraic attacks on inner rounds of DES. 2010.

[dat09] Data2mem user guide (ug658), 2009.

[DDL97] Boneh Dan, Richard A. Demillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *International Conference on Theory and Application of Cryptographic Techniques*, pages 37–51, 1997.

[DLV03] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on A.E.S. In *International Conference on Applied Cryptography and Network Security*, pages 293–306, 2003.

[FJLT13] Thomas Fuhr, Eliane Jaulmes, Victor Lomne, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *The Workshop on Fault Diagnosis & Tolerance in Cryptography*, pages 108–118, 2013.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED block cipher. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 326–341, 2011.

[Insa] https://www.riscure.com/security-tools/inspector-fi/.

[Insb] https://www.riscure.com/security-tools/inspector-sca/.

[Joy12] Marc Joye. Fault analysis in cryptography. *Information Security & Cryptography*, 2012.

[KDKF14] Yoongu Kim, R Daly, J Kim, and C Fallin. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *Proceeding of the International Symposium on Computer Architecuture*, pages 361–372, 2014.

[lib] https://gnupg.org/software/libgcrypt/index.html.

[LSG+10] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. Fault sensitivity analysis. In *Cryptographic Hardware and Embedded Systems, CHES 2010, International Workshop, Santa Barbara, Ca, Usa, August 17-20, 2010. Proceedings*, pages 320–334, 2010.

[RGB+16] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 1–18, 2016.

[Riv09] Matthieu Rivain. Differential fault analysis on DES middle rounds. In *Cryptographic Hardware and Embedded Systems - CHES 2009, International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 457–469, 2009.

[SHP10] Jörn Marc Schmidt, Michael Hutter, and Thomas Plos. Optical fault attacks on AES: A threat in violet. In *Fault Diagnosis and Tolerance in Cryptography*, pages 13–22, 2010.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: an ultra-lightweight blockcipher. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 342–357, 2011.

[Sko10] Sergei Skorobogatov. Optical fault masking attacks. In *Fault Diagnosis and Tolerance in Cryptography*, pages 23–29, 2010.

[TJ09] Randy Torrance and Dick James. *The State-of-the-Art in IC Reverse Engineering*. Springer Berlin Heidelberg, 2009.

[TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. *Community Mental Health Journal*, 49(6):658–667, 2011.

[WCWW13] An Wang, Man Chen, Zongyue Wang, and Xiaoyun Wang. Fault rate analysis: Breaking masked aes hardware implementations efficiently. *IEEE Transactions on Circuits & Systems II Analog & Digital Signal Processing*, 60(8):517–521, 2013.

[XZZT16] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 19–35, 2016.