One Fault is All it Needs

Breaking Higher-Order Masking with Persistent Fault Analysis*

Jingyu Pan[1,2,3], Shivam Bhasin[2], Fan Zhang[1,3], and Kui Ren[1,3]

[1] Zhejiang University, China
{joeypan,fanzhang,kuiren}@zju.edu.cn
[2] Temasek Laboratories, Nanyang Technological University, Singapore
sbhasin@ntu.edu.sg
[3] State Key Laboratory of Cryptology, China

Abstract. Persistent fault analysis (PFA) was proposed at CHES 2018 as a novel fault analysis technique. It was shown to completely defeat standard redundancy based countermeasure against fault analysis. In this work, we investigate the security of masking schemes against PFA. We show that with only one fault injection, masking countermeasures can be broken at any masking order. The study is performed on publicly available implementations of masking.

Keywords: fault attacks · masking · persistent.

1 Introduction

Fault attacks [3] are a type of physical attacks which considers an active attacker capable of disturbing the operation of the target system. Fault attacks have been powerful against standard cryptographic schemes, such as AES, RSA, etc.

Most fault attacks assume a *transient* fault model, where the injected disturbance or fault is temporary, and ideally it affects only one instance of the target function call (eg. one encryption). Some attacks also consider a *permanent* fault model which affects all calls to the target function. Such faults often arise from physical defects in the device. Recently at CHES 2018 [15], a new fault model was highlighted which remains between transient and permanent, called as *persistent* fault. Unlike transient fault, it affects several calls of the target function, however, *persistent* fault is not permanent, and disappears with a device reboot.

A specific fault analysis technique to exploit persistent fault on block ciphers was also developed and called as *Persistent Fault Attack (PFA) [15]*. PFA was shown to break fault countermeasures based on module redundancy and comparison. Masking [11] is one of the most-studied countermeasures against side-channel attacks.

* This work was supported in part by the National Natural Science Foundation of China under the grants 61472357,61571063, and by the the Open Fund of State Key Laboratory of Cryptology, China.

In this work, we investigate the security of some popular masking schemes against PFA. Publicly available implementations are used for the analysis. Our results show that masking countermeasures can be easily broken with PFA. We highlight the main advantage of PFA over other fault attacks. PFA needs only one fault injection as compared to typically one fault per encryption for other fault analysis technique. With one fault injection and multiple encryptions with same fault, PFA can break masking countermeasure at any order. This reduces the practical effort that an attacker should bare to minimum.

The rest of the paper is organized as follows. Section 2 recalls principles of PFA and masking. Section 3 applies PFA on general masking construction. Case study on security of public implementation of masking schemes against PFA is described in Section 4 and final conclusions are drawn in Section 5.

2 Background

This section recalls general background concepts about PFA and masking.

2.1 Persistent Fault Analysis (PFA)

PFA was recently introduced as a novel fault analysis technique in CHES 2018 [15]. Unlike other fault attacks which rely on transient or permanent fault model, PFA exploits persistent fault model. As stated earlier, under persistent fault model, the fault affects several consecutive encryptions. The fault, typically, alters a stored constant (like one or more entry in Sbox look-up) in the target algorithm.

For better comprehension, let us take an example of PRESENT cipher where a random nibble fault alters one Sbox element v to v'. In absence of fault, all elements in 4×4 Sbox including v, v' have an expectation of $(v) = (v') = \frac{1}{16}$. If a persistent fault is injected to change v to v', $(v) = 0, (v') = \frac{2}{16}$, while all other elements still hold the expectation $\frac{1}{16}$. If in a certain Sbox call in any round, the original output is v, it will be replaced by v', leading to faulty ciphertext. Some encryptions will still be correct as they won't access the element v of the Sbox during the whole encryption. This difference can be detected statistically over a big set of ciphertexts, just by observing the distribution of each nibble, leaking information of the whole last round key k. Fig. 1 illustrates PFA. A fault is injected into the Sbox and turns an element of Sbox from $v = 10$ into $v' = 12$. v, v' are not required to be known to the attacker and can be brute forced. The following statistical tools can be used for key recovery:

1. t_{min}: find the missing value in Sbox table. Then $k = t_{min} \oplus v$;
2. $t \neq t_{min}$: find other values t where $t \neq t_{min}$ and eliminate candidates for k;
3. t_{max}: find the value with with maximal probability $k = t_{max} \oplus v'$.

The attacker needs enough number of ciphertexts to confidently distinguish distribution of t_{min} or t_{max} from others. The minimum number of ciphertexts N can be computed by the classical coupon collector's problem where it needs

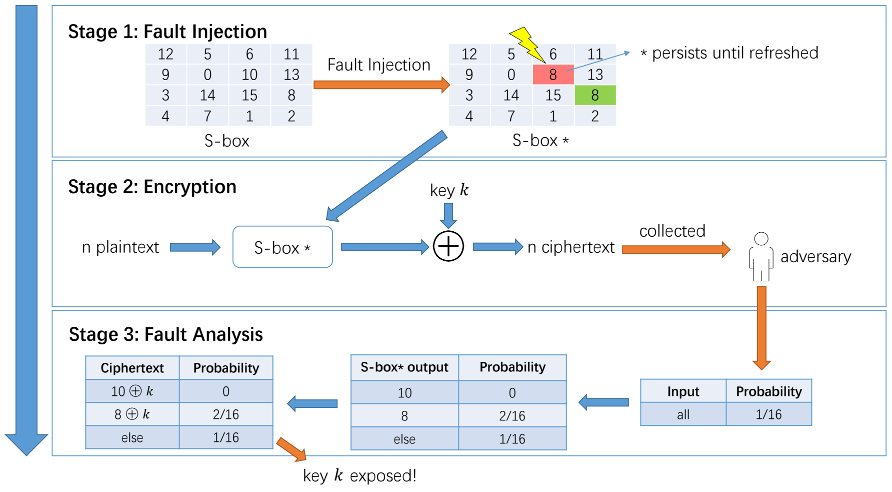
Stage 1: Fault Injection
Fault Injection
S-box
S-box *
* persists until refreshed
Stage 2: Encryption
n plaintext
S-box *
key k
n ciphertext
collected
adversary
Stage 3: Fault Analysis
Ciphertext / Probability: $10 \oplus k$ 0; $8 \oplus k$ 2/16; else 1/16
S-box* output / Probability: 10 0; 8 2/16; else 1/16
Input / Probability: all 1/16
key k exposed!

Fig. 1. Overview of Persistent Fault Attack.

$N = (2^b - 1) \times (\sum_{i=1}^{(2^b-1)} \frac{1}{i})$, where b is the bit width of x. For PRESENT ($b = 4$) $N \approx 50$, and for AES (b=8) $N \approx 1560$. More details on PFA and its application on redundancy based fault countermeasures can be found in [15].

2.2 Masking

Masking [11] is the most studied countermeasure against side-channel attacks. The key idea behind masking is to mask the side-channel activity of a sensitive intermediate value in a cryptographic algorithm by mixing it with a random value. Each encryption call requires fresh randomness to totally remove dependency between sensitive value and side-channel activity. Randomness are sometimes updated several times within an encryption to avoid sophisticated attacks like higher-order attacks. Theoretically, masking does not prevent against fault attacks, however, due to randomness involved, the fault analysis can be complicated.

2.3 Related Works

Masking has come under the scanner of fault attacks in few previous work. Boscher and Handschuh [4] showed that masking does not protect against classical differential fault attacks. While the analysis was a bit more restrictive in terms of the fault model and the number of faults that are required, the key recovery was possible with increased attack effort. A new kind of fault analysis called fault sensitivity analysis (FSA) was shown to break masking by Li et al [7]. FSA used some side-channel information with fault attack to achieve the goal, again with increased effort as compared to unprotected implementation. FSA was further combined with collision attack to enhance its power leading to stronger attack on several countermeasures including masking and threshold implementation [9]. Use of randomness was recommended as a fault countermeasure prerequisite by Lomne et al [8]. Recently in CHES 2018, a special class of fault attack called statistical ineffective fault attack (SIFA [6]) were used to target and break masking countermeasure at any masking order. SIFA requires several ineffective fault injection to statistically determine the key. In this work, we assess the security of several public implementations of masking countermeasure under PFA. As shown later, PFA on masking requires only one fault injection and breaks masking at any order d.

3 PFA on Masking

3.1 Fault Model

We follow the general PFA threat model that is:

1. The adversary can inject the persistent fault in some cipher constant (or look-up tables) before the encryption process. The (serialized) implementation of block cipher uses one look-up table for all words (bytes or nibbles) and all rounds.
2. The adversary is able to collect multiple ciphertext outputs with random plaintext (not known).

As required in PFA, fault injection to disturb memory content has been practically demonstrated in range of devices including micro-controllers, FPGA and ASIC [14, 12]. Persistent fault on modern CPU using rowhammer was presented in [15]. In the following, we analyze masking schemes under the said threat model.

3.2 General Idea

Block ciphers are composed of repetitions of a round function. In PFA, we are mainly concerned about the final round since it's directly related to the ciphertext. The last round of cipher with basic boolean masking can be written as follows:

$$c = (L(S'(x \oplus m) \oplus m^{'}) \oplus k) \oplus L(m^{'}) \tag{1}$$

where c denotes the ciphertext, L denotes some linear function (typically permutations), x denotes the last round input, m and m' denote penultimate and last round masks, respectively. k denotes the round key and $S^{'}(x)$ denotes the masked Sbox which can be calculated as $S'(x) = S(x \oplus m)$. Note that higher order masking can also be included in this analysis, where m can be calculated as $m = m_1 \oplus m_2 \oplus \cdots \oplus m_d$ with d as the masking order.

In our attack model against masking block ciphers, we assume the original (unmasked) Sbox is stored for look-up and a persistent fault is injected. The analysis scheme remains generic as illustrated in the previous section. For each Sbox call in the encryption, ideally a fresh set of masks are drawn and a new masked Sbox S' is computed. This is popularly known as the re-computation method.

If faulty value x' is injected to the i^{th} element of S where the original value $S(i) = x \neq x'$, it leads to the faulty element in the correspondingly calculated masked Sbox where $S'(i \oplus m) = x' \oplus m'$. Consequently, the $x \oplus m$ element is missing in the S' and the $x' \oplus m$ element is doubled. With this knowledge, the adversary can deduce that $c_* = L(x \oplus m') \oplus L(m') \oplus k = L(x) \oplus k$ will not appear in the output ciphertexts. Similarly, $c'_* = L(x') \oplus k$ will be doubled. Since the computation of c_*, c'_* does not depend on either m or m', the attack is equivalent to attacking an unmasked implementation. Even for d order masking, m and m' can be written as the combination of d mask, which eventually gets cancelled out to compute the ciphertext, making the complexity constant even when increasing order d.

4 Case Studies: Breaking Public Implementation of Masking Schemes with Single Fault

We target a few public implementations of masking in this section. The key advantage of PFA is that it requires only one fault injection and multiple encryptions, thus limiting the practical effort of injecting the faults. The required fault model is described before and several works have been practically validated in a range of devices. In the following, we focus on developing the analysis technique with simulations under compatible fault models.

4.1 Bytewise Masking AES

We apply PFA to the public implementation of bytewise masking available at [2]. It is a typical implementation that follows the general idea illustrated in the previous section. In this case, 6 randomly-generated masks denoted by $m, m^{'}, m_1, m_2, m_3, m_4$ are involved in each encryption, where $m_i, 1 \leq i \leq 4$ correspond to 4 rows of AES, respectively. For the MixColumns operation $MC(col_1, col_2, col_3, col_4)$, 4 output-masks have to be calculated in advance accordingly, denoted by $m^{'}_1, m^{'}_2, m^{'}_3, m^{'}_4$, such that $(m^{'}_1, m^{'}_2, m^{'}_3, m^{'}_4) = MC(m_1, m_2, m_3, m_4)$.

When all 10 masks are generated, a masked AES Sbox denoted by S' is pre-calculated prior to the encryption as:

$$S^{'}_{m,m'}(x) = S(x \oplus m) \oplus m^{'} \tag{2}$$

where S denotes the original AES Sbox in which a persistent fault will be injected, and m and m' are the generated masks. With a persistent fault in S, every S' will contain a fault, irrespective of mask values. One single fault would be enough to reveal the key with the statistical method.

Algorithm 1: Bytewise Masking AES

```
**Input:** plaintext $p = (p_1, p_2, p_3, p_4)$, where $p_i, 1 \leq i \leq 4$ represent the $i^{th}$ column vector of $p$, key $k$
**Output:** ciphertext $c$
1  $rk \leftarrow KeySchedule(k)$
2  $(m, m', m_1, m_2, m_3, m_4) \xleftarrow{\$} (\mathbb{F}_{2^8}, \dots)$
3  $(m'_1, m'_2, m'_3, m'_4) \leftarrow MixColumn(m_1, m_2, m_3, m_4)$
4  $S' \leftarrow GetMaskedSbox(S, m, m')$                // Eq (2)
5  $x \leftarrow p \oplus (m, \dots) \oplus rk[0]$
6  for $i = 1; i < 10; i++$ do
7      $x \leftarrow S'(x)$
8      $x \leftarrow ShiftRows(x)$
9      $x \leftarrow x \oplus (m_1, m_2, m_3, m_4) \oplus (m', \dots)$
10     $x \leftarrow MixColumn(x)$
11     $x \leftarrow x \oplus rk[i] \oplus (m'_1, m'_2, m'_3, m'_4) \oplus (m, \dots)$
12 end
13 $x \leftarrow S'(x)$
14 $x \leftarrow ShiftRows(x)$
15 $c \leftarrow x \oplus rk[10] \oplus (m', \dots)$
```

The algorithm of this bytewise masking AES is shown in Algorithm 1. The operations directly affected by the persistent fault are shown in red. Here we apply the t_{min} strategy. We use the available code for our analysis. We injected one persistent fault in S, by randomly changing one Sbox element. The attack was repeated 100 times and the average of all results is computed. Fig. 2 shows the guessing entropy or the no. of key candidates to test against no. of ciphertexts. By coupon's collector problem the minimum number of ciphertext required are ≈ 1560. In the experiments we found that with 1500 ciphertexts the attacker has on average less than 2 key byte candidates to test and a unique key with little over 2000 ciphertexts. The analysis remains exactly the same to recover all the bytes independently from same set of ciphertext, thus revealing the last round key and eventually the master key.

4.2 Coron's Higher-order Masking of Look-up Tables [5]

In Eurocrypt 2014, Coron presented a method to securely compute look-up tables in a block cipher, secure at any order d [5]. This scheme is an ideal target for PFA as it uses look-up tables by design, which is vulnerable to persistent faults. We target the publicly available implementation of AES protected with this scheme, provided by the author [1].

The key feature of Coron's countermeasure [5] is table recomputation. It uses independent masks with additional refresh of the masks between every successive shift of the input. One can view every line u of the randomized table as a n-

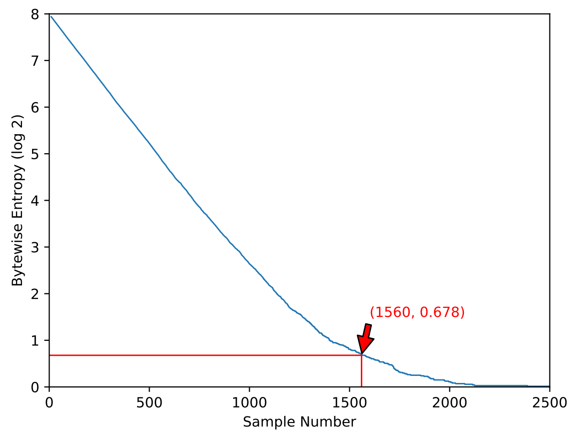

Fig. 2. Relation between number of samples and Bytewise Masking AES key guess entropy.

dimensional vector of elements in $\{0,1\}^k$, and for all inputs $u \in \{0,1\}^k$:

$$T(u) = (s_{u,1}, s_{u,2}, \ldots, s_{u,n})$$

where initially each vector $T(u)$ is a n-boolean sharing of the value $S(u \oplus x_1)$. The vectors $T(u)$ of the randomized table are then progressively shifted for all $u \in \{0,1\}^k$, first by x_2 and so on until x_{n-1}. Eventually the evaluation of $T(x_n)$ gives a vector of n output shares that corresponds to $S(x)$.

To refresh the masks between successive shifts one can generate a random n-sharing of 0, that is $a_1, \ldots, a_n \in \{0,1\}^k$ such that $\bigoplus_{i=1}^{n} a_i = 0$, and XOR the vector $T(u)$ with $(a_1, \ldots, a_n)$, independently for every u. More concretely, we can use the **RefreshMasks** procedure in Algorithm 2 from [11], which gives a masking of y as $y = y_1 \oplus \cdots \oplus y_n$ by XORing both y_1 and y_i with $r_i \xleftarrow{\$} \mathbb{F}_{2^k}$, in an iterative manner from $i = 2$ to n, where the original value of y_1 is y. The full description of the procedure of Coron's higher order masking of look-up tables is provided in Algorithm 3.[4]

Algorithm 3 uses two temporary tables T and T' in RAM. Both are generated on the basis of the look-up table $S : \{0,1\}^k \to \{0,1\}^k$. We show that, however, with as few as one single faulty element in table S, the following masking provides no protection against PFA. The operation marked in red in Algorithm 3 denotes the one directly involving injected persistent fault. It results in a faulty table S', which is same as table S but one element.

The attack is performed on AES implementation available at [1], which follows Algorithm 3. For each attack, a single fault is injected into S, and PFA is applied for $d = 1$. The masking offers no resistance against PFA as it reduces to

[4] For simplicity, we assume both the input and output of $S(x)$ are words of k bits.

Algorithm 2: RefreshMasks

```
**Input:** shares $(x_i)_i$ satisfying $\bigoplus_i x_i = x$
**Output:** shares $(x_i^{'})_i$ satisfying $\bigoplus_i x_i^{'} = x$
1 $(z_0^{'}, z_1^{'}, \ldots, z_d^{'}) \leftarrow (z_0, z_1, \ldots, z_d)$
2 for $i = 1; i < d + 1; i++$ do
3     $r_i \xleftarrow{\$} \mathbb{F}_{2^k}$
4     $z_0^{'} \leftarrow z_0^{'} \oplus r_i$
5     $z_i^{'} \leftarrow z_i^{'} \oplus r_i$
6 end
```

Algorithm 3: Coron's Masked Computation of $y = S(x)$

```
**Input:** shares $x_1, \ldots, x_n$ such that $\bigoplus_i x_i = x$
**Output:** shares $y_1, \ldots, y_n$ such that $\bigoplus_i y_i = y = S(x)$
1  for *all* $u \in \mathbb{F}_{2^k}$ do
2      $T(u) \leftarrow (S(u), 0, \ldots, 0) \in (\mathbb{F}_{2^k})^n$                // $(T(u)) = S(u)$
3  end
4  for $i = 1$ *to* $n - 1$ do
5      for *all* $u \in \mathbb{F}_{2^k}$ do
6          for $j = 1$ *to* $n$ do
7              $T^{'}(u)[j] \leftarrow T(u \oplus x_i)[j]$                   // $T^{'}(u) \leftarrow T(u \oplus x_i)$
8          end
9      end
10     for *all* $u \in \mathbb{F}_{2^k}$ do
11         $T(u) \leftarrow RefreshMasks(T^{'}(u))$       // $\bigoplus(T(u)) = S(u \oplus x_1 \oplus \cdots \oplus x_i)$
12     end
13 end
   // $\oplus(T(u)) = S(u \oplus x_1 \oplus \cdots \oplus x_{n-1})$ for all $u \in \mathbb{F}_{2^k}$
14 $(y_1, \ldots, y_n) \leftarrow RefreshMasks(T(x_n))$                    // $\oplus(T(x_n)) = S(x)$
```

the generic case presented in Section 3.2, where the key recovery remains independent of the mask. This results in the attack similar to unprotected AES with key recovery with around 2000 ciphertexts (see Fig. 2). The increase in masking order d has no impact on the attack because the combination of d different masks can be reduced to a single equivalent mask as $m = m_1 \oplus m_2 \oplus \cdots \oplus m_d$.

Next, we target other masking schemes which do not directly use the Sbox and thus making the analysis more complicated, yet possible.

4.3 Rivain and Prouff's Masking [11]

In CHES 2010, Rivain and Prouff [11] proposed an efficient method to mask the AES Sbox processing at any order. Specifically, the authors use the algebraic structure of the AES Sbox, which is the composition of an affine function over $\mathbb{F}_2^8$ with the power function $x \mapsto x^{254}$ over $\mathbb{F}_{256}$, and they showed that it can be expressed as a sequence of operations involving a few linear functions over $\mathbb{F}_2^8$, which is easy to mask, and four multiplications over $\mathbb{F}_{256}$. If this computation is performed completely on the fly without any look-up tables, PFA does not apply in principle.

Now, we look at the public implementation of this scheme available at [1]. Let's focus on the Sbox masking part, where component affine transformation is realized through table look-up [1]. The additive part of the affine transformation is $0x63$, thus it can be checked that:

$$Af(x_0) \oplus \cdots \oplus Af(x_d) = \begin{cases} Af(x) & \text{if } d \text{ is even,} \\ Af(x) \oplus 0x63 & \text{if } d \text{ is odd,} \end{cases} \tag{3}$$

where $x = x_0 \oplus x_1 \oplus \cdots \oplus x_d$, for a d order masking. The vulnerable table look-up is highlighted in red in Algorithm 4, which we target by PFA.

Algorithm 4: Rivain and Prouff's secure AES Sbox

```
Input: shares $x_i$ satisfying $\bigoplus_i x_i = x$
Output: shares $y_i$ satisfying $\bigoplus_i y_i = y = S(x)$
1 $(y_0, \ldots, y_d,) \leftarrow Exp254(x_0, \ldots, x_d)$
2 for $i = 0; i \leq d; i++$ do
3     $y_i \leftarrow Af(y_i)$
4 end
5 if $d \bmod 2 = 1$ then
6     $y_0 \leftarrow y_0 \oplus 0x63$
7 end
```

However, we need to update the strategy of PFA to target this implementation. Recall that the main idea of PFA is to make a distinct disturbance, which is predictable or observable for the adversary, on the distribution of the output. The previous cases are ideally vulnerable to PFA since the output of the target function (Sbox) is linearly dependent on one single look-up of a permutation table, thus it's rather easy to produce distinguishable and predictable faulty outputs with one single fault. When multiple look-up operations are involved in the target function, as in the case of Rivain-Prouff's Sbox, we show that the output is still distinguishable and predictable with one random fault injection for any masking order, to allow PFA.

Consider a random variable $r(v, v^*, \delta) \in \{0,1\}^b, b \in \mathbb{N}^+$ whose probability is

$$Pr(r = k) = \begin{cases} \frac{1}{2^b} + \delta & k = v^*, \\ \frac{1}{2^b} - \delta & k = v, \\ \frac{1}{2^b} & \text{else}, \end{cases} \tag{4}$$

where $v, v^* \in \{0,1\}^b$ and $0 < \delta \le \frac{1}{2^b}$. Therefore for independent $r_0(v, v^*, \delta)$ and $r_1(v, v^*, \Delta)$, we have

$$Pr(r_0 \oplus r_1 = k) = \begin{cases} \frac{1}{2^b} + 2\delta\Delta & k = 0, \\ \frac{1}{2^b} - 2\delta\Delta & k = v \oplus v^*, \\ \frac{1}{2^b} & \text{else}. \end{cases} \tag{5}$$

So $r_0(v, v^*, \delta) \oplus r_1(v, v^*, \Delta)$ is equivalent to $r(v \oplus v^*, 0, 2\delta\Delta)$. Similarly we can show that $r_0(v, v^*, \delta) \oplus r_2(v \oplus v^*, 0, \Delta)$ is equivalent to $r(v, v^*, 2\delta\Delta)$.

With one persistent random fault injection into the Af table, when the random input x is under uniform distribution, the output of the faulted table $Af^{'}(x)$ is equivalent to the random variable r above as $r(v, v^*, \frac{1}{2^8})$, where v denotes the original value of the element where the fault is injected, and v^* denotes the faulty value.

For masking order $d = 1$, by Equation (5), we have

$$Pr(Af^{'}(x_0) \oplus Af^{'}(x_1) = k) = \begin{cases} \frac{1}{2^8} + 2 \times (\frac{1}{256})^2 & k = 0, \\ \frac{1}{2^8} - 2 \times (\frac{1}{256})^2 & k = v \oplus v^*, \\ \frac{1}{2^8} & \text{else}, \end{cases} \tag{6}$$

which is equivalent to $r(v \oplus v^*, 0, 2 \times (\frac{1}{256})^2)$. The bias is much lower as compared to previous cases, requiring more samples for the attack.

For any odd masking order d, we can decompose $\bigoplus_{i=0}^{d} Af^{'}(x_i) = \bigoplus_{i=0}^{\frac{d}{2}} (Af^{'}(2i) \oplus Af^{'}(2i+1))$ to $\frac{d+1}{2}$ pairs of independent outputs of $Af^{'}$. Each pair is equivalent to $r(v \oplus v^*, 0, 2 \times (\frac{1}{256})^2)$. By applying Equation 5 $\frac{d+1}{2}$ times, we have $\bigoplus_{i=0}^{d} Af^{'}(x_i)$ is equivalent to $r(v \oplus v^*, 0, 2^d \times (\frac{1}{256})^{d+1}) = r(v \oplus v^*, 0, 2^{-7d-8})$. For any even masking order d, we consider it as a combination of the $d-1$ order masking and $Af^{'}(x_d)$, whose probability should be the same with $r_{(d-1)}(v \oplus v^*, 0, 2^{d-1} \times (\frac{1}{256})^d) \oplus r_d(v, v^*, \frac{1}{2^8})$ which is equivalent to $r(v, v^*, 2^d \times (\frac{1}{256})^{d+1}) = r(v, v^*, 2^{-7d-8})$. In Fig. 3, we apply this strategy to the public implementation of [1], where key k can be extracted with both t_{max} and t_{min} strategy, when d is odd.

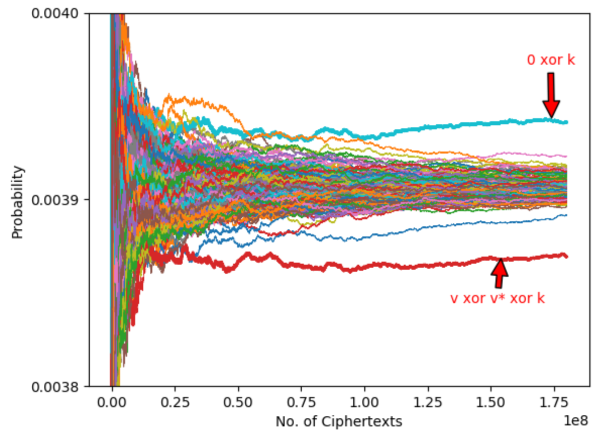

Fig. 3. Key Extraction for Rivain and Prouff scheme [1] with $d = 1$.

However, since $\delta = 2^{-7d-8}$, it decreases exponentially as masking order d increases, and thus more ciphertexts are required to perform PFA. In order to make an estimation of the number of ciphertexts required with higher masking order d, we study the case of AES. For each ciphertext byte, it has the probability of $\frac{1}{256}$ of appearing, so with n ciphertexts, the total number c of its appearance obeys binomial distribution as $c \sim \mathcal{B}(n, p)$, where $p = \frac{1}{256}$. Therefore the variance of $\frac{c}{n}$ is $\frac{p(1-p)}{n}$, and by central limit theorem, $\frac{c}{n}$ approximately follows normal distribution $\mathcal{N}(p, \frac{p(1-p)}{n})$. To perform PFA successfully, we need $\frac{\sqrt{\frac{p(1-p)}{n}}}{2^{-7d-8}} \propto$ constant. Therefore we have $n \propto 2^{14d}$, which means n grows exponentially as d increases.

4.4 Software Threshold [13]

Sasdrich *et al.* [13] extended the widely used threshold implementation (TI [10]) for software targets. They use PRESENT cipher as a case study, showing a first-order secure implementation. Interested readers can refer to [13] for details on software TI implementation of PRESENT. As public source code is not available, we implemented it in C language.

We implemented Algorithm 5. It uses a look-up table:

$$T(x^i, x^j) = A^{''}(f_{\mathcal{Q}_{12}}(A(x^i), A(x^j))$$

which is composed of 256 elements of 4 bits. Targeting at T is not optimal, as each element stands a much less chance of being accessed in the process of encryption. Instead, we target at the smaller look-up table A''' : 8FDACB9E43160752 which is an affine permutation of 4-bit elements and already marked in red in Algorithm 5.

Algorithm 5: First-Order Threshold Implementation of PRESENT

```
   Input: $\bar{x} = (x^1, x^2, x^3)$: shared plaintext
   $k$: cipher key
   Output: $\bar{y} = (y^1, y^2, y^3)$: shared ciphertext
1  $rk \leftarrow KeySchedule(k)$
2  for $i = 1; i \leq 31; i++$ do
3      $x^1 \leftarrow x^1 \oplus rk[i]$
4      $t^3 \leftarrow T(x^1, x^2)$
5      $t^2 \leftarrow T(x^3, x^1)$
6      $t^1 \leftarrow T(x^2, x^3)$
7      $t^3 \leftarrow A'''(t^3)$
8      $t^2 \leftarrow A'''(t^2)$
9      $t^1 \leftarrow A'''(t^1)$
10     $x^3 \leftarrow T(t^1, t^2)$
11     $x^2 \leftarrow T(t^3, t^1)$
12     $x^1 \leftarrow T(t^2, t^3)$
13     $x^1 \leftarrow P(x^1)$
14     $x^2 \leftarrow P(x^2)$
15     $x^3 \leftarrow P(x^3)$
16 end
17 $y^1 \leftarrow x^1 \oplus rk[32]$
18 $y^2 \leftarrow x^2$
19 $y^3 \leftarrow x^3$
```

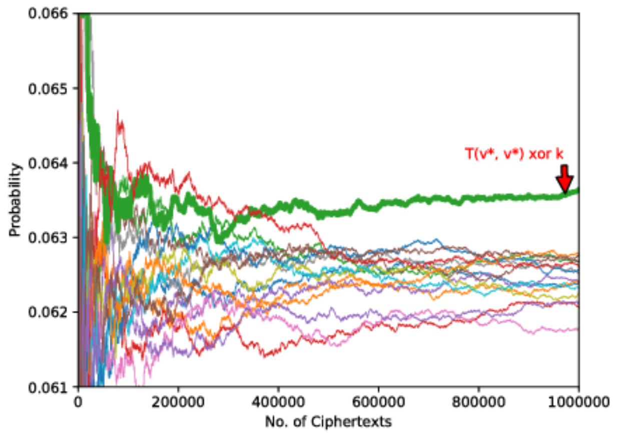

Fig. 4. Key Extraction with t_{max} strategy on Software TI [13].

Intuitively, one single fault seems insufficient for PFA since each access of the faulted table is relevant to only one share of all three. However, we can use the same model with the Rivain-Prouff's AES Sbox to estimate the probability distribution of the final output of threshold implementation.

For example, a faulty value 0 is injected into the first element of table A''', whose original value is 8. This injects a bias in the input of T (see Algorithm 5). While an input of 8 will never arrive, input 0 is doubled. In this condition, the probability distribution of the outputs of function T is biased as well. Let T' denote the biased T. The truth table of T' shows probability of 6 being the output $\frac{9}{256}$, and the probability of 12 is $\frac{23}{256}$, while all the others have probabilities that are much closer or equal to $\frac{16}{256}$.

We can use the same analysis model in the Rivain-Prouff's case and calculate the probability distribution of $T^{'}(x_0, x^1) \oplus T^{'}(x_2, x^0) \oplus T^{'}(x_1, x^2)$. Note that for any fault injection with a random fault f, $T(v^*, v^*)$ will have maximal probability to appear at output of T'. Correspondingly, with v denoting the original value where the fault is injected, $T(v, v)$ will always be the one with minimal probability. Therefore, either t_{max} or t_{min} strategy can be applied to extract the key k. In Fig. 4, we show how t_{max} strategy can be applied to recover the key with less than $400K$ ciphertexts.

5 Conclusions

PFA was recently introduced as a novel fault attack. In this work, we show that one persistent fault is enough to break masking at any masking order d. This is validated on public implementations.

To conclude, the main advantage of PFA over other fault analysis is that, PFA needs only one fault injection which could last for multiple encryptions, bringing the practical effort of injecting a fault to bare minimum. While avoiding usage of look-up tables completely can prevent PFA, it cannot be a practical solution. A larger look-up table decreases the disturbance caused by a single fault injection, and thus increases the number of ciphertexts required for key extraction. Yet such alternative is not always feasible as well. Research for novel countermeasures against PFA is motivated. Application of PFA to fault detection enhanced masking or other combined countermeasure is another interesting direction.

References

1. Higher Order Countermeasures for AES and DES, https://github.com/coron/htable
2. Masked-AES-Implementation, https://github.com/Secure-Embedded-Systems/Masked-AES-Implementation
3. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE **94**(2), 370–382 (2006)
4. Boscher, A., Handschuh, H.: Masking does not protect against differential fault attacks. In: Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on. pp. 35–40. IEEE (2008)
5. Coron, J.S.: Higher order masking of look-up tables. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 441–458. Springer (2014)
6. Dobraunig, C., Eichlseder, M., Gross, H., Mangard, S., Mendel, F., Primas, R.: Statistical ineffective fault attacks on masked aes with fault countermeasures. Tech. rep., Cryptology ePrint Archive, Report 2018/357, 2018. https://eprint. iacr. org/2018/357
7. Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta, K.: Fault sensitivity analysis. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 320–334. Springer (2010)
8. Lomné, V., Roche, T., Thillard, A.: On the need of randomness in fault attack countermeasures-application to aes. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on. pp. 85–94. IEEE (2012)
9. Moradi, A., Mischke, O., Paar, C., Li, Y., Ohta, K., Sakiyama, K.: On the power of fault sensitivity analysis and collision side-channel attacks in a combined setting. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 292–311. Springer (2011)
10. Nikova, S., Rijmen, V., Schläffer, M.: Secure Hardware Implementation of Nonlinear Functions. J. Cryptol. pp. 292–321 (2011). https://doi.org/10.1007/s00145-010-9085-7
11. Rivain, M., Prouff, E.: Provably secure higher-order masking of aes. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 413–427. Springer (2010)
12. Roscian, C., Sarafianos, A., Dutertre, J.M., Tria, A.: Fault model analysis of laser-induced faults in sram memory cells. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on. pp. 89–98. IEEE (2013)

13. Sasdrich, P., Bock, R., Moradi, A.: Threshold implementation in software. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 227–244. Springer (2018)
14. Selmke, B., Brummer, S., Heyszl, J., Sigl, G.: Precise laser fault injections into 90 nm and 45 nm sram-cells. In: International Conference on Smart Card Research and Advanced Applications. pp. 193–205. Springer (2015)
15. Zhang, F., Lou, X., Zhao, X., Shivam, B., He, W., Ding, R., Qureshi, S., Ren, K.: Persistent Fault Analysis on Block Ciphers. In: IACR Transactions on Cryptographic Hardware and Embedded Systems. vol. 2018, pp. 150–172 (2018)