Rowhammer Induced Intermittent Fault Attack on ECC-hardened memory

Anirban Chakraborty, *Member, IEEE,* Sarani Bhattacharya, *Member, IEEE,* Sayandeep Saha, *Member, IEEE* and Debdeep Mukhopdhyay, *Senior Member, IEEE*

***Abstract*—Fault attack is a class of active implementation based attacks which introduces controlled perturbations in the normal operation of a system to produce faulty outcomes. In case of ciphers, these faulty outcomes can lead to leakage of secret information, such as the secret key. The effectiveness and practicality of fault attacks largely depend on the underlying fault model and the type of fault induced. In this paper, we analyse the drawbacks of persistent fault model in case of error correction code (ECC) enabled systems. We further propose a novel fault attack called *Intermittent Fault Attack* which is well suited for ECC-enabled DRAM modules. We demonstrate the practicality of our attack model by inducing single bit faults using pinpointed Rowhammer technique in S-Boxes of block ciphers in a ECC protected system.**

***Index Terms*—Page Frame Cache, Rowhammer, Fault Attack, OpenSSL, Error Correction Codes.**

I. Introduction

Most of the modern computing devices include cryptographic primitives to enable secure computation and communication. The support for cryptography is provided in forms of hardware extension or software implementation or both. Albeit being mathematically robust, most of these implementations fall prey against attacks exploiting physical properties of the underlying systems, such as power consumption, EM radiation, timing variation, or faults . Such implementation-centric attacks have recently gained a lot of attention from both industry and academia due to their practicality and potency of compromising devices starting from embedded spectrum to cloud servers.

Fault attack (FA) is a class of active implementation based attacks which introduces controlled perturbation in the normal operation of a system resulting in faulty outcomes. The behaviour of a system under the influence of faults may reveal valuable information such as secret key of a block cipher. While controlled injection of faults is crucial for performing successful FAs, it have been shown in several occasions that even faults with minimal assumptions (such as, single/multiple byte faults, bit flip faults) may lead to successful key recovery. In fact, a single byte-fault is sufficient for extracting entire key of AES [1]. Moreover, such faults can be injected via several practical means in both embedded devices [2] and remotely located servers [3].

Since first conceived by Boneh et. al. [4] in 1996, several techniques have been proposed to exploit the information leakage resulting from fault injections (in the context of block ciphers)[1]. One common feature of all these techniques is that they utilize the statistical bias introduced in some intermediate state of a cipher to form a key distinguisher. One of the well-studied and commonly used fault analysis techniques is Differential Fault Analysis (DFA) where fault differentials in correct-faulty ciphertext pairs are exploited. DFA has been used extensively by security community to break ciphers like DES [6], AES [7], PRESENT [8], etc. While DFA works with random faults ignoring their physical properties, there exist techniques called Statistical Fault Analysis (SFA) [9] which typically exploit the device-centric fault properties. Such properties are manifested in the form of statistical bias in the fault distribution itself, and are dangerous in the sense that they can bypass countermeasures based on classical fault tolerance principles.

Recently, there have been two inclusions in the class of so-called SFA attacks – Statistical Ineffective Fault Analysis (SIFA) [10] and Persistent Fault Analysis (PFA) [11]. The SIFA attacks typically rely on data-dependent ineffectivity of faults. While the PFA has very similar mathematical properties, the fault model is very different from all other FAs proposed till date. More specifically, while existing FAs exploit transient faults, PFA targets the constants and tables in a cipher implementation hence resulting in a fault which persists through multiple encryption cycles. The advantage of such a persistent fault model is that only a single injection is sufficient to gather a lot of faulty ciphertexts. Moreover, the injection happens before the encryptions begin and hence tight control over the timing of injection is not required.

One of the main exploits of PFA are the table-based block cipher implementations [11]. In a server environment, persistent faults can be induced using well-known techniques such as Row-Hammer bugs in modern DRAM chips [12]. While recent operating systems try to throttle the exploitation of this bug by restricting the access to the address-mapping, recent work has shown that Row-Hammer faults can still be induced in a controlled manner from the user space exploiting micro-architectural components such as *page-frame-cache* [13]. This specific work has also shown that the analysis technique in PFA proposed in [11] (which targets only the last round of a cipher), can be generalized to target even penultimate rounds. Such generalization is crucial while attacking certain commercial cipher implementations such as OpenSSL-AES [14]. Another alternative technique for targeting penultimate rounds

[1]There exists several examples of FA on public key implementations [5]. However, in this draft we restrict ourselves to block ciphers.

has been proposed recently in [15]. However, the common part of all these techniques is that they require access to the ciphertexts. This criteria might not get fulfilled in all situations.

Modern DRAM memories often include Error-correcting codes (ECC) for preventing corruptions in computations. Such ECC-enabled memories are often considered as potential solutions to Row-Hammer induced faults [16]. Although there has been claims that even ECC-enabled memories may leak some information though timing channels, it is not obvious if such protections can also counter PFA-like attacks. There are two observations which might make one think that ECC is effective against PFA. The first observation is that **the fault cannot remain persistent anymore as it gets corrected once the faulted memory location is accessed.** As a result, **no faulty ciphertexts (resp. correct ciphertexts under the fault influence) can be obtained.**

In this paper, we further analyze the effectiveness of ECC for preventing fault attacks exploiting memory faults. More specifically, we show that *ECC cannot prevent attacks exploiting memory faults.* Referring to the first observation from the previous paragraph, here we introduce a new fault model called ***Intermittent fault model***, specific to ECC-enabled memories. The main feature of Intermittent fault model is that it remains in the system until the faulted location is accessed for the first time. Based on this fault model we present a novel attack strategy which does not require any access to the ciphertexts. The proposed attack, called ***Intermittent fault attack*** (InFA) can recover the entire key from a given block cipher implementation.

II. Intermittent Fault Model

In this section, we introduce a new fault model, called Intermittent Fault Model (InFA), suitable for error correction enabled memory systems.

Why PFA won't work in ECC memory?

Persistent Fault Attack or PFA is based on the underlying fact that faults induced by Rowhammer on S-Boxes persist until the S-Boxes are reloaded into the memory. That means, a fault once injected can affect multiple encryptions thereby producing a number of faulty ciphertexts and increasing the efficiency of statistical attacks. However, on a ECC-enabled memory, this simple fault model does not work. In ECC memory, once a fault is injected it persists until that particular memory row is accessed. Once the row that contains the fault is accessed by the memory controller, the error correction operation takes place and fault is corrected. In other words, a fault once induced in an ECC-enabled DRAM lasts as long as the faulty row is not accessed. Moreover, no faulty ciphertext is generated in this case as the S-Box computation occurs after correcting the error. Therefore, no statistical attacks can be performed on the acquired ciphertexts due to lack of faulty ones.

A. Error Correcting Codes

In modern systems, the primary component inside the processor responsible for ECC checks is the memory controller. Suppose the CPU wants to write a message of k bits into the memory, the memory controller would append r bits of extra information for error correction and detection to compose a $n = k + r$ bit *codeword* and store it in the DRAM. The r bits are calculated using *linear block codes* [17] - particularly the (7,4) binary code. Since the Hamming Distance (HD) between any codeword is atleast 3, the ECC scheme can detect upto 2 bit errors and correct a single bit fault. An extra parity bit is added to differntiate between single bit and double bit errors, which comprises the *single error correction and double-bit error detection (SECDED)* scheme [17].

B. The Fault Model

ECC-enabled memory modules has been touted as a potential countermeasure against Rowhmmer-induced faults for a long time. However, Cojocar et. al. [18] showed that it is possible to induce controlled faults in ECC memory which are used extensively in server environments. They also showed that the error correction operation imposes an additional overhead which leads to an increase in overall access time whenever a faulty location is accessed. This difference in time can create a reliable timing channel to distinguish whether the memory row accessed contains a faulty bit. This seemingly simple timing channel was utlized by Kwong et al. [19] to recover 1024-bit RSA key by inducing multiple faults in ECC memory.

III. Fault Analysis on Memory with Error Correction

One critical question regarding Improved PFA [15] and DRPFA [13] attack is that whether they still hold if in-memory error correction is present. It may seem that error correction can easily throttle the attacks described in this section. However, in reality, the vulnerability still holds.

The key observation in this context is that error correction operation do incur some extra time, which eventually results in an exploitable timing channel [18]. Unlike the other attacks, in this case, the attacker cannot obtain the faulty ciphertexts. Further, selective correct ciphertexts biased with the faults (just like SIFA attacks) are also unavailable. However, an attacker can still observe whether a fault induced by repeated hammering is getting corrected or not from the aforementioned timing difference. To further elaborate how this may lead to a successful attack, *let us assume a scenario where the fault affects one of the T-tables of AES. This is a reasonable assumption made based on the observations from our previous experiments. Also, as we were successful in inducing Rowhammer in the T-table after a profiling phase, it is possible to pinpoint the position of the fault. Thus for simplicity, we also assume that the attacker has the knowledge of the exact fault location inside the corrupted T-table. We further assume that the attacker exactly knows which of the T-tables (T_0, T_1, T_2 or T_3) has got faulted* [2]. Finally, we consider that the attacker can clearly identify the particular round when the error correction takes place. This is possible using the cache access

[2]We found that vulnerable locations within a page remains almost constant. Hence, if the attacker performs an initial profiling before the attack it is not difficult to infer the fault location within the table.

based attacks namely Prime+Probe [20] and Flush+Reload [21] where an adversary can monitor the cache memory to determine the cache accesses corresponding to different rounds of AES execution. To ease the understanding of AES per round accesses, authors in [22], [23] have devised methods to measure cache accesses in fine granularity. Cache accesses are utilized in this particular case to segregate between rounds of AES execution. Now, the error correction mechanism comes into play only when the faulty location is accessed. As discussed in [18], the error detection and correction operations incurs a timing overhead of hundreds of thousands of clock cycles, which can be measured by the attacker. So, an attacker can easily identify the round when the faulty location gets accessed during the course of encryption.

Referring to the structures of the T-tables and the associated AES implementation, it can be observed that each round involves 16 table look-ups. We only focus on the cases when the error gets corrected at the very first round of AES. Rest of the cases can be discarded. *Recall that an error correction takes place whenever the faulty entry in the T-table is accessed. Hence, an attacker, who can pinpoint the timing of the correction, can also deduce the input of the T-table at that time (since she has prior knowledge of which entry has got corrupted). This input actually corresponds to one byte of the plaintext XOR-ed with one byte of the first round key. Since the plaintext is known, the key byte can be directly deduced once the T-table input is determined.* [3]

This apparently simple attack has certain critical intricacies. Most importantly, in the presence of out-of-order execution it becomes extremely difficult (if not impossible) to deduce for which byte location the error correction has taken place. In order to uniquely determine the key bytes even in this context we apply a simple trick. We begin the attack with a randomly chosen plaintext. Note that, we want the error correction to happen at the first round, and we can identify the event while it happens at the first round. The plaintext is chosen in a way so that initially there is no error correction happening. Probability of obtaining such a plaintext is fairly high (around 85%). Now, we vary one of the bytes of the plaintext from 0 to 255 keeping all other bytes constant. *If faulty T-table and the location of the fault within it is fixed every time, for some value of the varying plaintext byte (let us denote the index of it by P_i), the error correction will take place over there (i.e. for the byte P_i).* Since we started with no error correction taking place in the first round, with this simple trick we can uniquely set and determine the target key byte by byte making the correction happen at the first round only for our target byte P_0. The entire first round key can be extracted by simply repeating this technique.

The advantage of the aforementioned trick is that it can uniquely identity the faulted byte location within a round. The round of fault, on the other hand, can be identified with the prime-and-probe as described previously. However, one should note that once the error correction takes place, the corrupted T-table becomes fault-free. In order to restart fault injection in the cipher computation, one need to perform the hammering once again. *Hence, if the corruption of computation happens somewhere other than the first round, we need to restart the hammering.* A crucial question here is that *how many times the hammering should be restarted before we recover one target key byte.* Below we provide a statistical estimation of this quantity.

Let us denote the event of a first round corruption as "success" and the event associated with the rest of the things as "failure". The "failure" event can be divided into two different sub-events – 1) the event when no fault happens at all, and 2) the event when the fault happens at some different round. One may observe that the restart is only required for the later event. In order to estimate the number of restarts required for the extraction of a single key byte, we thus need to consider two events – 1) the "success": fault happens at the first round, and 2) the "restart": fault happens at some different round.

The immediate next step would be to estimate the probability of success and the probability of restart (without loss of generality we denote this event as "failure" from now, as we need not to bother about the event when no corruption happens). Let us denote the probability of success as p and the probability of failure as $q = 1 - p$. Considering the facts that only one location t in a specific T-table T_a gets corrupted in our experiment, and there are total $4 \times 10 = 40$ accesses made to T_a in an entire encryption operation, the probability that no corruption (and hence no error correction) happens in an encryption is $(\frac{255}{256})^{40} = 0.855$. Consequently, the probability of at least one fault happening in an encryption is $1 - (\frac{255}{256})^{40} = 0.145$. In our experiments, the plaintext is varied only by one byte location (hence, total 256 possible plaintexts are tested, and a "success" event is bound to happen within that by the properties of XOR operations). Thus, the expected number of faulty encryptions happening within these 256 encryptions is given by $256 \times 0.145 = 37.12$.

The probability of success now can be estimated as $p = \frac{1}{37.12} = 0.026$ ($q = 1 - p$). The number of restarts can now be modelled with a random variable H having a truncated Geometric distribution $H \sim TGeom(n, p)$ [24] having the probability distribution function[4]:

$$g(h) = \begin{cases} \frac{pq^{h-1}}{1-q^{d-1}}, & \text{for } h \in \{1, 2, \cdots d-1\} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Here the parameter d in the distribution denote the maximum number of trials that is allowed to be performed (in our context, we know that roughly 40 restarts (on average) are possible in total for most of the plaintext-key pairs. Hence we adopt the truncated version of Geometric distribution for modelling instead of the actual one).

We are mainly interested in estimating the mean of this distribution. With our parameter settings, the mean turns out to

[3] In this context, it is worth recalling that a single round consists of total 16 T-table look-ups. One look-up takes place for each byte of AES state. The table T_0 is accessed for all the bytes in the first row of the state, and similarly T_1, T_2 and T_3 are accessed for rows 1, 2 and 3, respectively. So, having a fault at T_0 in the present context helps us to extract the key-bytes corresponding to the first row elements.

[4] The main difference between the Geometric distribution with its truncated version is that the first one has an infinite support ($\{1, 2, \cdots\}$), whereas the second one has a finite support within a predefined range ($\{a+1, a+2, \cdots, b-1\}$), where a and b are constants defining the range.

be roughly $16-18$. In other words, *the hammering experiment (for inducing faults in the T-table) should be repeated* $16-18$ *times on average to extract one byte of the round key. Our experiments also support this theoretical estimation.*

IV. Conclusion

In this paper, we introduced a novel fault analysis technique, called Intermittent Fault Attack, which is well suited for error correction code (ECC) enabled memory systems. The proposed attack model takes advantage of the timing channel produced during error correction in ECC memory to leak information of the intermediate states of block cipher operations adn eventually extract the secret key. As a practical use-case, we demonstrate the attack on OpenSSL-AES to perform complete key recovery.

References

[1] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *IFIP international workshop on information security theory and practices*. Springer, 2011, pp. 224–233.

[2] B. Yuce, P. Schaumont, and M. Witteman, "Fault attacks on secure embedded software: threats, design, and evaluation," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 111–130, 2018.

[3] M. Lipp, M. T. Aga, M. Schwarz, D. Gruss, C. Maurice, L. Raab, and L. Lamster, "Nethammer: Inducing rowhammer faults through network requests," *arXiv preprint arXiv:1805.04956*, 2018.

[4] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1997, pp. 37–51.

[5] A. Pellegrini, V. Bertacco, and T. Austin, "Fault-based attack of rsa authentication," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 2010, pp. 855–860.

[6] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Annual international cryptology conference*. Springer, 1997, pp. 513–525.

[7] S. S. Ali, D. Mukhopadhyay, and M. Tunstall, "Differential fault analysis of aes: towards reaching its limits," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 73–97, 2013.

[8] G. Wang and S. Wang, "Differential fault analysis on present key schedule," in *2010 International Conference on Computational Intelligence and Security*. IEEE, 2010, pp. 362–366.

[9] C. Dobraunig, M. Eichlseder, T. Korak, V. Lomné, and F. Mendel, "Statistical fault attacks on nonce-based authenticated encryption schemes," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2016, pp. 369–395.

[10] C. Dobraunig, M. Eichlseder, H. Gross, S. Mangard, F. Mendel, and R. Primas, "Statistical ineffective fault attacks on masked aes with fault countermeasures," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 315–342.

[11] F. Zhang, X. Lou, X. Zhao, S. Bhasin, W. He, R. Ding, S. Qureshi, and K. Ren, "Persistent fault analysis on block ciphers," Aug. 2018, pp. 150–172. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7272

[12] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 361–372. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665726

[13] A. Chakraborty, S. Bhattacharya, S. Saha, and D. Mukhopadhyay, "Explframe: Exploiting page frame cache for fault analysis of block ciphers," 2019.

[14] OpenSSL, "Openssl cryptography and ssl/tls toolkit," https://www.openssl.org/, 2019, [Online; accessed 25-June-2019].

[15] F. Zhang, Y. Zhang, H. Jiang, X. Zhu, S. Bhasin, X. Zhao, Z. Liu, D. Gu, and K. Ren, "Persistent fault attack in practice," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 172–195, 2020.

[16] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *CoRR*, vol. abs/1904.09724, 2019. [Online]. Available: http://arxiv.org/abs/1904.09724

[17] W. Ryan and S. Lin, *Channel codes: classical and modern*. Cambridge university press, 2009.

[18] L. Cojocara, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 2019 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2019.

[19] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.

[20] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," in *Topics in Cryptology – CT-RSA 2006*, D. Pointcheval, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–20.

[21] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[22] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games – bringing access-based cache attacks on aes to practice," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 490–505. [Online]. Available: https://doi.org/10.1109/SP.2011.22

[23] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Secretly monopolizing the cpu without superuser privileges," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 17:1–17:18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362920

[24] S. Chattopadhyay, C. Murthy, and S. K. Pal, "Fitting truncated geometric distributions in large scale real world networks," *Theoretical Computer Science*, vol. 551, pp. 22 – 38, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397514003521