Template-based Fault Injection Analysis of Block Ciphers

Ashrujit Ghoshal[0000−0003−2436−0230], Sikhar Patranabis[0000−0002−2309−7939], and Debdeep Mukhopadhyay

Indian Institute of Technology Kharagpur, India
`ashrujitg@iitkgp.ac.in`, `sikhar.patranabis@iitkgp.ac.in`, `debdeep@cse.iitkgp.ac.in`

Abstract. We present the first template-based fault injection analysis of FPGA-based block cipher implementations. While template attacks have been a popular form of side-channel analysis in the cryptographic literature, the use of templates in the context of fault attacks has not yet been explored to the best of our knowledge. Our approach involves two phases. The first phase is a profiling phase where we build templates of the fault behavior of a cryptographic device for different secret key segments under different fault injection intensities. This is followed by a matching phase where we match the observed fault behavior of an identical but black-box device with the pre-built templates to retrieve the secret key. We present a generic treatment of our template-based fault attack approach for SPN block ciphers, and illustrate the same with case studies on a Xilinx Spartan-6 FPGA-based implementation of AES-128.

Keywords — Template Attacks, Fault Injection, Fault Intensity

1 Introduction

The advent of implementation-level attacks has challenged the security of a number of mathematically robust cryptosystems, including symmetric-key cryptographic primitives such as block ciphers and stream ciphers, as well as public-key encryption schemes. Implementation attacks come in two major flavors - side-channel analysis (SCA) and fault injection analysis (FIA). SCA techniques typically monitor the leakage of a cryptographic implementation from various channels, such as timing/power/EM radiations, and attempt to infer the secret-key from these leakages [14, 16]. FIA techniques, on the other hand, actively perturb the correct execution of a cryptographic implementation via voltage/clock glitches [23, 2, 1], EM pulses [8] or precise laser beams [4, 5]. With the growing number of physically accessible embedded devices processing sensitive data in today's world, implementation level attacks assume significance. In particular, a thorough exploration of the best possible attacks on any cryptographic implementation is the need of the hour.

1.1 Fault Models for Fault Injection Analysis

Nearly all FIA techniques in the existing literature assume a given *fault model* (such as random faults [8] and/or stuck-at-faults [21]) in a given location of the cipher state. Some of these techniques, such as differential fault analysis (DFA) [20, 24, 18] and differential fault intensity analysis (DFIA) [10, 11] are found to be more efficient in the presence of highly localized faults, such as single bit flips, or faults restricted to a given byte of the cipher state. While DFA attacks are possible using multiple byte faults, e.g. diagonal faults [22], the fault pattern impacts the complexity of key-recovery. In particular, with respect to AES-128, faults restricted to a single diagonal allow more efficient key-recovery as compared to faults spread across multiple diagonals. Similarly, DFIA typically exploits the bias of fault distribution at various fault intensities, under the assumption that the fault is restricted to a single byte/nibble of the cipher state [11]. Other techniques such as fault sensitivity analysis (FSA) [15, 17] require the knowledge of the critical fault intensity at which the onset of faulty behavior is observed. This critical value is then correlated with the secret-key dependent cipher state value. Finally, FIA techniques such as safe-error analysis (SEA) [3] and differential behavioral analysis (DBA) [21] require highly restrictive fault models such as stuck-at faults, where a specific target bit of the cipher state is set to either 0 or 1. In recent literature, microcontroller-based implementation of cryptographic algorithms have been subjected to instruction-skip attacks [7, 12], where the adversary uses precise injection techniques to transform the opcode for specific instructions into that for NOP (no-operation).

Similarity between FIA and SCA. The above discussion clearly reveals that existing FIA techniques are *inherently dependent* on the ability of an adversary to replicate a *specific fault model on an actual target device.* Fault precision and fault localization contribute to the efficiency of the attack, while the occurrence of random faults outside the target model generate *noisy ciphertexts*, thereby degrading the attack efficiency. Observe that this is conceptually similar to the effect of noise on the efficiency of traditional SCA techniques such as simple power analysis (SPA) and differential power analysis (DPA). In particular, the success rate for these techniques is directly proportional to the signal-to-noise ratio (SNR) of an implementation.

Our Motivation. In this paper, we aim to devise a generalized FIA strategy that overcomes the dependency of existing techniques on specific fault models. Rather than analyzing the behavior of the target implementation under a given set of faults, our approach would *learn* the behavior of the device-under-test (DUT) under an unrestricted set of fault injection parameters, irrespective of the fault nature. Such an attack strategy would allow a larger exploitable fault space, making it more powerful than all reported FIA techniques. As discussed next, an equivalent of the same approach in the context of SCA is well-studied in the literature.

1.2 Template Attacks: Maximizing the Power of SCA

Template attacks (TA) were proposed in [6] as the strongest form of SCA in an information-theoretic setting. Unlike other popular SCA techniques such as DPA, TA does not view the noise inherent to any cryptographic implementation as a hindrance to the success rate of the attack. Rather, it models precisely the noise pattern of the target device, and extracts the maximum possible information from any available leakage sample. This makes TA a threat to implementations otherwise secure based on the assumption that an adversary has access to only a limited number of side-channel samples. On the flip side, TA assumes that the adversary has full programming capability on a cryptographic device identical to the target black-box device.

1.3 Our Contribution: Templates for Fault Injection Analysis

The existing literature on TA is limited principally to SCA, exploiting passive leakages from a target cryptographic device for key recovery. In this paper, we aim to extend the scope of TA to active FIA attacks. Figure 1 summarizes our template-based FIA technique. Our approach is broadly divided into two main phases:

– The first phase of the attack is a *profiling phase*, where the adversary is assumed to have programming access to a device identical to the black-box target device. The adversary uses this phase to characterize the *fault behavior of the device* under varying fault injection intensities. We refer to such characterizations as the *fault template* for the device. We choose the *statistical distribution of faulty ciphertext values* under different fault injection intensities as the basis of our characterization. The templates are typically built on small-segments

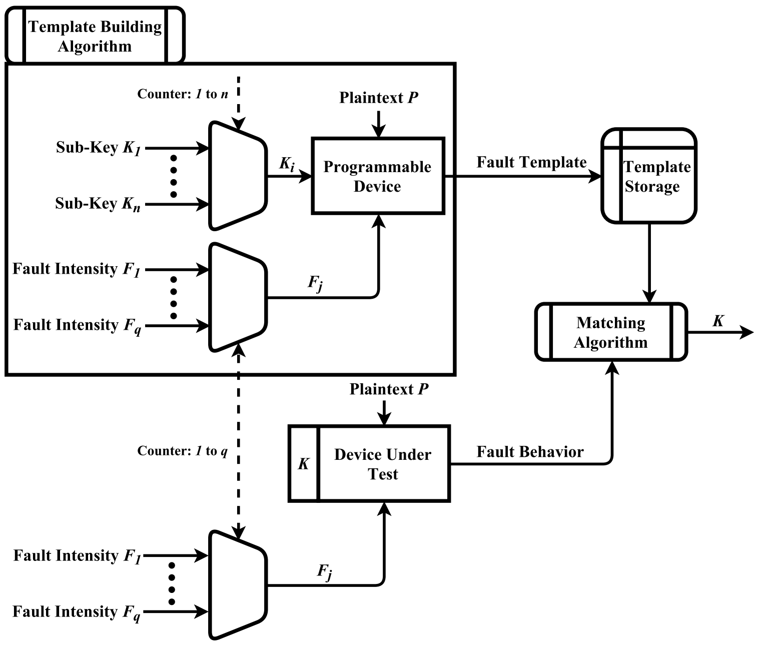


Fig. 1: Template-based Fault Injection Analysis: An Overview

of the overall secret-key, which makes a divide-and-conquer key recovery strategy practically achievable. Note that the matching phase does not require the correct ciphertext value corresponding to a given encryption operation.

– The second phase of the attack is the *matching phase*, where the adversary obtains the fault behavior of the target black-box device (with an embedded non-programmable secret-key K) under a set of fault injection intensities, and matches them with the templates obtained in the profiling phase to try and recover K. The idea is to use a maximum likelihood estimator-like distinguisher to identify the key hypothesis for which the template exhibits the maximum similarity with the experimentally obtained fault behavior of the target device.

1.4 Comparison with Existing FIA Techniques

In this section, we briefly recall existing FIA techniques, and explain their differences with our proposed template-based FIA approach. As already mentioned, our technique has two phases, and assumes that the adversary has programmable access to a device identical to the device under test. At the same time, it allows modeling the behavior of the device independent of specific fault models, as is done in most state-of-the-art FIA techniques. We explicitly enumerate these differences below.

Differential Fault Analysis (DFA): In DFA [9, 20, 13, 24], the adversary injects a fault under a specific fault model in target location of the cipher state, and analyzes the fault propagation characteristics using the knowledge of the fault-free and faulty ciphertexts. Our template-based FIA does not trace the propagation of the fault; rather it simply creates a template of the faulty ciphertext distribution under different fault injection intensities. This makes our approach independent of any specific fault model.

Differential Fault Intensity Analysis (DFIA): DFIA [11, 19] exploits the underlying bias of any practically achieved fault distribution on the target device, once again under a chosen fault model. It is similar in principle to DPA in the sense that it chooses the most likely secret-key value based upon a statistical analysis of the faulty intermediate state of the block cipher, derived from the faulty ciphertext values only. Our template-based FIA can be viewed as a generalization of DFIA with less stringent fault model requirements. Similar to DFIA, our approach also does not require the correct ciphertext values. However, our approach does not statistically analyze the faulty intermediate state based upon several key hypotheses. Rather, it pre-constructs separate templates of the *faulty ciphertext distribution* for each possible key value, and matches them with the experimentally obtained faulty ciphertext distribution from the black-box target device. Rather than focusing on specific fault models, the templates are built for varying fault injection intensities.

Fault Sensitivity Analysis (FSA): FSA [15, 17] exploits the knowledge of the critical fault intensity under which a device under test starts exhibiting faulty output behavior. The critical intensity is typically data-dependent, which allows secret-key recovery. FSA does not use the values of either the correct or the faulty ciphertexts. However, it requires a precise modeling of the onset of faults on the target device. Our methodology, on the other hand, uses the faulty ciphertext values, and is free of such precise critical fault intensity modeling requirements.

Safe Error Analysis (SEA): In SEA [3, 21], the adversary injects a fault into a precise location of the cipher state, and observes the corresponding effect on the cipher behavior. A popular fault model used in such attacks is the stuck-at fault model. The adversary injects a fault to set/reset a bit of the cipher state, and infers from the nature of the output if the corresponding bit was flipped as a result of the fault injection. Quite clearly, this fault model is highly restrictive. Our approach, on the other hand, allows random fault injections under varying fault intensities, which makes easier to reproduce in practice on real-world target devices.

2 Template-Based FIA: Detailed Approach

In this section, we present the details of our proposed template-based FIA. Given a target device containing a block cipher implementation, let $\mathcal{F}$ be the space of all possible fault intensities under which an adversary can inject a fault on this device. Now, assume that a random fault is injected in a given-segment S_k of the cipher state under a fault intensity $F_j \in \mathcal{F}$. Also assume that this state segment has value $P_{i'} \in \mathcal{P}$, and subsequently combines with a key segment $K_i \in \mathcal{K}$, where $\mathcal{P}$ and $\mathcal{K}$ are the space of all possible intermediate state values and key segment values respectively, resulting in a faulty ciphertext segment $C_{i,i',j,k}$. The granularity of fault intensity values depends on the injection equipment used - precise injection techniques such as laser pulses are expected to offer higher granularity levels than simpler injection techniques such as clock/voltage glitches. Note that we do not restrict the nature of the faults resulting from such injections to any specific model, such as single bit/single byte/stuck-at faults. With these assumptions in place, we now describe the two phases - the template building phase and the template matching phase - of our approach.

2.1 Template Building Phase

In this phase, the adversary has programmable access to a device identical to the device under test. By programmable access, we mean the following:

- The adversary can feed a plaintext P and master secret-key K of his choice to the device.
- Upon fault injection under a fault intensity $F_j \in \mathcal{F}$, the adversary can detect the target location S_k in the cipher state where the fault is induced

– The adversary has the knowledge of the corresponding key segment $K_i \in \mathcal{K}$ and the intermediate state segment $P_{i'} \in \mathcal{P}$. The key segment combines with the faulty state segment to produce the faulty ciphertext segment $C_{i,i',j,k}$.

Algorithm 1 Template Building Phase

```
Require: Programmable target device
Require: Target block cipher description
Ensure: Fault template $T$ for the target device
 1: Fix the set $\mathcal{S}$ of fault locations to be covered for successful key recovery depending
    on the block cipher description
 2: Fix the space $\mathcal{F}$ of fault injection intensities depending on the device characteristics
 3: Fix the number of fault injections $N$ for each fault intensity
 4: $T \leftarrow \phi$
 5: for each fault location $S_k \in \mathcal{S}$ do
 6:     for each corresponding intermediate state segment and key segment $(P_{i'}, K_i) \in$
    $\mathcal{P} \times \mathcal{K}$ do
 7:         for each fault injection intensity $F_j \in \mathcal{F}$ do
 8:             for each $l \in [1, N]$ do
 9:                 Run an encryption with plaintext segment $P_{i'}$ and the target key
    segment $K_i$ simultaneously
10:                 Inject a fault under intensity $F_j$ in the target location $S_k$
11:                 Let $C^l_{i,i',j,k}$ be the faulty ciphertext segment
12:             end for
13:             $T_{i,i',j,k} \leftarrow \left(C^1_{i,i',j,k}, \cdots, C^N_{i,i',j,k}\right)$
14:             $T \leftarrow T \cup T_{i,i',j,k}$
15:         end for
16:     end for
17: end for
18: return $T$
```

Let $C^1_{i,i',j,k}, \cdots, C^N_{i,i',j,k}$ be the faulty ciphertext outputs upon N independent fault injections in the target location S_k under fault injection intensity F_j, corresponding to the intermediate state segment $P_{i'}$ and key segment K_i. We refer to the tuple $T_{i,i',j,k} = \left(C^1_{i,i',j,k}, \cdots, C^N_{i,i',j,k}\right)$ as a *fault template instance*. This template instance is prepared and stored for possible tuples $(K_i, P_{i'}, F_j, S_k) \in \mathcal{K} \times \mathcal{P} \times \mathcal{F} \times \mathcal{S}$, where $\mathcal{S}$ is the set of all fault locations in the cipher state that need to be covered for full key-recovery. The set of all such template instances constitutes the *fault template* for the target device. Algorithm 1 summarizes the main steps of the template building phase as described above.

Note: The number of fault injections N required per fault intensity during the template building phase may be determined empirically, based upon the desired success rate of key recovery in the subsequent template matching phase. Quite evidently, increasing N improves the success rate of key recovery.

2.2 Template Matching Phase

Algorithm 2 Template Matching Phase

```
Require: Fault template $T$ corresponding to plaintext $P$
Ensure: The secret-key
 1: for each fault location $S_k \in \mathcal{S}$ do
 2:     for each fault injection intensity $F_j \in \mathcal{F}$ do
 3:         for each $l \in [1, N]$ do
 4:             Inject a fault under intensity $F_j$ in location $S_k$
 5:             Let ${C'}^{l}_{j,k}$ be the faulty ciphertext segment
 6:         end for
 7:         $E_{j,k} \leftarrow \left({C'}^{1}_{j,k}, \cdots, {C'}^{N}_{j,k}\right)$
 8:     end for
 9: end for
10: for each fault location $S_k \in \mathcal{S}$ do
11:     for each fault injection intensity $F_j \in \mathcal{F}$ do
12:         for each possible key hypothesis $K_i \in \mathcal{K}$ and intermediate state segment
    $P_{i'} \in \mathcal{P}$ do
13:             $\rho_{i,i',j,k} \leftarrow \mathcal{M}\left(E_{j,k}, T_{i,i',j,k}\right)$
14:         end for
15:     end for
16:     Store the pair $(K_i, P_{i'})$ pair such that $\sum_{F_j \in \mathcal{F}} \rho_{i,i'j,k}$ is maximum for the given
    fault location $S_k$.
17: end for
18: return the stored key hypothesis corresponding to each unique key segment loca-
    tion.
```

In this phase, the adversary has black-box access to the target device. Under the purview of black-box access, we assume the following:

- The adversary can feed a plaintext P of his choice to the device and run the encryption algorithm multiple times on this plaintext.
- Upon fault injection under a fault intensity $F_j \in \mathcal{F}$, the adversary can deduce the target location S_k in the cipher state where the fault is induced, by observing the corresponding faulty ciphertext $C'_{j,k}$.
- The adversary has no idea about the intermediate state segment $P_{i'}$ where the fault is injected, or the key segment K_i that subsequently combines with the faulty state segment to produce the ciphertext.

The adversary again performs N independent fault injections under each fault injection intensity F_j in a target location S_k, and obtains the corresponding faulty ciphertexts ${C'}^1_{j,k}, \cdots, {C'}^N_{j,k}$. All fault injections are performed during encryption operations using the same plaintext P as in the template building phase. These faulty ciphertexts are then given as input to a distinguisher $\mathcal{D}$. The distinguisher ranks the key-hypotheses $K_1, \cdots, K_n \in \mathcal{K}$, where the rank of K_i is estimated based upon the closeness of the experimentally obtained ciphertext distribution with the template instance $T_{i,i',j,k}$, for all possible intermediate state segments $P_{i'}$. The closeness is estimated using a statistical measure $\mathcal{M}$. The distinguisher finally outputs the key hypothesis K_i that is ranked consistently highly across all rank-lists corresponding to different fault injection intensities. Algorithm 2 summarizes our proposed template matching phase.

2.3 The Statistical measure M

An important aspect of the template matching phase is choosing the statistical measure M to measure the closeness of the experimentally observed faulty ciphertext segment distribution, with that corresponding to each template instance. We propose using a correlation-based matching approach for this purpose. The first step in this approach is to build a frequency-distribution table of each possible ciphertext segment value in each of the two distributions. Let the possible ciphertext segment values be in the range $[0, 2^{x-1}]$ where x is the number of bits in the ciphertext segment(for example, $[0, 255]$ for a byte, or $[0, 15]$ in case of a nibble). Also, let $f(y)$ and $f'(y)$ denote the frequency with which a given ciphertext segment value $y \in [0, 2^{x-1}]$ occurs in the template and the experimentally obtained distribution, respectively. Since there are exactly N sample points in each distribution, we have $\sum_{y\in[0,2^{x-1}]} f(y) = \sum_{y\in[0,2^{x-1}]} f'(y) = N$.

The next step is to compute the Pearson's correlation coefficient between the two distributions as:

$$\rho = \frac{\sum\limits_{y\in[0,2^{x-1}]} \left(f(y) - \frac{N}{2^x}\right) \cdot \left(f'(y) - \frac{N}{2^x}\right)}{\sqrt{\sum\limits_{y\in[0,2^{x-1}]} \left(f(y) - \frac{N}{2^x}\right)^2} \sqrt{\sum\limits_{y\in[0,2^{x-1}]} \left(f'(y) - \frac{N}{2^x}\right)^2}}$$

The Pearson's correlation coefficient is used as the measure M. The choice of statistic is based on the rationale that, for the correct key segment hypothesis, the template would have a similar frequency distribution of ciphertext segment values as the experimentally obtained set of faulty ciphertexts, while for a wrong key segment hypothesis, the distribution of ciphertext segment values in the template and the experimentally obtained ciphertexts would be uncorrelated.

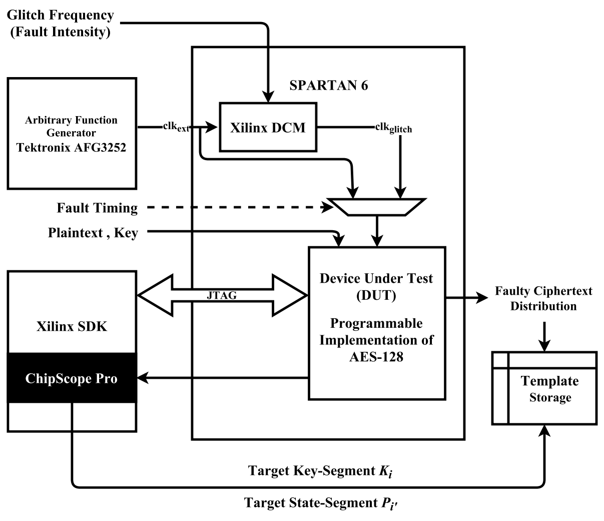
An advantage of the aforementioned statistical approach is that it can be extended to relaxed fault models such as multi-byte faults, that are typically not exploited in traditional FIA techniques. In general, if a given fault injection affects multiple locations in the block cipher state, the correlation analysis is simply repeated separately for each fault location. This is similar to the divide-and-conquer approach used in SCA-based key-recovery techniques.

3 Case Study: Template-Based FIA on AES-128

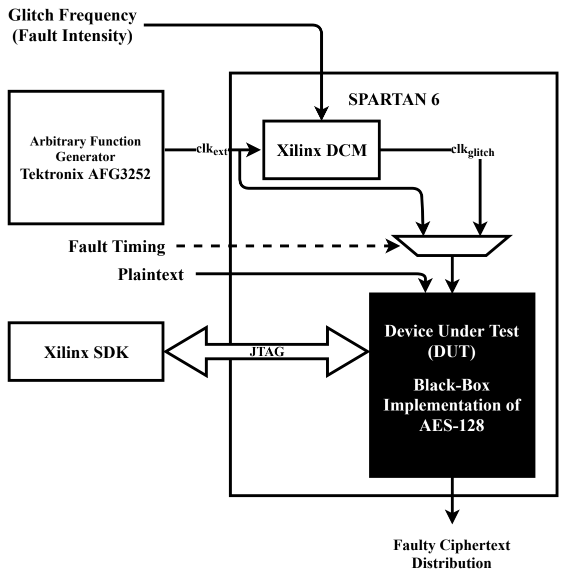
In this section, we present a concrete case study of the proposed template-based FIA strategy on AES-128. As is well-known, AES has a plaintext and key size of 128 bits each, and a total of 10 rounds. Each round except the last one comprises of a non-linear S-Box layer (16 S-Boxes in parallel), a linear byte-wise ShiftRow operation, and a linear MixColumn operation, followed by XOR-ing with the round key. The last round does not have a MixColumn operation. This in turn implies that if a fault was injected in one or more bytes of the cipher state after the 9^{th} round MixColumn operation, the faulty state byte (or bytes) combines with only a specific byte (or bytes) of the 10^{th} round key. For example, if a fault was injected in the first byte of the cipher state, the faulty byte would pass through the S-Box and ShiftRow operation, and combine with the first byte of the 10^{th} round key to produce the first byte of the faulty ciphertext. The exact relation between the fault injection location and the corresponding key segment depends solely on the ShiftRow operation, and is hence deterministic. This matches precisely the assumptions made in our attack description in the previous section. Consequently, this case study assumes that all faults are injected in the cipher state between the 9^{th} round MixColumn operation and the 10^{th} round S-Box operations. The aim of the fault attack is to recover byte-wise the whole 10^{th} round key of AES-128, which in turn deterministically reveals the entire secret-key. We note that fault injection in an earlier round will lead to extremely large templates, making the attack impractical.

3.1 The Fault Injection Setup

The fault injection setup (described in Figure 2) uses a Spartan 6 FPGA mounted on a Sakura-G evaluation board, a PC and an external arbi-



(a) Template Building Phase



(b) Template Matching Phase

Fig. 2: Experimental Set-Up

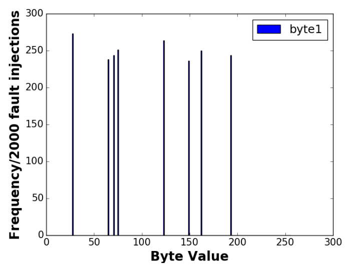
Table 1: Glitch Frequencies for Different Fault Models

Glitch Frequency (MHz)	**Faulty Bytes**	**Bit Flips per Byte**
125.3-125.5	1	1
125.6-125.7	1	2
125.8-126.0	1	3
126.1-126.2	2-3	1-3
> 126.2	> 3	> 5

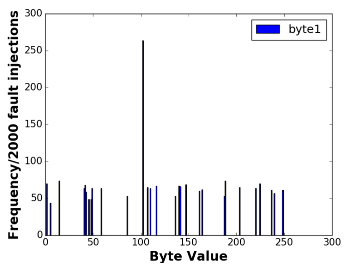
trary function generator (Tektronix AFG3252). The FPGA has a Device Under Test (DUT) block, which is an implementation of the block cipher AES-128. Faults are injected using clock glitches. The device operates normally under the external clock signal $\mathrm{clk}_{\mathrm{ext}}$. The glitch signal, referred to as $\mathrm{clk}_{\mathrm{fast}}$, is derived from the $\mathrm{clk}_{\mathrm{ext}}$ via a Xilinx Digital Clock Manager (DCM) module. The fault injection intensity in our experiments is essentially the glitch frequency, and is varied using a combination of the DCM configuration, and the external function generator settings. In the template building phase, the intermediate cipher state $P_{i'}$ and the intermediate round key K_i are monitored using a ChipScope Pro analyzer, while in the template matching phase, the DUT is a black box with no input handles or internal monitoring capabilities. Table 1 summarizes the glitch frequency ranges at which these fault models were observed on the target device.

3.2 Templates for Single Byte Faults

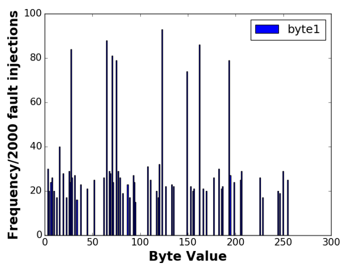
In this section, we present examples of fault templates obtained from the device under test, for glitch frequencies that result in single byte fault injections in the AES-128 module. Since only a single byte is affected between the 9th round MixColumn operation and the 10th round S-Box operations, we are interested in the distribution of the corresponding faulty byte in the ciphertext. Figure 3 presents fault templates containing ciphertext byte distributions for three categories of faults - single bit faults, two-bit faults, and three-bit faults. The templates correspond to the same pair of intermediate state byte and last round key byte for an AES-128 encryption. Quite evidently, the ciphertext distribution for each template reflects the granularity of the corresponding fault model. In particular, for a single bit fault, most of the faulty ciphertext bytes assume one of 8 possible values, while for three-bit faults, the ciphertext bytes assume more than 50 different values across all fault injections. In all cases, however, the distribution of ciphertext values is non-uniform,



(a) Single bit faults: 125.3-125.5 MHz



(b) Two-bit faults: 125.5-125.7 MHz



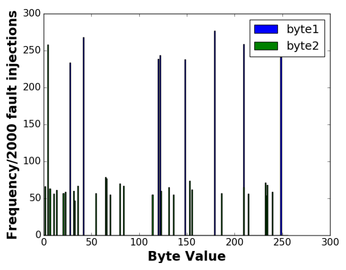
(c) Three-bit faults: 125.7-126.0 MHz

Fig. 3: Templates for Single Byte Faults: Distribution of Faulty Ciphertext Byte for Different Fault Injection Intensities

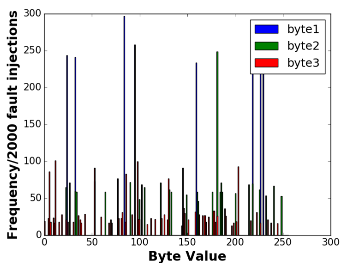
which provides good scope for characterizing the fault behavior of the device in the template building phase.

3.3 Templates for Multi-Byte Faults

In this section, we present examples of fault templates constructed for glitch frequencies that result in multi-byte fault injections. Figure 4 shows the distribution of different bytes injected with different faults. It is interesting to observe that at the onset of multi-byte faults, the distribution of faulty ciphertext bytes is *not uniformly random*; indeed, it is possible to characterize the fault behavior of the device in terms of templates under such fault models. Given the absence of MixColumn operation in the last round of AES, each faulty intermediate state byte combines independently with a random last round key byte. This allows a divide-and-conquer template matching approach, where the statistical analysis may be applied to each faulty ciphertext byte independently. This is a particularly use-



(a) 1-bit, 2-bit fault in 2 bytes: 126.1 MHz



(b) 1-bit, 2-bit, 3-bit fault across 3 bytes: 126.2 MHz

Fig. 4: Templates for Multi-Byte Faults: Distribution of Multiple Faulty Ciphertext Byte Values

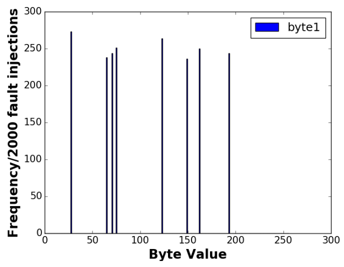
ful mode of attack, since it can be launched even without precise fault injection techniques that allow targeting a single byte of the cipher state.

3.4 Variation with Key Byte Values

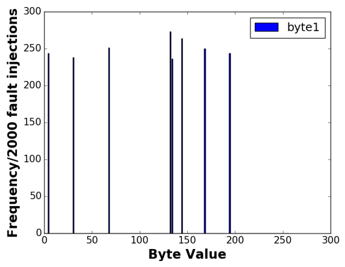
The success of our template matching procedure with respect to AES-128 relies on the hypothesis that for different key byte values, the ciphertext distribution corresponding to the same fault location is different. Otherwise, the key recovery would be ambiguous. We validated this hypothesis by examining the ciphertext distribution upon injecting a single bit fault in the first byte of the cipher state, corresponding to different key byte values. We illustrate this with a small example in Figure 5. Figures 5a, 5b, 5c and 5d represent the frequency distributions for faulty ciphertext byte corresponding to the same intermediate byte value of `0x00`, and key byte values `0x00`, `0x01`, `0x02` and `0x03`, respectively. Quite evidently, the three frequency distributions are unique and mutually non-overlapping. The same trend is observed across all 256 possible key byte values; exhaustive results for the same could not be provided due to space constraints.

3.5 Template matching for Key-Recovery

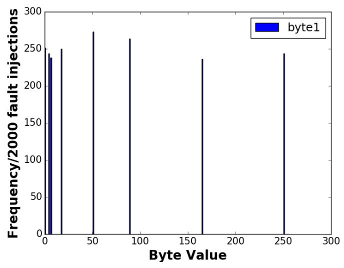
In this section, we present results for recovering a single key-byte for AES-128 under various fault granularities. As demonstrated in Figure 6, the correlation for the correct key hypothesis exceeds the average correlation over all wrong key hypotheses, across the three fault models - single bit faults, two-bit faults and three-bit faults. As is expected, precise single-bits faults within a given byte enable distinguishing the correct



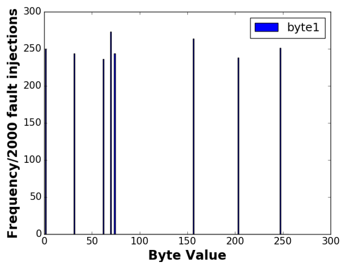
(a) Target Key Byte = 0x00



(b) Target Key Byte = 0x01



(c) Target Key Byte = 0x02



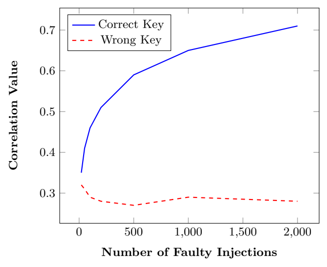
(d) Target Key Byte = 0x03

Fig. 5: Frequency Distributions for Faulty Ciphertext Byte : Same Intermediate State Byte but Different Key Byte Values

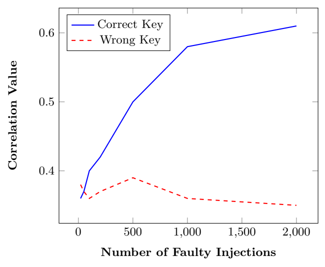
key hypothesis using very few number of fault injections (50-100); for less granular faults such as three-bit faults, more number of fault injections (200-500) are necessary. Finally, the same results also hold for multi-byte fault models, where each affected byte encounters a certain number of bit-flips. Since the key-recovery is performed byte-wise, the adversary can use the same fault instances to recover multiple key bytes in parallel.

4 Conclusion

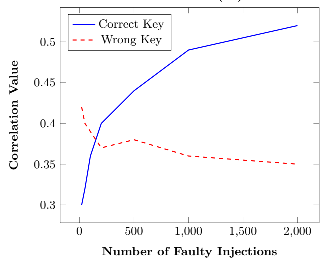
We presented the first template based fault injection analysis of block ciphers. We presented a generic algorithm comprising of a template building and a template matching phase, that can be easily instantiated for any target block cipher. The templates are built on pairs of internal state segment and key segment values at different fault intensities, while the number of fault instances per template depends on the statistical methodology used in the matching phase. In this paper, we advocated the use of the Pearson correlation coefficient in the matching phase; exploring alternative techniques in this regard is an interesting future work. In order



(a) Single Bit Faults



(b) Two-Bit Faults



(c) Three-Bit Faults

Fig. 6: Correlation between template and Observed Ciphertext Distribution: Correct Key Hypothesis v/s Wrong Key Hypothesis

to substantiate the effectiveness of our methodology, we presented a case-study targeting a hardware implementation of AES-128 on a Spartan-6 FPGA. Interestingly, our attack allowed exploiting even low-granularity faults such as multi-byte faults, that do not require high precision fault injection equipment. It may be emphasized that the attack is devoid of the exact knowledge of the underlying fault model. Such fault models also allowed parallel recovery of multiple key-bytes, thus providing a trade-off between the number of fault injections, and the number of recovered key-bytes. An interesting extension of this work would be apply template-based analysis against implementations with fault attack countermeasures such as spatial/temporal/information redundancy.

Acknowledgements

We would like to thank the anonymous reviewers for providing constructive and valuable comments. Debdeep would also like to thank his DST Swarnajayanti fellowship (2015-16) for partial support. He would also like to thank DRDO, India for funding the project, "Secure Resource - constrained communication Framework for Tactical Networks using Physically Unclonable Functions (SeRFPUF)" for partially supporting the research. He would also like to thank Information Security Education Awareness (ISEA), DIT, India for encouraging research in the area of computer security. Sikhar would like to thank Qualcomm India Innovation Fellowship 2017-18.

References

1. Agoyan, M., Dutertre, J.M., Naccache, D., Robisson, B., Tria, A.: When Clocks Fail: On Critical Paths and Clock Faults. Smart Card Research and Advanced Application pp. 182–193 (2010)
2. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pelosi, G.: A fault induction technique based on voltage underfeeding with application to attacks against aes and rsa. Journal of Systems and Software **86**(7), 1864–1878 (2013)
3. Blömer, J., Seifert, J.P.: Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In: Wright, R.N. (ed.) Financial Cryptography, Lecture Notes in Computer Science, vol. 2742, pp. 162–181. Springer (2003)
4. Canivet, G., Clédière, J., Ferron, J.B., Valette, F., Renaudin, M., Leveugle, R.: Detailed analyses of single laser shot effects in the configuration of a virtex-ii fpga. In: On-Line Testing Symposium, 2008. IOLTS'08. 14th IEEE International. pp. 289–294. IEEE (2008)
5. Canivet, G., Maistri, P., Leveugle, R., Clédière, J., Valette, F., Renaudin, M.: Glitch and laser fault attacks onto a secure aes implementation on a sram-based fpga. Journal of Cryptology **24**(2), 247–268 (2011)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 13–28. Springer (2002)
7. Choukri, H., Tunstall, M.: Round reduction using faults. FDTC **5**, 13–24 (2005)
8. Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of aes. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on. pp. 7–15. IEEE (2012)
9. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on aes. In: Applied Cryptography and Network Security. pp. 293–306. Springer (2003)
10. Fuhr, T., Jaulmes, E., Lomné, V., Thillard, A.: Fault attacks on aes with faulty ciphertexts only. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on. pp. 108–118. IEEE (2013)
11. Ghalaty, N.F., Yuce, B., Taha, M., Schaumont, P.: Differential fault intensity analysis. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on. pp. 49–58. IEEE (2014)
12. Heydemann, K., Moro, N., Encrenaz, E., Robisson, B.: Formal verification of a software countermeasure against instruction skip attacks. In: PROOFS 2013
13. Kim, C.H.: Differential fault analysis against aes-192 and aes-256 with minimal faults. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop on. pp. 3–9. IEEE (2010)
14. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in cryptology—CRYPTO'99. pp. 789–789. Springer (1999)
15. Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta, K.: Fault sensitivity analysis. In: CHES. vol. 6225, pp. 320–334. Springer (2010)

16. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards, vol. 31. Springer Science & Business Media (2008)
17. Mischke, O., Moradi, A., Güneysu, T.: Fault sensitivity analysis meets zero-value attack. In: 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, Busan, South Korea, September 23, 2014. pp. 59–67 (2014). https://doi.org/10.1109/FDTC.2014.16, http://dx.doi.org/10.1109/FDTC.2014.16
18. Mukhopadhyay, D.: An improved fault based attack of the advanced encryption standard. Africacrypt **5580**, 421–434 (2009)
19. Patranabis, S., Chakraborty, A., Nguyen, P.H., Mukhopadhyay, D.: A biased fault attack on the time redundancy countermeasure for aes. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 189–203. Springer (2015)
20. Piret, G., Quisquater, J.J.: A differential fault attack technique against spn structures, with application to the aes and khazad. In: CHES. vol. 2779, pp. 77–88. Springer (2003)
21. Robisson, B., Manet, P.: Differential behavioral analysis. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 413–426. Springer (2007)
22. Saha, D., Mukhopadhyay, D., Chowdhury, D.R.: A diagonal fault attack on the advanced encryption standard. IACR Cryptology ePrint Archive **2009**, 581 (2009)
23. Selmane, N., Guilley, S., Danger, J.L.: Practical setup time violation attacks on aes. In: Dependable Computing Conference, 2008. EDCC 2008. Seventh European. pp. 91–96. IEEE (2008)
24. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. WISTP **6633**, 224–233 (2011)