Improved Fault Templates of Boolean Circuits in Cryptosystems can Break Threshold Implementations

Debdeep Mukhopadhyay,
Department of Computer Science and Engineering,
Indian Institute of Technology Kharagpur
West Bengal, India
Email: debdeep@cse.iitkgp.ac.in

Abstract

Fault Template Analysis (FTA) has been shown as a powerful tool for attacking cryptosystems and exposing vulnerabilities which were previously not reported in existing literature. Fault templates can be utilized for attacking block ciphers in middle rounds which were known prior to be resistant against fault attacks. In this paper we revisit the potent of fault templates and show a more systematic methodology to develop fault templates of Boolean circuits using a well known concept in design verification, namely positive Davio's decomposition. We show that the improved FTAs, called FTA2.0, can be used to fault analyze block ciphers in the middle rounds using as few as two bit-flip faults. Further, it can be used to attack TI-implemented block ciphers by considering a Double Bit Upset (DBU) fault in a target share bit. The attack shows that varying the latency of the fault the adversary can obtain unmasked bits and can recover the secret key.

1 Introduction

Faults have been shown to be catastrophic to the security of cryptosystems. The seminal paper of Kocher exposed that faults can be catastrophic for cryptosystems. Subsequently several fault based cryptanalysis on ciphers have been developed to show that faults are a very powerful attack vector. The potent of fault based cryptanalysis arose from the development in two directions of research: i) There was a huge development of accurate and reasonably affordable fault injection tools. Various fault injection methods based on clock/voltage glitches, electromagnetic pulse injections, and laser shots are some popular techniques of fault injections on crypto-devices like FPGAs, smart-cards etc. ii) The fault based crptanalysis algorithms on crypto-standards like the Advanced Encryption Standard (AES) were significantly improved over prior research to eventually take around a single fault to deduce the key [1]. Different fault analysis techniques, like Differential Fault Analysis (DFA), Differential Fault Intensity Attacks (DFIA), and Statistical Fault Intensity (SIFA), Persistent Fault Attacks (PFA) were developed and show that not-only crypto-implementations, but even countermeasures like redundancy, concurrent error detection (CED), Infective countermeasures and the like can also be thwarted. using fault injections. A common aspect of these fault attack techniques are that all of them are non-profiled attacks and require access to faulty ciphertexts (also the fault-free ciphers in case of the classical DFA). This often restricts the attacks on the starting or last rounds. Recently, the first profiled fault attack, called Fault Template Attack (FTA) was proposed [2]. FTA is based on fault propagation characteristic which is dependent on the inputs to a Boolean circuit and is defined in a fault template collected in a profiling phase. Subsequently, the template is used in a template matching phase while launching the attack on a target device to recover the internal states and secret key of the victim cryptosystem. Notably, FTA does not require access to the ciphertext, a property which is useful to perform middle round attacks which are not attainable by other fault attacks. FTA has been applied successfully on masking countermeasures, including mathematically sound TI protections. In this paper, we first make the fault template building phase for Boolean circuits more systematic and name it FTA2.0. Subsequently, we delve into analyzing TI-protected schemes of popular ciphers like PRESENT and show how they can leak the actual unshared inputs in the face of the improved FTA2.0 method. In particular, we show that the ability to flip a register storing an inner round share bit twice during an encryption following a Double Bit Upset (DBU) fault model[1] can leak the unshared inputs. Interestingly, we show that a control on latency of the fault pattern can reveal all the state-bits even without knowledge of the input and outputs of the inner round. This makes FTA2.0 successful on most software implementations of such TI-protected schemes where the computations of the inner round output shares are evaluated in multiple cycles.

2 Fault Model

The manifestation of faults in electronic circuits are captured. by fault models. Proper definition of fault models are needed to analyze the threats of these faulty behavior on the secret computations. The fault model that we address here in the paper are transient faults which are more pertinent than permanent faults wrt. fault attacks. They are short-lived and thus they are more stealthy compared. to permanent faults which can render the device unusable thus making the attack less threatening. The transient faults are characterized by i) Fault Injection Time, denoted as t_{inj} which is the time instance when the value of a target register changes from its correct configuration, and ii) Fault Release Time, denoted as $t_{release}$, which is the instance of time when the faulty configuration of the target register is released. If the target register r is a single

[1]Modern day lasers can create such fault patterns at bit-level in registers

bit entity, then on release it returns back to its correct bit value as before t_{inj}. The fault duration is denoted as $d = t_{release} - t_{inj}$, and we consider such faults at the granularity of clock cycles. Thus, we denoted any fault as the following notation, $F(r, t_{inj}, t_{release})$.

3 Boolean Functions and FTA2.0

Let f be a Boolean function with n-variables $x_1, x_2, \cdots, x_i, \cdots, x_n$. By using positive Davio decomposition we can write,

$$f = f_{\overline{x_i}} + x_i \frac{\partial f}{\partial x_i}$$

Here,

$$\begin{aligned} f_{\overline{x_i}} &= f(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n) \\ \frac{\partial f}{\partial x_i} &= f(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n) \oplus f(x_1, \cdots, x_{i-1}, 1, \\ &\quad x_{i+1}, \cdots, x_n) \end{aligned}$$

Consider, a stuck-at-1 fault in the variable x_i. Since, $f_{\overline{x_i}}$ does not contain x_i, this part will not corrupt the output. So, the fault will propagate to the output iff $x_i = 0$, and $\frac{\partial f}{\partial x_i} = 1$.

As a concrete example, we consider the first output bit of the S-Box for PRESENT as shown in Eq. (1). This S-Box has 4 input bits denoted as x_1, x_2, x_3, x_4 and 4 output bits y_1, y_2, y_3, y_4 (where x_1 and y_1 are the Most Significant Bits (MSB) and x_4 and y_4 are the Least Significant Bits (LSB)).

$$y_1 = x_1x_2x_4 + x_1x_3x_4 + x_1 + x_2x_3x_4 + x_2x_3 + x_3 + x_4 + 1 \tag{1}$$

Applying, the above decomposition by target x_1 we have:

$$\begin{aligned} f_{\overline{x_1}} &= x_2x_3x_4 + x_2x_3 + x_3 + x_4 + 1 \\ \frac{\partial f}{\partial x_1} &= x_2x_4 + x_3x_4 + 1 \end{aligned}$$

Now, consider a stuck-at-1 fault at x_1. The fault is propagated to the output y iff $x_1 = 0$ and $x_2x_4 + x_3x_4 = 0 \Rightarrow x_4 = 0$, or $x_2 + x_3 = 0$.

Thus, the possible values of (x_1, x_2, x_3, x_4) can be any of the following states: $(0,0,0,0), (0,0,1,0), (0,1,0,0), (0,1,1,0), (0,0,0,1), (0,1,1,1)$.

Likewise, if we expand the function on the variable x_2, we have:

$$\begin{aligned} f_{\overline{x_2}} &= x_1x_3x_4 + x_1 + x_3 + x_4 + 1 \\ \frac{\partial f}{\partial x_2} &= x_1x_4 + x_3x_4 + x_3 \end{aligned}$$

In this case, if we consider a stuck-at-1 fault at x_2, then the fault propagates to the output iff $x_2 = 0$ and $x_1x_4 + x_3x_4 + x_3 = 1$ ie. $(x_1, x_2, x_3, x_4) = (0, 0, 1, 0), (1, 0, 0, 1), (1, 0, 1, 0), (1, 0, 1, 1)$.

Note, that if the fault propagates in both the cases we intersect the above possible values of (x_1, x_2, x_3, x_4), and we have $(0, 0, 1, 0)$. Thus the leakage of the input state leaks by observing only whether. the output is faulty.

In the worst case, we may need 4 fault injections. This is because when the fault injected at a variable x_i propagates to the output it leads to the attacker obtaining a conjunction in the variables which can be generalized as: $x_i = 0 \wedge \frac{\partial f}{\partial x_i} = 1$. It may be observed that when there is no fault propagation we have a disjunction, namely: $x_i = 1 \vee \frac{\partial f}{\partial x_i} = 0$. Since, we have 4 variables the minimal faults would be required as when the fault propagation occurs and we have equations which simplify to two equations with two variables which can be trivially solved. However, when the fault propagation does not occur, in the worst case-scenario, every fault injection will lead to a disjunction, which may leak either one of the input variables or another equation with three variables ie. without the variable on which the equation has been pivoted. Hence, one can observe that 4 fault injections are sufficient for the retrieval of all the possible input values.

NB: This also shows that our technique of FTA (Fault Template Attacks) is significantly different from Safe Error Attacks (SEA). In SEA the attack would require always 4 fault injections to retrieve the internal state of 4 bits. However, in case of FTA, the number of faults required can be 2, as we are considering the propagation criteria of the Boolean circuit which is not considered by FTA.

We call the fault propagation template as the Fault Template 2.0 (or FTA2.0). FTA2.0 can be characterized by the vector $S[n]$ for an n-variable Boolean function. Here $S[n]$ is a 0-1 vector which defines whether the fault propagates to the output bit. Each bit, $S[i] = 1 \Rightarrow (x_i = 0) \wedge \frac{\partial f}{\partial x_i} = 1$, while $S[i] = 0 \Rightarrow (x_i = 1) \vee \frac{\partial f}{\partial x_i} = 0$. The solution for the variable $x = (x_0, \cdots, x_{n-1})$ is thus a solution of this system of equations.

4 Application of FTA2.0 to Masked Circuits

For the sake of illustration, we first present the unshared version of F (Eq. (2)), and then the shares corresponding to it (Eq. (3)). Note that, in Eq. (2) x_0 denote the LSB and x_3 denote the MSB.

$$\begin{aligned} &F(x_3, x_2, x_1, x_0) = (f_3, f_2, f_1, f_0) \\ &\qquad f_3 = x_2 + x_1 + x_0 + x_3x_0; f_2 = x_3 + x_1x_0; f_1 = x_2 + x_1 + x_3x_0; \\ &\qquad f_0 = x_1 + x_2x_0. \end{aligned} \tag{2}$$

$$\begin{aligned} f_{10} &= x_1^2 + x_2^2x_0^2 + x_2^2x_0^3 + x_2^3x_0^2 & f_{11} &= x_2^2 + x_1^2 + x_3^2x_0^2 + x_3^2x_0^3 + x_3^3x_0^2 \\ f_{20} &= x_1^3 + x_2^3x_0^3 + x_2^1x_0^3 + x_2^3x_0^1 & f_{21} &= x_2^3 + x_1^3 + x_3^3x_0^3 + x_3^1x_0^3 + x_3^3x_0^1 \\ f_{30} &= x_1^1 + x_2^1x_0^1 + x_2^1x_0^2 + x_2^2x_0^1 & f_{31} &= x_2^1 + x_1^1 + x_3^1x_0^1 + x_3^1x_0^2 + x_3^2x_0^1 \end{aligned}$$

$$f_{12} = x_3^2 + x_1^2 x_0^2 + x_1^2 x_0^3 + x_1^3 x_0^2 \qquad f_{13} = x_2^2 + x_1^2 + x_0^2 + x_3^2 x_0^2 + x_3^2 x_0^3 + x_3^3 x_0^2$$
$$f_{22} = x_3^3 + x_1^3 x_0^3 + x_1^1 x_0^3 + x_1^3 x_0^1 \qquad f_{23} = x_2^3 + x_1^3 + x_0^3 + x_3^3 x_0^3 + x_3^1 x_0^3 + x_3^3 x_0^1$$
$$f_{32} = x_3^1 + x_1^1 x_0^1 + x_1^1 x_0^2 + x_1^2 x_0^1 \qquad f_{33} = x_2^1 + x_1^1 + x_0^1 + x_3^1 x_0^1 + x_3^1 x_0^2 + x_3^2 x_0^1$$

4.1 Target Implementation

It is reasonable to assume that the target implementation of the threshold S-box would be a multi-cycle design. We assume that we have the following 12-clock cycles to compute the protected S-box.

$$\begin{array}{c} T_0 : f_{10} \rightarrow T_1 : f_{11} \rightarrow T_2 : f_{12} \rightarrow T_3 : f_{13} \rightarrow T_4 : f_{20} \rightarrow T_5 : f_{21} \\ \downarrow \\ T_6 : f_{22} \rightarrow T_7 : f_{23} \rightarrow T_8 : f_{30} \rightarrow T_9 : f_{31} \rightarrow T_{10} : f_{32} \rightarrow T_{11} : f_{33} \end{array}$$

The clock cycles are shown by T_0 to T_{11}, each output share being computed in a different clock cycle.

4.2 Fault Location and Attack Outline

We consider a template building phase [2] wherein the faults are targeted at the registers storing the masks x_0^3 and x_2^1. In order to build the templates for obtaining the various bits of a nibble of the PRESENT cipher we consider the following faults:

1. x_0 by considering a fault pattern $F_0(x_2^1, T_4, T_8)$
2. x_1 by considering a fault pattern $F_1(x_0^3, T_2, T_6)$
3. x_2 by considering a fault pattern $F_2(x_0^3, T_0, T_4)$
4. x_3 by considering a fault pattern $F_3(x_0^3, T_1, T_5)$

We explain how these fault campaigns work to retrieve the key bits of the nibbles of the cipher state. We start with discussion on the fault F_2.

4.3 Building the Templates

We demonstrate how to develop a template to obtain the nibble bit x_2. Let us consider a fault in the share x_0^3 as per the fault model $F_2(x_0^3, T_0, T_4)$. This implies that we assume the fault to create a bit-flip in register x_0^3 at time T_0 and follow it with a release of the fault at time T_4.

Note that a permanent fault in one of the shares for x_0 will always result in an effective fault in the final computation. However as we do not keep the fault for the entire duration but rather keep the fault only during the computations $f_{10}, f_{11}, f_{12}, f_{13}, f_{20}$ as entailed in the fault model F_1, the fault may conditionally propagate depending on the nibble bits.

It may be emphasized that it is quite practical to assume that the attacker can control the fault timing to affect this. We trigger a laser gun a fixed time t_0 before the start of the computation for the share f_{10} at time T_0 when the injection is needed. The injection causes a bit flip in the register holding the share x_0^3, and it creates another bit flip in the same register at a time t_1 after the first injection point. This can be obtained by standard fault injection lasers even with a set-up with a single laser, and in particular using a more costly set-up which has double lasers (which can target simultaneously two fault locations - which in this case is the same target).

4.3.1 Leakage of x_2

Thus, we compute the partial derivatives wrt. x_0^3 to determine the conditions on which the output shares would be faulted.

$$\begin{aligned}
\frac{\partial f_{10}}{\partial x_0^3} &= x_2^2 \\
\frac{\partial f_{11}}{\partial x_0^3} &= x_3^2 \\
\frac{\partial f_{12}}{\partial x_0^3} &= x_1^2 \\
\frac{\partial f_{13}}{\partial x_0^3} &= x_3^2 \\
\frac{\partial f_{20}}{\partial x_0^3} &= (x_2^3 + x_2^1)
\end{aligned}$$

Thus, the fault propagates to the output iff f_0, or f_1, or f_2, or f_3 is faulted. Thus, we have $\frac{\partial f_0}{\partial x_0^3} = x_2$, $\frac{\partial f_1}{\partial x_0^3} = x_3^2$, $\frac{\partial f_2}{\partial x_0^3} = x_1^2$, $\frac{\partial f_3}{\partial x_0^3} = x_3^2$.

Thus, we see the fault always propagates to the output iff $x_2 = 1$ or $x_3^2 = 1$ or $x_1^2 = 1$. Note, that if we fix the input x, the bit $x_2 = 1$ occurs with probability 0 or 1, but the other (share) bits are true with a probability of 0.5 (as they are randomly chosen on every encryption). Thus, if $x_2 = 1$, the fault is always propagated, else, sometimes the fault would not be propagated. This leaks x_2. This can be suitably performed to leak other bits too.

4.3.2 Leakage of x_3

Now consider the fault $F_3(x_0^3, T_1, T_5)$. This keeps the target register x_0^3 same but the duration is shifted to T_1 to T_5. Thus the transient fault affects the computations $f_{11}, f_{12}, f_{13}, f_{20}, f_{21}$.

Thus, in order to comprehend the conditions for the fault to propagate we compute:

$$\begin{aligned}
\frac{\partial f_0}{\partial x_0^3} &= x_2^3 + x_2^1 \\
\frac{\partial f_1}{\partial x_0^3} &= x_3 \\
\frac{\partial f_2}{\partial x_0^3} &= x_1^2 \\
\frac{\partial f_3}{\partial x_0^3} &= x_3^2
\end{aligned}$$

Thus, again the fault propagates to the output iff $x_3 = 1$ or $x_2^3 + x_2^1 = x_2^2 = 1$ or $x_1^2 = 1$ or $x_3^2 = 1$, thus leaking the value of x_3.

4.3.3 Leakage of x_1

Similarly, we can control the fault timing to engulf the computations $f_{12}, f_{13}, f_{20}, f_{21}, f_{22}$ by considering a fault campaign in the model $F_1(x_0^3, T_2, T_6)$. Again note that this is a shifted duration of the fault in the same target register x_0^3. Thus, we compute the following partial derivatives to determine the conditions for fault propagation:

$$\begin{aligned}
\frac{\partial f_0}{\partial x_0^3} &= x_2^3 + x_2^1 \\
\frac{\partial f_1}{\partial x_0^3} &= x_3^3 + x_3^1 \\
\frac{\partial f_2}{\partial x_0^3} &= x_1 \\
\frac{\partial f_3}{\partial x_0^3} &= x_3^2
\end{aligned}$$

In this case the fault propagates to the output iff $x_1 = 1$ or $x_2^3 + x_2^1 = x_2^2$ or $x_3^3 + x_3^1 = x_3^2 = 1$, thus leaking the value of x_1.

4.3.4 Leakage of x_0

For x_0, we need to consider the fault in a different share register namely x_2^1 according to the model $F_0(x_2^1, T_4, T_8)$. This creates the first flip during the computation of f_{20} and then persist in the sequence $f_{20}, f_{21}, f_{22}, f_{23}, f_{30}$ before being released.

Like before we compute the following to determine the conditions for fault propagation to the output:

$$\begin{aligned}
\frac{\partial f_0}{\partial x_2^1} &= x_0 \\
\frac{\partial f_1}{\partial x_2^1} &= 0 \\
\frac{\partial f_2}{\partial x_2^1} &= 0 \\
\frac{\partial f_3}{\partial x_2^1} &= 0
\end{aligned}$$

Thus, depending on the value of x_0, the fault propagates always or never, clearly leaking the value of x_0.

4.3.5 Fault Template for the TI S-box

Based on the above analysis we can define the fault template wherein one can subject the TI-implemented S-Box to the above 4 faults $F_0 - F_3$, and relate the inputs to the fault propagation. One can capture the template ' by a table with 16 entries, where each entry represents whether the fault propagates when subjected to the above faults. To explain for a given input $x = x_3, x_2, x_1, x_0$, we denote the entry of the template table $T[x] = (p_0, p_1, p_2, p_3)$ (Table 2). Here, p_i denotes the expected number of times the fault according to model F_i creates a faulty output, where $0 \leq i \leq 3$. For each fault campaign we try say N times. It is evident $p_i \in \{0, \cdots, n-1\}$. It may be noted that the templates induces partitions on the input space. It can be observed that the fault propagation template induces partitions on the input space which can be denoted as $(0),(1),(2),(3),(4),(5),(6),(7),(8,10),(9,11),(12,14),(13,15)$.

In order to make the partitions completely unambiguous we can consider another fault model $F_4(x_0^2, T_2, T_{10})$. Note here the duration of the fault model is extended to 9 clock-cycles and the register faulted is x_0^2.

Again, we compute the following partial derivatives to determine the conditions for fault propagation:

$$\begin{aligned}
\frac{\partial f_0}{\partial x_0^2} &= x_2^1 \\
\frac{\partial f_1}{\partial x_0^2} &= x_3^1 \\
\frac{\partial f_2}{\partial x_0^2} &= x_1 \\
\frac{\partial f_3}{\partial x_0^2} &= 1 + x_3^2 + x_3^3
\end{aligned}$$

Thus, the fault propagates to the output iff $x_1 = 1$ or $x_2^1 = 1$ or $x_3^1 = 1$ or $x_3^2 + x_3^3 = 0$. With this incorporation the final template is shown underneath.

x_3	x_2	x_1	x_0	p_0	p_1	p_2	p_3
0	0	0	0	0	$3n/4$	$3n/4$	$7n/8$
0	0	0	1	n	$3n/4$	$3n/4$	$7n/8$
0	0	1	0	0	n	$3n/4$	$7n/8$
0	0	1	1	n	n	$3n/4$	$7n/8$
0	1	0	0	0	$3n/4$	n	$7n/8$
0	1	0	1	n	$3n/4$	n	$7n/8$
0	1	1	0	0	n	n	$7n/8$
0	1	1	1	n	n	n	$7n/8$
1	0	0	0	0	n	$3n/4$	n
1	0	0	1	n	n	$3n/4$	n
1	0	1	0	0	n	$3n/4$	n
1	0	1	1	n	n	$3n/4$	n
1	1	0	0	0	n	n	n
1	1	0	1	n	n	n	n
1	1	1	0	0	n	n	n
1	1	1	1	n	n	n	n

Table 1: Fault Template for TI PRESENT S-box

x_3	x_2	x_1	x_0	p_0	p_1	p_2	p_3	p_4
0	0	0	0	0	$3n/4$	$3n/4$	$7n/8$	n
0	0	0	1	n	$3n/4$	$3n/4$	$7n/8$	n
0	0	1	0	0	n	$3n/4$	$7n/8$	n
0	0	1	1	n	n	$3n/4$	$7n/8$	n
0	1	0	0	0	$3n/4$	n	$7n/8$	n
0	1	0	1	n	$3n/4$	n	$7n/8$	n
0	1	1	0	0	n	n	$7n/8$	n
0	1	1	1	n	n	n	$7n/8$	n
1	0	0	0	0	n	$3n/4$	n	$3n/4$
1	0	0	1	n	n	$3n/4$	n	$3n/4$
1	0	1	0	0	n	$3n/4$	n	n
1	0	1	1	n	n	$3n/4$	n	n
1	1	0	0	0	n	n	n	$3n/4$
1	1	0	1	n	n	n	n	$3n/4$
1	1	1	0	0	n	n	n	n
1	1	1	1	n	n	n	n	n

Table 2: Final Fault Template for TI PRESENT S-box

It may be noted that the templates for all the inputs are distinct, ie. for two distinct inputs $x \neq x'$, $T[x] \neq T[x']$. This shows that FTA2.0 provides a useful and powerful template to determine the unshared input x with a high accuracy.

4.4 Template Matching

The template shown in Table 2 is used as a reference to ascertain the input states by fault injection. *As mentioned each of the 16 entries in the table are distinct.* In the template matching phase, we leverage this observation to extract the unshared input of the S-box inputs from the fault templates.

During the template building phase we tune the fault injection to induce the faults according to the fault models F_0 to F_3. The number of faulty outputs are observed after performing N fault injections for each type of fault. The fault signature is stored by a vector $W = (w_0, w_1, w_2, w_3, w_4)$, where each entity w_i, $0 \leq i \leq 4$ stores the number of faulty outputs. This vector is then correlated with all the distinct rows, $P = (p_0, p_1, p_2, p_3, p_4)$. The internal state returned is $x = Corr_{max}(P, W)$. The leakage of the unshared internal state of two successive rounds of a block cipher can be used to obtain the round secret key. Note that the attack, as proposed in the original paper [2] does not require access to the plaintext or ciphhertext and hence can be used to target even a middle round encryption operation.

5 Conclusions

The above discussion shows that while it is a common practice to consider share bits to be stored in registers, the ability to create transient faults in such registers for multiple-cycles, following a commonly known Double Bit Upset (DBU) fault model can lead to Fault Template Attacks. These FTAs can be fine tuned during the template building phase, and eventually launched during the attack phase to leak the actual input unshared bits of the S-Box. This leaks intermediate rounds of the cipher which can compromise the security of the block cipher.

References

[1] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential fault analysis of the advanced encryption standard using a single fault. In *IFIP international workshop on information security theory and practices*, pages 224–233. Springer, 2011.

[2] Debapriya Basu Roy Sikhar Patranabis Sayandeep Saha, Arnab Bag and Debdeep Mukhopadhyay. Fault template attacks on block ciphers exploiting fault propagation. In *Proceedings of Eurocrypt 2020.*