When Clocks Fail
– On Critical Paths And Clock Faults –

Michel Agoyan[1], Jean-Max Dutertre[2],
David Naccache[1,3], Bruno Robisson[1], and Assia Tria[1]

[1] CEA-LETI
{michel.agoyan, bruno.robisson, assia.tria}@cea.fr

[2] École nationale supérieure des Mines de Saint-Étienne
dutertre@emse.fr

Centre microélectronique de Provence G. Charpak
Département SAS
80 Avenue de Mimet, F-13120 Gardanne, France

[3] École normale supérieure, Département d'informatique, Équipe de cryptographie,
45 rue d'Ulm, F-75230 Paris CEDEX 05, France
david.naccache@ens.fr

Abstract. Whilst clock fault attacks are known to be a serious security threat, an in-depth explanation of such faults still seems to be put in order.

This work provides a theoretical analysis, backed by practical experiments, explaining when and how clock faults occur. Understanding and modeling the chain of events following a transient clock alteration allows to accurately predict faulty circuit behavior. A prediction fully confirmed by injecting variable-duration faults at predetermined clock cycles.

We illustrate the process by successfully attacking an FPGA AES implementation using a DLL-based FPGA platform (one-bit fault attack).

1 Introduction

Fault attacks consist in modifying an electronic circuit's behavior to achieve malicious goals [2, 3]. Fault attacks exist in numerous variants ranging from a simple alteration of a round counter during symmetric encryption [4] to mathematical Differential Fault Attacks (DFA) where secret information is obtained by comparing (differentiating) correct and faulty encryption results [6, 12].

Faults can be caused by a variety of intrusive and non-intrusive means [1] such as lasers [16], electromagnetic perturbations [10, 13], voltage variations [9] or clock glitches [7].

In this work we present a new clock alteration technique for *scenarii* in which the attacker is given access to the target's clock. We start by explaining and modeling the chain of events causing the faulty behavior. This theoretical analysis

perfectly reflects experimental observations and allowed the injection of precise single-bit faults into a chip running the AES algorithm (an actual implementation of the attack described in [8]).

After introducing the model, we will overview the fault injector's design, the target chip's structure and the way in which the injector was used to extract AES keys.

2 Why Clock Faults Occur?

We inject faults by violating *synchrony*, a basic assumption under which traditional digital ICs operate. In essence, most[4] ICs execute calculations by processing data by combinatorial logic blocks separated by D flip-flop register banks sharing the same clock (figure 1).

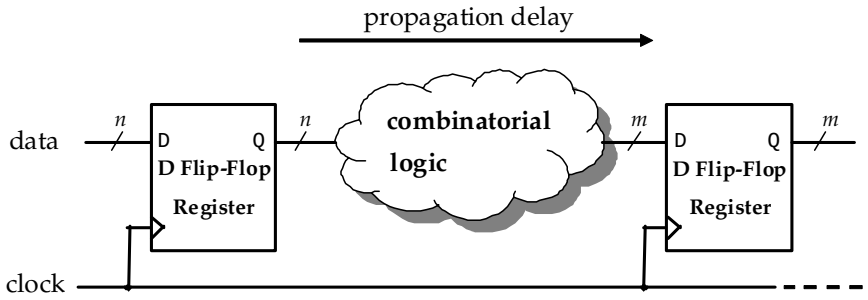

Fig. 1. Synchronous Representation of Digital ICs

Data is usually latched by registers at raising clock edges. Between two such edges, the computed data travels between registers and gets modified by the intermediate combinatorial logic blocks. The time needed to propagate data through combinatorial logic is called *propagation delay*. The propagation delay and a second delay element, inherent to the use of D flip-flop, called *set-up time*, define the circuit's maximal operating frequency (nominal circuit period). Indeed, to ensure proper circuit operation, the clock period must be strictly greater than the maximal propagation delay in concerned circuit (this maximal propagation delay is called *critical path*) plus the registers' set-up time. In other words:

$$T_{\text{clock}} > t_{\text{critical}} + t_{\text{set-up}} \tag{1}$$

[4] ICs that do not assume synchrony exist but we do not consider these in this work.

As a matter of fact any data bit entering a register is the result of a combinatorial calculation involving several previous register output bits. The transformation of the previous registers' output into the next register's input bit takes a determined delay. This delay depends on the the logic performed as well as on the data transiting through the logic. In addition, propagation time varies with circuit temperature and power supply voltage.

2.1 Overclocking

Overclocking consists in decreasing the clock period (or, put differently, increasing clock frequency). If setup delays are not respected, the D flip-flop's input is not given sufficient time to reach the latch. Causing faulty data to be latched instead. This led several authors to use overclocking as fault injection means [9, 15].

A decreased clock period can potentially affect logical paths whose propagation times exceed the decreased clock period minus the set-up time. From the attacker's perspective, the ability to control precisely the clock period is crucial for inducing faults with precision. Note that temperature and power supply changes may also be used to exert such control.

Fault attacks consist in injecting faults at precise moments. To avoid injecting faults continuously, overclocking must be brief and transient. The fault injection technique described in the next section allows doing so.

2.2 Injecting Clock Delay Faults

An attacker needs to control two parameters: the precise moment at which the fault occurs and the clock anomaly's duration. The latter must be controlled with high resolution, typically a few of tens of picoseconds.

Figure 2 illustrates a correct clock signal, CLK, and a modified clock meant to cause faults, FAULTY_CLK.

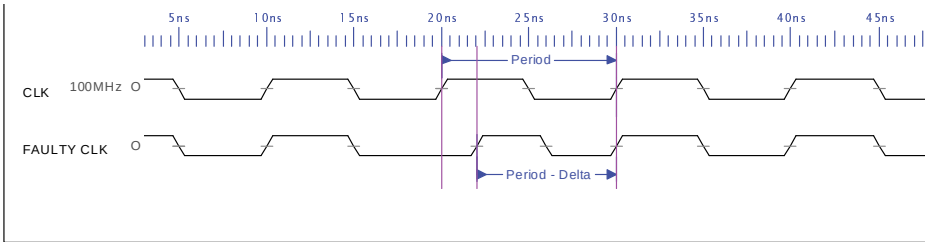

Fig. 2. Normal (CLK) *vs.* Faulty (FAULTY_CLK) Clock Signals

The two waveforms differ only between the delimiters positioned at 20ns and 30ns. During that interval, the FAULTY_CLK's period is reduced by Δ ns. The Δ

time decrement causes a set-up time violation fault. Note that the extension by Δ of the preceding clock cycle's low state has no adverse effect.

Generating FAULTY_CLK with sufficient accuracy and precision is a challenging task. To do so, we use the embedded Delay Locked Loop (DLL) of a recent FPGA family (Xilinx Virtex 5). Two clocks (CLK_DELAYED i) with programmable skews are generated from CLK. The skews of the CLK_DELAYED i signals, denoted δ_i, are programmable. FAULTY_CLK is obtained by switching between the CLK_DELAYED i using a TRIGGER signal. Figure 3 depicts this process in further details.

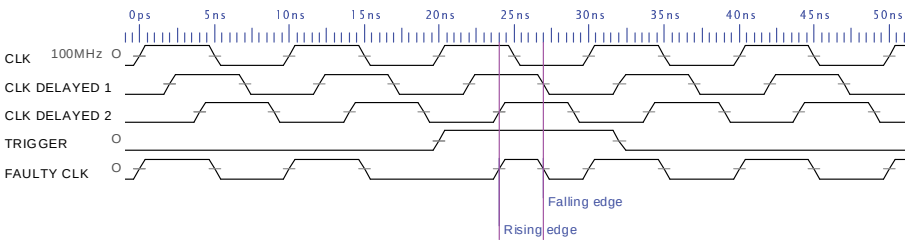

Fig. 3. Faulty (FAULTY_CLK) Clock Signal Generation

If CLK_DELAYED 2 is delayed by δ_2 time units, CLK_DELAYED 1 must be delayed by $\delta_1 = \frac{\delta_2}{2}$ to preserve a 50% duty cycle at FAULTY_CLK's transient fault interval.

The device assembles FAULTY_CLK by combining CLK_DELAYED 2's raising edge and CLK_DELAYED 1's falling edge. This is controlled by the signal TRIGGER that positions the perturbation in time. The accurancy at which FAULTY_CLK's shape can be controlled (in our setting 35ps) depends on δ_t, the smallest elementary delay that the DLL is able to provide. We will refer to FAULTY_CLK's altered signal chunk as the *faulting period* (in Figure 3, the interval between 24ns and 30ns).

An oscilloscope screen-shot of FAULTY_CLK is shown on Figure 4 (uppermost signal). Here, CLK's period (10ns) was reduced to $[t_1, t_2] = 49\delta_t \simeq 8.2$ns in FAULTY_CLK. The lowermost signal's high level indicates the AES' start. The implementation completes an encryption round in one cycle. Hence, the diagram shows a fault injection at the ninth round (*cf.* section 3.2).

As we write these lines, a Xilinx Virtex 5 development board costs less than $1000.

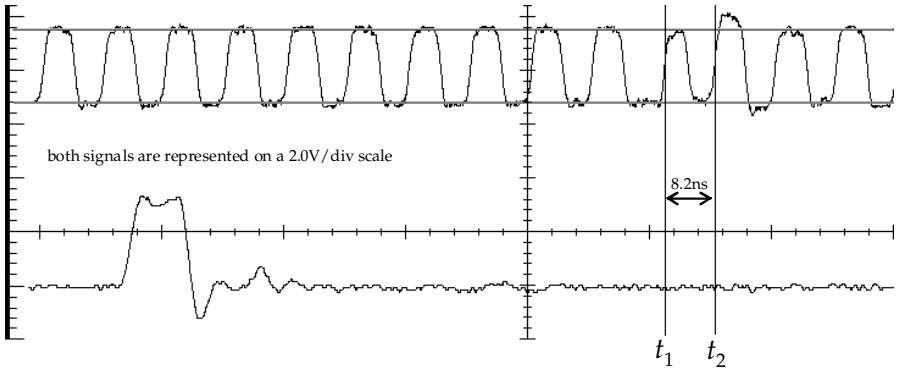

Fig. 4. FAULTY_CLK (uppermost signal) and AES_START (lowermost signal)

3 Clock Fault DFA on AES

We tested the attack setup on a concrete AES implementation. The following sections describe the target chip, the attack's theoretical principle ([8]) and report practical experiment results.

3.1 The Test Chip

The test chip (Xilinx Spartan 3AN FPGA) implements a hardware 128-bit AES [11, 5] written in VHDL. The design consists of three main blocks: a communication and control module (CCM), a key expansion module (KEM), and an encryption module (ENM).

The CCM manages the serial link through which plaintext and key material are input. The start signal triggering the encryption and the resulting ciphertext also transit through the CCM. In addition, the CCM controls the KEM and the ENM's operations during encryption.

The implementation uses a 128-bit data path and runs the KEM and the ENM in parallel. Consequently, an encryption round is completed in one clock cycle and the entire AES computation takes 11 clock cycles.

The KEM generates the round keys "on the fly". At each clock cycle, a new round key is transferred from the KEM to the ENM. We will not overview the KEM in detail as it is of little relevance to our attack. We nonetheless underline that the KEM's critical delay path is much smaller than the ENM's one – this is essential for the attack to work.

The ENM architecture is depicted on Figure 5. The ENM breaks-down into five submodules: `AddRoundKey`, `SubBytes`, `ShiftRows`, `MixColumns`, and `Mux`. As their names suggest, the first four correspond to the standard AES transformations. They are assembled with the multiplexer module, `Mux`, to form a closed loop, implementing a complete AES round.

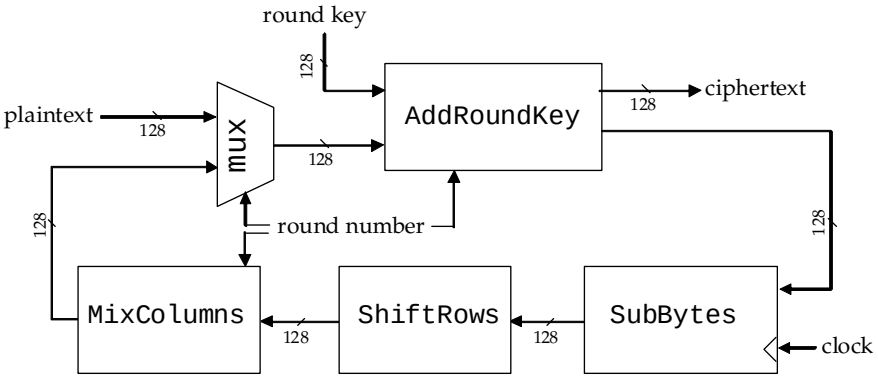

Fig. 5. AES Structure

The `Mux` module opens the loop for plaintext acquisition during the initial round and closes it afterwards. The `AddRoundKey` module has a dedicated bus (ciphertext) through which ciphertext is output after the final round. The `MixColumns` module is bypassed during the final round. `SubBytes` is the only clocked module (all the others being purely combinatorial blocks). This allows, as mentioned before, to complete an encryption round in one clock cycle. This loop architecture features a long data propagation path. Consequently, the design's critical delay path is located in the ENM. The nominal clock frequency of this AES implementation is 100 MHz.

3.2 Giraud's One-Bit Attack

This section recalls Giraud's DFA [8] (hereafter "Giraud's one-bit attack"). The attack is based on the restrictive assumption that the opponent can inject one-bit faults. We use the following notations:

M^i	the algorithm's state at the end of round i
K^i	i-th round key number
C	a correct ciphertext
D	a faulty ciphertext

The attacker injects a single bit fault into one byte of state M^9 just before the final round's `SubBytes` operation. This allows to retrieve the last round key K^{10}.

Consider the i-th state byte just before the final round M_i^9. The corresponding byte index in the ciphertext is $\texttt{ShiftRows}(i)$. As per the AES' specifications, for all $i \in \{0, \ldots, 15\}$:

$$C_{\texttt{ShiftRows}(i)} = \texttt{SubBytes}(M_i^9) \oplus K^{10}_{\texttt{ShiftRows}(i)} \tag{2}$$

If a one-bit fault e,[5] is injected into the j-th byte of M^9, we obtain at index j:

$$D_{\texttt{ShiftRows}(j)} = \texttt{SubBytes}(M^9_j \oplus e) \oplus K^{10}_{\texttt{ShiftRows}(j)} \tag{3}$$

and for index $i \in \{0, \ldots, 15\} \backslash \{j\}$:

$$D_{\texttt{ShiftRows}(i)} = \texttt{SubBytes}(M^9_i) \oplus K^{10}_{\texttt{ShiftRows}(i)} \tag{4}$$

Hence, a comparison (differentiation) between the correct and the faulty ciphertexts leaks information both on the fault's position and on the AES key. For $i \in \{0, \ldots, 15\} \backslash \{j\}$, equations 2 and 4 yield:

$$C_{\texttt{ShiftRows}(i)} = D_{\texttt{ShiftRows}(i)} \tag{5}$$

This allows to identify the faulty byte's index j because the only index for which $C \oplus D$ is nonzero is $\texttt{ShiftRows}(j)$. Moreover, at index j, equations 2 and 3 yield:

$$C_{\texttt{ShiftRows}(j)} \oplus D_{\texttt{ShiftRows}(j)} = \texttt{SubBytes}(M^9_j) \oplus \texttt{SubBytes}(M^9_j \oplus e) \tag{6}$$

Equation 6 is then solved for the eight possible values of e (the only hypothesis made on e is that $e = 2^i$ for some i). This provides a set of candidates for M^9_j.

At this point, new one-bit fault injections targeting the same byte are required to reduce the solution set to one item, namely M^9_j. The probability to find M^9_j in three attempts (*i.e.* three different faults) is $\simeq 99\%$.

$K^{10}_{\texttt{ShiftRows}(j)}$ is then calculated from M^9_j, using equation 2:

$$K^{10}_{\texttt{ShiftRows}(j)} = C_{\texttt{ShiftRows}(i)} \oplus \texttt{SubBytes}(M^9_j) \tag{7}$$

Equation 7 shows that the attack works independently on each round key byte. Indeed, no `MixColumns` transformation follows the fault injection and `MixColumns` is the only transformation capable of propagating faults among bytes. Consequently, the whole round key K^{10} can be progressively retrieved by the attacker. Finally, knowing K^{10}, the secret key K is found by reversing the key expansion algorithm.

3.3 The Attack Process

The experimental set-up (Figure 6) consists in a test chip board (TCB), a clock fault generator (CFG) and a computer.

[5] That is: $e = 2^i$ for $i = 0, \ldots, 7$.

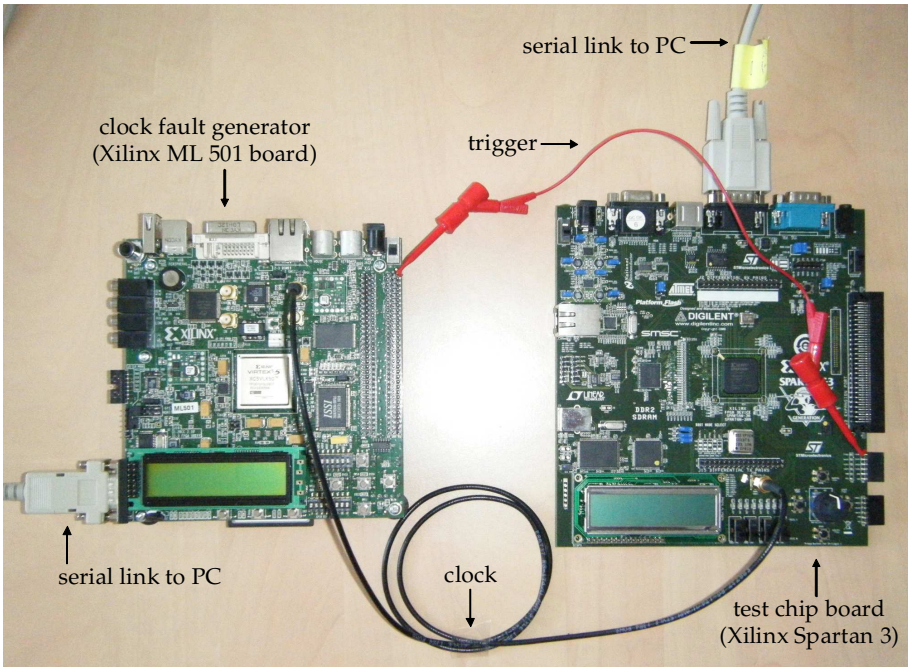

Fig. 6. Experimental Set-up

The TCB embeds the AES implementation described in the previous section. Its clock is provided by the CFG. The secret key, the plaintext and the encryption start signal are transmitted to the TCB from the computer *via* a serial link. As encryption ends, the ciphertext is offloaded *via* the serial link. The TCB provides a trigger signal to the CFG indicating the exact beginning of the encryption process. This is done to ease synchrony between fault injection and AES rounds. Note that the trigger could be replaced by the inspection of power consumption (SPA) to precisely locate the AES rounds in time.

The CFG generates a 100 MHz clock signal continuously fed into the TCB. When the trigger signal indicates that encryption was started, a countdown is started. As the countdown ends a faulting period is produced during round nine, as required by Giraud's one-bit attack. The serial link between the computer and the CFG is used to define the exact value of the faulting period decrement Δ.

The computer runs a test campaign as described in Algorithm 1.

The generation of a faulting period is automatic and hence not explicitly mentioned in the pseudo-code. Indeed, the trigger signal sent from the TCB to the CFG indicates the encryption's launch and causes a faulting period generation during the ninth round as shown in Figure 4. As the faulting period gradually decreases, more set-up time violation faults appear in the calculation process. The resulting faulty ciphertexts are ordered by decreasing faulting periods. Then,

Algorithm 1 Test Campaign Pseudo-Code

```
send the key K and the plaintext M to the test chip.
Δ ← 0.
while (clock_period > Δ) do
  encrypt and retrieve the ciphertext
  Δ ← Δ + δ_t
end while
```

faulty ciphertexts are successively compared with the correct ciphertext. This allows identifying the faulty bytes (Eq. 6 and Eq. 7). In addition, for each ciphertext byte, a list of induced faults is built in order of appearance. Assuming that the first injected fault stems from a one-bit fault induced just before the last `SubBytes`, we build the corresponding set of guessed bytes for K^{10} from equations 6 and 7. For the test campaign described in Algorithm 1, we obtain a set of guesses for every byte of K^{10}. To reduce progressively these sets to singletons we inject different one-bit faults repeating the test campaigns with the same key but with different plaintexts. Indeed, each data bit arriving to the `SubBytes`'s registers possesses its own logic path and propagation time (section 2). This propagation time highly depends on the data handled during encryption. Consequently, plaintext changes modify all propagation times. As propagation times vary, the injected one-bit faults differ with a 7/8 probability at the byte level.

As a result, one needs at least three (and sometimes four) test campaigns (same key and different plaintexts) to retrieve the entire round key. Finally, K is obtained by reversing the key expansion process.

3.4 Experimental Results

A first experiment targeting the final AES round was conducted to test the CFG. We implemented successfully Giraud's one-bit attack as well as its extension to two bits.

Injecting multiple-bit faults To test the hypothesis that a decrease in the faulting period causes more internal faults, we targeted the AES' tenth round while progressively increasing Δ.

As expected, the comparison of correct and faulty ciphertexts reveals that the device progressively transits from normal operation to multi-bit faults. This is done by exhibiting none, one-bit, two-bit and multiple-bit faults as described in Figure 7. Note that the above is not an attack but an experimental fault characterization experiment where the AES plays the role of a big propagation delay cause.

Figure 7 reports fault characterization experiments conducted with a constant key and two different plaintexts.

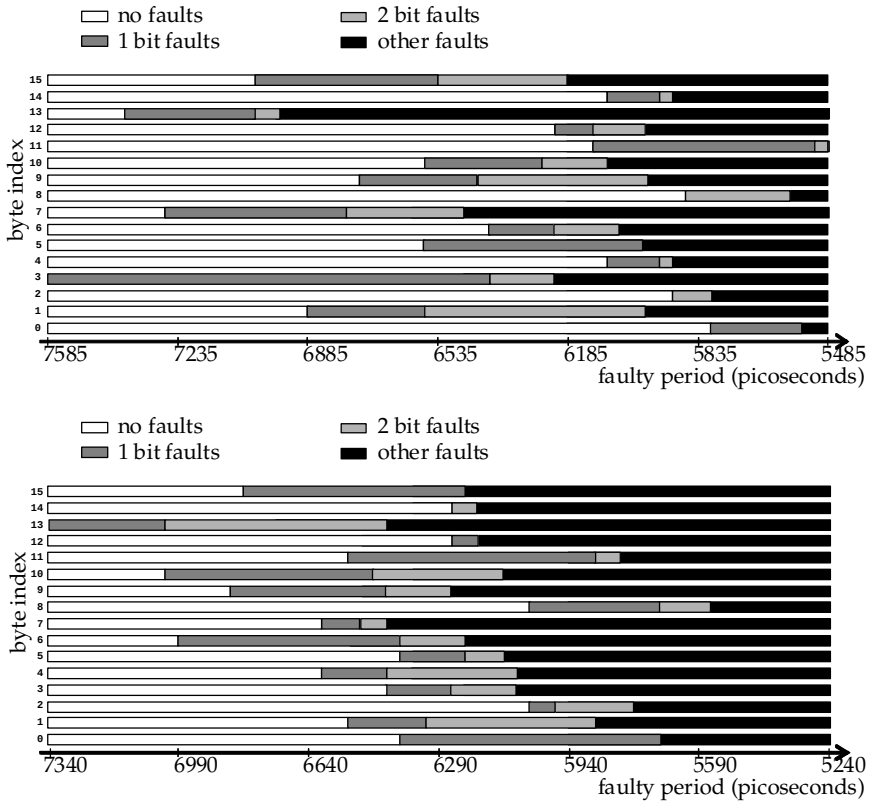

Fig. 7. Two fault injection experiments: same key, different plaintexts

Figure 7 shows the faults' timing and nature as a function of the faulting period's duration (the horizontal axis). Each horizontal bar, associated with a byte number, uses a color code to reflect the nature of faults (no fault, one-bit fault, two-bits fault, more than two bits fault) and their point of appearance in time. The first one-bit fault occurs in byte 3 for a faulting period of 7585 ps ($= 10000 - 69 \times 35$ ps), and the last fault appears on byte 0 for a 5800 ps faulting period. With the exception of bytes 2 and 8, the first fault is always a one-bit fault. This is compatible with the theoretical fault genesis model introduced at the beginning of this paper. Figure 8 shows how the number of byte candidates is progressively reduced. Each of the four sets in the example is associated with a different one-bit fault e_i (for $i \in \{1, 2, 3, 4\}$). Here, the correct round key byte, `0x25`, is found at the intersection of sets associated with e_1, e_3 and e_4. The fact that the set associated with e_2 stands apart indicates that the one-bit fault assumption about e_2 was wrong. Leaving aside e_2, the set of guesses is reduced by successive sets intersections until a singleton containing the round key byte is

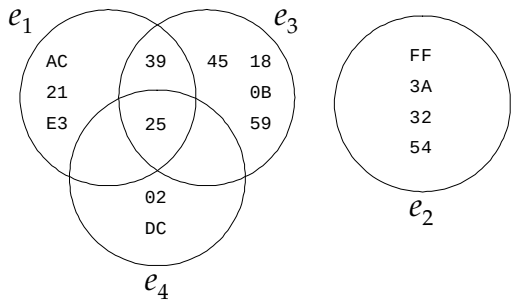

Fig. 8. Reducing the number of bytes candidates

reached. However, taking into account set e_2 requires the more complex intersection process described in Algorithm 2.

Algorithm 2 Determining the correct key byte

```
i ← 2
S ← e_1
while (|S| ≠ 1) do
  if S ∩ e_i = ∅ then
    S = S ∪ e_i
  else
    S = S ∩ e_i
  end if
  i ← i + 1
end while
return the single element of S as the correct key byte.
```

The probability of injecting successively a one-bit and a two-bit fault when reducing the faulty period can be estimated from Figure 7, where bytes 8, 5, 2 and 0 are counter-examples. Many examples seem to indicate that this probability is greater than 70% (experimentally).

The probability to cause successively one-bit, two-bit, and three-bit faults seems to be 50% (experimentally). These statistics were obtained thanks to the very small value of the faulty-period granularity, namely 35ps. This resolution seems to allow the progressive accumulation of faulty propagation paths as the faulting period is decreased.

Implementation of Giraud's one-bit attack We wrote a script that plays the elementary test campaign described in section 3.3. The test was run several

tens of times, for at least five different plaintexts per run. We always found the correct round key. The probability of injecting one-bit faults was found to be greater or equal to 90%.

Extension to two-bit attack The ability to inject and identify two-bit faults has prompted us to extend the attack. Note that two-bit attacks can defeat one-bit parity check countermeasures. To that end, we need to solve equation 6 with the assumption that e is a two-bit fault. The only adverse effect is an increase in the cardinality of potential solution sets. The experiment was successful: 13 to 14 round key bytes were found on average with the automated test campaign. This allowed to exhaustive-search the whole key.

Repeating the campaign with many different plaintexts, we were always able to inject two-bit fault at every byte location, even if we never succeeded in injecting two-bit faults simultaneously on all bytes.

4 Conclusion

This paper describes a new fault injection technique based on clock reshaping during a precise cycle to cause setup time violation faults. The new technique would be a good candidate to benchmark safe-error [17] or differential behaviorial [14] analysis

This technique is inexpensive, efficient and non-intrusive. As such, it underlines the importance of further research in countermeasure design.

References

1. Hagai Bar-El, Hamid Choukri, David Naccache Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. In *Special Issue on Cryptography and Security 94(2)*, pages 370–382, 2006.
2. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, Lecture Notes in Computer Science, pages 513–525. Springer, 1997.
3. Dan Boneth, Richard A. DeMillo, and Lipton Richard J. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology EUROCRYPT'97*, Lecture Notes in Computer Science, pages 37–51. Springer, 1997.
4. Hamid Choukri and Michael Tunstall. Round reduction using faults. *Proc. Second Int'l Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC '05)*, 2005.
5. J. Daemen and V. Rijmen. Aes proposal: Rijndael, 1998.
6. Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on aes. In *ACNS: applied cryptography and network security*, volume 2846 of *Lecture Notes in Computer Science*, pages 293–306. Springer, 2003.
7. Toshinori Fukunaga and Junko Takahashi. Practical fault attack on a cryptographic lsi with iso/iec 18033-3 block ciphers. In *Proc. of the 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC09*, pages 84–92, 2009.

8. Christophe Giraud. DFA on AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Advanced Encryption Standard − AES*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2005.
9. Sylvain Guilley, Laurent Sauvage, Jean-Luc Danger, Nidhal Selmane, and Renaud Pacalet. Silicon-level solutions to counteract passive and active attacks. In *FDTC '08: Proceedings of the 2008 5-th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 3–17, 2008.
10. Michael Hutter Jörn-Marc Schmidt. Optical and em fault-attacks on crt-based rsa: Concrete results. In *Proceedings of the 15th Austrian Workhop on Microelectronics*, 2007.
11. NIST. Announcing the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication, n. 197, November 26, 2001.
12. Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In *Proc. Cryptographic Hardware and Embedded Systems (CHES '03),*, Lecture Notes in Computer Science, pages 77–88, 2003.
13. JJ Quisquater and D Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of ESmart 2002*, page pp 185194. Eurosmart, 2002.
14. Bruno Robisson and Pascal Manet. Differential behavioral analysis. In *Cryptographic Hardware and Embedded Systems*, volume 4727 of *Lecture Notes in Computer Science*, pages 413–426. Springer, 2007.
15. Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical setup time violation attacks on AES. In *EDCC-7 '08: Proceedings of the 2008 Seventh European Dependable Computing Conference*, pages 91–96, 2008.
16. Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems − CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.
17. Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49:967–970, 2000.