Collision Timing Attack when Breaking 42 AES ASIC Cores

Amir Moradi, Oliver Mischke, and Christof Paar

Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
{moradi, mischke, cpaar}@crypto.rub.de

Abstract. A collision timing attack which exploits the data-dependent timing characteristics of combinational circuits is demonstrated. The attack is based on the correlation collision attack presented at CHES 2010, and the timing attributes of combinational circuits when implementing complex functions, e.g., S-boxes, in hardware are exploited by the help of the scheme used in another CHES 2010 paper namely fault sensitivity analysis. Similarly to other side-channel collision attacks, our approach avoids the need for a hypothetical model to recover the secret materials. The results when attacking all 14 AES ASIC cores of the SASEBO LSI chips in three different process technologies, $130nm$, $90nm$, and $65nm$, are presented. Successfully breaking the DPA-protected and the fault attack protected cores indicates the strength of the attack.

1 Introduction

While modern cryptographic algorithms can mostly be assumed secure from the mathematical point-of-view, their implementation can in general easily be broken by means of so-called side-channel attacks if no special countermeasures are applied. Sensitive information about the internal secret of the cryptographic device, e.g., the used encryption key in a symmetric cipher, leak through side channels. The yet known side channels include execution time [11], power consumption [12], electromagnetic radiation [18], and even faulty outputs in case of intentionally injected faults used by e.g., differential fault analysis [4].

In CHES 2010 a new side-channel attack called Fault Sensitivity Analysis (FSA) [13] was proposed, which uses the fact that the critical path of an AES S-box implementation is data dependent because of the nature of the underlying gates. Using clock glitches and a simple hypothetical (timing) model for the S-box – which can be obtained by simulation knowing the design details or by a profiling phase – they showed a complete break of an AES ASIC implementation using only 50 faulty ciphertexts.

Another contribution to CHES 2010 was a collision attack enhanced by means of correlation [14]. Compared to classical power analysis attacks, its main feature is that it does not rely on the knowledge of an underlying (hypothetical) power model. Instead, it directly correlates power traces to each other and – by finding colliding S-box computations – is able to recover the relation between key parts.

Using such an attack the authors have shown a complete break of a masked FPGA implementation of AES.

The combination and improvement of these two ideas is the main contribution of this article. We present an attack that exploits the timing characteristics of AES S-boxes, but in order to recover the secret does not need to know specifically how these characteristics are and how they relate to the inputs. Despite the similar name, the attack presented here is different to [7], where a collision timing attack on cache misses of an embedded system is presented. The aim of our proposed attack is to exploit the timing characteristics of combinational functions, e.g., S-boxes implemented in ASICs, thereby recovering the relation between key parts and restricting the key search space in a way that the secret key can be revealed knowing a single plaintext-ciphertext pair.

Similarly to [13], we have chosen the SASEBO-R boards as the evaluation platform. The board can hold different ASICs, and we have analyzed the SASEBO LSI2, both in 130nm and 90nm technology, as well as the SASEBO LSI3 in 65nm technology. Each of them contains the same 14 different implementations of AES, thereby the only difference is the process technology, which has a big influence on the timing characteristics. The implementations themselves differ in the style of the S-box realization and in side-channel countermeasures. One of the AES implementations is also specifically designed to withstand attacks using fault injection, which in theory should make the implementation resistant against attacks like FSA.

Using the attack presented here we are able to break all 14 AES implementations of each LSI regardless of the process technology. It is noteworthy that the effort required to mount the attack on different cores varies because of their different architecture and mostly their different S-box design. Nevertheless, the proposed attack can not only extract the desired secret from unprotected implementations, but it is especially able to overcome all DPA- and Fault-based countermeasures which are applied in these ASIC designs. In other words, randomizing the computation of combinational circuits by means of different masking schemes does not prevent the relation between its timing characteristics and the processed data. Preventing faulty ciphertexts by means of one of the most efficient fault detection schemes, furthermore, cannot prevent an attacker who injects the faults to measures the timing characteristics of the circuit.

In the later parts of this article the prerequisites, including a review of fault sensitivity analysis and correlation collision attack, are given in Section 2. Our proposed attack, namely collision timing attack, is expressed in Section 3, and the practical evaluation results on all 130nm, 90nm, and 65nm SASEBO LSI2 and LSI3 cores are presented in Section 4. Finally, Section 5 concludes our research.

2 Preliminaries

This section summarizes the underlying attacks, namely the Fault Sensitivity Analysis [13] and Correlation Collision Attack [14], which are the basis to the proposed Collision Timing Attack. We also explain how the timing data, which is required for the attack, can be captured and finally give some definitions which are used in the following sections.

2.1 Fault Sensitivity Analysis

The Fault Sensitivity Analysis (FSA) was introduced in [13]. Unlike Differential Fault Attacks (DFA) [4], no faulty ciphertexts are required. Instead, the attack works by increasing the fault intensity until a distinguishable characteristic can be observed, e.g., the first appearance of a faulty output. It was demonstrated that the attack is able to completely break the AES_PPRM1 implementation of an SASEBO-R LSI [1] using only 50 encryption operations and can reveal three key bytes of the AES_WDDL implementation of the same ASIC using 1200 encryptions.

The presented method of increasing the fault sensitivity in [13] is the shortening of clock glitches, i.e., two normal clocks get replaced by a short one and a longer one, whereby the length of the short one can be gradually decreased until a faulty output occurs or the fault becomes stable. Since the critical path of some gates, e.g., AND and OR gates, is data dependent, knowing the underlying model for this data dependency helps revealing the secret. For example, by simulation it could be ascertained that the timing delay of the PPRM S-box correlates to the Hamming weight (HW) of the S-box input.

Since no faulty ciphertexts are required, the attack might also be applicable to implementations which apply DFA countermeasures. While WDDL-AES should be in theory immune against set-up time violation attacks, by creating templates with a known key it was shown that at least some bits correlate to the timing delay which lead to the aforementioned recovery of three key bytes.

Besides the fact that no faulty ciphertexts are required, another difference to DFA attacks is that the fault does not need to be restricted to a small subspace. In contrary, by for example attacking the last round of the PPRM1-AES implementation, each faulty output byte can be independently observed and therefore the same complete faulty output can be used to attack all key bytes simultaneously. On the other hand, as stated in [13], while countermeasures like masking are only of limited use against DFA attacks , it may have a great impact on FSA attacks since the critical path is affected by the random mask bits.

2.2 Correlation Collision Attack

The correlation collision attack was firstly introduced as Correlation-Enhanced Power Analysis Collision Attack in [14]. While remaining a certain noise sensitivity, its major advantage compared to classical power analysis attacks is that it neither relies on a hypothetical power model nor a distinguisher nor requires a profiling phase. By enhancing linear collision attacks [6] by the methods of correlation-based DPAs, it is able to overcome SCA countermeasures as long as a minimal first order leakage remains.

Linear collisions in the power consumption occur if two instances of combinational circuits, in case of AES the S-boxes, or one instance at two different points in time process(es) the same value, i.e., for AES the 8-bit output and thereby the input of the S-box must be the same. In this case, it is possible to deduce the key relation of the attacked bytes since the rearranging of $\mathrm{Sbox}(i_1 \oplus k_1) = \mathrm{Sbox}(i_2 \oplus k_2)$ leads to $\Delta = i_1 \oplus i_2 = k_1 \oplus k_2$, where both i_1 and i_2 are known.

The correlation collision attack on AES works similarly, but starts by creating sets of mean traces for each possible input byte if performing the attack on the first round. To do this for two input bytes, namely i_1 and i_2, all traces are sorted based on the corresponding input byte value, and traces with the same value are averaged, thereby creating 256 different mean traces $M_1(\alpha)$ and $M_2(\alpha)$ for each of the two input bytes. Computing the variances for each set of mean traces will not only reveal the point in time where the corresponding bytes are processed by the S-box, which is necessary to align the traces for the attack, a peak in the variance trace is also the necessary requirement for the attack to be successful since this indicates an however slightly first order leakage.

If the previous assumption holds true, i.e., the power consumption of two S-box computations are highly similar, comparing pairs of mean sets also shows a high similarity between certain mean traces. Therefore, when attacking the input bytes i_1 and i_2 and $\Delta = k_1 \oplus k_2$, then $M_1(\alpha) \approx M_2(\alpha \oplus \Delta)$. The correct Δ can be found by computing the correlation between the two sets of mean traces for each of the 256 candidates of Δ, which will yield a very high correlation coefficient since no hypothetical model is applied but instead a direct collision between the averaged real power consumptions will occur. Knowing enough Δs between different pairs of M_i shrinks the key space to 2^8, after which the correct key can be revealed by knowing a single plaintext-ciphertext pair or by extending the attack on the second round as explained in [14].

2.3 How to Measure the Timing

The focus of this work is to analyze the timing characteristics of combinational circuits like S-boxes. As explained in [13], when the input of a combinational circuit changes, its output stops toggling after a certain time (so-called Δt). The maximum value of Δt for different inputs is known as the longest critical path of the circuit, and defines the maximum frequency of the clock signal which triggers the flip-flops providing the input and storing the output of the considered combinational circuit. Timing characteristics of a circuit are therefore defined as a set of Δt's ($\{\Delta t^1, \Delta t^2, \ldots, \Delta t^n\}$), where Δt^i is the minimum Δt for the given input i.

Let us suppose that the target combinational circuit is a part of a bigger circuit, e.g., a co-processor, which provides some I/O signals for communication. If the output signals of the target combinational circuit are accessible from the outside — which is quite unlikely —, one can easily probe the output signals for the given input i and measure the time when the output signals get stable. Otherwise, if the output of the target combinational circuit is stored into registers which are triggered by a clock signal that can be controlled from the outside, as shown in [13], one can steadily shorten the time interval between the input transition and the output storage (known as *setup time*) till an incorrect value is stored into the registers while input i is given to the combinational circuit. The minimum time interval when the considered register stores the correct value can be concluded to Δt^i.

Note that this procedure is similar to *clock glitches*, which are mostly used in *differential fault attacks* (DFA) to intentionally inject faults or to skip the execution of an instruction and analyze the faulty outputs based on the target algorithm [4, 5, 10, 16]. However, measuring Δt in this case does not deal with the faulty outputs; once a faulty output is detected, Δt^i can be concluded.

It should be noted that, because of the environmental noise, it might be required to repeat the same procedure and shorten the clock glitch period until the probability of detecting faulty output gets higher than a threshold. Also, if the target combinational circuit is not a single-bit function and it is possible to detect which output bit is faulty, one can measure Δt for each output bit independently.

Therefore, we define the adversary model and define his capabilities in order to be able to measure Δt^i of the target combinational circuit for the given input i:

- The adversary has access to and can control the clock signal which triggers the registers providing the input and saving the output of the target combinational circuit.
- He knows in which clock cycle the target combinational circuit processes the desired data, e.g., known or guessed input/output.
- He can control the target device in a way that the same input value i is repeatedly processed by the target combinational circuit during shortening the time interval of the clock glitch.
- He is equipped with appropriate instruments to shorten the duration of the clock glitch with suitable accuracy.

2.4 Definitions

Bitwise Capture: $\mathrm{BitCap}^i_{b,\Delta t}$ is the result of a *Bernoulli trial* whether the output of the target combinational circuit at bit b is faulty while processing the input i and when Δt is the time interval of the clock glitch. Correspondingly, $\rho^i_{b,\Delta t}$ is defined as the probability of "success" in independently repeated $\mathrm{BitCap}^i_{b,\Delta t}$ trials.

Capture: $\mathrm{Cap}^i_{\Delta t} = \bigvee\limits_b \mathrm{BitCap}^i_{b,\Delta t}$. In other words, $\mathrm{Cap}^i_{\Delta t}$ is the same as the above defined trial regardless of a certain output bit, and is meaningful when differentiating between different faulty output bits is not possible, e.g., if a circuit is equipped with a fault detection scheme and prevents the propagation of faulty results. $\rho^i_{\Delta t}$ is also the probability of "success" in independently repeated $\mathrm{Cap}^i_{\Delta t}$ trials.

Time: To represent the timing characteristics of the target combinational circuit, we define $T_b^i = \Delta t;\ \rho^i_{b,\Delta t} \approx \rho_{TH}$ as the time required to compute the corresponding output bit b to input i, where ρ_{TH} is a threshold for the probability and is defined based on physical characteristics of the target circuit and is also based on the maximum probability achieved by shortening Δt. Accordingly, the time required to complete the computation of all bits when processing input i is defined as $T^i = \Delta t;\ \rho^i_{\Delta t} \approx \rho_{TH}$.

Remark: Depending on the target device, its architecture, and the role of the target combinational circuit inside the target device, it might not be possible to know the input i processed. However, if the output of the target combinational circuit is accessible, one can make all the above defined terms based on the fault-free output o, i.e., $\mathrm{BitCap}^o_{b,\Delta t}$, $\rho^o_{b,\Delta t}$, $\mathrm{Cap}^o_{\Delta t}$, $\rho^o_{\Delta t}$, T^o_b, and T^o.

3 Collision Timing Attack

For simplicity let us suppose that the target combinational circuit is an S-box of the first round of an AES encryption, i.e., $\mathrm{Sbox(i \oplus k)}$, where i is the corresponding input plaintext byte and k the target key byte.

If (bitwise) timing characteristics of an S-box i.e., T^i (T^i_b), show a diversity of Δt depending on input i, one can perform an attack and recover the secret knowing how the secret k contributes in T^i (T^i_b). In other words, if the timing characteristics of an S-box itself regardless of k and prior key addition $(\oplus)$ are known as an extra information or are obtained by profiling using a circuit similar to the target, one can make a hypothetical leakage function and examine its similarity to T^i (T^i_b) for each key guess. A similar approach has been presented in [13], where the timing characteristics of an AES S-box implementation were profiled and an attack similar to a correlation power analysis using a HW model was successfully performed. In fact, a set of $\mathrm{Cap}^o_{\Delta t}$ for a specific Δt is used in [13] to mount the attack at the last round of the AES encryption.

One may also try using information theoretic tools, e.g., mutual information analysis [9], to overcome the uncertainty of the leakage model. However, it is necessary to use a suitable leakage model that cannot be selected without extra knowledge about the (timing) characteristics of the target combinational function, or several different models must be examined to find a suitable one. It is noteworthy that the leakages $(\mathrm{Cap}^i_{\Delta t})$ consist of only two values ("fail" and "success"). This causes probability distributions (used in e.g., mutual information analysis) to be represented by only two bins in a histogram, and using other schemes to estimate the probability distributions, e.g., kernel density estimation, in this case leads to increasing the noise. Here, when using histograms, mutual information will also be identical to the variance of means.

In contrary to a correlation attack with a leakage model or a mutual information analysis using a distinguisher, we apply a correlation collision attack [14] to avoid the necessity of considering any leakage model or distinguisher. Here the

Algorithm 1 Correlation timing attack algorithm (at the last round of an AES encryption)

```
Input: $T1^o : \left(\varDelta t^{o=0}, \varDelta t^{o=1}, \dots, \varDelta t^{o=255}\right)$; $o = \mathrm{Sbox}(i) \oplus k1$
Input: $T2^o : \left(\varDelta t^{o=0}, \varDelta t^{o=1}, \dots, \varDelta t^{o=255}\right)$; $o = \mathrm{Sbox}(i) \oplus k2$
for $0 \le \varDelta \le 255$ do
    $C^{\varDelta} = \mathrm{Correlation}(T1^o, T1^{o \oplus \varDelta})$
end for
return $\arg\max_{\varDelta} C^{\varDelta}$
```

correlation collision attack compares the timing characteristics T^i (T_b^i) of two S-box instances running on two input sets, each of which is previously XORed by a secret key byte. Suppose that $T1^i$ and $T2^i$ (or their corresponding bitwise versions) are the timing characteristics of the S-box when processing $\mathrm{Sbox}(i \oplus k1)$ and $\mathrm{Sbox}(i \oplus k2)$ respectively. As stated in [14], the aim of a correlation collision attack is to find the linear difference between $k1$ and $k2$, i.e., $\varDelta = k1 \oplus k2$.

This can be extended when attacking the last round of the AES encryption, thanks to the absence of the MixColumns in the last round. Means, supposing $T1^o$ and $T2^o$ as the timing characteristics of the S-box followed by the key addition when calculating $o = \mathrm{Sbox}(i) \oplus k1$ and $o = \mathrm{Sbox}(i) \oplus k2$, the correlation collision attack can – exactly similar to the previous case – recover $\varDelta = k1 \oplus k2$ comparing $T1^o$ and $T2^o$ for all possible guesses of $\varDelta$. For clarification of the attack scheme see Algorithm 1.

4 Practical Evaluation

We are attacking the AES implementations of three ASIC chips built for the SASEBO-R board, namely the SASEBO LSI2 (130*n*m), LSI2 (90*n*m), and LSI3 (65*n*m). Each chip consists in the same 14 different cores, which we group in *i*) unprotected (mainly varying in the style of the S-box implementation), *ii*) DPA protected (from masked-AND gates [22] and threshold implementation [15] to MDPL [17], WDDL [21], and Pseudo-RSL [19]), and *iii*) fault attack protected [20].

As stated previously, we are using a similar approach for fault injection as in [13]. An additional external clock, generated by an programmable digital function generator, is fed into the SASEBO-R control FPGA where it is multiplied by a factor of 32 using a Digital Clock Management (DCM) unit. This fast clock signal is then used together with some logic to shape the glitchy clock signal. An internal circuit controls the clock signal of the LSI to infer the glitchy clock at the preferred instance of time synchronized to the AES computation of the target core.

We have first tried to generate the glitchy clock inside the control FPGA without using an external function generator, but the width of the glitchy clock could only be adjusted in large steps (e.g., of around 170 *p*s [8]), which were not small enough to reach the desired results. Therefore, we had to use a function generator to externally provide the precise clock frequencies. As it is represented in the following, we change the width of the glitchy clock in steps of 25 ps to 1 ps. Also, the multiplication of the clock frequency is necessary because of the limitation (maximum frequency of 15 MHz) of the function generator we have used, while the frequencies necessary to inject a fault in the combinational circuit are up to 300MHz. Also, the DCMs inside the Virtex2 control FPGA of the SASEBO-R can, when fed with a low frequency input signal, only generate output frequency up to 210 MHz. Since some of the cores, especially of the 65nm LSI3, require a higher frequency for fault injection, for these cores it was necessary to daisy chain two DCMs, one for generating a high frequency signal out of the function generator output and another one to reach the maximum supported output frequency which can only be generated by the DCM using a high frequency input [23].

In all cores of the LSIs 16 instances of the S-box are implemented to perform the complete SubBytes operation in each clock cycle. According to [2, 3], all cores – except the one supporting a counter mode and the fault-protected one – realize a round-base architecture, i.e., S-boxes and MixColumns are performed simultaneously in each clock cycle except for the last round where MixColumns is absent. Therefore, extracting the timing characteristics of the S-boxes in the first 9 rounds is not easily possible, and one needs to inject and play with the width of the clock glitches in the last round, when the target cores only compute the SubBytes operation followed by the final key addition and the result is stored in registers (similar scheme as used in [13]). In addition, one can see from the design architecture of the cores (see Fig. 5.1 of [2] and Fig. 13.1 of [3]) that the round key of the last round is already computed in the previous round and is stored into a register. The glitchy clock at the last round, hence, does not affect the key scheduling computations.

In the following the results of the attacks on the different cores and different LSIs are presented. Because of the high number of broken cores, only a subset of the performed attacks are presented in detail, giving additional information about the differences to the not mentioned cores as required.

From the 14 AES cores of each LSI, all cores have been successfully broken. These 14 cores could be broken on each of the 3 LSIs regardless of the process technology, giving a total of 42 successfully broken ASIC implementations using our proposes collision timing attack.

4.1 Attacking the Unprotected Cores

We start showing the results of the attack on the first AES core of the 130nm chip, namely AES_Comp, whose S-boxes have been made using a composite field approach. As stated before, 16 separate S-box instances have been implemented which are active at the same time. Therefore, it is not possible to compare the timing characteristics of one S-box instance when processing e.g., two values with different key bytes, that would be an ideal case for a timing collision attack. In contrast, the timing characteristics of different S-box instances must be

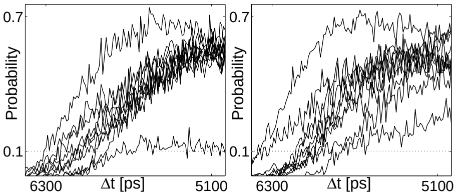


Fig. 1. First 10 $\rho^{o}_{b=0,\Delta t}$ curves for S-box instances no. (left) 0 and (right) 4 of AES_Comp (130nm)

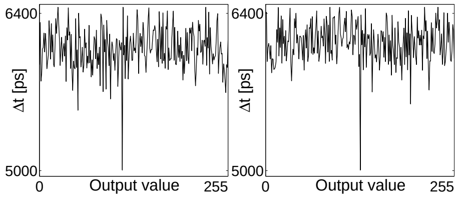
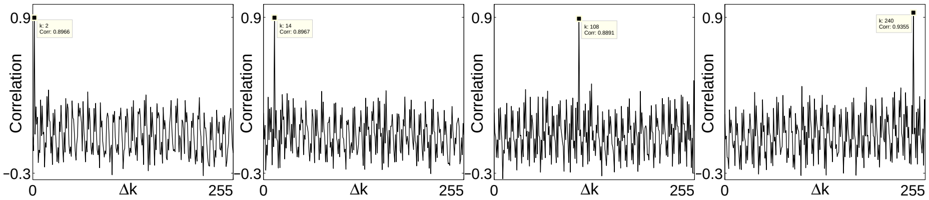


Fig. 2. Bitwise timing characteristics $T^{o}_{b=0}$ of S-box instances no. (left) 0 and (right) 4 of AES_Comp (130nm)



(a) (b) (c) (d)

Fig. 3. Result of the attack on the last round of AES_Comp (130nm) recovering Δk between key bytes (a) (0,1), (b) (0,2), (c) (0,3), (d) (0,4)

compared, which may slightly vary because of different placement and routing even when using the same netlist.

Since changing the glitchy clock width in our setup requires reseting the DCM(s), we have collected $\text{BitCap}^{o}_{b,\Delta t}$ for a specific Δt while random plaintexts are given to the core. This was repeated shortening Δt by a step of 12ps and finally exploiting the bitwise timing characteristics T^{o}_{b}. Figure 1 shows $\rho^{o}_{b,\Delta t}$ of the LSB (i.e., $b = 0$) for some output byte values of two S-box instances[1] extracted from their corresponding bitwise captures (10 000 captures for each Δt). Also, Fig. 2 presents the bitwise timing characteristics $T^{o}_{b=0}$ of these two S-box instances obtained by defining $\rho_{TH} = 0.1$ (as can be seen in Fig. 1). The diversity of Δt for these two S-boxes shows the dependency between the timing characteristics and the output values. Performing the attack Algorithm 1 on $T^{o}_{b=0}$ of S-box instance number 0 and all other instances led to recovering all 15 independent relations betweenthe 16 bytes of the last round key; part of the result is shown in Fig. 3. The attack works the same considering other output bits to derive T^{o}_{b} as well as on other LSI chips.

In order to perform the attack on the cores AES_PPRM1, AES_PPRM3, AES_Comp_ENC_top, and AES_PKG the same procedures as explained above have been repeated. As a reference for the timing characteristics and the number of captures collected to mount the attack on different cores in different LSIs, we

[1] S-box instance numbers start from 0 and are corresponding to ciphertext byte indexes.

have provided a list shown in Table 1. Attacking the AES_TBL core, where S-boxes have been realized by look-up tables (case statements), is different to the aforementioned cores. We illustrate this case when explaining how to mount the attack on the WDDL and MDPL cores.

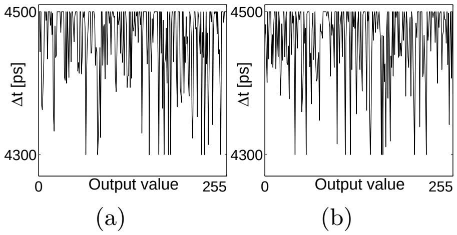
The AES_CTR core, which enables the counter mode, employs a 4-stage pipeline architecture to speed up the computations (see Fig. 5.6 of [2] and Fig. 13.6 of [3]). The S-box is divided into three parts, where the longest part is the inversion; therefore, we have selected this part when computing the last round to exploit the timing characteristics and mount the attacks. As stated in Section 2.3, we required to repeat the encryptions with the same IV – knowing both plaintexts and ciphertexts – to exploit the timing characteristics. Means, the adversary must have control over the IV; otherwise, he cannot determine whether the output is faulty. According to Table 4.7 and Table 4.15 of [2], the AES_CTR core has a longer critical path and is slower compared to the AES_Comp core. Indeed, it comes from the integer addition module used to increment the IV. However, the encryption core itself is quite fast, and because of its pipeline architecture we expected a shorter critical path for the inversion unit. Our practical experiments confirmed our assumption, and 4200ps was the minimum Δt for fault-free operation of 130nm chip showing the operation frequency of around 240MHz. Although we faced several problems making faulty results in the AES_CTR core of the 90nm and 65nm chips because of the very short critical path, the mounted attacks could successfully reveal the secrets.

4.2 Attacking the DPA-Protected Cores

Most of the DPA-protected cores can be attacked in the same way as the unprotected ones. In Fig. 4 one can see that even when using the masked AND-gates of the AES_MAO (65nm) core, the timing characteristics for different outputs still differ. Consequently, it is possible to extract the relation between the key bytes, which is depicted in Fig. 5. Interestingly the randomness provided by the masked gates does not have much impact on the timing characteristics, as shown in Fig. 5, where the results after obtaining 10 000 captures (the same technique as used to attack the unprotected cores), when shortening Δt with a step of 25ps are presented.

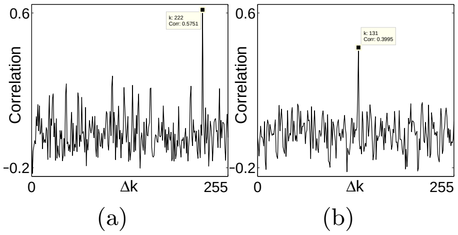
Attacking the other DPA-protected cores is the same except on those realizing WDDL and MDPL logic styles. The result of the attack on the AES_WO core, which is implemented using an pseudo-RSL [19] logic style, is shown in Fig. 6 and Fig. 7. Although we have used 10 000 captures for each Δt in steps of 10ps to attack the AES_WO core, attacking the AES_PR core (which is another realization of pseudo RSL) and the AES_TI (which is a threshold implementation using 4 shares) required considerably more captures. As stated in Table 1 we have used 1 000 000 captures for each of these cores to successfully mount the attack. To the best of our knowledge it is due to the amount of randomness provided by the DPA countermeasures.

The AES_WDDL and AES_MDPL cores require an slightly adjusted approach, since they need two clock cycles per round because of the used master-



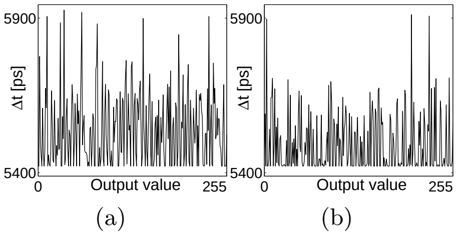
(a) (b)

Fig. 4. Bitwise timing characteristics $T^o_{b=0}$ of S-box instances no. (a) 5 and (b) 6 of AES_MAO (65nm)



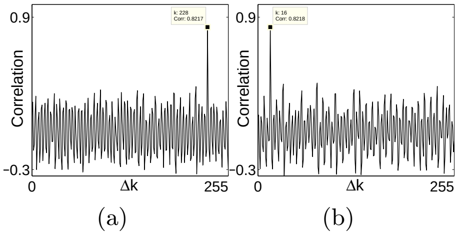
(a) (b)

Fig. 5. Result of the attack on the last round of AES_MAO (65nm) recovering Δk between key bytes (a) (5,6), (b) (5,7)



(a) (b)

Fig. 6. Bitwise timing characteristics $T^o_{b=1}$ of S-box instances no. (a) 7 and (b) 8 of AES_WO (90nm)



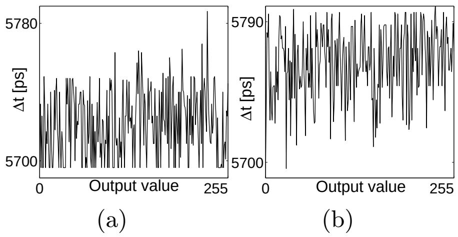
(a) (b)

Fig. 7. Result of the attack on the last round of AES_WO (90nm) recovering Δk between key bytes (a) (7,8), (b) (7,9)

slave flip-flops. Also, an injected fault by a clock glitch at the evaluation phase can only lead to a bit flip from 1 to 0, not vice versa, because of the pre-discharge phase of both WDDL and MDPL styles. Interestingly, we have seen – for reasons unknown to us – the same behavior when attacking the AES_TBL core. Therefore, bitwise timing characteristics T^o_b does not provide any information for those output values o in which bit b is zero. Our solution is to avoid using bitwise characteristics, and apply the attack on timing characteristics T^o, e.g., those shown in Fig. 8, which are of the AES_MDPL (65nm) core. Two attack results are also shown in Fig. 9. It should be noted that, to attack the AES_MDPL and AES_WDDL cores, we only used 10 000 captures for each Δt with steps of 5ps. On the other hand, a successful attack on the AES_TBL required around 1 000 000 captures, which might be because of marginal differences between the critical paths of the circuit realizing the look-up table.

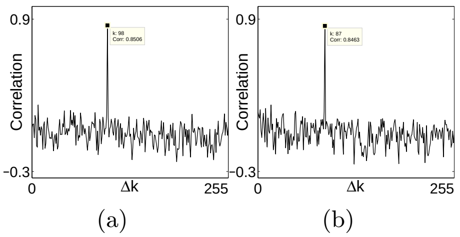
4.3 Attacking the Fault-Protected Core

The scheme presented in [20] has been used to implement the fault-protected core where the round function is split into two parts. The first part consists of the ShiftRows and SubBytes transformation and their inverse, while the second



(a) (b)

Fig. 8. Timing characteristics T^o of S-box instances no. (a) 2 and (b) 3 of AES_MDPL (65nm)



(a) (b)

Fig. 9. Result of the attack on the last round of AES_MDPL (65nm) recovering Δk between key bytes (a) (2,3), (b) (2,6)

part consists of the MixColumns transformation and AddRoundKey, also both for encryption and decryption.

While one of the parts is used for computing the next internal state, the other one is used to check the validity of the previous computation by applying the inverse operation and comparing the result to the original state. As an example, during the first clock cycle of the first round only the ShiftRows and SubBytes operations are performed. Then, during the second clock cycle, the result is sent through the MixColumns and AddRoundkey circuit while simultaneously performing InverseShiftRows and InverseSubBytes on the same data thereby recomputing the original round input to be checked with the stored one. The second half of the first round computation is checked in the same way during the first clock cycle of the second round by applying the InverseMixColumns and InverseAddRoundkey. In sum, a complete encryption needs 21 clock cylces (for detailed information about this architecture see Fig. 5 of [20]).

Since each computation is checked by its inverse counterpart, static faults can also be detected. In theory the performance penalty of this countermeasure is low, since, as stated in [20], while the double amount of clock cycles are necessary for a complete AES computation, at the same time the critical path is shortened by splitting up the round function. However, as it is explained later we have found that because of the comparison function, which compares two 128-bit values, the critical path of the circuit gets quite long and makes the circuit considerably slower compared to the other cores (as can also be seen in Table 4.7 and Table 4.15 of [2]).

In addition, since MixColumns does not immediately follow SubBytes and they are separated by a register, in contrary to the other cores one can exploit the timing characteristics of the S-boxes at the first round and also extend the attack on the later rounds. In order to mount the attack on the first round there are two options:

- If the glitchy clock appears in the first clock cycle of the first round (see Fig. 5(a) of [20]), the faulty result is saved into the D0 register, which is detected in the next clock cycle by the comparison circuit. When we have tried this on different LSIs, 6600ps, 5500ps, and 4800ps were the minimum

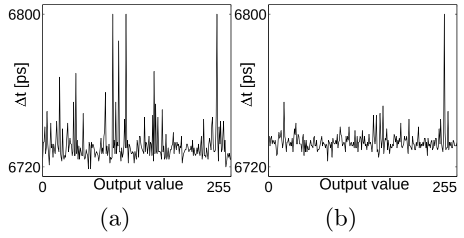


Fig. 10. Timing characteristics T^i of S-box instances no. (a) 0 and (b) 2 of AES_FA (65nm)

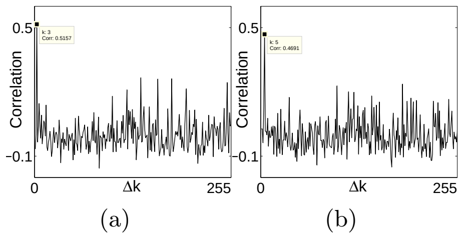


Fig. 11. Result of the attack on the first round of AES_FA (65nm) recovering Δk between key bytes (a) (0,2), (b) (0,4)

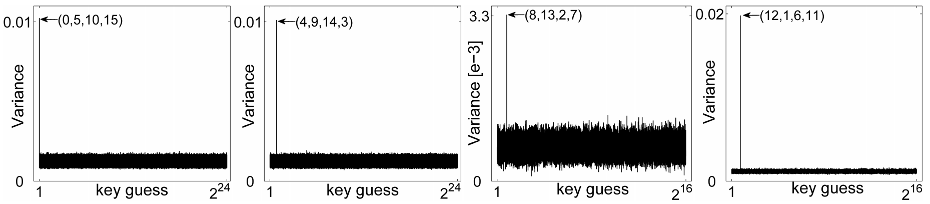
width of the clock glitch with which respectively the 130nm, 90nm, and 65nm chips still worked fault free.

- If the glitchy clock is injected in the second clock cycle (Fig. 5(b) of [20]), where the critical path of InverseShiftRows and InverseSubBytes followed by the comparator unit is longer than MixColumns and AddRoundkey, the fault detection bit (output of the comparator circuit) is monitored while the computation of the InverseSubBytes or the comparison has not finished yet. Examining this in practice showed that the faults get detected when the width of the clock glitch is less than 10 700ps, 6700ps, and 6800ps for the 130nm, 90nm, and 65nm chips respectively. In fact, it shows that the increase of the critical path caused by the comparison circuit strongly affects the performance (throughput) of the design, as stated previously.

It is noteworthy that the faults in the key schedule unit are not detected except in the last round by comparing the last round key with a pre-computed one. The glitchy clocks may affect the round key computations, but since it is performed only once per two clock cycles, inferring clock glitches in the second clock cycle does not affect the key scheduling. Therefore, we have selected the second clock cycle to shorten the width of the glitchy clock and mount the attack.

In contrary to the other cores, here the fault detection unit provides only one bit indicating the appearance of a fault, but from the adversary point of view it is not possible to distinguish where (in which S-box instances) the fault occurred. Therefore, not only the bitwise capturing is infeasible, but also the fault detection unit makes the capturing a very imprecise estimation process. Two timing characteristics of two S-box instances of the AES_FA (65nm) core obtained by $\rho_{TH} = 0.5$ and two attack results are shown in Fig. 10 and in Fig. 11 respectively. We have used 500 000 captures for each clock glitch width when shortening Δt by a step of 10ps.

As stated before, the capturing process here is very inaccurate because of the effect of the fault occurrence on other S-box instances when considering a specific S-box instance. In other words, when at a specific Δt an S-box instance is always faulty, it prevents extracting any information about the behavior of the other S-box instances at that Δt. Therefore, we could only recover the relation



(a) first column (b) second column (c) third column (d) fourth column

Fig. 12. Results of the variance check approach on the captures collected using the glitchy clock at the second round of AES_FA (65nm)

between some key bytes, e.g., between bytes (0,2), (0,4), (0,6), (0,7), and (0,10). We should emphasize that this behavior is not the same for all LSIs; different relations between key bytes are recovered when attacking the first round. Our solution, to completely break the core and recover all the key bytes, is to extend the attack on the second round. When the glitchy clock is injected in the second clock cycle of the second round, we have collected the result of 50 000 trials with random plaintext for a specific Δt when the probability of the fault occurrence is about 0.5. The attack process is divided into 4 parts, targeting each 4-byte MixColumns output independently. We start with the first column when the key bytes (0,5,10,15) of the first round key are required to compute the first MixColumns output. Since the relation between key bytes e.g., (0,10) has been revealed by the last attack shown, the search space for the key bytes (0,5,10,15) is reduced to 2^{24}. Therefore, for each key guess the output of the MixColumns is computed, and based on each MixColumns output byte the 4 captures and 4 probabilities using the collected 50 000 trails are generated. Interestingly, it is not required to mount the collision attack. Instead, a variance check approach as stated in [14] can find the correct key guess. The idea behind it is that the probabilities ρ^i are different for different input values. For a wrong key guess the trials will be randomly distributed into the captures, and the probabilities have a small variance compared to the case that the captures have been made by the correct key guess. The result of this approach is shown in Fig. 12(a), where the correct key guess is obviously distinguishable. Using a 16-core PC to compute all variance values for the 2^{24} key space lasted roughly one hour. Since there are 4 captures and 4 probabilities ρ^i each of which corresponds to one MixColumns output byte, the variance check can be done on each of these 4 probabilities independently. So, if the variances of probabilities of one output byte did not show any distinguishable guess, other output bytes can be examined. This process is repeated on the other columns using the recovered key bytes and the recovered relations so that the search space sometimes gets smaller than 2^{24}. Although different relations between key bytes are recovered when attacking different LSIs, the variance check approach at the second round could completely reveal the secret in all LSIs searching in a space up to 2^{24}.

4.4 Difficulties

During the practical experiences, which have partially been shown above, we faced several problems which required a couple of engineering hours to be solved. In sum it took around 6 months to successfully mount the attacks on all cores of all LSIs. Some of the problems which we dealt with are listed below:

- As stated in [2] and [3], each core has its own clock tree which had a strong impact on glitchy clocks. The capacitive and resistive features of the clock line of the LSIs which is supplied by the control FPGA changed the glitchy clock shape and modified the situation whether the registers are triggered two times, or if one of the positive edges is filtered by the clock tree elements. Therefore, we had to put different capacitors and resistors in the SASEBO-R board to change the rising and falling slopes of the clock signal to reach the desired situation.
- When injecting the clock glitch at the last round we saw that some output bytes – and even some output bits – get faulty very rapidly compared to the slow slope of ρ^o. For instance, see Fig. 13, where some probability curves arise very sharply and make their corresponding attacks impossible.The reason for such a behavior is that some registers do not respond to both rising edges of the glitchy clock but are triggered only once. Consequently, the timing characteristics of those related S-boxes could not be extracted. Again playing with the capacitance and resistance of the clock line was our solution.
- Since the temperature has an effect on the critical path and the speed of the ASICs, some values are given in Table 4.7 of [2], we had to not only keep the room temperature constant during capturing, but also keep the board and the LSIs in different temperatures while playing with capacitances and resistances to solve the problem mentioned above.
- Since DCMs outputs have an increased jitter when they are used to multiply the clock inputs, the glitchy clock width also had significant jitter that made the capturing process noisy. The situation got worse when we had to cascade two DCMs for some cores, especially in $65n$m, to reach the desired Δt. In this case, the second DCM often could not get locked because of the high jitter of the first DCM. So, we had to provide another circuit controlled by the PC to automatically reset the control FPGA (in fact the DCMs) until the DCMs get locked and provide the requested high frequencies.
- Since we have required high amounts of captures, e.g., 1 000 000, for different Δt values to successfully mount the attacks, we have developed a special design for the control FPGA to speed up the capturing process. Our control FPGA communicates with the target LSI, makes the glitchy clock on the desired clock cycle, and finally after performing a couple of capturing process sends the result back to the PC. In this way we could efficiently increase the speed of the capturing up to couple of thousands per second.
- According to [2] and [3], the clock signal of the interface circuit of the LSIs is separated from the core clocks. So, the glitchy clock does not appear on the interface circuit which makes the attacks easier. It might be a challenging

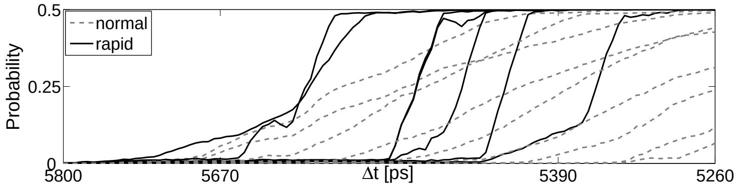


Fig. 13. Average of $\rho^{o}_{b=1}$ for all 16 S-boxes of AES_MDPL (65nm)

case when the interface circuit sees the glitches, and the control flow of the target core gets infected.

5 Conclusions

We have presented a collision attack which efficiently utilizes the data dependent timing characteristics of combinational circuits to reveal the secrets. While the attack is based on the idea of [13], it is significantly more powerful since it does not require any knowledge about the characteristics of the target combinational circuit.

We demonstrated the power of the attack by successfully breaking all 14 AES implementations of the SASEBO LSI2 and LSI3 in all currently available process technologies. It is indicated in [13] that while masking does not prevent DFA attacks, it may actually provide security against FSA-based attacks because of the randomized inputs of the combinatorial functions. However, breaking all DPA-protected cores of the mentioned ASICs we have shown that randomizing countermeasures itself cannot prevent data-dependent timing characteristics of the combinational circuit, and they therefore remain vulnerable against the attack introduced here. Finally, our introduced attack is even capable of extracting the secret key out of an implementation applying a fault detection scheme by extending the attack on the second round of the cipher and a few hours of computation. In short, the results shown in this work imply the need for a special unit in the – specially side-channel protected – designs in order to detect the clock glitches to thwart such a kind of attacks.

Acknowledgment

The authors would like to thank Akashi Satoh and Research Center for Information Security (RCIS) of Japan for the prompt and kind help in obtaining SASEBOs and crypto LSIs.

References

1. Side-channel Attack Standard Evaluation Board (SASEBO-R). Further information are available via `http://staff.aist.go.jp/akashi.satoh/SASEBO/en/board/sasebo-r.html`.
2. ISO/IEC 18033-3 Standard Cryptographic LSI – with Side Channel Attack Countermeasures – Specification, ver 1.0. `http://staff.aist.go.jp/akashi.satoh/SASEBO/resources/crypto_lsi/CryptoLSI2_Spec_Ver1.0_English.pdf`, 2009.
3. Standard Cryptographic LSI Specification – Countermeasures against Side Channel Attacks (65nm) – Specification, ver 0.9. `http://staff.aist.go.jp/akashi.satoh/SASEBO/resources/crypto_lsi/CryptoLSI3_Spec_Ver0.9_English.pdf`, 2010.
4. E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO 1997*, volume 1294 of *LNCS*, pages 513–525. Springer, 1997.
5. J. Blömer and J.-P. Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In *FC 2003*, volume 2742 of *LNCS*, pages 162–181. Springer, 2003.
6. A. Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. In *CHES 2008*, volume 5154 of *LNCS*, pages 30–44. Springer, 2008.
7. A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke. Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs. In *CT-RSA 2010*, volume 5985 of *LNCS*, pages 235–251. Springer, 2010.
8. S. Endo, T. Sugawara, N. Homma, T. Aoki, and A. Satoh. An on-chip glitchy-clock generator and its application to safe-error attack. In *COSADE 2011*, pages 175–182, 2011.
9. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual Information Analysis. In *CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.
10. C. Giraud. DFA on AES. In *AES Conference 2004*, volume 3373 of *LNCS*, pages 27–41. Springer, 2004.
11. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
12. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
13. Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta. Fault Sensitivity Analysis. In *CHES 2010*, volume 6225 of *LNCS*, pages 320–334. Springer, 2010.
14. A. Moradi, O. Mischke, and T. Eisenbarth. Correlation-Enhanced Power Analysis Collision Attack. In *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, 2010. The extended version is available on ePrint `http://eprint.iacr.org/2010/297`.
15. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 2006*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.
16. G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In *CHES 2003*, volume 2779 of *LNCS*, pages 77–88. Springer, 2003.
17. T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In *CHES 2005*, volume 3659 of *LNCS*, pages 172–186. Springer, 2005.

18. J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In *E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.
19. M. Saeki, D. Suzuki, K. Shimizu, and A. Satoh. A Design Methodology for a DPA-Resistant Cryptographic LSI with RSL Techniques. In *CHES 2009*, volume 5747 of *LNCS*, pages 189–204. Springer, 2009.
20. A. Satoh, T. Sugawara, N. Homma, and T. Aoki. High-Performance Concurrent Error Detection Scheme for AES Hardware. In *CHES 2008*, volume 5154 of *LNCS*, pages 100–112. Springer, 2008.
21. K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE 2004*, pages 246–251. IEEE Computer Society, 2004.
22. E. Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. Cryptology ePrint Archive, Report 2003/236, 2003. `http://eprint.iacr.org/`.
23. XILINX. Virtex-II Pro and Virtex-II Pro X FPGA User Guide. Technical report, version 4.2, 2007. `http://www.xilinx.com/support/documentation/user_guides/ug012.pdf`.

Appendix

Table 1. Specification of all 14 AES cores of 3 LSIs including the Δt ranges and the number of captures used to mount the attacks

IP core	Description	LSI2 130nm		LSI2 90nm		LSI3 65nm	
		Δt range [ps]	No. of Captures	Δt range [ps]	No. of Captures	Δt range [ps]	No. of Captures
AES_Comp	composite field S-box	6450 $\Delta : 12$ 5000	10 000	5320 $\Delta : 10$ 5130	10 000	3650 $\Delta : 10$ 3370	10 000
AES_TBL	table look-up S-box by case statement	5475 $\Delta : 25$ 4900	1 000 000	3960 $\Delta : 20$ 3550	1 000 000	3570 $\Delta : 10$ 3420	1 000 000
AES_PPRM1	S-box by 1-stage AND-XOR	11350 $\Delta : 25$ 7775	10 000	6135 $\Delta : 20$ 5555	10 000	5325 $\Delta : 25$ 5000	10 000
AES_PPRM3	S-box by 3-stage AND-XOR	6425 $\Delta : 25$ 5150	10 000	5230 $\Delta : 10$ 5130	10 000	3650 $\Delta : 10$ 3420	10 000
AES_Comp ENC_top	composite field S-box, only encryption	6325 $\Delta : 25$ 5100	10 000	5200 $\Delta : 10$ 5130	10 000	3700 $\Delta : 10$ 3410	10 000
AES_CTR	composite field S-box, counter mode	4120 $\Delta : 20$ 3660	10 000	3310 $\Delta : 5$ 3260	10 000	3280 $\Delta : 5$ 3210	10 000
AES_FA Round 1	composite field S-box, fault detection	10600 $\Delta : 10$ 10440	50 000	6560 $\Delta : 5$ 6510	1 000 000	6800 $\Delta : 10$ 6670	500 000
Round 2		10700	50 000	7050	50 000	6980	50 000
AES_PKG	composite field S-box, precomp. roundkeys	6325 $\Delta : 25$ 5100	10 000	5360 $\Delta : 10$ 5130	10 000	3850 $\Delta : 10$ 3370	10 000
AES_MAO	DPA count. by Masked And Operation	8475 $\Delta : 25$ 6250	10 000	5900 $\Delta : 5$ 5850	10 000	4500 $\Delta : 25$ 4300	10 000
AES_MDPL	DPA count. by MDPL logic style	12825 $\Delta : 25$ 10850	10 000	9350 $\Delta : 25$ 8050	10 000	5800 $\Delta : 5$ 5260	10 000
AES_TI	DPA count. by Threshold Implementation	10860 $\Delta : 20$ 9800	1 000 000	5900 $\Delta : 5$ 5850	1 000 000	6340 $\Delta : 20$ 5940	1 000 000
AES_WDDL	DPA count. by WDDL logic style	6750 $\Delta : 10$ 5730	10 000	5250 $\Delta : 5$ 5150	50 000	3835 $\Delta : 5$ 3675	10 000
AES_PR	DPA count. by pseudo RSL logic style	31685 $\Delta : 10$ 31055	100 000	14400 $\Delta : 20$ 13840	100 000	6650 $\Delta : 25$ 6150	100 000
AES_WO	DPA count. by pseudo RSL (evaluation)	7575 $\Delta : 25$ 6475	10 000	5910 $\Delta : 10$ 5430	10 000	3900 $\Delta : 25$ 3600	10 000