# ASCA, SASCA and DPA with Enumeration: Which One Beats the Other and When?

Vincent Grosso(✉) and François-Xavier Standaert

ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Louvain-la-Neuve, Belgium
vincent.grosso@uclouvain.be

**Abstract.** We describe three contributions regarding the Soft Analytical Side-Channel Attacks (SASCA) introduced at Asiacrypt 2014. First, we compare them with Algebraic Side-Channel Attacks (ASCA) in a noise-free simulated setting. We observe that SASCA allow more efficient key recoveries than ASCA, even in this context (favorable to the latter). Second, we describe the first working experiments of SASCA against an actual AES implementation. Doing so, we analyse their profiling requirements, put forward the significant gains they provide over profiled Differential Power Analysis (DPA) in terms of number of traces needed for key recoveries, and discuss the specificities of such concrete attacks compared to simulated ones. Third, we evaluate the distance between SASCA and DPA enhanced with computational power to perform enumeration, and show that the gap between both attacks can be quite reduced in this case. Therefore, our results bring interesting feedback for evaluation laboratories. They suggest that in several relevant scenarios (e.g. attacks exploiting many known plaintexts), taking a small margin over the security level indicated by standard DPA with enumeration should be sufficient to prevent more elaborate attacks such as SASCA. By contrast, SASCA may remain the only option in more extreme scenarios (e.g. attacks with unknown plaintexts/ciphertexts or against leakage-resilient primitives). We conclude by recalling the algorithmic dependency of the latter attacks, and therefore that our conclusions are specific to the AES.

## 1 Introduction

**State-of-the-art.** Strategies to exploit side-channel leakages can be classified as Divide and Conquer (DC) and analytical. In the first case, the adversary recovers information about different bytes of (e.g.) a block cipher key independently, and then combines this information, e.g. via enumeration [36]. In the second case, she rather tries to recover the full key at once, exploiting more algorithmic approaches to cryptanalysis with leakage. Rephrasing Banciu et al., one can see these different strategies as a tradeoff between pragmatism and elegance [2].

In brief, the "DC+enumeration" approach is pragmatic, i.e. it is easy to implement, requires little knowledge about the target implementation, and can take advantage of a variety of popular (profiled and non-profiled) distinguishers,


T. Iwata and J.H. Cheon (Eds.): ASIACRYPT 2015, Part II, LNCS 9453, pp. 291–312, 2015.
DOI: 10.1007/978-3-662-48800-3_12

such as Correlation Power Analysis (CPA) [6], Mutual Information Analysis (MIA) [14], Linear Regression (LR) [34] or Template Attacks (TA) [8]. We will use the term Differential Power Analysis (DPA) to denote them all [22].

By contrast, analytical approaches are (more) elegant, since they theoretically exploit all the information leaked by an implementation (vs. the leakages of the first and/or last rounds independently for DC attacks). As a result, these attacks can (theoretically) succeed in conditions where the number of measurements available to the adversary is very limited. But this elegance (and the power that comes with it) usually implies stronger assumptions on the target implementation (e.g. most of them require some type of profiling). The Algebraic Side-Channel Attacks (ASCA) described in [30] and further analyzed in [7,32] are an extreme solution in this direction. In this case, the target block cipher and its leakages are represented as a set of equations that are then solved (e.g. with a SAT solver, or Groebner bases). This typically implies a weak resistance to the noise that is usually observed in side-channel measurements. As a result, various heuristics have been suggested to better deal with errors in the information leakages, such as [24,39]. The Tolerant Algebraic Side-Channel Attacks (TASCA) proposed in [25,26] made one additional step in this direction, by replacing the solvers used in ASCA by an optimizer. But they were limited by their high memory complexity (since they essentially deal with noise by exhaustively encoding the errors they may cause). More recently, two independent proposals suggested to design a dedicated solver specialized to byte-oriented ciphers such as the AES [16,27]. The latter ones were more efficient and based on smart heuristics exploiting enumeration. Eventually, Soft Analytical Side-Channel Attacks (SASCA) were introduced at Asiacrypt 2014 as a conceptually different way to exploit side-channel leakages analytically [38]. Namely, rather than encoding them as equations, SASCA describe an implementation and its leakages as a code, that one can efficiently decode using the Belief Propagation (BP) algorithm. As a result, they can directly exploit the (soft) information provided by profiled side-channel attacks (such as LR or TA), in an efficient manner, with limited memory complexity, and for multiple plaintexts. Concretely, this implies that they provide a natural bridge between DC attacks and analytical ones.

**Our Contribution.** In view of this state-of-the-art, we consider three open problems regarding DC and analytical strategies in side-channel analysis.

First, we observe that the recent work in [38] experimented SASCA in the context of noisy AES leakages. While this context allowed showing that SASCA are indeed applicable in environments where ASCA would fail, it leaves the question whether this comes at the cost of a lower efficiency in a noise-free context open. Therefore, we launched various experiments with noise-free AES leakages to compare ASCA and SASCA. These experiments allowed us to confirm that also in this context, SASCA are equally (even slightly more) efficient.

Second, the experiments in [38] exploited simulations in order to exhibit the strong noise-resilience of SASCA (since the amount of noise can then be used as a parameter of such simulations). But this naturally eludes the question of the profiling of a concrete device, which can be a challenging task, and for

which the leakage functions of different target intermediate values may turn out to be quite different [13]. Therefore, we describe the first working experiments of SASCA against an actual AES implementation, for which a bivariate TA exploiting the S-box input/output leakages would typically be successful after more than 50 measurements. We further consider two cases for the adversary's knowledge about the implementation. In the first one, she has a precise description in hand (i.e. the assembly code, typically). In the second one, she only knows AES is running, and therefore only exploits the generic operations that one can assume from the algorithm specification.[1] Our experiments confirm that SASCA are applicable in a simple profiled scenario, and lead to successful key recoveries with less traces than a DC attack (by an approximate factor up to 5). They also allow us to discuss the profiling cost, and the consequences of the different leakage functions in our target implementation. A relevant observation regarding them is that weak leakages in the MixColumns operations are especially damaging for the adversary, which can be explained by the (factor) graph describing an AES implementation: indeed, XORing two values with limited information significantly reduces the information propagation of the BP algorithm execution. This suggest interesting research directions for preventing such attacks, since protecting the linear parts of a block cipher is usually easier/cheaper.

Third, we note that SASCA are in general more computationally intensive than DC attacks. Therefore, a fair comparison should allow some enumeration power to the DC attacks as well. We complement our previous experimental attacks by considering this last scenario. That is, we compare the success rate of SASCA with the ones of DC attacks exploiting a computational power corresponding to up to $2^{30}$ encryptions (which corresponds to more than the execution time of SASCA on our computing platform). Our results put forward that SASCA remain the most powerful attack in this case, but with a lower gain.

**Summary.** These contributions allow answering the question of our title. First, SASCA are in general preferable to ASCA, with both noise-free and noisy AES leakages. Second, the tradeoff between SASCA and DC attacks is more balanced. As previously mentioned, DC attacks are more pragmatic. So the interest of SASCA essentially depends on the success rate gains it provides, which itself depends on the scenarios. If multiple plaintexts/ciphertext pairs are available, our experiments suggest that the gain of SASCA over DPA with enumeration is somewhat limited, and may not justify such an elegant approach. This conclusion backs up the results in [2], but in a more general scenario, since we consider multiple-queries attacks rather than single-query ones, together with more a powerful analytical strategy. By contrast, if plaintexts/ciphertexts are unknown (which renders DPA [17] and enumeration more challenging to apply), or if the number of plaintexts one can observe is very limited (e.g. by design, due to a leakage-resilient primitive [10]), SASCA may be the best/only option.

[1] Admittedly, such a generic scenario still assumes that the target implementation closely follows the specifications given in [11] which may not always be the case, e.g. for bitslice implementations [29], or T-table based implementations [9].

**Preliminary Remark.** Our focus in this paper is on a couple of extreme approaches to side-channel analysis, i.e. the most pragmatic DC attacks against 8-bit targets of the first AES round, and the most elegant ASCA/SASCA exploiting most/all such targets in the implementation. Quite naturally, the other analytical attacks mentioned in this introduction would provide various tradeoffs between these extremes. Besides, more computationally-intensive DPA attacks (based on larger key hypotheses) are also possible, as recently discussed by Mather et al. [23]. Such attacks are complementary and may further reduce the gain of SASCA over DPA, possibly at the cost of increased computational requirements (e.g. the latter work exploited high-performance computing whereas all our experiments were carried out on a single desktop computer).

## 2 Background

In this section we first describe the measurement setup used in our experiments. Then, we describe two tools we used to identify and evaluate information leakages in the traces. Finally, we recall the basics of the different attacks we compare.

### 2.1 Measurement Setup

Our measurements are based on the open source AES FURIOUS implementation (http://point-at-infinity.org/avraes) run by an 8-bit Atmel ATMEGA644p microcontroller at a 20 MHz clock frequency. We monitored the power consumption across a $22\Omega$ resistor. Acquisitions were performed using a Lecroy WaveRunner HRO 66 ZI providing 8-bit samples, running at 400 Msamples/second. For SASCA, we can exploit any intermediate values that appear during the AES computation. Hence, we measured the full encryption. Our traces are composed of 94 000 points, containing the key scheduling and encryption rounds. Our profiling is based on 256 000 traces corresponding to random plaintexts and keys. As a result, we expect around 1 000 traces for each value of each intermediate computation. We use $\boldsymbol{l}^i_{n,x}$ for the value $x$ of the $n^{th}$ intermediate value in the $i^{th}$ leakage trace, and $\boldsymbol{l}^i_{n,x}(t)$ when we access at the $t^{th}$ point (sample) of this trace.

### 2.2 Information Detection Tools

Since SASCA can exploit many target intermediate values, we need to identify the time samples that contain information about them in our traces, next referred to as Points Of Interest (POI). We recall two simple methods for this purpose, and denote the POI of the $n^{th}$ intermediate value in our traces with $t_n$.

**(a) Correlation Power Analysis (CPA) [6].** is a standard side-channel distinguisher that estimates the correlation between the measured leakages and some key-dependent model for a target intermediate value. In its standard version, an a-priori (here, Hamming weight) model is used for this purpose.

In practice, this estimation is performed by sampling (i.e. measuring) traces from a leakage variable $L$ and a model variable $M_k$, using Pearson's correlation coefficient:

$$\rho_k(L, M_k) = \frac{\hat{\mathsf{E}}[(L - \hat{\mu}_L)(M_k - \hat{\mu}_{M_k})]}{\sqrt{\hat{\mathsf{var}}(L)\hat{\mathsf{var}}(M_k)}}.$$

In this equation, $\hat{\mathsf{E}}$ and $\hat{\mathsf{var}}$ respectively denote the sample mean and variance operators, and $\hat{\mu}_L$ is the sample mean of the leakage distribution $L$. CPA is a univariate distinguisher and therefore launched sample by sample.

**(b) The Signal-to-Noise Ratio (SNR) [21].** of the $n^{th}$ intermediate value at the time sample $t$ can be defined according to Mangard's formula [21]:

$$\mathsf{SNR}_n(t) = \frac{\hat{\mathsf{var}}_x\left(\hat{\mathsf{E}}_i\left(\boldsymbol{l}^i_{n,x}(t)\right)\right)}{\hat{\mathsf{E}}_x\left(\hat{\mathsf{var}}_i\left(\boldsymbol{l}^i_{n,x}(t)\right)\right)}.$$

Despite connected (high SNRs imply efficient CPA if the right model is used), these metrics allow slightly different intuitions. In particular, the SNR cannot tell apart the input and output leakages of a bijective operation (such as an S-box), since both intermediate values will generate useful signal. This separation can be achieved by CPA thanks to its a-priori leakage predictions.

### 2.3 Gaussian Templates Attacks

Gaussian TA [8] are the most popular profiled distinguisher. They assume that the leakages can be interpreted as the realizations of a random variable which generates samples according a Gaussian distribution and work in two steps. In a profiling phase, the adversary estimates a mean $\hat{\mu}_{n,x}$ and variance $\hat{\sigma}^2_{n,x}$ for each value $x$ of the $n^{th}$ intermediate computation. In practice, this is done for the time sample $t_n$ obtained thanks to the previously mentioned POI detection tools. Next, in the attack phase and for each trace $\boldsymbol{l}$, she can calculate the likelihood to observe this leakage at the time $t_n$ for each $x$ as:

$$\hat{\Pr}[\boldsymbol{l}(t_n)|x] \sim \mathcal{N}(\hat{\mu}_{n,x}, \hat{\sigma}^2_{n,x}).$$

In the context of standard DPA, we typically have $x = p \oplus k$, with $p$ a known plaintext and $k$ the target subkey. Therefore, the adversary can easily calculate $\hat{\Pr}[k^*|p, \boldsymbol{l}(t_n)]$ using Bayes theorem, for each subkey candidate $k^*$:

$$\hat{\Pr}[k^*] = \prod_i \hat{\Pr}[k^*|p, \boldsymbol{l}^i(t_n)].$$

To recover the full key, she can run a TA on each subkey independently.

By contrast, in the context of SASCA, we will directly insert the knowledge (i.e. probabilities) about any intermediate value $x$ in the (factor) graph describing the implementation, and try to recover the full key at once.

Note that our SASCA experiments consider univariate Gaussian TA whereas our comparisons with DPA also consider bivariate TA exploiting the S-box input and output leakages (i.e. the typical operations that a divide-and-conquer adversary would exploit). In the latter case, the previous means and variances just have to be replaced by mean vectors and covariance matrices. This choice is motivated by our focus on the exploitation of multiple intermediate AES computations. It could be further combined with the exploitation of more samples per intermerdiate computation, e.g. thanks to dimensionality reduction [1].

### 2.4 Key Enumeration and Rank Estimation

At the end of a DC side-channel attack (as the previous TA), the attacker has probabilities on each subkey. If the master key is not the most probable one, she can perform enumeration up to some threshold thanks to enumeration algorithms, e.g. [36]. This threshold depends on the computational power of the adversary, since enumerating all keys is computationally impossible. If the key is beyond the threshold of computationally feasible enumeration, and in order to gain intuition about the computational security remaining after an attack, key rank estimation algorithms can be used [15,37]. A key rank estimation takes in input the list of probabilities of all subkeys and the probability of the correct key (which is only available in an evaluation context), and returns an estimation on the number of keys that are more likely than the actual key. Rank estimation allows to approximate $d^{th}$-order success rates (i.e. the probability that the correct key lies among the $d$ first ones rated by the attack) efficiently and quite accurately. The security graphs introduced in [37] provide a visual representation of higher-order success rates in function of the number attack traces.

### 2.5 Algebraic Side-Channel Attacks

ASCA were introduced in [30] as one of the (if not the) first method to efficiently exploit all the informative samples in a leakage trace. We briefly recall their three main steps and refer to previous publications for the details.

1. *Construction* consists in representing the cipher as an instance of an algebraic problem (e.g. Boolean satisfiability, Groebner bases). Because of their large memory (RAM) requirements, ASCA generally build a system corresponding to one (or a few) traces only. For example, the SAT representation of a single AES trace in [32] has approximatively $18,000$ equations in $10,000$ variables.

2. *Information extraction* consists in getting exploitable leakages from the measurements. For ASCA, the main constraint is that actual solvers require hard information. Therefore, this phase usually translates the result of a TA into deterministic leakages such as the Hamming weight of the target intermediate

values. Note that the attack is (in principle) applicable with any type of lekages given that they are sufficiently informative and error-free.

3. *Solving.* Eventually, the side-channel information extracted in the second phase is added to the system of equations constructed in the first phase, and generic solvers are launched to solve the system and recover the key. In practice, this last phase generally has large RAM requirements causing ASCA to be limited to the exploitation of one (or two) measurement traces.

Summarizing, ASCA are powerful attacks since they can theoretically recover a key from very few leakage traces, but this comes at the cost of low noise-resilience, which motivated various heuristic improvements listed in introduction. The next SASCA are a more founded solution to get rid of this limitation.

### 2.6 Soft Analytical Side-Channel Attacks

SASCA [38] describe the target block cipher implementation and its leakages in a way similar to a Low-Density Parity Check code (LDPC) [12]. Since the latter can be decoded using soft decoding algorithms, it implies that SASCA can directly use the posterior probabilities obtained during a TA. Similar to ASCA, they can also be described in three main steps.

1. *Construction.* The cipher is represented as a so-called "factor graph" with two types of nodes and bidirectional edges. First, variable nodes represent the intermediate values. Second, function nodes represent the a-priori knowledge about the variables (e.g. the known plaintexts and leakages) and the operations connecting the different variables. Those nodes are connected with bidirectional edges that carry two types of messages (i.e. propagate the information) through the graph: the type $q$ message are from variables to functions and the type $r$ messages are from functions to variables (see [20] for more details).

2. *Information extraction.* The description of this phase is trivial. The probabilities provided by TA on any intermediate variable of the encryption process can be directly exploited, and added as a function node to the factor graph.

3. *Decoding.* Similar to LDPC codes, the factor graph is then decoded using the BP algorithm [28]. Intuitively, it essentially iterates the local propagation of the information about the variable nodes of the target implementation.

Since our work is mostly focused on concrete investigations of SASCA, we now describe the BP algorithm in more details. Our description is largely inspired by the description of [20, Chapter 26]. For this purpose, we denote by $x_i$ the $i^{th}$ intermediate value and by $f_i$ the $i^{th}$ function node. As just mentioned, the nodes will be connected by edges that carry two types of messages. The first ones go from a variable node to a function node, and are denoted as $q_{v_n \to f_m}$. The second ones go from a function node to a variable node, and are denoted as $r_{f_n \to v_m}$. In both cases, $n$ is the index of the sending node and $m$ the index of the recipient node. The messages carried correspond to the scores for the different values of

the variable nodes. At the beginning of the algorithm execution, the messages from variable nodes to function nodes are initialized with no information on the variable. That is, for all $n, m$ and for all $x_n$ we have:

$$q_{v_n \to f_m}(x_n) = 1.$$

The scores are then updated according to two rules (one per type of messages):

$$r_{f_m \to v_n}(x_n) = \sum_{x_{n'}, n' \neq n} \big( f_m(x_{n'}, x_n) \prod_{n'} q_{v_{n'} \to f_m}(x_{n'}) \big). \tag{1}$$

$$q_{v_n \to f_m}(x_n) = \prod_{m' \neq m} r_{f_{m'} \to v_n}(x_n). \tag{2}$$

In Eq. 2, the variable node $v_n$ sends the product of the messages about $x_n$ received from the others function nodes ($m' \neq m$) to the function node $f_m$, for each value of $x_n$. And in Eq. 1, the function node $f_m$ sends a sum over all the possible input values of $f_m$ of the value of $f_m$ evaluated on the vector of $(x_{n'}, n' \neq n)$'s, multiplied by the product of the messages received by $f_m$ for the considered values of $x_{n'}$. The BP algorithm essentially works by iteratively applying these rules on all nodes. If the factor graph is a tree (i.e. if it has no loop), a convergence should occur after a number of iterations at most equal to the diameter of the graph. In case the graph includes loops (e.g. as in our AES implementation case), convergence is not guaranteed, but usually occurs after a number of iterations slightly larger than the graph diameter. The main parameters influencing the time and memory complexity of the BP algorithm are the number of possible values for each variable (i.e. $2^8$ in our 8-bit example) and the number of edges. The time complexity additionally depends on the number of inputs of the function nodes representing the block cipher operations (since the first rule sums over all the input combinations of these operations).

## 3 Comparison with ASCA

ASCA and SASCA are both analytical attacks with very similar descriptions. As previously shown in [38], SASCA have a clear advantage when only noisy information is available. But when the information is noise-free, the advantage of one over the other has not been studied yet. In this section, we therefore tackle the question "which analytical attack is most efficient in noise-free scenario?". To this end, we compare the results of SASCA and ASCA against a simulated AES implementation with noise-free (Hamming weight) leakages. We first describe the AES representation we used in our SASCA (which will also be used in the following sections), then describe the different settings we considered for our simulated attacks, and finally provide the results of our experiments.

### 3.1 Our Representation for SASCA

As usual in analytical attacks, our description of the AES is based on its target implementation. This allows us to easily integrate the information obtained

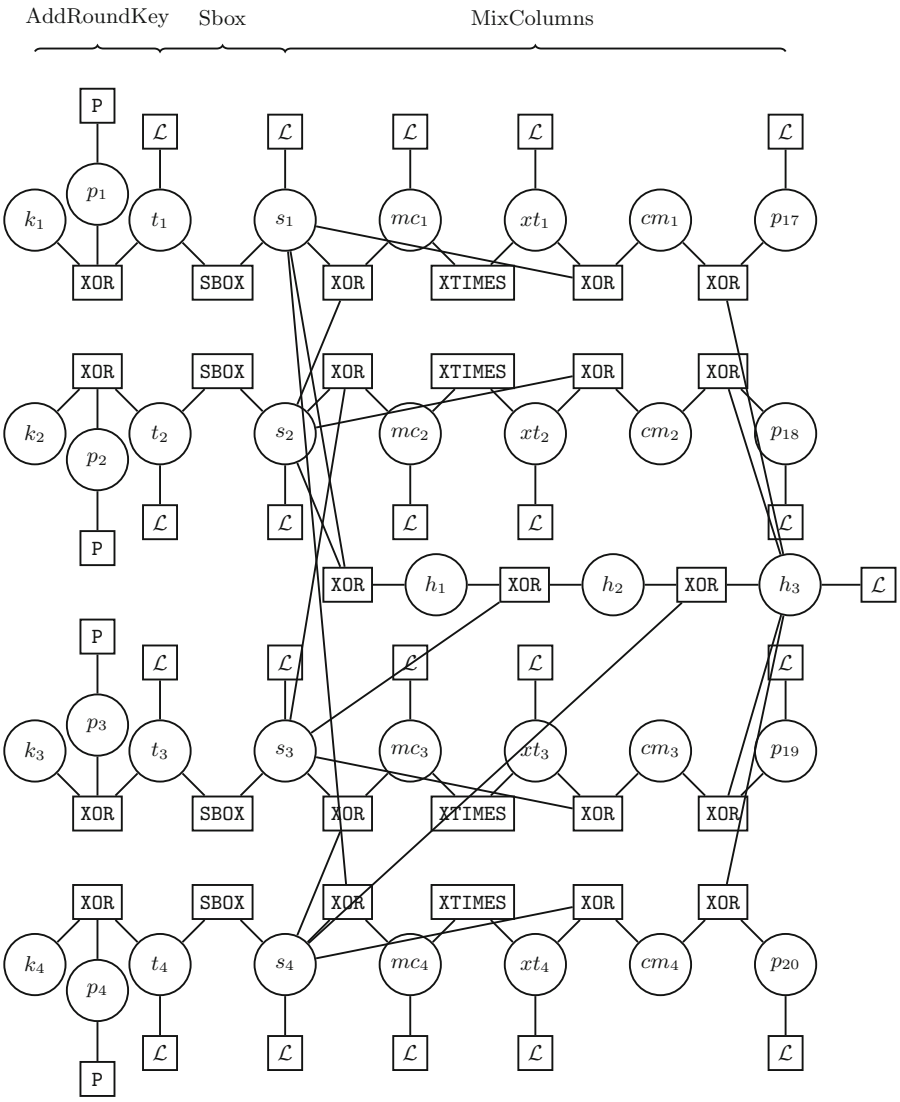


**Fig. 1.** Graph representation of one column of the first AES round.

during its execution. For readability purposes, we start by illustrating the graph representation for the first round of one column of the AES in Fig. 1. To build this graph for one plaintext, we start with 32 variable nodes (circles), 16 for the 8-bit subplaintexts ($p_i$), and 16 for the 8-bit subkeys ($k_i$). We first add a new variable node in the graph representation each time a new intermediate value is computed in the AES FURIOUS implementation,[2] together with the

[2] Excluding memory copies which only increase the graph diameter.

corresponding function nodes (rectangles). There are three different operations that create intermediate values. First, the Boolean XOR takes two variables as inputs and outputs a new variable that is equal to the bitwise XOR of the two inputs. Next, two memory accesses to look-up tables are used for the S-box and Xtimes operations, which take one variable as input, and create a new variable as output. We finally add two types of leaf nodes to these three function nodes. The P's reflect the knowledge of the plaintext used, and the $\mathcal{L}$'s give the posterior probability of the value observed using Gaussian templates. A summary of the different function nodes used in our AES factor graph is given in Table 1.

**Table 1.** Summary of the function nodes used in our AES factor graph.

| | |
|---|---|
| $\texttt{XOR}(a,b,c) = \begin{cases} 1 & \text{if } a = b \oplus c, \\ 0 & \text{otherwise.} \end{cases}$ | $\texttt{SBOX}(a,b) = \begin{cases} 1 & \text{if } a = sbox(b), \\ 0 & \text{otherwise.} \end{cases}$ |
| $\texttt{XTIMES}(a,b) = \begin{cases} 1 & \text{if } a = xtimes(b), \\ 0 & \text{otherwise.} \end{cases}$ | $\texttt{P}(x_n) = \begin{cases} 1 & \text{if } x_n = p, \\ 0 & \text{otherwise.} \end{cases}$ |
| $\mathcal{L}(x_n) = \Pr[x_n \vert \boldsymbol{l}(t_n)]$. | |

The graph in Fig. 1 naturally extends to a full AES execution. And when using several traces, we just keep a single description of the key scheduling, that links different subgraphs representing the different plaintext encryptions. Our description of the key scheduling requires 226 variable nodes and 210 function nodes. Our description of the rounds requires 1036 variable nodes and 1020 function nodes. The key scheduling nodes are connected by 580 edges, and each round of the encryption contains 292 edges. As a result and overall, the factor graph for one plaintext contains 1262 variable nodes, 1230 function nodes and 3628 edges. On the top of that we finally add the leakage function nodes which account for up to 1262 edges (if all leakages are exploited). Concretely, each variable node represents an intermediate value that can take $2^8$ different values. Hence, if we represent each edge by two tables in single precision of size 256, the memory required is: $256 \times (3628 \times 2 + 1262) \times 4$ bytes $\approx 8\,\text{MB}$.[3]

### 3.2 Comparison Setup

Our noise-free evaluations of ASCA and SASCA are based on single-plaintext attacks, which is due to the high memory requirements of ASCA (that hardly extend to more plaintexts). In order to stay comparable with the previous work in [32], we consider a Hamming weight ($W_H$) leakage function and specify the location of the leakages as follows:

- 16 $W_H$'s for AddRoundKey,
- 16 $W_H$'s for the output of SubBytes and ShiftRows,
- 36 $W_H$'s for the XORs and 16 $W_H$ for the look-up tables in MixColumns.

[3] For the leakage nodes, messages from variable to function ($q_{v_n \to f_m}$) are not necessary.

As previously mentioned, these leakages are represented by $\mathcal{L}$ boxes in Fig. 1. We also consider two different contexts for the information extraction:

- Consecutive weights (cw), i.e. the $W_H$'s are obtained for consecutive rounds.
- Random weights (rw), i.e. we assume the knowledge of $W_H$'s for randomly distributed intermediate values among the 804 possible ones.

Eventually, we analyzed attacks in a Known Plaintext (KP) and Unknown Plaintext (UP) scenario. And in all cases, we excluded the key scheduling leakages, as in [32]. Based on these settings, we evaluated the success rate in function of the quantity of information collected, counted in terms of "rounds of information", where one round corresponds to 84 $W_H$'s of 8-bit values.

### 3.3 Experimental Results

The results of our SASCA with noise-free leakages are reported in Fig. 2, and compared to the similar ASCA experiments provided in Reference [32].

We first observe that 2 consecutive rounds of $W_H$'s are enough to recover the key for SASCA with the knowledge of plaintext and when the leakages are located in the first rounds.[4] Next, if we do not have access to the plaintext, SASCA requires 3 consecutive rounds of leakage, as for ASCA. By contrast, and as previously underlined, the solving/decoding phase is significantly more challenging in case the leakage information is randomly distributed among the intermediate variables. This is intuitively connected to the fact that the solver and decoder both require to propagate information through the rounds, and that this information can rapidly vanish in case some intermediate variables are unknown. The simplest example is a XOR operation within MixColumns, as mentioned in introduction. So accumulating information on closely connected intermediate computations is always the best approach in such analytical attacks. This effect is of course amplified if the leakages are located in the middle rounds and the plaintext/ciphertext are unknown, as clear from Fig. 2.

Overall, and since both SAT-solvers and the BP algorithm with loops in the factor graph are highly heuristic tools, it is of course difficult to make strong statements about their respective leakage requirements. However, these experiments confirm that at least in the relevant case-study of Hamming weight AES leakages, the better noise-resilience of SASCA does not imply weaker performances in a noise-free setting. Besides, and in terms of time complexity, the attacks also differ. Namely, the resolution time for ASCA depends of the quantity of information, whereas it is independent of this quantity in SASCA, and approximately 20 times lower than the fastest resolution times for ASCA.

Note finally that moving to a noisy scenario can only be detrimental to ASCA. Indeed, and as discussed in [26], ASCA requires correct hard information for the

[4] We considered leakages for the two first rounds in this case, which seems more natural, and is the only minor differences with the experiments in [32], which considered middle rounds. However, we note that by considering middle round leakages with known plaintext, we then require three rounds of $W_H$'s, as for ASCA.

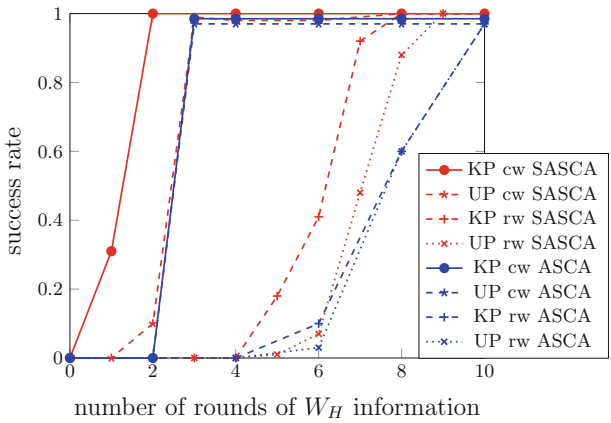


**Fig. 2.** Experimental results of comparison of ASCA and SASCA.

key recovery to succeed. In case of noisy measurements, this can only be guaranteed by considering less informative classes of leakages or similar heuristics. For example, previous works in this direction considered Hamming weights $h$'s between $h - d$ and $h + d$ for increasing distances $d$'s, which rapidly makes the attack computationally hard (and cannot be mitigated with multiple plaintext leakages because of the high RAM requirements of ASCA). So the efficiency gain of SASCA over ASCA generally increases with the measurement noise.

## 4 SASCA Against a Concrete AES Implementation

In this section, we complete the previous simulated experiments and explore whether SASCA can be transposed in the more realistic context of measured leakages. To the best of our knowledge, we describe the first uses of SASCA against a concrete AES implementation, and take advantage of this case-study to answer several questions such as (*i*) how to perform the profiling of the many target intermediate values in SASCA?, (*ii*) what happens when the implementation details (such as the source code) are unknown?, and (*iii*) are there significant differences (or even gaps) between concrete and simulated experiments?

### 4.1 Profiling Step

We first describe how to exploit the tools from Sect. 2.2 in order to detect POIs for our 1230 target intermediate values (which correspond to 1262 variable nodes minus 32 corresponding to the 16 bytes of plaintext and ciphertext). In this context, directly computing the SNRs or CPAs in parallel for all our samples turns out to be difficult. Indeed, the memory requirements to compute the mean trace of an intermediate value with simple precision requires 94,000 (samples) $\times$ 256 (values) $\times$ 4 (bytes) $\approx$ 91MB, which means approximately 100 GB for the 1,230

values. For similar reasons, computing all these SNRs or CPAs sequentially is not possible (i.e. would require too much time). So the natural option is to trade time and memory by cutting the traces in a number of pieces that fit in RAM. This is easily done if we can assume some knowledge about the implementation (which we did), resulting in a relatively easy profiling step carried out in a dozen of hours on a single desktop computer. A similar profiling could be performed without implementation knowledge, by iteratively testing the intermediate values that appear sequentially in an AES implementation.

A typical outcome of this profiling is given in Fig. 3, where we show the SNR we observed for the intermediate value $t_1$ from the factor graph in Fig. 1 (i.e. the value of the bitwise XOR of the first subkey and the first subplaintext). As intuitively expected, we can identify significant leakages at three different times. The first one, at $t = 20,779$, corresponds to the computation of the value $t_1$, i.e. the XOR between $p_1$ and $k_1$. The second one, at $t = 22,077$, corresponds to the computation of the value $s_1$, i.e. a memory access to the look-up table of the S-box. The third one, at $t = 24,004$, corresponds to memory copies of $s_1$ during the computation of MixColumns. Indeed, the SNR cannot tell apart intermediate values that are bijectively related. So we used the CPA distinguisher to get rid of this limitation (taking advantage of the fact that a simple Hamming weight leakage model was applicable against our target implementation).

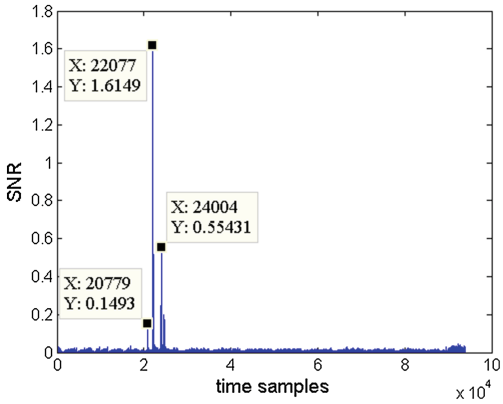


**Fig. 3.** SNR-based profiling of a single intermediate value.

A summary of the results obtained after our profiling step is given in Table 2, where the most interesting observation is that the informativeness of the leakage samples strongly depends on the target intermediate values. In particular, we see that memory accesses allow SNRs over 2, while XOR operations lead to SNRs below 0.4 (and this SNR is further reduced in case of consecutive XOR operations). This is in strong contrast, with the simulated cases (in the previous section and in [38]), where all the variables were assumed to leak with the same SNR. Note that the table mentions both SNR and CPA values, though our

**Table 2.** Summary of profiling step results.

| Assembly code | Graph description | SNR | $\rho(W_H)$ |
|---|---|---|---|
| Add round key | | | |
| `ld H1, Y+` | * | * | * |
| `eor ST11, H1` | _Xor t1 p1 k1 | 0.1493 | 0.5186 |
| Sbox | | | |
| `ldi ZH, high(sbox<<1)` | * | * | * |
| `mov ZL, ST11` | * | * | * |
| `lpm ST11, Z` | _Sbox s1 t1 | 1.6301 | 0.4766 |
| MixColumns | | | |
| `ldi ZH, high(xtime<<1)` | * | * | * |
| `mov H1, ST11` | * | * | * |
| `eor H1, ST21` | _Xor h1 s1 s2 | 0.1261 | 0.6158 |
| `eor H1, ST31` | _Xor h2 h1 s3 | 0.0391 | 0.1449 |
| `eor H1, ST41` | _Xor h3 h2 s4 | 0.3293 | 0.5261 |
| `mov H2, ST11` | * | * | * |
| `mov H3, ST11` | * | * | * |
| `eor H3, ST21` | _Xor mc1 s1 s2 | 0.2802 | 0.6163 |
| `mov ZL, H3` | * | * | * |
| `lpm H3, Z` | _Xtime xt1 mc1 | 2.8650 | 0.6199 |
| `eor ST11, H3` | _Xor cm1 xt1 s1 | 0.0723 | 0.2508 |
| `eor ST11, H1` | _Xor p17 cm1 h3 | 0.1064 | 0.3492 |
| Key schedule | | | |
| `ldi H1, 1` | * | * | * |
| `ldi ZH, high(sbox<<1)` | * | * | * |
| `mov ZL, ST24` | * | * | * |
| `lpm H3, Z` | _Sbox sk14 k14 | 2.2216 | 0.5553 |
| `eor ST11, H3` | _Xor ak1 sk14 k1 | 0.1158 | 0.5291 |
| `eor ST11, H1` | _XorCste k17 ak1 1 | 0.3435 | 0.5140 |

selection of POIs was based on the (more generic) first criteria, and CPA was only used to separate the POIs of bijectively related intermediate values.[5]

### 4.2 Experimental Results

Taking advantage of the previous POI detection, we now want to discuss the consequences of different assumptions about the implementation knowledge. These investigations are motivated by the usual gap between Kerckhoff's laws [18], which

[5] We used a relatively noisy setup on purpose (e.g. we did not filter our measurements), in order to magnify the effectiveness of SASCA in such challenging contexts.

advises to keep the key as only secret in cryptography, and the practice in embedded security, that usually takes advantage of some obscurity regarding the implementations. For this purpose, we considered three adversaries:

1. *Informed.* The adversary has access to the implementation details (i.e. source code), and can exploit the leakages of all the target intermediate values.
2. *Informed, but excluding the key scheduling.* This is the same case as the previous one, but we exclude the key scheduling leakages as in the simulations of the previous section (e.g. because round keys are precomputed).
3. *Uninformed.* Here the adversary only knows the AES is running, assumes it is implemented following the specifications in [11], and only exploits generic operations (i.e. the inputs and outputs of AddRoundKey, SubByte, ShiftRows and MixColumns, together with the key rounds' inputs and outputs).

In order to have fair comparisons, we used the same profiling for all three cases (i.e. we just excluded some POIs for cases 2 and 3), and we used 100 sets of 30 traces with different keys and plaintexts to calculate the success rate of SASCA in these different conditions. The results of our experiments are in Fig. 4. Our first and main observation is that SASCA are applicable to actual implementations, for which the leakages observed provide more or less information (and SNR) depending on the intermediate values. As expected, the informed adversary is the most powerful. But we also see that excluding the key scheduling leakages, or considering an uninformed adversary, only marginally reduces the attack success rates. Interestingly, there is a strong correlation between this success rate and the number of leakage samples exploited, since excluding the key scheduling implies the removal of 226 leakage function nodes, and the uninformed adversary has 540 leakage function nodes less than the informed one (mostly corresponding to the MixColumns operation). So we can conclude that SASCA are not only a threat for highly informed adversaries, and in fact quite generically apply to unprotected software implementations with many leaking points.

**Simulation Vs. Measurement.** In view of the previous results, with information leakages depending on the target intermediate values, a natural question is whether security against SASCA was reasonably predicted with a simulated analysis. Of course, we know that in general, analytical attacks are much harder to predict than DPA [31], and do not enjoy simple formulas for the prediction of their success rates [22]. Yet, we would like to study informally the possible connection between simple simulated analyses and concrete ones. For this purpose, we compare the results obtained in these two cases in Fig. 5. For readability, we only report results for the informed and uninformed cases, and consider different SNRs for the simulated attacks. In this context, we first recall Table 2 where the SNRs observed for our AES implementation vary between $2^1$ and $2^{-2}$. Interestingly, we see from Fig. 5 that the experimental success rate is indeed bounded by these extremes. (Tighter and more rigorous bounds are probably hard to obtain for such heuristic attacks). Besides, we also observe that the success rates of the

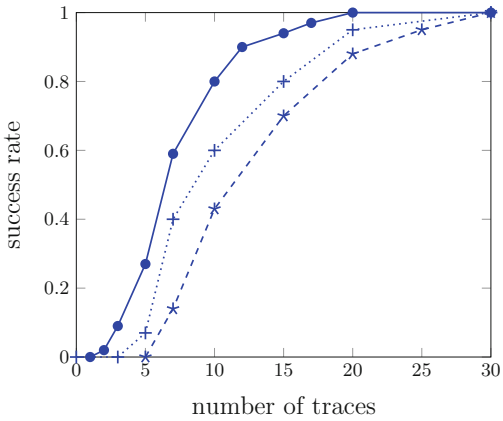


**Fig. 4.** Success rate in function of the # of traces for different adversaries: informed one (—●—), informed one without key scheduling leakages (···+···) and uninformed one (-✶-).

measurements and simulations are closer in the case of the uninformed adversary, which can be explained by the fact that we essentially ignore MixColumns leakages in this case, for which the SNRs are lower.

## 5 Comparison with DPA and Enumeration

In this section, we start from the observation that elegant approaches to side-channel analysis generally require more computational power than standard DPA. Thus, a fair comparison between both approaches should not only look at the success rate in function of the number of traces, but also take into account the resolution time as a parameter. As a result, and in order to compare SASCA and the pragmatic DPA on a sound basis, this section investigates the result of DC attacks combined with computational power for key enumeration.

### 5.1 Evaluation of Profiled Template Attacks

In order to be as comparable as possible with the previous SASCA, our comparison will be based on the profiled TA described in Sect. 2.3.[6] More precisely, we considered a quite pragmatic DC attack exploiting the bivariate leakages corresponding to the AddRoundKey and SubByte operations (i.e. $\{s_i\}_{i=1}^{16}$ and $\{t_i\}_{i=1}^{16}$ in Fig. 1). We can take advantage of the same detection of POIs as described in the previous section for this purpose. This choice allows us to keep the computational complexity of the TA itself very minimal (since relying only on 8-bit hypotheses). As previously mentioned, it also aims to make comparison

[6] We considered TA for our DPA comparison because they share the same profiled setting as SASCA. Comparisons with a non-profiled CPA can only be beneficial to SASCA. More precisely, we expect a typical loss factor of 2 to 5 between ($W_H$-based) CPA and TA, according to the results in [35] obtained on the same device.

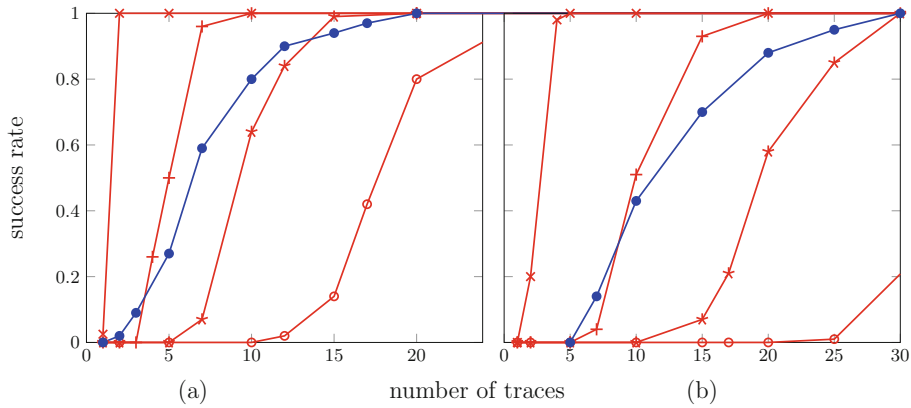


**Fig. 5.** Experimental results for SASCA for an informed adversary (a) and uninformed adversary (b). Red curves are for simulated cases (—×—, —+—, —✳—, —○—) for SNR $(2^1, 2^{-1}, 2^{-2}, 2^{-3})$. Blue curves (—●—) are for experiments on real traces (Color figure online).

as meaningful as possible (since we compare two attacks with one sample per target operation that only differ by their number of target operations). Following, we built the security graph of our bivariate TA, as represented in Fig. 6, where the white (resp. black) curve corresponds to the maximum (resp. minimum) rank observed, and the red curve is for the average rank. It indicates that approximately 60 plaintexts are required to recover the key without any enumeration (which is in line with Footnote 5). But more interestingly, the graph also highlights that allowing enumeration up to ranks (e.g.) $2^{30}$ allows to reduce the required number of measured traces down to approximately 10.

### 5.2 Comparing SASCA and DPA with Enumeration

In our prototype implementation running on a desktop computer, SASCA requires roughly one second per plaintext, and reaches a success rate of one after 20 plaintexts (for the informed adversary). In order to allow reasonably fair comparisons, we first measured that the same desktop computer can perform a bit more than $2^{20}$ AES encryptions in 20 seconds. So this is typically the amount of enumeration that we should grant the bivariate TA for comparisons with SASCA.[7] For completeness, we also considered the success rates of bivariate TA without enumeration and with $2^{30}$ enumeration power.[8] The results of these last experiments

[7] We omit to take the (time and memory) resources required for the generation of the list of the most probable keys to enumerate into account in our comparisons, since these resources remain small in the total enumeration cost. Using the state-of-the-art enumeration algorithm [36], we required 2.7MB + 0.55 seconds to generate a list of $2^{20}$ keys, and 1.8GB + 3130 seconds to generate a list of $2^{32}$ keys.

[8] Which is also more than allowed by the new suboptimal key enumeration in [3].

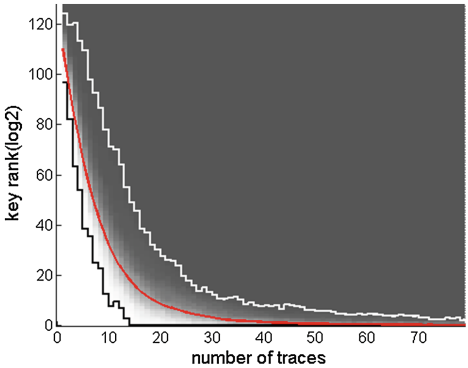


**Fig. 6.** Security graph of a bivariate TA.

are in Fig. 7. Overall, they bring an interesting counterpart to our previous investigations. On the one hand, we see that SASCA remains the most powerful attack when the adversary has enough knowledge of the implementation. By contrast in the uninformed case, the gain over the pragmatic TA with enumeration is lower. So as expected, it is really the amount and type of leakage samples exploitable by the adversary that make SASCA more or less powerful, and determine their interest (or lack thereof) compared to DC attacks. In this respect, a meaningful observation is that the gap between SASCA and DPA without enumeration (here approximately 5) is lower than the approximate factor 10 that was observed in the previous simulations of [38]. This difference is mainly due to the lower SNRs observed in the MixColumns transform.

Eventually, we note that in view of these results, another natural approach would be to use enumeration for SASCA. Unfortunately, our experiments have shown that enumeration is much less effective in the context of analytical attacks. This is essentially caused by the fact that DC attacks consider key bytes independently, whereas SASCA decode the full key at once, which implies that the subkey probabilities are not independent in this case, and can be degraded when running the loopy BP too long. Possible tracks to improve this issue include the use of list decoding algorithms for LDPC codes (as already mentioned in [13]), or enumeration algorithms that can better take subkey dependencies into account (as suggested in [19] for elliptic curve implementations).

## 6 Conclusion and Open Problems

This paper puts forward that the technicalities involved in elaborate analytical side-channel attacks, such as the recent SASCA, are possible to solve in practice. In particular, our results show that the intensive profiling of many target intermediate values within an implementation is achievable with the same (SNR &CPA) tools as any profiled attack (such as the bivariate TA we considered). This profiling only requires a dozen of hours to complete, and then enables very efficient SASCA that recover the key of our AES implementation in a couple

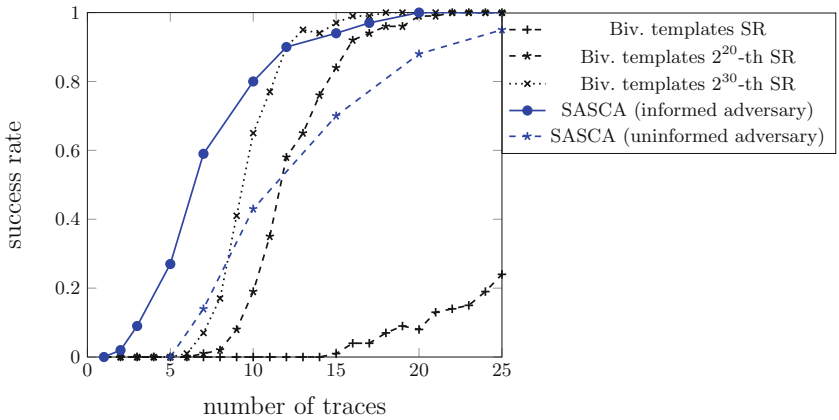


**Fig. 7.** Comparison between elegant and pragmatic approaches.

of seconds and traces, using a single desktop computer. Furthermore, these successful attacks are even possible in a context where limited knowledge about the target implementation is available, hence mitigating previous intuitions regarding analytical attacks being "only theoretical". Besides this positive conclusion, a fair comparison with DC attacks also highlights that the gap between a bivariate TA and a SASCA can be quite reduced in case enumeration power is granted to the DC adversary, and several known plaintexts are available. Intuitively, the important observation in this respect is that the advantage of SASCA really depends on the amount and type of intermediate values leaking information, which highly depends on the algorithms and implementations analyzed.

The latter observation suggests two interesting directions for further research. On the one hand, the AES Rijndael is probably among the most challenging targets for SASCA. Indeed, it includes a strong linear diffusion layer, with many XOR operations through which the information propagation is rapidly amortized. Besides, it also relies on a non-trivial key scheduling, which prevents the direct combination of information leaked from multiple rounds. So it is not impossible that the gap between SASCA and standard DPA could be larger for other ciphers (e.g. with permutation based diffusion layers [4], and very minimum key scheduling algorithms [5]). On the other hand, since the propagation of the leakage information through the MixColumns operation is hard(er), one natural solution to protect the AES against such attacks would be to enforce good countermeasures for this part of the cipher, which would guarantee that SASCA do not exploit more information than the one of a single round. Ideally, and if one can prevent any information propagation beyond the cipher rounds, we would then have a formal guarantee that SASCA is equivalent to DPA.

**Acknowledgements.** F.-X. Standaert is a research associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the European Commission through the ERC project 280141 (CRASH).

## References


1. Archambeau, C., Peeters, E., Standaert, F.-X., Quisquater, J.-J.: Template attacks in principal subspaces. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 1–14. Springer, Heidelberg (2006)
2. Banciu, V., Oswald, E.: Pragmatism vs. Elegance: comparing two approaches to simple power attacks on AES. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 29–40. Springer, Heidelberg (2014)
3. Bogdanov, A., Kizhvatov, I., Manzoor, K., Tischhauser, E., Witteman, M.: Fast and memory-efficient key recovery in side-channel attacks. IACR Cryptol. ePrint Arch. **2015**, 795 (2015)
4. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
5. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.-X., Steinberger, J., Tischhauser, E.: Key-alternating ciphers in a provable setting: encryption using a small number of public permutations. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 45–62. Springer, Heidelberg (2012)
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
7. Carlet, C., Faugère, J.-C., Goyet, C., Renault, G.: Analysis of the algebraic side channel attack. J. Crypt. Eng. **2**(1), 45–62 (2012)
8. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr, B.S., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. LNCS, vol. 2523. Springer, Heidelberg (2002)
9. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002)
10. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25–28, 2008, Philadelphia, PA, USA, pp. 293–302. IEEE Computer Society (2008)
11. Pub, FIPS 197. Advanced encryption standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
12. Gallager, R.G.: Low-density parity-check codes. IRE Trans. Inf. Theor. **8**(1), 21–28 (1962)
13. Gérard, B., Standaert, F.-X.: Unified and optimized linear collision attacks and their application in a non-profiled setting: extended version. J. Crypt. Eng. **3**(1), 45–58 (2013)
14. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008)
15. Glowacz, C., Grosso, V., Poussier, R., Schueth, J., Standaert, F.-X.: Simpler and more efficient rank estimation for side-channel security assessment. IACR Cryptol. ePrint Arch. **2014**, 920 (2014)
16. Guo, S., Zhao, X., Zhang, F., Wang, T., Shi, Z.J., Standaert, F.X., Ma, C.: Exploiting the incomplete diffusion feature: A specialized analytical side-channel attack against the AES and its application to microcontroller implementations. IEEE Trans. Inf. Forensics Secur. **9**(6), 999–1014 (2014)

17. Hanley, N., Tunstall, M., Marnane, W.P.: Unknown plaintext template attacks. In: Youm, H.Y., Yung, M. (eds.) WISA 2009. LNCS, vol. 5932, pp. 148–162. Springer, Heidelberg (2009)
18. Kerckhoffs, A.: La cryptographie militaire, ou, Des chiffres usités en temps de guerre: avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef. Librairie militaire de L, Baudoin (1883)
19. Lange, T., van Vredendaal, C., Wakker, M.: Kangaroos in side-channel attacks. In: Joye, M., Moradi, A. (eds.) CARDIS 2014. LNCS, vol. 8968, pp. 104–121. Springer, Heidelberg (2015)
20. MacKay, D.J.C.: Information Theory, Inference, and Learning Algorithms, vol. 7. Cambridge University Press, Cambridge (2003)
21. Mangard, S.: Hardware countermeasures against DPA – a statistical analysis of their effectiveness. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 222–235. Springer, Heidelberg (2004)
22. Mangard, S., Oswald, E., Standaert, F.-X.: One for all - all for one: unifying standard differential power analysis attacks. IET Inf. Secur. **5**(2), 100–110 (2011)
23. Mather, L., Oswald, E., Whitnall, C: Multi-target DPA attacks: pushing DPA beyond the limits of a desktop computer. In: Sarkar and Iwata [33], pp. 243–261
24. Mohamed, M.S.E., Bulygin, S., Zohner, M., Heuser, A., Walter, M., Buchmann, J.: Improved algebraic side-channel attack on AES. J. Crypt. Eng. **3**(3), 139–156 (2013)
25. Oren, Y., Kirschbaum, M., Popp, T., Wool, A.: Algebraic side-channel analysis in the presence of errors. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 428–442. Springer, Heidelberg (2010)
26. Oren, Y., Renauld, M., Standaert, F.-X., Wool, A.: Algebraic side-channel attacks beyond the hamming weight leakage model. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 140–154. Springer, Heidelberg (2012)
27. Oren, Y., Weisse, O., Wool, A.: A new framework for constraint-based probabilistic template side channel attacks. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 17–34. Springer, Heidelberg (2014)
28. Pearl, J.: Reverend Bayes on inference engines: a distributed hierarchical approach. In: Waltz, D.L. (ed) Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, August 18–20, 1982, pp. 133–136. AAAI Press (1982)
29. Rebeiro, C., Selvakumar, D., Devi, A.S.L.: Bitslice implementation of AES. In: Pointcheval, D., Mu, Y., Chen, K. (eds.) CANS 2006. LNCS, vol. 4301, pp. 203–212. Springer, Heidelberg (2006)
30. Renauld, M., Standaert, F.-X.: Algebraic side-channel attacks. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 393–410. Springer, Heidelberg (2010)
31. Bauer, A., Coron, J.-S., Naccache, D., Tibouchi, M., Vergnaud, D.: On the broadcast and validity-checking security of PKCS#1 v1.5 encryption. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 1–18. Springer, Heidelberg (2010)
32. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N.: Algebraic side-channel attacks on the AES: why time also matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (2009)
33. Sarkar, P., Iwata, T. (eds.): Advances in Cryptology - ASIACRYPT 2014. LNCS, vol. 8873. Springer, Berlin Heidelberg (2014)
34. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)

35. Standaert, F.-X., Gierlichs, B., Verbauwhede, I.: Partition *vs.* Comparison side-channel distinguishers: an empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 253–267. Springer, Heidelberg (2009)
36. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.-X.: An optimal key enumeration algorithm and its application to side-channel attacks. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 390–406. Springer, Heidelberg (2013)
37. Veyrat-Charvillon, N., Gérard, B., Standaert, F.-X.: Security evaluations beyond computing power. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 126–141. Springer, Heidelberg (2013)
38. Veyrat-Charvillon, N., Gérard, B., Standaert, F.-X.: Soft analytical side-channel attacks. In: Sarkar and Iwata [33], pp. 282–296
39. Zhao, X., Zhang, F., Guo, S., Wang, T., Shi, Z., Liu, H., Ji, K.: MDASCA: an enhanced algebraic side-channel attack for error tolerance and new leakage model exploitation. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 231–248. Springer, Heidelberg (2012)