Soft Analytical Side-Channel Attacks

Nicolas Veyrat-Charvillon[1], Benoît Gérard[2], and François-Xavier Standaert[3]

[1] IRISA-CAIRN, Campus ENSSAT, 22305 Lannion, France
[2] DGA Maîtrise de l'Information, 35998 Rennes, France
[3] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium

Abstract. In this paper, we introduce a new approach to side-channel key recovery, that combines the low time/memory complexity and noise tolerance of standard (divide and conquer) differential power analysis with the optimal data complexity of algebraic side-channel attacks. Our fundamental contribution for this purpose is to change the way of expressing the problem, from the system of equations used in algebraic attacks to a code, essentially inspired by low density parity check codes. We then show that such codes can be efficiently decoded, taking advantage of the sparsity of the information corresponding to intermediate variables in actual leakage traces. The resulting soft analytical side-channel attacks work under the same profiling assumptions as template attacks, and directly exploit the vectors of probabilities produced by these attacks. As a result, we bridge the gap between popular side-channel distinguishers based on simple statistical tests and previous approaches to analytical side-channel attacks that could only exploit hard information so far.

1 Introduction

The great majority of side-channel attacks published in the literature follow a divide and conquer strategy (DC). That is, they first attack independent parts of the key separately (divide), and then combine these pieces of information (conquer). Information on individual parts of the key is obtained by studying correlations between key-dependent leakage predictions and the actual side-channel measurements. The information can then be combined either by simply concatenating the most probable values of each key part together, or by using an enumeration algorithm [27,28]. Examples of distinguishers exploiting such a strategy include Kocher et al.'s Differential Power Analysis (DPA) [13], Brier et al.'s Correlation Power Analysis (CPA) [2], Gierlichs et al.'s Mutual Information Analysis (MIA) [9], Chari et al.'s Template Attacks (TA) [4] and Schindler et al.'s Stochastic Approach (SA) [23]. The popularity of these tools is due to their simplicity and versatility: they can be adapted to essentially any implementation, have low time complexity and work in a gray box manner. That is, they do not require a precise understanding of the underlying hardware, but their data complexity is highly dependent on the quality of the adversary's leakage predictions. Therefore, the knowledge of implementation details and some engineering intuition can usually be exploited to improve their time and data complexity.

P. Sarkar and T. Iwata (Eds.): ASIACRYPT 2014, PART I, LNCS 8873, pp. 282–296, 2014.

In this context, one fundamental question regarding DC distinguishers is whether they are sufficient for security evaluations. That is, are the security levels estimated with such tools close enough to the worst-case? In view of the previously listed qualities (and in particular, their excellent time complexity), the most likely drawback candidate for DC strategies is a suboptimal data complexity. As a result, a number of research works have investigated whether the application of analytical strategies (i.e. targeting the full key at once) could provide improved results. To the best of our knowledge, one of the first attempts in this direction was Mangard's Simple Power Analysis (SPA) against the AES key expansion algorithm [15]. The Side-Channel Collision Attacks (SCCA) in [1,24,25] were next interesting steps, in which the key is recovered by solving a set of (mostly) linear equations corresponding to the first cipher round(s). Following, Algebraic Side-Channel Attacks (ASCA) were introduced in [21,22] and probably constitute the most representative example of analytical strategy to date. Under certain conditions, they are able to extract the key of an AES implementation from a single leakage trace, in an unknown plaintext/ciphertext scenario.

So to some extent, ASCA could be viewed as an extreme opposite to DC attacks, with a minimum data complexity coming at the cost of a (much) more complex and sensitive solving phase – hence raising questions regarding their practical relevance. For example in the first papers from Renauld et al., the adversary represents the target block cipher and its leakages as an instance of satisfiability problem that she sends to a generic SAT solver (other types of solvers, e.g. based on Gröbner bases, have also been analyzed [3]). The main issue with this approach is a very weak resistance to noise, since the solver essentially needs to be fed with correct hard information. For this purpose, the usual strategy was to group certain leakage values according to a model with lower cardinality, e.g. the well-known Hamming weight one, in order to trade robustness for informativeness. Improved heuristics are presented in [17,29]. More recently, Oren et al. proposed to replace the use of a solver by that of an optimizer, leading to Tolerant ASCA (TASCA) able to exploit more general models [18,19]. Yet, even these last attempts were quite inefficient in exploiting soft information, mainly because of the difficulty to translate a vector of probabilities (e.g. as provided by classical TA) into an optimizer-friendly format. In fact TASCA essentially encode these vectors as exhaustive hard information, hence limiting the number of leaking operations that could be included in the optimizer to a couple of rounds (compared to the full cipher in ASCA), because of memory issues. Eventually, the results in [10] provide yet another powerful approach to analytical side-channel attacks, based smart enumeration and specialized to the AES, but so far they also remain limited to the exploitation of hard information.

This state-of-the-art seems to suggest that the probabilistic information provided by side-channel leakages can be easily exploited with DC attacks, while analytical strategies require a preprocessing step to translate this soft information into hard one. In this paper, we argue that this intuition is flawed, and in fact relates to the way of formulating the problem rather than to its nature. That is, while previous analytical attacks were expressing the target block

ciphers and their leakages as equations, we propose to describe the same problem as a code. As a result, and for the first time, we detail a Soft Analytical Side-Channel Attack (SASCA) that combines the best of two worlds, namely the noise robustness and low time complexity of DC strategies with the low data complexity of analytical ones. In this respect, our first contribution is to exhibit a natural way to encode a side-channel cryptanalysis problem. Next, we show that we can efficiently decode such problems thanks to the Belief Propagation algorithm (BP). Using these new tools, we are able (*i*) for low noise levels: to attack the AES FURIOUS implementation that was targeted in previous works on ASCA/TASCA with a single leakage trace, with significantly reduced time and memory complexities, (*ii*) for large noise levels: to attack the same implementation with multiple plaintexts, but with 2^3 to 2^4 less traces than a standard TA. Summarizing, the proposed technique bridges the gap between DPA and ASCA.

Related Works. While the motivation for SASCA quite directly derives from previous works in ASCA/TASCA, its mathematical modeling fundamentally differs from them and is in fact much closer to some results exploiting techniques from coding theory. In particular, the application of Hidden Markov Models in the context of time-randomized implementations [5,12] or side-channel disassemblers [6], and the decoding of Low Density Parity Check (LDPC) codes in the context of SCCA [8] were sources of inspiration for the following work.

Cautionary Note. In order to show the applicability of SASCA at different noise levels, our empirical results are based on simulated experiments. Yet, we insist that SASCA is (in general) just as realistic as any TA, since it relies on the same assumptions for the profiling phase (i.e. the knowledge of a single key – see Appendix A). Furthermore, we paid attention to exploit exhaustive templates (i.e. used 256 profiles per intermediate value attacked) which can be generalized to any leakage function and corresponds to the worst-case time complexity.

2 Soft Analytical Side-Channel Attacks

We first emphasize the differences between previous solver- or optimizer-based approaches to analytical side-channel attacks and our decoder-based solution. We then describe the BP algorithm and discuss its connection to the exploitation of side-channel leakage. We finally detail how to describe an AES implementation as a factor graph, that can be efficiently decoded by BP. The following descriptions assume a profiled attack scenario, as usual in worst-case evaluations [26].

2.1 Solving (or Optimizing) vs. Decoding

In the course of a profiled side-channel attack, the adversary extracts information from leakage traces. This information comes from the processing of intermediate values throughout the cryptographic computations. By comparing these leakages with previously estimated templates, she obtains for each target value X_i a conditional posterior distribution $\Pr[X_i|L]$. Provided the device is not perfectly

side-channel resistant, most of the posterior distributions will have an entropy lower than that of a uniform distribution. In this context, the most interesting pieces of information relate to the encryption key. For this purpose, not only the leakages that directly correspond to key bytes – informally denoted as SPA leakages – are exploited, but also those of intermediate variables that depend on both the key and the (usually known) plaintext – informally denoted as DPA leakages – such as the SBOX outputs in the first AES round, typically. For example, starting from the posterior probability of the output value S_{out} given the leakage L_{out}, one can deduce its image before the substitution layer:

$$\Pr[S_{in} = v|L_{out}] = \Pr[S_{out} = \text{SBOX}(v)|L_{out}].$$

For a known plaintext value P, one can compute a posterior distribution on a key byte K by unrolling the computation one step further:

$$\Pr[K = k|P = p, L_{out}] = \Pr[S_{out} = \text{SBOX}(k \oplus p)|L_{out}].$$

These simple equations show that it is possible to derive information about the key using intermediate variables. Furthermore, one can easily combine the leakage obtained from multiple plaintexts, by marginalizing $\Pr(K = k)$ over the corresponding traces: this is in fact what DC attacks do. Next, and since multiple key-dependent variables can usually be found within cryptographic implementations, a natural problem is to find ways to exploit them efficiently. But this is exactly where the DC strategy faces limitations. Namely, combining the leakage of these intermediate variables is trivial as long as they *only* depend on a single key byte, e.g. the SBOX inputs and outputs in a first block cipher round. One just deals with the additional variable as with an additional plaintext in this case. Taking the example of AES, this can even be extended to the first MIXCOLUMNS operation, if 32-bit key hypotheses are performed by the adversary. But the DC approach is inherently limited to the exploitation of predictable parts of the key. So as soon as the diffusion is complete (which very rapidly occurs in modern ciphers and therefore corresponds to most of their intermediate computations), the leakages are left unexploited by such strategies. This limitation directly leads to the main problem we tackle in this paper, namely: *How to efficiently exploit the leakage of any intermediate variable in a side-channel attack?*

Previous ASCA were a first attempt to answer this question, by trying to solve a system of equations describing the target cryptographic algorithm, complemented with the information extracted. These attacks typically begin by sieving intermediate values, keeping only the most probable ones. A usual approach is to coalesce the leakages by Hamming weights for this purpose. The set of remaining values is then verified one against another (e.g. using heuristic SAT-solvers). Unfortunately, this algebraic approach cannot easily deal with the probability distributions output by TA, which are thus discretized and sieved. Whenever the measurement noise is not negligible, this introduces "errors" that are fatal to algebraic solvers. As mentioned in introduction, optimizers allow mitigating this problem, but are still limited in the exhaustive way they encode the probabilities (which is too expensive for describing more than a couple of AES rounds).

Our method works differently, by operating directly on the posterior distributions of the intermediate values extracted from leakage traces, and propagating the information throughout the computation steps of the algorithm. When attacking a cryptographic implementation, we first build a large graphical model containing the intermediate variables, which are linked by constraints corresponding to the atomic operations executed. For instance, the exclusive-OR and SBOX functions are usually found in software implementations of the AES. Next, the goal is to find the marginal distribution of the key, given the distributions of all the intermediate variables. While this is generally a hard problem, we observe that an important feature of cryptographic algorithms is that intermediate values tend to appear only in a few places. A similar behavior is present in Gallager codes [7], also called Low-Density Parity Check codes (LDPC). In such a code, codeword bits are linked together by a small number of parity constraints (i.e. linear in the codeword size). Decoding such a construction is generally performed via application of the BP algorithm, also known as sum-product algorithm. Our application in the following sections is a (conceptually) simple extension, where values are not limited to bits, and parity constraints go beyond exclusive-ORs.

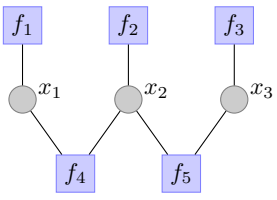
2.2 The Belief-Propagation Algorithm

Our description of the BP algorithm is largely based on the (excellent) description provided in [14, chapter 26]. Let us consider a set of N variables $\mathbf{x} \equiv \{x_n\}_{n=1}^{N}$, and define a function P^* of $\mathbf{x}$ which is a product of M *factors*:

$$P^*(\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_m),$$

where each factor $f_m(\mathbf{x}_m)$ is a function of a subset $\mathbf{x}_m$ of $\mathbf{x}$. The P^* function is typically depicted using a *factor graph*, in which circles correspond to variables x_i and squares to functions f_m. An edge is drawn between x_i and f_m if $x_i \in \mathbf{x}_m$, meaning that the m-th factor depends on the i-th variable. For example, the parity functions and factor graph of a simple 3-repetition code are shown below:

$$\begin{aligned} f_1(x_1) &= \Pr(x_1 = 1) \\ f_2(x_2) &= \Pr(x_2 = 1) \\ f_3(x_3) &= \Pr(x_3 = 1) \\ f_4(x_1, x_2) &= \begin{cases} 1 & \text{if } x_1 \oplus x_2 = 0 \\ 0 & \text{otherwise} \end{cases} \\ f_5(x_2, x_3) &= \begin{cases} 1 & \text{if } x_2 \oplus x_3 = 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$



The task we are interested in is that of *marginalization*. That is, we aim to be able to compute the following function:

$$Z_n(x_n) = \sum_{\mathbf{x}, \mathbf{x}_n = x_n} P^*(\mathbf{x}),$$

and more importantly it normalized version $P_n(x_n) = Z_n(x_n)/Z$, where:

$$Z = \sum_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x}_m).$$

These tasks are intractable in general. Even when the factor functions are limited to three variables, the cost of computing the exact marginal is believed to grow exponentially with the number of variables N. The BP algorithm can circumvent this problem and compute marginals efficiently as long as the factor graph is tree-like. We will denote by $\mathcal{N}(m)$ the set of variables involved in factor f_m, by $\mathcal{M}(n)$ the set of factors where variable x_n appears, and shorthand the set of variables in $\mathbf{x}_m$ with x_n excluded as: $\mathbf{x}_m \setminus n \equiv \{x_{n'} : n' \in \mathcal{N}(m) \setminus n\}$. The algorithm works by passing two types of messages along the edges of the factor graph, from variables to factors ($q_{n\to m}$) and from factors to variables ($r_{m\to n}$). The sets of messages are updated using two rules:

$$q_{n\to m}(x_n) = \prod_{m' \in \mathcal{M}(n)\setminus m} r_{m'\to n}(x_n).$$

$$r_{m\to n}(x_n) = \sum_{\mathbf{x}_m \setminus n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)\setminus n} q_{n'\to m}(x'_n) \right).$$

Convergence should occur after a finite number of iterations, at most equal to the longest path. Once the network has converged, the marginal function (also called *belief*) of a variable x_n can be recovered by multiplying together all incoming messages at the corresponding node:

$$Z_n(x_n) = \prod_{m \in \mathcal{M}(n)} r_{m\to n}(x_n).$$

The normalized value $P_n(x_n) = Z_n(x_n)/Z$ is easily obtained by summing together the marginal functions $Z = \sum_{x_n} Z_n(x_n)$. As already mentioned, the BP algorithm returns the exact marginals as long as the factor graph is a tree-shaped graphical model. Yet, in many useful cases such as decoding, the graph contains cycles. Fortunately, BP can be applied directly on general factor graphs as well, raising the so-called "loopy" BP. While this version does not guarantee to return the correct marginals, and may even not converge to a fixed point in some cases, it usually gives results that are good enough for most applications.

2.3 Efficient Representation of an AES Implementation

Our method for SASCA consists in an application of the BP algorithm to the decoding of keys using plaintexts, ciphertexts and side-channel traces. In this section, we illustrate it in the context of an implementation of the AES in an 8-bit device. For this purpose, the x_i variables defined in the description of the BP algorithm will represent the intermediate values handled by the cryptographic algorithm, and the parity functions will be separated into two sets:

– The first set corresponds to the *a priori* knowledge on the variables acquired through side-channel leakages, denoted as $f_i(x_i) = \Pr[x_i = v|L]$.
– The second set corresponds to the operations executed by the implementation. In the case of a binary operation $\text{OP}(x_{i_1}, x_{i_2})$, the function is defined by:

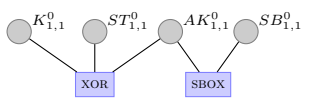
$$f_i(x_{i_1}, x_{i_2}, x_{i_3}) = \begin{cases} 1 & \text{if } \text{OP}(x_{i_1}, x_{i_2}) = x_{i_3}, \\ 0 & \text{otherwise.} \end{cases}$$

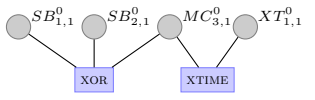
Based on these notations, an adversary first has to encode the AES computations in a form that is compatible with the BP algorithm. For illustration, and because it is publically available, we will describe how to build a factor graph for the AES FURIOUS implementation (`http://point-at-infinity.org/avraes`).

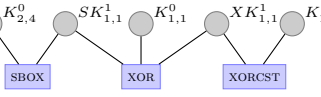
Concretely, our program takes in a description which is very similar to the assembly code of AES FURIOUS, with the memory related operations left out, but where any assignment requires a newly named variable. Namely, variable nodes are denoted by names starting with a capital letter, such as `K[2,4]_0` for the intermediate key in row 2 and column 4 of key scheduling round 0, which also happens to be the second master key byte (noted $K^0_{2,4}$ in the factor graph), or `SB[2,1]_0` for the SBOX output in row 2 and column 1 of round 0 ($SB^0_{2,1}$ in the factor graph). These variable nodes correspond to intermediate values computed during encryption, such as the state (`ST`), key addition or MIXCOLUMNS intermediate results (`AK` and `MC`), outputs of XTIME operations (`XT`), ... Besides, factor node names start with an underscore such as `_Xor` (exclusive OR) and `_Xtime` (polynomial multiplication by x). They correspond to instructions executed during encryption. For example, Table 1 gives samples of the correspondence between the assembly code, input description and factor graph. Note that the factor nodes for the prior probabilities of the variables are not drawn.

Table 1. Factor graph representation of an AES encryption

Assembly code	Graph description	Factor graph

`ld H1, Y+`
`eor ST11, H1`
`mov ZL, ST11`
`lpm ST11, Z`
	`*`
	`_Xor AK[1,1]_0 ST[1,1]_0 K[1,1]_0`
	`*`
	`_Sbox SB[1,1]_0 AK[1,1]_0`
		

`mov H3, ST11`
`eor H3, ST21`
`mov ZL, H3`
`lpm H3, Z`
	`*`
	`_Xor MC[3,1]_0 SB[1,1]_0 SB[2,1]_0`
	`*`
	`_Xtime XT[1,1]_0 MC[3,1]_0`
		

`mov ZL, ST24`
`lpm H3, Z`
`eor ST11, H3`
`eor ST11, H1`
	`*`
	`_Sbox SK[1,1]_1 K[2,4]_0`
	`_Xor XK[1,1]_1 SK[1,1]_1 K[1,1]_0`
	`_XorCst K[1,1]_1 XK[1,1]_1 0x1`
		

There are two notable differences between SASCA and the classical decoding of LDPC codes. First, variable nodes are not binary digits, but rather elements of

$\mathbf{GF}(2^8)$. Second, factor nodes are not limited to exclusive OR's, but may include any of the variety of functions used in cryptographic implementations (e.g. XOR, SBOX, XTIME). However, these factor nodes are not much more complex than for classical decoding, as illustrated with our three previous examples:

$$\text{XOR}(A, B, C) = \begin{cases} 1 & \text{if } A \oplus B = C, \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{SBOX}(A, B) = \begin{cases} 1 & \text{if } A = S(B), \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{XTIME}(A, B) = \begin{cases} 1 & \text{if } A = Xt(B), \\ 0 & \text{otherwise.} \end{cases}$$

This natural representation of operations is very efficient, as opposed to the contrived way AES encryptions are translated to SAT instances (roughly, it corresponds to 1,200 equations and variables in $\mathbf{GF}(2^8)$ compared to 18,000 equations in 10,000 variables in SAT-based ASCA). Taking advantage of it, the SASCA adversary then tries to compute the key marginal probability for $P_n(K)$ given the leakages. For this purpose, one simply has to incorporate the implicit factor nodes corresponding to prior knowledge on variable nodes, as given by the templates of the side-channel attack. For instance, the factor for the output of the first SBOX in the first round $f_m(SB^0_{1,1})$ is the posterior distribution $Pr[SB^0_{1,1}|L]$. In addition, any known value (for instance the plaintext bytes) has a prior knowledge with entropy zero, and any value that does not leak (either because it is protected or precomputed) has a uniform prior. Eventually, the loopy BP algorithm propagates information throughout the factor graph: if successful (i.e. in case of convergence), it should return the approximate marginal probabilities of the key bytes $P_n(K^0_{1,1})$ to $P_n(K^0_{4,4})$, i.e. the answer we are looking for.

2.4 Attacking with Several Traces

The ability to efficiently exploit (i.e. combine the information of) several leakage traces is one of the reasons that have made DPA attacks so popular – since it typically leads to the noise vs. data complexity tradeoff that is at the core of most side-channel attacks. It also remains one of the main practical issue for ASCA and follow-up works. So far, the only way several traces can be useful is when they are repetitions of the same encryption (without randomizations), so that the noise can be averaged out. By contrast, adding traces corresponding to multiple plaintexts could only be managed with the construction of larger systems, that are too memory consuming for TASCA, and increasing the probability that one piece of hard information in such systems is incorrect for ASCA.

Interestingly, SASCA are able to improve the key recovery success rate with each additional trace observed. Practically, the factor graph used for decoding is first replicated for each trace. Yet, since the master key stays the same during the course of the attack, the part of our factor graph corresponding the key

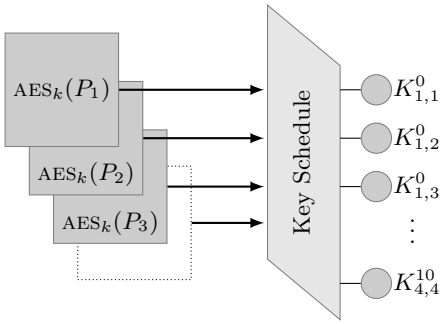


Fig. 1. Factor graph connections for several traces

scheduling also remains constant: it forms a kind of "backbone" where all the encryption rounds connect, as depicted in Figure 1. As a result, whenever several messages are used, the probability distributions are propagated from each replicated graph towards the key schedule. The impact of such propagation is in fact very similar to the one resulting from using several traces in a classical TA, where probabilities are multiplied together and the success rate increases.

3 Experimental Results

We now validate the method described in the previous section with illustrative simulated attacks against the AES FURIOUS implementation. For this purpose, we assume a setup that is essentially similar to the one used to demonstrate the applicability of ASCA to the AES in [22]. The only difference is that we will consider implementations with and without the key scheduling leakages. As previously explained (and illustrated in Table 1), all the operations found in the assembly code are translated into factor nodes, excluding memory related operations. For illustration, we considered Hamming weight leakages affected by a noise of variance σ_n^2, but the attack is independent of this choice: any function could be incorporated without performance penalty. The only important parameter in our case is the informativeness of the leakages which, in the first-order setting we investigate, can be measured with a Signal-to-Noise Ratio (SNR) [16]. Since the signal (i.e. variance) of a Hamming weight leakage function for 8-bit intermediate values equals 2, one can simply derive the SNR as $2/\sigma_n^2$. For illustration, we compared our results with the ones of two standard TA. Namely, one univariate exploiting only the first-round S-box output leakages, and one bivariate exploiting the first-round S-box input and output leakages.

The results of our experiments are shown in figure 2. The x-axis corresponds to the number of messages used for the attack (in log scale), and the y-axis is a stack of success rate curves for decreasing SNRs (i.e. increasing noise levels). An alternative view is provided in Figure 3, which sums up these simulation results by showing the data complexity gains of SASCA over TA. It appears from both figures that these gains are significant and consistently observed for

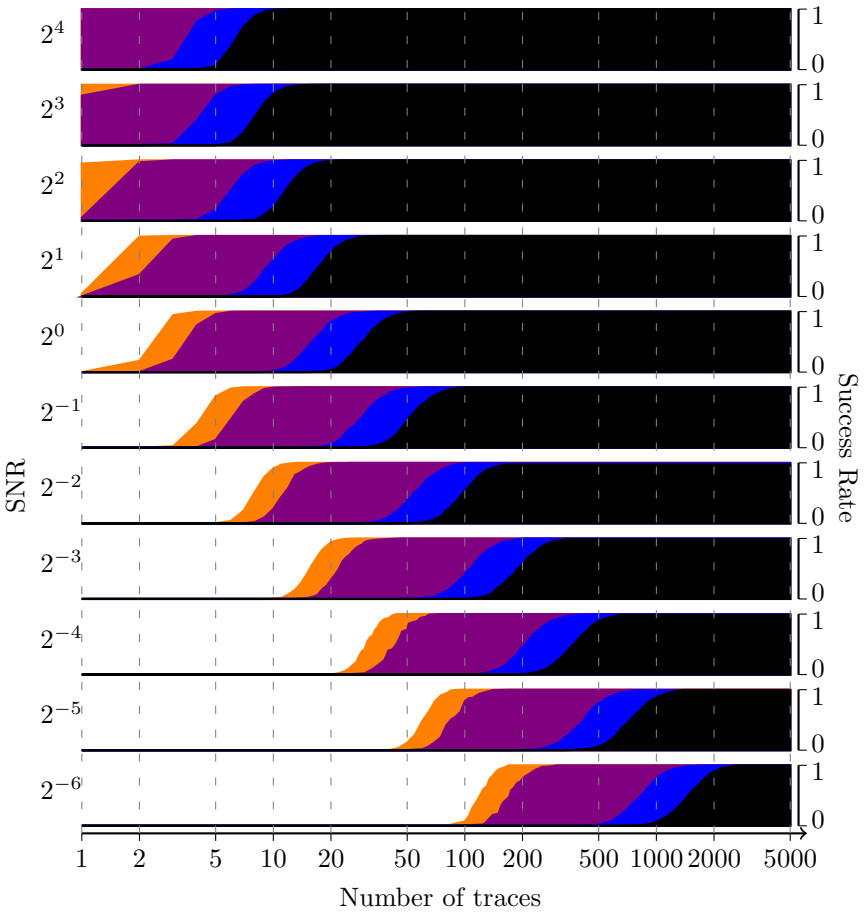


Fig. 2. Attacks results for our simulated FURIOUS implementation. Each graph gives the success rate (SR, ranging from 0 to 1) for a given signal-to-noise ratio (SNR, ranging from 2^4 down to 2^{-6}) as a function of the number of traces (in logarithmic scale, ranging from 1 to 5000). The attacks are:

- univariate TA targetting the SBOX output (in dark gray),
- bivariate TA targetting the SBOX input and output (in blue),
- SASCA attack ignoring the key schedule leakages (in violet),
- SASCA attack exploiting all the intermediate values (in orange).

any noise level. Eventually, the unknown inputs and outputs scenario is detailed for SASCA in Figure 4. We see that its impact is limited if the key scheduling leaks (confirming the results from [15]) and more significant otherwise.

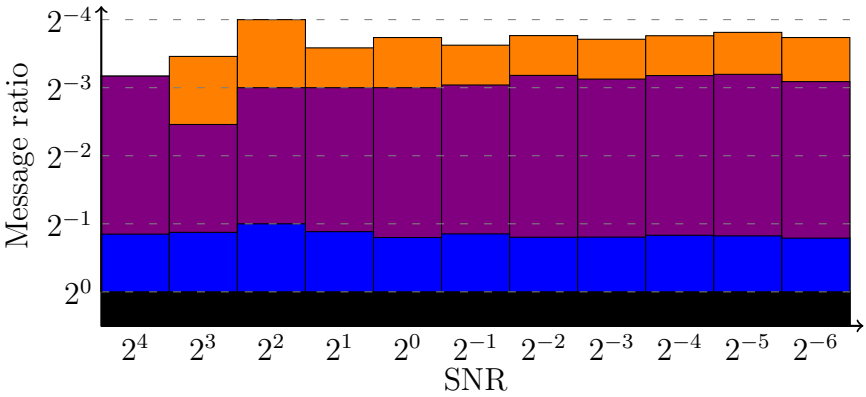


Fig. 3. Data complexity gain of SASCA compared to TA given as the fraction of measurements needed to reach a success rate of 0.9 (same colors as in Figure 2)

Discussion. Compared to previous results in ASCA/TASCA, our new tools bring two main advantages. First, from the SNR point of view, these works were typically limited to scenarios where a single leakage trace was enough to recover the master key (i.e. to SNRs $> 2^2$). We can deal with any SNR. Second, the time and memory complexity of the BP decoding is much improved compared to SAT-solver based ASCA and optimizer-based TASCA. Our implementation deals with a factor graph of size proportional to the number of messages, with a relatively high (yet easily tractable in practice) constant of approximately 16M per message. Its computation time is proportional to both the diameter of the graph (constant after the second message) which sets the number of decoding iterations, and the number of measurements which sets the amount of messages exchanged at each iteration. This makes the evolution of the time and memory complexity of SASCA quite comparable to the one in divide and conquer TA (i.e. linear in the number of messages). Yet, decoding the AES encryption factor graph with the BP algorithm implies a larger computation time of approximately one second per message in our prototype implementation, running on an Intel i7-2720QM. This (constant) overhead is the main penalty to enjoy the substantially smaller data complexities of SASCA (i.e. similar to ASCA/TASCA) which is, as expected, the main advantage of analytical strategies over DC ones.

As detailed in Appendix A, the practical relevance of such attacks is quite similar to TA, since it requires the same profiling assumptions (i.e. the knowledge of a single key). Admittedly, the profiling effort is significantly more expensive for SASCA, since it requires characterizing all the target intermediate values. But since all these target values can be profiled independently, building their templates can be done quite efficiently (with essentially the same amount of measurements as needed to characterize the first-round operations exploited in TA), and is easily automated with standard side-channel attack techniques.

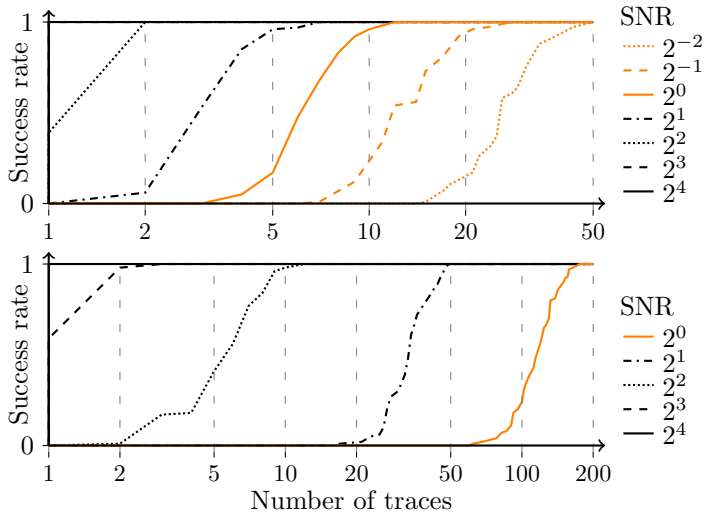


Fig. 4. SASCA with unknown input and output for different SNRs.The x-axis is the number of traces used for the attack (in log scale), and the y-axis gives the probability of key recovery. The top graph corresponds to a leaky key schedule, and the lower graph gives the results for a leak-free key schedule.

4 Conclusions

By modeling the side-channel analysis problem adequately, SASCA bring the missing link between standard DC distinguishers and analytical strategies for key recoveries. As a result and for the first time, we are able to efficiently exploit the probabilistic information of all the leaking operations in a software implementation. Our resulting attacks are optimal in data complexity and efficient in time and memory. Yet, we note that the tools exploited in this first instantiation of SASCA can certainly be improved. For example, the BP algorithm performs too many computations for our needs. Indeed, it propagates every distribution throughout the factor graph whereas in practice, we are mostly interested in the key. Hence, further works could exploit the propagation of messages only towards the schedule (i.e. perform Bayesian inference). This would additionally allow the attack to be performed one message at a time, by accumulating information retrieved from each trace onto the nodes of the key schedule, hence reducing the memory requirements to that of a single trace (i.e. 16M).

In view of the improved noise robustness of SASCA, an important open problem is to determine whether the strong results obtained with this new type of analytical strategy also apply to implementations protected with countermeasures. Masking, shuffling and leakage-resilient cryptography appear as the most interesting targets in this respect. Besides, the experiments in this work considered a worst-case scenario where the adversary could take advantage of all the leaking operations of an AES implementation (i.e. assuming the knowledge of

the source code, essentially). But the investigation of an intermediate scenario where the adversary would exploit less leaking observations (e.g. the ones he could guess without knowing the source code) and its resulting time and data complexity is another interesting scope for additional investigations.

Acknowledgements. François-Xavier 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 (acronym CRASH) and by the PAVOIS project (ANR 12 BS02 002 01).

References

1. Bogdanov, A., Kizhvatov, I., Pyshkin, A.: Algebraic methods in side-channel collision attacks and practical collision detection. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 251–265. Springer, Heidelberg (2008)
2. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, Quisquater (eds.) [11], pp. 16–29
3. Carlet, C., Faugère, J.-C., Goyet, C., Renault, G.: Analysis of the algebraic side channel attack. Journal of Cryptographic Engineering 2(1), 45–62 (2012)
4. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
5. Durvaux, F., Renauld, M., Standaert, F.-X., van Oldeneel tot Oldenzeel, L., Veyrat-Charvillon, N.: Efficient removal of random delays from embedded software implementations using hidden markov models. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 123–140. Springer, Heidelberg (2013)
6. Eisenbarth, T., Paar, C., Weghenkel, B.: Building a side channel based disassembler. Transactions on Computational Science 10, 78–99 (2010)
7. Gallager, R.G.: Low-density parity-check codes. IRE Transactions on Information Theory 8(1), 21–28 (1962)
8. Gérard, B., Standaert, F.-X.: Unified and optimized linear collision attacks and their application in a non-profiled setting. In: Prouff, Schaumont (eds.) [20], pp. 175–192
9. 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)
10. 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 Transactions on Information Forensics and Security 9(6), 999–1014 (2014)
11. Joye, M., Quisquater, J.-J. (eds.): CHES 2004. LNCS, vol. 3156. Springer, Heidelberg (2004)
12. Karlof, C., Wagner, D.: Hidden markov model cryptanalysis. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 17–34. Springer, Heidelberg (2003)
13. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

14. MacKay, D.J.C.: Information theory, inference, and learning algorithms. Cambridge University Press (2003)
15. Mangard, S.: A simple power-analysis (SPA) attackon implementations of the AES key expansion. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 343–358. Springer, Heidelberg (2003)
16. Mangard, S., Oswald, E., Standaert, F.-X.: One for all - all for one: unifying standard differential power analysis attacks. IET Information Security 5(2), 100–110 (2011)
17. Mohamed, M.S.E., Bulygin, S., Zohner, M., Heuser, A., Walter, M., Buchmann, J.: Improved algebraic side-channel attack on AES. Journal of Cryptographic Engineering 3(3), 139–156 (2013)
18. 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)
19. Oren, Y., Renauld, M., Standaert, F.-X., Wool, A.: Algebraic side-channel attacks beyond the hamming weight leakage model. In: Prouff, Schaumont (eds.) [20], pp. 140–154
20. Prouff, E., Schaumont, P. (eds.): CHES 2012. LNCS, vol. 7428. Springer, Heidelberg (2012)
21. 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)
22. 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)
23. 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)
24. Schramm, K., Leander, G., Felke, P., Paar, C.: A collision-attack on AES: Combining side channel- and differential-attack. In: Joye, Quisquater (eds.) [11], pp. 163–175
25. Schramm, K., Wollinger, T., Paar, C.: A new class of collision attacks and its application to DES. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 206–222. Springer, Heidelberg (2003)
26. Standaert, F.-X., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
27. 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)
28. 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)
29. Zhang, F., Zhao, X., Guo, S., Wang, T., Shi, Z.: Improved algebraic fault analysis: A case study on piccolo and applications to other lightweight block ciphers. In: Prouff, E. (ed.) COSADE 2013. LNCS, vol. 7864, pp. 62–79. Springer, Heidelberg (2013)

A Attack Requirements

In this section, we provide a brief discussion of the profiling step that precedes the application of SASCA. In particular, we argue that the profiling overhead and required knowledge for this purpose are similar to those of standard TA.

Profiling Overhead. Similarly to classical TA, SASCA require profiling the leakage corresponding to their target intermediate values. In this respect, the only difference is that they can take advantage of many such values, whereas DC strategies only exploit the first round(s) leakages. In general, one can assume that all target intermediate values leak a similar amount of information. And if it is not the case, it is usually the first round(s) leakages that have lower SNRs. As a result, and given that the set of profiling traces corresponds to random inputs, one can essentially build all the SASCA templates with the same traces as for a TA, by simply re-organizing these traces according to the target intermediate values. This process can be automated based on the implementation knowledge, and its computational cost grows linearly with the number of targets. Concretely, this cost should be small for most concrete implementations, and if needed can be speeded up by assuming sets of intermediate values to leak according to the same model (possibly at the cost of some information loss).

Required Knowledge. Since templates are built by grouping the leakage traces according to some target intermediate values, it requires being able to predict these values. Both for TA and SASCA, this is usually achieved thanks to some key knowledge (or a profiling device). So both attacks can be based on the same assumptions. In fact, their main difference is that any intimate knowledge of the target implementations can – *but does not have to* – be exploited by SASCA (while, e.g. the middle round leakages are useless for DC attacks). The experiments in this paper consider a worst-case scenario where the adversary knows the implementation source code. Another extreme scenario would be to consider only "standard" attack points that can be guessed from the algorithms specifications (e.g. S-boxes inputs/outputs), which would reduce the gain of SASCA compared to TA. Any intermediate situation could be investigated, corresponding to various tradeoffs between implementation details and attack efficiency.