Algebraic Side-Channel Attacks Beyond the Hamming Weight Leakage Model

Yossef Oren[1], Mathieu Renauld[2], François-Xavier Standaert[2], Avishai Wool[1]

[1] Cryptography and Network Security Lab, School of Electrical Engineering
Tel-Aviv University, Ramat Aviv 69978, Israel
{yos, yash}@eng.tau.ac.il
[2] Université catholique de Louvain, Crypto Group
Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium
{mathieu.renauld, fstandae}@uclouvain.be

Abstract. Algebraic side-channel attacks (ASCA) are a method of cryptanalysis which allow performing key recoveries with very low data complexity. In an ASCA, the side-channel leaks of a device under test (DUT) are represented as a system of equations, and a machine solver is used to find a key which satisfies these equations. A primary limitation of the ASCA method is the way it tolerates errors. If the correct key is excluded from the system of equations due to noise in the measurements, the attack will fail. On the other hand, if the DUT is described in a more robust manner to better tolerate errors, the loss of information may make computation time intractable. In this paper, we first show how this robustness-information tradeoff can be simplified by using an optimizer, which exploits the probability data output by a side-channel decoder, instead of a standard SAT solver. For this purpose, we describe a way of representing the leak equations as vectors of aposteriori probabilities, enabling a natural integration of template attacks and ASCA. Next, we put forward the applicability of ASCA against devices which do not conform to simple leakage models (e.g. based on the Hamming weight of the manipulated data). We finally report on various experiments that illustrate the strengths and weaknesses of standard and optimizing solvers in various settings, hence demonstrating the versatility of ASCA.

1 Introduction

In an algebraic side-channel attack (ASCA), the attacker is provided with a **device under test** (DUT) which performs a cryptographic operation (e.g. encryption). While performing this operation the device emits a measurable **side-channel leakage** that is expected to be data dependent. A typical example of such a leakage is a power consumption or electromagnetic radiation trace. As a result of the data dependence, a certain amount of **leaks** is modulated into the trace. These leaks are functions of the internal state of the DUT, which can teach the attacker about intermediate computations at various stages of the cryptographic operation. The trace, and its embedded leaks, are subjected to some **noise** due to interference and to the limitations of the measurement setup [15, §1.2]. In order to recover the secret key from a power trace using an ASCA, the attacker generally performs different steps as we describe next.

1. In a first offline phase, the DUT is analyzed in order to identify the position of the leaking operations in the traces, for instance by using classical side-channel attacks like CPA [5] or template attacks [7].
2. Next, in a second offline phase, the DUT is profiled and a decoding process is devised, in order to map between a single power trace and a vector of leaks. A common output of the decoder would be the Hamming weight of the processed data as in [18], but many other decoders are possible.
3. After the offline phase, the attacker is provided with a small number of power traces (typically, a single trace). The traces are accompanied by **auxiliary information** such as known plaintext and ciphertext. The decoding process is applied to the power trace, and a vector of leaks is recovered. This vector of leaks may contain some errors, e.g. due to the effect of noise.
4. The leak vector, together with a formal description of the algorithm implemented in the DUT, is represented as a system of equations. This equation set also includes any additional auxiliary information.
5. A machine solver evaluates the equation set and attempts to find a candidate key satisfying it. In the case of an optimizing solver, a goal function is also specified to define the optimality of each candidate solution. The solver may fail to terminate in a tractable time, or otherwise return a candidate key.
6. Eventually, and optionally, some post-processing can be used, e.g. in order to brute force the remaining key candidates provided by the solver.

As indicated in the above list, there are several conditions which must all hold true before such an attack succeeds. First, the correct key should not be excluded from the set of solutions to the equation system. This can happen if the traces are too noisy, or if the decoder is not adapted to the attacked device. Next, the solver should not run for an intractable time. This can happen if not enough side-channel information is provided. Finally, the returned key should be the correct key, or at least a key close enough to allow an efficient enumeration.

Related work. ASCA were introduced by Renauld et al. in [17,18], and first applied to the block ciphers PRESENT [4] and AES [14]. These works showed how keys can be recovered from a single measurement trace of these algorithms implemented in an 8-bit microcontroller, provided that the attacker can identify the Hamming weights of several intermediate computations during the encryption process. Already in these papers, it was observed that noise was the main limiting factor for efficient ASCA. To mitigate this issue, a heuristic solution was introduced in [18], and further elaborated in [22]. The main idea was to adapt the leakage model in order to trade some loss of information for more robustness, for example by grouping hard to distinguish Hamming weight values together into sets. We will denote this approach as set-ASCA. Other improvements regarding the error tolerance of ASCA have also been discussed in [13]. In parallel, an alternative proposal was introduced at CHES 2010, and denoted as Tolerant ASCA (TASCA) [15]. Here, the idea was to include the imprecise Hamming weights in the equation set, and to deal with these imprecisions via the solver. The authors showed how leaking implementations of Keeloq [9] could be attacked in this way, and recently extended their results to the AES case [16].

Our contribution. One primary limitation of the ASCA method lies in its intolerance to errors: if the correct key is excluded from the system of equations, the attack will fail. This can be somehow mitigated by the robustness vs. information tradeoff, but only up to a certain point, as the loss of too much information makes the computation time intractable. In this work, we first show how an optimizing solver can use probability data to retain the robustness required to be error-tolerant, while losing less information than a SAT solver, at the cost of a larger problem representation. For this purpose, we describe a novel way of describing measurement equations directly as vectors of aposteriori probability, using the objective function of the optimizing solver. Next, we discuss the generalization of ASCA from the case of Hamming weight leakages to generic (template-based) models. We show that template attacks and ASCA can be naturally integrated, both with standard solvers and optimizers. We additionally provide experimental results allowing to put forward the strengths and weaknesses of the newly proposed probabilistic TASCA and set-ASCA. Overall, the resulting attacks allow strongly reduced data complexity template attacks, when compared to standard divide-and-conquer key recovery attacks.

Document structure. The rest of this paper is organized as follows. Section 2 describes our experimental setup. Section 3 discusses how to exploit probabilistic information in TASCA and evaluates the performances of this improved attack, compared to set-ASCA and the original TASCA. Section 4 investigates attacks against a device that does not leak according to the well-known Hamming weight leakage model. Finally, concluding remarks are given in Section 5.

2 Experimental setup

Our analysis considers two simulated implementations of the AES Rijndael in 8-bit microcontrollers as DUT. We assumed that no leaks from the key expansion process are available to the solver and that the DUT performs round key expansion in advance. This corresponds to a more challenging scenario, as it was established in [10] that the Hamming weights leaked from an 8-bit microcontroller implementation of the AES during key expansion are sufficient for full key recovery, even without any additional state information. We also assumed that the plaintext and the ciphertext are known to the attacker. We finally exploited the information the device leaks about the 8-bit operands commuting on its data bus. In total, it corresponds to 100 values per round, as described below:

- The **AddRoundKey** operation leaks information about the 16 state bytes after the XOR with the key, as well as information about the key bytes themselves, giving a total of 32 leaks per round.
- The **SubBytes** operation is implemented as a look-up table (LUT) and leaks information about its 16 output state bytes (and not any other internal state information), for a total of 16 leaks per round.
- The **ShiftRows** operation does not leak any information.

– The **MixColumns** operation is implemented using 8-bit XTIME and XOR operations as specified in [8, §5.1], and leaks 36 additional bytes of internal state and 16 leaks for its final state, resulting in a total of 52 leaks per round.

Note that the optimizer we used to perform TASCA was not memory-efficient enough to represent the entire AES encryption in equation form. As a result, we provided it with a known plaintext and the cipher equations for the first round of encryption only. By contrast, the SAT solver was provided with a plaintext/ciphertext pair, and all the cipher equations. In order to have comparable experiments, we only exploited the 100 first round leakages, in both cases.

Regarding the leakage models, we considered two different scenarios. First, we used the templates obtained from a PIC microcontroller. As illustrated in Figure 1, this device closely follows a Hamming weight leakage model. Next, we used the templates obtained from the AES S-box implemented in a 65-nanometer CMOS technology, previously analyzed in [19]. In particular, we selected one of the S-boxes for which the leakage model is not correlated with the Hamming weight of the manipulated data, as illustrated in Figure 2. In both cases, the signal-to-noise ratio was similar and relatively high (with the variance of the signal approximately 10 times larger than the noise variance), yet leading to some decoding errors, as will be investigated next. These setups were selected in order to illustrate the efficiency of ASCA in different implementation contexts.

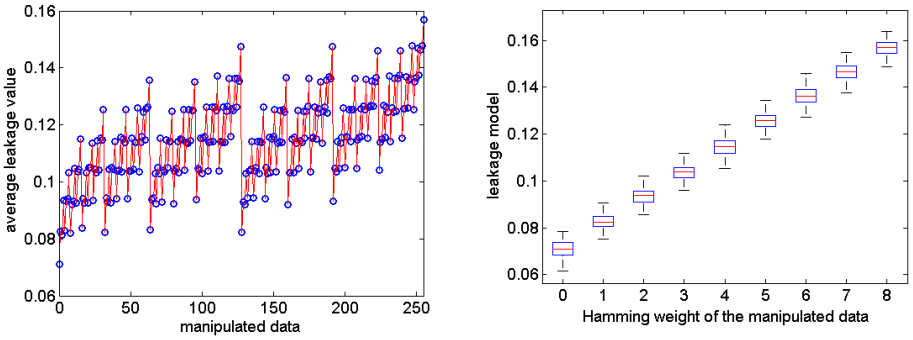

Fig. 1: PIC leakage model: average values (left) and grouped by HWs (right).

The solver used for the TASCA experiments was SCIP version 1.2.0 compiled for Windows 64-bit [3]. This solver is currently the best non-commercial solver available for non-linear optimization problems, as listed by [12]. The solver was run on a quad-core Intel Core i7 950, running at 3.06GHz with 8MB cache. For the set-ASCA experiments we used CryptoMiniSAT 2.9 [21]. This solver won several prizes in SAT competitions (SAT Race 2010 [20] and SAT competition 2011 [1]) and is well adapted to deal with cryptographic problems, as XOR operations (very frequent in cryptographic algorithms) are managed by the solver using specific optimized clauses. The solver was run on on a quad-core Intel Core Intel Xeon X5550 processor, running at 2.67 GHz with 8MB cache.

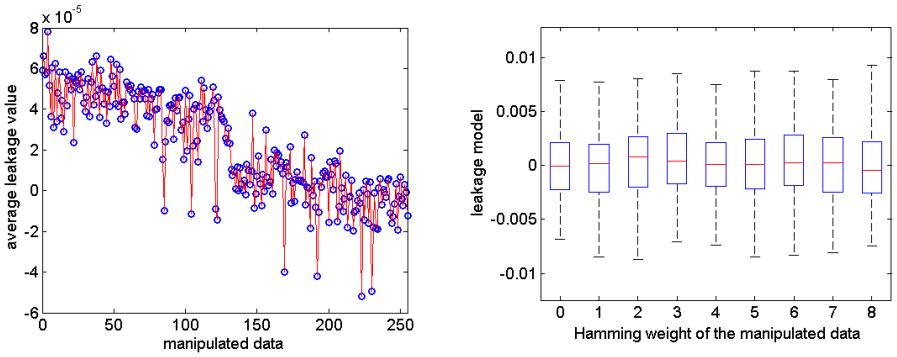

Fig. 2: 65nm S-box leakage model: average values (left) and grouped by HWs (right).

Finally, we note that because of the previously mentioned memory limitation issue, the success condition is defined differently for set-ASCA and TASCA. In the set-ASCA scenario, the entire encryption operation is included in the equation set, meaning that the solver either outputs the correct key or otherwise runs for an intractable time. By contrast, the TASCA solver only exploits the first round equations and, therefore, can sometimes return an incorrect key. We deal with this condition by measuring the amount of incorrect bytes in the result – if 4 bytes or less are incorrect, we assume that the correct key can be recovered from this partially correct key by brute force[3] and declare success. We also recall that the amount of leaks exploited (i.e. 100) was the same in all our experiments.

3 Exploiting probabilistic information

As stated in the previous section, in an ASCA the attacker takes the output of a decoding process and converts it into a series of measurement equations. An example for such a decoding process would be a nearest neighbor decoder, a naïve Bayes decoder [11, §13.2] or a template decoder [7]. In most cases this decoder does not only output "hard" data (i.e. the most likely leak value) but also some additional "soft" information, such as confidence information, a ranking of several possible leaks by decreasing order of likelihood or, most generally, a full vector listing the aposteriori probability for each possible leak value, conditioned

[3] Assume that e of the 16 bytes are incorrect. The attacker must go over all $\binom{16}{e} \approx 2^{4e}/e!$ possible locations for those errored bytes, then try $256^e = 2^{8e}$ possible candidate assignments for these positions, resulting in an approximate total effort of $2^{4e} \cdot 2^{8e} = 2^{12e}$ AES operations. Most modern Intel CPUs have a native implementation of AES (AES-NI), which allows a sustained rate of more than 2^{31} AES operations per second [2]. Thus, an attacker can use a single machine with an AES-NI implementation to probe the neighborhood of a candidate key and find the correct key within less than 24 hours, even if 4 of the 16 bytes are incorrect.

on the received trace. In this section, we discuss an improvement of TASCA which is capable of taking advantage of this soft (probabilistic) information.

For this purpose, let us start from the standard scenario of a Hamming weight-based ASCA. In this context, the tradeoff between robustness and information is generally achieved by choice of the **set size** k. It defines the number of acceptable values for each individual side-channel leak in the equation set, relative to the apriori selection of a leakage model (e.g. the Hamming weight of the manipulated data). The value of k can be either determined as a global constant for all equations in the set (e.g. as in [16,18]), or determined on a per-leak basis according to some heuristic (e.g. as in [13,22]). In the case of a precisely-defined equation set, in which $k = 1$, only the most likely value output by the decoder is accepted. This representation provides the most information, but it cannot tolerate any errors. As the set size k grows, so does the robustness of the equation set, but this comes at the price of a loss of information. The original work on ASCA [17] investigated only the case of $k = 1$. Thus, the single value chosen as most likely by the decoder was entered into the equation set. In the set-ASCA experiments of [13,18,22], more than one value was listed as possible to the solver, sacrificing information for robustness against errors. In this case, each leakage equation would accept the k most likely values, as output by the decoder. The TASCA attack of [16] also uses a set, and additionally uses a goal function to mark one of the value in the set as likelier than the others, without further quantification of this likelihood. This representation is more informative than in a set-ASCA, but it still does not fully take advantage of the information provided by the decoder.

We now present a more expressive way of representing the probability information provided by the decoding phase. In the most general case, the decoder outputs a full probability vector for each leak, listing the aposteriori probability of the leak having each possible value, conditioned on the specific trace being received. This output is typical for e.g. template decoders [7]. In the case of a Hamming weight-based template decoder, each potential leak will have an associated vector of 9 aposteriori probabilities corresponding to Hamming weights 0 to 8. If we further assume that individual leaks are uncorrelated, then the combined probability of all leaks in the trace is proportional to the product of the individual aposteriori probabilities. Of course, most of these combinations are impossible, since they violate the cipher equations. The goal of the solver in this case would be to find the set of leaks that maximizes the product of aposteriori probabilities while still corresponding to a valid encryption. As shown in [18], the exact values of the Hamming weight leaks provide enough information to uniquely and efficiently find the correct key of an AES encryption. If we define the exact value of leak i as x_i, we can define the objective of the attack as:

$$x_1 \cdots x_m = \arg \max_{x_1 \cdots x_m} \prod_{i=1 \cdots m} \Pr(x_i | trace) \, s.t. \, \text{cipher eq'ns are satisfied.}$$

Since the goal function of the SCIP solver is expressed as a sum of integers which must be minimized, we represent the objective using this equivalent expression:

$$x_1 \cdots x_m = \arg \min_{x_1 \cdots x_m} \sum_{i=1 \cdots m} -\log\left(\Pr(x_i|trace)\right) \; s.t. \text{ cipher eq'ns are satisfied.}$$

The representation of a probability vector $\overline{p_x}$ for a certain leaked Hamming weight x with set size k as a side-channel leak equation is thus split into two parts: the *constraint set* and the *goal term.* The constraint set is very straightforward – it considers k different events called "HW(x) is 0", "HW(x) is 1", etc., describes each event in terms of the relevant combination of bits in the leaked byte, and finally requires that one and only one of these events be true in for each leak in a satisfiable solution. The goal term matches each event with a corresponding probability. Each probability p is represented in the goal term as $-\lfloor C \log p \rfloor$, where C is an implementation parameter. The goal terms of all leaks in the system are then summed together to create the global goal function.

3.1 Experimental Validation

In order to compare set-ASCA, basic TASCA and TASCA with probabilities in the Hamming weight leakage model, we designed a first set of experiments, based on simulated leakages from the PIC device illustrated in Figure 1. For each experiment, we list the *decoding success rate* – the proportion of traces for which all 100 correct leaks are included in the 100 k-sized sets provided by the decoder – and the *key recovery success rate* – the proportion of traces for which the solver returned the correct key within a reasonable time. We also report on the (median and maximum) solving time and show the average *number of correct key bytes* in case of successful attacks. For set-ASCA, both the plaintext and ciphertext are included in the equation system, meaning that an attack can only succeed when every 16 key bytes are correct. On the other hand, in TASCA instances only the plaintext is used, meaning that when the set size increases, several keys can be valid according to the algebraic representation. In this latter case the average number of correct key bytes can be below 16. As explained in Section 2, the attack is still considered successful when at least 12 out of the 16 key bytes are correct. Our results are summarised in Table 1.

These experiments lead to a number of interesting observations. First and as expected, they clearly illustrate the information vs. robustness tradeoff. That is, the probability of decoding success grows as the set size grows (better robustness). However, this impacts the performances of the different attacks in different manners. For set-ASCA, the solving time quickly increases to the point of intractability, because of a lack of information. By contrast, the basic TASCA are more resistant to the loss of information: as the set size grows the running time increases, but the key recovery probability is much less affected. Yet, the limited information available to the solver causes parts of the key to be recovered incorrectly in some cases, which then requires an additional brute forcing step. Combining probabilistic information with a set size of 3 finally allowed the

attack	set size	decoding success	key rec. success	med. solving time	max. solving time	# of correct key bytes
set-ASCA	1	0%	0%	N/A	N/A	N/A
set-ASCA	2	83%	83%	2 seconds	6 seconds	16
set-ASCA	3	100%	0%	24+ hours	24+ hours	N/A
basic TASCA	1	0%	0%	N/A	N/A	N/A
basic TASCA	2	83%	75%	43.7 minutes	11.8 hours	14.48
basic TASCA	3	100%	80%	16.8 hours	66 hours	13.25
prob. TASCA	1	0%	0%	N/A	N/A	N/A
prob. TASCA	2	83%	82%	56.7 minutes	10.07 hours	15.88
prob. TASCA	3	100%	100%	8.2 hours	143 hours	16

Table 1: set-ASCA, basic TASCA and probabilistic TASCA experimental results against the PIC microcontroller simulated leakages with Hamming Weight model.

optimizer to recover the correct key in virtually all experiments. It also reduced the running time compared to the basic TASCA. Yet, both TASCA approaches are still much slower than the set-ASCA approach when it succeeds (e.g. for set sizes $k \leq 2$), due to the more complex design of the optimizer. This is also reflected by the larger memory requirements of the TASCA solving phase.

Summarizing, the TASCA approaches allow improved flexibility as they systematically deal with the information vs. robustness tradeoff during the solving phase. By contrast, set-ASCA shift this problem to the decoder phase. In case of low-noise scenarios, or whenever the adversary can average the measurements, set-ASCA is the method of choice because of its reduced memory requirements and solving times. It also allows exploiting all the leaks (i.e. not only the first round ones). By contrast, the more the measurements are noisy and/or hard to interpret by the adversary (e.g. because of countermeasures), the more the TASCA approaches becomes interesting, thanks to its optimizing features.

4 Beyond the Hamming weight model

As illustrated in Section 2, Figure 2, the leakage of certain devices (e.g. in 65nm and smaller technologies) cannot always be precisely expressed with simple models. As a result, it is interesting to investigate how ASCA/TASCA can be extended towards these more challenging scenarios. In this section, we show how to move from Hamming weight-based models to more generic ones.

For this purpose, let us assume that the attacker has performed template-based profiling of the DUT [7]. Given a power trace, the he can now create a probability vector for each leak, where each entry in this vector matches a certain possible leak value, and each value in the vector is the *aposteriori* probability of this leak conditioned on the power trace being processed. Assuming the DUT has

an 8-bit architecture, each such vector contains 256 entries. The decoding process will output a number of such vectors – one for every leak in the equation set. As for the Hamming weight model, we use this side-channel information to restrict the size of the solution space. In order to do so, we define a parameter called the **support size** k', which is comparable (though not identical) to the set size k in the previous section. It corresponds to the amount of possible values associated to each leak. These values are chosen according to the probability vector: the k' most probable values are considered possible, and the others are rejected. Hence, the value of k' must be carefully chosen in order to avoid rejecting the correct value from the set of possible ones, making the problem unsolvable.

Representing this generic leakage model as clauses or equations is less easy than for the Hamming weight model. For the set-ASCA, the easiest way to represent a set of k' possible values for a leaked byte x is to exclude all impossible values. For example, in order to exclude the value $x = 9 \Leftrightarrow (x_0, ..., x_7) = (0, 0, 0, 0, 1, 0, 0, 1)$, we add to the SAT problem the clause $(x_0 \cup x_1 \cup x_2 \cup x_3 \cup -x_4 \cup x_5 \cup x_6 \cup -x_7)$. Each set of k' possible values is thus translated into $256 - k'$ clauses with 8 literals per clause. In order to speed up the solving process, we additionally apply some simplification techniques, e.g. reducing the length and number of closes. For the TASCA, the representation of a probability vector $\overline{p_x}$ for a certain leaked byte x with support size k' as a side-channel leak equation is again split into two parts: the *constraint set* and the *goal term*. The constraint set describes k' different events called "x is 0", "x is 1", etc., and requires that one and only one of these events is true for each leak. The goal term matches each event with a corresponding probability. An example of such a representation can be found in Appendix A. Since this representation is especially suited for template-based profiling, we call it **template** TASCA and set-ASCA.

4.1 Impact of the support size and goal function

A first natural question in this new setting is: how small must the support size be for the attacks to succeed, and what is the impact of the probabilistic information that can be added to the optimizer? To answer it, we designed an experiment in which we compared many pairs of template-TASCA instances of single-round AES with different support sizes. In each pair, one of the instances was provided with an **unweighted** probability vector (that is, all nonzero elements in the probability vector are considered of equal probability), while the other was provided with a **weighted** vector function. The latter one was simulated (independent of any actual leakage model) such that the single correct byte value always had a higher probability than all the other ones in the support. For the rest, the instances were identical, with only plaintext provided, such that the solver could potentially output an incorrect key as in the previous section.

The results of this experiment are illustrated in Figure 3. As we can see, they can be divided into four distinct phases. In the first phase (support sizes up to 10), the performance of the weighted and unweighted instances is identical, probably because enough information is available in the support of the function,

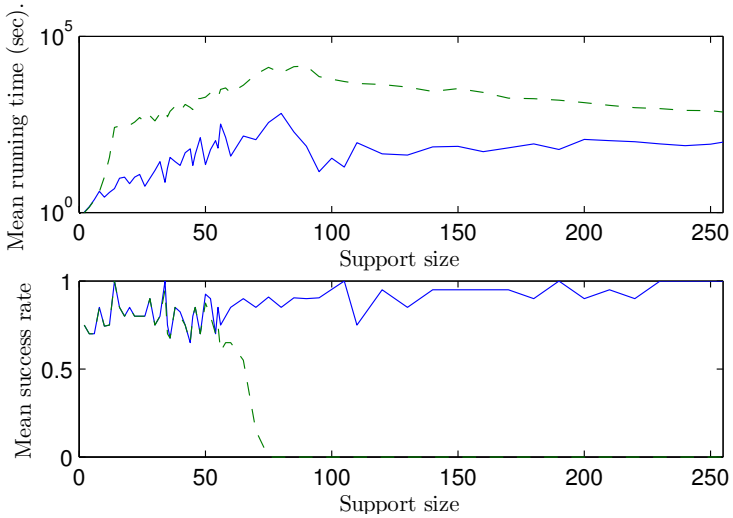

Fig. 3: Template-TASCA attacks with weighted (solid line) and unweighted (dashed line) probability vectors: experimental running time and success rate.

making the additional information in the goal function redundant. In the second phase (support sizes 10 – 50) both the weighted and the unweighted instances end in successful key recovery, but the weighted instances are faster by two orders of magnitude. We see that in this range there is still enough information in the unweighted instances to precisely specify the correct key, but the added information of the goal function allows the optimizer to reach the correct answer more quickly. In the third phase (support sizes 50 – 70) the success rate of the unweighted instances slowly falls to 0, probably because more and more incorrect keys can satisfy the constraint set. However, the additional information in the goal function causes the optimizer to prefer the likeliest solution, which in our case was the correct one. Finally, in the fourth phase (support sizes 70 and up) the large amount of possible keys in the support makes the success rate of the unweighted instances marginally small. Note that even with a full support ($k' = 256$) the performance was still good. This implies that all information about the instances can be encoded into the goal function and not into the constraints, and thus that the correct key will never be excluded from the equation set.

Furthermore, it could be argued that the running time of the weighted templates is faster than that of the unweighted ones because the correct guess is always the highest ranked. To investigate this scenario, we repeated the same experiment, this time setting the rank of the correct key candidate to 2. The change in rank caused the running time of the weighted case to increase, but had no effect on the success rate. We verified this behavior for ranks of up to 14.

attack	set size	dec. SR	key rec. SR	med. solving time	max. solving time	# of correct key bytes
set-ASCA	64	15.5%	15.5%	2 sec.	2 sec.	16
set-ASCA	90	90%	90%	265 sec.	24+ hours	16
set-ASCA	100	100%	29%	24+ hours	24+ hours	16
prob. TASCA	64	15.5%	15.5%	35.88 sec.	86.03 sec.	16
prob. TASCA	90	90%	90%	245.72 sec.	869.4 sec.	16
prob. TASCA	100	100%	100%	342.76 sec.	21271 sec.	16
prob. TASCA	256	100%	100%	62254 sec.	48+ hours	16

Table 2: Template set-ASCA and probabilistic TASCA experimental results against simulated leakages from a 65nm S-box, with generic template model.

4.2 Experimental validation

As in the previous section, we verified the effectiveness of our attacks by performing several experiments. This time, we considered a DUT where the simulated leakages are generated according to the model of the 65nm ASIC implementing one AES S-box presented in Section 2. As illustrated in Figure 2, the leakage function of this device is very different from the Hamming weight model. Thus, it constitutes a perfect target for our template-based set-ASCA and TASCA. In a first step, we profiled the AES S-box, resulting in 256 templates corresponding to the 256 possible transition values. Each univariate template assumes a Gaussian noise and was characterized by a mean value μ and a noise standard deviation σ. In a second step, we used Bayesian inversion to simulate the classification probability $\Pr(x_i|trace)$ from the template output $\Pr(trace|x_i)$.

The results of the attacks are summarized in Table 2, where we selected different support sizes k'. As expected, smaller support sizes lead to more unsatisfiable/unsolvable problems, but these problems are solved faster, meaning a higher success rate for the computation phase. As soon as $k' \geq 100$, all the problems are solvable, but the solving process becomes much longer. We compared two attacks: the set-ASCA and the probabilistic TASCA. Both essentially confirmed our previous observations. Namely, the set-ASCA instances are very fast to solve for low support sizes, but suddenly increase in difficulty between $k' = 90$ and $k' = 100$. By comparison, probabilistic TASCA instances for low support sizes are much slower to solve than set-ASCA ones. Nevertheless, the difficulty of solving TASCA instances increases slower than for set-ASCA ones. In the end, probabilistic TASCA is able to solve problems with support size $k' = 256$, which is totally infeasible for set-ASCA (as $k' = 256$ means no side-channel information for set-ASCA instances). Summarizing, we again observe a tradeoff between efficiency (set-ASCA) and flexibility (probabilistic TASCA).

Besides, Table 3 presents a comparison of the Hamming weight model and the template model in terms of set size and support size. For each set size, *i.e.* for each number of possible Hamming weight values, the table details the corresponding average support size $\bar{k}'$ and the minimum and maximum support sizes $k'_{\min}$ and $k'_{\max}$. For instance if $k = 2$, the two consecutive Hamming weight

values $\mathrm{HW}(x) = \{0 \text{ or } 1\}$ correspond to $k'_{\min} = 9$ possible transition values out of 256. Similarly, the two Hamming weight values $\mathrm{HW}(x) = \{3 \text{ or } 4\}$ correspond to $k'_{\max} = 126$ possible transition values out of 256. On average, a leak that is represented by a set of 2 possible Hamming weight values can also be represented by a set of $\bar{k}' = 95$ possible transition values. Contrarily to the attacks using the template model where the support size is the same for every leak, the attacks using the Hamming weight model present different support sizes. Therefore, some sets of Hamming weight values offer more information than others. In other words, the Hamming weight information is not uniformly distributed over the 100 considered leaks in the first AES round. This table allows us to compare and better understand the results from Table 1 and Table 2. For example, we observe that solving set-ASCA problems with the Hamming weight model for set size $k = 2$ takes about 2 seconds, while solving set-ASCA problems with the template model for a similar support size $k' = 90$ takes more than 250 seconds. Hence, set-ASCA seems to take advantage of the non-uniform information proposed by the Hamming weight model. This confirms observations already made in [6]: the SAT solver usually exploits small parts of the equation system where the information is most concentrated. Interestingly, the same is not true for probabilistic TASCA: template instances with support size 90 or 100 are faster to solve than Hamming weight instances with set size 2. Our hypothesis is that for probabilistic TASCA, the goal function contains more information when using the template model than the Hamming weight model, as the Hamming weight model does not make any distinction between different transition values with the same weight. As a consequence, the advantage offered by non-uniform information is counterbalanced by a less informative goal function.

Set size k	$\bar{k}'$	$k'_{\min}$	$k'_{\max}$
1	50	1	70
2	95	9	126
3	134	37	186

Table 3: Comparison between set sizes (Hamming weight model) and corresponding average $\bar{k}'$, minimum $k'_{\min}$ and maximum $k'_{\max}$ support sizes (template model).

5 Concluding remarks

In this paper we showed how both optimizers and solvers can be used to perform ASCA even if the leakage function does not conform to the Hamming weight model. The solver-based approach (set-ASCA) was shown to be faster than the optimizer-based approach (TASCA) when a high degree of robustness is not required (for example, if the traces can be preprocessed by averaging many traces). However, in cases when robustness is required, the optimizer approach was shown to be both faster and with higher success rate than the solver-based approach. This is due to the additional flexibility afforded by the optimizer goal function,

which allowed us to construct a generic representation of the measured leak as a vector of aposteriori probabilities. The new flexible representation presented in this paper allows TASCA and set-ASCA attacks to be used as a natural match for template attacks. To carry out a combined Template-TASCA or Template-set-ASCA, the attacker should not only create templates for the original key bytes, but also for all intermediate values. The solver step will then replace any traditional post-processing step used in template attacks such as brute-force key enumeration. As a result, we further illustrated how an ASCA can be used effectively as a post-processing step of a template attack, dramatically reducing its data complexity. We believe that attention should be given to this capability when evaluating the security of systems using template attacks.

Future work. Optimizers are less efficient than solvers in terms of running time, but since a solver does not have any efficient way of representing the objective function which contains the aposteriori probabilities, its running time quickly becomes intractable when high robustness is desired. It may be possible to increase the robustness of the solver-based approach by finding a better way of choosing the set size k or support size k'. For example, instead of choosing the k' most likely value, the solver can set a threshold probability and include in its support all values with a higher probability than this threshold. The solver might also be used in an adaptive manner - slowly increasing the support size while the solver returns unsatisfiability, until we reach the minimal sized support for which a solution exists. Quite naturally, the opposite approach would be interesting too. Namely, the search for improved optimizers, allowing to represent more complex problems with reduced memory efficiency would be another way to close the gap between set-ASCA and TASCA. Finally, it would be interesting to carefully investigate the connection between the offline and online phases of a template attack on the success of template-TASCA and template set-ASCA. A better model obtained through better profiling in the offline phase should intuitively allow the use of lower-quality data in the online attack phase, and vice versa.

Acknowledgements. Mathieu Renauld is a PhD student funded by the Walloon region through the SCEPTIC project. François-Xavier Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in part by the ERC project 280141 (acronym CRASH) and with support from Wallonia-Brussels International. The authors wish to thank the anonymous reviewers for their encouraging and insightful comments.

References

1. SAT 2011 Competition. http://www.cril.univ-artois.fr/SAT11/phase2.pdf.
2. Kahraman Akdemir, Martin Dixon, Wajdi Feghali, Patrick Fay, Vinodh Gopal, Jim Guilford, Erdinc Ozturc, Gil Worlich, and Ronen Zohar. Breakthrough AES Performance with Intel AES New Instructions. Technical report, Intel Corporation, October 2010. `http://software.intel.com/file/27067`.
3. Timo Berthold, Stefan Heinz, Marc E. Pfetsch, and Michael Winkler. SCIP – Solving Constraint Integer Programs. SAT 2009 competitive events booklet, 2009.

4. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
5. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
6. Claude Carlet, Jean-Charles Faugère, Christopher Goyet, and Guénaël Renault. Analysis of the algebraic side channel attack. *J. Cryptographic Engineering*, 2(1):45–62, 2012.
7. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
8. Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael, 1998.
9. Steven Dawson. Code Hopping Decoder using a PIC16C56. Microchip confidential, leaked online in 2002, 1998.
10. Stefan Mangard. A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion. In Pil Joong Lee and Chae Hoon Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2002.
11. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
12. Vasco Manquinho and Olivier Roussel. Pseudo-Boolean Competition 2009. Online, July 2009. `http://www.cril.univ-artois.fr/PB09/`.
13. Mohamed Saied Emam Mohamed, Stanislav Bulygin, Michael Zohner, Annelie Heuser, and Michael Walter. Improved Algebraic Side-Channel Attack on AES. Cryptology ePrint Archive, Report 2012/084, 2012. `http://eprint.iacr.org/`.
14. Information Technology Laboratory (National Institute of Standards and Technology). *Announcing the Advanced Encryption Standard (AES)*. Computer Security Division, Information Technology Laboratory, NationalInstitute of Standards and Technology, Gaithersburg, MD, 2001.
15. Yossef Oren, Mario Kirschbaum, Thomas Popp, and Avishai Wool. Algebraic Side-Channel Analysis in the Presence of Errors. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2010. `http://iss.oy.ne.ro/TASCA`.
16. Yossef Oren and Avishai Wool. Tolerant Algebraic Side-Channel Analysis of AES. Cryptology ePrint Archive, Report 2012/092, 2012. `http://iss.oy.ne.ro/TASCA-eprint`.
17. Mathieu Renauld and François-Xavier Standaert. Algebraic Side-Channel Attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Inscrypt*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2009.
18. Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009.
19. Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2011.

20. Carsten Sinz. SAT-Race 2010. http://baldur.iti.uka.de/sat-race-2010/results.html, 2010.
21. Mate Soos. CryptoMiniSat2. http://www.msoos.org/cryptominisat2/.
22. Xinjie Zhao, Tao Wang, Shize Guo, Fan Zhang, Zhijie Shi, Huiying Liu, and Kehui Wu. SAT based Error Tolerant Algebraic Side-Channel Attacks. 2011 Conference on Cryptographic Algorithms and Cryptographic Chips (CASC2011), July 2011.

A Appendix: A Sample Template-TASCA Instance

The appendix demonstrates the format of leak equations used in a Template-TASCA attack, following the notation introduced in Subsection 4. The equations are given in the OPB format supported by the SCIP solver [3].

Assume that during the cryptographic operation the DUT processes two bytes x and y. Using a template profiling step, the attacker creates a model of the leakages produced by the processing of x and y. Given a trace, the attacker can now use this information to calculate vectors of aposterioti probabilities for x and for y ($\overline{p_x}, \overline{p_y}$), conditioned on the specific trace having been received. The support size has been set to $k' = 4$. The vectors passed to the solver are $\overline{p_x} = \{\frac{1}{2}, \frac{1}{3}, \frac{1}{12}, \frac{1}{12}, 0 \cdots 0\}, \overline{p_y} = \{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, 0, 0, \frac{2}{5}, 0 \cdots 0\}$. The attacker also chooses the implementation parameter $C = 10$ to efficiently capture the probability information while limiting the ultimate size of the goal term. The attacker then uses the aposteriori probability vectors to generate the following equations:

```
* Leak Equations:
+1 ~x_is_00 +1  ~x_0 ~x_1 ~x_2 ~x_3 ~x_4 ~x_5 ~x_6 ~x_7 = 1;
+1 ~x_is_01 +1   x_0 ~x_1 ~x_2 ~x_3 ~x_4 ~x_5 ~x_6 ~x_7 = 1;
+1 ~x_is_02 +1  ~x_0  x_1 ~x_2 ~x_3 ~x_4 ~x_5 ~x_6 ~x_7 = 1;
+1 ~x_is_03 +1   x_0  x_1 ~x_2 ~x_3 ~x_4 ~x_5 ~x_6 ~x_7 = 1;
+1 x_is_00 +1 x_is_01 +1 x_is_02 +1 x_is_03 = 1;

+1 ~y_is_00 +1  ~y_0 ~y_1 ~y_2 ~y_3 ~y_4 ~y_5 ~y_6 ~y_7 = 1;
+1 ~y_is_01 +1   y_0 ~y_1 ~y_2 ~y_3 ~y_4 ~y_5 ~y_6 ~y_7 = 1;
+1 ~y_is_02 +1  ~y_0  y_1 ~y_2 ~y_3 ~y_4 ~y_5 ~y_6 ~y_7 = 1;
+1 ~y_is_05 +1   y_0 ~y_1  y_2 ~y_3 ~y_4 ~y_5 ~y_6 ~y_7 = 1;
+1 y_is_00 +1 y_is_01 +1 y_is_02 +1 y_is_05 = 1;

* Goal term:
min: +6 x_is_00 +10 x_is_01 +24 x_is_02 +24 x_is_03 ...
   +16 y_is_00 +16 y_is_01 +16 y_is_02 +9 y_is_05;
```

In addition to these leak equations, the instance will also contain additional equations which describe the cryptographic meaning of the variables x and y, as well as equations which capture the auxiliary information available to the attacker (such as known plaintext).