Representation-, Leakage- and Cipher- Dependencies in Algebraic Side-Channel Attacks

Mathieu Renauld[*], François-Xavier Standaert[**]

UCL Crypto Group, Université catholique de Louvain, B-1348 Louvain-la-Neuve.
e-mails: `mathieu.renauld,fstandae@uclouvain.be`

Abstract. By combining the extraction of side-channel information leakages with classical cryptanalysis techniques, the recently introduced Algebraic Side-Channel Attacks trade a part of the data complexity in standard DPA attacks for more computations. But predicting the success rate of such attacks is made harder because of the numerous parameters that come into play when solving large algebraic systems of equations. In this paper, we study the impact of three of these parameters empirically, along with the metrics needed to quantify them. First, we analyze the efficiency of different representations of the side-channel information as low degree boolean equations. Second, we investigate the impact of different types of information leakages on the attack resolution times. Third, we discuss how these conclusions depend on the target ciphers. From simulated experiments performed in various contexts, we finally provide some more general intuitions for the security of leaking devices.

1   Introduction

In the classical cryptanalysis setup for block ciphers, the adversary has a black box access to the cryptographic device, meaning that he knows a set of plaintexts and corresponding ciphertexts. In the side-channel cryptanalysis setup, the adversary also obtains a physical access to the cryptographic device. This physical access can be used to perform measurements during the encryption process and thus provides useful information on the computations of the device. With the increasing number of small electronic devices widely used in everyday life, side-channel attacks can be applied to various targets (smart cards, RFID tags,...), raising new security issues. Indeed, it turns out that even if the encryption algorithm is secure in the classical cryptanalysis setup, its implementation can be insecure in the side-channel cryptanalysis setup.

Usual side-channel attacks like the Kocher's DPA [9] try to recover the key bits directly from the observed leakages. Other attacks, though, use the side-channel leakages as information that is then exploited through an offline cryptanalytic phase. For example, the idea of the side-channel collision attack (see [4], [10]) is to use the side-channel information to detect collisions (different intermediate computations that have the same output) in the first two rounds of the AES. Once several collisions are detected, the adversary uses an offline algebraic resolution phase in order to compute the secret key. More recently, the Algebraic Side-Channel Attack (ASCA) was proposed against the

---

block ciphers PRESENT [14] and AES [15]. The ASCA can be seen as a generalization of collision attacks: the adversary tries to recover a lot of small pieces of side-channel information about the encryption process. This information must not be very precise, but must be correct with very high probability. For example the adversary can try to recover the Hamming weights of the data transiting on the bus of the device, but any other information can be helpful. This side-channel information is used to enhance an algebraic computation phase: the block cipher is translated into a big system of equations updated with equations describing the side-channel information. The system is then given to a solver (a SAT solver for example) which tries to find the correct values for the key bits. The ASCA presents interesting points: it can succeed with a data complexity of 1, thus with only one measured encryption (whereas a standard DPA usually requires tens or hundreds of measurements), it can also succeed in an unknown plaintext/ciphertext context, and it can break some masked implementations (*e.g.* [12]).

Because of this combination of classical and side-channel cryptanalysis, evaluating the success rate of an ASCA is a difficult task. Indeed, multiple parameters come into play, some of which are usually not taken into account when studying *e.g.* standard DPA. Also, the metrics we need to evaluate this success rate may be different from the ones used for other side-channel attacks. In this work, we present metrics adapted to the ASCA and investigate some of the relevant parameters that influence the success rate of such attacks. More specifically, we focus on three points: (1) representation dependence - what is the impact of the representation of the problem, (2) leakage dependence - which leaked information is more or less favorable to the ASCA, and (3) cipher dependence - what is the impact of the structure of the target block cipher. In the end, our goal is to gain useful intuitions on how the ASCA behaves in function of the parameters: which parameters have the most impact in which situation.

The paper is structured as follows. Section 2 begins with a short background presentation, that is the description of the main concepts we use. Section 3 presents our specific setup: the choices and assumptions we make. In section 4, we detail the first dependence: the representation of our problem as a CNF formula. Then, in section 5 we investigate the second dependence: the information leakage. Finally, in section 6, we introduce the third dependence, the target cipher, and we present our observations about the interactions between these parameters.

2   Background

Comparing side-channel attacks and countermeasures on a fair basis is not an easy task. Indeed, the success of a side-channel attack depends on many parameters : the device and implementation targeted, the measurement setup, the resources of the adversary, the statistical tools used,... In [18], Standaert *et al.* proposed a framework for the analysis of cryptographic implementations. In this section, we present a short description of this framework and of the ASCA.

2.1 The framework

We present here (figure 1) a generic side-channel key recovery to illustrate the main concepts. The framework considers two distinct parts: the *target implementation* and the *side-channel adversary*.
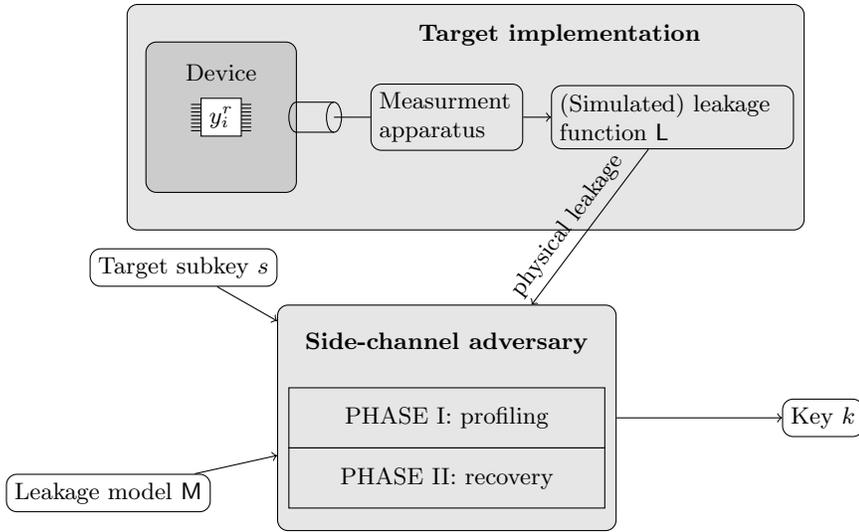

Fig. 1. Side-channel key recovery.

The side-channel adversary exploits a leakage from the target implementation. In our case, this leakage is the power consumption of the device measured through *e.g.* an oscilloscope. The side-channel key recovery consists in recovering a secret key k contained in the device. To do so, the adversary has to perform several *subkey* recovery phases. A subkey is any value depending on the key bits. In practice, every intermediate value processed during an encryption (noted y_i^r - the ith result processed during round r) and every function of these intermediate values qualify as subkeys (*e.g.* a straightforward idea is to consider the bytes of k as subkeys). The important concepts in figure 1 are:

- The *leakage function* L describes the values leaked by the device. The adversary cannot choose the leakage function, which is determined by the physics of the device and the measuring apparatus. For example the leakage can be dependent on the Hamming weight of the values processed (see [11]).
- The *leakage model* M is the model used by the adversary to predict the leaked value in function of the current state of the device. The leakage model is the choice of the adversary; he can arbitrarily select a leakage model or build a model during a profiling phase (a training phase on a practice device). For example, the leakage model could be the Hamming weight model, or the "single bit" model where the adversary assumes that the leakage is correlated with the value of one specific bit (typically the msb).

2.2 Algebraic attacks and ASCA
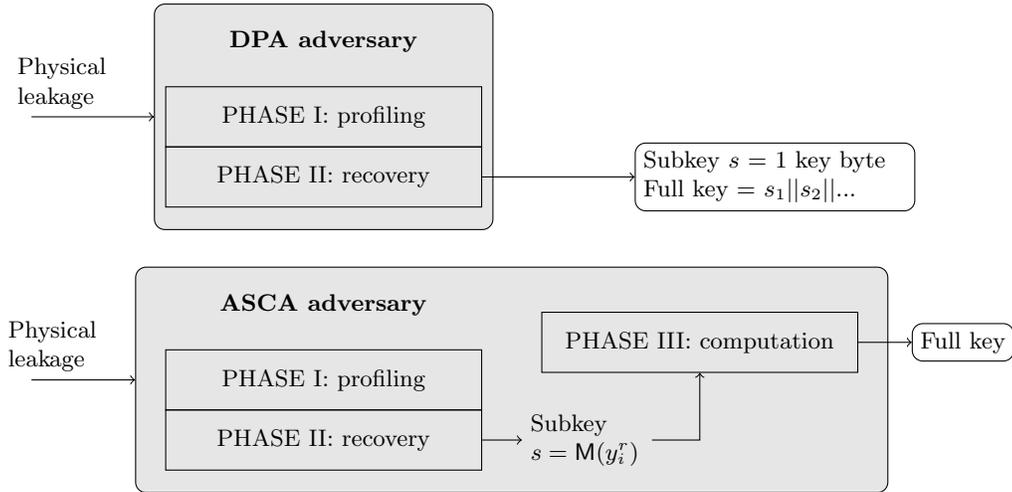
Fig. 2. Comparison of a standard DPA adversary and an ASCA adversary.

As illustrated in figure 2, a standard DPA usually targets subkeys such that their concatenation directly yields the master key k (*i.e.* it has little offline time complexity). By contrast, in an ASCA, the adversary will try to recover simpler target subkeys (*i.e.* that can be recovered with high probability and little data complexity) and to exploit this information in a more elaborated offline cryptanalysis step.

The basic idea of algebraic attacks is to describe a block cipher as a system of boolean equations. The main unknown variables in the system are the key bits, so that solving the system is equivalent to recovering the key. Theoretically, knowing a few pairs of plaintext/ciphertext is enough to determine the secret key. However, solving such a big system of equations is a hard problem and is generally intractable with today's computers. There exist several techniques to attempt to solve this kind of systems: SAT solvers [17], XSL algorithm [5], Gröbner bases [7],...

Algebraic side-channel attacks (ASCA) were recently presented, applied to the block ciphers PRESENT [14] and AES [15]. The main idea of this type of attacks is to combine a side-channel recovery phase with an algebraic cryptanalysis phase in order to recover the secret key. Indeed, side-channel information can usually be translated into algebraic equations; it seems natural to add this information to the system of equations describing the targeted block cipher. If enough side-channel information is added, the resulting enhanced algebraic system can be solved within a reasonable amount of time. In this paper, we transform the system of equations into an equivalent satisfiability problem and we solve it using a SAT solver (we use the zChaff solver [19]).

4

While the selection of target subkeys is usually straightforward in a standard DPA, it can be more tricky in ASCA. In this paper, following the suggestion of [14, 15], we will consider subkeys that directly correspond to the leakage model: $s = \mathsf{M}(y_i^r)$. For example, if the leakage model is the Hamming weight model, the subkeys are the Hamming weights of the data processed. Then, we use the algebraic computation step to (try to) recover the master key k in one go.

Phases II and III together in figure 2 form the attack part (phase I being the training part against a tweakable device). The attack can be evaluated in function of complexity parameters for these two phases: τ, m and q (time, memory and data complexity; the data complexity being the number of encryptions measured). The ASCA illustrates thus the tradeoff that can be made in the attack part: a side-channel attack with a high number of queries (high q) can usually recover the key without performing many calculations (τ and m are negligible). But it is also possible to build an attack that can work with very few queries ($q = 1$ or 2) at the price of a (possibly very) intensive computation phase (high τ and m).

2.3 Evaluation metric

In [18], the authors decompose the security evaluation in two metrics: an information theoretic metric used to compare implementations (how much information leaks, independently of the adversary), and security metrics used to compare adversaries (how efficiently the adversary can exploit the available information). The security metric is based on the success rate of the attack. In our case we have to change this metric. Indeed, the success rate of the ASCA is very dependent on the amount of time allowed for the computations. Theoretically, the solver will always find the correct key. The main difference between a successful and a failed ASCA is the time required to reach the solution. Thus we define our security metric based on the solving time: the *median solving time*, the minimum time t such that at least 50% of the solving times are below t. We choose the median time over the average time because the solving times follow an exponential distribution, and the median gives less importance to outsiders.

3 Our setup

In this work, we consider only perfect subkey recovery phases (with 100% accuracy). Indeed, we would like to study the impact of the leakage function on the success of the attack, without considering the possibility of recovering several hundred leaked values with high probability (see [14] for discussions about this problem). So, we did not perform any real measurement, but we used simulated leakage functions depending only on the intermediate 8-bit values transiting on the bus of the device: $\mathsf{L}(y_i^r)$. Here we present the other choices we made in our setup.

Algorithm: we consider two encryption algorithms: the block ciphers AES and PRESENT. The AES [2] is a widely used block cipher, chosen as a standard by the NIST. PRESENT, proposed by Bogdanov *et al.* in [13], is a lightweight block cipher designed for constrained environments (RFID tags, sensor networks,...). These two targets are complementary: the AES is a well known and studied block cipher with a complex algebraic

structure, whereas PRESENT is a block cipher adapted to small devices with a much simpler algebraic structure. The impact of the algorithm is investigated in section 6.

Target operations: for each block cipher, it is important to know which operations leak, or in other words which data are processed by the device. The more operations during the encryption process (*i.e.* the more clock cycle in our implementation, the more side-channel leakage we can recover. We consider the same target operations as in [14] and [15]: the outputs of the XOR operations (with the subkeys for example) and the outputs of the substitutions[1]. The AES is implemented with an operation-based MixColumn, giving 4*13 additional targets per MixColumn layer, for a total of 788 potential targets. The 31-round PRESENT gives 496 potential targets.

Leakage functions: we simulated different families of leakage functions. All the following leakage functions are defined over an 8-bit intermediate value y_i^r processed at some point by the device.

- $rand(n) : \mathsf{L}(y_i^r) = \mathbf{Z}_{y_i^r}$. The leakage function of the simulated device is determined by a random vector $\mathbf{Z}$; each element of $\mathbf{Z}$ being chosen at random between 0 and $n - 1$.
- $parity(n, m) : \mathsf{L}(y_i^r) = \sum_{j=0}^{j<n} \left(2^j . \bigoplus_{k=0}^{k<m} y_i^r[j*m+k] \right)$. In other words, the leakage function provides the adversary with the parity bits of the n first groups of m bits of y_i^r.
- $parity - rand(n, m) : \mathsf{L}(y_i^r) = m.parity(1, n) + rand(m)$. It is a combination of the two previous functions.
- $W_H : \mathsf{L}(y_i^r) = \sum_j y_i^r[j]$. The classical Hamming weight function.
- $linear(n)$. Somewhat a generalization of the Hamming weight function. This function is based on a linear combination of the values of the bits of y_i^r:
  1. The 256 values of the vector $\mathbf{V} \in \mathbb{R}^{256}$ are calculated as follows: $v_{y_i^r} = \sum_j \alpha_j y_i^r[j]$ with random α_j (here, we choose for each simulated device random $\alpha_j \in \mathbb{R}$ between 1 and 5).
  2. The next step is to reduce the number of different values to n. To do so, we successively merge the two closest values v_i and v_j together into a new value $v_{i,j} = (v_i + v_j)/2$. The values v_i and v_j are removed from $\mathbf{V}$ and replaced by v_{ij}. We keep track of the indexes of the merged values.
  3. The previous step is iterated while $\mathbf{V}$ contains more than n elements. When the number of elements in $\mathbf{V}$ becomes equal to n, the procedure is stopped. The vector $\mathbf{Z}$ defining the leakage function is built by assigning to each index i the final value $v_{i_1,...,i_k}$ which contains the original value of index i: $z_i = v_{i_1,...,i_k}$ if $i \in \{i_1, ..., i_k\}$.

Some of these leakage functions are purely theoretical ones ($rand$, $parity - rand$), whereas others could be considered as realistic ones. The Hamming weight/Hamming distance leakage function is often used (see [3]). The linear leakage function is considered *e.g.* in stochastic models [16]. And the $parity(1, 1)$ leakage function (the attacker knows

---

[1] We consider a precalculated key schedule that leaks no direct information about the key bits.

the value of one bit of the value processed) corresponds to probing attacks and has been used in Dinur's and Shamir's side-channel cube attack [6]. In section 5, we detail how we measure these leakage functions.

4   Impact of the representation

Our setup being presented, we can now focus on the first issue we want to investigate: the representation dependence. The SAT solvers does not use an optimal algorithm, but rather smart heuristics. These heuristics don't work the same way with two equivalent representations[2] of the same problem. Finding a "good" representation for a given problem is thus important for a successful attack. In this section we study the impact of the representation on the success of the attack.

Most SAT solvers take as input a boolean formula in *conjunctive normal form* (CNF). A CNF is a conjunction (AND, $\wedge$) of clauses, each clause being a disjunction (OR, $\vee$) of literals (a literal is a variable, x, or NOT a variable, $-x$). The goal of the SAT solver is to prove that a formula is satisfiable, by finding a valid assignment for all variables, or unsatisfiable, by finding conflictual clauses. In the case of the ASCA, the algebraic system of equations is translated into a CNF formula, the SAT solver has to find the unique solution in order to prove the satisfiability of the system. The SAT solver principally uses an exhaustive search with some smart heuristics. The solver usually works better with short clauses, as the literals guessed are propagated faster through short clauses and thus the possible conflicts appear earlier.

Let F be the CNF formula built from the algebraic system of equations representing the block cipher, with one known pair plaintext/ciphertext. Without additional information, the solver should not be able to prove that F is satisfiable in a reasonable amount of time. The adversary then adds side-channel information to F: for each different leaked value $\mathsf{L}(y_i^r) = l$, he adds a formula C_l which defines the set of possible values y_i^r associated to l. There are several ways to translate a leakage function into the different formulas C_l.

Enumeration: the first idea is to simply enumerate all impossible values associated to l: "if $\mathsf{L}(y_i^r) = l$, then $C_l := (y_i^r \neq 0) \wedge (y_i^r \neq 1) \wedge ...(y_i^r \neq 255)$". This is the exhaustive method which works for every possible leakage function and produces long clauses (with 8 literals since y_i^r is a 8-bit value).

Compact representation: some leakage functions can be represented as a simple algebraic relation between the bits of y_i^r. This simple relation can be exploited to describe the leakage function with very short clauses. For example, the function $parity(1,1)$ (that is $\mathsf{L}(y_i^r) = y_i^r[0]$) has two possible output values ($l = 0$ or $l = 1$) and the corresponding sets of clauses consist only in 1-literal clauses ($C_0 := -y_i^r[0]$ and $C_1 := y_i^r[0]$). Let us notice that the function $parity(1,1)$ could also be represented as an exhaustive enumeration of all impossible values y_i^r corresponding to each leakage values l. However, it seems that the information added to the formula is easier to exploit when it

---

[2] By "equivalent representation", we mean two CNF formulas that have the same set of valid assignments.

is a compact representation and not a full enumeration; $C_0 := -y_i^r[0]$ and $C_1 := y_i^r[0]$ are the optimal representations of $l = 0$ and $l = 1$ for $parity(1,1)$.

It turns out that every leakage function can be compacted, to a certain extent. The leakage functions with a simple algebraic representation ($parity$, W_H) benefit the most from the compact representation, but even for random functions ($rand$, $parity - rand$) one can find more compact representations than the exhaustive enumeration. Ideally, we need a method to produce the optimal sets of clauses describing a leakage function. It is not evident of what is an optimal set of clause; we decided to optimize the representation in regard to the average number of literals per clause, denoted as $\overline{n_{lit}}$. An optimal set of clause is thus a set of clause with the lower $\overline{n_{lit}}$ possible. We apply the following heuristic to build the different formulas C_l:

1. Build the set of all 6560 possible clauses from a set of 8 variables.
2. C_l is empty.
3. Each clauses are tested succesively from the shortest to the longest: a clause c is added to C_l if (1) the clause c is true for every value y_i^r associated to the leakage l, and (2) adding the clause is helpful, that is the formula $C_l \wedge c$ has less satisfiable assignments than C_l.

The previous heuristic does not always give the optimal representation, as a variation in the order in which the clauses are tested can lead to a different formula C_l which can be slightly longer than the optimal representation. However, it seems that this heuristic is good enough for our attacks. As a comparison, we performed two attacks against a block cipher PRESENT implemented on a simulated device with a Hamming weight leakage function ($\mathsf{L}(y_i^r) = \sum_j y_i^r[j]$). The side-channel information is expressed in the first case as an enumeration (beween 186 and 255 clauses with 8 literals per clause), and in the second case as a compact set of clauses derived from the simple algebraic relation of the Hamming weight (between 8 and 112 clauses with 1 to 8 literals per clause). The median solving time for the first case is 2.95 seconds whereas the median solving time is only 0.44 second for the second case.

5   Metrics for the information leakage

In the previous section we investigated the representation dependence. In this section, we define some metrics to evaluate the leakage dependence. More specifically, we present two metrics. The first one is an information theoretic metric. Indeed, to evaluate the leakage of information from the device, an information theoretic metric is a natural candidate. Moreover, as demonstrated in [18], such an information theoretic metric can be related to the asymptotic security of an implementation in a standard DPA attack. In the following, we aim to evaluate if such an intuition holds in ASCA. Our second metric is related to the representation problem. We show that these two metrics are complementary and related in certain aspects. We finish the section by presenting an alternative metric, the algebraic immunity.

The information theoretic metric used in [18] measures the information between the observed leakage l with the target subkey s. In our case, knowing exactly the target

8

subkey s is not equivalent to a successful attack; we thus want to go a step further. We then consider the amount of information that the observed leakage l gives about the target subkey s along with the amount of information that the subkey s gives about the intermediate value y_i^r. In our setup it is easy to combine these two values, as we assume perfect subkey recovery phases with 100% accuracy. We can thus say that the mutual information between L and Y_i^r is the same as the mutual information between S and Y_i^r. We use as information theoretic metric this mutual information: $I(Y_i^r; L)$.

Let Y_i^r be a variable corresponding to one specific internal data transiting on the bus of the device (the ith result processed during the rth round) and y_i^r is a realization of this variable. L is the variable corresponding to the side-channel observation and l is a realization. The mutual information between Y_i^r and L is defined as:

$$I(Y_i^r; L) = \sum_{y_i^r, l} \Pr\left[y_i^r, l\right] \log_2\left(\frac{\Pr\left[y_i^r, l\right]}{\Pr\left[y_i^r\right].\Pr\left[l\right]}\right). \tag{1}$$

This metric tells us how much the incertitude on the variable Y_i^r is reduced if we know the variable L. This seems a useful metric; intuitively, a leakage function that gives a lot of information on the values transiting on the bus of the device should provide an easier to solve SAT problem.

Let us notice that knowing the amount of information given by the leakage function on the byte y_i^r is useful, but not enough to fully characterize the leakage function. For instance, the distribution of this information over the 8 bits of y_i^r could be also an important factor for the success of the computation phase. For example, let L_1 be a leakage function defined by $\mathsf{L}_1(y_i^r) = y_i^r[0]$ (with $y_i^r[j]$ the jth bit of the 8-bit value y_i^r, and a realization of the variable $Y_i^r[j]$) and let L_2 be a leakage function defined as $\mathsf{L}_2(y_i^r) = \sum_j y_i^r[j]$ (that is the classical Hamming weight function). As it can be observed in table 1, L_1 concentrates all its information on the first bit of y_i^r, while L_2 spreads its information evenly over the 8 bits of y_i^r.

Leakage funcion	$I(Y_i^r; L)$	$I(Y_i^r[0]; L)$	$I(Y_i^r[1]; L)$	...	$I(Y_i^r[7]; L)$
L_1	1	1	0	...	0
L_2	2.544	0.0976	0.0976	...	0.0976

Table 1. Comparison of two leakage functions.

In the end, we could also consider the mutual information between L and subsets of 1,2,...,8 bits of Y_i^r ($Y_i^r[j]$, $Y_i^r[j,k]$, ... with $0 \le j < k \le 7$). Each of these values could be important to evaluate the success of the computation phase. However, we choose to consider only the value $I(Y_i^r; L)$ as our information theoretic metric.

Our second metric is related to the CNF representation of the leakage function. When we defined the optimal representation, we used the average number of literals per clause $\overline{n_{lit}}$. We thus consider this as the second metric of the leakage function. The mutual

information on the whole byte $I(Y_i^r; L)$ and the average number of literals per clause are complementary in order to characterize a leakage function: the two metrics give a different input on the same leakage function. This complementarity is illustrated in table 2. But the two metrics are also somewhat related. More precisely, the average number of literals per clause $\overline{n_{lit}}$ of the optimal compact representation is related to the mutual information between the leakage function and the sets of 1,2,...,8 bits of y_i^r. Typically, a lot of information about a set of n bits means that the optimal representation will probably contain clauses of n or less literals. For example, the leakage functions $parity(n, m)$ are translated into clauses of n literals.

Leakage function L	$\overline{n_{lit}}$	$I(Y_i^r; L)$
$parity(1, 1)$	1	1
$parity(2, 1)$	1	2
$parity(1, 2)$	2	1

Table 2. Comparison of three different leakage functions. Both the average number of literals in a clause and the mutual information on the whole byte are useful to distinguish these leakage functions.

Independently of the success rates, the different leakage functions can be compared on the basis of the mutual information $I(Y_i^r; L)$ and the average number of literals in a clause $\overline{n_{lit}}$ (see figure 3). We observe that $\overline{n_{lit}}$ can be in a linear relation with $I(Y_i^r; L)$, in the case of the random leakage functions ($rand$, $parity - rand$) or quite uncorrelated from $I(Y_i^r; L)$ ($linear$ and $parity$). Indeed, a random leakage function is essentially characterized by its number of different leaked values n_l, but a leakage function with a specific algebraic structure cannot be reduced to this single value.
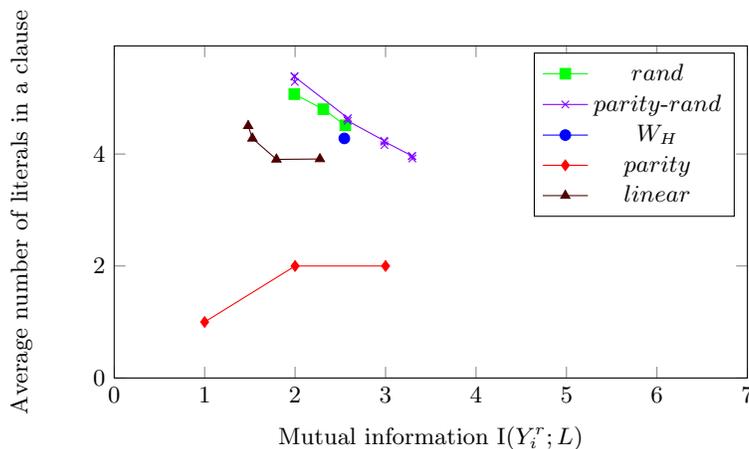
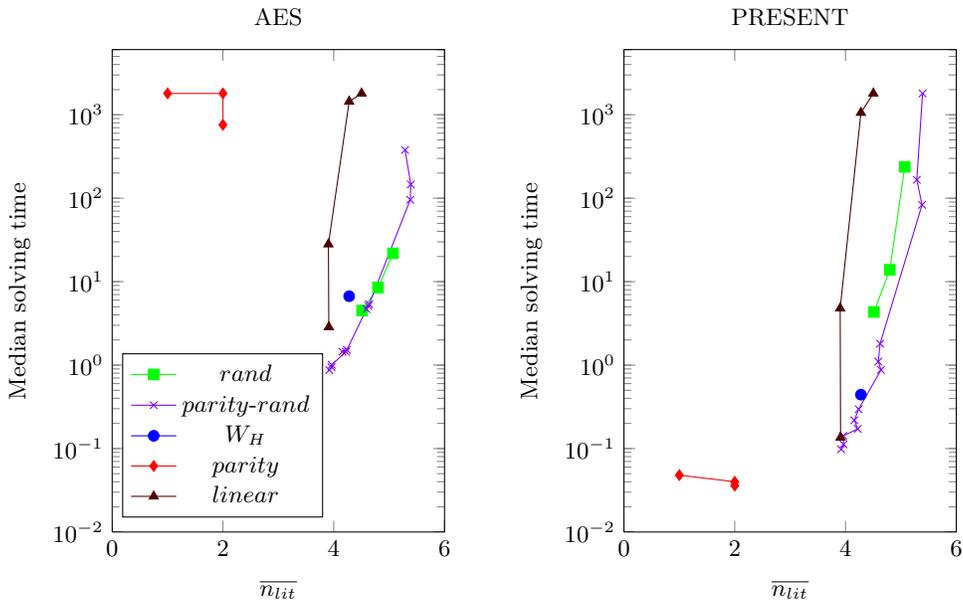**Fig. 3.** Average number of literals per clause in function of the mutual information.

Other metrics could be considered in order to evaluate the complexity of a leakage function, but none is expected to give a perfectly accurate relation between the leakage function and the median solving time. We would like to mention the *algebraic immunity* $\mathcal{AI}$. $\mathcal{AI}$ is a concept introduced to evaluate the resistance of stream ciphers to algebraic attacks (see for example [1]). The concept was extended in [8] to block ciphers, where the function f considered is the S-box of the block cipher. In our case, the $\mathcal{AI}$ of the leakage function L could be computed in order to find a measure of the resistance of the implementation to the ASCA. However, it seems that the $\mathcal{AI}$ is not very helpful in our case: the $\mathcal{AI}$ is a discrete criterion and it turns out that many very different leakage functions share the same $\mathcal{AI}$ value. Some results using the algebraic immunity as a metric are presented in appendix C.

6 Impact of the leakage function and block cipher
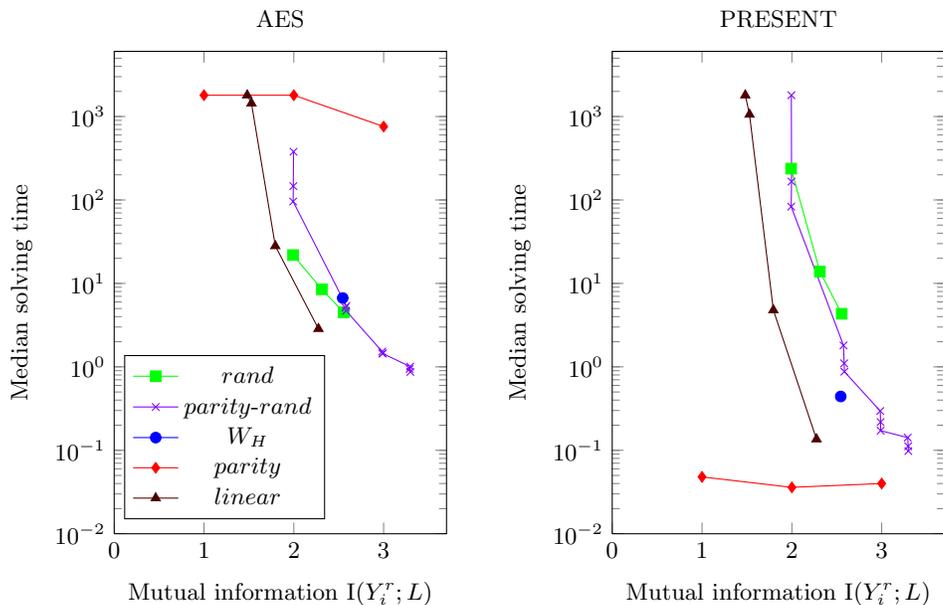
Fig. 4. Median solving time in function of the mutual information $\mathrm{I}(Y_i^r; L)$. Left: attack against the AES, right: attack against PRESENT.

In the two previous sections, we investigated the representation problem and the different metrics for the information leakage. In this section, we introduce the third parameter: the cipher dependence. As announced in section 3, we compare two block ciphers: PRESENT and the AES. This comparison focuses on the number of leakages required to perform an ASCA and on the impact of the leakage function.

Fig. 5. Median solving time in function of the average number of literals in a clause $\overline{n_{lit}}$. Left: attack against the AES, right: attack against PRESENT.

The first point of comparison between PRESENT and the AES is the number of leakages required in order to perform a successful attack. Indeed, the ASCA assumes that the adversary is able to recover the correct value for an important fraction of the total number of available leakages. Comparing the resuls from [14] and [15] (see table 3), we observe that an ASCA against the AES requires a lot more leakages than against PRESENT (in this case, the leakage model is the Hamming weight model).

	PRESENT	AES
Consecutive known W_H	64 known W_H	252 known W_H
Random known W_H	224 known W_H	>756 known W_H
Total number of W_H	496 W_H	788 W_H

Table 3. Required number of known Hamming weights in order to perform an ASCA with about 90% success rate. The detailed setups for this results are found in [14, 15].

The second point of comparison is the leakage function (see figures 4 and 5; each dot is an averaging over 100 experiments). Now we assume that the adversary recovers all the leaked values; let us notice that such an adversary recovers more side-channel leakages from the AES than from PRESENT.

The first observation we make is that, except for the short times (< 2 seconds), the solving times are similar for PRESENT and the AES. For example, the instance of the $rand(6)$ leakage function gives a median solving time of 4.94 s for the AES and 4.33 s for

PRESENT. This means that, in this case (all leakages are known), the attack is nearly independant from the target algorithm. On the other hand, the attack is dependent on the implementation of the cipher. For example, an AES implemented with less available leakages is much harder to break (see [15]).

Our second observation is about the link between the success rate and our 2 metrics (information theoretic metric $I(Y_i^r; L)$ and averag number of literals per clause $\overline{n_{lit}}$). We notice that there is a relation between the metrics and the success rate. This relation, for most of the leakage function families, is similar for PRESENT and the AES. Again, the impact of the leakage function is independent of the target algorithm. There are exceptions: namely, the *parity* family proves to be a very different challenge depending on the algorithm. The function $parity(1,1)$, for example, gives a median solving time of 0.05 s when attacking PRESENT, but over 1800 s when attacking the AES.

The big difference between PRESENT and the AES in the last example can be explained if we take a closer look at the key schedules of both algorithms (see appendix B for illustrations). Knowing the first bits of every values processed by the device means that the adversary knows some key bits at each round. In the AES, it turns out that the 16 known key bits are always lost for the next round. On the other hand, in PRESENT, the 2 known key bits are very often conserved from round to round (they are only lost when they go through the key schedule substitution, but this substitution takes no more than 4 key bits per round). That means that after several round, a big amount of the key is known to the adversary, just by looking at the key schedule. From this example, we conclude that part of the variability of our results is due to unexpected interactions between the algorithm and the leakage function. As a corroborative indication, let us notice that the *rand* leakage functions, that present no specific algebraic structure, don't give noticeably worse times for the AES than for PRESENT.

7   Conclusion

In this paper, we studied the representation, leakage and cipher dependencies in Algebraic Side-Channel Attacks. Our goal was to observe the impact of these parameters on the success rate of the ASCA, and to provide general intuitions for the security of leaking devices.

As our results show it, these dependencies are real. Namely, a compact representation has a positive impact on the attack, same as a high quantity of information between the leakage and the value processed by the device. We observed that the cipher dependencies is really preponderant only in "extreme" cases, that is lot of missing information or specific interactions between the leakage function and the algorithm. When the adversary can recover all available side-channel information, the preponderant factor becomes the implementation rather than the algorithm. In general, coutermeasures like increasing the size of the bus of the device or reducing the number of clock cycles of the implementation are always effective.

For this work, we used a SAT solver in order to deduce the secret key from the system of equations. Due to this solving strategy, our work is empirical (using different solvers

can give very different computation times for the same problem). An interesting extension would be to use a more systematic solving strategy in order to produce sound explanations of the observed phenomenons. For this purpose methods like Gröbner bases [7] seems promising.

References

1. F. Armknecht, C. Carlet, P. Gaborit, S. Knzli, W. Meier, O. Ruatta, *Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks*, in the proceedings of EUROCRYPT 2006, LNCS, vol. 4004, pp. 147-164, St. Petersburg, Russia, May - June 2006.
2. FIPS 197, *"Advanced Encryption Standard,"* Federal Information Processing Standard, NIST, U.S. Dept. of Commerce, November 26, 2001.
3. E. Brier, C. Clavier, F. Olivier, *Correlation Power Analysis with a Leakage Model*, in the proceedings of CHES 2004, LNCS, vol. 3156, pp. 135-152, Cambridge, MA, USA, August 2004.
4. A. Bogdanov, I. Kizhvatov, A. Pyshkin, *Algebraic Methods in Side-Channel Collision Attacks and Practical Collision Detection*, in the proceedings of INDOCRYPT 2008, LNCS, vol. 5365, pp. 251-265, Kharagpur, India, December 2008.
5. N. Courtois, J. Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, in the proceedings of Asiacrypt 2002, LNCS, vol. 2501, pp. 267-287, Queenstown, New Zealand, December, 2002.
6. I. Dinur, A. Shamir, *Side Channel Cube Attacks on Block Ciphers*, available online on the Cryptology ePrint Archive, Report 2009/127, http://eprint.iacr.org/2009/127.
7. J.-C. Faugère, *Groebner Bases. Applications in Cryptology*, FSE 2007 Invited Talk, available at: http://fse2007.uni.lu/slides/faugere.pdf.
8. S. Fischer, W. Meier, *Algebraic Immunity of S-Boxes and Augmented Functions*, in the proceedings of FSE 2007, LNCS, vol. 4593, pp. 366-381, Luxembourg, Luxembourg, March 2007.
9. P. Kocher, J. Jaffe, B. Jun, *Differential Power Analysis*, in the proceedings of Advances in Cryptology - CRYPTO '99, LNCS, vol. 1666, pp. 388-397, Santa Barbara, California, USA, August 1999.
10. H. Ledig, F. Muller, F. Valette, *Enhancing Collision Attacks*, in the proceedings of CHES 2004, LNCS, vol. 3156, pp. 176-190, Cambridge, MA, USA, August 2004.
11. S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks*, Springer, 2007.
12. E. Oswald, K. Schramm, *An Efficient Masking Scheme for AES Software Implementations*, in the proceedings of WISA 2005, Lecture Notes in Computer Science, vol 3786, pp 292-305, Jeju Island, Korea, August 2005.
13. A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, C. Vikkelsoe, *PRESENT: An Ultra-Lightweight Block Cipher*, in the proceedings of CHES 2007, LNCS, vol. 4727, pp. 450-466, Vienna, Austria, September 2007.
14. M. Renauld, F.-X. Standaert, *Algebraic Side-Channel Attacks*, in the proceedings of IN-SCRYPT 2009, to appear.
15. M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon, *Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA*, in the proceedings of CHES 2009, LNCS, vol. 5747, pp. 97-111, Lausanne, Switzerland, September, 2009.
16. W. Schindler, K. Lemke, C. Paar, *A Stochastic Model for Differential Side-Channel Cryptanalysis*, CHES 2005, Lecture Notes in Computer Science, vol 3659, pp 30-46, Edinburgh, Scotland, September 2005.

17. M. Soos, K. Nohl, C. Castelluccia, *Extending SAT Solvers to Cryptographic Problems*, in the proceedings of SAT 2009, LNCS, vol. 5584, pp. 244-257, Swansea, UK, June 2009.
18. F.-X. Standaert, T.G. Malkin, M. Yung, *A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks*, in the proceedings of Eurocrypt 2009, LNCS, vol. 5479, pp. 443-461, Cologne, Germany, April 2009.
19. http://www.princeton.edu/∽chaff/.

A Leakage functions used

For our experiments, we used the following leakage function families:

- $rand(4)$, $rand(5)$ and $rand(6)$: the leakage corresponding to each value y_i^r is chosen, for each simulated device, at random between 4, 5 or 6 different values.
- $parity(1,1)$, $parity(2,2)$ and $parity(3,2)$: the first leakage function is equivalent to knowing the first bit of each byte y_i^r processed by the device (the "single bit" leakage function). For the second (third) leakage function, we know the value of the parity bit for the first two (three) groups of 2 bits of y_i^r.
- $parity - rand(n, m)$, with $n = \{6, 7, 8\}$ and $m = \{2, 3, 4, 5\}$.
- W_H: the classical Hamming weight function on 8 bits.
- $linear(3)$, $linear(4)$, $linear(5)$ and $linear(6)$.

Let us notice that the generation of leakage functions from families $rand$, $parity - rand$ and $linear$ involve some random value; each leakage function generated from the same family with the same parameters values can be different (but with some general properties shared with all functions from this family).

B Interactions between the "single bit" leakage function and the key schedule

Figure 6 illustrates the interaction between the "single bit" leakage function and the key schedule of PRESENT. The "single bit" leakage model assumes that the adversary knows the value of the first bit of each 8-bit data processed by the device. With such a model and a known plaintext, the adversary knows 8 bits from the subkey of the first round. In the subsequent rounds, the adversary learns 2 additional subky bits per round. Even if some of the known subkey bits at round i can be lost at round $i + 1$ due to the S-box used in the key schedule, the adversary is still able to learn a big portion of the key, without even attempting to solve the algebraic system.

Figure 7 shows the same situation but applied to AES key schedule. In this case, the adversary knows 16 subkey bits at the first round (assuming a known plaintext). Each following round brings 16 new known bits. The main difference between PRESENT and the AES in this regard is that, here, the 16 previously known bits are lost. Thus, the total number of known bits per round stays the same. Moreover, only the first 4 bits are really new; the remaining 12 ones can be computed from the first 4 ones and the 16 bits known at the previous round.

C Results using an algebraic immunity-based metric

The algebraic immunity of an S-box is defined in [8] as follows:
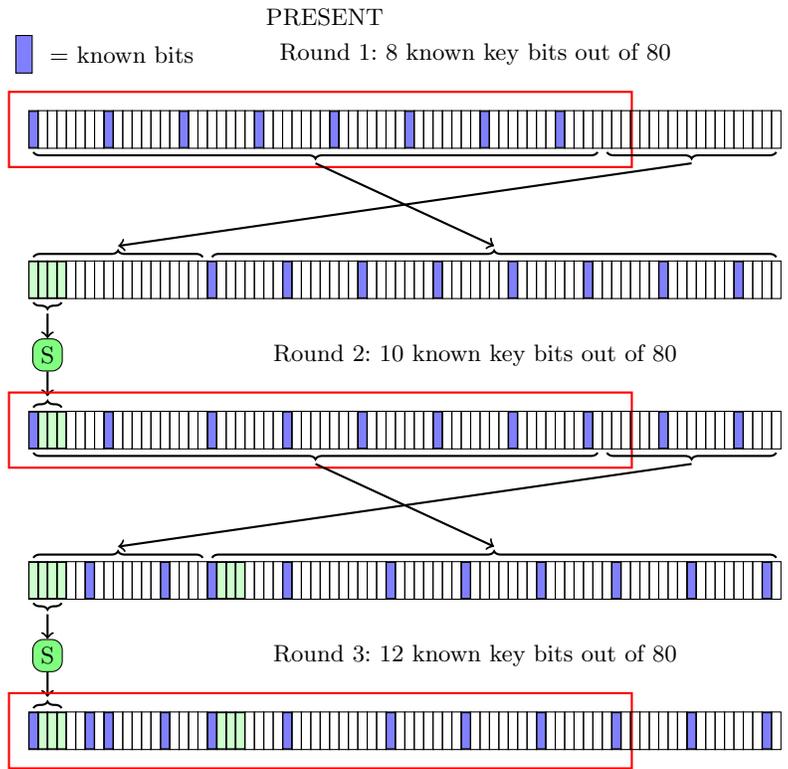
**Fig. 6.** PRESENT key schedule (omitting the constant addition).
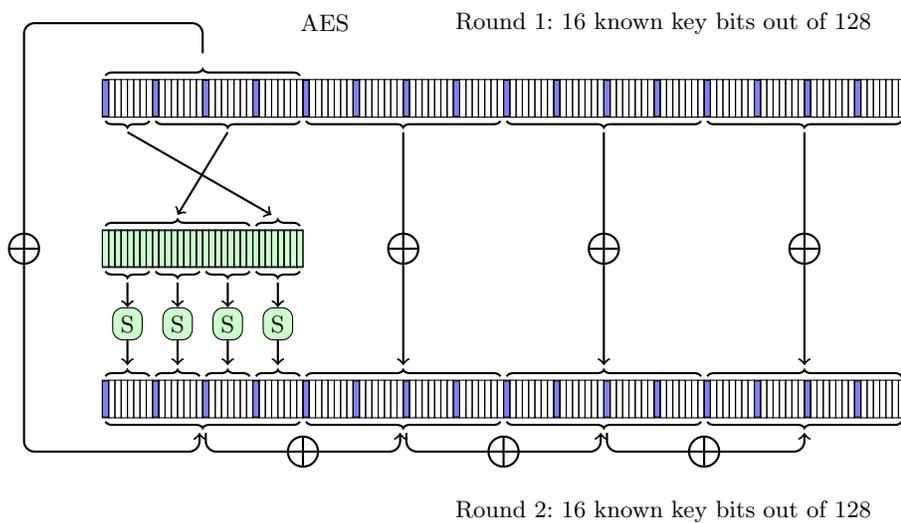
Fig. 7. AES key schedule.

Definition 1. *Consider an S-box $S : \mathbb{F}^n \to \mathbb{F}^m$. Given some fixed output y, let d be the minimum degree of a non-zero conditional equation $F_y(x)$ which holds for every $x \in S^{-1}(y)$. Then the algebraic immunity $\mathcal{AI}$ of S is defined by the minimum of d over all $y \in \mathbb{F}^m$.*

We wanted to extend this discrete value in order to take in account all possible leaked values. We define thus the *weighted algebraic immunity* as:

Definition 2. *Consider an S-box $S : \mathbb{F}^n \to \mathbb{F}^m$. Given some fixed output y, let d_y be the minimum degree of a non-zero conditional equation $F_y(x)$ which holds for every $x \in S^{-1}(y)$. Then the weighted algebraic immunity $w\mathcal{AI}$ of S is defined by the average value of d_y over all $y \in \mathbb{F}^m$, weighted by the probability of getting the output y: $w\mathcal{AI} = \sum_{y \in \mathbb{F}^m} \frac{|S^{-1}(y)|}{|\mathbb{F}^m|} d_y$, where $|s|$ is the cardinality of set s. ($|S^{-1}(y)|$ is the cardinality of the set of preimages of y for example).*

Figure 8 presents the success of the attack (measured by the median solving time) in function of the weighted algebraic immunity of the leakage function used.
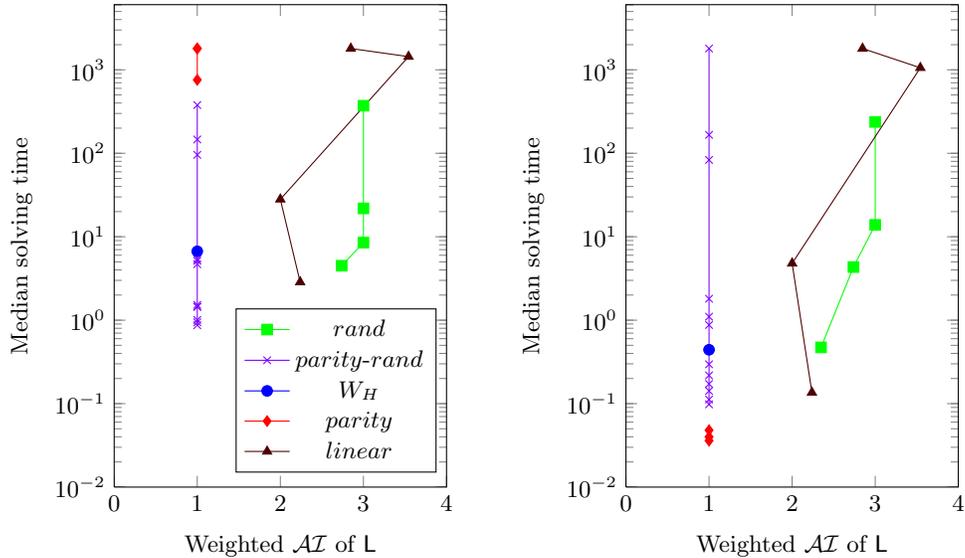
Fig. 8. Median solving time in function of the weighted algebraic immunity of the leakage function. Left: attack against the AES, right: attack against PRESENT.