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. In 2002, algebraic attacks using overdefined systems of equations have been proposed as a potentially very powerful cryptanalysis technique against block ciphers. However, although a number of convincing experiments have been performed against certain reduced algorithms, it is not clear whether these attacks can be successfully applied in general and to a large class of ciphers. In this paper, we show that algebraic techniques can be combined with side-channel attacks in a very effective and natural fashion. As an illustration, we apply them to the block cipher PRESENT that is a stimulating first target, due to its simple algebraic structure. The proposed attacks have a number of interesting features: (1) they exploit the information leakages of all the cipher rounds, (2) in common implementation contexts (*e.g.* assuming a Hamming weight leakage model), they recover the block cipher keys after the observation of a single encryption, (3) these attacks can succeed in an unknown-plaintext/ciphertext adversarial scenario and (4) they directly defeat countermeasures such as boolean masking. Eventually, we argue that algebraic side-channel attacks can take advantage of any kind of physical leakage, leading to a new tradeoff between the robustness and informativeness of the side-channel information extraction.

1 Introduction

In classical cryptanalysis against block ciphers, an adversary is usually provided with the inputs/outputs of a target algorithm. Side-channel attacks additionally provide him with some partial information on the cipher intermediate values, leaked by a device performing a cryptographic computation. Such attacks are therefore much less general - since they are specific to a given implementation - but often much more powerful than classical cryptanalysis. Hence they are considered very seriously by cryptographic devices (*e.g.* smart cards) manufacturers. Following the publication of the first Differential Power Analysis (DPA) in the late nineties [14], various types of side-channel attacks have been proposed in order to carry out effective key recoveries (see, *e.g.* [17] for a survey). Most of these techniques share a divide-and-conquer strategy in which different parts of a target key (*e.g.* physical bytes, typically) are recovered separately. They also generally exploit the leakages corresponding to the first (or last) rounds of a block cipher, where the diffusion is sufficiently weak for some parts of the intermediate key-dependent computations to be easily enumerated and predicted.

⋆ Work supported in part by the Walloon Region research project SCEPTIC.
⋆⋆ Associate researcher of the Belgian Fund for Scientific Research (FNRS - F.R.S.).

As a matter of fact, these side-channel attacks are quite demanding in leaked information since they use physical measurements to identify key bytes exactly. Also, they usually do not exploit particular weaknesses of the block ciphers. Therefore, an intriguing question is to know if an adversary could use side-channels to recover simple targets rather than exact key byte values and then use this partial information in a more elaborated offline cryptanalysis step. In other words, can we stop measuring earlier and still have the complexity of a key recovery that does not grow exponentially with the key size?

In this paper, we answer this question positively and show that combining powerful (template-like) side-channel attacks with algebraic cryptanalysis allows performing key recoveries with extremely restricted access to the target devices. Still, the attack is general and can work in a flexible manner that includes the two following phases. First, the adversary selects as many intermediate computations in the target algorithm as possible and measures their physical leakage. For each of these intermediate computations, he recovers some partial information. This partial information can be represented by a surjective function of which the output has been recovered thanks to a side-channel attack. As a typical example, if a device leaks an information that is strongly correlated with the Hamming weight of the target intermediate computations results, this function could be the Hamming weight function. But any other type of function (and hence leakage model) could be considered. It is eventually the adversary's choice to select a target that is both informative and robust. At the extremes, a bijective function is the most informative but recovering its output by a side-channel attack may require several measurements - and a surjective function with only one possible output value yields no information at all. Then, during a second (offline) phase, the adversary exploits this partial information on the algorithm intermediate values with an algebraic attack. That is, he writes the block cipher as a system of quadratic (or cubic, ...) equations and adds the previously defined functions with known outputs to the system. In practice, the approach we follow in this paper is to convert the system of equations representing the block cipher into a SAT problem and to use an automated solver to perform the key recoveries. It turns out that this solution yielded very good results. However, it remains an open question to determine better techniques for this purpose.

The proposed attacks differ from most previously known side-channel attacks in a number of interesting aspects. First, they potentially exploit the leakage of all the cipher rounds (classical DPA generally exploits the first or last rounds only). Second, they can succeed in an unknown plaintext/ciphertext adversarial context (classical DPA usually requires the knowledge of either the plaintexts or the ciphertexts). Third, they require much less observations to succeed. In certain reasonable implementation contexts, we show that the leakage trace of a single encryption can be sufficient to perform a complete key recovery. This implies that constructions based on re-keying strategies such as [20, 21] can sometimes be broken in practice. Eventually, they can deal with block ciphers protected with a masking countermeasure, *e.g.* [12]. In particular, we show experiments that break such masked designs with a single trace, nearly as easily as unprotected ones.

In summary, classical side-channel attacks can be viewed as a combination of two sub-problems: (1) "how to efficiently recover partial information on certain parts of a cipher state?" and (2) "how to efficiently exploit this partial information?". Algebraic side-channel attacks raise a new question, namely: "which partial information should we try to recover?". It relates both to the previously mentioned tradeoff between robustness and informativeness and to the selection of the best target key classes mentioned as an open question in [28].

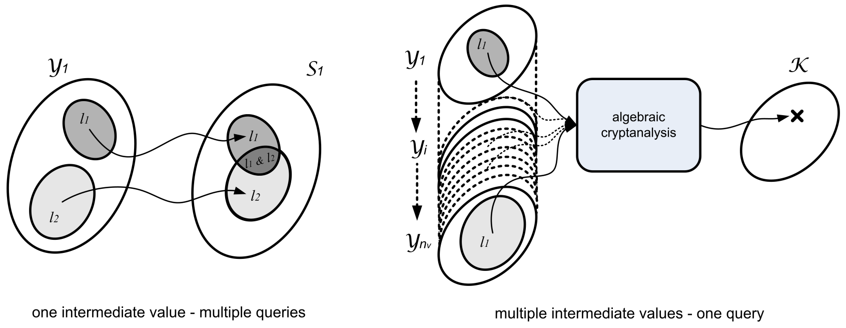

Fig. 1. Standard DPA versus algebraic side-channel attacks.

More precisely, the difference between algebraic side-channel attacks and standard DPA attacks is illustrated in Figure 1. As already mentioned, standard DPA attacks exploits a divide-and-conquer strategy and recover several pieces of a secret key independently. For example, in the left part of the figure, the set $\mathcal{S}_1$ typically contains the 256 candidates for a key byte. In order to recover the correct one, the adversary targets a single intermediate value (in the set $\mathcal{Y}_1$). Typically, it could be the output of an S-box in the first cipher round. Each leakage trace l_i provides him with information about this intermediate value that is then "translated" into subkey information. By combining the leakage corresponding to several plaintexts (*i.e.* by increasing the data complexity q), he finally identifies the key byte exactly. By contrast, an algebraic side-channel attack aims to limit the data complexity to $q = 1$ and exploits several (n_v) intermediate values within a single leakage trace. This information is then combined in an offline cryptanalysis step in order to recover the master key at once. Note that the data complexity is not always equivalent to the number of measurements since the same leakage trace can be measured several (n_r) times. But the data complexity is the most relevant quantity to compare from a cryptanalytic point of view, in particular regarding constructions such as [20, 21].

Related works. The following results can be related to three different lines of research. First, they aim to recover partial information from a leaking device in the most efficient way. They consequently exploit techniques such as template attacks (*e.g.* [9, 29]) and stochastic models [25]. Second, they take advantage of algebraic cryptanalysis in the black box setting, introduced by Courtois and

Pieprzyk in [10]. In particular, we exploit solutions based on SAT solvers as described in [1, 11]. Eventually, several other papers suggested to combine side-channel attacks with classical cryptanalysis. The most studied problem is probably the one of collision-based side-channel attacks, detailed *e.g.* in [15, 26, 27]. Techniques borrowed from square attacks [7] and differential cryptanalysis [13] against block ciphers have also been proposed in 2005 and 2006, respectively. More recently, impossible and multiset collision attacks were presented at CHES 2007 [3]. All these attacks have objectives similar to ours. They usually try to exploit the information leakages for more than the first block cipher rounds with advanced cryptanalysis. The goal is to break implementations for which only those rounds would be protected against side-channel attacks or to reduce the number of measurements required to perform a key recovery. We finally mention the recent and very efficient collision-based attacks of [6] that also use algebraic techniques and therefore closely connect to the present paper. In fact, our proposed cryptanalysis can be viewed as a generalization of such collision-based attacks. We similarly aim to reduce the data complexity ([6] found $4 \leq q \leq 20$, we claim $q = 1$). The main difference is that we are not restricted to one particular type of information (*i.e.* collisions) and are not limited to the exploitation of the first/last rounds of a block cipher. In principle, our algebraic attacks can take advantage of any information leakage, from any part of a cryptographic computation. A consequence is that they can be easily extended to protected implementations (*e.g.* masked), contrary to collision-based ones [5].

2 Target cipher

PRESENT is a Substitution-Permutation Network with a block size of 64 bits [4]. The recommended key size is 80 bits, which should be sufficient for the expected applications of the cipher. However a 128-bit key-scheduling is also proposed. The encryption is composed of 31 rounds. Each of the 31 rounds consists of a XOR operation to introduce a round key K_i for $1 \leq i \leq 32$, where K_{32} is used for post-whitening, a linear bitwise permutation and a non-linear substitution layer. The non-linear layer uses a single 4-bit S-box which is applied 16 times in parallel in each round. The cipher is described in pseudo-code in Appendix A.

3 Offline phase: algebraic attack

3.1 Deriving the system of equations

The first step in an algebraic cryptanalysis is to describe the target cryptosystem with a set of polynomial equations involving the key bits as variables. Given such a representation, exposing the secret key is equivalent to solving the system of equations. For this purpose, we will denote the bits of the plaintext, ciphertext and key as P_i, C_i and K_i and look for a set of equations involving these variables and describing the cryptosystem PRESENT. The most obvious solution would be to build a system of equations of the form:

$$C_1 = f_1(P_1, ..., P_{64}, K_1, ..., K_{80})$$
$$C_2 = f_2(P_1, ..., P_{64}, K_1, ..., K_{80})$$

$$C_{64} = f_{64}(P_1, ..., P_{64}, K_1, ..., K_{80})$$

However, this kind of representation is quite useless for practical attacks. Due to the diffusion in the cryptosystem, each equation would involve every single bit of the plaintext and the key. Due to the 31 successive rounds of non-linear substitutions, these equations would also include a lot of high degree monomials. In order to avoid such limitations, the idea developed by Courtois and Pieprzyk in [10] is to introduce new internal variables in order to work with a large number of small, low degree polynomial equations instead of a small number of huge, high degree equations. For PRESENT, we decided to add three groups of variables:

- one variable x_i for each input bit of each S-box in the cryptosystem,
- one variable y_i for each output bit of each S-box in the cryptosystem,
- one variable k_i for each bit of each sub-key.

In practice, the substitutions are the only non-linear elements of PRESENT and are therefore the most challenging parts of the cipher to reduce to low degree equations. Fortunately, it has been shown in [2] that for small S-boxes, such equations can be constructed in a simple and systematic manner. As an illustration, the construction of a system of low degree equations for a 3-bit S-box is described in Appendix B. Extending this technique to the complete 31-round PRESENT, we can build a system of approximately 40 000 equations in 7000 variables (50 000 monomials). The most interesting characteristic of this system is its sparsity. If we represent such a system like a matrix, a line being an equation, a column being a monomial, the proportion of non-null elements is very low (approximately 0.0155%). Using a compact representation of the system matrix consequently improves the attack performances considerably.

3.2 Conversion to a SAT problem

Bard *et. al* showed in [1] how to reduce a system of equations to a SAT problem. Most SAT solvers take a formula in *conjunctive normal form* (CNF) as input. A problem in CNF is a conjunction (AND) of clauses, each clause being a disjunction (OR) of literals. The literals are variables (x) or variable negations ($\bar{x}$).

In order to reduce our problem, there are two main steps. First, we need to translate every monomial of degree higher than 1. If we consider a monomial $x_1x_2x_3x_4$, we can turn it into a dummy variable a and a set of clauses:

$$(x_1 \vee \bar{a}) \wedge (x_2 \vee \bar{a}) \wedge (x_3 \vee \bar{a}) \wedge (x_4 \vee \bar{a}) \wedge (a \vee \bar{x_1} \vee \bar{x_2} \vee \bar{x_3} \vee \bar{x_4}), \quad (1)$$

that is equivalent to $a = x_1x_2x_3x_4$. Hence we can transform each occurrence of the monomial $x_1x_2x_3x_4$ into an occurrence of the dummy variable a in the system of equations and include the previous set of clauses in a CNF. So, for each monomial of degree $d > 1$, we introduce one dummy variable and $d + 1$ clauses. Secondly, we need to translate the exclusive disjunctions (XOR) of our original equations into conjunctions and disjunctions. Translating long XOR-equations in conjunctive normal form is problematic because the number of new clauses is exponential in the number of terms in the equation. Hence, we use again dummy variables in order to bound the number of new clauses in the formula. We transform each equation $x_1 \oplus x_2 \oplus x_3 \oplus ... \oplus x_n = 0$ into:

$$\begin{aligned} x_1 &\oplus x_2 \oplus x_3 \oplus b_1 = 0 \\ b_1 &\oplus x_4 \oplus x_5 \oplus b_2 = 0 \\ ... & \\ b_m &\oplus x_{n-1} \oplus x_n = 0 \end{aligned}$$

This way, we separate each n-term equation into an equivalent set of $m = \lceil n/2 \rceil - 1$ (for $n > 2$) 4-term equations[1], via the addition of m dummy variables. After that, each 4-term equation of the form $a \oplus b \oplus c \oplus d$ is turned into an equivalent set of 8 clauses that we add to the previously initiated CNF:

$$\begin{aligned} (\bar{a} \vee b \vee c \vee d) &\wedge (a \vee \bar{b} \vee c \vee d) \wedge (a \vee b \vee \bar{c} \vee d) \wedge (a \vee b \vee c \vee \bar{d}) \wedge \\ (\bar{a} \vee \bar{b} \vee \bar{c} \vee d) &\wedge (\bar{a} \vee \bar{b} \vee c \vee \bar{d}) \wedge (\bar{a} \vee b \vee \bar{c} \vee \bar{d}) \wedge (a \vee \bar{b} \vee \bar{c} \vee \bar{d}) \end{aligned}$$

Combining these two steps, we can build a CNF formula from our system of equations. We can estimate the number of different literals in the formula: $n_{literal} \simeq n_{mon} + n_{equ} * (\lceil n_{term}/2 \rceil - 1)$, were n_{mon} and n_{equ} are respectively the number of monomials and equations in the system and n_{term} is the average number of terms in an equation (typically quite low, because of the sparsity of the system). We can similarly estimate the number of clauses: $n_{clause} \simeq n_{mon} * (d + 1) + n_{equ} * (\lceil n_{term}/2 \rceil - 1) * 8$, where d is the average degree of the monomials appearing in the system. Interestingly, the size of the CNF and the number of literals are linearly dependent in most of the cryptosystem parameters (block and key size, number of rounds). In fact, only the size of the S-boxes has more impact, because it not only modifies n_{mon} and n_{equ} but also d and n_{term}.

3.3 Solving the system

We selected zChaff, a SAT solver that was developed by Princeton University [8] and won the 2004 SAT competition [24]. This solver is not the best SAT solver available anymore, but it works fine for our experiments. zChaff uses the Chaff algorithm, which is a refinement of the DPLL algorithm [19]. These algorithms use a recursive backtracking procedure to find an adequate solution [18]. In summary, at each step s the procedure assigns a random value (say FALSE) to a literal x_{i_s} and simplifies the formula. If no conflict (empty clause) is detected, the procedure is repeated for the next step $s+1$. If one or more conflicts are detected,

[1] Using a cutting number of 4 is arbitrary but yielded satisfactory results in our context. 5-term, 6-term, ... equations could be similarly investigated.

the procedure backtracks: the formula is restored as it was before assigning x_{i_s}. Then it assigns the opposite value (TRUE) to x_{i_s} and continues as previously. If both values (TRUE and FALSE) were already tried for x_{i_s}, the procedure backtracks to $x_{i_{s-1}}$, and so on. If the procedure assigns a value to all the literals and finds no conflicts, the problem is declared satisfiable. If the procedure must but cannot backtrack ($s = 0$), the problem is declared unsatisfiable. In practice, solving the system of equations described in Sections 3.1, 3.2 with the previous SAT solver is generally hard. Therefore, the idea we propose in this paper is to take advantage of the additional information provided by side-channel leakages on the intermediate values during the execution of PRESENT. The next section describes the online part of this attack. We explain how a leakage model that can be efficiently exploited in an algebraic cryptanalysis was selected and constructed for a given device. In addition, we discuss the generalization to other leakage models and the resulting information *vs.* robustness tradeoff in Section 6.

4 Online phase: side-channel attacks

Our experiments target an implementation of PRESENT with an 80-bit key in a PIC 16F877 8-bit RISC-based micro-controller and exploit the measurement setup described in [29]. This target device is particularly convenient for a first investigation since it typically leaks a power consumption that is strongly correlated with the Hamming weight of the data it manipulates. For example, the left part of Figure 2 illustrates the power consumption corresponding to different 8-bit values commuting on the PIC bus, having different Hamming weights: the bold traces represent the mean power consumption for a given Hamming weight between 0 and 8; the grey traces represent single measurements. This picture visually suggests that the Hamming weight of the data commuting on the bus can be recovered with very high confidence in a single trace. In practice, we used a Bayesian template attack such as described in [9] to perform a partial key recovery for which the target subkey is the Hamming weight of a data commuting on the bus. That is, we have $|\mathcal{S}| = 9$ (rather than $|\mathcal{S}| = 256$ in standard DPA attacks). In this setting, we experimented a single-byte success rate (defined in Appendix C) of 99.3%. That is, given a leakage sample corresponding to some 8-bit byte x, we can recover $W_H(x)$ with probability 0.993[2].

Importantly and contrary to most other side-channel attacks, algebraic side-channel attacks do not only exploit the leakage corresponding to one byte of the intermediate cipher state at once. On the contrary, they aim to exploit as much partial information about these intermediate values as possible. Hence, our attacks exploit a powerful profiling step such that the adversary recovers the leakages corresponding to the computation of $2 \times 8 \times 31 = 496$ bytes during the encryption of a single plaintext. Those bytes relate to the computation of the key additions (8 bytes per round) and the S-boxes (8 bytes per round) for all the 31 cipher rounds. As for any profiling step, this requires that the adversary

[2] Improved techniques such as [29] could be used in more critical contexts.

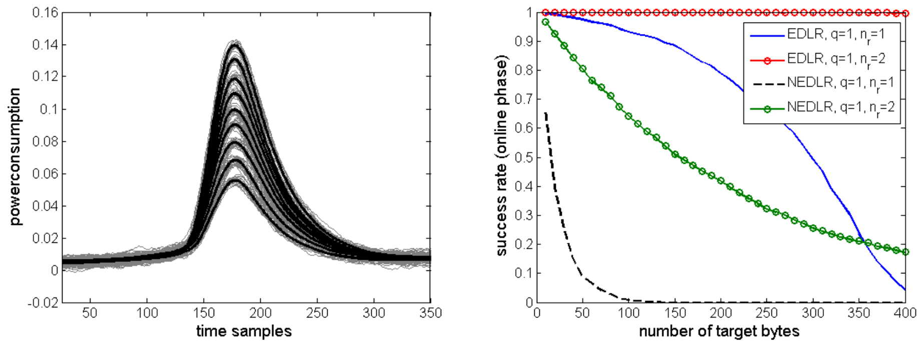

Fig. 2. Leakage traces, mean leakage traces and multiple byte success rate.

can manipulate a device with a known key prior to the attack. But once this profiling is performed, it can be re-used for as many online attacks as possible.

Because a SAT solver can hardly deal with errors in the information it is provided with, the important quantity in our attacks is not the single-byte success rate but the multiple-byte success rate. In practice, it can be improved by using simple error detection and likelihood rating techniques such as:

- Detection of impossibilities: we systematically rejected the leakages samples that give rise to incoherent inputs and outputs for the S-boxes.
- Selection of most likely Hamming weights: when using only a subset of the 496 available leakages, we used the ones with highest probabilities first.

In addition and when necessary, the success rate of our attacks can be increased by repeating a measurement for the same input plaintext, therefore keeping a constant data complexity $q = 1$. The right part of Figure 2 represents the multiple-byte success rate as a function of the number of target bytes in the implementation of PRESENT, with or without error detection and likelihood rating (EDLR), and for $q = 1$ and $n_r = 1, 2$. As a matter of fact, the complexity of this online phase depends on how many bytes of information are required to solve the system of equations given in Section 3.1, 3.2. Yet, it remains that for our target device, it is easier to recover the Hamming weight of a byte than the exact value of this byte, which is the main motivation behind our attack.

We mention that we did not consider side-channel leakages from the key scheduling of PRESENT (*i.e.* we assumed implementations with securely precomputed round keys). This allows avoiding the algebraic counterparts of simple power analyzes such as [16] that would trivially break the implementation. In other words, we focused our attention on the more challenging scenarios where only the cipher rounds are leaking. Note that although the leakage of the key scheduling algorithm is not exploited in our attacks, its algebraic description is included in the system of equations representing PRESENT.

5 Combining algebraic and side-channel attacks

Following the previous section, the partial information provided by side-channel leakages can be represented by a surjective function of which the output is known. Given a leakage model and a number of target bytes, an adversary can directly inject this additional information into the system of Section 3.1, or in its CNF representation (since the leakage information can be converted into a set of clauses). In practice, we exploited the Hamming weights of the key addition and S-box outputs in PRESENT. As mentioned in Section 2, a maximum of 496 bytes can be extracted. It corresponds to a SAT problem for a 31-round PRESENT that includes approximately 130 000 variables and 1 100 000 clauses.

5.1 First experimental results

We first applied algebraic side-channel attacks assuming that all the Hamming weights of the S-box inputs and outputs in PRESENT are correctly extracted. For comparison purposes, we attacked different reduced-round and extended-round versions of the algorithm. The results of these attacks are summarized in Figure 3 from which we obtain the following observations:

1. The success rate of these attacks equals one for all versions of PRESENT.
2. Hence, the important quantity to analyze is the resolution time which seems to follow an exponential probability distribution.
3. The average resolution time is approximately linear in the number of rounds.

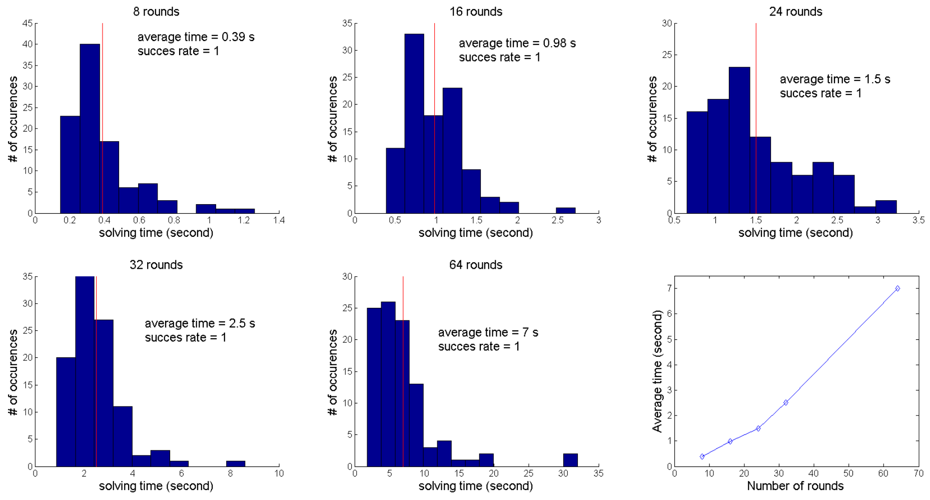

Fig. 3. 8, 16, 24, 32 and 64-round PRESENT, complete W_H leakages.

5.2 Advanced scenarios

The experiments in the previous section are quite disconnected from practical attacks since we assume that all Hamming weights are recovered. In practice, the subkey extraction may suffer from errors and consequently, only a subset of the 496 bytes Hamming weights in a 31-round PRESENT can be exploited. In this section, we consider advanced scenarios where only parts of the 496 bytes are targeted. Specifically, we consider two types of contexts: (1) consecutive weights, *i.e.* the adversary recovers a number of Hamming weights that correspond to consecutive rounds in the block cipher, starting from the middle ones and (2) random weights, *i.e.* the adversary recovers Hamming weights that correspond to random bytes in the cipher rounds. We measure the amount of information extracted in "number of rounds of W_H information", one round corresponding to the Hamming weights of 16 intermediate bytes. And we consider that an attack has failed whenever the solver has not found a solution within 3600 seconds.

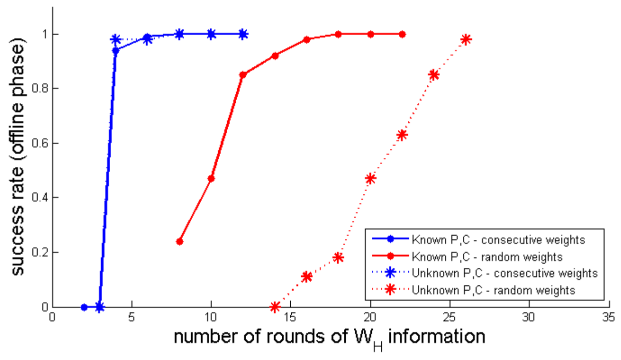

Fig. 4. 31-round PRESENT, partial W_H leakages and unknown P,C (the average solving times for these advanced scenarios are available in Appendix D).

The plain curves in Figure 4 correspond to this more realistic scenario. They illustrate that reaching high success rates is significantly more difficult with random weights. It also shows that recovering the Hamming weights of 4 consecutive rounds is generally sufficient to solve the system of equations. It is interesting to trade the effectiveness of this offline algebraic phase with the one of the online phase in Section 4. Indeed, taking advantage of the error detection and likelihood rating techniques implies a non consecutive selection of weights.

Unknown plaintexts-ciphertexts. In addition, the same figure shows the success rates when considering attacks in an unknown plainext and ciphertext adversarial scenario. Quite naturally, it does not affect the results when consecutive rounds are considered. But unknown plaintexts/ciphertexts imply the need of larger amounts of information when this information is randomly distributed over the block cipher intermediate values. It is worth noting that most side-channel attacks (*e.g.* Kocher's original DPA [14]) would fail in a similar context.

Masked implementations. We also performed experiments against an implementation protected with a masking countermeasure. In short, masking a software implementation aims to trade the efficiency and cost of this implementation for an improved security against side-channel attacks. In general, the most challenging part to mask in a block cipher is the non-linear S-box. That is, assuming a masked plaintext $p \oplus m$, we need to generate an output mask q for the S-box $\mathsf{S}(p \oplus k)$. For example, [17] describes a masked software implementation of the AES in Section 9.2.1. In order to limit the cost overheads, the same pair of masks (m, q) is used for all the AES S-boxes and in all the AES rounds (4 other masks are used in the MixColumn operation). Applying this solution to PRESENT would yield very little additional variables for our algebraic attacks. Hence, we considered a more robust countermeasure (algebraically speaking) denoted as duplication in [12] or GLUT in [22] in which a different mask is used for all the S-boxes and is propagated through the rounds as in Figure 5: an S-box S' is then used to generate the output masks q. Since PRESENT has 4-bit S-boxes, this is feasible at a reasonable cost (the memory requirements of S' equals $2^8 \times 4$).

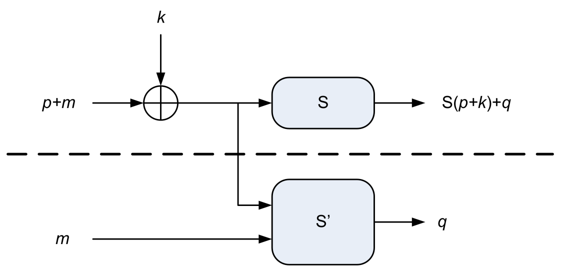

Fig. 5. Masked implementation of PRESENT.

Interestingly, two different strategies can be considered to break this masked implementation. A simple one is to just neglect the masks and solve the system as if it had unknown plaintext and ciphertext (*i.e.* to consider only the upper part of Figure 5). A more elaborated solution is to build a system of equations including the new S-boxes S' and to solve it with known plaintext and ciphertext. We performed both solutions and the results are summarized in Figure 6. These attacks show that even a masked implementation can be broken with the observation of a single encrypted plaintext in our experimental setting. They also confirm the intuition of the previous section that unknown plaintext/ciphertext only increases the difficulty of recovering the block cipher key if random weights are provided to the adversary. In this context, building a more complex system including the mask propagation leads to better results. It is worth noting that when building a complete system, more Hamming weights are also extracted per round. Eventually, we mention that refreshing the masks in every round would not improve the security either, since the combined information on $p \oplus m \oplus k$ and $p \oplus m \oplus m' \oplus k$ would allow to easily break the masking too.

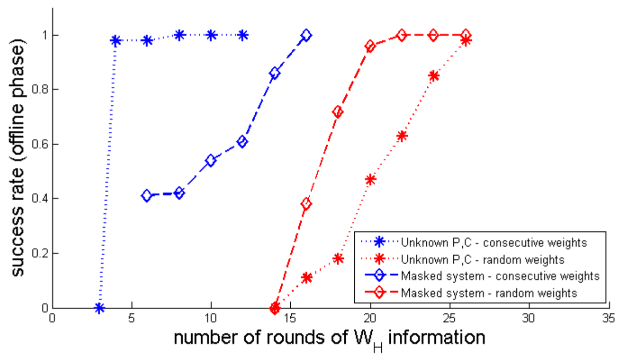

Fig. 6. Masked 31-round PRESENT, partial W_H leakages and unknown P,C.

Global success rate. Eventually, the previous success rates have been computed for the offline phase of our attacks only. But we can similarly compute the global success rate, combining the ones of the online side-channel attack and the algebraic cryptanalysis. For example, Figure 7 presents this global success rate for an unprotected implementation, in a known plaintext/ciphertext scenario and with randomly distributed leakages. It illustrates the tradeoff between the two phases of the attack. Recovering more Hamming weights increases the probability of mistake for one of them - but if all correct, they significantly facilitate the offline computations. We mention that in this figure, we assume that an incorrect Hamming weight leads to a failure. But advanced strategies could be considered, by better dealing with incorrect information (*e.g.* extracting sets of Hamming weights including the correct one rather than exact Hamming weights).

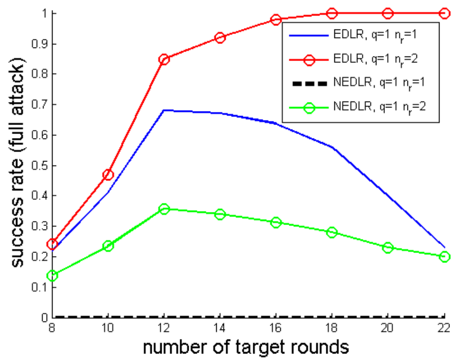

Fig. 7. Global success rate, randomly distributed W_H and known P,C.

6 Information *vs.* robustness tradeoff

Although the construction of a leakage model and the selection of a target subkey in Section 4 appears natural for our running device, they are in fact arbitrary to a certain extent. Indeed, an adversary can generally choose any leakage model as soon as this model fits reasonably well to the actual measurements. And given a leakage model, he still has to decide which subkey to recover. In the previous section and given a byte x commuting on the PIC bus, we decided to guess the value of the Hamming weight of this byte (which is motivated by reasonable physical assumptions). But another adversary could decide to guess the exact value of x (a far more challenging goal) or one bit of x. The selection of a target subkey with respect to a leakage model gives rise to a tradeoff between the robustness and informativeness of the partial key recovery that we now discuss.

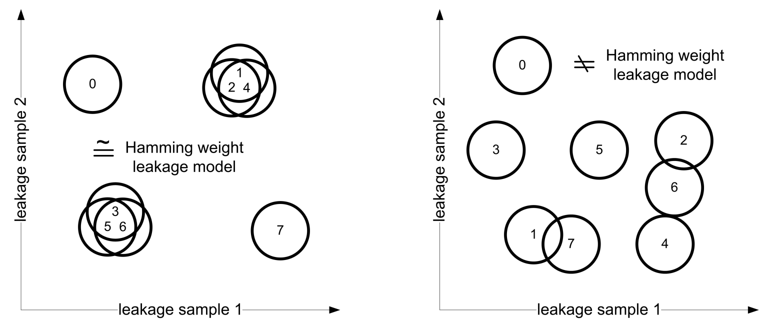
leakage sample 2
leakage sample 1
0
1
2 4
≅ Hamming weight leakage model
3
5 6
7
≠ Hamming weight leakage model
0
3
5
2
6
1
7
4

Fig. 8. Two illustrative leakage probability density functions.

As an illustration, let us imagine two leaking device with a 3-bit ALU and bus. Let us also assume that these devices leak two normally distributed samples as intuitively represented in Figure 8. That is, each of the 8 possible values that can appear on the device bus gives rise to a leakage represented by a two-dimensional Gaussian curve (*i.e.* a circle on the figure). From this example, it quite directly appears that a good hypothetical leakage model for the left leakage function would be a Hamming weight-like function. By contrast, the right leakage function does not show such Hamming weight dependencies (since values with identical Hamming weights do not give rise to similar leakages). In both cases, it is the adversary's choice to select a target subkey that is easy to recover and informative. Recovering Hamming weights is usually relevant for standard CMOS devices. But in general, one can decide to extract any type of information from the leakages. And the better the target subkey actually corresponds to the actual leakages, the more robust and efficient the subkey recoveries. Of course and on the contrary, more informative subkeys generally give rise to an easier solving for the system of equations described in Section 3.

In summary, classical DPA attacks usually directly target key bytes in block ciphers and this requires to combine the leakages corresponding to several encrypted plaintexts. Algebraic side-channel attacks allow focusing on easier targets and can exploit any information on the intermediate values of a block cipher that physical leakages may effectively determine. This information can be easily represented by a surjective function from the target intermediate value space (*i.e.* $\{0,1\}^8$ in our example) to an hypothetical leakage space (*i.e.* $\{0,1,\ldots 9\}$ in our example). It is then the goal of the SAT solver to find if the extracted information is sufficient to perform a complete key recovery from it. The answer depends on the quality of the solver, the specifications of the block cipher and the quality of its implementation. Hence, algebraic side-channel attacks raise a new research problem, namely: "what is the smallest amount of information that a side-channel attack has to provide to break a block cipher?" In Section 4, we show that finding the Hamming weights of the intermediate bytes in PRESENT is sufficient. But is is an open question to determine if other types of information (*e.g.* obtained from for dual-rail circuits) can be sufficient.

7 Conclusions and open problems

This paper introduces algebraic side-channel attacks allowing to successfully recover a block cipher key with the observation of a single side-channel trace. While most side-channel attacks can be generically applied independently of the algorithms, the proposed technique takes advantage of cryptanalytic tools that are dependent both on the amount of physical information leakage and the structure of the target cipher. It trades some of the flexibility of standard DPA for a reduced data complexity. Still, it remains very flexible, since given a system of equations describing a block cipher (or a masked block cipher), any type of physical information can in principle be exploited in a straightforward manner. These results raise two main open questions. A first one is to prevent algebraic side-channel attacks. Intuitively, this would require to increase the algebraic complexity of both the target algorithms and the information leakages. Experiments performed against the AES Rijndael (see [23]) suggest that moving to more elaborated ciphers than PRESENT is not sufficient. Hence, additionally moving from 8-bit platforms where the Hamming weight leakages are very informative to larger devices is an interesting direction for increasing the security of cryptographic devices. The impact of former countermeasures against DPA in this new context (*e.g.* time randomizations) should also be investigated. A second question is to improve and optimize the different parts of an algebraic side-channel attack. It implies to study both the offline cryptanalytic aspects and the selection of good leakage models. Properly combining these two phases would allow determining the best tradeoff between the robustness and informativeness of a side-channel attack that is discussed in Appendix 6.

References

1. G. Bard, N. Courtois, C. Jefferson, *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers*, Cryptology ePrint Archive, Report 2007/024.
2. A. Biryukov, C. De Cannière, *Block Ciphers and Systems of Quadratic Equations*, in the proceedings of FSE 2003, Lecture Notes in Computer Science, vol 2887, pp 274-289, Lund, Sweden, February 2003.
3. A. Biryukov, D. Khovratovich, *Two New Techniques of Side-Channel Cryptanalysis*, in the proceedings of CHES 2007, Lecture Notes in Computer Science, vol 4727, pp 195-208, Vienna, Austria, September 2007.
4. 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, Lecture Notes in Computer Science, vol 4727, pp 450-466, Vienna, Austria, September 2007.
5. A. Bogdanov, *Improved Side-Channel Collision Attacks on AES*, in the proceedings of SAC 2007, LNCS, vol 4876, pp 84-95, Ottawa, Canada, August 2007.
6. 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.
7. V. Carlier, H. Chabanne, E. Dottax, H. Pelletier, *Generalizing Square Attack using Side-Channels of an AES Implementation on an FPGA*, in the proceedings of FPL 2005, pp 433-437, Tampere, Finland, August 2005.
8. http://www.princeton.edu/∽chaff/.
9. S. Chari, J. Rao, P. Rohatgi, *Template Attacks*, in the proceedings of CHES 2002, LNCS, vol 2523, pp 13-28, CA, USA, August 2002.
10. 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.
11. N. Courtois, G. Bard, *Algebraic Cryptanalysis of the Data Encryption Standard*, in the proceedings of 11th IMA International Conference, Lecture Notes in Computer Science, vol 4887, pp 274-289, Cirencester, UK, December 2007.
12. L. Goubin, J. Patarin, *DES and Differential Power Analysis*, CHES 1999, LNCS, vol 1717, pp 158-172, Worcester, MA, USA, August 1999.
13. H. Handschuh, B. Preneel, *Blind Differential Cryptanalysis for Enhanced Power Attacks*, in the proceedings of SAC 2006, Lecture Notes in Computer Science, vol 4356, pp 163-173, Montreal, Canada, August 2006.
14. P. Kocher, J. Jaffe, B. Jun, *Differential Power Analysis*, in the proceedings of Crypto 1999, LNCS, vol 1666, pp 398-412, Santa-Barbara, CA, USA, August 1999.
15. 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.
16. S. Mangard, *A Simple Power Analysis (SPA) Attack on Implementations of the AES Key Expansion*, in the proceedings of ICISC 2002, Lecture Notes in Computer Science, vol 2587, pp 343-358, Seoul, Korea, December 2002.
17. S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks*, Springer, 2007.
18. D. Mitchell, *A SAT Solver Primer*, in the proceedings of EATCS Bulletin (The Logic in Computer Science Column), Vol 85, pp 112-133, February 2005.
19. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, *Chaff: Engineering an Efficient SAT Solver*, in the proceedings of DAC 2001, Las Vegas, June 2001.

20. C. Petit, F.-X. Standaert, O. Pereira, T.G. Malkin, M. Yung, *A Block Cipher based PRNG Secure Against Side-Channel Key Recovery*, In the proceedings of ASIACCS 2008, pp 56-65, Tokyo, Japan, March 2008.
21. K. Pietrzak, *A Leakage-Resilient Mode of Operation*, in the proccedings of Eurocrypt 2009, LNCS, vol 5479, pp 462-483, Cologne, Germany, April 2009.
22. E. Prouff, M. Rivain, *A Generic Method for Secure S-box Implementation*, WISA 2007, LNCS, vol 4867, pp 227-244, Jeju Island, Korea, August 2007.
23. M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon, *Algerbaic Side-Channel Attacks on the AES: Why Time also Matters in DPA*, to appear in the proceedings of CHES 2009, Lecture Notes in Computer Science, September 2009.
24. http://www.lri.fr/∽simon/contest04/results/, SAT '04 competition main page.
25. 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.
26. K. Schramm, T.J. Wollinger, C. Paar, *A New Class of Collision Attacks and Its Application to DES*, in the proceedings of FSE 2003, Lecture Notes in Computer Science, vol 2887, pp 206-222, Lund, Sweden, February 2003.
27. K. Schramm, G. Leander, P. Felke, C. Paar, *A Collision-Attack on AES: Combining Side Channel and Differential Attack*, in the proceedings of CHES 2004, LNCS, vol 3156, pp 163-175, Cambridge, MA, USA, August 2004.
28. F.-X. Standaert, T.G. Malkin, M. Yung, *A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks*, in the proccedings of Eurocrypt 2009, LNCS, vol 5479, pp 443-461, Cologne, Germany, April 2009.
29. F.-X. Standaert, C. Archambeau, *Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages*, in the proceedings of CHES 2008, LNCS, vol 5154, Washington DC, USA, August 2008.

A Pseudo-code of PRESENT

```
P' ← P
Round key 1 ← K
for $i$ = 1 to 31 do
    P' ← P' $\oplus$ Round key $i$
    P' ← Substitution(P')
    P' ← Permutation(P')
    Round key $i+1$ ← Update(Round key $i$)
end for
C ← P' $\oplus$ Round key 32
```

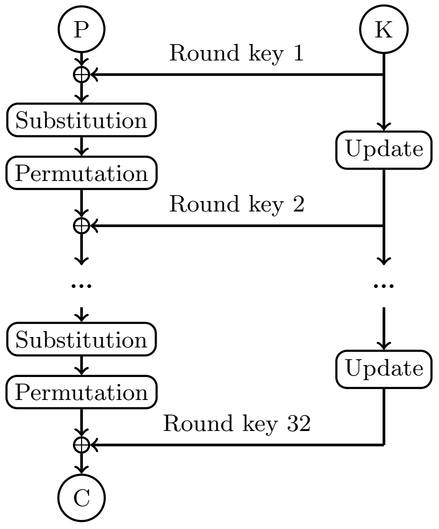

Fig. 9. A top-level algorithmic description of PRESENT.

B Building the system of equations for a 3-bit S-box

Let us consider the 3-bit to 3-bit substitution defined by the lookup table : $\mathsf{S} = \{5, 3, 0, 4, 7, 2, 6, 1\}$, with input bits x_1, x_2, x_3 and output bits y_1, y_2, y_3 (the most significant bits being x_1 and y_1). First, one has to decide the monomials that will appear in the equations, which significantly impacts their complexity. In our example, we choose the monomials x_i, y_i, and $x_i y_j$ ($\forall\ 1 \leq i, j \leq 3$) so that we eventually have 16 monomials (including the independent term). Then a 16×2^3 matrix is built in which each row represents a monomial and each column represents a possible input for the substitution. The matrix is filled as follows: the element (i, j) is the value of the monomial i if the substitution input is j. Finally, we perform a Gaussian elimination on the matrix : we add and swap the rows until the matrix becomes upper triangular (*i.e.* all the elements below the diagonal are null). In our example, we obtain 8 linearly independent combinations of monomials which are null for every possible input of the substitution, as illustrated by the following matrices:

$$\begin{array}{c}1\\x_1\\x_2\\x_3\\y_1\\y_2\\y_3\\x_1y_1\\x_1y_2\\x_1y_3\\x_2y_1\\x_2y_2\\x_2y_3\\x_3y_1\\x_3y_2\\x_3y_3\end{array}
\begin{bmatrix}1&1&1&1&1&1&1&1\\0&0&0&0&1&1&1&1\\0&0&1&1&0&0&1&1\\0&1&0&1&0&1&0&1\\1&0&0&1&1&0&1&0\\0&1&0&0&1&1&1&0\\1&1&0&0&1&0&0&1\\0&0&0&0&1&0&1&0\\0&0&0&0&1&1&1&0\\0&0&0&0&1&0&0&1\\0&0&0&1&0&0&1&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\\0&0&0&1&0&0&0&0\\0&1&0&0&0&1&0&0\\0&1&0&0&0&0&0&1\end{bmatrix}
\Longrightarrow
\begin{array}{c}1\\x_3\\x_2\\x_3+y_2\\x_1\\1+x_2+y_3\\1+x_2+x_3+y_1\\x_1+x_1y_2\\1+x_1+x_2+y_3+x_1y_1\\x_1+x_3+y_1+y_3+x_1y_3\\x_1+y_1+y_2+y_3+x_2y_1\\1+x_1+x_2+x_3+y_1+x_1y_2+x_2y_2\\x_1+x_1y_2+x_2y_3\\1+x_2+x_3+y_2+y_3+x_1y_2+x_3y_1\\1+x_1+x_2+y_2+y_3+x_3y_2\\x_1+y_2+x_3y_3\end{array}
\begin{bmatrix}1&1&1&1&1&1&1&1\\0&1&0&1&0&1&0&1\\0&0&1&1&0&0&1&1\\0&0&0&1&1&0&1&1\\0&0&0&0&1&1&1&1\\0&0&0&0&0&1&0&1\\0&0&0&0&0&0&1&1\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{bmatrix}$$

Hence, the equations we are looking for in order to describe our substitution are:

$$\begin{aligned}
0 &= 1 + x_1 + x_2 + y_3 + x_1y_1\\
0 &= x_1 + x_3 + y_1 + y_3 + x_1y_3\\
0 &= x_1 + y_1 + y_2 + y_3 + x_2y_1\\
0 &= 1 + x_1 + x_2 + x_3 + y_1 + x_1y_2 + x_2y_2\\
0 &= x_1 + x_1y_2 + x_2y_3\\
0 &= 1 + x_2 + x_3 + y_2 + y_3 + x_1y_2 + x_3y_1\\
0 &= 1 + x_1 + x_2 + y_2 + y_3 + x_3y_2\\
0 &= x_1 + y_2 + x_3y_3
\end{aligned}$$

Note that in general, large S-boxes are quite unfavorable for an algebraic representation. The number of linearly independent equations produced is equal to $n_{equ} = n_{mon} - 2^n$, where n_{mon} is the number of monomials used and n is the input size of the substitution. This relation implies that the number of monomials (and thus the complexity of the equations) needs to grow exponentially to match the size of the substitution input. This explains why PRESENT, with its small 4-bit to 4-bit S-boxes is an interesting first target for our attack.

C Definition of a subkey recovery success rate

Let the adversary $\mathsf{A}_{\mathsf{E}_K,\mathsf{L}}$ be an algorithm with limited time complexity τ, memory complexity m and queries q to the target implementation $(\mathsf{E}_K, \mathsf{L})$. Its goal is to guess a subley $s = \gamma(x, k)$ with non negligible probability. For this purpose, we assume that the adversary $\mathsf{A}_{\mathsf{E}_K,\mathsf{L}}$ outputs a guess vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{S}|}]$ with the different key candidates sorted according to the attack result: the most likely candidate being g_1. A success rate of order 1 (*resp.* 2, ...) relates to the probability that the correct subkey is sorted first (*resp.* among the two first ones, ...) by the adversary. More formally, we define the experiment:

$$\begin{array}{l} \text{Experiment } \mathbf{Exp}^{\text{sc-kr-}o}_{\mathsf{A}_{\mathsf{E}_K,\mathsf{L}}} \\ [k \xleftarrow{R} \mathcal{K};\ s = \gamma(x,k);\ \mathbf{g} \leftarrow \mathsf{A}_{\mathsf{E}_k,\mathsf{L}};]; \\ \textbf{if } s \in [g_1, \ldots, g_o],\ \textbf{then} \text{ return } 1,\ \textbf{else} \text{ return } 0; \end{array}$$

The o^{th}-order success rate of the side-channel key recovery adversary $\mathsf{A}_{\mathsf{E}_K,\mathsf{L}}$ against a subkey variable S is straightforwardly defined as:

$$\mathbf{Succ}^{\text{sc-kr-}o,S}_{\mathsf{A}_{\mathsf{E}_K,\mathsf{L}}}(\tau, m, q) = \Pr\ [\mathbf{Exp}^{\text{sc-kr-}o}_{\mathsf{A}_{\mathsf{E}_K,\mathsf{L}}} = 1] \tag{2}$$

When not specified otherwise, a first-order success rate is assumed. [28] also defines an alternative security metric (the guessing entropy) that measures the effectiveness of a side-channel adversary in a more flexible fashion: it corresponds to the average position of the correct key candidate in the guess vector $\mathbf{g}$.

D Average solving times in advanced scenarios

Table 1. Solving times in advanced scenarios, when a 100% success rate is reached. Experiments are performed on an Intel-based server with a Xeon E5420 processor cadenced at 2.5GHz running a linux 32-bit 2.6 Kernel.

Scenario	Nbr. of W_H for 100% success rate	Average solving time (s)
Known P,C - consecutive weights	8 rounds	79,69
Known P,C - random weights	18 rounds	117,1
Unknown P,C - consecutive weights	8 rounds	45,59
Unknown P,C - random weights	26 rounds	214,12
Masked system - consecutive weights	16 rounds	393,86
Masked system - random weights	22 rounds	154,32