Pragmatism vs. Elegance: comparing two approaches to Simple Power Attacks on AES

Valentina Banciu and Elisabeth Oswald

University of Bristol, Department of Computer Science
Merchant Venturers Building
Woodland Road, BS8 1UB, Bristol, UK
{valentina.banciu, elisabeth.oswald}@bristol.ac.uk

Abstract. Simple side-channel attacks trade off data complexity (i.e. the number of side-channel observations needed for a successful attack) with computational complexity (i.e. the number of operations applied to the side-channel traces). In the specific example of Simple Power Analysis (SPA) attacks on the Advanced Encryption Standard (AES), two approaches can be found in the literature, one which is a pragmatic approach that involves basic techniques such as efficient enumeration of key candidates, and one that is seemingly more elegant and uses algebraic techniques. Both of these different techniques have been used in complementary settings: the pragmatic attacks were solely applied to the key schedule whereas the more elegant methods were only applied to the encryption rounds. In this article, we investigate how these methods compare in what we consider to be a more practical setting in which adversaries gain access to erroneous information about both key schedule and encryption rounds. We conclude that the pragmatic enumeration technique better copes with erroneous information which makes it more interesting in practice.

1 Introduction

Historically, simple side-channel analysis seems an under-researched area in the context of implementations of symmetric schemes: after a short remark by [8], initially only Mangard's article [9] discusses an SPA-style attack on the key schedule of the Advanced Encryption Standard (AES). Thereafter, interest was only revived by the advent of algebraic side-channel analysis (ASCA) (see [13,14, 16]). In contrast to Mangard's SPA attack, which used a pragmatic enumeration technique applied to the AES key schedule, ASCA represent the whole block cipher (encryption rounds and key schedule) as a system of equations (in the input, output, and key) that explicitly includes side-channel information. Then some standard solvers (e.g. SAT solver) are employed to (elegantly) solve this system which leads to the extraction of the key.

In these early works little emphasis was put on the fact that, in practice, side-channel information tends to be noisy. Consequently, all early methods implicitly assumed an ideal measurement setup, or some (clever) trace processing, and the

use of templates. More recently this shortcoming was picked up in a series of papers [10–12, 17] which move away from simply using a standard SAT solver to (e.g.) tools that can incorporate probability information about the side-channel observations. This is a step towards making ASCA-style attacks potentially more applicable to practice. However, approaches such as the one in [12] still assume some form of template-based side-channel information extraction so do not move away much from the afore mentioned implications for practice. Other recent contributions in this area [2, 10] focus on how the algebraic representation of AES (which can be written in more than one way) influences the computation time/complexity.

By looking at this historical development, one might begin to wonder about the seeming divergence of the 'two' different approaches to SPA. On the one hand, there is the somewhat trivial technique described by Mangard, which only takes key schedule information and extracts the key without much computational effort. On the other hand, there is the elegant technique of algebraic attacks, which only takes round information and extracts the key with considerable computational resources. From a practical perspective (different to the one related to error tolerance above), one can hence wonder why nobody has looked into the strategy of combining key schedule and round information with the aim of using observations concentrated at the beginning or end of AES. This point of staying 'close to' the extremes of AES is motivated by the practical aspect of extracting the side-channel information from the acquired traces: the closer to the beginning (or end) the information is located, the closer one is to the trigger point which can imply a more robust process of finding and extracting the required information. Naturally, practitioners would prefer methods which are robust per se, but also incorporate some error tolerance.

In this article we compare and contrast the two main approaches to SPA on AES in a setting that we consider more practical than what was considered in previous work: we aim to exploit erroneous side-channel information from the beginning of AES (including the key schedule) using an extension of Mangard's simple enumeration technique, as well as using an algebraic method focusing on Hamming weight as leakage model.

Our paper[1] is structured as follows. We briefly review AES in Sect. 2. We explain our extension of Mangard's attack including results in Sect. 3. Thereafter we explain our implementation of algebraic attacks including results in Sect. 4. We conclude in Sect. 5. Appendix A provides results of some more experiments that we performed. These experiments use Hamming distance as power model and show that our conclusion remains valid: the pragmatic approach copes better with erroneous information and hence is more suitable for practice.

[1] This paper has been accepted for publication at COSADE 2014 and will be available at link.springer.com.

2 A Brief Recap of AES

The Advanced Encryption Standard (AES) is a symmetric block cipher, with a fixed block size of 128 bits, and a variable key size of 128, 192, or 256 bits corresponding to 10, 12 and 14 rounds respectively. We use the 128-bit variant as an example in this article, to which we shall refer as simply AES throughout this document. In this section, we give a brief overview of the encryption and key schedule algorithms and explain what intermediate values we assume to be leaking.

2.1 AES Encryption Round

At the start of the encryption process, the 16-byte plaintext block is copied to a 4×4 array called `state`. The byte elements of the initial plaintext array are copied in column order. Thereafter an encryption round consisting of four round transformations is repeatedly applied to the state. The round transformations are as follows:

1. $AddRoundKey(\texttt{state}; RK_i)$ performs a bitwise xor of the current round key and state. One would expect that all memory transfers (i.e. loading of the state as well as key bytes) and the output are leaking, although previous work typically only takes the leakage of the output into account.
2. In the $SubBytes(\texttt{state})$ step, each byte in the state matrix is replaced according to a look-up table. This operation provides *non-linearity*. We only use the leakage of the output (as the input leakage is already being used from the step before).
3. $ShiftRows(\texttt{state})$ operates on the rows of `state`, performing a cyclical left shift of the bytes in each row by a certain offset: row n is shifted $n-1$ positions. We assume that this is done implicitly via memory access and so we do not use any leakage.
4. $MixColumns(\texttt{state})$ combines the four bytes of each column of the state. An efficient implementation of this representation on an 8-bit microcontroller is described in the original AES proposal [5], and we list it here to keep our work self contained. Let $in_i, out_i, i = 1 \ldots 4$ be the input, respectively output bytes of a single column, and consider the index i modulo 4. Then, a single column is computed as follows:

$$\begin{aligned} Tmp &= in_1 \oplus in_2 \oplus in_3 \oplus in_4 \\ Tm_i &= in_i \oplus in_{i+1} \\ Tm_i &= \texttt{xtime}(Tm_i) \\ out_i &= in_i \oplus Tm_i \oplus Tmp \end{aligned} \tag{1}$$

   where `xtime` is the multiplication by `02` over $\texttt{GF}(2^8)$. Given the target platform that we have in mind, we would assume that only 2-operand instructions are available on the target platform and hence the exclusive-or of all inputs in_i is done in three steps and the computation of any out_i takes two

steps. However, previous work such as [16] set a precedent of only considering leakage of the variables Tmp, Tm_i and out_i and so to keep our work in this respect comparable to theirs we only take 13 out of the 19 leakage points per column.

Adding up the leakage points as explained above amounts to 21 points per column (4 from AddRoundKey, 4 from SubBytes, and 13 from MixColumns).

2.2 AES Key Schedule

For the key expansion, the secret key is represented as 4 concatenated words, $SK = W_1 \parallel W_2 \parallel W_3 \parallel W_4$. Then, subsequent round keys $RK_{1...10}$ are derived in lexicographic order, each key depending on the previous. The operations used the key expansion are as follows:

1. $RotWord(W)$ performs a cyclic shift to the left on a word by one byte. We assume that each byte in the word will leak.
2. $SubWord(W)$ substitutes each of the 4 constituent bytes of W according to the AES S-box, which can be implemented as a 256-bit lookup table. We expect leakage for each S-box look-up.
3. `Rcon`, which is a predefined round constant, is exclusive-ored to a byte of the key. We expect the result of exclusive-or to leak.

2.3 Further Implementation Aspects

SPA attacks are typically studied in the context of software implementations on simple (i.e. serial) micro-processors. This implies that we expect to observe leakages for all state bytes as and when they are processed. As explained before, we adhere to this by-and-large and only deviate from this principle to keep our work comparable with previous publications.

Typical power models that are found in practice (for small micro-processors) are the Hamming weight (short HW, i.e. the number of non-zero bits) of a byte, or the Hamming distance (short HD, i.e. the number of non-zero bits in the exclusive-or of two bytes). Leakages of this kind are observed mainly because of intermediate values being written to (and read from) memory, which causes bus transfers. Obviously, for HD leakage one then needs to know precisely which two intermediate values are processed in sequence.

Notice that for our attacks we did not use data from an actual device. This is motivated by the fact that we are not interested in the problem of how to find and best extract the available leakage from real traces. Our contribution is with regards to how to best (i.e. mathematically) exploit the extracted leakage. So we use simulations to generate (truly leaked) HW (and HD) values and then 'embed' them in sets of a given size to simulate noise (i.e. the fact that one might not have certainty for the HW (or HD) of correct leakages). These sets are ordered sequences with the correct leakage as centre value, e.g. if the correct leakage for an intermediate value is 5, a set of size three is $\{4, 5, 6\}$. We assume a uniform distribution for the 'incorrect' values within each set in our experiments.

Algorithm 1 An informal description of an enumeration attack aimed at recovering four bytes of the secret key SK using leakages of a single AES round.

for $i = 1 \rightarrow 4$ **do**
    generate $KeySet_i$ such that each key in $KeySet_i$ satisfies the observed leaks $\mathcal{L}(PT_i \oplus SK_i)$, $\mathcal{L}(SB_i)$ and $\mathcal{L}(SK_i)$.
end for
for all $K_1 \in KeySet_1$ **do**
    for all $K_2 \in KeySet_2$ **do**
        for all $K_3 \in KeySet_3$ **do**
            for all $K_4 \in KeySet_4$ **do**
                retain values that also match $\mathcal{L}(Tm_i)$, $\mathcal{L}(Tmp)$, and $\mathcal{L}(out_i)$
            end for
        end for
    end for
end for
return four sets $\{K_i\}$ of 8-bit values that simultaneously satisfy all observed leakages

3 Pragmatic Attack on AES

Like [9], we assume that the attacker is able to extract the relevant information from the power traces and assign it to the respective intermediate value in both the encryption round and the key schedule. Differently to [9], we assume, however, that the extracted information is possibly erroneous. Consequently, each leakage point translates into a set of leakage values (rather than a single value). A necessary condition for our attack to produce meaningful results is then that each set includes the correct leakage value.

Whilst we did not aim for the most efficient implementation that is conceivable, we paid some attention to choosing strategies that speed up testing keys against leakages. The basic strategy of an SPA attack such as [9] is that by observing leakages relating to different intermediate (key dependent) values, one learns something about the involved key bytes and hence reduces the overall search space for the key. Illustrating this on a simple example that is the starting point for an SPA attack, we note that by observing leakages on (e.g.) a plaintext byte PT (we denote this with $\mathcal{L}(PT)$) and on the key addition with this byte $PT \oplus SK$ (i.e. we see a leakage $\mathcal{L}(PT \oplus SK)$), we can enumerate and in fact precompute all those values of SK which satisfy the observed leakages (we hence enumerate the set $K_{v,w} = \{k | \mathcal{L}(PT \oplus k) = w, \mathcal{L}(PT) = v\}$). It is in fact sufficient to fix the Hamming weight leakage of the plaintext to an arbitrary value (we chose 0) because $(PT \oplus k) = ((0 \oplus k) \oplus PT)$, which means that the possible key set corresponding to any nonzero plaintext byte can be easily derived by adding PT as offset to the key set corresponding to the null value byte. We hence can optimise and store only one such table for $PT = 0$.

Just observing such a single leak reduces hence the key space and we use this reduced key space to further process and incorporate leakages from our traces, i.e. for each possible key resulting from only looking at the first key addition,

we can also check the leakage from the $SubBytes$ operation, which then reduces our key space further. One can again build (precompute) tables that enumerate possible key byte values for given input and output leakages, so this step in a practical attack corresponds to a table lookup.

Advancing further into the AES round means that after $ShiftRows$, which we assumed would give no explicit leakages because it would be done as part of writing the byte back into the state, we work with intermediate values that arise from the $MixColumns$ operation. Here, we choose not to attempt further precomputations, but rather took leakages 'on the fly' to further prune the key space, see Alg. 1 for an informal algorithmic description of this process as applied to a single column in one round.

3.1 Attack Results

We performed all our analysis using noisy Hamming weight leakages, i.e. we chose sets of different sizes that contain the correct leakage (ranging from set size 1, which corresponds to no noise, to set size 5, which corresponds to tolerating 2 bits of noise).

All computations that we now discuss were performed by using a single node on a high-performance computing facility. Such a single node is comprised of two 2.8 GHz 4-core Intel Harpertown E5462 processors, with one GB RAM per core. Our code ran in Matlab on this platform. We terminated attacks after 48 hrs or if they ran out of RAM memory on the node. We give the percentage of attacks that terminated successfully (i.e. that terminated within the 48hrs limit and did not run out of memory) for each experiment. We provide 'indicative execution times' for all experiments: these are mean values taken over the successfully terminated experiments. We want to caution against making any inferences from these times, because although we made some effort to produce 'efficient' attack implementations, we by no means claim any optimality in any respect (recall that we ran the attacks using Matlab). Consequently, these indicative execution times are best understood along the lines of that some attacks terminate within the order of several hours whereas others terminate within the order of seconds, etc. We also note that the timings produced only refer to the effort of reducing the key search space using the side channel information. The overall time required for an attack, i.e. reducing the key search space and the performing the brute-force search, would very much also depend on the brute-force search.

Attack using leaks from the first encryption round. By referring back to the description of an AES round and the expected leaks that we gave in Sect. 2.1, we note that we have 21 exploitable leakage points to attack 4 bytes of the first round key (which corresponds to one column of the state). Consequently we assume that we have 84 such points available to attack an entire round. The attack strategy that we explained in the previous section, which works on one column, can independently (and hence in parallel) be replicated and applied to all four state bytes.

Table 1a shows that allowing for more noisy leakages increases the computational effort quickly, as one would expect. Clearly for noisy leaks the reduction in key space size renders the attack actually impractical.

Attack using leaks from first encryption round and key schedule Rather than making more complicated inferences to incorporate more information from the second encryption round, it seems more natural now to incorporate the strategy of [9] and draw on the information that is present in the key schedule. The attack of Mangard requires, depending on how many key hypotheses one wishes to brute force test at the end, 40 up to 81 intermediate values from the key schedule to succeed. We chose, for the sake of consistency, to use leaks from the first round of the key schedule only.

When faced with 'merging' the two attack strategies one has different options. We decided to use the result of the attack on the round as a starting point to the attack on the key schedule. In other words, we start the attack on the key schedule with an already reduced key space.

Table 1c shows the results of the combined attack. The incorporation of the noisy key schedule leakages has had a significant impact especially in the case of set size five (i.e. 2 bits of noise). Now even this case leads to a final key space size that can be searched through and hence leads to a practical attack.

Just for comparison we also give the numbers of Mangard's attack on the key schedule only in Tab. 1b (re-implemented and adapted to target a single round with possibly noisy leaks). It should be obvious that by itself the strategy does not tolerate noise very well. We can hence conclude that using leaks from both encryption round and key schedule is indeed the most natural and promising attack path.

4 Elegant Attack on AES

The elegant attacks that we now want to consider are essentially algebraic attacks that incorporate additional information about the key bytes because of leakages. The technique is viewed as elegant because one can (in theory) feed the system of equations describing AES into some black box solver which returns the key provided enough side-channel information is supplied.

As mentioned in the introduction, recent research has drawn attention to the fact that it makes a significant difference (to the various black box solvers) how and which equations are fed into them, and hence there is scope to optimise attacks by rewriting the algebraic representation of a cipher—clearly the black box solver is more of a grey box then.

From a practical perspective, anyone implementing an algebraic attack that uses side-channel information needs to hence make two important choices. Firstly, how to represent the cipher and secondly, which sort of solver to use. In our study here we incorporated techniques that were published in previous work to ensure we have a reasonably efficient representation. Of the many available solvers, we used SAT solvers (we use state-of-the-art software, i.e. CryptoMiniSat 2, and did not develop our own tools).

Table 1: Summary of results of attack on one round

(a) Encryption round only

Set size	Approximate key space size	Indicative execution time	Successful termination
1	1	0.02 s	100%
2	2^{20}	2.9 s	100%
3	2^{48}	73.9 s	100%
4	2^{64}	27 mins	100%
5	2^{116}	2.5 hrs	78%

(b) Key schedule only

Set size	Approximate key space size	Indicative execution time	Successful termination
1	2^{58}	0.4 s	100%
2	2^{74}	5 s	100%
3	2^{95}	10 s	72%
4	2^{106}	30 s	40%
5	2^{115}	40 s	22%

(c) Round and key schedule

Set size	Approximate key space size	Indicative execution time	Successful termination
1	1	0.03 s	100%
2	2^{12}	27 s	100%
3	2^{13}	4 mins	80%
4	2^{52}	35 mins	20%
5	2^{60}	12 hrs	10%

Whilst most side-channel attacks follow a divide-and-conquer strategy, when performing an algebraic attack, the adversary aims for full key recovery in one go and is able to make use of all available side-channel information at once. We assumed that attackers would include leakages corresponding to round operations and, in contrast to previous work, the key schedule.

4.1 Solver-Specific Requirements

To be able to make use of a standard solver, one needs to translate the high-level description of a cipher into a format that the solver can work with. Essentially this translation requires two steps. The first step is to linearise the system of equations that represents the cryptographic algorithm. This can be done by introducing a new variable for each higher degree monomial in the algorithm's representation (monomials might represent (e.g.) bits of intermediate values or bytes). The second step is to translate this linear system into an appropriate format, e.g. conjunctive normal form (CNF) for SAT solvers or a system of Boolean inequalities for Pseudo-Boolean Optimizers.

Linear layers, such as $AddRoundKey$ or $MixColumns$, give rise to relatively simple equations. Non-linear layers, i.e. $SubBytes$, lead to fairly complex equations, and there is some scope for optimising them. Based on work by [3], an expression for an 8-bit S-box in polynomials of maximal degree 8 was given in [6]. Still, it was shown in [4] that SAT solvers give best performances when the degree of equations and the size of terms is limited to smaller values. Using some specific algebraic properties of $SubBytes$, Courtois et al. also derive a system of 23 quadratic equations describing it, which is shown to be maximal. We used this approach in our work.

Overall, we thus represented all intermediate values as variables with appropriate equations linking them to each other. An initial count of the expected number of variables is consequently as follows. For the key schedule, 128 variables are required for each round key, and for the secret key. Auxiliary variables can be used for the output of the S-box, but are not needed for xoring with `Rcon` since this is fixed; this operation can be just as well modelled without introducing any equations, since xoring with 1 is equivalent to negation. Thus, the equations for each round key describe only the S-box (23×4) and the xoring with `temp`. Additionally, 128 variables are required for each intermediate output state of $AddRoundKey$, $SubBytes$ and $MixColumns$ during the encryption process, and for the plaintext. The number of equations is calculated as follows: 23×16 for each of the 10 S-box layers, 128 for the 11 key addition layers and the equations corresponding to the 9 rounds of $MixColumns$, which can be represented either as recommended in the Rijndael proposal [5], leading to 13×4 equations per round, or as in [7] as a direct bitslice implementation, leading to 128 equations per round. Of these, the equations corresponding to the S-box are the only non-linear ones. When translating the system to CNF, dummy variables are necessary for linearisation and for keeping the size of each term up to 4 monomials (as recommended by [1,4]), in particular approximately 500 auxiliary variables and 400 equations are required per S-box, which leads to a final form consisting in approximately $100,000$ variables and $130,000$ equations.

Finally, equations representing side-channel information are added to the system. We adopt the same strategy of [15], to explicitly list all possible values corresponding to each leakage point. However, we do use the pre-computation strategies described in Sect. 3, to build explicit values corresponding to the input and output pairs of each S-box.

4.2 Attack Results

We ran several experiments with our implementation. In these experiments we varied the number of encryption rounds from which we source information as well as the amount of noise that we want to tolerate by varying the set size. Table 2 gives an overview of the results for AES. Remember that our attack (in contrast to previous work on algebraic attacks) uses key schedule information in addition to round information. There is little difference between attacking only one or many rounds (the timings have some variation and the reported means are hence about equal) with regards to timings. We speculate that this is because

Table 2: Indicative solving time (in seconds) for AES, using encryption and key schedule leakage

Attacked rounds	1 round	2 rounds	3 rounds	4 rounds	5 rounds
set=1	10.39	10.85	11.03	11.10	11.30
set=2	41.24	43.11	43.25	43.49	43.73

the complexity of the equations solved stays the same irrespective of how many rounds are used. Obviously, the more rounds one includes the more intermediate values need to by extracted. For each encryption round, 84 intermediate values are used, corresponding to 32 values for the output of $AddRoundKey$ and $SubBytes$ and 4×13 values corresponding to the intermediate values of $MixColumns$. Additionally, for each round key at most 21 intermediate values can be exploited, out of which 16 correspond to the key bytes, 4 to the S-box output and 1 to the xoring with the round constant.

As expected the set size is the main factor that influences the overall computation time. We limited any solver run to 48 hours (alike previous work). Given this constraint, none of our attempts to solve instances of set size three or larger was successful. However in contrast to previous work we could solve all instances of set size two that terminated within the 48 hrs cut-off time. Clearly adding some key schedule information helps the solver.

5 Conclusion

The research presented in this paper was based on the question of how elegant (black box) solvers compare with a simple and reasonably efficient extension of Mangard's SPA attack, in a scenario where some erroneous side-channel information is available. In contrast to previous work, we considered the scenario in which an attacker has access to erroneous leakages from both the encryption round and the key schedule (but limited to a single or a few rounds).

Our implementation of a pragmatic SPA attack shows that with very few leakage points (we only use leakage points that occur within the first round of AES and the key schedule) we can reduce the key space even with noisy leakages to a size which can be searched through using today's computing technology. We speculate that with a more efficient implementation, this could be improved further by taking more rounds (of the key schedule and the encryption) into account. Including key schedule information in the elegant ASCA-style approach helps, but we were not able to push beyond set size two. However, all our attacks with set size two were successful, even when limited to using leaks from the first round only, which is some practically relevant progress.

Our conclusion from the performed experiments is that the pragmatic approach seems to be more suited for actual practical attacks because of its ability to better tolerate noisy leakages and its concrete result that allows to actually rule out keys and provide a concrete reduction of the key space. This is in con-

trast to using algebraic solvers, which either terminate successfully, or leave you with no further information.

Acknowledgments. Valentina Banciu has been supported by EPSRC via grant EP/H049606/1. Elisabeth Oswald has been supported in part by EPSRC via grant EP/I005226/1.

References

1. G.V. Bard, N. Courtois, and C. Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. *IACR Cryptology ePrint Archive*, 2007:24, 2007.
2. C. Carlet, J.-C. Faugère, C. Goyet, and G. Renault. Analysis of the Algebraic Side Channel Attack. *Journal of Cryptographic Engineering*, 2(1):45–62, 2012.
3. N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. *IACR Cryptology ePrint Archive*, 2002:44, 2002.
4. N. Creignou and H. Daude. Satisfiability threshold for random XOR-CNF formulas. *Discrete Applied Mathematics*, 96:41–53, 1999.
5. J. Daemen and V. Rijmen. AES proposal: Rijndael. In *First Advanced Encryption Standard (AES) Conference*, 1998.
6. D. Gligoroski and M.E. Moe. On deviations of the AES S-box when represented as vector valued Boolean function. *International Journal of Computer Science and Network Security*, 7(4):156–163, 2007.
7. E. Käsper and P. Schwabe. Faster and Timing-Attack Resistant AES-GCM. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009)*, pages 1–17, 2009.
8. P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
9. S. Mangard. A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion. In *ICISC*, pages 343–358, 2002.
10. M.S.E. Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter, and J. Buchmann. Improved Algebraic Side-Channel Attack on AES. In *HOST*, pages 146–151, 2012.
11. Y. Oren, M. Kirschbaum, T. Popp, and A. Wool. Algebraic Side-Channel Analysis in the Presence of Errors. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2010)*, pages 428–442, 2010.
12. Y. Oren, M. Renauld, F.-X. Standaert, and A. Wool. Algebraic Side-Channel Attacks Beyond the Hamming Weight Leakage Model. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2012)*, pages 140–154, 2012.
13. M. Renauld and F.-X. Standaert. Algebraic Side-Channel Attacks. In Bao F. and Yung, M. and Lin, D. and Jing, J., editor, *Information Security and Cryptology (INSCRYPT) 2009*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2009.
14. M. Renauld and F.-X. Standaert. Combining Algebraic and Side-Channel Cryptanalysis against Block Ciphers. In *30-th Symposium on Information Theory in the Benelux*, 2009.
15. M. Renauld and F.-X. Standaert. Representation-, Leakage- and Cipher- Dependencies in Algebraic Side-Channel Attacks. In *Industrial track of ACNS 2010*, 2010.

16. M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon. Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009)*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009.
17. X. Zhao, F. Zhang, S. Guo, T. Wang, Z. Shi, H. Liu, and K. Ji. MDASCA: an enhanced algebraic side-channel attack for error tolerance and new leakage model exploitation. *Constructive Side-Channel Analysis and Secure Design (COSADE 2012)*, pages 231–248, 2012.

A More experimental results

Table 3: Summary of results with HD model

(a) Pragmatic attack on one round

Set size	Final key space size	Execution time	Success rate
1	2^{13}	0.03 s	100%
2	2^{48}	7 mins	90%
3	2^{58}	4.5 hrs	32%
4	2^{66}	20 hrs	8%
5	N/A	>24 h	0%

(b) Algebraic attack up to several rounds

Attacked rounds	1 round	2 rounds	3 rounds	4 rounds	5 rounds
set=1	20.13	20.08	20.13	19.88	19.68
set=2	641.52	601.46	600.33	609.30	640.31