Differential Behavioral Analysis

Bruno Robisson and Pascal Manet

CEA-LETI, SESAM Laboratory,
Centre Microélectronique de Provence.
Avenue des Anémones, 13541 Gardanne, France
bruno.robisson@cea.fr, pascal.manet@cea.fr

Abstract. This paper describes an attack on cryptographic devices called Differential Behavioral Analysis (or DBA). This is an hybrid attack between two already powerful attacks: differential power analysis (DPA) for the statistical treatment and safe-error attack for the fault type. DBA, simulated on an algorithmic model of AES appears to be very efficient. The attacker is able to recover the entire secret key with byte-wise "stuck-at" faults injected repetitively. A theorical as well as a more realistic approach are presented.

Keywords. Differential Behavioral Analysis, Differential Power Analysis, Fault Attacks, Safe-Error, Block Ciphers, AES.

Introduction

Several methods, called "attacks", have been proposed to retrieve the secret information stored in cryptographic devices like smart cards. One of the most powerful and studied method, called Differential Power Analysis (or DPA [KJJ99], [BCO04]), exploits the fact that the power consumption of the chip depends on its internal computations (among them several depend on the value of the secret key). It is of particular concern, since it does not destroy the physical integrity of smart cards and it can be quickly mounted with cheap instrumentation equipments. A second type, called "fault attacks", consists in modifying the circuit's behavior in order to bypass hardware or software protections or to exploit computational errors to find the cryptographic keys([BDL97], [BS97], [Gir05], [PQ03], [CT05], [BK06]). The faults are injected into the device by various means as laser, glitches on clock, spikes on voltage supply or electromagnetic perturbations [BECN$^+$04]. Among fault attacks, safe-error attack (SEA) only checks if the computation is correctly performed or not ([YJ00], [BS03]). A third type, which is far more complicated, consists in analyzing the design of the chip by using destructive means such as abrasion, chemical etching or SEM and then probing the most informative signals with, for example, focused ion beam [KK99].

We propose in this paper a new attack, called Differential Behavioral Analysis (DBA), which exploits both SEA principle and DPA statistical approach. This "hybrid attack" combines a large part of the qualities of these two methods. The attack is validated by simulation on an algorithmic model of AES and the obtained results lead us to conclude that DBA may be performed on a real device.

The rest of this paper is organized as follows: in the first section, DBA algorithm is presented. Section 2 applies monobit DBA to AES and some improvements are proposed. Then, multibit DBA is performed on AES. Comparison with existing attacks will be done before concluding.

1 Differential Behavioral Analysis

DBA consists in matching the behavior of the chip to be analyzed in the presence of a fault with a model of behavior which is parameterized by the value of a partial key (that is a restricted number of bits of the key). This attack thus borrows the study of the behaviors of a chip in the presence of faults from safe-error attack and the mathematical treatment from DPA attacks.

1.1 Hypothesis

DBA concerns hardware implementations of cryptographic algorithms that are subject both to DPA and to fault attacks (particularly DES and AES). Just like these attacks, the implemented algorithm has to be known and to be executed with known variable plaintexts (chosen or not) and with or without perturbations. These perturbations should have the following properties:

- Type: they should induce a "stuck-at" fault but this value is not necessarily known;
- Location: the fault should occur on bits corresponding to some particular intermediate values;
- Focalization: they should affect only a small number of bits (typically less than 8);
- Value: the "stuck-at" value is the same for all the affected bits;
- Repetitivity: they should induce the same "stuck-at" fault on the same bits for different plaintexts.

At last, the attacker has to distinguish between the normal or abnormal behavior in presence of the perturbations described above. To detect these two types of behaviors, the attacker can be led to distinguish between the correct and faulty ciphering, the start of an alarm or not, the raise time of the alarm or a more or less premature stop in computation (by a simple analysis of power consumption, for example).

1.2 Algorithm

Let a chip perform a known cipher function from a plaintext and a key K_0 (unknown and to be found). Let $\mathcal{T}$ and $\mathcal{K}$ be respectively the set of possible values for plaintexts and keys. $T \subset \mathfrak{T}$ is the set of plaintexts used to perform DBA. This set T can be either chosen or not, depending on the means of the attacker. The DBA consists of the four following stages:

- Stage 0 (choice of parameters): first, the attacker chooses two sets K and B from the knowledge that he has on the cryptoalgorithm under study. $K \subset \mathcal{K}$ is the set of values of the partial key and B is the set of N attackable bits so that each one is a function of the plaintext and a partial key. Let us note $\{b_0, b_1, ..., b_{N-1}\}$ the elements of B.
  Second, the attacker chooses from the knowledge that he has on his fault injection benches, two parameters M and f. $M \leq N$ corresponds to the maximum number of bits that are supposed to be modified by the fault injection. For example, if the attacker knows that his fault injection method creates only single faults, he will choose $M = 1$. But with no information on the impact of the fault injection, he will prefer to choose $M = N$. Let S_M be all the possible partial sets S_M^j from B with at most M elements and at least one, such that:
  $s_M^1 = \{b_0\}$
  $s_M^2 = \{b_1\}$
  ...
  $s_M^N = \{b_{N-1}\}$
  $s_M^{N+1} = \{b_0, b_1\}$
  ...
  $s_M^{C_N^M} = \{b_0, b_1, ..., b_{N-1}\}$

  In the same way, the attacker chooses a value $f \in \{0; 1\}$ which corresponds to the value of the "stuck-at" that is supposed to be injected.
  At last, for each partial key $k_p \in K$ and each plaintext $t_i \in T$, let us note $r^f_{S_M^j}(k_p, t_i)$ the function which returns 0 if all the bits of S_M^j are stuck at f and 1 otherwise (meaning at least one of the bits has the value $not(f)$).
- Stage 1 (experimentations): for all t_i in T :

  - Step 1: ciphering of plaintext t_i, and storing the behavior $C_0(t_i)$ (in this case normal) of the chip during the first round.
  - Step 2: ciphering of plaintext t_i with a perturbation such as described in subsection 1.1 and storing the behavior $C'(t_i)$ of the chip during the first round.
  - Step 3: associating plaintext t_i with value $c(t_i)$ so that $c(t_i)$ is 0 if $C'(t_i) = C_0(t_i)$ and 1 otherwise. After stage 1, we obtain a function $c(t_i)$ that returns 1 if the fault injected during ciphering of plaintext t_i with key K_0 did produce an error during the first round, and returns 0 in the other case.

- Stage 2 (model matching): for each S_M^j in S_M, each $k_p \in K$ and each $t_i \in T$, let us compute theorical values of the bits $r^f_{S_M^j}(k_p, t_i)$.
  We then calculate the sum of the matching behaviors of these two sets with

the following formula (which is a correlation measurement but not a correlation coefficient):

$$\Delta^T\left(k_p, r^f_{S^j_M}\right) = \frac{\sum_{t_i \in T}\left[r^f_{S^j_M}(k_p, t_i) \times c(t_i) + \left(1 - r^f_{S^j_M}(k_p, t_i)\right) \times (1 - c(t_i))\right]}{|T|}$$

- Stage 3 (interpretation): the curve made of the points $\{k_p, \Delta^T(k_p, r^f_{S^j_M})\}$, is called DBA curve associated with the bits S^j_M, the texts T and the "stuck-at f" fault model. We overlay on a same graphic the DBA curves associated with all the possible combinations of bits stuck and examine this graphic. We will see in this paper that in most case $\Delta^T(k_p, r^f_{S^j_M})$ reaches its maximum for S^j_M being the real injected fault and for $k_p = K_0$. Thus, DBA enables to retrieve information about the partial key k_p but also which logical value is induced by the perturbation.

In the following section, we propose to apply DBA to an algorithmic model of an AES-128 ([NIS01], [DR02]). No fault injection campaign has still been done on a real device to validate DBA but simulation of faulty cryptographic algorithm has been performed.

2 Mono-bit DBA on AES-128

2.1 Case study

For pedagogical purpose, we suppose in this section that T can be chosen by the attacker and that he knows:

- which S-box outputs are impacted by the faults,
- that the injected "stuck-at" value is equal to zero,
- that his fault injection method modifies just one bit. That's why we also call such a restricted attack "mono-bit" DBA.

2.2 DBA parameters

In the conditions described above, the attacker will choose $M = 1$ and $f = 0$.

In AES, the 128 bits of the first round key is XORed bitwise with the 128 bits of the plaintext. This result is split into 16 blocks of 8 bits. Each of them becomes the input of a S-box, which returns 8 bits in a non linear way. Consequently, each of the 128 bits which constitutes the output of the S-boxes depends on only 8 bits of the plaintext and 8 bits of the key. This analysis of the AES algorithm allows us to choose the set K of partial keys and the set B of attackable bits: K is chosen as the whole set of the $2^8 = 256$ distinct values of the key which exhausts all the possible values at the entry of the considered S-box and B is chosen as the set of the 8 bits at the output of this S-box. At last, T is the whole set of the $2^8 = 256$ distinct values of the plaintext which exhausts all the possible values at the entry of the considered S-box.

Experimentations: faulty simulations Simulations of the AES behavior in the presence of faults are performed with a modified software description of the algorithm. In this description, the “state” defined in the FIPS standard, can be modified at the ouput of each transformation (AddRoundkey(), SubByte(), etc.). The modifications can simulate a transient or permanent “stuck-at” fault or a “bit-flit”.

In this experimentation, a transient “stuck-at” zero fault is injected on a given bit at the output of the S-box during the first round. It is important to note that even if this faulty bit is chosen during the simulation process, it is *a priori* unknown during the real attack. Correct and faulty encryptions have been performed for all the elements t_i. The normal and abnormal behaviors have been obtained by comparing output results. The key used for the encryption is chosen randomly.

Model matching and interpretation The 8 DBA curves, each of them corresponding to an element S_1^j, are computed thanks to the algorithm described in subsection 1.2. They are depicted on Figure 1. We observe that a peak appears clearly. It is associated with the curve which corresponds to the bit that was corrupted during simulation (in this case S_1^4). The peak is located at the decimal value corresponding to the partial key used for encryption (in that case 139). As, the amplitude of this peak is positive on the top plot, the attacker can conclude that the value of f that has been injected is indeed 0. At last, as the amplitude of the peak is equal to one, the attacker can conclude that the fault injection impacts the circuit in an identical manner for all the faulty executions of the algorithm.

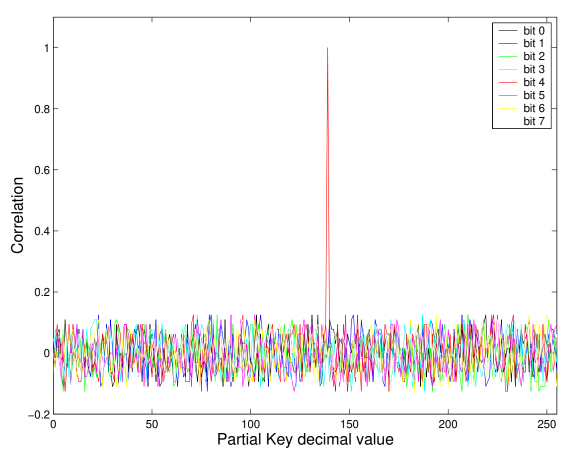

Fig. 1. Mono-bit DBA results on a simulated AES

Note that if the value of f is unknown, the attacker will also test with $f = 1$ and, as $M = 1$, will obtain the set of inverted curves.

2.3 Improvements of single-bit DBA on AES-128

We show in this section that the attack described above is still successful with relaxed constraints on the fault injection in terms of location and repetitivity.

Location In previous paragraphs, the set B, called attack bits, is composed of bits at the output of S-boxes and the faults are injected on those bits. In AES, some bits are perfectly correlated (that is either identical or opposed for all plaintexts) to these attack bits.

For example, ShiftRows which switches the orders of bytes, does not affect the value of the bits but only their location. MixColumns multiplies four bytes by constants and adds them to obtain a four new bytes value. Assuming that three of the input bytes are constant (it is the case when the attacker can choose as plaintexts the set which exhaustes the values at the input of the S-box), the values of two of the four output bytes are identical or opposed (depending on the values of the other constant bytes) to the corresponding input byte. In the same way, as AddRoundKey just adds a constant value (the key does not change), the value at the output of this transformation if perfectly correlated to its input.

As a consequence, in the case of chosen plaintexts, the DBA will be successfull if faults are injected at the output of SubBytes, ShiftRows, MixColumns and AddRoundKey even if the attack bits B of the DBA algorithm are the ouputs of the SubBytes.

Furthermore, the same attack can be performed on the last round of the AES. In this case, the hypothesis is made on the last RoundKey and the fault has to occur before the last SubBytes. In such a case, inverse S-box will be used instead of S-box.

Repetitivity

Minimum number of faulty texts The DBA described in section 2.1 was realized with 256 chosen plaintexts. We evaluate in this paragraph what would be the minimum number of plaintexts necessary to retrieve the whole key. For this purpose, for each of the attack bit S_1^j, we defined as a criterion the value, called $d_{S_1^j}(T)$, of the highest peak, divided by the value of the highest peak in the set of all the other partial keys, that is:

$$d_{S_1^j}\left(T\right) = \frac{\Delta^T\left(k_p = K_0, r_{S_1^j}\right)}{\max\left\{\Delta^T\left(k_p, r_{S_1^j}\right) \mid k_p \neq K_0\right\}}$$

Note that if the fault injections are strictly repetitive, the highest peak value (associated with the correct partial key) is always 1.

In order to make this criterion independent of the values of the texts in T and of the value of K_0, we have chosen randomly 100 sets T (with a chosen cardinal $|T|$) and computed the mean of $d_{S_1^j}(T)$ for all these sets. Then, this value, called $D_{S_1^j}(|T|)$, has been computed for cardinals $|T|$ varying from 10 to 128.

Figure 2 depicts the curves associated with each $S_1^j \subset B$, made of the points $\{|T|, D_{S_1^j}(|T|)\}$. DBA clearly points out that the fault has been injected on bit number 7 at the output of the S-box. This figure also shows that for this bit, the criterion is always higher than 1 and that the second peak is about 15% smaller when $|T| > 16$. It means that with a set T made of only sixteen plaintexts, the attacker is able to recover the correct partial key with no ambiguity. Such an identical analysis has been performed for faults injected on every bit at the output of a S-box.

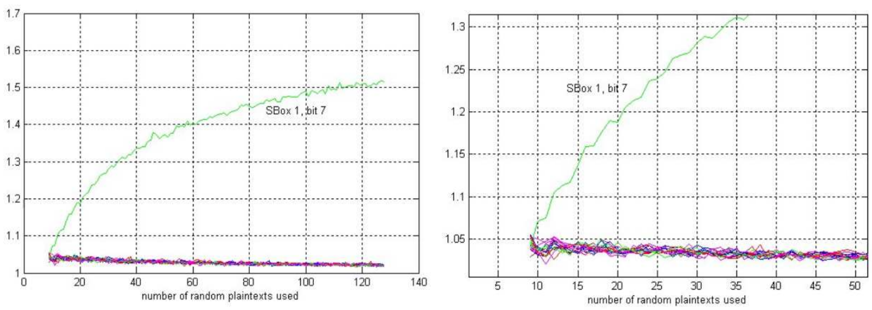

Fig. 2. Evaluation of the criterion for different numbers of random plaintexts for all the bits at the output of S-box 1. On the right, zoom on the lowest numbers.

This study shows that the partial key can be recovered with only about sixteen plaintexts whatever the bit impacted by the fault injection is. So, the whole key may be retrieved with approximately $16 \times 16 = 256$ faulty cipherings.

Wrong injection The DBA described above requires repetitivity of the injection process. As it seems to be a strong assumption, we studied if DBA is still successful when this hypothesis is not strictly true. To this aim, we used the method described above but forced 10 and 20% wrong values in the results of the fault injections. As some faults don't match the model anymore, the highest peak value is no more equal to 1 and decreases along with the wrong injection rate (for example 0.9 for 10% wrong values). Results are given on Figure 3. With 10% wrong injections, 25 plaintexts are required to reach the same criterion value and 60 for 20% wrong injections.

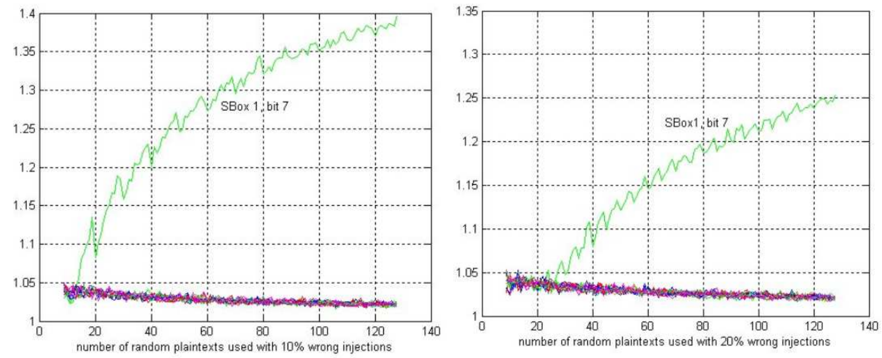

Fig. 3. Evaluation of the criterion for different number of plaintexts when faults don't match the model (left 10%, right 20%).

This study shows that even with a non strictly repetitive injection process, DBA is successful. But the higher the wrong injection rate is, the more plaintexts required to recover the key are.

3 Multibit DBA on AES-128

Because the attacker is not sure that the injected faults affect only one bit, we suppose in this section that he just knows which S-box is impacted by a fault potentially multiple but with the same "stuck-at" value.

3.1 DBA parameters

In the conditions described above, the attacker will choose $M = 8$ and f taking its values in $\{0;1\}$ (because he does not know which "stuck-at" value is injected). B, K and T are the same sets than those defined in 2.1. S_8, which is the set of all the possible partial sets from B with at most 8 elements and at least one, is constituted of 255 elements. At last, each $r^f_{S^j_8}(k_p, t_i)$ takes the value 0 if all the bits of S^j_8 are equal to f and 1 if one at least has another value, as explained in subsection 1.2.

3.2 Experimentations: faulty simulations

The same kind of faulty simulations as described in 2.1 are done but in this case, "stuck-at" of a unique value p can be injected on a given number q of bits at the output of the S-box during the first round. Note that p and q are the "really" injected values when f and M are the hypothesis made during the computations.

3.3 Case study with $q = 3$ stuck-at wires

Figure 4 shows the results obtained when faulty simulations are made with $q = 3$ arbitrary chosen bits that are stuck at $p = 0$. There are 255 curves on each plot ($f = 0$ and $f = 1$) corresponding to the 255 possible combinations S_8^j of stuck bits on one byte. There are 8 levels of curves, each level corresponding to a number $m < M$ of bits supposed to be stuck.

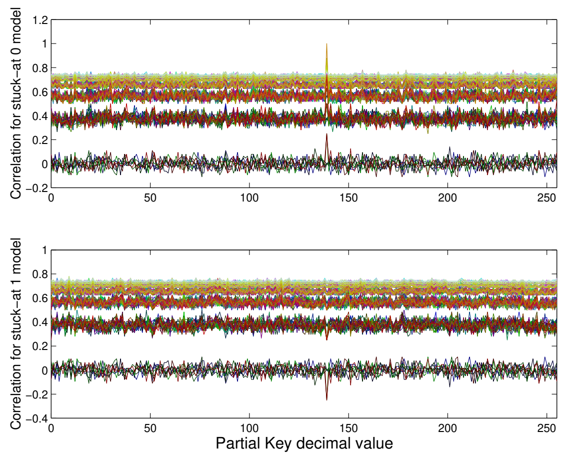

Fig. 4. Multibit DBA with $q = 3$ bits "stuck-at" $p = 0$: upper graph stands for $f = 0$ model and lower for $f = 1$.

On each plot, the 8 lower curves are identical to the curves in single-bit DBA and stand for one stuck bit. Above these are $C_8^2 = 28$ curves standing for two stuck bits. The level of the curves increases with the number of bits stuck m.

The highest peak is obtained for the curve corresponding to the correct fault injected and the correct key value, showing that the DBA is clearly successful.

3.4 Case study with $q > 4$

We performed the same attacks with $q = 5, 6, 7$ and 8 bits stuck at $p = 0$. Some results are shown on Figure 5. In order not to complicate the figures, we only kept one plot (corresponding to $f = 0$) on the following figures.

Note that for $q = 7$ or $q = 8$, wrong key values (at most 3) may appear due to the small number of correct behaviors (2 for 7 stuck bits and 1 for 8).

This study shows that the more bits are stuck during injection campains, the worse signal to noise ratio is ($d_{S_8^j}(T)$ is close to 1).

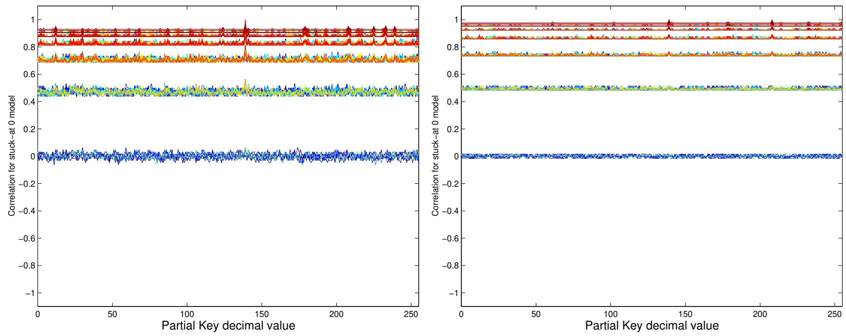

Fig. 5. Multibit DBA with $q = 5$ bits stuck-at $p = 0$ on the left and $q = 7$ on the right.

3.5 Effect of wrong injection

We repeated the method described in paragraph 3.3 but forced wrong values in the simulations of the fault injections. The results obtained for a fault injected on $q = 3$ bits but with probabilities of 10% and 40% wrong values are depicted on Figure 6. It appears, that the worse the injections are, the lower the peaks. But there is also an interesting result: the peaks corresponding to other fault models (and the same partial key) are still high and lead to a good detection until 40% probability of wrong injection. It also appears to be true for a number of stuck bits between 1 and 4.

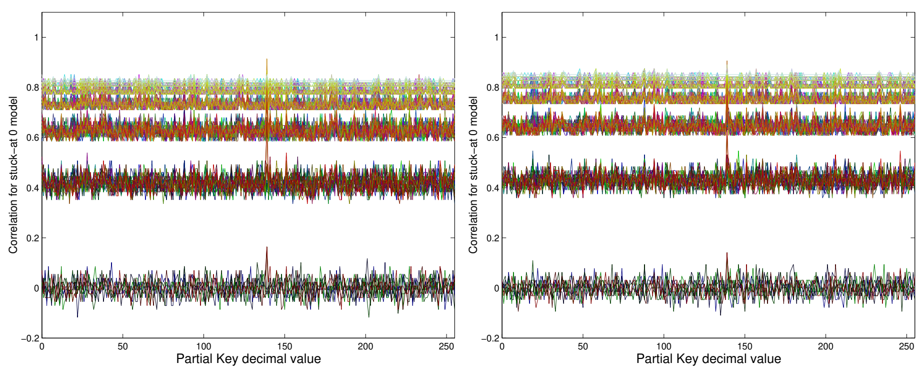

Fig. 6. multibit DBA with $q = 3$ bits “stuck-at” $p = 0$: results with probabilities of 10% (left) and 40% (right) wrong values for the fault injection.

The peak that points out the correct partial key appears also on lower curves. These peaks stand for unperfect fault model but do have a correlation. Detection can thus be made by checking if the peaks for one key value appear for different number of bits stuck. We can thus consider a new way to compute the criterion based on a combination of these curves. Let's consider only one curve for each number of stuck bits, the curves that has the highest peak. The sum of these curves enables to give a better discriminancy to the peak.

For example, the results obtained for a fault injected on $q = 6$ bits, with probabilities of 10% wrong values are depicted on Figure 7-left. It shows that it is difficult to see the correct key because there are only $1/2^6$ of computations that lead to a non-faulty behavior (that is, in that case, only 4 from the whole set of 256 plaintexts). From Figure 7-right, even with 10% injections that don't match the 6-bit stuck model, multibit DBA can retrieve the correct partial key from the whole set of plaintexts.

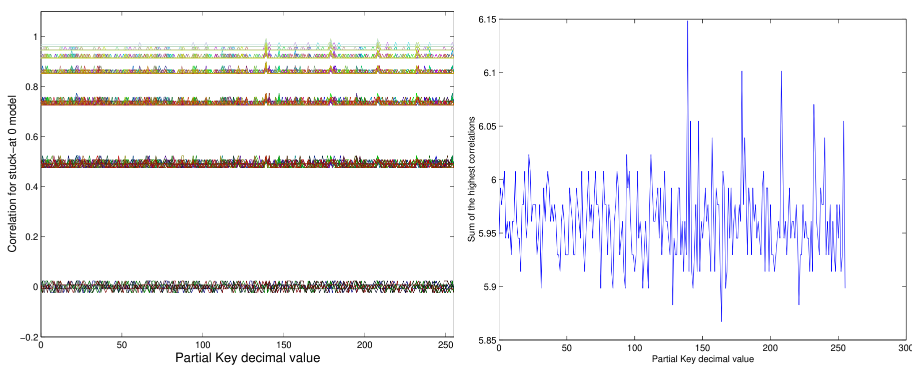

Fig. 7. multibit DBA with $q = 6$ bits "stuck-at" $p = 0$: correlations (left), sum of correlations (right) with a probability of 10% wrong behaviors.

4 Comparison with previous attacks

The claim of DBA is to keep the advantages of DPA and SEA, that is:

- SEA and DBA only exploit the fact that the computation is correctly performed or not, contrary to DFA which needs correct and faulty ciphertexts to retrieve the key.
- SEA and DBA may use the means that are implemented to counteract DFA ([BBKP02], [KWMK02], [KKT04], [MSY05] and even [MRL$^+$06]).
- A DPA or DBA attack on one bit leads to several bits of the key, thanks to the non-linearity of the S-boxes.
- The attacker means to retrieve information about the secret key are not restricted to the "logical" bits targeted by these attacks.
- DBA (resp. DPA) requires few information about the behavior of the circuit in presence of faults (resp. the power consumption of the chip). It only requires that this behavior (resp. this power) depends on the data.

- As DBA and DPA are based on correlation of models to measurements, these two attacks provide introspection *i.e.* the attacker can improve his injection means and its models through experiments.

But contrary to other published SEA, DBA does not require the knowledge of the "stuck-at" value and supports fault injection on several bits.

We also show that DBA is particularly well-suited to attack asynchronous circuits. Some protocols, widely used in such chips, are designed such that the data transfer is controlled by the data themselves. The fault injections on those data thus modify the behavior of the entire circuit (by inducing delay or dead-lock). This property may theoretically be an effective counter-measure against DFA [MR06] but it renders the shape of the power consumption of the chip related to the value of the plaintext (and on the key). This property may unfortunately be used to mount DBA with light changes: permanent "stuck-at" zero (on a wire not a logical value) and the ability to get power consumption measurements. More details can be found in appendix.

Conclusion

We described in this paper an attack on cryptographic devices which mixes the principles of SEA and the probabilistic treatment of DPA. In the paper, the DBA has been validated in simulation on an AES. It appears that the attacker is able to recover the whole secret key with quite realistic means: the fault injection has to be repetitive, has to affect a small number of bits (less than 8) and has to induce a "stuck-at" value of an identical but possibly unknown value.

We also showed that when the attacker is able to inject fault on just one bit, the minimum number of faulty injections is about sixteen in order to recover 8 bits of the AES key. Such results have been obtained when fault injection affects more bits (less than 8) but we noted that the more bits are stuck the worse the signal to noise ratio is.

We also showed that even if fault injection is not strictly repetitive, DBA retrieves the partial key. But the highest the wrong injection rate is, the more plaintexts needed to recover the key are. For example, when the attacker injects fault on just one bit but with 20% wrong injections, he needs around sixty executions to retrieve the 8 bits of the key.

Further work will consist in relaxing again the means of the attacker (especially the assumption concerning the constant value of the "stuck-at") and in applying DBA on real devices. Two crypto-processors (an AES and an asynchronous DES), on which structure we tested the theorical attack, have been designed for this purpose.

Acknowledgements

This work was funded by the CIMPACA/Micro-PackS [CIM] BTRS Project. The authors also would like to thank Michel Agoyan, Jean-Baptiste Rigaud, Julien Francq and Selma Laabidi for their support during the simulations and their useful comments.

References

[ADI] http://cmp.imag.fr/information/gallery/details.php?id_circ=64&y=2005.

[BBKP02] Guido Bertoni, Luca Breveglieri, Israel Koren, and Vincenzo Piuri. Fault detection in the Advanced Encryption Standard. In *Proceedings of MPCS 2002*, Ischia, Italy, 2002.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *CHES*, pages 16–29, 2004.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[BECN+04] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. In *First Workshop on Fault Detection and Tolerance in Cryptography*, Florence, Italy, June 1, 2004.

[BK06] Johannes Blömer and Volker Krummel. Fault based collision attacks on aes. In *FDTC*, pages 106–120, 2006.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In B.S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[BS03] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the Advanced Encryption Standard (AES). In R.N. Wright, editor, *Financial Cryptography – FC 2003*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.

[CIM] http://www.arcsis.org/micro-packaging.0.html.

[CT05] Hamid Choukri and Michael Tunstall. Round reduction using faults. In *FDTC '05: Proceedings of the second Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 13–24, 2005.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer, 2002.

[Gir05] Christophe Giraud. DFA on AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Advanced Encryption Standard – AES*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2005.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. *Lecture Notes in Computer Science*, 1666:388–397, 1999.

[KK99] Oliver Kömmerling and Markus G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology, Chicago, 10–11 May, 1999.*, pages 9–20, 1999.

[KKT04] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Robust protection against fault injection attacks on smart cards implementing the Advanced Encryption Standard. In *2004 International Conference on Dependable Systems and Networks (DSN 2004)*, pages 93–101. IEEE Computer Society, 2004.

[KWMK02] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent error detection scheme for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Transactions on Computer-Aided Design*, 21(12):1509–1517, 2002.

[MR06] Yannick Monnet and Marc Renaudin. Designing resistant circuits against malicious faults injection using asynchronous logic. *IEEE Trans. Comput.*, 55(9):1104–1115, 2006. Member-Regis Leveugle.

[MRL+06] Yannick Monnet, Marc Renaudin, Régis Leveugle, Christophe Clavier, and Pascal Moitrel. Case study of a fault attack on asynchronous des crypto-processors. In *FDTC*, pages 88–97, 2006.

[MSY05] Tal G. Malkin, François-Xavier Standaert, and Moti Yung. A comparative cost/security analysis of fault attack countermeasures. In *Second Workshop on Fault Detection and Tolerance in Cryptography*, pages 109–123, Edinburgh, UK, September 2, 2005.

[NIS01] NIST. Announcing the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication, n. 197, November 26, 2001.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and Khazad. In C.D. Walter, editor, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

A DBA applied to an asynchronous DES

We have designed and fabricated an integrated asynchronous circuit which implements the DES algorithm. All the blocks communicate thanks to the four phase RTZ (Return To Zero) protocol; the data are dual-rail encoded and the invalid or NULL state is "00". The targeted technology was the 0.13 μm from STMicroelectronics. The circuit is 0.94 mm^2 large (with a serial interface and an synchronous/asynchronous interface) and it computes a DES encryption (or decryption) in 180 ns. More information about the chip are available at [ADI].

Simulations of the DES behavior in the presence of faults are performed with Modelsim simulator on the post-place-and-route simulation model. We record the switching activity of the chip at each simulation steps. This count is a rough estimation of the power consumption of the chip. The fault, a permanent "stuck-at" zero, is injected on a wire of a dual rail (at the output of the S-box) by using the command *force* of the simulator. It appears that when a "stuck-at" zero fault is applied on one wire of a dual-rail, the behavior of the circuit depends on the expected value of the wire : if the stuck wire was to transmit a zero, the circuit's behavior is unchanged for the first round. On the contrary, if this wire was to transmit a one, the data on the rail remains invalid and the circuit stops. In other words, the chip stops its computation after a time which depends on the value of the plaintext (and of the key). In order to illustrate this claim, power estimation curves are depicted on Figure 8 in the case when circuit functions normally (left), in the case when circuit stops during round 1 (middle) and during round 2 (right).

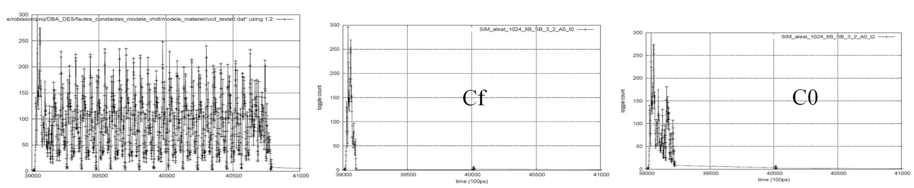

Fig. 8. Power estimation of the chip when it computes normally (left) and when fault injection stops the chip during round 1 (middle, behavior Cf) and during round 2 (right, behavior C0).

As the power consumption of an asynchronous chip is representative of the circuit's activity, the attacker is able to distinguish C0 and Cf only by analyzing the power consumption of the chip: if the wire that was "stuck-at" zero was to transmit a 0 during the first round, the consumption peak during this round appears normally (behavior C0) even if the chip will probably stop later; if this wire was to transmit a 1, the chip's consumption falls to zero before the end of the first round (behavior Cf). This simple power analysis allows the attacker to mount DBA as described in section 1.