Secret External Encodings Do not Prevent Transient Fault Analysis

Christophe Clavier

Gemalto, Security Labs,
La Vigie, Avenue du Jujubier, ZI Athélia IV,
F-13705 La Ciotat Cedex, France
christophe.clavier@gemalto.com

Abstract. Contrarily to Kerckhoffs' principle, many applications of today's cryptography still adopt the *security by obscurity* paradigm. Furthermore, in order to rely on its proven or empirical security, some realizations are based on a given well known and widely used cryptographic algorithm. In particular, a possible design would obfuscate a standard block cipher `E` by surrounding it with two *secret* external encodings $\mathtt{P}_1$ and $\mathtt{P}_2$ (one-to-one mappings), leading to the proprietary algorithm $\mathtt{E'} = \mathtt{P}_2 \circ \mathtt{E} \circ \mathtt{P}_1$.
A claimed advantage of this approach is that, since inputs and outputs of the underlying function `E` are not known by a potential attacker, such a construction is usually believed to inherently prevent any kind of transient fault analysis that may apply on the core function `E`. In this paper, we show that this latter argument is not true, by exhibiting a key recovery attack which applies to the whole class of externally encoded DES or Triple-DES. Moreover, our attack remains applicable even in the presence of the classical counter-measure against fault attacks which consists in executing the algorithm twice and returning an output only if both results are identical.

Keywords: Smart Cards, Physical Attacks, Fault Analysis, Secret Algorithm, Cryptographic Design, External Encoding, DES

1 Introduction

Contrarily to Kerckhoffs' principle, many applications of today's cryptography still adopt the *security by obscurity* paradigm. For public and civil applications, this is especially true in GSM and pay-TV domains where the specifications of the cryptographic function are often kept secret. A usually claimed advantage of concealing the cryptographic function's details is to protect against known physical attacks such as side-channel analysis (SPA, DPA, CPA,. . .) [2, 10] or fault analysis (DFA, CFA,. . .) [1, 4, 5, 7], which would otherwise be used to reveal the user's secret key.

The main result of this paper is to partially invalidate this belief. More precisely, we focus on a particular way of designing such a proprietary algorithm which consists in surrounding a well known and widely used block cipher E with two *secret* external encodings P_1 and P_2 (one-to-one mappings over the input and output spaces), leading to the new secret obfuscated block cipher $E' = P_2 \circ E \circ P_1$. The motivation for such a design strategy is twofold. First, it seems reasonable to base the construction on a well known block cipher E in order to inherit its proven or empirical cryptographic strength. Second, the two secret encodings P_1 and P_2 ensure that inputs to and outputs from E can not be known by an attacker, so that physical attacks requiring this knowledge should not be feasible.

In this paper, we present a fault-based key recovery attack which applies to this design when the core block cipher E is the DES or the Triple-DES. Our attack works for any P_1 and P_2, so that the whole class of externally encoded (Triple-)DES (c.f. Figure 1) is potentially endangered[1].

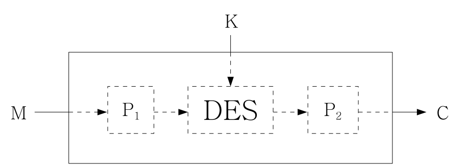

Fig. 1. A DES obfuscated by secret layers P_1 and P_2.

In Section 2 we present the previous work related to fault analysis of secret cryptographic functions, as well as our threat model and the conditions needed for the attack. Section 3 gives two variants of our attack: first, a basic version that illustrates the main principles is explained; then, an improved attack is described and simulation results are presented. Possible counter-measures are discussed in the next section. Finally, Section 5 concludes this work and proposes some possible directions for further research.

[1] As a particular case where P_1 and P_2 are XOR-maskings with external keys, our attack notably applies to the DESX construction [9] and allows to recover its internal secret key.

2 Preliminaries

Boneh *et al.* first introduced in [5] the use of transient computational errors as a means to extract secret keys of cryptographic algorithms. Their attack applies to RSA in CRT mode and has been shortly followed by similar results applicable to the DES algorithm [4] and other functions. These methods all rely on the fact that the cryptographic algorithm is public. In [3], Biham and Shamir tackled the *unknown cryptosystem* case and proposed an attack based on the assumption that it is possible to permanently and progressively reset bits of the key stored in a non-volatile memory. This technique has been later improved and extended by Paillier in [12]. Since it requires permanent faults, this model is quite demanding. Moreover, definitively damaging the device under attack may be undesirable. Nevertheless, as far as we know, no transient fault attack on unknown cryptosystems has ever been published. Our proposed attack precisely attains this goal under the following assumptions:

The target is a classical software implementation of the DES on an 8-bit architecture. We assume that the attacker is able to precisely control which instruction is executed when he injects a fault.

Concerning the fault model, we assume that a fault injected during the execution of a XOR between two 8-bit operands results in a zero[2] output whatever the input operand values were. Let us mention that this fault model is realistic as we identified some chips vulnerable to this kind of faults on which we practically performed attacks relying on it.

Finally, the attacker is supposed to have control over the input given to the encryption function `E'` as well as knowledge of its output[3].

Compared to fault analysis on public cryptosystems, our attack needs a large number (many thousands) of fault injections. We see this drawback as the fair price to pay for the 'magic' property of being able to retrieve the key regardless of the two secret external encodings P_1 and P_2.

3 Ineffective Fault Analysis

3.1 Fault Injection as a Probing Tool

Our attack, described in Sections 3.2 and 3.3, is based on the main observation that a fault injection capability may be used as a probing tool.

[2] Note that our attack works equally well if the faulted XOR output is supposed to be any arbitrary known constant instead of zero.

[3] These assumptions may be relaxed. It is only required to be able to replay many times different arbitrary inputs, and to detect whether two outputs are equal.

More precisely, if an attacker targets a particular XOR instruction in the algorithm, then he is able to detect whether the output of this instruction is equal to zero or not. For some arbitrary input, if the output of the algorithm when a fault is injected during the targeted XOR is the same as that of a normal execution, this indicates that the natural value of the XOR result is zero[4]. Some information about an intermediate value is thus obtained by observing two equal outputs of the algorithm. Equivalence between outputs happens when the injected fault has *no effect* on the targeted instruction and its intermediate result. This event being the most informative one exploited in our attack, we call such kind of attack an *Ineffective Fault Analysis* (IFA). Though it is rather similar to *safe-error analysis* ([8, 13, 14]), IFA is slightly different since the fault targets a true instruction, whose output is possibly not modified, rather than a fake instruction. In the case of IFA, the event of an unchanged algorithm output results from a *data* related condition, whereas it is *algorithm* specific in safe-error analysis.

In the context of this paper, for any given plaintext, IFA allows an attacker to detect whether the output of any arbitrary XOR of the embedded DES is zero.

3.2 The Basic Attack

We refer to [11] for a complete description of the DES algorithm. Nevertheless, and before describing the attack in detail, we remind the reader that the DES key schedule is structured in such a way that the key may be considered as partitioned into two 28-bit half-keys, which we respectively denote K^A and K^B. During any round, the 24 key bits involved in S-boxes 1 to 4 are a subset of K^A, while the 24 key bits involved in S-boxes 5 to 8 belong to K^B. Even though our attack does *not* rely on this property, we will take advantage of it to computationally simplify the faults exploitation by considering the two half-key spaces separately.

We assume a straightforward implementation of the DES on an 8-bit architecture. In this typical implementation, there are 12 XOR operations per round. As shown in Figure 2, for each round $h = 2, \ldots, 16$, we are concerned with two groups of XOR instructions. The first group, made up of four so-called `xor_left` instructions executed at the end of round $(h-1)$, computes the four bytes $(r_1, \ldots, r_4)$ of the 32-bit value R_h which enters the round h. Then $(r_1, \ldots, r_4)$ is expanded into eight 6-bit values

[4] Or at least that this value is equivalent to zero through the remainder of the algorithm. This comment will become clearer in the example given in Section 3.2.

$(s_1, \ldots, s_8)$ which are XOR-ed with the round key $K_h = (k_1, \ldots, k_8)$, through the eight so-called `xor_key` instructions, to provide the S-box inputs $(x_1, \ldots, x_8)$. Each 4-bit S-box output is computed as $y_j = S_j(x_j)$.

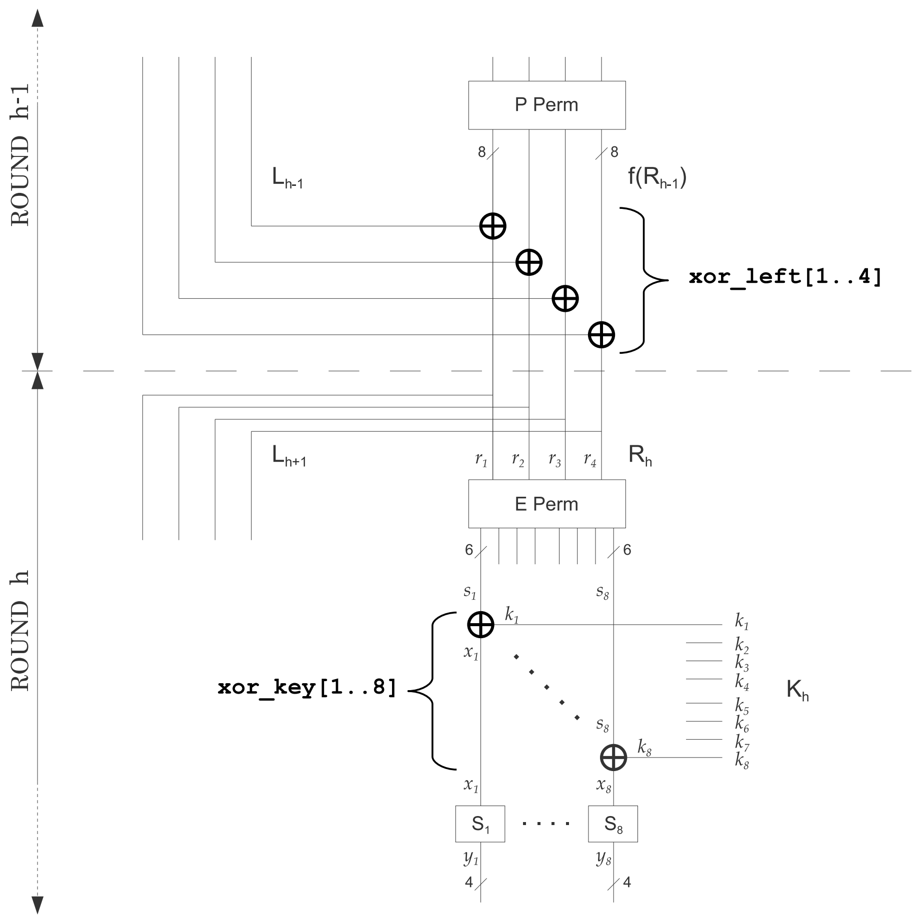

Fig. 2. The 12 XOR instructions per round targeted by the attack.

The central idea in this basic attack is to infer information about the key from couples of two ineffective faults on two different executions with the same input. While information about two intermediate values of some computation is so obtained, we stress that our attack does not require the ability to inject multiple faults on the same execution.

First, suppose that for some plaintext M, a fault injected during `xor_left`[i] (for $i \in \{1, \ldots, 4\}$) at round $(h-1)$ turns out to be ineffec-

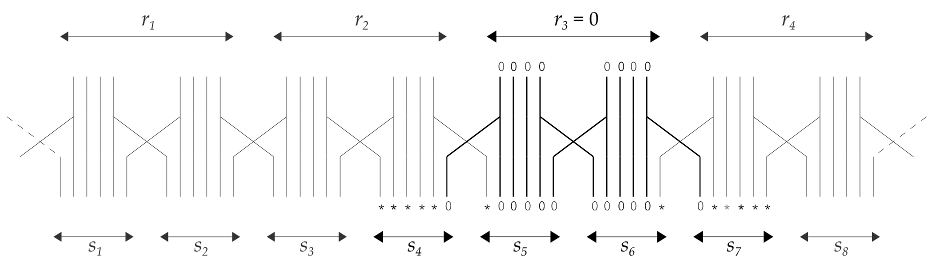

Fig. 3. A zero byte through the expansive permutation.

tive. This implies that the corresponding output byte r_i is zero. Thus, 8 of the 32 bits of R_h are known to be 0. The following permutation expands them to 12 bits which are involved in four adjacent[5] S-boxes at round h. Now, suppose that for another execution with the same plaintext M, a fault on `xor_key[j]` (for $j \in \{2i-1, 2i\}$) at round h turns out also to be ineffective. In this setting, we show that some valuable information about the 6-bit k_j may be inferred. This is the basic principle behind our attack. We now give an example of this reasoning:

Example: For some M, an ineffective fault on `xor_left[3]` at round $(h-1)$ gives $r_3 = 0$. Figure 3 shows that when $r_3 = 0$, the 6-bit inputs s_4 to s_7 of `xor_key[4]` to `xor_key[7]` of the next round belong to $(*,*,*,*,*,0)$, $(*,0,0,0,0,0)$, $(0,0,0,0,0,*)$ and $(0,*,*,*,*,*)$ respectively. Suppose that, for the same M, a fault on `xor_key[5]` at round h appears also to be ineffective. One could first conclude that the `xor_key[5]` output x_5 is equal to 0. But actually this rather means that x_5 belongs to the set $\mathcal{A}_5 = \{(0,0,0,0,0,0),(0,0,0,1,0,1),(1,0,0,0,1,0),(1,0,1,1,0,1)\}$ of the four pre-images of $S_5(0)$ by S_5. This is due to the non-injective property, for each S-box, that any 4-bit output has exactly four pre-images. We can now derive that $k_5 = x_5 \oplus s_5 \in \mathcal{A}_5 \oplus (*,0,0,0,0,0)$, which leads to 8 possible values for k_5 corresponding to 3 bits of information retrieved about the key K.

With the same reasoning, an output identity when faulting `xor_key[6]` would imply that $k_6 \in \mathcal{A}_6 \oplus (0,0,0,0,0,*)$, revealing 3 other bits of information about the key. Note that for the other two S-boxes (S_4 and S_7),

[5] One consider the eight S-boxes form a ring. For example, S-boxes 8, 1, 2 and 3 are adjacent.

it is not possible to determine the value of neither the right-most bit of k_4 nor the left-most bit of k_7[6].

Definition 1 (Winning event). *We call* winning event *at locus* (h, i, j) *a pair of observations, for the same plaintext, of two ineffective faults: one on* `xor_left[i]` *at round* $(h-1)$*, and another one on* `xor_key[j]` *at round* h*, where* $j \in \{2i-1, 2i\}$*.*

Winning events such as the one at locus $(h, 3, 5)$ described in the previous example are the core events exploited in this attack.

Obtaining a winning event at some locus obviously depends on the plaintext. Indeed, the values of $L_{h-1}[i]$ and $R_{h-1}[i]$ which govern the (in-)effectiveness of a fault on `xor_left[i]`, as well as the value of the '$*$' bit of s_j which influences the (in-)effectiveness of a fault on `xor_key[j]`, all depend on the plaintext. So if a winning event at some locus is not obtained for a given plaintext, it may well be obtained for another one. Nevertheless, for a winning event at some locus to be obtained, the key bits corresponding to the five '0' bits of s_j must be equal to their corresponding values in one representative of $\mathcal{A}_j$. Consequently, given a key K, there are some locus at which no winning event may occur whatever the plaintext, and others at which winning events occur for some plaintexts.

Definition 2 (Winnable locus). *Given some key* K*, we say that* (h, i, j) *is a* winnable locus *if the five bits of* k_j *at '0' positions (those where an ineffective fault on* `xor_left[i]` *at round* $(h-1)$ *implies a bit value of* s_j *equal to 0), are equal to their counterpart values in one of the four representatives of* $\mathcal{A}_j$*.*

Example: For K = `CD3ABC5876AC062B`, locus $(7, 3, 5)$ is winnable because the subkey k_5 entering S-box 5 at round 7 is equal to $(1, 0, 0, 1, 0, 1)$ whose five rightmost bits are equal to those of $\mathcal{A}_5$'s element $(0, 0, 0, 1, 0, 1)$.

The probability (over all keys) for any given locus to be winnable is $4 \cdot 2^{-5} = 0.125$. Furthermore, there are $8 * (16 - 1) = 120$ interesting loci along the DES (the first round is not exploitable), so that, in the simplified model where all K_h are viewed as almost independent, 15 winnable loci per key are expected on average. A counting simulation on 27 000 randomly generated keys gives a number of winnable loci distributed as shown in Figure 4, with an average of 14.986.

[6] This is due to the fact that, by design of any S-box S_j, each lateral bit is always represented with both 0 and 1 values amongst $\mathcal{A}_j$.

For each winnable locus, and whenever a winning event is obtained, the key space may be reduced according to the previously explained constraint. The optimal residual entropy, obtained after having exploited all winnable loci, is distributed as shown in Figure 5. The percentiles of this distribution for the frequency levels $(0.10, 0.50, 0.90)$ are $(14.17, 21.32, 29.98)$, meaning that for one key out of two, the full exploitation of winnable loci reduces the key space from 2^{56} to less than $2^{21.32}$ keys.

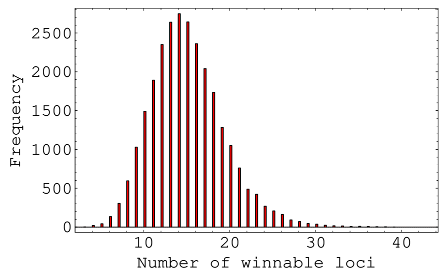

Fig. 4. Number of winnable loci per key.

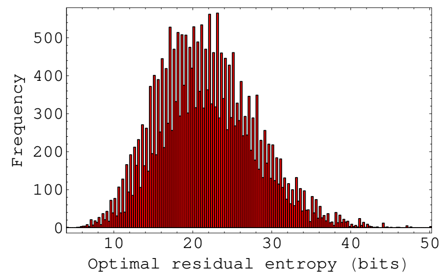

Fig. 5. Optimal residual entropy per key after exploitation of all winnable loci.

In Appendix A we summarize the procedure of the attack as described in this section. The stopping condition is left to the attacker: it may involve the number of faults injected so far, the current residual entropy of the key space, or other considerations.

A simulation of this attack according to this procedure allowed us to quantify the number of fault injections required. Out of 27 000 experiments, the number of faults needed for winning all possible winnable loci is distributed as shown in Figure 6. With 100 000 faults, all winnable loci are won in 54.5 % of cases, and all but at most 1 are won in 87.3 % of cases. As the number of already won loci increases, the probability to win another one strongly decreases. This suggests that there is no point in continuing faulting for a long time. The median residual entropy after 50 000 and 100 000 faults is respectively 26.49 and 22.32 bits.

3.3 An Improved Version of the Attack

The attack described in Section 3.2 essentially eliminates keys which are not compatible with any observed winning event. In this section we present a modified version which improves on it in two directions.

First, we extend the kind of events which are exploited. For example, when a winning event at locus $(h, 3, 5)$ is observed, and for the same M

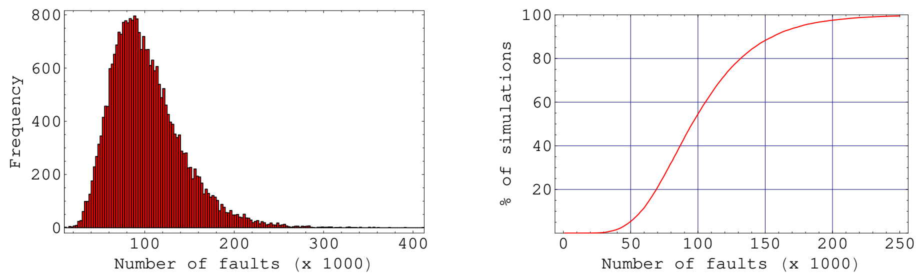

Fig. 6. Number of faults needed for winning all winnable loci.

the attacker knows that $r_2 = 0$ (this is the case if a fault on `xor_left[2]` at round $(h-1)$ is ineffective), then all 6 bits of k_5 are constrained instead of 5. This results in an extra information about the key which was not exploited by the basic attack. As another example, suppose that an ineffective fault is observed on `xor_left[3]` at round $(h-1)$, but not on `xor_key[5]` at round h. Even if this is not a winning event at locus $(h, 3, 5)$, we may infer information about k_5, namely that k_5 does *not* belong to $\mathcal{A}_5 \oplus (*, 0, 0, 0, 0, 0)$. Here also, this informative event was not considered in the basic attack.

The second improvement consists in assigning an *a posteriori* probability to each key, conditioned by the observations.

The result of these two improvements is that, not only the space of compatible keys is further reduced, but also its exhaustive search is shortened by trying keys in their decreasing order of probability.

Definition 3 (Ineffectiveness Vector). *We call* ineffectiveness vector *at round h, denoted $\mathbf{e} = (\mathbf{e}_{left}, \mathbf{e}_{key})$, the boolean vector $\mathbf{e}_{left}$ of the observed ineffectiveness of faults injected on `xor_left[1]` to `xor_left[4]` at round $(h-1)$, together with the boolean vector $\mathbf{e}_{key}$ of the observed ineffectiveness of faults injected on `xor_key[1]` to `xor_key[8]` at round h.*

For example, the winning event at locus $(h, 3, 5)$ described in Section 3.2 may have been produced by the ineffectiveness vector $(\mathbf{e}_{left}, \mathbf{e}_{key})$, where $\mathbf{e}_{left} = (0, 0, 1, 0)$, and $\mathbf{e}_{key} = (0, 0, 0, 0, 1, 0, 0, 0)$.

For each $\sigma \in \{A, B\}$, let $\mathbf{e}^{\sigma}_{key}$ denote that part of $\mathbf{e}_{key}$ related to the four S-boxes involving K^{σ} (so that $\mathbf{e}_{key} = (\mathbf{e}^{A}_{key}, \mathbf{e}^{B}_{key})$), and $\mathbf{e}^{\sigma}$ denote $(\mathbf{e}_{left}, \mathbf{e}^{\sigma}_{key})$.

Any observed ineffectiveness vector may be used to assign an *a posteriori* probability to each half-key K^{σ} by means of Bayes' formula:

$$p(K^\sigma|\mathbf{e}^\sigma) = p(\mathbf{e}^\sigma|K^\sigma) \cdot \frac{p(K^\sigma)}{p(\mathbf{e}^\sigma)} \tag{1}$$

From Eq. (1), we derive a recursive form which allows to update the *a posteriori* probability of a key, based on a newly observed ineffectiveness vector:

$$p\Big(K^\sigma|(\mathbf{e}_1^\sigma, \ldots, \mathbf{e}_n^\sigma)\Big) = \frac{p(\mathbf{e}_n^\sigma|K^\sigma)}{p(\mathbf{e}_n^\sigma)} \cdot p\Big(K^\sigma|(\mathbf{e}_1^\sigma, \ldots, \mathbf{e}_{n-1}^\sigma)\Big) \,. \tag{2}$$

Note that evaluating the denominator $p(\mathbf{e}_n^\sigma)$ is not necessary as it is independent from the key. With the aim of comparing key probabilities together, omitting it will only affect all probabilities by the same multiplicative factor. Thus, while considering a new observation $\mathbf{e}^\sigma$, the process of updating key probabilities just comes down to multiplying the (not normalized) probability of each key K^σ by $p(\mathbf{e}^\sigma|K^\sigma)$.

Assuming a random behavior for R_h, evaluating $p(\mathbf{e}^\sigma|K^\sigma)$ is done by counting the number of round inputs compatible with the observation. Indeed, we have:

$$p(\mathbf{e}^\sigma|K^\sigma) = \frac{\#\{R_h : \mathbf{e}_{left} \text{ and } \mathbf{e}_{key}^\sigma \text{ are satisfied when } K^\sigma \text{ is used}\}}{2^{32}} \tag{3}$$

Note that this counting operation may be optimized as $\mathbf{e}_{key}^\sigma$ depends on only 18 bits of R_h.

The procedure given in Appendix B describes a way to implement this improved attack. We decided to exploit an ineffectiveness vector $\mathbf{e}^\sigma = (\mathbf{e}_{left}, \mathbf{e}_{key}^\sigma)$ only when at least one of its two most influential `xor_left` instructions shows to be ineffective under fault. Four `xor_key` fault injections are thus saved in cases where a negligible amount of information would have been gathered.

We performed extensive simulations of this attack with different numbers of faults ranging from 15 000 to 100 000. In each case, 10 000 simulations were done. The residual entropy with respect to different percentile levels and for each considered number of faults is given in Table 1. The median residual entropy for 50 000 and 100 000 faults are 13.95 and 6.68 bits. Compared to corresponding figures of Section 3.2, this demonstrates a considerable gain for this method over the basic attack. Figure 7 provides a graphical view of the decreasing entropy of the resulting key space.

Table 1. Percentiles of the residual entropy (in bits) as a function of the number of injected faults

Number of faults	Percentile level 5 %	10 %	25 %	50 %	75 %	90 %	95 %
15 000	23.59	26.33	30.98	36.10	40.46	43.92	46.37
25 000	14.35	16.92	21.51	26.62	31.63	35.86	38.31
35 000	9.17	11.27	15.38	20.23	25.2	29.60	32.31
50 000	5.13	6.80	9.85	**13.95**	18.65	22.96	25.64
70 000	2.81	3.93	6.23	9.57	13.57	17.44	19.95
100 000	1.40	2.26	4.03	**6.68**	10.07	13.59	15.87

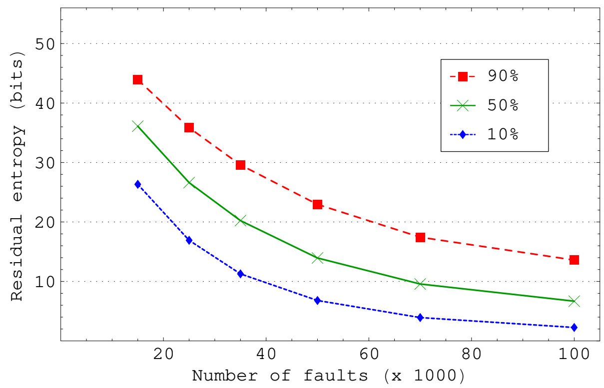

Fig. 7. Percentiles of the residual entropy as a function of the number of faults.

4 Countermeasures

Having explained our attack in Sections 3.2 and 3.3, we now analyze the conditions for this attack to be feasible, and the countermeasures which may prevent it.

As we already mentioned, the embedded DES we attack must be implemented in software. We think that the proposed attack is not applicable when using a DES co-processor. We also relied on an 8-bit architecture. This condition is not strictly required but it greatly impacts the complexity of the attack. For example, if we have a 16-bit architecture, the expected number of faults needed before obtaining an ineffective one when targeting a `xor_left` instruction would be 2^{16} instead of 2^8. The complexity figures we mentioned for the 8-bit case would thus become prohibitive for a practical realization on architectures with wider data paths.

Because the attacker needs to know which instruction is corrupted when a fault is injected, we think that the classical *random delays* countermeasure (either software or hardware) should prevent the attack, or at least make its realization very difficult. Indeed, an important condition is to be able to interpret an identity of outputs as being the consequence of a natural zero output of the targeted XOR. If random delays exist, this rare particular event may well be lost in many false positive neutral faults. The problem of false-negatives also exists when random delays are implemented.

For similar reasons, the *random order* countermeasure will also perturb the attacker. Nevertheless, and while we have not further investigated this idea, we foresee a way to adapt the attack to this case. When this countermeasure is implemented alone, and by repeatedly injecting faults on the same `xor_left` (resp. `xor_key`) instruction for the same input, the attacker is able to infer the number of `xor_left[`i`]` (resp. `xor_key[`j`]`) for which a fault is ineffective. The observation obtained by the attacker is not the complete ineffectiveness vector $(\mathbf{e}_{left}, \mathbf{e}_{key})$ anymore, but rather the Hamming weights of $\mathbf{e}_{left}$ and $\mathbf{e}_{key}$. Probably at the cost of a larger number of needed faults, we think that it should still be possible to assign probabilities to keys based upon this partial information about the ineffectiveness vector.

A classical counter-measure against side-channel attacks such as SPA, DPA, CPA, ... is the *data masking* (also called *blinding*) [6] which results, when correctly implemented, in a perfect first-order unpredictability of intermediate values. A direct consequence of this property is that the attack we described is not possible anymore: any ineffective fault, consequence of a *physical* zero value of the XOR output on the faulted execution, is compatible with any *logical* masked value and gives no useful information to the attacker. Note that this first-order anti side-channel countermeasure is not effective against a variant of our attack if it is possible to inject multiple faults at chosen timings on the same execution.

Finally, we consider the classical countermeasure against DFA and CFA which consists in computing the cryptographic function twice and comparing both results: if both results are the same, the value is output by the command; if they differ, a fault is detected and no ciphertext is returned. As already noticed in [13] for general safe-error attacks, we emphasize that this countermeasure does *not* impede our attack. A valid output indicates that the fault was ineffective, while no output means that it was not. The attack is even slightly simplified as the attacker does not need to ask for computations without fault. The counter-measure would remain efficient if a limit is imposed on the number of allowed detected faults.

5 Conclusion

We have presented a fault-based key recovery attack on a software implemented DES. This attack relies on the following fault model: when a fault is injected during a XOR instruction, the output of this XOR is forced to zero whatever the input operand values were. A large amount of information about the secret key K is retrieved without knowing the DES input, nor its output. Only the ability to detect that two DES outputs are equal is required. An important consequence is that our attack applies to the whole class of externally encoded DES or Triple-DES[7], defined as secret block ciphers built by embedding a DES or a Triple-DES between two arbitrary secret permutations. This is particularly meaningful as it potentially endangers proprietary cryptographic algorithms based on this obfuscating design and invalidates the supposed immunity of these secret functions against fault analysis. As far as we know, our attack is the first published example of transient fault analysis against a class of secret cryptographic functions.

Finally, we suggest some possible directions to extend our result. Further investigations could aim at designing a variant of this attack that would rely on other realistic fault models, or that would apply to other externally encoded block ciphers. For example, a similar result applicable to an externally encoded AES would threaten the most common usage of the MILENAGE [15] scheme for authentification and key generation functions.

Acknowledgements

The author would like to thank Eric Brier and Benoît Chevallier-Mames for fruitful discussions related to the ideas presented in this paper.

The work described in this document has been financially supported by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

References

1. F. Amiel, C. Clavier and M. Tunstall. Fault Analysis of DPA-Resistant Algorithms. In *Fault Diagnosis and Tolerance in Cryptography, FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 223–236. Springer-Verlag, 2006.

[7] The Triple-DES case is treated by applying the attack on K_1 and K_2 separately.

2. E. Brier, C. Clavier and F. Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag, 2004.
3. E. Biham and A. Shamir. The Next Stage of Differential Fault Analysis: How to break completely unknown cryptosystems. 30 October 1996 (draft). Available at `www.fit.vutbr.cz/~cvrcek/cards/nextstage.ps`
4. E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Advances in Cryptology – CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag, 1997.
5. D. Boneh, R.A. DeMillo and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology – EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.
6. L. Goubin and J. Patarin. DES and Differential Power Analysis (The 'Duplication' Method). In *Cryptographic Hardware and Embedded Systems - CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer-Verlag, 1999.
7. L. Hemme. A Differential Fault Attack Against Early Rounds of (Triple-)DES. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 254–267. Springer-Verlag, 2004.
8. M. Joye, J.-J. Quisquater, S.-M. Yen and M. Yung. Observability Analysis: Detecting When Improved Cryptosystems Fail. In *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 263–276. Springer-Verlag, 2002.
9. J. Kilian and P. Rogaway. How to protect DES against exhaustive key search. In *Advances in Cryptology – CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 252–267. Springer-Verlag, 1996.
10. P. Kocher, J. Jaffe and B. Jun. Differential Power Analysis. In *Advances in Cryptology – CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
11. National Bureau of Standards. Data Encryption Standard. *Federal Information Processing Standard* 46, 1977.
12. P. Paillier. Evaluating Differential Fault Analysis of Unknown Cryptosystems. In *Public Key Cryptography 1999*, volume 1560 of *Lecture Notes in Computer Science*, pages 235–244. Springer-Verlag, 1999.
13. S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, **49**(9):967-970, 2000.
14. S.-M. Yen, S.-J. Kim, S.-G. Lim and S.-J. Moon. A Countermeasure Against One Physical Cryptanalysis May Benefit Another Attack. In *Information Security and Cryptology, ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 414–427. Springer-Verlag, 2002.
15. 3GPP TS 35.206. Specification of the MILENAGE algorithm set: An example algorithm Set for the 3GPP Authentication and Key Generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification. Available at `http://www.3gpp.org/ftp/Specs/html-info/35206.htm`

A Basic Algorithm

Algorithm 1 The Basic Attack

```
while stopping condition is not satisfied do
    Pick a plaintext M at random
    C ← E(M, K)
    for h from 2 to 16 do
        for i from 1 to 4 do
            C* ← E(M, K) with fault on xor_left[i] at round h − 1
            if C* = C then
                for j from 2i − 1 to 2i do
                    C* ← E(M, K) with fault on xor_key[j] at round h
                    if C* = C then
                        Reduce the relevant half-key space (K^A or K^B)
                         according to the winning event (h, i, j)
                    end if
                end for
            end if
        end for
    end for
end while
```

B Improved Algorithm

Algorithm 2 The Improved Attack

For each 2^{28} possible K^A, set proba$(K^A) \leftarrow 1$
For each 2^{28} possible K^B, set proba$(K^B) \leftarrow 1$
while stopping condition is not satisfied **do**
    Pick a plaintext M at random
    $C \leftarrow \mathtt{E}(M, K)$
    for h from 2 to 16 **do**
        for i from 1 to 4 **do**
            $C^* \leftarrow \mathtt{E}(M, K)$ with fault on `xor_left[`i`]` at round $h - 1$
            $\mathbf{e}_{left}[i] \leftarrow (C^* \stackrel{?}{=} C)$
        end for
        if $(\mathbf{e}_{left}[1] = \text{True})$ or $(\mathbf{e}_{left}[2] = \text{True})$ **then**
            for j from 1 to 4 **do**
                $C^* \leftarrow \mathtt{E}(M, K)$ with fault on `xor_key[`j`]` at round h
                $\mathbf{e}^A_{key}[j] \leftarrow (C^* \stackrel{?}{=} C)$
            end for
            for all K^A such that proba$(K^A) > 0$ **do**
                proba$(K^A) \leftarrow$ proba$(K^A) \cdot p((\mathbf{e}_{left}, \mathbf{e}^A_{key})|K^A)$
            end for
        end if
        if $(\mathbf{e}_{left}[3] = \text{True})$ or $(\mathbf{e}_{left}[4] = \text{True})$ **then**
            for j from 1 to 4 **do**
                $C^* \leftarrow \mathtt{E}(M, K)$ with fault on `xor_key[`$j + 4$`]` at round h
                $\mathbf{e}^B_{key}[j] \leftarrow (C^* \stackrel{?}{=} C)$
            end for
            for all K^B such that proba$(K^B) > 0$ **do**
                proba$(K^B) \leftarrow$ proba$(K^B) \cdot p((\mathbf{e}_{left}, \mathbf{e}^B_{key})|K^B)$
            end for
        end if
    end for
end while