Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output

Benedikt Gierlichs[1], Jörn-Marc Schmidt[2], and Michael Tunstall[3]

[1] KU Leuven Dept. Electrical Engineering-ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
benedikt.gierlichs@esat.kuleuven.be
[2] Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
joern-marc.schmidt@iaik.tugraz.at
[3] Department of Computer Science, University of Bristol
Merchant Venturers Building, Woodland Road
Bristol BS8 1UB, United Kingdom
tunstall@cs.bris.ac.uk

Abstract. Implementation attacks pose a serious threat for the security of cryptographic devices and there are a multitude of countermeasures that are used to prevent them. Two countermeasures used in implementations of block ciphers to increase the complexity of such attacks are the use of dummy rounds and redundant computation with consistency checks to prevent fault attacks. In this paper we present several countermeasures based on the idea of infective computation. Our countermeasures ensure that a fault injected into a cipher, dummy, or redundant round will infect the ciphertext such that an attacker cannot derive any information on the secret key being used. This has one clear advantage: the propagation of faults prevents an attacker from being able to conduct any fault analysis on any corrupted ciphertexts. As a consequence, there is no need for any test at the end of an implementation to determine if a fault has been injected and a ciphertext can always be returned.

Keywords: Implementation Attacks, Dummy Rounds, Infective Computation.

1 Introduction

Implementation attacks are currently one of the most powerful threats for cryptographic devices. Instead of considering a cryptographic device like a smart card as a black box, these attacks try to benefit from characteristics of the implementation, either by measuring properties of the device during a computation or by actively manipulating the execution of an algorithm. These can be grouped into two main types: Passive attacks based on measuring and analyzing properties

of the device (referred to as a side-channel attack), and active attacks where an attacker seeks to modify the behavior of a device (referred to as a fault attack). Both attack methods may reveal cryptographic keys used inside a device, irrespective of the theoretical (black-box) security of the underlying cryptographic algorithm.

Side-channel analysis was introduced to the cryptographic community by Kocher [1], who noted that the time taken to compute a cryptographic algorithm can potentially reveal the cryptographic key being used. The idea of the attack was later extended by Kocher et al. [2], who noted that the power consumption of a microprocessor depends on the code being executed. Moreover, the instantaneous power consumption is dependent on the data being manipulated at that point in time (typically proportional to the Hamming weight of the data [3]). This allows information on cryptographic keys to be determined, since one can verify a hypothetical set of values that occur after the input is combined with a key. It was later observed that the same attacks could be applied to the electromagnetic emanations surrounding a microprocessor [4, 5]. A suitable countermeasure to such attacks is to mask all intermediate states of an algorithm with some random value [6, 7]. One can further complicate the task of an attacker by executing an algorithm in some non-deterministic order, such that data is not processed at a set point in time. This spreads the information an attacker would want to exploit over numerous points in time. One way to introduce time randomization in an implementation of a block cipher is the use of dummy rounds. That is, each round observed by an attacker will be a dummy round with some probability, and produce false hypotheses for an attacker.

Fault analysis seeks to exploit the effect of a fault inserted into an instance of a cryptographic algorithm [8–10]. When implementing a block cipher a simple countermeasure is, as for side-channel analysis, to use time randomization, e.g. dummy rounds, that can render precise fault injection more difficult. However, dummy rounds do not affect an attacker directly, since injecting a fault into a dummy round will have no observable effect. Another simple countermeasure is to introduce redundancy in a certain number of rounds at the beginning and the end of a block cipher. This can, for instance, involve repeating the execution of functions in software implementations or using parallel blocks in hardware. At the end of the redundant computation the implementation checks if all redundantly computed results are equal and suppresses the output if they are not.

Contribution. In this paper we show how one can link cipher rounds, dummy rounds, and redundant rounds to further increase the security in implementations of block ciphers. The countermeasures disturb the ciphering process each time a fault is injected into a cipher, dummy, or redundant round, and give an attacker no information with which to deduce information on a secret key. As a consequence, an implementation does not need to use any consistency checks, which may themselves become targets for a fault attack and which inherently leak information about the success of fault injection.

Our countermeasures even provide some protection against an adversary who attempts to inject the *same* fault in both (all) branches of redundant computation to bypass the final consistency check. While the check actually aids the adversary since it will only output the ciphertext if the fault injection was successful, our proposed algorithms always output a ciphertext (exploitable or not). Thus, an adversary has to analyze the output without knowing whether the fault injection was successful and the output contains exploitable information, or not.

Organization. The rest of this paper is organized as follows: Section 2 provides background information on fault analysis of block cipher implementations and countermeasures. Section 3 presents our countermeasures and their application to S-P networks and Feistel ciphers. Section 4 provides the security evaluation of the countermeasures in both cases. Section 5 explains how the countermeasures can be hardened to prevent the detection of dummy rounds by side-channel analysis and Section 6 concludes the paper.

2 Background

Since the introduction of fault attacks by Boneh et al. [9], the idea of exploiting erroneous results to reveal secrets was applied to many cryptographic algorithms. The ideas are steadily improved for reducing the number of faults required. For example, it is noted in [11] that a *single* fault may be sufficient to break an AES implementation by using differential fault analysis. For such attacks, the same plaintext is encrypted twice, while a fault is injected during one of the computations. From the resulting difference in the outputs, the secret key or parts of it are derived. In contrast to differential fault analysis, collision fault analysis [8] relies on finding a plaintext that maps to the *same* output as a faulty encryption of a different plaintext. This technique is often applied to attack early rounds of an algorithm. Safe-error attacks [12] and ineffective fault analysis [10] do not require the actual output of the computation. The information whether the result is erroneous is sufficient. The same idea can also be used to easily detect dummy operations in a computation.

In order to detect fault attacks some kind of redundancy is typically introduced into implementations of cryptographic algorithms. A straightforward approach is computing the same algorithm twice and comparing the results. If the results differ, the output of the algorithm is suppressed. In order to protect the algorithm without repeating it, the proposals include limiting the repetition to a few rounds, introducing a parity byte for the state [13], and generating digest values that are tailored to the operations of the cipher [14]. While these approaches try to reduce the overhead compared to doubling the cipher, other proposals aim at a higher detection rate for injected faults, such as involving the inverse round-function of the cipher for the check [15] and enlarging the field the algorithm computes in [16].

However, irrespective of how the redundancy is introduced, the check of the result before it is released is a potential target for an adversary. Kim and

Quisquater demonstrated that one can attack a cryptographic algorithm and then inject a fault in the verification stage [17]. The natural response would be to make the verification itself redundant but van Woudenberg et al. [18] have shown that three faults can be used to attack two tests after the execution of a cryptographic algorithm, i.e. inject one fault in the algorithm itself and use two faults to overcome the redundant verification.

In order to prevent an adversary from using multiple fault-injection to bypass checking routines, Yen and Joye introduced the principle of infective computing [12]. Their proposal defines a method of generating RSA signatures where any fault injected into a computation changes the output of the cryptographic algorithm in such a way that it does not reveal any secret information. That is, the output of the cryptographic algorithm should not be exploitable. The usual way to implement infective computation is to introduce a (secret) error in an input, then to compute the result, and finally to remove the effect of the previously introduced error. If a fault is injected in the computation, then the output of the algorithm is incorrect and cannot be exploited because the initial error is unknown. An alternative way is to introduce additional computation on secret data in the algorithm that will have no effect on the result, if no fault is injected.

In the following section we describe how the latter approach can be applied to block ciphers using dummy rounds.

3 Smart Use of Dummy Rounds and Redundant Computation

We first explain how our approach can be used to make smart use of dummy rounds. This leads to an algorithm where dummy rounds can no longer be identified by fault injection, and where a fault injected into a dummy round renders the ciphertext useless to an attacker. Then we extend the approach to redundant implementations with dummy rounds.

We provide algorithms for the application of our countermeasures to S-P networks and Feistel ciphers that use the following notation:

`BlockCipher` — The entire block cipher under consideration which takes a plaintext P and enciphers it with a secret key K to produce a ciphertext C. We will consider a block cipher that consists of n rounds.

`RoundFunction` — The round function of the block cipher. It operates on a given register and requires the correct subkey k_i for that round.

`RandomBit` — This function returns a random bit that governs whether a dummy round will occur or not. If this bit is equal to zero a dummy round will be computed, otherwise a cipher round will take place. This function can be replaced by any suitable function that returns one bit, if, for example, one wants to limit the number of dummy rounds that could occur in a given instantiation of the block cipher.

We denote a bitwise logical AND operation by $\wedge$, a bitwise logical NOT operation by $\neg$, and a bitwise logical exclusive-OR (XOR) operation by $\oplus$.

3.1 Smart Use of Dummy Rounds

We propose a countermeasure where a dummy round takes a secret input value β that is known to produce a result of β after one round when combined with the secret dummy round-key k_0. That is

$$\texttt{RoundFunction}(\beta, k_0) = \beta\,.$$

The result of the dummy round is then XORed into the cipher state, and then XORed with β, ensuring that any fault is propagated into the block cipher. The result of the dummy round further overwrites β held in registers. This ensures that any fault is propagated into subsequent dummy rounds. The same operations are conducted when a cipher round is computed. This means that any fault that modifies the output of a dummy round will affect the input of *every* subsequent round, making any collision or differential fault analysis impossible.

Application to S-P networks. Algorithm 1 shows how the countermeasure can be applied to a straightforward implementation of a S-P network. The

Algorithm 1: S-P network with smart dummy rounds.

Input: P, k_i for $i \in \{1, \ldots, n+1\}$ ($n+1$ subkeys from key K), (β, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

```
1 State: R_0 ← P ;    Dummy state R_1 ← β ;    i ← 1 ;
2 while i ≤ n do
3 |   λ ← RandomBit() ;    // λ = 0 implies a dummy round
4 |   κ ← i λ ;
5 |   R_¬λ ← RoundFunction(R_¬λ, k_κ)    // infection of the dummy state
6 |   R_0 ← R_0 ⊕ R_1 ⊕ β ;    // infection of the cipher state
7 |   i ← i + λ ;
8 end
9 return R_0
```

algorithm clearly achieves the goal of propagating a fault injected into a dummy round into *every* subsequent round.

Application to Feistel ciphers. Another commonly used mechanism for block ciphers is the Feistel structure. This operates by dividing the input of one round into two equally sized sets and using a round function on one set before combining it with the other set using an XOR. For example, if we define L_i and R_i as the left and right hand inputs to the i^{th} round of a block cipher respectively, then L_{i+1} and R_{i+1} are computed in the following manner:

$$\begin{aligned} L_{i+1} &\leftarrow R_i \\ R_{i+1} &\leftarrow \texttt{RoundFunction}(R_i, k_{i+1}) \oplus L_i. \end{aligned}$$

Algorithm 2 shows how the countermeasure can be applied to a straightforward implementation of a Feistel cipher. It uses the following additional notation: let P_r be the right-hand side and P_l be the left-hand side of the plaintext, respectively. Further, let α be some arbitrary, constant value where the relationship between α, β and k_0 is $\texttt{RoundFunction}(\beta, k_0) \oplus \alpha = \beta$.

Algorithm 2: Feistel cipher with smart dummy rounds.

Input: P, k_i for $i \in \{1, \ldots, n\}$ (n subkeys from key K), (β, α, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

```
1  State R_0 ← P_r ;    Dummy state R_1 ← β ;    i ← 1 ;
2  State T_0 ← P_l ;    T_1 ← α ;
3  while i ≤ n do
4  |  λ ← RandomBit() ;    // λ = 0 implies a dummy round
5  |  κ ← i λ ;
6  |  R_¬λ ← RoundFunction(R_¬λ, k_κ) ⊕ T_¬λ    // infection of the dummy state
7  |  R_0 ← R_0 ⊕ R_1 ⊕ β ;    // infection of the cipher state
8  |  T_0 ← T_0 ⊕ R_1 ⊕ β ;    // infection of the cipher state
9  |  i ← i + λ ;
10 end
11 return T_0 ‖ R_0
```

We can note that a fault in a dummy round will affect both R_0 and T_0 and therefore provoke a larger change in the resulting ciphertext.

3.2 Smart Use of Redundant Rounds and Dummy Rounds

Algorithms 1 and 2 both achieve the goal of propagating faults injected into a *dummy* round into *every* subsequent round. However, both algorithms still require the output of some of the first and last rounds to be checked to prevent collision and differential fault analysis based on a fault injected into a *cipher* round.

To overcome this limitation we further propose algorithms where each round is repeated and any fault in a single round (cipher, dummy, or redundant round) will affect the resulting ciphertext such that no information is available to an attacker. This means that a verification stage is not necessary since an attacker will not receive any information from a fault. As before, an attacker will not be able to determine if any particular round is a dummy round by means of fault injection.

Note that repeating each round implies that each round can be observed twice through some side-channel, which can ease side-channel analysis. However, the time randomization due to the dummy rounds provides some level of protection. In addition, although each cipher round is repeated, this will occur in a somewhat random manner. That is, a cipher round may be followed by a dummy round

or a redundant round with some probability determined by how `RandomBit` is defined, and the round counter only increases once the redundant round has been computed.

As above, we assume that β and k_0 are chosen such that the result of the round function is β. That is

$$\texttt{RoundFunction}(\beta, k_0) = \beta\,.$$

The algorithms also use additional notation:

`SNLF` — This stands for Some NonLinear Function, which is used to ensure that any fault in the block cipher will not provide an attacker with any information. We discuss the reasons for this in more detail in Section 4. In our algorithms we assume that $\texttt{SNLF}(0) \mapsto 0$.

Application to S-P networks. Algorithm 3 shows how the countermeasure can be applied to a redundant implementation of an S-P network.

Algorithm 3: Redundant S-P Network with Dummy Rounds.

Input: P, k_i for $i \in \{1, \ldots, n+1\}$ ($n+1$ subkeys from key K), (β, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

State $R_0 \leftarrow P$; Redundant state $R_1 \leftarrow P$; Dummy state $R_2 \leftarrow \beta$;
$C_0 \leftarrow 0$; $C_1 \leftarrow 0$; $C_2 \leftarrow \beta$; $i \leftarrow 0$;
while $i < 2\,n$ **do**
  $\lambda \leftarrow \texttt{RandomBit}()$; // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda)$;
  $\zeta \leftarrow \lceil i/2 \rceil\,\lambda$; // ζ is actual round counter, 0 for dummy
  $R_\kappa \leftarrow \texttt{RoundFunction}(R_\kappa, k_\zeta)$;
  $C_\kappa \leftarrow R_\kappa \oplus C_2 \oplus \beta$; // infect C_κ to propagate a fault
  $\epsilon \leftarrow \lambda(\neg(i \wedge 1)) \cdot \texttt{SNLF}(C_0 \oplus C_1)$; // check if i is even
  $R_2 \leftarrow R_2 \oplus \epsilon$;
  $R_0 \leftarrow R_0 \oplus \epsilon$;
  $i \leftarrow i + \lambda$;
end
$R_0 \leftarrow R_0 \oplus \texttt{RoundFunction}(R_2, k_0)$;
return R_0

Algorithm 3 progresses by computing the same round twice before advancing to the next round using i as a counter. When i is an even number the difference between the result of a cipher round and a redundant round should be equal to zero and any difference is XORed into β. This difference also goes through some nonlinear function to make it difficult for an attacker to make any hypotheses about any fault that has been induced. If i is an odd number, or a dummy

round occurs, the difference between the cipher round and the redundant round is multiplied by zero since it will be non-zero during the normal functioning of the algorithm. Tables 1(a), 1(b) and 1(c) illustrate the functioning of the algorithm with examples.

Table 1. Examples of Algorithm 3.

(a) A dummy round. A fault will change C_2 infecting every subsequent round.

$\lambda = 0$
$\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda) = 2$
$\zeta \leftarrow \lceil i/2 \rceil\ \lambda = 0$
$R_2 \leftarrow \texttt{RoundFunction}(R_2, k_0) = \beta$
$C_2 \leftarrow R_2 \oplus C_2 \oplus \beta = \beta$
$\epsilon \leftarrow 0 \cdot \texttt{SNLF}(C_0 \oplus C_1) = 0$
$R_2 \leftarrow R_2 \oplus \epsilon$
$R_0 \leftarrow R_0 \oplus \epsilon$
$i \leftarrow i + 0$

(b) A round where i is even. A fault will change R_0 that will infect every subsequent round where i is odd and every round after the next dummy round.

$i = \text{even}, \lambda = 1$
$\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda) = 0$
$\zeta \leftarrow \lceil i/2 \rceil$
$R_0 \leftarrow \texttt{RoundFunction}(R_0, k_\zeta)$
$C_0 \leftarrow R_0 \oplus C_2 \oplus \beta = R_0$
$\epsilon \leftarrow 0 \cdot \texttt{SNLF}(C_0 \oplus C_1) = 0$
$R_2 \leftarrow R_2 \oplus \epsilon$
$R_0 \leftarrow R_0 \oplus \epsilon$
$i \leftarrow i + 1$

(c) A round where i is odd. A fault will change R_1 that will infect every subsequent round where i is even and every round after the next dummy round.

$i = \text{odd}, \lambda = 1$
$\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda) = 1$
$\zeta \leftarrow \lceil i/2 \rceil$
$R_1 \leftarrow \texttt{RoundFunction}(R_1, k_\zeta)$
$C_1 \leftarrow R_1 \oplus C_2 \oplus \beta = R_1$
$\epsilon \leftarrow 1 \cdot \texttt{SNLF}(C_0 \oplus C_1)$
$R_2 \leftarrow R_2 \oplus \epsilon$
$R_0 \leftarrow R_0 \oplus \epsilon$
$i \leftarrow i + 1$

An extra dummy round is added to the end of the algorithm to ensure that any fault that has been propagated through the algorithm masks the ciphertext in such a way that no information is available to an attacker. Moreover, this will ensure the countermeasure is effective if an attacker is able to affect the `RandomBit` function such that no dummy rounds occur.

The extra dummy round and the use of the nonlinear function are discussed in more detail using an example in Section 4. We do not need to add a dummy round at the beginning of the block cipher to protect against collision fault analysis since an attacker will not be able to find a collision. That is, if an attacker injects a fault in one of the early rounds of a block cipher, the effect of

this fault will be propagated by the proposed countermeasure in such a way that an attacker cannot hope to find a collision by making hypotheses on a subset of the secret key bits.

Application to Feistel ciphers. In Algorithm 4 we define a redundant algorithm for use with Feistel ciphers using the same principles.

Algorithm 4: Redundant Feistel Cipher with Dummy Rounds.

Input: P, k_i for $i \in \{1, \ldots, n\}$ (n subkeys from key K), (β, α, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

State $R_0 \leftarrow P_r$; Redundant State $R_1 \leftarrow P_r$; Dummy state $R_2 \leftarrow \beta$;
State $T_0 \leftarrow P_l$; Redundant State $T_1 \leftarrow P_l$; Dummy state $T_2 \leftarrow \alpha$;
$C_0 \leftarrow 0$; $C_1 \leftarrow 0$; $C_2 \leftarrow \beta$; $i \leftarrow 1$;
while $i \leq 2\,n$ **do**
  $\lambda \leftarrow \texttt{RandomBit}()$; // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda)$;
  $\zeta \leftarrow \lceil i/2 \rceil\, \lambda$; // ζ is actual round counter, 0 for dummy
  $R_\kappa \leftarrow \texttt{RoundFunction}(R_\kappa, k_\zeta) \oplus T_\kappa$;
  $C_\kappa \leftarrow R_\kappa \oplus C_2$; // infect C_κ to propagate a fault
  $\epsilon \leftarrow \lambda(\neg(i \wedge 1)) \cdot \texttt{SNLF}(C_0 \oplus C_1)$; // check if i is even
  $T_0 \leftarrow T_0 \oplus \epsilon$;
  $R_0 \leftarrow R_0 \oplus \epsilon$;
  $R_2 \leftarrow R_2 \oplus \epsilon$;
  $i \leftarrow i + \lambda$;
end
$C_2 \leftarrow \texttt{RoundFunction}(R_2, k_0) \oplus T_2$;
$R_0 \leftarrow R_0 \oplus C_2$;
return $T_0 \| R_0$

Remark. In our algorithms we assume that the block cipher consists of either an S-P network or a Feistel construction. In both cases it is straightforward to compute a constant pair (β, k_0) resp. triplet (α, β, k_0) by simply inverting the round function and choosing a suitable k_0 for a chosen β (and α).

In defining the countermeasures to protect a secure implementation, one would use a redundant implementation to protect a certain number of rounds at the beginning and the end of the block cipher. That is, one would use a round as defined in Algorithm 1 resp. 2 for rounds that do not require explicit protection and a round as defined in Algorithm 3 resp. 4 for those that do. Further, the dummy rounds should not be distinguishable from every other round by means of side-channel analysis. We detail how this can be achieved in Section 5.

3.3 Performance

We assess the performance overhead of the proposed algorithms, first for S-P networks and then for Feistel ciphers. We begin with determining the cost of making "normal" dummy rounds smart, then we add the cost of making the implementations redundant. We assume that the baseline implementation already uses dummy rounds, and the analysis is independent of the chosen function `RandomBit`, i.e. of the number of dummy rounds. The number of redundant rounds is also not covered by our analysis.

For S-P networks, the overhead to make dummy rounds smart (Algorithm 1) is: one additional constant (β) the size of the cipher state, and two XOR operations on operands the size of the cipher state in each round. The generation of the random bits, the additional (dummy) rounds, and the second cipher state are already required by using dummy rounds without our countermeasure.

In order to further equip the implementation of the S-P network with smart redundant rounds, as done in Algorithm 3, four additional states, and one round execution followed by an XOR at the end of the implementation are required. In addition, some nonlinear function, four XORs and four logic operations have to be computed for each round.

For Feistel ciphers, the overhead to make dummy rounds smart (Algorithm 2) is: one additional state of half the block size, two constants of half the block size (α and β), and four additional XOR operations on operands half the block size per round.

The fault protection for Feistel ciphers in Algorithm 4 requires five additional states of half the block size, and one round execution followed by two XORs at the end of the implementation. Moreover, each round requires an additional evaluation of some nonlinear function, two XOR operations, and four logic operations. Table 2 shows a summary of the results.

Table 2. Performance overheads: from dummy rounds to smart dummy rounds (left), and from smart dummy rounds to smart dummy rounds with redundancy (right).

	Smart Dummy rounds		+ Redundant Rounds	
	Overall	per round	Overall	per round
S-P Network	+ 1 Const	+ 2 XOR	+ 4 States + 1 XOR + 1 RF	+ 4 XOR + 4 Logic + 1 NLF
Feistel Cipher	+ 1/2 State + 2/2 Const	+ 4 XOR	+ 5/2 States + 2 XOR + 1RF	+ 2 XOR + 4 Logic + 1 NLF

4 Evaluating the Countermeasure

In this section we evaluate the security of the above presented redundant algorithms for S-P networks and Feistel ciphers. We describe how the weakest instance of the presented countermeasure would affect the strongest attacks available. Any other attacks from the literature will typically require a more complex analysis, e.g. a larger fault, more faulty ciphertexts, or a fault further into the algorithm, and the effect of the last dummy round will become more pronounced. Some complex attacks and those requiring particular circumstances, such as collision fault analysis, are not considered since they are completely prevented by the countermeasure.

4.1 S-P networks

In this section we discuss Algorithm 3 in terms of its resistance to differential fault analysis. To demonstrate this we use the AES as an example, since it will be the most likely instance of our countermeasure in S-P networks.

The Advanced Encryption Standard (AES) [19] was standardized in 2001 from a proposal by Daemen and Rijmen [20]. Note that we restrict ourselves to considering AES-128 and that in discussing the AES we consider that all variables are arranged in a 4×4 array of bytes, known as the state matrix. For example the 128-bit plaintext $P = (p_1, p_2, \ldots, p_{16})_{(256)}$ is arranged as follows:

$$\begin{pmatrix} p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \\ p_4 & p_8 & p_{12} & p_{16} \end{pmatrix}.$$

The encryption itself is conducted by the repeated use of a round function that comprises the following operations executed in sequence:

`SubBytes` — The only nonlinear step of the block cipher, consisting of a substitution table applied to each byte of the state.

`ShiftRows` — A byte-wise permutation of the state that operates on each row.

`MixColumns` — Each column of the state matrix is considered as a vector where each of its four elements belong to $\mathbb{F}(2^8)$. A $4{\times}4$ matrix M whose elements are also in $\mathbb{F}(2^8)$ is used to map this column into a new vector. This operation is applied to the four columns of the state matrix. Here M is defined as

$$M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

where all the elements in M are elements of $\mathbb{F}(2^8)$ expressed in decimal. This function is not included in the last round.

`AddRoundKey` — XORs each byte of the array with a byte from a corresponding subkey. An initial subkey addition precedes the first round.

The simplest, yet strongest, example of differential fault analysis was proposed by Piret and Quisquater [21], where it is assumed that a one byte fault is induced at the beginning of the ninth round. If, for example, the byte at index one is modified by a fault, the difference between the result and what it ought to be at the end of the ninth round becomes:

$$\begin{pmatrix} 2\,\theta & 0 & 0 & 0 \\ \theta & 0 & 0 & 0 \\ \theta & 0 & 0 & 0 \\ 3\,\theta & 0 & 0 & 0 \end{pmatrix}$$

An attacker can compare a faulty and the correct ciphertext using the following relationship, using the notation defined above,

$$\begin{aligned} 2\,\theta &= S^{-1}(c_1 \oplus k_1) \oplus S^{-1}(c'_1 \oplus k_1) \\ \theta &= S^{-1}(c_{14} \oplus k_{14}) \oplus S^{-1}(c'_{14} \oplus k_{14}) \\ \theta &= S^{-1}(c_{11} \oplus k_{11}) \oplus S^{-1}(c'_{11} \oplus k_{11}) \\ 3\,\theta &= S^{-1}(c_8 \oplus k_8) \oplus S^{-1}(c'_8 \oplus k_8)\,, \end{aligned}$$

where c'_i and k_i for $i \in \{1, \ldots, 16\}$ represents the bytes from the state matrix of the faulty ciphertext and the last subkey respectively. This will allow the 2^{32} possible key hypotheses for $\{k_1, k_8, k_{11}, k_{14}\}$ to be reduced to 2^8 possibilities. The procedure can then be repeated to derive the entire last subkey.

With our countermeasures in place, the equations would become:

$$\begin{aligned} 2\,\theta &= S^{-1}(c_1 \oplus k_1) \oplus S^{-1}(c'_1 \oplus k_1 \oplus (b_1 \oplus \gamma_1)) \\ \theta &= S^{-1}(c_{14} \oplus k_{14}) \oplus S^{-1}(c'_{14} \oplus k_{14} \oplus (b_{14} \oplus \gamma_{14})) \\ \theta &= S^{-1}(c_{11} \oplus k_{11}) \oplus S^{-1}(c'_{11} \oplus k_{11} \oplus (b_{11} \oplus \gamma_{11})) \\ 3\,\theta &= S^{-1}(c_8 \oplus k_8) \oplus S^{-1}(c'_8 \oplus k_8 \oplus (b_8 \oplus \gamma_8))\,, \end{aligned}$$

where we define b_i for $i \in \{1, \ldots, 16\}$ as the output of the $\texttt{SNLF}(C_0 \oplus C_1)$ at the end of the penultimate round, and γ_i for $i \in \{1, \ldots, 16\}$ as the result of the last dummy round. The `SNLF` cannot be implemented as the `SubBytes` operation since it does not map a zero to a zero as required, but one could, for example, use inversion in $\mathbb{F}(2^8)$.

In this example we consider that the `MixColumns` operation is not included to simplify the analysis. However, an equivalent analysis would be straightforward if the dummy round included a `MixColumns` operation. One can simplify the above by rewriting each $b_i \oplus \gamma_i$, for $i \in \{1, 8, 11, 14\}$, as one unknown byte. However, an attacker will not be able to use the equations to reduce the number of possible keys since there will be 2^{48} solutions.

4.2 Feistel structures

We discuss Algorithm 4 in terms of its resistance to differential fault analysis and use the Data Encryption Standard (DES) as an example. We consider DES relevant since it is still widely used in banking.

DES was introduced by NIST in the mid 1970s [22], and was the first openly available cryptography standard. DES can be considered as a transformation of two 32-bit variables (L_0, R_0), i.e. the message block, through sixteen iterations of the Feistel structure to produce a ciphertext block (L_{16}, R_{16}). The Expansion and P-permutations are bitwise permutations. For clarity of expression, these permutations will not always be considered and the round function for round n will be written as:

$$\begin{aligned} L_n &= R_{n-1} \\ R_n &= S(R_{n-1} \oplus K_n) \oplus L_{n-1} \end{aligned}$$

where S is a nonlinear substitution function.

In this section we describe the strongest attack that can be applied to an implementation of DES. This fault attack on DES involves injecting a fault in the fifteenth round and was described by Biham and Shamir [23]. The last round of DES can be expressed in the following manner:

$$\begin{aligned} R_{16} &= S(R_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(L_{16} \oplus K_{16}) \oplus L_{15}\,. \end{aligned}$$

For ease of expression we ignore the bitwise permutations since they will only impact an implementation of the described attack. If a fault occurs during the execution of the fifteenth round, i.e. R_{15} is randomized by a fault to become R'_{15}, then:

$$\begin{aligned} R'_{16} &= S(R'_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(L'_{16} \oplus K_{16}) \oplus L_{15} \end{aligned}$$

and

$$\begin{aligned} R_{16} \oplus R'_{16} &= S(L_{16} \oplus K_{16}) \oplus L_{15} \oplus S(L'_{16} \oplus K_{16}) \oplus L_{15} \\ &= S(L_{16} \oplus K_{16}) \oplus S(L'_{16} \oplus K_{16})\,. \end{aligned}$$

This provides an equation in which only the last subkey, K_{16}, is unknown. All of the other variables are available from the ciphertext block. This equation holds for each S-box in the last round, which means that it is possible to search for key hypotheses in sets of six bits. This will return an expected 2^{24} key hypotheses for the last round key and, therefore, 2^{32} hypotheses for the block cipher key.

If an attacker attempts to apply this attack to an instance of Algorithm 4, the last dummy round would need to be taken into account. A faulty ciphertext would have the form:

$$\begin{aligned} R'_{16} &= S(R'_{15} \oplus K_{16}) \oplus L_{15} \oplus (b \oplus \gamma) \\ &= S(L'_{16} \oplus K_{16}) \oplus L_{15} \oplus (b \oplus \gamma)\,, \end{aligned}$$

where b denotes the result of the $\texttt{SNLF}(C_0 \oplus C_1)$ function at the end of the last round and γ denotes the result of the last dummy round. We obtain the difference

$$R_{16} \oplus R'_{16} = S(L_{16} \oplus K_{16}) \oplus S(L'_{16} \oplus K_{16}) \oplus (b \oplus \gamma)\,.$$

As in the previous example, this would have too many solutions to provide any information on the last subkey since $b \oplus \gamma$ is unknown.

5 Further Strengthening the Countermeasure

In Algorithm 1 we require that

$$\texttt{RoundFunction}(\beta, k_0) = \beta\,. \tag{1}$$

If the pair (β, k_0) is fixed for a given implementation, the computation during all dummy rounds will be identical. Therefore, the pattern in a given side-channel generated by a dummy round would have a given form. An attacker could potentially exploit this weakness to identify dummy rounds by cross correlation or template analysis [24].

To solve this problem one can simply refresh the pair (β, k_0) with a frequency determined by how powerful an attacker is assumed to be. More precisely, one would randomly generate β (or k_0) and change k_0 (or β) such that (1) holds. This fully randomizes the computation during a dummy round but it may still be possible to identify a dummy round as a round with equal input and output.

A better, but also more costly, approach would be to generate a triplet of random values (β, k_0, δ) where

$$\texttt{RoundFunction}(\beta, k_0) = \delta\,.$$

The input and output of a dummy round would be random and an attacker would no longer be able to identify a dummy round. However, this approach requires that the effect of the δ be corrected. For example, in Algorithm 1, line 6 would need to be replaced with

$$R_0 \leftarrow R_0 \oplus R_1 \oplus \beta \oplus \delta.$$

This makes the countermeasure more expensive but ensures that an attacker would be unable to identify a dummy round by any means.

The application of this latter idea to Algorithm 2 would require a randomly generated quartet $(\beta, k_0, \alpha, \delta)$, since we would require that

$$\texttt{RoundFunction}(\beta, k_0) \oplus \alpha = \delta\,.$$

Otherwise, strengthening the countermeasure for Feistel ciphers would be similar to strengthening that for S-P networks in Algorithm 1.

6 Conclusion

In this paper we describe algorithms where dummy and redundant rounds can be used to implement infective computation in block ciphers. This would allow a block cipher to be implemented where no check is required to detect whether a fault has occurred since no information would be available to an attacker. This would prevent an attacker from implementing a multiple-fault attack that affects tests at the end of a block cipher [25, 18].

We have demonstrated that the algorithms will be secure against the strongest available differential fault analysis that can be applied to AES and DES. We have not demonstrated this for all fault attacks that have been described in the literature since many attacks will not be possible. For example, collision fault analysis requires an attacker to inject a fault in one of the first rounds of a block cipher to produce a faulty ciphertext. The attacker will then try to find a plaintext that will produce the same ciphertext without a fault. Given that such a fault will re-infect the implementation numerous times, such an attack becomes impossible.

The current state-of-the-art in implementing block ciphers is to implement some form of consistency check in order to detect faults, so that the output can be withheld if a fault has occurred. This is potentially vulnerable to an attacker who can inject the same fault into both redundant paths of the algorithm, thus by-passing the check at the end of the algorithm. This would be a complex attack, but the verification would aid an attacker since a faulty result would only appear once the fault injection is successful. With our countermeasures an attacker will always receive a ciphertext. Thus, an adversary has to analyze the output without knowing whether the fault injection was successful and the output contains exploitable information, or not. Given that injecting two identical faults into a single run of an unknown implementation with random dummy rounds is very hard and will only succeed with a very low probability, it would be very difficult to determine at what point a sufficient number of exploitable faulty ciphertexts have been collected, and to distinguish them from the ones that do not provide any information on the secret key.

The use of the proposed countermeasures will also help to avoid a situation where a new fault attack is published that allows an attacker to exploit faults over more rounds than that considered necessary when a given algorithm is implemented. This is because such attacks will typically require a relatively large number of faulty ciphertexts [26, 27], and our countermeasure will insert faulty ciphertexts containing no information into the analysis. This will hinder or prevent new attacks.

Acknowledgements

The work described in this paper has been supported in part by the European Commission through the ICT Programme under contract ICT-2007-216676 ECRYPT II and under contract ICT-SEC-2009-5-258754 TAMPRES, by the EPSRC via grant EP/I005226/1, by the Research Council of KU Leuven: GOA

TENSE (GOA/11/007), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the Flemish Government FWO G.0550.12N and by the Hercules Foundation AKUL/11/19. Benedikt Gierlichs is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

References

1. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N., ed.: CRYPTO '96. Volume 1109 of LNCS., Springer (1996) 104–113
2. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In Wiener, M.J., ed.: CRYPTO '99. Volume 1666 of LNCS., Springer (1999) 388–397
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In Joye, M., Quisquater, J.J., eds.: CHES 2004. Volume 3156 of LNCS., Springer (2004) 16–29
4. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In Ç. K. Koç, Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer (2001) 251–261
5. Quisquater, J.J., Samyde, D.: Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Attali, I., Jensen, T.P., eds.: Smart Card Programming and Security, International Conference on Research in Smart Cards — E-smart 2001. Volume 2140 of LNCS., Springer (2001) 200–210
6. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In Wiener, M., ed.: CRYPTO '99. Number 1666 in LNCS, Springer (1999) 398–412
7. Goubin, L., Patarin, J.: DES and differential power analysis: the "duplication" method. In Ç.K. Koç, Paar, C., eds.: CHES '99. Number 1717 in LNCS, Springer (1999) 158–172
8. Blömer, J., Seifert, J.P.: Fault based cryptanalysis of the Advanced Encryption Standard (AES). In Wright, R.N., ed.: Financial Cryptography 2003. Volume 2742 of LNCS., Springer (2003) 162–181
9. Boneh, D., DeMillo, R., Lipton, R.: On the importance of checking cryptographic protocols for faults. In Fumy, W., ed.: EUROCRYPT '97. Volume 1233 of LNCS., Springer (1997) 37–51
10. Clavier, C.: Secret external encodings do not prevent transient fault analysis. In Paillier, P., Verbauwhede, I., eds.: CHES 2007. Volume 4727 of LNCS., Springer (2007) 181–194
11. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the Advanced Encryption Standard using a single fault. In Ardagna, C.A., Zhou, J., eds.: WISTP 2011. Volume 6633 of LNCS., Springer (2011) 224–233
12. Yen, S.M., Joye, M.: Checking before output may not be enough against fault based cryptanalysis. IEEE Transactions on Computers **49**(9) (2000) 967–970
13. Karpovsky, M.G., Kulikowski, K.J., Taubin, A.: Robust protection against fault-injection attacks on smart cards implementing the Advanced Encryption Standard. In: International Conference on Dependable Systems and Networks (DSN 2004), IEEE (2004) 93–101
14. Genelle, L., Giraud, C., Prouff, E.: Securing AES implementation against fault attacks. In Naccache, D., Oswald, E., eds.: Fault Diagnosis and Tolerance in Cryptography, IEEE (2009) 51–62

15. Malkin, T., Standaert, F.X., Yung, M.: A comparative cost/security analysis of fault attack countermeasures. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: Fault Diagnosis and Tolerance in Cryptography. Volume 4236 of LNCS., Springer (2006) 159–172
16. Medwed, M., Schmidt, J.M.: A continuous fault countermeasure for AES providing a constant error detection rate. In Breveglieri, L., Joye, M., Koren, I., Naccache, D., Verbauwhede, I., eds.: Fault Diagnosis and Tolerance in Cryptography, IEEE (2010) 66–71
17. Kim, C.H., Quisquater, J.J.: Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures. In Sauveron, D., Markantonakis, C., Bilas, A., Quisquater, J.J., eds.: WISTP 2007. Volume 4462 of LNCS., Springer (2007) 215–228
18. van Woudenberg, J.G.J., Witteman, M.F., Menarini, F.: Practical optical fault injection on secure microcontrollers. In Breveglieri, L., Guilley, S., Koren, I., Naccache, D., Takahashi, J., eds.: Fault Diagnosis and Tolerance in Cryptography, IEEE (2011) 91–99
19. FIPS PUB 197: Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA (November 2001)
20. Daemen, J., Rijmen, V.: AES proposal: Rijndael. In: AES Round 1 Technical Evaluation CD-1: Documentation, NIST (August 1998) `http://www.nist.gov/aes`.
21. Piret, G., Quisquater, J.J.: A differential fault attack technique against SPN structure, with application to the AES and KHAZAD. In Walter, C.D., Ç. K. Koç, Paar, C., eds.: CHES 2003. Volume 2779 of LNCS., Springer (2003) 77–88
22. NIST: Data Encryption Standard (DES) (FIPS–46-3). National Institute of Standards and Technology. (1999)
23. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In Kaliski, B.S., ed.: CRYPTO '97. Volume 1294 of LNCS., Springer (1997) 513–525
24. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In Kaliski Jr., B.S., Ç. K. Koç, Paar, C., eds.: CHES 2002. Volume 2523 of LNCS., Springer (2002) 172–186
25. Kim, C.H., Quisquater, J.J.: New differential fault analysis on AES key schedule: Two faults are enough. In Grimaud, G., Standaert, F.X., eds.: CARDIS 2008. Volume 5189 of LNCS., Springer (2008) 48–60
26. Derbez, P., Fouque, P.A., Leresteux, D.: Meet-in-the-middle and impossible differential fault analysis on AES. In Preneel, B., Takagi, T., eds.: CHES 2011. Volume 6917 of LNCS., Springer (2011) 274–291
27. Rivain, M.: Differential fault analysis on DES middle rounds. In Clavier, C., Gaj, K., eds.: CHES 2009. Volume 5747 of LNCS., Springer (2009) 457–469