Destroying Fault Invariant with Randomization -A Countermeasure for AES against Differential Fault Attacks

Harshal Tupsamudre, Shikha Bisht and Debdeep Mukhopadhyay

Department of Computer Science and Engg.
IIT Kharagpur, India
{thanil, shikhab, debdeep}@cse.iitkgp.ernet.in

Abstract. Researchers have demonstrated the ineffectiveness of deterministic countermeasures and emphasized on the use of randomness for protecting cryptosystems against fault attacks. One such countermeasure for AES was proposed in LatinCrypt 2012, which masks the faulty output with secret values. However this countermeasure does not affect the erroneous byte in the faulty computation of the last AES round and is thus shown to be flawed in FDTC 2013. In this paper, we examine the LatinCrypt 2012 countermeasure in detail and identify its additional flaws in order to develop a robust countermeasure. We bring out the major weakness in the infection mechanism of the LatinCrypt 2012 countermeasure which not only makes the attack of FDTC 2013 much more flexible, but also enables us to break this seemingly complex countermeasure using Piret & Quisquater's attack that requires only 8 pairs of correct and faulty ciphertexts. Finally, we combine all our observations and propose a countermeasure that employs randomness much more effectively to prevent state-of-the-art differential fault attacks against AES.

Keywords: Infection Countermeasure, AES, Randomness, Fault Attack.

1 Introduction

Ever since the demonstration of fault attacks by Dan Boneh *et.al* [1] on RSA cryptosystem, fault analysis has been extensively studied and cryptosystems such as DES and AES have been shown vulnerable to fault attacks. The purpose of fault attacks is to retrieve the secret key used in the cryptosystems. This is done by injecting a fault in a specific operation of the cipher and exploiting the erroneous result. With respect to AES, there are multiple flavors of fault attacks. While some of them exploit the relation between the faulty and fault free ciphertext [2–5], some attacks can succeed with the knowledge of faulty ciphertexts only [6]. There are attacks which require as many as 128 faults to recover the secret key [7] whereas there are also attacks which require as few as one random fault to retrieve the entire secret key of AES [8].

With so many variants of attacks introduced so far, it is now a well known fact that fault attacks are a serious threat to the cryptographic implementations and therefore, sound countermeasures are required to protect them. We focus our discussion on AES, for which many countermeasures have been suggested. These countermeasures can be broadly classified into two categories - detection and infection. The detection countermeasure is usually implemented by duplicating the computation and finally comparing the results of two computations. But in this countermeasure, the comparison step itself is prone to fault attacks. The infection countermeasure on the other hand, aims to destroy the fault invariant by diffusing the effect of a fault in such a way that it renders the faulty ciphertext unexploitable. Infection countermeasures are preferred to detection as they avoid the use of attack vulnerable operations such as comparison.

In FDTC 2012, Lomné *et.al* [9] showed that infection countermeasures which use deterministic diffusion to infect the intermediate output are not secure and emphasized on the need of randomness in these countermeasures. In LatinCrypt 2012, Gierlichs *et.al* [10] proposed an infection countermeasure for AES which infects the faulty computation with random values. Despite the use of randomness in the infection mechanism, the countermeasure for AES128 [10] was attacked by Battistello and Giraud in FDTC 2013 [11]. They observed that if a fault is injected in any byte of the last three rows of the 10^{th} round input, then the erroneous byte remains unaffected by the infection method and can be exploited to retrieve the corresponding key byte. This attack assumes a *constant byte fault model* to retrieve 12 bytes of AES128 key using 36 faults on average and recovers the remaining 4 key bytes corresponding to the top row using a brute-force search.

In this paper, we concern ourselves with the countermeasure proposed in [10], study its flaws in light of two different attacks and subsequently propose a modified countermeasure that prevents the differential fault attacks.

Contribution. The main objective of this paper is to develop an infection countermeasure for AES based upon the idea proposed by Gierlichs *et. al* [10]. For this purpose, we show that the infection method employed in the countermeasure [10] is not strong as we can remove the infection and obtain exploitable faulty ciphertext. Using this observation, we can attack the top row of the 10^{th} cipher round input, which makes the attack presented in [11] more flexible. Furthermore, we show that despite the presence of infection we can mount a more practical attack, i.e the Piret & Quisquater's attack [4] on this countermeasure, thus exposing its weakness against classical fault attacks. We finally present a modified algorithm that avoids all the pitfalls of the countermeasure [10] thereby thwarting state-of-the-art differential fault attacks.

Organization. The rest of this paper is organized as follows. Section 2 sets the background by briefly explaining the infection scheme proposed in [10], followed by the attack description [11]. In Section 3, we examine additional flaws in the scheme [10] which make the attack of [11] more flexible and finally demonstrate an efficient attack on [10]. Based on the observations in section 2 and 3, we present the modified countermeasure in section 4. Section 5 concludes the paper.

2 Preliminaries

In the rest of the discussion, we use the following notations:
RoundFunction - The round function of AES128 block cipher which operates on a 16 byte state matrix and 16 byte round key. In a $RoundFunction$, the SubByte, ShiftRow and MixColumn transformations are applied successively on the state matrix, followed by the KeyXor operation. AES128 has 10 rounds in addition to the initial Key Whitening step, which we refer to as the 0^{th} round.

S - The SubByte operation in the *RoundFunction*.
SR - The ShiftRow operation in the *RoundFunction*.
MC - The MixColumn operation in the *RoundFunction*.
$\mathbf{I^i}$ - The 16 byte input to the i^{th} round of AES128, where $i \in \{0, \ldots, 10\}$.
K - The 16 byte secret key used in AES128.
$\mathbf{k^j}$ - The 16 byte matrix that represents $(j-1)^{th}$ round key, $j \in \{1, \ldots, 11\}$, derived from the main secret key K.
$\boldsymbol{\beta}$ - The 16 byte secret input to the dummy round.
$\mathbf{k^0}$ - The 16 byte secret key used in the computation of dummy round.
The 16 bytes $(m_0 \ldots m_{15})$ of a matrix are arranged in 4×4 arrays and follow a column major order. We denote multiplication symbol by $\cdot$, a bitwise logical AND operation by $\wedge$, a bitwise logical OR operation by $\vee$, a bitwise logical NOT operation by $\neg$ and a bitwise logical XOR operation by $\oplus$.

In this section, we begin by explaining the countermeasure for AES128 proposed in [10], followed by a brief description of the attack [11] mounted on it.

2.1 Infection Countermeasure

Algorithm 1 Infection Countermeasure [10]

```
Inputs : $P$, $k^j$ for $j \in \{1, \ldots, n\}$, $(\beta, k^0)$, $(n = 11)$ for AES128
Output : C = BlockCipher$(P, K)$

 1. State $R_0 \leftarrow P$, Redundant state $R_1 \leftarrow P$, Dummy state $R_2 \leftarrow \beta$
 2. $C_0 \leftarrow 0$, $C_1 \leftarrow 0$, $C_2 \leftarrow \beta$, $i \leftarrow 1$
 3. while $i \leq 2n$ do
 4.     $\lambda \leftarrow RandomBit()$        // $\lambda = 0$ implies a dummy round
 5.     $\kappa \leftarrow (i \wedge \lambda) \oplus 2(\neg\lambda)$
 6.     $\zeta \leftarrow \lambda \cdot \lceil i/2 \rceil$        // $\zeta$ is actual round counter, 0 for dummy
 7.     $R_\kappa \leftarrow RoundFunction(R_\kappa, k^\zeta)$
 8.     $C_\kappa \leftarrow R_\kappa \oplus C_2 \oplus \beta$        // infect $C_\kappa$ to propagate a fault
 9.     $\epsilon \leftarrow \lambda(\neg(i \wedge 1)) \cdot SNLF(C_0 \oplus C_1)$        // check if $i$ is even
10.     $R_2 \leftarrow R_2 \oplus \epsilon$
11.     $R_0 \leftarrow R_0 \oplus \epsilon$
12.     $i \leftarrow i + \lambda$
13. end
14. $R_0 \leftarrow R_0 \oplus RoundFunction(R_2, k^0) \oplus \beta$
15. return$(R_0)$
```

Algorithm 1 depicts the infection countermeasure proposed in [10] for AES128. At the beginning of this algorithm, plaintext P is copied to both R_0 and R_1 and a secret value β is copied to R_2. In this algorithm, every round of AES is executed twice. The redundant round which operates on R_1, occurs before the cipher round which operates on R_0. There are dummy rounds which occur randomly across the execution of this algorithm, in addition to one compulsory dummy round in step 14. The input to the dummy round is a secret value β and a secret key k^0, which is chosen such that $RoundFunction(\beta, k^0) = \beta$. To prevent the information leakage through side channels *e.g. power analysis,* dummy SubByte, ShiftRow and MixColumn operations are added to the 0^{th} round and a dummy MixColumn operation is added to the 10^{th} round of AES128. The intermediate computation of cipher, redundant and dummy round is stored in C_0, C_1 and C_2 respectively. A random bit λ decides the course of the algorithm as follows:

1. $\lambda = 0$, dummy round is executed.
2. $\lambda = 1$ and parity of i is even, cipher round is executed.
3. $\lambda = 1$ and parity of i is odd, redundant round is executed.

After the computation of every cipher round, the difference between C_0 and C_1 is transformed by Some Non Linear Function($SNLF$) which operates on each byte of the difference $(C_0 \oplus C_1)$. $SNLF$ maps all but zero byte to non-zero bytes and $SNLF(0) = 0$. Authors in [10] have suggested to use inversion in $GF(2^8)$ as $SNLF$. In case of fault injection in either cipher or redundant round, the difference $(C_0 \oplus C_1)$ is non-zero and the infection spreads in subsequent computations through R_0 and R_2 according to steps 9-11. Also, if the output of dummy round, C_2, is not β, the infection spreads in the subsequent computations through the steps 8-11. Finally in the step 14, the output of last cipher round is xored with the output of dummy round and β, and the resulting value is returned.

2.2 Attack on the Infection Countermeasure

In the absence of any side channel and with the countermeasure [10] in place, it seems difficult to identify whether a fault is injected in the target round by analysing the faulty ciphertext. For example, in the implementation of AES128 without countermeasure, if a fault is injected in the input of 9^{th} round, then the expected number of faulty ciphertext bytes which differ from the correct ciphertext is 4. In this countermeasure, the presence of compulsory dummy round ensures that the expected number of different bytes is 16 when the 9^{th} round computation is faulty. Moreover, the occurence of random dummy rounds makes it difficult to inject the same fault in both the branches of the computation.

Despite the strength of the countermeasure [10], authors in [11] showed how to attack it using a *constant byte fault model*. They observed that only one dummy round occurs after the 10^{th} cipher round of AES128, which limits the infection to only 4 bytes if the 10^{th} round's computation is faulty. The attack details are as follows:

Suppose a fault f disturbs I_1^{10}, i.e. the first byte of second row in 10^{th} cipher round input I^{10}. The difference between the faulty and redundant intermediate state after the step 7 of Algorithm 1 is:

$$R_0 \oplus R_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \varepsilon \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

where $\varepsilon = S[I_1^{10} \oplus f] \oplus S[I_1^{10}]$.
R_2 and R_0 are infected in steps 10 and 11. After the infection steps, we obtain:

$$R_0 \oplus R_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \varepsilon \oplus SNLF[\varepsilon] \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Finally, in the step 14, dummy round operates on infected R_2 which further infects R_0. But, the ShiftRow operation of dummy round shifts the infection to column 3 and leaves the faulty byte of R_0 in column 4 unmasked. The output of compulsory dummy round differs from β in column 3 and therefore, the final difference between the correct ciphertext C and faulty ciphertext C^* is:

$$\therefore C \oplus C^* = \begin{pmatrix} 0 & 0 & \beta'_8 \oplus \beta_8 & 0 \\ 0 & 0 & \beta'_9 \oplus \beta_9 & \varepsilon \oplus SNLF[\varepsilon] \\ 0 & 0 & \beta'_{10} \oplus \beta_{10} & 0 \\ 0 & 0 & \beta'_{11} \oplus \beta_{11} & 0 \end{pmatrix} \tag{1}$$

where β'_8, β'_9, β'_{10}, β'_{11} are the infected bytes of the compulsory dummy round output. Since the byte C^*_{13} is unaffected by the infected output of dummy round, it is exploited to retrieve the byte k^{11}_{13} of the 10^{th} round key using two more pairs of faulty and correct ciphertexts. Similarly, the remaining 11 key bytes corresponding to last three rows of k^{11} can be retrieved. For details on attack procedure, the reader is referred to [11].

If a fault is injected in any byte of the last three rows of I^{10}, the resulting erroneous byte is left unmasked and hence is exploited in the attack. However, if a fault is injected in any byte of the top row, the erroneous byte is masked by the infected output of compulsory dummy round. *This attack does not target the remaining 4 key bytes that correspond to the top row and they are computed using a brute force search.*

Observation 1: Ideally, the countermeasure should infect the entire result if a fault is injected in any of the rounds. But Algorithm 1 fails to protect the last round and it is exploited in the attack. Moreover, in this algorithm, the last cipher round is always the penultimate round. Thus, using a side channel, one can always observe a posteriori whether a fault was injected in the last but one round.

In the next section, we present additional flaws in the countermeasure [10] which were considered while developing the countermeasure presented in section4.

3 Further Loop Holes in the Countermeasure: Attacking the Infection Technique

It might seem that if the output of compulsory dummy round infects the erroneous byte of 10^{th} round's output, then the attack [11] can be thwarted. However, in this section, we demonstrate that the infection caused by compulsory dummy round is ineffective and can be removed.

3.1 Infection Caused by Compulsory Dummy Round

In Algorithm 1, since the input as well as the output of dummy round is β, *i.e.* $RoundFunction(\beta, k^0) = \beta$, we can write:

$$MC(SR(S(\beta))) \oplus k^0 = \beta$$

Using this relation, the xor of $RoundFunction(R_2, k^0)$ and β in step 14 of Algorithm 1 can now be expressed as:

$$\begin{aligned} RoundFunction(R_2, k^0) \oplus \beta &= MC(SR(S(R_2))) \oplus k^0 \oplus MC(SR(S(\beta))) \oplus k^0 \\ &= MC(SR(S(R_2))) \oplus MC(SR(S(\beta))) \end{aligned}$$

Since SubByte operation is the only non-linear operation in the above equation,

$$\therefore RoundFunction(R_2, k^0) \oplus \beta = MC(SR(S(R_2) \oplus S(\beta))) \tag{2}$$

If $R_2 = \beta$ then the execution of compulsory dummy round in step 14 has no effect on the final output R_0, but if $R_2 \neq \beta$ then the output of compulsory dummy round infects the final output R_0. However, this infection can be removed using the above derived equation and the desired faulty ciphertext can be recovered. On the basis of equation (2), the xor of correct ciphertext C and faulty ciphertext C^* in equation (1) can now be expressed as:

$$C \oplus C^* = \begin{pmatrix} 0 & 0 & 3 \cdot x & 0 \\ 0 & 0 & 2 \cdot x & \varepsilon \oplus SNLF[\varepsilon] \\ 0 & 0 & 1 \cdot x & 0 \\ 0 & 0 & 1 \cdot x & 0 \end{pmatrix}$$

where $x = S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus S[\beta_{13}]$ (for details refer Appendix A). Ideally, every byte of C^* should be infected with an independent random value but here the compulsory dummy round in Algorithm 1 infects only column 3 of C^* and that too, with interrelated values and leaves the rest of the bytes unmasked.

In the following discussion, we show the significance of this result, by attacking the top row of I^{10}, which was not shown in [11]. Subsequently, we show that the infection can be removed even if the fault is injected in the input of the 9^{th} cipher round. We prove this by mounting the classical Piret & Quisquater's attack [4] on the countermeasure [10].

3.2 Attacking the Top Row

We now demonstrate the attack on the top row of I^{10} to retrieve the remaining 4 bytes of k^{10}.

Suppose a fault f disturbs I_0^{10} i.e. the first byte of 10^{th} cipher round input I^{10}. The difference between the faulty and redundant intermediate state after the step 7 of Algorithm 1 is:

$$R_0 \oplus R_1 = \begin{pmatrix} \varepsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

where $\varepsilon = S[I_0^{10} \oplus f] \oplus S[I_0^{10}]$.
R_2 and R_0 are infected in steps 10 and 11. After the infection steps, we obtain:

$$R_0 \oplus R_1 = \begin{pmatrix} \varepsilon \oplus SNLF[\varepsilon] & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Finally, in the step 14, dummy round operates on infected R_2 which further infects R_0. In this case, the ShiftRow operation of dummy round does not shift the infection and the erroneous byte of R_0 in column 1 is masked. The final difference between the correct ciphertext C and faulty ciphertext C^* is:

$$\therefore C \oplus C^* = \begin{pmatrix} \varepsilon \oplus SNLF[\varepsilon] \oplus \beta'_0 \oplus \beta_0 & 0 & 0 & 0 \\ \beta'_1 \oplus \beta_1 & 0 & 0 & 0 \\ \beta'_2 \oplus \beta_2 & 0 & 0 & 0 \\ \beta'_3 \oplus \beta_3 & 0 & 0 & 0 \end{pmatrix} \tag{3}$$

where β'_0, β'_1, β'_2, β'_3 are the infected bytes of the compulsory dummy round output. and $\varepsilon = S[I_0^{10} \oplus f] \oplus S[I_0^{10}]$. Here, we cannot use the attack technique described in [11] directly, because the erroneous byte of 10^{th} cipher round has also been infected with the output of compulsory dummy round in step 14. *This is different from the case when fault is injected in any of the last three rows of* 10^{th} *cipher round input.* In order to carry out the attack [11], we need to remove the infection caused by the dummy round.
Now, we can use equation (2) to write the above matrix as:

$$C \oplus C^* = \begin{pmatrix} \varepsilon \oplus SNLF[\varepsilon] \oplus 2 \cdot y & 0 & 0 & 0 \\ 1 \cdot y & 0 & 0 & 0 \\ 1 \cdot y & 0 & 0 & 0 \\ 3 \cdot y & 0 & 0 & 0 \end{pmatrix} \tag{4}$$

where $y = S[\beta_0 \oplus SNLF[\varepsilon]] \oplus S[\beta_0]$ (for details refer Appendix B). We can use the value of $1 \cdot y$ from $C \oplus C^*$ to remove the infection from C^* and therefore unmask the erroneous byte. As a consequence, we can perform the attack suggested in [11] to get the key byte k_0^{11}. *By attacking the top row, now the attacker has the flexibility to mount the attack on any of the 12 bytes of* 10^{th} *cipher round instead of always targeting the last three rows.*

Observation 2: It is quite evident from this attack that the infection mechanism used in the countermeasure [10] is not effective. The purpose of this infection countermeasure is defeated as we can easily remove the infection and recover the desired faulty ciphertext. This is a major flaw in this countermeasure as it makes even the 9^{th} round susceptible to the fault attack which we will illustrate in the following discussion.

3.3 Piret & Quisquater's Attack on the Countermeasure

The presence of compulsory dummy round in the countermeasure [10] ensures that a fault in the 9^{th} cipher round input of AES128 infects all 16 bytes in the output. Even though the countermeasure infects all the bytes of the resulting ciphertext, we show that we can again remove the infection caused by compulsory dummy round using equation (2) and obtain the desired faulty ciphertext. To mount this attack, we consider the following two facts:

1. The authors of [10] have mentioned that an attacker can affect the *RandomBit* function in the Algorithm 1, so that the random dummy round never occurs. To counteract this effect, they added a compulsory dummy round at the end of the algorithm which ensures that the faulty ciphertext is infected in such a way that no information is available to the attacker.
2. Also, because of performance issues, Algorithm 1 should terminate within a reasonable amount of time and hence, the number of random dummy rounds should be limited to a certain value.

First, we show that if random dummy rounds never occur in the while loop, then despite the presence of compulsory dummy round in step 14, we can mount the Piret & Quisquater's attack [4] on this countermeasure and *recover the entire key using only 8 faulty ciphertexts.* Subsequently, we show that even if the random dummy rounds occur, we can still mount this attack [4].

Attack in the Absence of Random Dummy Rounds. Consider the scenario where the attacker influences the *RandomBit* function so that no dummy round occurs except the compulsory dummy round in step 14. We observe that if a fault is injected in the 9^{th} cipher round, then the rest of the computation is infected thrice. Once, after the 9^{th} cipher round in step 11, then after the 10^{th} cipher round in step 11 and finally after the execution of compulsory dummy round in step 14. To be able to mount Piret & Quisquater's attack [4], we first analyze the faulty ciphertext and identify whether a fault was injected in the input of 9^{th} cipher round. After identifying such faulty ciphertexts, we remove the infection caused by the output of compulsory dummy round and 10^{th} cipher round. Once the infection is removed, we can proceed with the attack described in [4].
The attack procedure can be summarized as follows:

1. Suppose a random fault f is injected in the first byte of the 9^{th} cipher round input. Before the execution of step 14, the output of faulty computation differs from the output of correct computation in 4 positions *viz.* 0, 13, 10 and 7 which comprises a diagonal. But the execution of compulsory dummy round in step 14 infects all the 16 bytes of the faulty computation. Therefore, the resulting faulty ciphertext T^* differs from the correct ciphertext T in 16 bytes. We use equation (2) to represent this difference as:

$$T \oplus T^* = \begin{pmatrix} m_0 \oplus 2F_1 \oplus 1F_2 & 1F_3 & 3F_4 \oplus 1F_5 \oplus 1F_6 & 3F_7 \\ 1F_1 \oplus 3F_2 & 1F_3 & 2F_4 \oplus 3F_5 \oplus 1F_6 & m_1 \oplus 2F_7 \\ 1F_1 \oplus 2F_2 & 3F_3 & m_2 \oplus 1F_4 \oplus 2F_5 \oplus 3F_6 & 1F_7 \\ 3F_1 \oplus 1F_2 & m_3 \oplus 2F_3 & 1F_4 \oplus 1F_5 \oplus 2F_6 & 1F_7 \end{pmatrix} \quad (5)$$

   where F_i, $i \in \{1, \ldots, 7\}$, represents the infection caused by the compulsory dummy round in step 14 and m_j, $j \in \{0, 1, 2, 3\}$, represents the difference between the correct and faulty computation before the execution of step 14 in Algorithm 1 (for more details refer Appendix C). Now, we can deduce the values of F_1 and F_2 from column 1, F_3 from column 2, F_4, F_5 and F_6 from column 3 and F_7 from column 4 and thus remove the infection caused by the compulsory dummy round from T^*.
2. After removing the infection caused by compulsory dummy round, we get:

$$T \oplus T^* = \begin{pmatrix} m_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_1 \\ 0 & 0 & m_2 & 0 \\ 0 & m_3 & 0 & 0 \end{pmatrix}$$

   We can now remove the infection caused by the 10^{th} cipher round. Each m_j can be written as $z_j \oplus SNLF[z_j]$, $j \in \{0, 1, 2, 3\}$, where $SNLF[z_j]$ represents the infection caused in step 11 of Algorithm 1, after the execution of 10^{th} cipher round and z_j represents the difference between the outputs of correct and faulty computations before step 11 (for more details refer Appendix C). If $SNLF$ is implemented as inversion in $GF(2^8)$, we get two solutions of z_j for every m_j. Since the 4 equations represented by m_j are independent, we obtain 2^4 solutions for $T \oplus T^*$. Here, T is known, therefore we have 2^4 solutions for T^* as well.

3. After removing the infection caused by 10th cipher round, the attacker makes hypotheses on 4 bytes of the 10th round key k^{11} and uses the faulty and correct output of 9th cipher round to verify the following relations:

$$\begin{aligned}
2 \cdot f' \oplus SNLF[2 \cdot f'] &= S^{-1}[T_0 \oplus k_0^{11}] \oplus S^{-1}[T_0^* \oplus k_0^{11}] \\
1 \cdot f' \oplus SNLF[1 \cdot f'] &= S^{-1}[T_{13} \oplus k_{13}^{11}] \oplus S^{-1}[T_{13}^* \oplus k_{13}^{11}] \\
1 \cdot f' \oplus SNLF[1 \cdot f'] &= S^{-1}[T_{10} \oplus k_{10}^{11}] \oplus S^{-1}[T_{10}^* \oplus k_{10}^{11}] \\
3 \cdot f' \oplus SNLF[3 \cdot f'] &= S^{-1}[T_7 \oplus k_7^{11}] \oplus S^{-1}[T_7^* \oplus k_7^{11}]
\end{aligned}$$

where $SNLF[b \cdot f']$, $b \in \{1, 2, 3\}$ is the infection caused in step 11, after the execution of 9th cipher round. The above set of equations is solved for all 2^4 possible values of T^* (for the complexity analysis of the attack, refer Appendix D).

Identifying Desired Faulty Ciphertexts. As done in [4], we call a ciphertext resulting from a fault injected in the input of 9th round as desired faulty ciphertext, otherwise we call it undesired. It is not difficult to identify whether the given faulty ciphertext is desired or not. With the countermeasure [10] in place, if a fault affects a byte of column i in the 9th round input, where $i \in \{0,1,2,3\}$, we observed that the following relations hold in the xor of faulty and correct ciphertext:

$$\begin{aligned}
(T \oplus T^*)_{(4\cdot(i+1))\%16} &= (T \oplus T^*)_{(4\cdot(i+1))\%16+1} \\
(T \oplus T^*)_{(4\cdot(i+1))\%16+2} &= 3 \cdot (T \oplus T^*)_{(4\cdot(i+1))\%16} \\
(T \oplus T^*)_{(4\cdot(i+3))\%16+2} &= (T \oplus T^*)_{(4\cdot(i+3))\%16+3} \\
(T \oplus T^*)_{(4\cdot(i+3))\%16} &= 3 \cdot (T \oplus T^*)_{(4\cdot(i+3))\%16+2}
\end{aligned} \tag{6}$$

where $(T \oplus T^*)_j$ represents the j^{th} byte in matrix $T \oplus T^*$. One can see from equation (5), that the above relation arises because the compulsory dummy round uses the same value to mask more than one byte of the faulty computation.

Attack Considering Random Dummy Rounds. In the attack explained above, we assumed that the attacker influences the *RandomBit* function in the countermeasure [10] so that the dummy rounds do not occur in the while loop. Now, we consider the case where the number of random dummy rounds occuring in every execution of Algorithm 1 is exactly d[1]. Since $\lambda = 0$ corresponds to a dummy round and $\lambda = 1$ corresponds to an AES round, we can view the computation of Algorithm 1 as if decided by a binary string of length $(22 + d)$, where $(22 + d)^{th}$ *RoundFunction* is always the 10th cipher round. We choose to inject the fault in $(22 + d - 2)^{th}$ round as it can be a 9th cipher or a 10th redundant or a dummy round. This increases the probability of injecting the fault in 9th cipher round.

Assuming that every string of length $(22 + d)$, consisting of exactly 22 1's and d 0's, is equally likely, then the probability that $(22 + d - 2)^{th}$ *RoundFunction* is a 9th cipher round is the same as that of a binary string of length $(22 + d)$ that ends in '111'. Since the while loop in Algorithm 1 always terminates with the execution of 10th cipher round, the binary string always ends with a 1. Therefore this probability is: $\frac{(19+d)!/((19)!\cdot(d)!)}{(21+d)!/((21)!\cdot(d)!)}$ (refer Appendix E). If $d = 20$ then the probability that 40th *RoundFunction* is a 9th cipher round is nearly 0.26.

[1] If the value of d varies across different executions, one can still compute a mean value of d by observing the number of *RoundFunctions* through a side channel.

Simulation Results. We carried out Piret & Quisquater's attack [4] on Algorithm 1 using a random byte fault model with no control over fault localization. We implemented the Algorithm 1 in C and used the GNU Scientific Library(GSL) for *RandomBit* function. The simulation details are as follows:

1. The value of d is kept constant and 1000 tests are performed.
2. Each test executes Algorithm 1 until 8 desired faulty ciphertexts are obtained. However, as the target $(22+d-2)^{th}$ $RoundFunction$ can also be a dummy or 10^{th} redundant round, the undesired faulty ciphertexts obtained in such cases are discarded. The equation set (6) can be used to distinguish between desired and undesired faulty ciphertexts.
3. An average of the faulty encryptions over 1000 tests is taken, where number of faulty encryptions in a test = (8 desired faulty ciphertext + undesired faulty ciphertexts).
4. Subsequently, the value of d is incremented by 5 and the above procedure is repeated.

The probability that the targeted $RoundFunction$ is a 9^{th} cipher round decreases with higher values of d but it still remains non-negligible. In other words, higher the value of d, more is the number of faulty encryptions required in a test as evident from Fig.1.

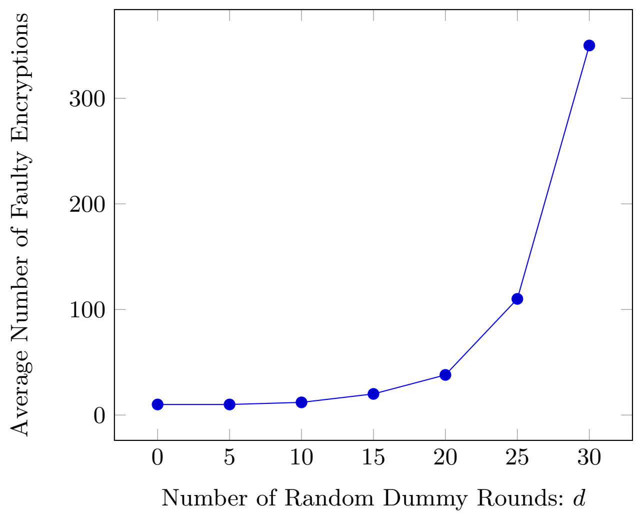

Fig. 1: Piret & Quisquater's Attack on Algorithm 1

Observation 3: The feasibility of Piret and Quisquater's attack shows that the infection method employed in the countermeasure [10] fails to protect against classical fault attacks.

4 Improved Countermeasure

In this section, we propose an improved countermeasure based upon the principles used in the Algorithm 1. The *observations* enumerated in this paper were used as a guideline for developing this countermeasure. As evident from the attacks explained earlier, the infection countermeasure for protecting AES against the differential fault attacks should have the following properties:

1. If a fault is injected in any of the cipher, redundant or dummy round, all bytes in the resulting ciphertext should be infected.
2. As shown in Section 3 of this paper, merely infecting all bytes in the output is not sufficient. Therefore, the infection technique should result in such a faulty ciphertext that any attempts to make hypothesis on the secret key used in AES are completely nullified.
3. The countermeasure itself should not leak any information related to the $RoundFunction$ computations which can be exploited through a side channel.

Given below is an algorithm, which is designed to possess all the aforementioned properties. It uses cipher, redundant and dummy rounds along the lines of Algorithm 1 but exhibits a rather robust behaviour against fault attacks.

Algorithm 2 Improved Countermeasure

```
Inputs : $P$, $k^j$ for $j \in \{1, \ldots, n\}$, $(\beta,\ k^0)$, $(n = 11)$ for AES128
Output : C = BlockCipher$(P, K)$

1. State $R_0 \leftarrow P$, Redundant state $R_1 \leftarrow P$, Dummy state $R_2 \leftarrow \beta$
2. $i \leftarrow 1$, $q \leftarrow 1$
3. $rstr \leftarrow \{0,1\}^t$        // $\#1(rstr) = 2n, \#0(rstr) = t - 2n$
4. while $q \leq t$ do
5.         $\lambda \leftarrow rstr[q]$        // $\lambda = 0$ implies a dummy round
6.         $\kappa \leftarrow (i \wedge \lambda) \oplus 2(\neg\lambda)$
7.         $\zeta \leftarrow \lambda \cdot \lceil i/2 \rceil$        // $\zeta$ is actual round counter, 0 for dummy
8.         $R_\kappa \leftarrow RoundFunction(R_\kappa, k^\zeta)$
9.         $\gamma \leftarrow \lambda(\neg(i \wedge 1)) \cdot BLFN(R_0 \oplus R_1)$        // check if $i$ is even
10.        $\delta \leftarrow (\neg\lambda) \cdot BLFN(R_2 \oplus \beta)$
11.        $R_0 \leftarrow (\neg(\gamma \vee \delta) \cdot R_0) \oplus ((\gamma \vee \delta) \cdot R_2)$
12.        $i \leftarrow i + \lambda$
13.        $q \leftarrow q + 1$
14. end
15. return$(R_0)$
```

Following additional notations are used in this algorithm:

1. **rstr**: A 't' bit random binary string, consisting of $(2n)$ 1's corresponding to AES rounds and $(t - 2n)$ 0's corresponding to dummy rounds.
2. **BLFN**: A boolean function that maps a 128 bit value to a 1 bit value. Specifically, $BLFN(0) = 0$ and for nonzero input $BLFN$ evaluates to 1.
3. $\boldsymbol{\gamma}$: A one bit comparison variable to detect fault injection in AES round.
4. $\boldsymbol{\delta}$: A one bit comparison variable to identify a fault injection in dummy round.

Apart from these elements, Algorithm 2 exhibits the following features which makes it stronger than Algorithm 1:

1. In Algorithm 2, matrix R_2 represents the state of the dummy round and is initialized to a random value β. This state matrix R_2 bears no relation with any of the intermediate states or the round keys of AES. When a fault is

induced in any of the rounds, Algorithm 2 outputs a matrix R_2. For fault analysis to succeed, the faulty output should contain some information about the key used in the cipher. However, the new countermeasure outputs matrix R_2 which is completely random and does not have any information about the key used in the AES, which makes the differential fault analysis impossible. Since in the case of fault injection, Algorithm 2 outputs dummy state R_2, the pair (β, k^0) should be refreshed in every execution[2].

2. In Algorithm 2, more than one dummy round can occur after the execution of last cipher round and consequently the 10th cipher round is not always the penultimate round.
3. Since the number of dummy rounds in Algorithm 2 is kept constant, the leakage of timing information through a side channel is also prevented.

For a clear illustration, Table 1 shows the functioning of Algorithm 2. If any of

Table 1: Computation of Algorithm 2

Step	Redundant Round	Cipher Round	Dummy Round
5.	$\lambda = 1$, i is odd	$\lambda = 1$, i is even	$\lambda = 0$
6.	$\kappa \leftarrow 1$	$\kappa \leftarrow 0$	$\kappa \leftarrow 2$
7.	$\zeta \leftarrow \lceil i/2 \rceil$	$\zeta \leftarrow \lceil i/2 \rceil$	$\zeta \leftarrow 0$
8.	$R_1 \leftarrow RoundFunction(R_1, k^{\zeta})$	$R_0 \leftarrow RoundFunction(R_0, k^{\zeta})$	$R_2 \leftarrow RoundFunction(R_2, k^0)$
9.	$\gamma \leftarrow 0$	$\gamma \leftarrow BLFN(R_0 \oplus R_1)$	$\gamma \leftarrow 0$
10.	$\delta \leftarrow 0$	$\delta \leftarrow 0$	$\delta \leftarrow BLFN(R_2 \oplus \beta)$
11.	$R_0 \leftarrow R_0$	$R_0 \leftarrow (\neg(\gamma) \cdot R_0) \oplus ((\gamma) \cdot R_2)$	$R_0 \leftarrow (\neg(\delta) \cdot R_0) \oplus ((\delta) \cdot R_2)$
12.	$i \leftarrow i + 1$	$i \leftarrow i + 1$	$i \leftarrow i + 0$
13.	$q \leftarrow q + 1$	$q \leftarrow q + 1$	$q \leftarrow q + 1$

the cipher or redundant round is disturbed, then during the computation of cipher round, $(R_0 \oplus R_1)$ is non-zero and BLFN$(R_0 \oplus R_1)$ updates the value of γ to 1. As a result, R_0 is replaced by R_2 in step 11. Similarly, if the computation of dummy round is faulty, $(R_2 \oplus \beta)$ is non-zero and δ evaluates to 1. In this case too, R_0 is replaced by R_2. Also, if the state of comparison variables γ and δ is 1 at the same time, then in step 11, R_0 is substituted by R_2 as this condition indicates fault in comparison variables themselves. In case of undisturbed execution, Algorithm 2 generates a correct ciphertext. Refer Appendix F for more details.

5 Conclusion

Recent works [6], [9] suggest the use of randomness to build sound countermeasures for protecting AES against the fault attacks. The infection countermeasure in [10] introduces the element of randomness through the use of dummy round but is still ineffective against fault attacks which target the last and penultimate round. This is because the infection uses the same unknown value to mask the erroneous byte as well as the non-erroneous bytes. One can easily deduce the value of this unknown mask from the xor of correct and faulty output. Also in the case of erroneous computation of 10th cipher round, the infection doesn't affect every byte in the faulty output. However, the modified countermeasure presented in this paper affects every erroneous as well as non-erroneous byte with independent random values irrespective of the round in which the fault is injected. Since these random values bear no relation with the intermediate output or the secret key, analysis of the resulting faulty ciphertext is a futile exercise for the attacker.

[2] One should note that even a new pair of (β, k^0) cannot protect Algorithm 1 against the attacks described in this paper.

References

1. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.
2. Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.
3. Christophe Giraud. DFA on AES. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *Advanced Encryption Standard – AES*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2005.
4. Gilles Piret and Jean-Jacques Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad. In Colin D. Walter, Çetin K. KoÇ, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.
5. Debdeep Mukhopadhyay. An Improved Fault Based Attack of the Advanced Encryption Standard. In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 421–434. Springer, 2009.
6. Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault Attacks on AES with Faulty Ciphertexts Only. In Wieland Fischer and Jörn-Marc Schmidt, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2013*, pages 108–118. IEEE Computer Society, 2013.
7. Johannes Blömer and Jean-Pierre Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In Rebecca N. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.
8. Debdeep Mukhopadhyay, Michael Tunstall, and Subidh Ali. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In Claudio A. Ardagna and Jianying Zhou, editors, *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, volume 6633 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2011.
9. Victor Lomné, Thomas Roche, and Adrian Thillard. On the Need of Randomness in Fault Attack Countermeasures - Application to AES. In Guido Bertoni and Benedikt Gierlichs, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2012*, pages 85–94. IEEE Computer Society, 2012.
10. Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology – LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 305–321. Springer, 2012.
11. Alberto Battistello and Christophe Giraud. Fault Analysis of Infective AES Computations. In Wieland Fischer and Jörn-Marc Schmidt, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2013*, pages 101–107. IEEE Computer Society, 2013.

A Execution of Infected Compulsory Dummy Round: First attack

After the execution of 10th cipher round, a fault f in I_1^{10} infects the byte β_{13} of R_2 in the step 10 of Algorithm 1:

$$R_2 = R_2 \oplus \epsilon = \begin{pmatrix} \beta_0 & \beta_4 & \beta_8 & \beta_{12} \\ \beta_1 & \beta_5 & \beta_9 & \beta_{13} \oplus SNLF[\varepsilon] \\ \beta_2 & \beta_6 & \beta_{10} & \beta_{14} \\ \beta_3 & \beta_7 & \beta_{11} & \beta_{15} \end{pmatrix}$$

where $\varepsilon = \mathrm{S}[I_1^{10}] \oplus S[I_1^{10} \oplus f]$. Thus, the input R_2 of compulsory dummy round is infected. Execution of compulsory dummy round in step 14 on the infected R_2 is shown below.
After the ShiftRow and SubByte operation:

$$R_2 = \begin{pmatrix} S[\beta_0] & S[\beta_4] & S[\beta_8] & S[\beta_{12}] \\ S[\beta_5] & S[\beta_9] & S[\beta_{13} \oplus SNLF[\varepsilon] & S[\beta_1] \\ S[\beta_{10}] & S[\beta_{14}] & S[\beta_2] & S[\beta_6] \\ S[\beta_{15}] & S[\beta_3] & S[\beta_7] & S[\beta_{11}] \end{pmatrix}$$

For clarity purpose, the output of MixColumn and KeyXor operations of only 3rd column is shown:

$$\begin{aligned} \beta'_8 &= 2 \cdot S[\beta_8] \oplus 3 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_2] \oplus 1 \cdot S[\beta_7] \oplus k_8^0 \\ \beta'_9 &= 1 \cdot S[\beta_8] \oplus 2 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 3 \cdot S[\beta_2] \oplus 1 \cdot S[\beta_7] \oplus k_9^0 \\ \beta'_{10} &= 1 \cdot S[\beta_8] \oplus 1 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 2 \cdot S[\beta_2] \oplus 3 \cdot S[\beta_7] \oplus k_{10}^0 \\ \beta'_{11} &= 3 \cdot S[\beta_8] \oplus 1 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_2] \oplus 2 \cdot S[\beta_7] \oplus k_{11}^0 \end{aligned}$$

Since $RoundFunction(\beta, k^0) = \beta$, we can write the 3rd column of β as:

$$\begin{aligned} \beta_8 &= 2 \cdot S[\beta_8] \oplus 3 \cdot S[\beta_{13}] \oplus 1 \cdot S[\beta_2] \oplus 1 \cdot S[\beta_7] \oplus k_8^0 \\ \beta_9 &= 1 \cdot S[\beta_8] \oplus 2 \cdot S[\beta_{13}] \oplus 3 \cdot S[\beta_2] \oplus 1 \cdot S[\beta_7] \oplus k_9^0 \\ \beta_{10} &= 1 \cdot S[\beta_8] \oplus 1 \cdot S[\beta_{13}] \oplus 2 \cdot S[\beta_2] \oplus 3 \cdot S[\beta_7] \oplus k_{10}^0 \\ \beta_{11} &= 3 \cdot S[\beta_8] \oplus 1 \cdot S[\beta_{13}] \oplus 1 \cdot S[\beta_2] \oplus 2 \cdot S[\beta_7] \oplus k_{11}^0 \end{aligned}$$

The remaining columns in β and in the output of dummy round are same. In step 14, the result of compulsory dummy round is xored with β.

$$\therefore RoundFunction(R_2, k^0) \oplus \beta = \begin{pmatrix} 0 & 0 & 3 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 3 \cdot S[\beta_{13}] & 0 \\ 0 & 0 & 2 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 2 \cdot S[\beta_{13}] & 0 \\ 0 & 0 & 1 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_{13}] & 0 \\ 0 & 0 & 1 \cdot S[\beta_{13} \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_{13}] & 0 \end{pmatrix}$$

B Execution of Infected Compulsory Dummy Round: Top Row Attack

After the execution of 10^{th} cipher round, a fault f in I_0^{10} infects the byte β_0 of R_2 in the step 10 of Algorithm 1:

$$R_2 = R_2 \oplus \epsilon = \begin{pmatrix} \beta_0 \oplus SNLF[\varepsilon] & \beta_4 & \beta_8 & \beta_{12} \\ \beta_1 & \beta_5 & \beta_9 & \beta_{13} \\ \beta_2 & \beta_6 & \beta_{10} & \beta_{14} \\ \beta_3 & \beta_7 & \beta_{11} & \beta_{15} \end{pmatrix}$$

where $\varepsilon = \mathrm{S}[I_0^{10}] \oplus S[I_0^{10} \oplus f]$. Thus, the input R_2 of compulsory dummy round is infected. Execution of compulsory dummy round in step 14 on the infected R_2 is shown below.
After the ShiftRow and SubByte operation:

$$R_2 = \begin{pmatrix} S[\beta_0] \oplus SNLF[\varepsilon] & S[\beta_4] & S[\beta_8] & S[\beta_{12}] \\ S[\beta_5] & S[\beta_9] & S[\beta_{13} & S[\beta_1] \\ S[\beta_{10}] & S[\beta_{14}] & S[\beta_2] & S[\beta_6] \\ S[\beta_{15}] & S[\beta_3] & S[\beta_7] & S[\beta_{11}] \end{pmatrix}$$

For clarity purpose, the output of MixColumn and KeyXor operations of only 3^{rd} column is shown:

$$\beta'_0 = 2 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 3 \cdot S[\beta_5] \oplus 1 \cdot S[\beta_{10}] \oplus 1 \cdot S[\beta_{15}] \oplus k_0^0$$
$$\beta'_1 = 1 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 2 \cdot S[\beta_5] \oplus 3 \cdot S[\beta_{10}] \oplus 1 \cdot S[\beta_{15}] \oplus k_1^0$$
$$\beta'_2 = 1 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_5] \oplus 2 \cdot S[\beta_{10}] \oplus 3 \cdot S[\beta_{15}] \oplus k_2^0$$
$$\beta'_3 = 3 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_5] \oplus 1 \cdot S[\beta_{10}] \oplus 2 \cdot S[\beta_{15}] \oplus k_3^0$$

Since $RoundFunction(\beta, k^0) = \beta$, we can write the 1^{st} column of β as:

$$\beta'_0 = 2 \cdot S[\beta_0] \oplus 3 \cdot S[\beta_5] \oplus 1 \cdot S[\beta_{10}] \oplus 1 \cdot S[\beta_{15}] \oplus k_0^0$$
$$\beta'_1 = 1 \cdot S[\beta_0] \oplus 2 \cdot S[\beta_5] \oplus 3 \cdot S[\beta_{10}] \oplus 1 \cdot S[\beta_{15}] \oplus k_1^0$$
$$\beta'_2 = 1 \cdot S[\beta_0] \oplus 1 \cdot S[\beta_5] \oplus 2 \cdot S[\beta_{10}] \oplus 3 \cdot S[\beta_{15}] \oplus k_2^0$$
$$\beta'_3 = 3 * S[\beta_0] \oplus 1 \cdot S[\beta_5] \oplus 1 \cdot S[\beta_{10}] \oplus 2 \cdot S[\beta_{15}] \oplus k_3^0$$

The remaining columns in β and in the output of dummy round are same. In step 14, the result of compulsory dummy round is xored with β.

$$\therefore RoundFunction(R_2, k^0) \oplus \beta = \begin{pmatrix} 2 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 2 \cdot S[\beta_0] & 0 & 0 & 0 \\ 1 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_0] & 0 & 0 & 0 \\ 1 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 1 \cdot S[\beta_0] & 0 & 0 & 0 \\ 3 \cdot S[\beta_0 \oplus SNLF[\varepsilon]] \oplus 3 \cdot S[\beta_0] & 0 & 0 & 0 \end{pmatrix}$$

C Diffusion of fault and infection in Piret & Quisquater's attack

In this appendix, we explain how the fault diffuses and infects the computation of Algorithm 1, when a fault is injected in the input of 9^{th} cipher round. Let I^9 denote the input to the 9^{th} cipher round. Suppose a fault f is injected in the first byte of 9^{th} cipher round input I^9.

$$I^9 = \begin{pmatrix} I_0^9 \oplus f & I_4^9 & I_8^9 & I_{12}^9 \\ I_1^9 & I_5^9 & I_9^9 & I_{13}^9 \\ I_2^9 & I_6^9 & I_{10}^9 & I_{14}^9 \\ I_3^9 & I_7^9 & I_{11}^9 & I_{15}^9 \end{pmatrix}$$

After the execution of 9^{th} cipher round in the step 7 of Algorithm 1, the difference between the faulty and redundant intermediate state is:

$$R_0 \oplus R_1 = \begin{pmatrix} A & 0 & 0 & 0 \\ B & 0 & 0 & 0 \\ C & 0 & 0 & 0 \\ D & 0 & 0 & 0 \end{pmatrix}$$

where $A = 2 * f'$, $B = 1 * f'$, $C = 1 * f'$ and $D = 3 * f'$.
After infection in step 11, this difference is:

$$R_0 \oplus R_1 = \begin{pmatrix} A \oplus SNLF[A] & 0 & 0 & 0 \\ B \oplus SNLF[B] & 0 & 0 & 0 \\ C \oplus SNLF[C] & 0 & 0 & 0 \\ D \oplus SNLF[D] & 0 & 0 & 0 \end{pmatrix}$$

In step 10, R_2 is also infected.

$$R_2 = \begin{pmatrix} \beta_0 \oplus SNLF[A] & \beta_4 & \beta_8 & \beta_{12} \\ \beta_1 \oplus SNLF[B] & \beta_5 & \beta_9 & \beta_{13} \\ \beta_2 \oplus SNLF[C] & \beta_6 & \beta_{10} & \beta_{14} \\ \beta_3 \oplus SNLF[D] & \beta_7 & \beta_{11} & \beta_{15} \end{pmatrix}$$

Since 10^{th} redundant round executes without any error, after the execution of 10^{th} cipher round in the step 7 of Algorithm 1, the difference between the faulty and redundant computation is:

$$R_0 \oplus R_1 = \begin{pmatrix} z_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & z_1 \\ 0 & 0 & z_2 & 0 \\ 0 & z_3 & 0 & 0 \end{pmatrix}$$

where $z_0 = S[I_0^{10} \oplus A \oplus SNLF[A]] \oplus S[I_0^{10}]$, $z_1 = S[I_1^{10} \oplus B \oplus SNLF[B]] \oplus S[I_1^{10}]$, $z_2 = S[I_2^{10} \oplus C \oplus SNLF[C]] \oplus S[I_2^{10}]$, $z_3 = S[I_3^{10} \oplus D \oplus SNLF[D]] \oplus S[I_3^{10}]$.
In step 11, R_0 is further infected, therefore the difference between faulty and redundant computation at the end of the while loop is:

$$R_0 \oplus R_1 = \begin{pmatrix} m_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_1 \\ 0 & 0 & m_2 & 0 \\ 0 & m_3 & 0 & 0 \end{pmatrix}$$

where $m_j = z_j \oplus SNLF[z_j]$, $j \in \{0, 1, 2, 3\}$.
R_2 is also infected in the step 10. After infection, R_2 is

$$\begin{pmatrix} \beta_0 \oplus SNLF[A] \oplus SNLF[z_0] & \beta_4 & \beta_8 & \beta_{12} \\ \beta_1 \oplus SNLF[B] & \beta_5 & \beta_9 & \beta_{13} \oplus SNLF[z_1] \\ \beta_2 \oplus SNLF[C] & \beta_6 & \beta_{10} \oplus SNLF[z_2] & \beta_{14} \\ \beta_3 \oplus SNLF[D] & \beta_7 \oplus SNLF[z_3] & \beta_{11} & \beta_{15} \end{pmatrix}$$

Thus, at the end of the while loop, 4 bytes of R_0 and 7 bytes of R_2 are infected.

In step 14, when compulsory dummy round operates on R_2, the infection in the input of R_2 spreads to all the 16 bytes. Using equation (2), we can write the final difference $T \oplus T^*$ as:

$$T \oplus T^* = \begin{pmatrix} m_0 \oplus 2F_1 \oplus 1F_2 & 1F_3 & 3F_4 \oplus 1F_5 \oplus 1F_6 & 3F_7 \\ 1F_1 \oplus 3F_2 & 1F_3 & 2F_4 \oplus 3F_5 \oplus 1F_6 & m_1 \oplus 2F_7 \\ 1F_1 \oplus 2F_2 & 3F_3 & m_2 \oplus 1F_4 \oplus 2F_5 \oplus 3F_6 & 1F_7 \\ 3F_1 \oplus 1F_2 & m_3 \oplus 2F_3 & 1F_4 \oplus 1F_5 \oplus 2F_6 & 1F_7 \end{pmatrix}$$

where $F_1 = S[\beta_0 \oplus SNLF[A] \oplus SNLF[z_0]] \oplus S[\beta_0]$, $F_2 = S[\beta_{10} \oplus SNLF[z_2]] \oplus S[\beta_{10}]$, $F_3 = S[\beta_3 \oplus SNLF[D]] \oplus S[\beta_3]$, $F_4 = S[\beta_{13} \oplus SNLF[z_1]] \oplus S[\beta_{13}]$, $F_5 = S[\beta_2 \oplus SNLF[C]] \oplus S[\beta_2]$, $F_6 = S[\beta_7 \oplus SNLF[z_3]] \oplus S[\beta_7]$, and $F_7 = S[\beta_1 \oplus SNLF[B]] \oplus S[\beta_1]$.

D Complexity Analysis

A random byte fault in the input of 9th cipher round results in 2^4 solutions for T^*. Every solution of T^* gives 1036 candidate values for 4 bytes of the 10th round key k^{11} as described in [4]. Thus the expected number of candidate values for 4 bytes of k^{11} is $2^4 * 1036 = 16576$. If we repeat this attack process on another pair of faulty and correct ciphertext we expect to get no more than 2 values for 4 bytes of k^{11} [4]. Our experiments also reveal that we are left with at most 2 candidate values for every 4 bytes of k^{11}.

E Probability Computation

Consider a set $L = \{s \in \{0,1\}^{n+d}: \#|1| = n \wedge \#|0| = d\}$.
The number of unique binary strings, consisting of exactly n 1's and d 0's, i.e. $\mid L \mid$ is $(n+d)!/(n! \cdot d!)$.
Thus, the number of unique binary string (S_{total}) consisting of $n = 21$ 1's and d 0's $= (21+d)!/(21! \cdot d!)$.
And the number of unique binary string ($S_{favourable}$) consisting of $n = 21$ 1's, d 0's and ending with '11' $= (19+d)!/(19! \cdot d!)$.
Therefore, the probability of uniformly selecting a binary string from the set L with $n = 21$ and terminating with '11' $= \frac{S_{favourable}}{S_{total}}$.

F Values of Bit Variables during the Execution of Algorithm 2

Table 2: Status of Variables during Execution of Algorithm 2

$\boldsymbol{i\%2}$	$\boldsymbol{\lambda}$	γ	$\boldsymbol{\delta}$	**comments**
1	1	0	0	correct computation of redundant round
0	1	0	0	correct computation of cipher round
X	0	0	0	correct computation of dummy round
0	1	1	0	detection of fault in AES round
X	0	0	1	detection of a fault in dummy round
X	X	1	1	detection of fault in comparison bit variable