Detecting and locating faults in VLSI implementations of the Advanced Encryption Standard

Guido Bertoni[1], Luca Breveglieri[1], Israel Koren[2], Paolo Maistri[1], Vincenzo Piuri[3]

[1]Department of Electronics and Information Technology
Politecnico di Milano, Milano, ITALY
[2]Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA
[3]Department of Information Technologies
Università di Milano, Crema, ITALY

bertoni@elet.polimi.it, brevegli@elet.polimi.it, koren@ecs.umass.edu, maistri@elet.polimi.it, piuri@dti.unimi.it

Abstract

Concurrent fault detection for hardware implementations of the Advanced Encryption Standard (AES) *may provide protection against random faults, and against an attacker who may maliciously inject faults in order to find the encryption secret key. We have recently developed such a scheme which is based on the parity code. In this paper we prove that the parity-based code detects all odd-order faults and allows the location of most single transient and permanent faults.*

1 Introduction

Fault detection and tolerance schemes for various implementations of cryptographic algorithms have recently been considered. The motivation is twofold: there exist attacks for breaking the encryption code, which are based on the deliberate injection of faults [3, 6, 8]; and, the circuit implementation of cryptographic algorithms can be quite complex, increasing the likelihood of device failures [5]. Preliminary work on introducing fault detection (and possibly tolerance) in hardware implementations of cryptographic units includes [4] where a redundancy-based fault detection scheme was proposed for the Advanced Encryption Standard (AES), while [7] proposes a detection scheme based on time redundancy.

A different approach for fault detection in implementations of AES has been proposed in [2] and further developed in [1]. This approach is based on the simple idea of using the well-known parity bit error detection code (EDC) with a number of parity bits, and has been proven to be very successful in achieving a high fault coverage, for both transient and permanent faults, at the cost of reasonable hardware overhead.

It was shown in [1], based on simulation experiments, that the proposed coding scheme detects all odd-order transient faults, but a complete formal proof was not given. Moreover, the question arises whether the parity-based scheme would also allow the *location* of faults (transient and permanent), which would facilitate developing fault tolerance techniques.

The goal of this paper is to further develop the parity-based coding scheme for AES, in order to answer these open questions. To this end, we developed a precise mathematical model describing the diffusion of errors caused by faults in the computational flow of AES. This model allows us to prove in a formal way that the parity-based coding scheme achieves a 100% coverage for transient faults of odd order. Moreover, this model allows us to show that the parity-based coding scheme is also capable of locating all single-bit transient and most permanent faults at the byte level.

The paper is organized as follows. In Section 2 a brief review of the AES encryption algorithm, Rijndael, is presented. In Section 3 the essentials of the error model for AES are described. In Section 4 we prove that our parity-based coding scheme achieves 100% coverage for transient faults of odd order. In Section 5 the fault location capabilities of the parity-based code are analyzed for transient single-bit faults and for permanent faults. Section 6 summarizes our results and presents future research directions.

2 The Rijndael algorithm

The AES algorithm, Rijndael [5], is composed of three procedures: Key Schedule, Encryption and Decryption. The most commonly used version of AES inputs a secret key and a plain text block of size 128 bits each, and outputs a ciphered text block of 128 bits. The encryption algorithm is iterative and consists of the repetition of 10 rounds. Each round processes the data block and adds to it (modulus 2) a round key, derived from the secret key by means of the key schedule algorithm. Figure 1 shows the encryption and key schedule procedures; this is also the architecture of the simplest VLSI implementation of AES, and is assumed in this paper.

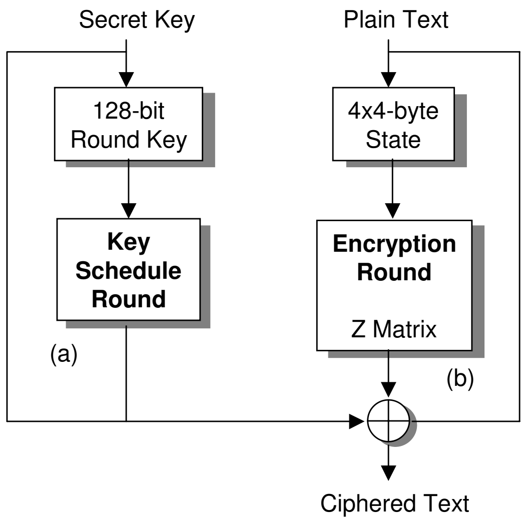

Figure 1. *Block diagram of Rijndael: Key Schedule (a) and Encryption (b)*

AES works on the 16 bytes of the data block to be encrypted by structuring it into a square matrix of order 4×4 bytes. This is called the "state" matrix and is shown in Figure 2. The body of one AES encryption round is a chain of four transformations, called: SubBytes (or Sbox), ShiftRows, MixColumns and AddRoundKey. Each transformation applies to the entries of the state S some algebraic operations (addition, multiplication and inversion), both linear and non-linear, which are defined over a finite field, namely the Galois Field $GF\left(2^8\right)$ [9]. The four transformations are:

SubBytes: byte-wise non-linear substitution.

ShiftRows: rotation of the rows of the state S.

MixColumns: linear algebraic transformation of the columns of the state S.

AddRoundKey: bit-wise addition of the current round key to the state S.

More details about AES can be found in [5] and also in [1].

$$S = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

Figure 2. *State matrix*

3 AES error model

In this section the coding scheme [1] and the associated error model of AES are described. The model describes the way errors diffuse in the AES computation flow. Attention is focused on encryption, while key schedule and decryption are assumed to be fault-free.

The coding scheme associates a set of parity bits with the state S, which are predicted at every round transformation. Checkpoints, where the predicted and generated parity bits are compared, may be scheduled at the end of each round. The total number of such checkpoints and their exact position can be determined by the designer but at least one check must be performed at the end of the last round.

A parity-based coding scheme is in principle capable of detecting every fault which results in an error involving the complementation of an odd number of bits in the state S, and also of an even number of bits, provided at least one byte of the state S contains an odd number of erroneous bits.

3.1 Preliminaries

Each byte element $s_{r,c}$ of the state matrix S is associated with a parity bit $p_{r,c}$, for every $0 \leq r, c \leq 3$. We define an error bit $e_{r,c}$ as follows:

$$e_{r,c} = p\left(s_{r,c}\right) + p_{r,c} \qquad \text{for } 0 \leq r, c \leq 3$$

where $p\left(\right)$ is the parity operator and addition is modulo 2. Since even parity is used, the value $e_{r,c} = 0, 1$ indicates whether the 9-bit sequence $s_{r,c}, p_{r,c}$ has a correct (even) or incorrect (odd) parity status, respectively.

In order to study the propagation of incorrect parity states during the rounds, a 4×4 error state matrix E is defined, as in Figure 3. To investigate the dispersion of errors in the encryption process, it suffices to examine how the round transformations modify the error state matrix E. This depends on the prediction of the output parity bits; the details are described in [1], while here only a brief overview is given.

$$E = \begin{bmatrix} e_{0,0} & e_{0,1} & e_{0,2} & e_{0,3} \\ e_{1,0} & e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,0} & e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,0} & e_{3,1} & e_{3,2} & e_{3,3} \end{bmatrix}$$

Figure 3. *Error matrix*

SubBytes (or Sbox). Sbox is usually implemented as a look-up table. The output parity bits are stored with the data bits. Hence, the error status of each input byte element propagates unaltered through Sbox. Therefore, Sbox maps the error state matrix E to itself.

AddRoundKey. The behavior is the same as above, since the round key produced by the key schedule is assumed to be error-free.

ShiftRows. The parity states of the input bytes are preserved but rotated.

MixColumns. The error state matrix is modified by a linear transformation, which is described in [1].

Of course, it is also necessary to include a set of parity generators for computing the parity bits of the data block, and a set of comparators for checking them against the predicted parity bits.

3.2 Error model for the AES Encryption

It is possible to determine analytically how the error state E is modified through the computation flow of one AES encryption round, and hence of a sequence of rounds.

We first define a column error vector V with 16 entries, obtained by scanning the error state matrix E by columns:

$$V = [e_{0,0}, e_{1,0}, e_{2,0}, e_{3,0}, e_{0,1}, e_{1,1}, \ldots, e_{3,3}]^T$$

V contains the same information as the error state matrix E, only differently organized.

One AES encryption round modifies the column error vector V as follows: $V \mapsto ZV$. The square matrix Z is of order 16 and is shown in Figure 4. The proof can be obtained by a careful analysis of the internal transformations of the round, but is omitted for the sake of brevity. Clearly, the application of $n \geq 1$ AES encryption rounds is represented by $V = Z^n V$. It is quite easy to check that matrix Z is orthogonal (over $GF(2)$), and hence is non-singular.

$$\begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0
\end{pmatrix}$$

Figure 4. *The matrix Z modeling the error dispersion in one AES round.*

4 Coverage of the faults of odd order

In this section we prove that the parity-based EDC for the encryption algorithm of AES has a fault coverage of 100% at the byte level for multiple transient faults of odd order. The proof is based on Linear Algebra techniques and uses the above-introduced error dispersion model.

To prove the coverage of faults, it is first necessary to define a metric for errors, and then investigate the effect of AES on the dispersion of the errors themselves. Let $X = [x_i]$ be a vector of order $n \geq 1$ with entries x_i over $GF(2)$. Define the *parity norm* $\|X\|_2 \in GF(2)$ of X to be $\|X\|_2 = \sum_{i=0}^{n-1} x_i = x_0 + x_1 + \cdots + x_{n-1}$ with all additions modulus 2. By definition, for every X_1 and X_2 of equal order it holds that $\|X_1 + X_2\|_2 = \|X_1\|_2 + \|X_2\|_2$. We extend now the parity norm to a sequence $\underline{X} = (X_1, X_2, \ldots, X_h)$ of $h \geq 1$ vectors X_i, as follows: $\|\underline{X}\|_2 = \sum_{i=1}^{h} \|X_i\|_2 = \|X_1\|_2 + \cdots + \|X_h\|_2$.

Multiplying a column vector times a square orthogonal matrix does not change the parity norm of the vector. Therefore, the application of $n \geq 1$ AES encryption rounds does not change the parity norm of the column error vector V; that is: $\forall n \geq 1 \quad \|Z^n V\|_2 = \|V\|_2$.

Next, it is shown how AES acts upon the error metric defined above. A multiple transient fault of order $k \geq 1$ at the byte level consists of exactly k single-bit errors at the beginning of the rounds, assuming that an odd number of bits is commuted. All injection patterns are assumed to be equally likely. Two bit errors may well be injected into the same byte position of the data block, in two different rounds.

Let $n \geq 1$ be the number of AES rounds, numbered $1, \ldots, n$. A multiple fault $\underline{m}$ in the AES encryption algorithm can be represented as follows: $\underline{m} = \langle \underline{n}, \underline{X} \rangle$, where $\underline{n} = (n_1, n_2, \ldots, n_h)$ $(1 \leq n_i \leq n)$ is a strictly increasing sequence of $h \leq n$ integers, and $\underline{X} = (X_1, X_2, \ldots, X_h)$ is a sequence of $h \leq n$ column error vectors X_i of order 16. Vector X_i represents the bit errors injected during the encryption process at the beginning of round number i $(1 \leq i \leq n)$.

Given a multiple fault $\underline{m}$, the order $ord\,(\underline{m})$ of $\underline{m}$ is defined to be the total number of entries of value 1 in the vectors of the sequence $\underline{X}$. Moreover, the parity norm of $\underline{m}$ is defined as follows: $\|\underline{m}\|_2 = \|\underline{X}\|_2$. Based on these definitions, it is possible to see that $\|\underline{m}\|_2 = ord\,(\underline{m}) \mod 2$. Based on linearity, it follows that $encryption\,(\underline{m}) = \sum_{i=1}^{h} Z^{n-n_i} X_i$. Now, the fundamental point is that, given any multiple fault $\underline{m} = \langle \underline{n}, \underline{X} \rangle$, the AES encryption preserves its parity norm, that is: $\|encryption\,(\underline{m})\|_2 = \|\underline{m}\|_2$. The proof is not shown here, however it depends on the linearity of the error model of the AES encryption round and on the orthogonality of the matrix Z.

As a consequence, the parity-based EDC for the AES encryption algorithm detects all multiple faults of odd order. Let $\underline{m}$ be a multiple fault of odd order: it follows that $\|\underline{m}\|_2 = 1$ and hence $\|encryption\,(\underline{m})\|_2 = \|\underline{m}\|_2 = 1$. A fault is detectable if and only if the final column error vector V is non-null. Suppose that $encryption\,(\underline{m}) = \underline{O}$, then it follows that $\|encryption\,(\underline{m})\|_2 = \|O\|_2 = 0$. But this would contradict the previous conclusion.

The conclusion holds also in the case when the faulty bits of the multiple fault $\underline{m}$ are injected between the individual round transformations, not only at the beginning of the rounds. To prove this fact, it is only necessary to model in a finer way the action of encryption over the multiple fault.

5 Localization of faults

In this section we show that the parity-based EDC is not only able to detect transient faults (at the byte level), but also exhibits fault locating properties. In fact, by analyzing the final error signatures (i.e., which out of the 16 bytes have erroneous parity bits), we can identify the faulty byte and the round in which the fault has occurred for the first time, in most relevant cases.

One such relevant case is that of a single-bit transient fault. Such a fault can be represented by an error state matrix E containing a single entry of value 1. Observe the following simulation, showing how the error state matrix evolves after each round (for a total of 8 rounds), starting from a matrix showing the injection of a single bit error into the leftmost top byte of the error state of the initial round.

Beginning of round 1	**End of round 1**	**End of round 2**	**Rounds from 3^{rd} to 7^{th}**	**End of Round 7**	**End of Round 8**
1 0 0 0	1 0 0 0	1 0 1 0	....	1 0 0 0	1 0 0 0
0 0 0 0	1 0 0 0	1 0 0 1	....	0 0 0 0	0 0 0 0
0 0 0 0	1 0 0 0	1 0 1 1	....	0 0 1 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 1 1	....	0 0 0 1	0 0 0 0

Note that the above error state matrices differ from one another, though after round 8 the initial matrix is obtained (and from that round on the simulation is cyclic). Should this property extend to all the remaining error state matrices associated with single-bit faults (in total 16 such matrices) generating unique error matrices, it would be possible to locate the fault and the round it appeared in, with a latency of at most 8 rounds.

	Z^0	Z^1	Z^2	Z^3	Z^4	Z^5	Z^6	Z^7
X_1	1	7	60679	21713	43946	21077	44359	33793
X_2	2	57344	3803	35626	22357	43594	36443	32801
X_3	4	3328	46861	29777	44714	21848	42781	1057
X_4	8	176	2942	35374	23893	6826	11102	33824
X_5	16	112	53374	19733	47786	9557	54394	16408
X_6	32	14	60848	45736	30037	42154	58808	536
X_7	64	53248	28891	17687	60074	21893	29146	16912
X_8	128	2816	47072	41704	54613	43681	46562	16904
X_9	256	1792	2029	53588	43691	21842	18349	388
X_{10}	512	224	56078	10891	21847	19114	23438	8576
X_{11}	1024	13	3511	20852	43694	22613	7591	8452
X_{12}	2048	45056	32267	11914	21853	43546	24107	8324
X_{13}	4096	28672	32464	5453	43706	21797	31444	6208
X_{14}	8192	3584	45293	43186	21877	43684	47333	6146
X_{15}	16384	208	56176	5957	43754	34133	55921	4162
X_{16}	32768	11	57527	59554	21973	41386	58037	2114

Figure 5. *Matrix of the* $Y_{n,h} = Z^n X_h$ *signatures* $(0 \le n < 8,\ 1 \le h \le 16)$.

Figure 5 shows the results $Y_{n,h}$ of the expressions $Z^n X_h$ $(0 \le n < 8,\ 1 \le h \le 16)$ for all the vectors X_h with exactly one entry of value 1 at position h, where $Z^0 = I$. The vectors $Y_{n,h}$ are displayed in decimal form (considering them as natural integers of 16 bits). For example, the first row in Figure 5 corresponds to the matrices in the example above with the decimal values 1, 7, 60679, …, 33793 and 1. Each vector $Y_{n,h}$ represents the error signature, after n rounds, caused by the injection of the single-bit error vector X_h. The entries of the table in Figure 5 are all different from one another, thus confirming that it is possible to locate all single-bit transient faults (round number and byte position), with a latency of at most 8 rounds.

The above mentioned property can be formally proven. First of all, note that $Z^8 = I$, i.e., the powers of the matrix Z constitute a cyclic group of period 8. This is a direct consequence of the fact that Z is orthogonal and thus, invertible. The value 8 of the period can be proven by direct computation, which is omitted. We must, therefore, position the parity checkpoints at a distance of 7 rounds or less in order to allow the location of faults.

Let X_h be a (column) error vector, having a single entry of value 1 at position h $(1 \le h \le 16)$. Then $Z^n X_h$ $(0 \le n < 8)$ yields a vector having a single entry of value 1 if and only if column h of Z contains a single entry of value 1. But the eight matrix powers Z^n of Z do not contain columns having a single entry of value 1 (this can be checked by direct computation, here omitted). Now, two identical signatures $Y_{n_1,h_1} = Y_{n_2,h_2}$ $(0 \le n_1, n_2 < 8)$ would imply $Z^{n_1} X_{h_1} = Z^{n_2} X_{h_2}$, hence (since Z is invertible), assuming for instance $n_1 \le n_2$, $X_{h_1} = Z^{n_2 - n_1} X_{h_2}$, i.e. $X_{h_1} = Z^m X_{h_2}$, with $0 \le m < 8$. But this is impossible, as said before.

The above procedure for the location of transient faults can be extended to permanent faults as well. The location procedure is however more sophisticated, since a permanent fault manifests itself as a sequence of errors injected into the error state E at several rounds, in a fixed byte position, but in a time and data dependent way. However, the error dispersion model presented in this paper can be exploited for this purpose as well. A straightforward way to identify the location of a permanent fault is through computing the matrix of all the signatures that might occur. This two-dimensional matrix has 16 columns (which correspond to the bytes where the permanent fault can occur) and 255 rows. Each row corresponds to one of the possible different manifestations of a permanent fault during the eight rounds. Depending on the input data to the cipher and the type of fault (short or open circuit), a permanent fault might affect the computation of the given byte at any one of the eight rounds. Omitting the case where no fault is manifested, this yields $2^8 - 1 = 255$ different possibilities for a permanent fault to inject errors during the eight rounds.

	0	1	2	3	4	5	6	7	8	9	...	15
0x01	1	2	4	8	16	32	64	128	256	512	...	32768
0x02	33793	32801	1057	33824	16408	536	16912	16904	388	8576	...	2114
0x03	33792	32803	1061	33832	16392	568	16976	17032	132	9088	...	34882
...	...	...	...	...	...	...	...	...	...	...	...	...
0x11	43947	22359	44718	23901	47802	30069	60138	54741	43947	22359	...	54741
...	...	...	...	...	...	...	...	...	...	...	...	...
0x77	15163	56540	52942	29555	46003	52685	60652	14135	15163	56540	...	14135
0x78	16425	30958	14510	50863	660	36583	35555	27388	10560	61048	...	64618
0x79	16424	30956	14506	50855	644	36551	35491	27260	10304	60536	...	31850
...	...	...	...	...	...	...	...	...	...	...	...	...
0x87	10560	61048	44600	44998	37890	59278	58250	64618	16425	30958	...	27388
...	...	...	...	...	...	...	...	...	...	...	...	...
0xFF	26985	38550	38550	26985	38550	26985	26985	38550	26985	38550	...	38550

Figure 6. *Excerpt from the matrix of permanent fault signatures*

An excerpt of the permanent fault signature matrix is shown in Figure 6. The rows correspond to the "temporal manifestation" of the permanent fault, i.e., which rounds are actually affected by the fault. For example, row 1 corresponds to the case where only the first round is affected, while row 79_{16} (whose bit representation is 0111 1001) corresponds to the case where the $1^{st}, 4^{th}, 5^{th}, 6^{th}$ and 7^{th} rounds were affected. All the fault signatures in the matrix are represented as a decimal value, using the same notation used in Figure 5. The matrix includes $16 \cdot 255 = 4080$ entries, out of which 3072 are unique, allowing to immediately identify the byte in which the permanent fault has occurred and the rounds during which the permanent fault has manifested itself. There is no overlap with the matrix shown in Figure 5, except for those cases where the permanent fault manifests itself only once, thus actually behaving as a transient fault. The remaining 1008 entries include:

- The 240 entries of the fifteen rows corresponding to temporal fault manifestations which are represented by two equal hex digits (e.g., `0x11, 0x22, ..., 0xFF`). Several entries in these rows appear multiple times within the same row but are not equal to any entry of another row. Thus, we can identify the rounds where the permanent fault has manifested itself but the byte can not be immediately identified. For 12 out of these 15 rows each entry appears twice within that row (e.g., row `0x11`) with the two candidate bytes at a distance of 8 positions apart. In the two rows `0x55` and `0xAA` each entry appears 4 times while in the row `0xFF` each entry appears 8 times.
- The 768 entries of 24 pairs of rows, e.g., rows `0x78` and `0x87` (see Figure 6), where the entries in one row are identical to those in the second row but are rotated by 8 positions. For every fault signature in these 48 rows we have 2 admissible temporal manifestations (i.e., the same signature appears in two rows) and 2 admissible fault locations (i.e., the two identical signatures in the two rows appear in two different bytes and the byte positions are at a distance of 8 apart). The first rows in the

24 pairs of rows include `0x0F, 0x1E, 0x2D, 0x3C, 0x4B, 0x5A, 0x69, 0x78` and also `0x05, 0x0A, 0x14, 0x28, 0x27, 0x93, 0xC9, 0xE4, 0xAF, 0xD7, 0xEB, 0xF5, 0x1B, 0x8D, 0xC6, 0x63`. The index of the corresponding row in each pair is obtained by rotating the row index by 4 bit positions (i.e., exchanging the two hex digits).

Although each of the above 1008 fault signatures does not allow an immediate fault localization, we can repeat the encryption process with a different data input (and/or a different key) which would very likely lead to a different temporal manifestation of the same permanent fault. The new fault signature may, in principle, be again one of the 1008 entries which do not allow us to identify the fault location. However, the likelihood of such an occurrence is very small. The probability that a permanent fault will result in one of the 1008 signatures is (assuming uniform distribution) $\frac{1008}{4080} = 0.247$. If the second test is independent of the first one then the probability that two consecutive tests will fail to identify the fault location is $0.247^2 = 0.061$.

All the fault signatures in the matrix shown in Figure 6 may also correspond to two or more transient faults occurring in several randomly selected bytes and during several randomly selected rounds. For example, consider a permanent fault occurring in the first byte, similar to the example at the beginning of this section, and manifesting itself only in the first two rounds. The manifestation of the fault in the second round may cancel the parity error in the first byte leaving only the second and third bytes with an incorrect parity. The decimal value of the error matrix will be now 6 instead of 7. The same error matrix will be obtained if instead of the above permanent fault, two transient faults in the second and third bytes will occur at the beginning of round 2. In both cases, i.e., the single permanent fault manifesting itself only in the first two rounds and the two transient faults indicated above, will yield the same final signature of 33792 (see Figure 6).

Therefore, we can evaluate the probability of successfully identifying the fault location only under certain assumptions regarding the type of faults which can occur. If we assume that only single transient faults can occur, then the probability of locating the fault is 1.000, but if we also allow a single permanent fault to occur, then:

$$\mathcal{P}\{\text{Properly Locating the Fault}|\text{A Single Fault occurred}\} = \frac{4080 - 1008}{4080} = 0.753$$

If the obtained fault signature after eight rounds is equal to one of the 1008 signatures discussed above, an immediate identification of the fault is not possible. However, a second experiment would usually suffice to resolve the ambiguity. Besides ambiguous fault signatures we may also mis-identify a permanent fault if two separate transient faults occur during the eight rounds. We have analyzed the possible fault signatures which can be obtained for all double transient faults and realized that out of these $\binom{128}{2}$ signatures, 384 also appear in the signature matrix for permanent faults. Thus,

$$\mathcal{P}\{\text{Mis-identifying a Permanent Fault}|\text{A Double Transient Fault occurred}\} = \frac{384}{\binom{128}{2}} = 0.047$$

6 Conclusion

In this paper we have developed an analytical error model for the parity-based EDC for the AES encryption algorithm. The model allows us to achieve two goals: formal proof of 100% detection coverage for multiple transient faults (at the byte level), and the development of a procedure for locating single-bit transient and permanent faults. A possible extension would be to classify all multiple faults yielding a particular error signature. The fault locating procedures are likely to extend to the implementations of the key schedule and decryption, since these algorithms share the same basic operations as encryption.

The availability of a relatively simple and low-cost coding scheme for locating faults in a hardware implementation of AES opens the way to the introduction of fault tolerance, based on reconfiguration. Since AES architectures have an iterative or regular structure at various levels of abstraction, reconfiguration seems feasible and a promising future research direction.

Acknowledgment: The work of Israel Koren has been supported in part by DARPA/AFRL NEST program under contract number F33615-02-C-4031.

References

[1] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Error Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," *IEEE Transactions on Computers*, Special Issue on Cryptographic Hardware and Embedded Software, pp. 492-505, April 2003.

[2] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "A Parity Code Based Concurrent Fault Detection for Implementations of the Advanced Encryption Standard," *Proc. of the 2002 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 51-59, November 2002.

[3] D. Boneh, R. DeMillo, R. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," *Journal of Cryptology*, vol. 14, pp. 101-119, 2001.

[4] R. Karri, W. Kaijie, P. Mishra, K. Yongkook, "Fault-based Side-Channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture," *Proc. of the 2001 Defect and Fault Tolerance in VLSI Systems*, pp. 418-426, 2001.

[5] B. Gladman, "A Specification for Rijndael, the AES Algorithm," http://fp.gladman.plus.com/, 2001.

[6] M. Akkar, C. Giraud, "An Implementation of DES and AES, Secure against some Attacks," *Proceedings of CHES '01*, pp. 315-325, 2001.

[7] S. Fernández-Gòmez, J. Rodríguez-Andina, E. Mandado, "Concurrent Error Detection in Block Ciphers," *Proc. of the 2000 Intern. Test Conference*, ITC '00, pp. 979-984, 2000.

[8] F. Bao, R. Deng, Y. Han, A. Jeng, D. Narasimhalu, T. Nagir, "Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults," *The Second Workshop on Secure Protocols, (Pads)*, April, 1997, LNCS, Springer-Verlag, 1997.

[9] R. Lidl, H. Niederreiter, *Introduction to Finite Fields and their Applications*, Cambridge University Press, 1986.

[10] J. Blöemer and J.-P. Seifert, *Fault based cryptanalysis of the Advanced Encryption Standard*, Cryptology ePrint Archive, Report 2002/075, 2002.

[11] S. Skorobogatov, R. Anderson, "Optical Fault Induction Attacks," *Proc. of 2002 IEEE Symposium on Security and Privacy*, 2002.