Detecting Faults in Four Symmetric Key Block Ciphers

L. Breveglieri[1], I. Koren[2], Paolo Maistri[1]

[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

{brevegli,maistri}@elet.polimi.it, koren@ecs.umass.edu

Abstract

Fault detection in encryption algorithms is gaining in importance since fault attacks may compromise even recently developed cryptosystems. In this work, we analyze the different operations used by various symmetric ciphers and propose possible detection codes and frequency of checking. Several examples (i.e., AES, RC5, DES and IDEA) are presented to illustrate our analysis.

1: Introduction

Recently, schemes for detecting faults in hardware implementations of several symmetric key encryption algorithms have been developed. The motivation behind the increased interest in such detection schemes is based on two important observations. First, ciphered communication is very sensitive to errors in the input data or faults occurring during the computation, due to the strong non-linearity of the encryption functions. Analysis of the effect of faults occurring during the encryption process, described in [2] for the Advanced Encryption Standard (AES) algorithm and in [4] for RC5 [18], has shown that even a single bit error leads, after only a few rounds of the algorithm, to a completely corrupted result.

The second reason for the increased importance of fault detection, besides the data integrity issue, is the observation that attacks based on fault injection are feasible [6]. The authors of [6] showed that a cryptographic device computing DES could be compromised by injecting a fault during the computation. Depending on the cipher algorithm employed, useful data can be extracted by analyzing the resulting erroneous output. This approach was later applied successfully to more recent algorithms, such as AES [7, 11]. It is worthwhile to note that fault injection attacks are not limited to symmetric block ciphers, although the latter received the most attention in recent publications. In [10] it is shown how an error in some parameters of an ECC (Elliptic Curve Cryptosystem) may reveal information which could lead to the secret key; in [1], fault injection attacks against RSA-capable smart cards were studied. The authors of [8] showed how an erroneous RSA signature can lead to an easier factorization of the modulus, thus breaking the cryptosystem.

Some preliminary studies of fault detection schemes have already been performed. In [12], Karri et al. have proposed to use the existing hardware for an immediate decryption of the cipher text. They

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

rely on the fact that the unit responsible for decryption is normally not used during the encryption process and is often independent of the encryption datapath since it involves the inverse transformations. In US patent [9] the embedding of error detection capabilities in DES was proposed: additional bits are generated from the input and provided to the ciphering device in parallel with the plain text. These bits, acting as check bits for the error detecting code, are propagated and updated during the encryption process. Since the cipher is mainly based on substitution tables, these tables are extended in order to include the additional check bits. Checking for inconsistencies at the end of the process may thus reveal a possible error in the computation.

A different approach is needed when the operations are more complex than substitutions. Some research has been performed in this direction in [3, 4, 13]. In [3], an error detection code was proposed for the AES cipher. The proposed code is based on the well-known parity code and uses one parity bit for each byte of the 128-bit-long input. In [4], a similar approach is applied to RC5.

In this paper we analyze the different types of basic arithmetic operations employed by encryption algorithms and list different suitable fault detection schemes. We then recommend appropriate fault detection techniques for the analyzed ciphers. We also discuss the required frequency of error checking, Since, due to error masking, error indication may disappear, we indicate for each cipher whether to check for errors at every round or only when the encryption is complete.

2: Symmetric ciphers

This section identifies the operations used by the various ciphers, and makes some preliminary observations regarding fault detection in dedicated VLSI implementations of ciphers. To this end, we consider two principal approaches to fault detection, namely, duplication (the brute force solution) and the use of an error detecting code. The latter can be either more or less efficient than duplication depending on the structure of the selected code.

Table 1. Four Symmetric ciphers and the operations they use in the Encryption (or Decryption) data-path (the sizes of the operands are indicated).

		Operations - Data-Path of the Encryption part of the Cipher							
	Input		mod n						mod $G(x)$
Ciphers	**Size**	XOR	$+$	$\times$	Expansion	S-box	Rotation	Permutation	$\times$
DES	64	32, 48			$32 \rightarrow 48$	$6 \rightarrow 4$		1	
IDEA	64	16	16	16				16	
RC5	64	32	32				32		
Rijndael	128	8				$8 \rightarrow 8$		8	8

The list of the symmetric ciphers considered in this paper (see Table 1) is far from being exhaustive; however, it includes the winner of the last AES competition [16], together with some other previously proposed algorithms like DES [15], RC5 [18] and IDEA [14]. In what follows, the analysis of the fault detection issues is restricted to the Encryption data-path of the ciphers, excluding both the Key Schedule and the Decryption parts. Software implementations for all four ciphers are available and all have been used in practice to some extent. Two out of the four, namely DES and Rijndael (AES), are more commonly used and have dedicated VLSI devices; however, the other two are also

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

well suited for VLSI implementations. RC5, especially, would yield very low cost implementations due to its extreme simplicity.

Usually, symmetric ciphers have an iterative structure, which greatly simplifies the design and implementation of fault detection mechanisms.
The encryption process of a data block consists of repeating a number of identical rounds (sometimes two or more alternating round types). Each round may consist of a series of internal transformations, and uses a round key, derived from the secret key. The operations, which may be applied to the whole data block or to part of it, are:

- Bit-wise XOR (Exclusive OR); this is a modular arithmetic (modulo 2) operation.
- Modular addition, subtraction and multiplication of integers; the modulus depends on the cipher, but is usually 2^w or $2^w + 1$ (with $w = 16$ or 32).
- Expansion, meaning that the data block is expanded to a larger number of bits.
- S-Box, or Substitution-Box. It is included in many different ciphers, and consists of a replacement of bytes or words by means of a look-up table. The way it is defined varies with the cipher, and there does not seem to exist any general rule.
- Rotation and shift of the bytes or words of the data block.
- Permutation of the bits, bytes or words of the data block.
- Polynomial modular multiplication of the bytes or words of the data block. Usually one of the two factors is fixed, so that multiplication actually reduces to scaling.

All the above operations, considered in isolation, admit specific error detection codes (EDCs), and some are so simple and inexpensive as to allow duplication. Still, there does not seem to exist an EDC which is inherently optimized for all of them. The designers of the various symmetric ciphers have provided some qualitative explanations of the reasons for preferring certain basic operations in the internal transformations. These reasons are based on the need of achieving diffusion (each bit of the input should affect every bit of the output) and confusion (all the regularities of the input are uniformized in the output) while processing the input data block.

The presence of explicitly non-linear operations makes the design of an error detecting code more difficult and the code may become inefficient. The mixture of different and incompatible algebraic structures may also pose a problem: a code which is efficient for one structure may be very inefficient when applied to the other. This may force the insertion of a checkpoint and the generation of new check bits for a different code.

The above mentioned criteria are however qualitative, and still leave many degrees of freedom. We stress that, as far as we know, there exists no specific and well-defined criterion for designing reliable and fault-resistant symmetric cyphers. Moreover, none of the ciphers listed in Table 1 has been implemented in a way that specifically addresses these issues.

We next make some detailed observations regarding Table 1. First, there is only one operation among those analyzed, namely XOR, which is used in all the ciphers considered. Note also that expansion is used only by DES; polynomial multiplication (or scaling) is used only by Rijndael (AES); rotation is used only by RC5, while natural multiplication is used only by IDEA. All the remaining operations are common to several ciphers.

The following details regarding Table 1 are necessary for the rest of the paper. The operation $(+)$ is integer addition modulo 2^w with $w = 16$ or 32. The operation $(\times)$ is integer multiplication modulo $2^{16} + 1$. Expansion consists of transforming 4-bit nibbles into sequences of 6 bits. S-Box

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

is a substitution of bit sequences and its definition depends on the cipher. Permutation consists of the exchange of bits, bytes or words. Polynomial multiplication is defined over the finite field $GF\left(2^8\right)$ with one factor always a constant.

3: Operations and error detection codes

This section provides a brief overview of the fault detection techniques used in this paper and discusses their application to the selected ciphers. We focus on the basic multiple transient fault model, which is appropriate for algorithms operating on relatively large data blocks (64 to 128 or more bits). Although our analysis is based on the hypothesis of multiple bit errors, we still consider and analyze first the single bit error, which allows a more precise description of the model.

Two types of error detection techniques are discussed: duplication followed by comparison of the results, and error detecting codes (EDCs). Clearly, duplication can be applied to every cipher and has a 100% fault coverage. It is however a brute force solution and has a high hardware overhead.

Error detecting codes are, in principle, more promising than duplication: they may achieve a relatively high coverage, at least for low order errors, with a relatively low hardware overhead. Moreover, EDCs could be applied to the entire data block or to parts thereof (bytes, words), thus allowing several variants. We therefore focus in the next sections mainly on employment of EDCs. The basic ones are arithmetic residue codes with the modulus 3, 7 and 15 and parity codes. Residue codes are most suited for modular arithmetic operations, while parity codes are appropriate for logical and polynomial operations.

Note that the use of these EDCs implies the need of a code (used initially and ahead of every checkpoint), a set of code prediction circuits (one for each internal operation), and a comparator for comparing the actual and predicted check bits at each checkpoint. Since all ciphers are iterative, scheduling checkpoints implies deciding on the checkpoint frequency in the round flow and whether the checkpoint should be executed in the middle or at the end of the round. The selected schedule depends on the tradeoff between the desired fault coverage and the acceptable hardware and performance overhead.

All the above listed EDCs can be applied at different levels of granularity. Table 1 suggests that the right levels to select from are byte level (8 bits) and word level (16 or 32 bits), since most internal operations work on data of such sizes. Applying an EDC at the level of the entire data block does not seem to be a good choice since the data block is large and is always fragmented into shorter bit sequences for processing. A global code would entail a large overhead for prediction while providing only a low fault coverage. We therefore consider residue and parity codes at the byte and word level.

The parity code is obtained by XORing all the bits of the word or byte, depending on the level of the code. The residue code of a word is obtained by taking the modulo $(2^s - 1)$ of the word itself, where s is the number of check bits. A simple way of computing the residue is taking the weighted sum of the word bits: $res(A) = res(a_{n-1} \dots a_0) = \sum_{i=0}^{n-1} a_i \; res(2^i)$.

3.1: Feasibility of EDCs

Table 2 shows the estimated cost of the various EDCs listed above, when applied to the operations of the ciphers in Table 1. We distinguish between EDCs which have an acceptable cost (a "yes"

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

entry in the table) and expensive EDCs (an "exp." entry). An EDC is considered expensive when its implementation (generation, prediction and comparison of check bits) requires an overhead comparable to that of duplication. We next provide explanations on the entries of Table 2. The

Table 2. The cost of applying error correcting codes to the various operations: exp. - very expensive; yes - reasonably feasible.

	Per byte			Per word			
Operation	**Parity**	**Res.** 3	**Res.** 7	**Parity**	**Res.** 3	**Res.** 7	**Res.**. 15
XOR	yes	yes	yes	yes	yes	yes	yes
$+ \mod n$	yes	yes	yes	yes	yes	yes	yes
$\times \mod n$	exp.	yes	yes	exp.	yes	yes	yes
Expansion	separate treatment (applies only to DES)						
S-Box	yes	exp.	exp.	yes	exp.	exp.	exp.
Rotation	yes	yes	exp.	yes	yes	exp.	yes
Permutation	yes	yes	yes	yes	yes	yes	yes
$\times \mod G(x)$	yes	exp.	exp.	yes	exp.	exp.	exp.

generation and prediction techniques of the various EDCs are omitted for brevity. Residue base 15 is considered at the level of words of 16 or 32 bits. It is not considered at the byte level since it uses 4 check bits resulting in 50% bit overhead, which is unacceptable for an error detection code.

Both parity and residue codes are reasonable for bit-wise XOR, although parity is obviously more suitable: residue code prediction must be corrected by subtracting the residue of the logical AND of the operands (see [4]). Expansion is used only by DES: due to the unconventional size of the input to the S-Box, any code is quite expensive. S-Box is a non-linear substitution, hence its treatment is more complex and is detailed below.

Both parity and residue codes are feasible for integer modular addition and subtraction; still, residue is better suited than parity code, which can be an acceptable solution for a single byte (parity prediction must consider all the carries generated in the operation). Residue code is appropriate for integer modular multiplication, while parity is expensive since all the intermediate carries must be considered. In contrast, parity is feasible for polynomial multiplication in $GF\left(2^8\right)$ when one of the two factors is fixed (scaling) [3], while the residue code is expensive. Finally, both parity and residue codes incur a reasonable overhead for rotation at the byte and word level, with the exception of residue base 7 [2]. This holds also for permutations at the byte and word levels.

3.1.1: S-Box

S-Box is a non-linear substitution which is usually implemented by means of a look-up table. Two kinds of faults are possible: those affecting the contents of the look-up table and those affecting the address decoder. Both can be detected in the following way. The 8 input bits are inputted to three units: (1) The ordinary S-Box look-up table; (2) a look-up table providing the correct parity bit for the corresponding entry in the above S-Box table; and (3) an input parity checker. The parity bit generated by (2) is then XORed with the parity check provided by (3) so that the final parity bit is incorrect if the parity of the incoming byte was wrong. When the input is fault-free, the correct parity is produced; if this is not the case, an incorrect result is propagated and its associated parity is deliberately altered.

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

A similar scheme can be applied to residue codes with modulus 3, 7 or 15, but the overhead will be larger since the table (unit (2) above) will be larger than that for the parity code.

3.2: Preferred EDCs

Some preliminary conclusions can be drawn regarding the preferable error detection code for each symmetric cipher. Most operations allow simple prediction rules both for parity and residue codes. Few operations are an exception to this rule: integer multiplication, for instance, allows only for residue prediction: this forces the choice of residue codes for ciphers employing integer multiplications, such as IDEA. On the other hand, expansion in DES and polynomial multiplication in AES are better suited to parity codes, which are hence the suggested choice for those ciphers. As stated in Section 3.1.1, the addressing and storage overheads for residue code prediction in S-Boxes suggest that parity code is preferable.

Table 3. Suggested error detecting codes for protecting various symmetric ciphers.

Cipher	Suggested Code
DES	Parity
IDEA	Residue, but expensive
RC5	Parity or Residue
Rijndael (AES)	Parity, per byte

IDEA uses only exclusive ORs, natural additions and modular multiplications. However, the product uses the modulus $\left(2^{16}+1\right)$ so the parity code is not a reasonable choice, but even residue codes are expensive since the unusual modulus makes the computation of the corrective term a very complex task. Other operations have affordable prediction rules for both codes which can hence be reasonable choices, such as in the case of RC5. The results are summarized in Table 3.

4: Frequency of checking

In the previous sections we examined various options for detecting faults in the studied symmetric ciphers. All the detection techniques (i.e., parity and residue, at the byte or word level), are able to detect a single transient fault (see for instance [17]). This means that, if checking for errors is performed at the end of each internal transformation of every round, the coverage of single bit transient faults is 100%.

However, this is a very high checking frequency with a considerable hardware overhead. Since the ciphers are all iterative and consist of a repetition of a basic round, it might be sufficient to check less frequently, for instance once at the end of each round or even only once at the end of the whole sequence of rounds. This will speed up the clock rate and reduce the time latency. In a pipelined architecture, reducing the check frequency allows to implement fewer checkers and thus reduce the hardware overhead as well.

We next study this possibility for three of the previously mentioned ciphers, namely Rijndael (AES), RC5 and DES, since these exhibit different behaviors. Define the *error signature* as the difference between the actual values and the predicted values of the check bits for the data block. Then, the

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

exact error propagation model depends on the particular cipher studied. We assume that the key scheduling algorithm is fault-free, and the rounds following the ones affected by the fault are also fault-free: consequently, they *evolve* the error signature by spreading or canceling the errors.

To determine how an error signature propagates throughout the cipher, it is necessary to examine how the predicted values of the check bits at the *output* of each transformation contained in the round depend on the values of these bits at the *input* of the transformation.

Let D be the data block to be processed. The error signature is represented by a vector $E = [e_j]$ satisfying

$$\begin{aligned} e_j &= r_j - p_j \\ \text{where} & \\ r_j &= \text{actual check bits of the } j^{th} \text{ element} \\ p_j &= \text{predicted check bits of the } j^{th} \text{ element} \end{aligned}$$

and where the operator $(-)$ depends on the algebraic structure over which the EDC is defined. In the absence of errors, $r_j = p_j$, and thus the null vector O is the signature corresponding to the error-free case.

In general, the prediction rule of the EDC check bits in any internal transformation of the round has the following form:

$$p'_j = f_j\left(p_1, p_2, \ldots, p_m, D\right)$$

where p_j are the predicted check bits from the preceding transformation. Ideally, the new predicted EDC check bits p'_j should depend only on the p_j's, but in most cases there is also a dependence on part of the data block D. Note however that only the dependence on the p_j's is essential for determining the propagation of the errors. If, for some values of the datum D, the p'_j's were independent of the p_j's, then the propagation chain of the error signature would be interrupted and a checkpoint would have to be inserted.

By definition, the element e'_j of the error signature at the output of an internal transformation is

$$e'_j = f_j\left(r_1, \ldots, r_m, D\right) - f_j\left(p_1, \ldots, p_m, D\right)$$

The above formula can be rewritten as follows:

$$e'_j = f_j\left(p_1 + e_1, \ldots, p_m + e_m, D\right) - f_j\left(p_1, \ldots, p_m, D\right) \tag{1}$$

Equation (1) gives the propagation rule of the error signature. In simple cases the right hand side of the above equation depends only on the e_j's (and not on the p_j's or D). More generally, it is frequently possible to separate the dependence on e_j from that on p_j and the datum D. This is not always true and must be confirmed in every case, though most EDCs satisfy it. We stress that, should this be false, the analysis of the propagation of the error signature would be much more complex, since it would depend on the datum D as well.

Thus, the error signature is updated as follows: $E' = F\left(E\right)$, where F is some function determined by f_j in equation (1). In most cases, F proves to be a linear error propagation model or is reducible to a composition of linear models, thus simplifying considerably the error signature propagation rules.

The error propagation analysis is therefore very dependent on the considered cipher. The analysis for AES is partially outlined in [3], and can serve as a guideline for the remaining ciphers. The

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

checkpoints for AES were scheduled only at the end of the encryption: as shown in [5], the evolution of the error state model can be described by a 16×16 matrix defined over $GF(2)$: this matrix is non-singular and its 8^{th} power is the Identity matrix. Hence, any single fault will propagate (spreading and contracting) to the end and will never be completely canceled. Even the RC5 cipher gives similar results [4]: both residue and parity codes can be propagated through the operations used in RC5, and the propagation model (not described here for brevity) allows a single fault to reach the comparator at the end of the encryption.

DES requires a completely different approach: due to its permutation unit, which acts at the bit level, it is not possible to derive a simple prediction rule for the check bits. Hence, a checkpoint must be scheduled *within* the round, and then the code check bits must be regenerated. As shown in Section 5, the inner checkpoint makes the check at the round level redundant, at least for the word parity code. This is the approach used in [9], where only inner and final checkpoints were scheduled, and none at the end of the round. IDEA also requires internal checkpoints, due to the prediction overhead for its multiplications. However, since there are 4 multiplications per round, the checkpoint implementation would be extremely costly in terms of overhead and latency.

Such frequent checking is required also when the code is larger-grained than the operations: e.g., consider 8-bit S-Boxes and a word parity code; before accessing the look-up table, we need to check the consistency of the code, compute the S-Box and finally recompute the new code, with an additional global overhead.

5: Detection coverage of multiple faults

The AES cipher was already studied in [3], with respect to parity code at the byte level. Here we briefly summarize those results: the interested reader can refer to the original paper for further details. The parity code is a natural choice due to the use of polynomial multiplication in $GF(2^8)$ and achieves impressive results due to the high regularity and symmetry of AES. The parity code is able to detect all the faults that cause an odd number of errors. Those causing an even number of errors are mostly detected, but 100% coverage is not guaranteed. The most likely case of undetectable errors is pairs of errors occurring in the same byte, which are intrinsically not detectable by parity codes. Most notably, the parity code also allows for the location of the fault [5].

The results of our study of the RC5 cipher are shown in Figures 1 and 2. The experiments were run both with parity (Figure 1) and residue codes (Figure 2); the level of redundancy was chosen out of 1, 2 or 4 bits per word; checkpoints were scheduled at each round or just at the end of the encryption. Exhaustive simulations were performed for one or two faults; larger sets (from 3 to 20) were tested by random generation of $1,000,000$ plain text, key and injection combinations. The figures show the percentage of undetected faults using a logarithmic scale. Single faults are not considered, since they are always detected by the codes.

The continuous lines show the detection capability of the code when the check is performed at the end of the entire encryption. The result is strictly dependent on the level of redundancy. The dotted lines show the percentage of undetected faults when checking is performed at the end of each round. This percentage decreases exponentially with the number of faults; this is a result of the fault model we chose where each fault is uniformly distributed over all rounds. Note that the residue code modulo 15 is the only code that is able to detect the faults in all the test cases; very few fault sets are not detectable by this code if every round is checked, and those cases did not occur in the simulation. The percentage of undetected faults for this code is thus 0% and is not

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

shown on the graph. On the other hand, all odd-order faults are fully detected only by round-level check of the parity codes. This does not apply to residue codes.

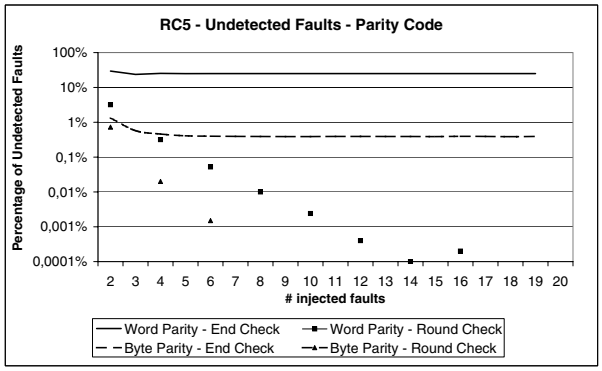

Figure 1. *Undetected faults in the RC5 encryption datapath with parity codes.*

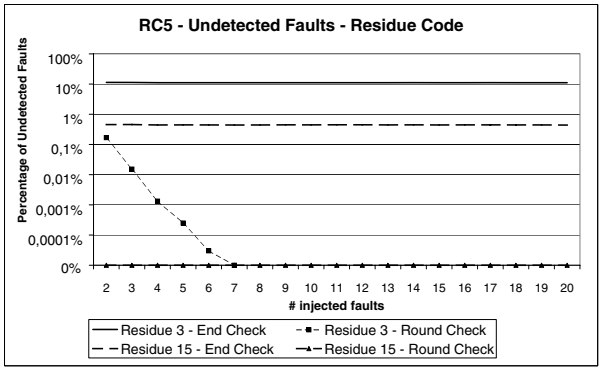

Figure 2. *Undetected faults in the RC5 encryption datapath with residue codes.*

The results of similar experiments for the DES cipher are shown in Figures 3 and 4. Only the parity code was studied; the level of redundancy was set to 1 (Figure 3) or 4 (Figure 4) bits per word; checkpoints were scheduled at the end of encryption, at the end of each round or even within the round, between the S-Boxes and the permutation unit. Parity codes allow detecting all odd-order faults, therefore only even-order faults are shown. The same pattern is used in both figures when the same frequency of checkpoints is followed.

Note that one parity bit per word gives very poor detection capability: checking at the end allows detecting only half the injected faults, because parity must be regenerated after the substitution boxes anyway even if the data is already corrupted. In fact, the S-Boxes act at a finer level than the parity code. Moreover, if we use just one single bit per word, there is no difference between checking at the end of each round and checking also within the round. The finer-grained checkpoints results in a decreased percentage of undetected faults as the number of faults increases.

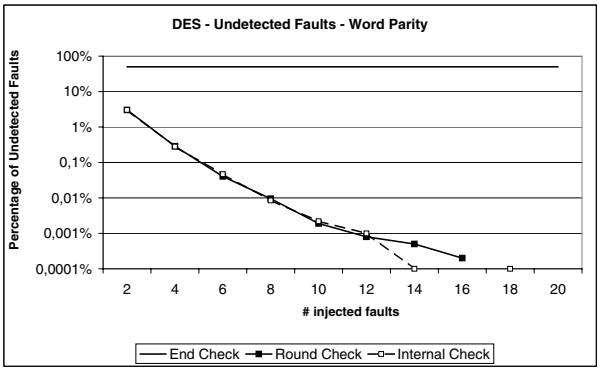

Figure 3. *Undetected faults in the DES encryption datapath with word parity code.*

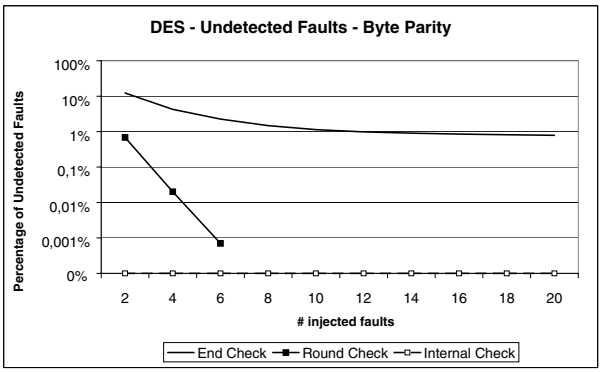

Figure 4. *Undetected faults in the DES encryption datapath with byte parity code.*

If we increase the level of redundancy to 4 parity bits per word, the behavior of the parity code depends heavily on the frequency of checkpoints. Checking at the end of each round gives a good protection against even-order faults, and it scales well when increasing the amount of injected

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

faults. Checking within the round allows detecting all the injected faults, which was confirmed also by an extensive simulation of the double fault injection. The only drawback is the presence of some false positives: a fault in the unmodified word can be canceled by a similar fault in the other word at the beginning of the following round. This is obvious, since Feistel ciphers swap the two halves of the input at each round. Checking at each round signals these cases as false positives, which doesn't happen when the checkpoint is scheduled only at the end of encryption. Finally, checking this code at the end of encryption gives moderate results, with the percentage of detected faults ranging from about 90% to 99%.

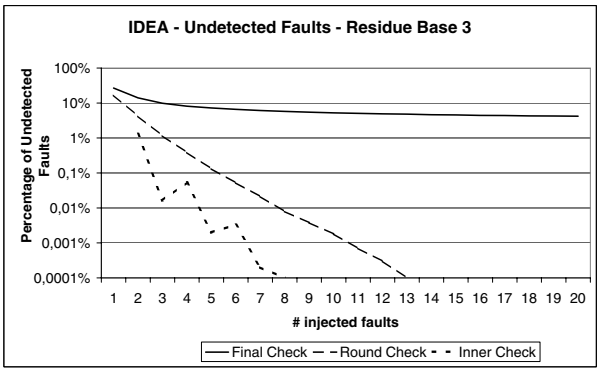

Figure 5. *Undetected faults in the IDEA encryption datapath with residue-3 code.*

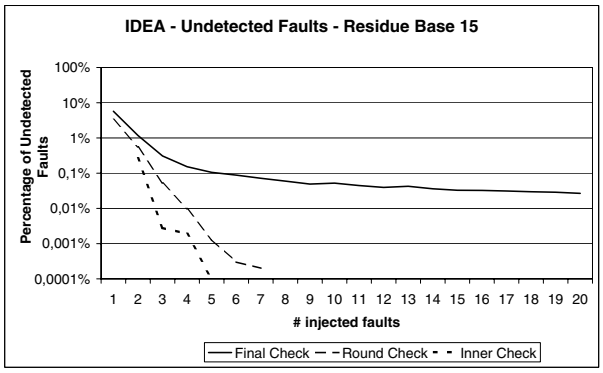

Figure 6. *Undetected faults in the IDEA encryption datapath with residue-15 code.*

Figures 5 and 6 depict the results of similar experiments performed on the IDEA cypher using residue codes, which are the only choice (though an expensive one) due to the uncommon 16-bit multiplication. Exhaustive simulations were performed for single and double injected faults. When injecting three or more faults, $1,000,000$ random inputs and keys were generated for each number of injected faults. Checkpoints were scheduled at three different levels (as for DES), where the internal ones were located just before executing products. This was justified by the fact that predicting through multiplication is expensive, hence the code should be verified and then regenerated after the operation is completed. Since IDEA uses 16-bit words, we chose radix-3 and radix-15 residue codes for each word, which introduce 12.5% and 25% redundancy, respectively. The considerations made in the case of DES hold here as well, with some interesting differences: first, only the finest-grained checkpoint frequency allows to detect all the single faults. Moreover, the detection percentage is quite poor when compared to that of AES [3], and internal checkpoints are mandatory if we want to break the 99%-detection barrier. Round-level checkpoints are a good compromise, while a unique final checkpoint gives the lowest detection rate. The situation improves when using radix-15 residue code, but the overhead introduced by the redundancy becomes as high as 25%, not including the additional cost due to the prediction units and comparators.

6: Conclusions

In this paper we have presented suggestions for providing fault detection capabilities in recent block ciphers. Some preliminary results have been shown regarding AES, DES, RC5 and IDEA. Observations were made that can be extended to other ciphers on the basis of the operations included in each datapath. In particular, we have shown that the detection capability of any code

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

depends on the type of the code, the frequency of checkpoints and the level of redundancy.

Acknowledgment: The work of Israel Koren has been supported in part by DARPA/AFRL NEST program under contract number F33615-02-C-4031. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Projects Agency, AFRL, or the US Government.

References

[1] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, J.-P. Seifert, "Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures," http://citeseer.nj.nec.com/525626.html, 2002.

[2] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "On the Propagation of Faults and their Detection in a Hardware Implementation of the Advanced Encryption Standard," *Proc. of the IEEE International Conf. on ASAP '02*, pp. 303-312, 2002.

[3] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "Error analysis and detection procedures for a hardware implementation of the Advanced Encryption Standard," *IEEE Transactions on Computers,* Volume 52, Issue 4, pp. 492-505, 2003.

[4] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "Concurrent Fault Detection in a Hardware Implementation of the *RC*5 Encryption Algorithm," *Proc. of the IEEE International Conference on ASAP 2003*, pp. 410-419, 2003.

[5] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "Detecting and locating faults in VLSI implementations of the advanced encryption standard," *Proc. of 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT'03* pp. 105 - 113, 2003.

[6] E. Biham, A. Shamir, "Differential Cryptanalysis of the Data Encryption Standard," Springer-Verlag, 1993.

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

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

[9] A. S. Butter, C. Y. Kao, J. P. Kuruts, "DES encryption and decryption unit with error checking," US patent US5432848, July 1995.

[10] M. Ciet and M. Joye, "Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults," http://citeseer.nj.nec.com/ciet03elliptic.html, 2003.

[11] C. Giraud, "DFA on AES," http://citeseer.nj.nec.com/558158.html, 2003.

[12] R. Karri, K. Wu, P. Mishra, K. Yongkook, "Fault-based side-channel cryptanalysis tolerant Rijndael symmetric block cipher architecture," *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2001*, pages 427-435, 2001.

[13] R. Karri, G. Kuznetsov, M. Goessel, "Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers," *Proceedings of CHES 2003*, Springer-Verlag, pages 113-124, 2003.

[14] X. Lai, J. Massey, S. Murphy, "Markov Ciphers and Differential Cryptanalysis," *Lecture Notes in Computer Science, Advances in Cryptology*, proc. of EuroCrypt '91, pp. 17-38, 1991.

[15] National Bureau of Standard, "Data Encryption Standard," *FIPS pub. n. 46*, US Dept. of Commerce, 1977.

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

[17] W. Peterson, E. Weldon, *Error-Correcting Codes*, 2^{nd} ed., The MIT Press, Cambridge, MA, U.S.A., 1972.

[18] R. Rivest, "The RC5 Encryption Algorithm," *K. U. Leuven Workshop on Cryptographic Algorithms*, Springer-Verlag, 1995.

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE