

Low Cost Self-Testing Implementation for MISTY1 Cryptographic Algorithm

Rodica Tirtea, Mircea Vladutiu, Geert Deconinck, *Member, IEEE*

Abstract—Side-channel attacks (i.e. fault analysis attacks) exploit vulnerabilities generated by faults in cryptographic implementations. Given the consequences of a successful attack (which can retrieve key information with a quite low cost), error detection mechanisms need to be implemented to protect cryptographic implementations. However the available mechanisms generate large overhead both in hardware and time and other low cost error detection techniques are required.

We propose in this paper a new error detection technique, relying on information redundancy. This technique uses complemented duplication. A case study is presented for MISTY1 cryptographic algorithm. The error propagation for MISTY1 is analyzed. Trade-off analysis of different resources required for detection mechanisms is included. The cost of the detection mechanism using this technique is lower than the costs of the previously available techniques.

I. INTRODUCTION

THE standardized encryption algorithms are computationally secure, and, usually they are designed to resist to all known attacks targeting their mathematical model. However a secure mathematical model is not enough in practice. Side channel attacks were developed, and their underlying idea is to exploit the way cryptographic algorithms are implemented rather than the algorithm itself.

A cryptographic primitive (i.e. block cipher, digital signature scheme, etc), when implemented in software or in hardware, requires the use of devices, and, these devices are interacting with the environment. Data (e.g. power consumption, timing) can be gathered based on these interactions in order to assembly information about algorithms (e.g. operation being performed, operands values). As such the implementation is vulnerable to the external environment (e.g. variation of power supply, clock rate) which may cause effects useful for an attacker (e.g. faults).

In this paper we address fault analysis attack [1]. Exploitation of faults in the implementation can reveal information regarding the key used for encryption. Such exploits have been reported for almost all standardized

algorithms (public-key or symmetric-key encryption/decryption, digital signatures, identification schemes, etc).

Error detection mechanisms have been recently deployed in cryptographic implementations in order to overcome fault analysis attacks. In this paper the proposed methods and mechanisms for error detection are identified and a new technique using *complemented duplication* is described. This technique is applied to MISTY1 encryption algorithm, a 64-bit block cipher with 128-bit key, evaluated and selected in both NESSIE [2] and CRYPTREC [3] evaluation initiatives.

This paper has the following structure. First the fault analysis attack is presented. This gives the reasons for developing error detection mechanism in cryptographic implementation. An overview of proposed error detection mechanisms is given in the second section. Complemented duplication error detection technique is presented in section three. In section four, MISTY1 is shortly described and in section five we analyze the error propagation for the algorithm. Using an architecture built based on the technique from section three, implementation results are presented and then analyzed in section six. In the last section the conclusions are summarized.

II. PROPOSED TECHNIQUES FOR DETECTING FAULT ANALYSIS ATTACKS

A. Fault Analysis Attacks

In 1996, Boneh, DeMillo and Lipton [1] presented a theoretical model for breaking various cryptographic schemes by taking advantage of random hardware faults.

The model consists of a black-box containing some cryptographic secret [4]. The box interacts with the outside world by following a cryptographic protocol. The model supposes that from time to time the box is affected by a random hardware fault causing it to output incorrect values. For example, the hardware fault flips an internal register bit at some point during the computation. In [4] is shown that for many digital signatures and identification schemes these incorrect outputs completely expose the secrets stored in the box. At the beginning, this attack was considered to be applicable only for public key cryptosystems and not for secret-key algorithms [5].

Biham and Shamir propose a related attack called *Differential Fault Analysis (DFA)* in [5]. They show in [5] that DFA is applicable to almost any secret key

Rodica Tirtea is with Computer Science Department, University of Oradea, Romania (e-mail: rtirtea@uoradea.ro).

Mircea Vladutiu is with Computer Science Department, "Politehnica" University of Timisoara, Romania (e-mail: mvlad@cs.upt.ro).

Geert Deconinck is with ESAT/ELECTA Department, K.U.Leuven, Leuven, Belgium (email: gdec@esat.kuleuven.be).

cryptosystem proposed in the open literature. DFA attacks rely on the information gathered from the comparison of the encryption/decryption results from a correct operation and a faulty operation for the same encrypted/decrypted message. In their attack they assumed that the injected faults could affect bit(s) in encryption/decryption rounds or/and in key scheduling algorithm. They present this attack both for *Data Encryption Standard (DES)* and for any unknown block algorithm.

As the transient fault model for DFA [1] was claimed to be unrealistic, Biham and Shamir decided to develop a more practical fault model based on permanent hardware faults. They showed that their model could be used to break DES [5]. A smartcard implementation of DES was used to describe the attack. Two assumptions were analyzed. For the first assumption, was analyzed the case in which DES is implemented in hardware as a single round which is used 16 times for the 16 round of the algorithm. The other assumption uses unrolled implementation, when all rounds are using 16 separate hardware area. For the second assumption, the attacker is able to get easier bits of the subkeys. They called this *Non-Differential Fault Analysis (N DFA)* [5]. For this attack they proposed the cut of a wire or permanent destroy of a single memory cell.

B. Error detection techniques proposed to overcome fault analysis attacks

Different methods have been proposed to overcome the fault analysis attack. Most of these are focusing the probabilistic attacks (where the attacker has little or no control for the injected fault location and type). These methods include redundancy-based error detection schemes inspired from already existing fault tolerance techniques. Time-redundancy based concurrent error detection assumes two (several) encryption (or decryption), one after the other, and comparison of the result.

Such methods, in which encryption is done several times and the results are compared (e.g. in case of fault-tolerant smartcard design), are not always suitable [5]. For some fault models, key information can be retrieved based on the comparison, or, if the plaintext register is damaged, a faulty plaintext will be encrypted every time, and the fault will pass undetected.

Another type of redundancy-based technique, where the encrypted message is decrypted and the input value is compared, was proposed by Karri et al. in [6]. This method, called *Concurrent Error Detection (CED)* was applied at algorithm and round level for five cryptographic algorithms. The method can be used for symmetric encryption due to the inverse relationship that exists between encryption and decryption at algorithm level, round level and operation level. A trade-off analysis is presented regarding area overhead, performance penalty and fault detection latency for four cases – no fault tolerance (no concurrent error detection schemes),

concurrent error detection schemes at algorithm, round and operation level. Based on their implementations, round level concurrent error detection has the most convenient balance of area overhead (between 19% and 31% depending on the algorithm), performance penalty (performance degradation 11% to 26%) and fault detection latency (4 up to 8 cycles) [6]. It can be noticed that even if theoretically such concurrent error detection determines 100% hardware or/and time overhead, thanks to optimization and re-use of combinational logic, the final costs are reduced.

Bertoni et al., in [7], described and evaluated for the specific case of AES [8] a scheme based on error detection codes (i.e. parity codes). The method described in the paper proposes the use of a parity bit with each byte element. It requires prediction of the parity bits from one round to the other, being in this way algorithm dependent. This approach has a lower cost (limited hardware overhead, estimated between 10% and 20% [7], and short detection latency) compared with concurrent error detection schemes from [6]. Its fault coverage for single bit faults is high; however in case of multiple bit faults the coverage is lower. This method takes advantage of the specific structure of the AES algorithm.

Karpovsky et al., in [9], used symmetric nonlinear (cubic) error detection codes for a fault analysis attack resistant AES implementation. Their method however requires 75% hardware overhead.

From the point of view of fault coverage, the redundancy-based techniques are more efficient and easier to generalize even if the theoretical cost appears to be higher. Other techniques, using error detection codes, have to be specifically designed for the given cryptographic algorithm. However all techniques assume considerable overhead if fault-tolerant implementations are used for cryptographic algorithms.

III. USING COMPLEMENTED DUPLICATION FOR ERROR DETECTION

We propose in this paper another error detection technique, relying on complemented duplication, similar with concurrent error detection method proposed in [6]. However there are major differences. We call this method *complemented duplication error detection (CD)*.

The CD error detection method relying on the duplication with complementary logic technique described in [10]. For complementary logic technique one module is designed using positive logic while the other module uses negative logic (i.e. positive logic implies that the higher of the two voltages used in a logic circuit represents a logic ‘1’ while the lower represents ‘0’; negative logic uses the higher of the two voltages to represent a logic ‘0’ while the lower represents ‘1’) [10].

Our method does not assume using of positive logic and

negative logic, but only the idea of using a second component with complemented inputs being “processed” to generate the complemented outputs of the first component.

This technique is also similar with so called operation level concurrent error detection method proposed in [6], as for each operation there is a ‘complemented’ operation executed concurrently. However there are major differences – the inputs are different (complemented) and there is no requirement for storage of intermediate values.

Complemented duplication error detection relies on information redundancy and assumes for instance concatenation of each block of data, key, or intermediate value with its complement. New operations are applied to the complements, such that, second part of the block to be always the complement of the first. This method is algorithm dependent (i.e. specific technique) because for each operator $op1$ applied between A and B blocks of data, so that

$$A \text{ op1 } B = C$$

another operator $op2$ should be applied to the complement blocks $notA$ and $notB$ so that

$$notA \text{ op2 } notB = notC,$$

where $notX$ is the complement of X (see Fig. 1).

For instance, if $op1$ is the operator OR, for complements will be applied operator AND. Also, if $op1$ is the operator AND, for complements will be applied operator $op2$, in this case OR.

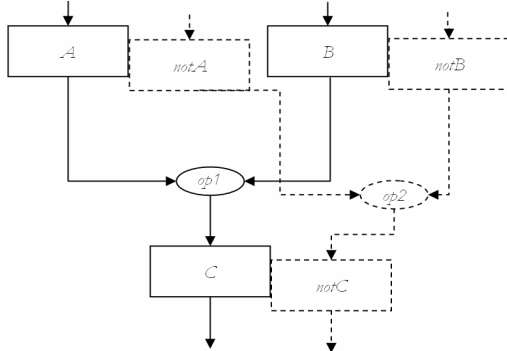


Fig. 1. Using complemented error detection. General Representation.

If the entire algorithm is described using the method presented in Fig. 1, then at the end of the algorithm implementation, after the last round, a self-testing comparator is required to check if the results are complements. This method, using self-testing comparator is able to detect all single faults.

Given the description of this method, we notice that both flows (using direct values and the complements) can be computed in parallel, and this method does not require time overhead. Also, detection latency is only given by the time required to compare the results of the two flows.

The CD error detection method can be described using the Boolean algebra and dual functions.

Definition. Let V be a vector of n bits given by $V=(v_{n-1}, \dots, v_1, v_0)$. For every combinational Boolean function $f(V)$

there exists a dual function $f_d(V)$ defined by $f_d(V')=f'(V)$, where V' is the complement of V and f' is function f complemented.

Example. Dual functions can be obtained by replacing AND operations with OR operations, OR operations with AND operations, bits of value ‘1’ with ‘0’, and bits of value ‘0’ with ‘1’.

The characteristics and disadvantages of the original duplication with complementary logic technique [10] do not apply directly to CD error detection for FPGA implementation. The different physical implementations for the modules (using one positive and the other negative logic) reduce the probabilities of same design or masking problems.

One of the disadvantages of the duplication with complementary logic was the change of topology given by the hardware limitations of replacing e.g. large number of input NOR gates with the same large number of input NAND gates [10].

However, this is not a problem for FPGA boards, and for CD error detection method, as all the functions including the gates functionalities are implemented using look-up-tables (LUT’s).

Another disadvantage of the duplication with complementary logic mentioned in [10] is due to the difficulty to convert a complex function in the complementary logic. For CD error detection in case of FPGA implementation this depends on the complexity of the implemented function as well.

In the following sections, after introducing the operations used in MISTY1 cryptographic algorithm, the cost of protected implementation using complemented error detection is compared with the costs of unprotected implementation of the algorithm.

IV. SHORT PRESENTATION OF MISTY1

We chose MISTY1 encryption algorithm to exemplify and apply our specific error detection mechanism. MISTY1 is a 64-bit block cipher with 128-bit key, evaluated and selected in two evaluation initiatives in Europe and Japan: in NESSIE [2] and CRYPTREC [3]. This is one of the reasons for choosing MISTY1. Another argument is given by its design which allows a low cost hardware implementation (MISTY1 is well suited for hardware implementation). Also, the *3rd Generation Partnership Project (3GPP)* adopted a variant of MISTY1, which is called KASUMI, as a mandatory algorithm in GSM confidentiality and integrity mechanisms [11].

Data randomizing part of MISTY1 uses for all rounds (8 recommended) a function FO (Fig 2), and for all odd rounds, and after the last one, FL functions (Fig. 3). FO function includes FI functions. FL uses bitwise AND, OR and XOR operations. FO function relies on bitwise XOR operation and sub-functions FI. FI uses XOR bitwise

operation and two substitution boxes S7 and S9. For implementation purposes S7 and S9 can be defined as tables or implemented using logic expressions using bitwise XOR and AND. The operations and sub-functions used are summarized in Table I. For further details and description of MISTY1 please refer to [12].

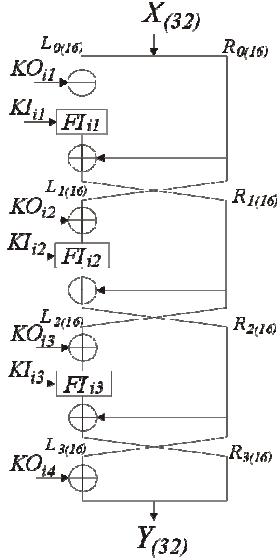


Fig. 2. FO function.

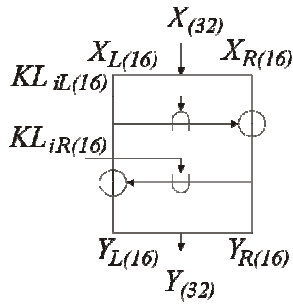


Fig. 3. FL function.

As can be seen in Table I, for all these operation another operation on the complemented inputs can be used. As such, for each function in MISTY1 a complement function can be defined.

Function	Operations and/or functions
FL	AND, OR, XOR
S7	AND, XOR
S9	AND, XOR
FI	XOR, S7, S9
FO	XOR, FI

V. ERROR PROPAGATION IN MISTY1 ALGORITHM

The propagation of the errors in case of MISTY1 algorithms depends on the propagation of errors in each round and subround, respectively, of each function which

describes the algorithm's round. We assume in this analysis that only a *single bit* may become faulty at any given time instant. In this section the propagation of *single bit fault* for round and subround functions of the algorithm is analyzed.

In this section we analyze the propagation of errors in the two main components (see Fig. 2 and 3) of the MISTY1 algorithm: the sub-round function FL and the round function FO. We assume that a single bit fault is affecting the input data. The purpose of this analysis is to identify the effect of a fault on the output of both components (i.e. if there is any dependency between the location of the fault and of the error(s), and if there is a 'avalanche' effect, as it should be in each round of an encryption algorithm).

FL function and error propagation. Based on the structure of the FL function, as can be seen in Fig. 3 we can draw some observations, which have been confirmed by simulation tests. Depending on the location of the injected fault, the output of the function FL is affected by two errors. If a fault is injected in the left half side of the input, then two errors are detected – one on the same position at the output, in the left half, of function FL and one in the same position but on the right half of the output of FL function. As well, if a fault is injected in the right half side of the input, then two errors are detected: one on the same position of the right half of the output of the function FL, and the second in the same position but in the left half of the output. During the simulation tests no fault injected passed undetected.

FO function and error propagation. In contrast with FL function, FO function does not show such dependency on location, neither on the numbers of errors generated. However, FO produces a 'avalanche' effect (the purpose of cryptographic algorithms is to 'hide' the input, as such even if there is minor difference in inputs the outputs should be different and no correlation easy to make).

Using an experimental environment (ModelSim XE III 6.0d for simulation of the VHDL code) we simulated the behavior of the function FO in case of a single bit injection. We injected the fault in different locations and we compared the results between execution without fault injected and execution with fault injected. The results of these simulations are presented in Fig. 4.

The results shown in Fig. 4 are based on about eight millions of pairs of test vectors (faulty and non faulty). On the horizontal axis the number of erroneous bits (out of 32 output bits) is indicated. On the vertical axis (up) the percentage of total tests generating the same number of fault is given (ex. 13.75%, meaning almost 1100 000 test vectors of the tests detected 14 bits affected by errors). 94% of the test vectors had between 10 and 20 bits erroneous, while 98% between 9 and 22 error bits.

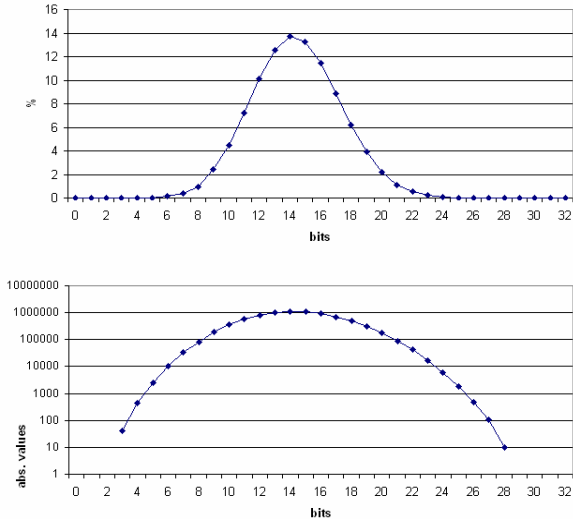


Fig. 4. FO error propagation. Percentage (up) and logarithmic distribution (down).

During the simulations no fault passed undetected, and no test vector generated less than 3 errors, or more than 29 errors. This can be noticed in Fig. 4 (down) where logarithmic representation is used for the values of vertical axis. During these simulations no correlation regarding the location of the bits affected by error has been noticed.

Analyzing the results of these simulations for error propagation some conclusions can be drawn. On one hand, using a parity bit error detection method applied for the entire output of FL (32 bits) function is not a reliable method – no injected errors can be detected. The use of a parity bit is suited only if it is applicable for each half (16 bits) or each byte and assuming a single faulty bit.

On the other hand for the FO function, given the ‘avalanche’ effect, no assumption can be made regarding the even or odd number of bits that are faulty at the output if a fault is injected. As such a parity bit method would not detect all faulty outputs.

VI. APPLYING COMPLEMENTED ERROR DETECTION FOR MISTY1 ALGORITHM

A. MISTY1 implementation in VHDL

For this implementation VHDL and Xilinx’s ISE 8.1i environment targeting Virtex FPGA (device xc1000, package bg560) was used. This device was selected in order to allow for comparison with [11]; [11] can be used as reference due to published implementation results for all components of MISTY1. We compare in Table II our implementation results with the ones published in [11].

If the results of our implementation are compared with the reference results, we can notice that for S boxes the results are disadvantageous. For S7 and S9 our implementation is less advantageous. However, functions FI and FO incorporate S7 and S9 and are compensating this disadvantage, as they have better simulation results. FL

function has the same number of 4 input look-up-tables used (32 of 4 LUT’s).

TABLE II.
IMPLEMENTATION RESULTS

Component	Implementation [11] # of 4 LUT’s	Implementation (this work) # of 4 LUT’s	Comparison
FL	32	32	0.00
S7	44	48	9.09
S9	44	53	20.45
FI	172	170	-1.16
FO	655	602	-8.09

Based on this comparison, we conclude that our implementation is comparable with the reference one and can be used for analyzing the cost of error detection.

B. MISTY1 implementation with CD error detection

We used for the implementation which includes the CD error detection the technique described in section III (for each function/operation the dual function/component has been added as in Fig. 1.). The results of our implementation (without and with complemented error detection mechanisms included) are listed in Table III.

We introduced in this table the number of slices as well. This is relevant in order to notice that not all resources reported as used are actually mapped i.e. not all slices have both their 4 LUT’s tables used. In case of FL function, 18 slices contain a number of 36 4 LUT’s, however only 32 are used – in this case 4 LUT’s with 4 inputs each are unused.

For the CD error detection mechanism a *self-checking checker* was deployed [13]. For our implementation the output of the checker uses a distance-two error detecting code *1-out-of-2*, where $\{(0,1), (1,0)\}$ are valid code words, first for correct output, second for ‘error’ output (see Fig. 1) and $\{(0,0), (1,1)\}$ are invalid code words. As such unidirectional errors are detected.

Regarding the cost of implementation, from Table III it can be noticed that the largest hardware overhead was generated in case of FI function i.e. 10%, for the number of LUT’s. In the same time, as every slice contains two 4-input LUT’s next to other logic, it can be noticed that even in case of complemented duplication not all 4 LUT’s from the FPGA slices are used. Time related values were not included due to small differences. For our simulations the maximum combinational path delay increased with 3.75% (for complemented duplication compared with the unprotected implementation), in case of FO function.

C. Analysis of overhead costs

Due to low overhead we analyze the FL function and the logic used to implement the component. We notice that the same logic can be used, in the same time, to implement the dual function (Fig. 5). The logic used for the steps P3 and P4 may be used simultaneous with the steps P1 and P2 to produce the output of the first half of FL dual function.

TABLE III.
ANALYSIS OF OVERHEAD FOR CD ERROR DETECTION

Component	# of 4 LUT's	# of slices	# of 4 LUT's (CD error detect.)	# of slices (CD error detect.)	area overhead (in # of 4 LUT's)	area overhead (in # of slices)
FL	32	18	32	18	0 %	0 %
S7	48	27	48	27	0 %	0 %
S9	53	30	55	31	3.77 %	3.33%
FI	170	97	187	106	10.00 %	9.28%
FO	602	345	651	371	8.14 %	7.54%

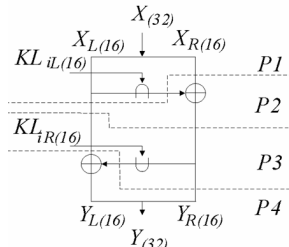


Fig. 5. FL function broken down in steps.

During the execution of P3 and P4, P1 and P2 can be used to execute the second half of the dual function of FL. Logic for inversion is required as well as multiplexer to set the flow. These logic control functionalities are easily implemented using the rest of the logic from the FPGA board where a CLB may be execute any of: any 6-input function, an 8:1 multiplexer, any selected functions of up to 19 inputs.

As such, no extra time and FPGA LUT's are required to execute FL and its dual function – as they can be executed in parallel using the same hardware (this validates the 0% hardware overhead result for FL function, in Table III). Further analysis could be carried out for the other functions, where the stages of these functions cannot be separated such as for the case of FL function.

However, the software used to map the VHDL description, on the logic of the FPGA board, performs optimization. As such the results of implementation/simulation on FPGA (where optimization can (re-)allocate the supplementary hardware to the unused resources of already counted slices/LUT's) cannot be compared with theoretical ones which are not considering optimization.

VII. CONCLUSIONS

In this paper we propose a new method for error detection in cryptographic implementations: complemented duplication error detection. This method is applied in the paper for MISTY1 cryptographic algorithm.

An error propagation analysis for two functions of MISTY1 algorithm is also presented.

The results of our implementations in VHDL of MISTY1 functions are similar with reference results. As such this implementation and its characteristics can be used as comparison reference for the implementation incorporating complemented duplication error detection. Based on the

comparison analysis we noticed that complemented duplication error detection techniques does not generate large overhead (i.e. hardware overhead is up to 10%) and has reduced detection latency. We analyze and justify these results due to MISTY1 architecture.

Further research will consider analysis of overhead cost of this technique for other algorithms.

REFERENCES

- [1] D. Boneh, R. DeMillo, R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", Eurocrypt '97, LNCS 1233, Springer-Verlag, pp.37-51, 1997.
- [2] NESSIE consortium, "Portfolio of recommended cryptographic primitives", February 27, 2003 [Online]. Available: <https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/decision-final.pdf>.
- [3] CRYPTREC, Evaluation of Cryptographic Techniques site [Online]. Available at: <http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html>.
- [4] D. Boneh, R. DeMillo, and R. Lipton, "On the importance of checking cryptographic protocols for faults", *Journal of Cryptology*, Springer-Verlag, Vol. 14, No. 2, pp. 101-119, 2001.
- [5] E. Biham, A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems", Proceedings of Crypto'97, 1997 [Online]. Available: www.cs.technion.ac.il/~biham/publications.html.
- [6] R. Karri, K. Wu, P. Mishra, et al., "Concurrent Error Detection of Fault Based Side-Channel Cryptanalysis of 128-Bit Symmetric Block Ciphers", *IEEE Trans. on COMPUTER-AIDED DESIGN of Integrated Circuits and Systems*, Vol. 21, No. 12, December 2002, pp. 1509-1517.
- [7] G. Bertoni, L. Breveglieri, I. Koren, et al., "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard", *IEEE Trans. on Computers*, Vol. 52, No. 4, April 2003, pp. 492-505.
- [8] J. Daemen, V. Rijmen, "The Rijndael block cipher, AES Proposal: Rijndael" [Online]. Available: <http://csrc.nist.gov/CryptoToolkit/aes/rijndael>.
- [9] M. Karpovsky, K. J. Kulikowski, A. Taubin, "Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard", Proc. Of Int. Conf. on Dependable Systems and Networks (DSN'04), Florence, Italy, 2004, pp. 93-102.
- [10] Barry W. Johnson, *Design & analysis of fault tolerant digital systems*, Addison-Wesley Longman Publishing Co., Inc., 1989.
- [11] Rouvroy G, Standaert F-X, Quisquater J-J, Legat J-D. "Efficient FPGA Implementation of Block Cipher MISTY1", Proc. of the 10th Reconfigurable Architectures Workshop (RAW), Nice, France, 2003, 7 pages.
- [12] M. Matsui, Supporting Document of MISTY1 [Online]. Available: <https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions.htm>.
- [13] T. R. N. Rao, E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice-Hall, 1989.