Compact ASIC Implementation of the ICEBERG Block Cipher with Concurrent Error Detection

Huiju Cheng and Howard M. Heys
Electrical and Computer Engineering
Memorial University of Newfoundland
St. John's, Canada

***Abstract* — ICEBERG is a block cipher that has been recently proposed for security applications requiring efficient FPGA implementations. In this paper, we investigate a compact ASIC implementation of ICEBERG and consider the novel application of concurrent error detection to protect the implementation from fault-based attacks. The compact architecture of ICEBERG requires about 5800 gates with a throughput of 552 Mbps in an ASIC implementation based on 0.18 μm CMOS technology. The addition of an effective multiple parity concurrent error detection scheme to protect the hardware from fault attacks results in a 62% area overhead.**

I. Introduction

A compact hardware implementation of a block cipher is attractive for low-cost embedded applications. Unfortunately, the Advanced Encryption Standard (AES) [1] is not optimal for compact hardware implementation due to the large eight-bit substitution boxes and differences between the encryption and decryption circuits.

ICEBERG [2][3] has been recently proposed as a secure, efficient block cipher targeted to FPGA implementation. It is a 64-bit block cipher with a 128-bit key. The substitution layer of ICEBERG is based on small 4-bit S-boxes resulting in a reduced hardware complexity for the encryption and decryption processes. ICEBERG can be implemented in different architectures such as loop and pipelined designs.

In this paper, we investigate the compact ASIC implementation of ICEBERG. Further, in order to minimize the impact of fault-based attacks, we investigate the design of a multiple parity based concurrent error detection scheme for the compact implementation.

II. ICEBERG Algorithm

ICEBERG is an iterative involutional block cipher (that is, each round operation is an involution) with 16 round functions for the encryption and decryption processes. Each round function is composed of a non-linear layer, γ, and a linear layer, ε_K, including key addition. The nonlinear layer is composed of successive application of S-boxes and bit permutations.

Two types of 4×4 S-boxes are applied in ICEBERG with the substitution layers $S0$ and $S1$ separately composed of 4×4 S-boxes which are used to perform the substitution for the 64-bit input data. The bit permutation layer $P8$ consists of eight parallel permutations on 8-bit blocks of data. The non-linear layer can be represented as:

$$\gamma : \{0,1\}^{64} \rightarrow \{0,1\}^{64} : \gamma \equiv S_0 \circ P8 \circ S_1 \circ P8 \circ S_0$$

In software implementations, the γ function can be easily replaced by eight identical 8×8 S-boxes.

The linear layer, ε_K, includes the consecutive application of a linear diffusion layer, bit permutations and a linear key addition layer σ_K which is composed of a bitwise XOR between the 64-bit data block and 64 bits of round key. The diffusion layer, D, is constructed with a simple involutional matrix multiplication. Two types of bit permutation are applied in ε_K: $P64$ and $P4$. The $P64$ layer performs a 64-bit permutation and it guarantees that two bits from the same byte are always mapped to two different bytes. The $P4$ layer applies in parallel sixteen 4-bit permutations. Hence, the linear layer function can be represented as follows:

$$\varepsilon_K : \{0,1\}^{64} \rightarrow \{0,1\}^{64} : \varepsilon_K \equiv P64 \circ P4 \circ \sigma_K \circ D \circ P64$$

The round function ρ_K can be represented as:

$$\rho_K : \{0,1\}^{64} \rightarrow \{0,1\}^{64} : \rho_K \equiv \varepsilon_K \circ \gamma$$

The key schedule of ICEBERG consists of key expansion and key selection. Each 128-bit round key is expanded based on the previous round key with the application of non-linear substitution boxes, shift operations and bit permutations. After the key expansion, the key selection function will be applied to the 128-bit round key to select the 64 bits to be mixed with the data.

Since ICEBERG is an involutional cipher, the decryption process is performed in the same way as encryption even without using the round keys in a reverse order as long as the constant values applied in the key schedule are properly chosen. As a result, the encryption and decryption processes can share the same hardware.

III. Previous Implementations

The ICEBERG block cipher is designed for efficient FPGA implementations. Three architectures of ICEBERG implemented in FPGA have been proposed in [3] for different optimization purposes. A fully pipelined unrolled architecture and a half pipelined architecture are designed for high-speed hardware implementation and a loop architecture is designed for compact hardware implementation.

Based on the FPGA implementation results for a Xilinx Virtex-II device, the fully pipelined unrolled architecture of ICEBERG can achieve a maximum throughput of about 19 Gbps, while the loop architecture requires minimum hardware resources of about 631 slices with a throughput of about 1 Gbps. However, with the consideration of the ratio of throughput/area, the half pipelined architecture achieves the highest hardware efficiency.

Support for this research was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) and CMC Microsystems.

IV. COMPACT ASIC IMPLEMENTATION

Although ICEBERG was proposed for efficient FPGA implementations, its ASIC implementations are also very efficient. We have explored a compact architecture of ICEBERG and implemented it using a 0.18 µm CMOS standard cell library based on the TSMC 1P6M process. Synopsys Design Analyzer (version 2001.08) is applied as the synthesis tool.

A. Compact Architecture of ICEBERG

Our compact design utilizes a loop architecture based on one round function of ICEBERG and is illustrated in Figure 1. Only one register is inserted in the round function. The major components of the encryption process include the *S*0 substitution layer, *S*1 substitution layer, diffusion layer, key addition layer and permutation layers. The *S*0 layer is composed of sixteen 4×4 *S*0-boxes which are used for substitution of the 64-bit input data in parallel. The *S*1 layer is constructed similarly to *S*0 with sixteen 4×4 *S*1-boxes. The diffusion layer performs a multiplication with a binary involutional matrix and is implemented in linear XORs on sixteen 4-bit data blocks. The key addition layer is implemented in XORs between the input data and the round keys. The permutation layers, *P*8, *P*64, and *P*4, are simply implemented as wirings and do not require any logic gates. Since ICEBERG is a 64-bit involutional block cipher, the decryption process can share exactly the same hardware with the encryption process.

The round function of the key schedule is composed of key expansion and key selection. The major components consist of the left or right shift layer, the *S*0 layer, the permutation layer *P*128 and the Boolean operations in key selection. The left or right shift layer can be simplified as just wirings of the input data. The *S*0 layer in the key schedule is implemented in the same way as in the encryption process. The 128-bit permutation layer *P*128 is correspondingly implemented as wirings. Since no part of the datapath is shared between the encryption process and key schedule, the round keys used for each round function can be generated on the fly and no storage is needed to store all the round keys.

B. Hardware Performance Analysis

Based on the compact architecture of ICEBERG, a 0.18 µm CMOS standard cell library is applied in our hardware performance analysis. During the synthesis, we focus on area optimization. The area of the circuit is evaluated in terms of equivalent 2-input NAND gate counts.

Table 1 shows the hardware complexity analysis of each component of ICEBERG. The whole datapath implemented in ASIC requires about 5.8k gates with a maximum frequency of 138 MHz. The encryption process needs sixteen clock cycles and since 64 bits of data can be processed in parallel, the throughput of the implementation can reach 552 Mbps. As shown in Table 2, these results compare favourably to compact implementations of other block ciphers, especially when considering the tradeoffs between compactness and throughput. For example, although the AES implementation in [4] is more compact, this comes at a great expense of speed resulting in a throughput to area ratio of more than 30 times less than the implementation of ICEBERG investigated in this paper.

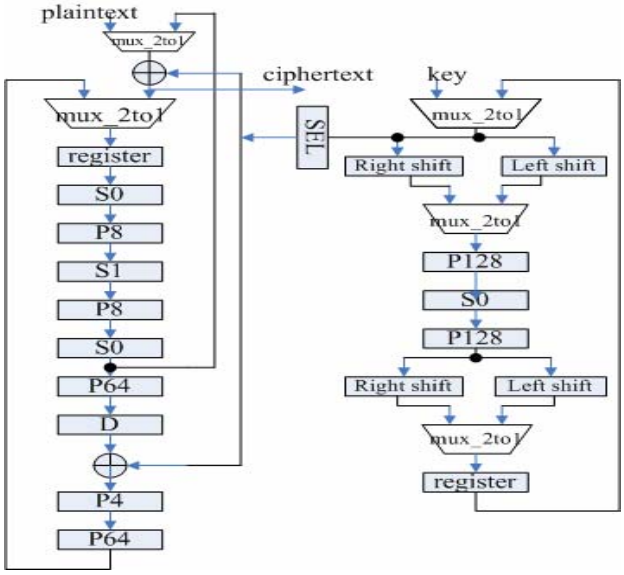

Figure 1 Compact Architecture of ICEBERG

Table 1. Hardware Complexity Analysis of Compact ICEBERG

Component	Area (gates)	Percentage
S-boxes	2400	41.2%
Registers	1536	26.4%
Multiplexers	1024	17.6%
XOR & Boolean Logic	857	14.7%
Datapath	5817	100%

Table 2. Comparison of Compact Cipher Implementations

Cipher Implementation	Area (gates)	Throughput (Mbps)	Throughput /Area (Mbps/kgates)
ICEBERG	5800	552	95.2
AES [4]	3400	9.9	2.9
AES [5]	6700	112	16.7
Camellia [6]	14100	143	10.1

V. CONCURRENT ERROR DETECTION DESIGN

In a hardware implementation of a block cipher, transient or permanent faults can be intentionally induced to mount a fault-based cryptanalysis [7]. Such methods exploit the secret information leaked by the erroneous behaviour caused by injected faults. It is known, for example, that implementations of AES can be susceptible to fault attacks based on various fault models, such as stuck-at faults [8]. Concurrent error detection (CED) is a technique to detect any transient or permanent faults that occur in the system and, by suppressing the resulting faulted outputs, thereby mitigate the susceptibility of the system to attack.

Numerous approaches to CED for ciphers have been proposed. Approaches based on hardware redundancy [9] compare the output of cipher operations from the datapath to an alternate datapath performing the same computations. Such an approach is effective in detecting a broad range of permanent or transient faults, but costs greater than 100% of the area of the original circuit. The use of CED based on error control coding schemes is a very effective mechanism for

detecting both permanent and transient errors, with minimal area overhead [10][11][12][13].

In this paper, we investigate a multiple parity based CED scheme for ICEBERG with the parity generated for each byte of the input 64-bit data block. For the operations of the cipher algorithm, the parities of the output are predicted from the input data and then are compared with the parities of the actual output to detect any mismatch caused by faults.

A. Multiple Parity Based CED Scheme

In our parity based CED scheme, we have applied parity to both linear and nonlinear components. For the nonlinear substitution in the design of the datapath, the 4×4 S-boxes are extended to 4×5 S-boxes by attaching a parity bit generated from the XOR of both input parity and output parity. Two consecutive parities attached in the S-boxes are XORed with each other to predict the output parity because all the parities in our CED scheme are byte based. The predicted output parities are then compared with the parities of the actual S-box outputs. For the CED circuit designed for linear components such as multiplexers and registers, the byte-based input parities directly pass through duplicated smaller multiplexers and registers without any modification to predict the parities of the actual output.

The most complex part of the CED scheme for the encryption process is the protection of the three layers of 4×4 S-boxes. Although two layers of the byte-based permutation *P*8 are inserted between the S-box layers, the parities remain the same after those permutations. Therefore, no parity prediction circuit is necessary for *P*8. For the S-box, one extra parity bit is stored as the fifth bit of each S-box and the parity is generated by the XOR of both input parity and output parity. Figure 2 illustrates the three processing steps of the parity modification for the S-boxes. The permutation layers between the substitution layers are not shown in this figure since they will not change the byte-based parities.

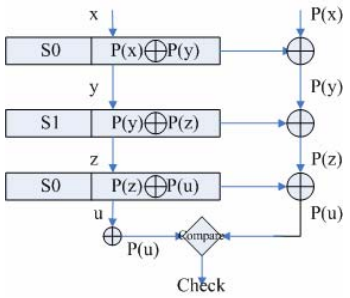

Figure 2. CED for S-boxes

In the above figure, the notation P(·) represents the parity of one byte of data and "⊕" represents the XOR operation. We can see that in the CED circuit for the three layers of S-boxes, the parities are predicted by the XORs of input parities and attached parities stored in the S-boxes. Accordingly, the parities of the actual output are equal to the modified parities of the input if no error occurs in the S-boxes. All the faults that lead to single-bit errors and most multiple fault scenarios would be detected under such a parity based CED scheme designed for these three layers of S-boxes.

For the CED scheme of the encryption process, three check points are inserted. The first check point is inserted for the parity checking between the parities of the actual output of the first 64-bit XOR layer and the predicted parities of the input data of the first 2-to-1 multiplexer. Consequently, any fault that occurs in the first multiplexer and first XOR layer can be detected by the first check point. The second check point is inserted after the three layers of S-boxes to detect any fault that occurs in the S-box hardware. The third check point is inserted after the successive components of diffusion and second XOR layer for key addition. Due to the property of diffusion, the input parities will not change after the diffusion and, hence, no parity modification is needed for that component. In addition, the CED circuit for the second XOR layer is the same as the first one.

In the key schedule, three more check points are inserted in the corresponding CED circuit. The fourth check point is used to detect faults that occur in the multiplexers and registers. The fifth check point is inserted after the nonlinear layer of *S*0. The generation of the parity bit attached as the fifth bit in the *S*0 layer of the key schedule is different from that used in the encryption process. These S-boxes are extended from 4×4 S-boxes to 4×5 S-boxes with a fifth output bit being the parity of just the four output bits of the S-box. Therefore, the fifth check point is used to compare the parities of the actual output of the S-boxes and the predicted output parities stored in the fifth bit of the S-boxes. The sixth check point is inserted after the key selection component. Because Boolean operations like AND and OR are included in this component, the parity based CED technique is not suitable to be applied because the parities are not easy to predict after the key selection. Therefore, the hardware redundancy technique is a better choice and the whole key selection component is duplicated. The output of the duplicated hardware is compared with that of the original hardware to detect any fault that occurs in the key selection.

B. Hardware Performance of CED Scheme

Based on the compact architecture of ICEBERG with parity based CED, we evaluated the hardware performance in ASIC using a 0.18 µm CMOS standard cell library. The hardware overhead caused by our CED scheme is listed in Table 3. From the table, we can see that the area of our compact architecture of ICEBERG increases to 9303 gates after applying the CED scheme resulting in a hardware area overhead of 62%. The corresponding throughput degradation is about 7%. These results should be put in context by considering that a CED scheme based on hardware redundancy would require more that 100% area overhead. Similar studies for AES [5] and Camellia [6] found that compact implementations with CED resulted in 10.9k gates for AES and 26k gates for Camellia.

Table 3. Overhead for CED Scheme

	No CED	Multiple Parity CED	Overhead
Area (gates)	5817	9303	62%
Throughput (Mbps)	552	512	7%

C. Error Detection Capability

The basic components of our compact architecture of ICEBERG are the XORs, S-boxes, registers, and multiplexers. For the linear components, such as XORs, registers and multiplexers, a single stuck-at fault would only

result in one bit error which can be easily detected by the parities. For the nonlinear S-boxes, depending on the implementation of the logic, a single stuck-at fault may result in an even number of errors at the output which can not be detected by parities. For example, if all the outputs of the 4×5 S-boxes, which include one attached parity bit, are jointly optimized, not all the possible single faults will be detected since the sharing of gates might result in a single fault causing two errors at the S-box output. To avoid this, in our implementation of S-boxes, each output bit of the S-box is independently implemented in combinational logic. As a result, a single fault injected inside the S-box will only lead to one erroneous output bit which can be detected by the parities. Hence, under the protection of our multiple parity based CED scheme with independently implemented S-box bits, all the single faults in the system can be detected.

As well, using our CED approach, most multiple faults in the linear components of our implementation will be detected since they will result in at least one byte including an odd number of errors. In order to investigate the effect of multiple faults within the S-box layers, we have designed a simulation experiment to evaluate the fault coverage of the CED scheme. The 64-bit output data of the S-box layer and the attached byte-based parities used for parity prediction are separately implemented. Under the assumption that each S-box bit costs ten gates in an ASIC implementation, we randomly injected multiple stuck-at-1 faults into the 640 gates and determined whether the multiple injected faults could be detected by the parities. We assumed that no matter how many faults are injected in the ten gates for the same bit, one error will be caused at the S-box output. One million tests have been simulated for the number of multiple faults from 2 to 16. The resulting fraction of fault cases that are not detected is presented in Figure 3. For the double faults, we find that the percentage of undetected faults is about 10.98%, but this is not shown in Figure 3 in order to have the graph reasonably scaled for other multiple faults. To confirm the experimental results, we have also computed, using combinatorics, the theoretical expectation for the fraction of faults that are not detected for the various scenarios. For example, for two faults, we determine the probability of undetected fault scenarios by calculating the number (22400) of scenarios resulting in both faults being in the two S-boxes covered by the same parity and dividing by the total number (204480) of possible locations for the two faults. This results in a 10.9% probability that two faults will be undetected.

As expected, the experimental results and the theoretical results are very close. From Figure 3, we can see that our multiple parity based CED applied to the S-boxes has a very high fault coverage for multiple fault scenarios. Hence, it is fair to conclude that all single faults and most multiple faults would be detected.

VI. Conclusion

In this paper, a loop architecture of the involutional block cipher ICEBERG has been investigated and evaluated by using a 0.18 μm CMOS standard cell library. To protect our ICEBERG hardware from faults intentionally induced for cryptographic attacks, we have investigated a multiple parity based CED scheme which provides high fault coverage with low hardware cost for our compact architecture of ICEBERG. We conclude that compact ASIC implementations of ICEBERG provide high throughput for small area and can be effectively protected using a parity-based CED scheme.

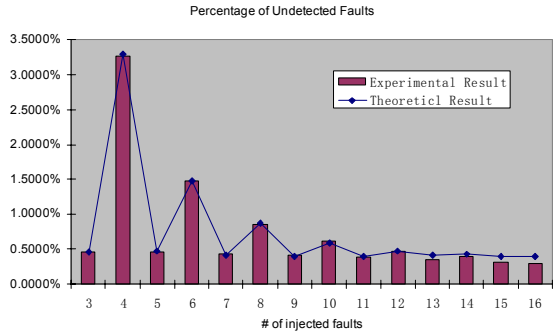

Figure 3. Percentage of Undetected Faults

References

[1] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)", FIPS Publication 197, Nov. 2001.

[2] F. Standaert, G. Piret, G. Rouvroy, J. Quisquater, and J. Legat, "ICEBERG: an Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware", *Fast Software Encryption (FSE 2004), Lecture Notes in Computer Science*, Vol. 3017, pp. 279-299, 2004.

[3] F.-X. Standaert, G. Piret, G. Rouvroy, and J.-J. Quisquater, "FPGA Implementations of the ICEBERG block cipher", *INTEGRATION: The VLSI Journal*, vol. 40, no. 1 , pp. 20-27, 2007.

[4] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand", *IEE Proceedings on Information Security*, vol. 152, no. 1, pp. 13-20, 2005.

[5] N. Yu and H.M. Heys, "A Hybrid Approach to Concurrent Error Detection for a Compact ASIC Implementation of the Advanced Encryption Standard", *IASTED International Conference on Circuits, Signal, and Systems* (*CSS 2007*), Banff, Alberta, Canada, July 2007.

[6] H. Cheng and H.M. Heys, "Compact Hardware Implementation of the Block Cipher Camellia with Concurrent Error Detection", *IEEE Canadian Conference on Electrical and Computer Engineering* (*CCECE 2007*), Vancouver, British Columbia, Canada, April 2007.

[7] D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", *Advances in Cryptology – EUROCRYPT* '97, *Lecture Notes in Computer Science*, vol. 1233, Springer, pp. 37-51, 1997.

[8] J. Blomer and J. Seifert, "Fault Based Cryptanalysis of Advanced Encryption Standard (AES)", *Financial Cryptography (FC 2003), Lecture Notes in Comp. Sci.*, vol. 2742, Springer, pp. 162-181, 2003.

[9] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Fault-based Side-channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'01)*, 2001.

[10] K. Wu, R.Karri, G. Kouznetzov and M.Goessel, "Low Cost Concurrent Error Detection for the Advanced Encryption Standard," *International Test Conference 2004 (ITC 2004)*, pp. 1242-1248, 2004.

[11] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," *IEEE Transaction on Computers*, vol. 52, no.4, pp. 492-505, April 2003.

[12] M. Karpovsky, K. Kulikowski, and A. Taubin, "Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard," *International Conference on Dependable System and Networks (DSN '04)*, 2004.

[13] C-H. Yen and B-F Wu, "Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard", *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 720-731, 2006.