# Fault Attacks Resistant Architecture for KECCAK Hash Function

Fatma Kahri, Hassen Mestiri, Belgacem Bouallegue, Mohsen Machhout
Electronics and Micro-Electronics Laboratory (E.µ.E.L), Faculty of Sciences of Monastir,
University of Monastir, Tunisia

***Abstract*—The KECCAK cryptographic algorithms widely used in embedded circuits to ensure a high level of security to any systems which require hashing as the integrity checking and random number generation. One of the most efficient cryptanalysis techniques against KECCAK implementation is the fault injection attacks. Until now, only a few fault detection schemes for KECCAK have been presented. In this paper, in order to provide a high level of security against fault attacks, an efficient error detection scheme based on scrambling technique has been proposed. To evaluate the robust of the proposed detection scheme against faults attacks, we perform fault injection simulations and we show that the fault coverage is about 99,996%. We have described the proposed detection scheme and through the Field-Programmable Gate Array analysis, results show that the proposed scheme can be easily implemented with low complexity and can efficiently protect KECCAK against fault attacks. Moreover, the Field-Programmable Gate Array implementation results show that the proposed KECCAK fault detection scheme realises a compromise between implementation cost and KECCAK robustness against fault attacks.**

***Keywords*—Cryptographic; KECCAK SHA-3; Fault detection; Embedded systems; FPGA implementation***

## I. Introduction

In August 2015, the cryptographic hash algorithm SHA-3 was finalised by the National Institute of Standard and Technology (NIST), when the KECCAK algorithm was adopted. Currently, the KECCAK algorithm replaced the Secure Hash Algorithm (SHA-2) which has been in use since 2009 [1-2].

Currently, various hardware implementation architectures and optimisations of KECCAK algorithm have been proposed for different applications and their performances have been evaluated by using ASIC and FPGA [3-7].

Improving the performance of the KECCAK circuits is a critical problem when the circuits are used in embedded systems. Cryptographic algorithm KECCAK is currently used in a very large variety of scenarios as the financial transactions, which has high security requirements. Moreover, the necessity to secure the KECCAK algorithm against various attacks as fault injection attacks [8-9]

KECCAK hash function is used for data integrity in conjunction with digital signature schemes. Also, for several reasons a message is typically hashed first. Then, the hash-value, as a representative of the message, is signed in place of the original message [10-11].

Yet, the malicious injected and the natural faults decrease the KECCAK robustness in may cause secure data leakage in non-secure implementation. The injected faults are caused by ambient environment, power consumption, computation time or electromagnetic radiation; the cryptographic systems are sensitive to these errors. We noted that the random errors are presenting false results which make these systems unreliable. Also we can inject faults temporarily in the cryptographic system in reason to retrieve the secret key or state. Many error detection schemes have been implemented to make a robust hardware design and to secure cryptographic systems against faults injection attacks [12-21].

In [12] Bayat-Sarmadi et al. proposed a new fault detection scheme for the KECCAK hash function. This is based on rotated by a random number before each round operation, and shifted back after KECCAK operations without changing the results. Then, they implement another copy of the hardware KECCAK algorithm to perform a comparison between the two copies results. Moreover, they perform fault attacks simulations and they show that the detection capability of close to 100% is derived.

Luo et al. presented in [20] a new detection scheme based on parity checking in reason to protect the operations KECCAK. This scheme consists of comparing the parity inputs with the parity outputs of each operation. The simulation security results show that the scheme leads to high security against fault attacks.

In this paper, we proposed a new fault detection scheme for obtaining an efficient KECCAK implementation with a high level of security against faults attacks. This scheme based on the scrambling technique to secure KECCAK algorithm.

The paper is organised as: Section 2 describes the background knowledge. In Section 3 we present the KECCAK design. Section 4 presents the KECCAK fault detection scheme. Section 5 deals with the detection capability evaluation of the proposed architecture. In Section 6, the FPGA implementation results and performances are discussed and compared. Finally, in Section 7, we conclude the paper.

## II. Preliminaries

### A. *Algorithm KECCAK*

The KECCAC algorithm is based on the sponge construction. The KECCAK hash function is the permutation f. This is applied to a fixed length state of b, with $b = r + c$; c is a capacity, r is a bit rate. The higher security and speed level correspond to higher values of c and r respectively. The hash procedure is as follow: first, to get a fixed size message, the input message is padded. Then, five internals steps are applied for each round. Finally, the squeezing phase occurs. The sponge function is composed of two phases: Absorbing and squeezing phases. Figure 1 shows the Sponge Function.

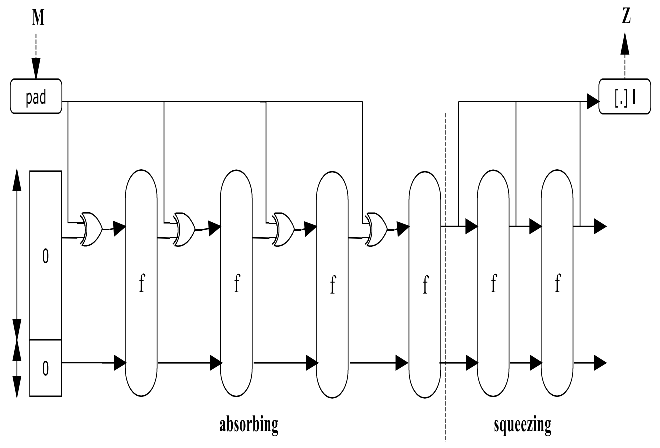

Fig. 1. Sponge Function

The state is composed of an array of 5×5 lanes. w is a length of lane, when w ∈{1, 2, 4, 8, 16, 32, 64}, and (b = 25w). The sponge construction is applied to KECCAK-f, so we applied the padding to the message input for obtaining the KECCAK-f [r,c]. With c is capacity and r is bitrate. All the operations on the indices are done modulo 5. A signify the complete permutation state array, and A[x,y] show a particular lane in that state. The intermediate variables are B[x,y], C[x] and D[x]. RC[i] present the round constants. While the constants R[x,y] are the rotation offsets. The binary cyclic shift operation is indicated by Rot (w,r). The bit is shifted by position i to position i + r (modulo the lane size). The constants R[x,y] are the cyclic shift offsets and are specified in Table 1.

TABLE I. CONSTANTS R[X,Y] OF KECCAK ALGORITHM

| | X=3 | X=4 | X=0 | X=1 | X=2 |
|---|---|---|---|---|---|
| **Y=2** | 25 | 39 | 3 | 10 | 43 |
| **Y=1** | 55 | 20 | 36 | 44 | 6 |
| **Y=0** | 28 | 27 | 0 | 1 | 62 |
| **Y=4** | 56 | 14 | 18 | 2 | 61 |
| **Y=3** | 21 | 8 | 41 | 45 | 15 |

Table 2 shows the constants rounds RC[i]. These values are specified in hexadecimal notation for lane size 64. The hash function KECCAK-f consists of 24 rounds, there are identical. The process for each round has had five steps: Theta (θ), Rho (ρ), Pi (π), Chi (χ) and Iota (ι). They feature simple logical operations and permutations of the state bits. Should be noted that the initial state is all zero and in each round, the introduced data is mixed with the current state.

TABLE II. VALUE OF RC[I] CONSTANT

| | |
|---|---|
| RC[0] 0x0000000000000001 | RC[12] 0x000000008000808B |
| RC[1] 0x0000000000008082 | RC[13] 0x800000000000008B |
| RC[2] 0x800000000000808A | RC[14] 0x8000000000008089 |
| RC[3] 0x8000000080008000 | RC[15] 0x8000000000008002 |
| RC[4] 0x000000000000808B | RC[16] 0x800000000000808B |
| RC[5] 0x0000000080000001 | RC[17] 0x8000000000000080 |
| RC[6] 0x8000000080008081 | RC[18] 0x000000000000800A |
| RC[7] 0x8000000000008081 | RC[19] 0x800000008000000A |
| RC[8] 0x000000000000008A | RC[20] 0x8000000080008081 |
| RC[9] 0x0000000000000088 | RC[21] 0x8000000000008080 |
| RC[10] 0x0000000000008082 | RC[22] 0x0000000080000001 |
| RC[11] 0x000000080000000A | RC[23] 0x8000000800008008 |

***θ* step**:

$$C[x]=A[x,0]\oplus A[x,1]\oplus A[x,2]\oplus A[x,3]\oplus A[x,4]$$
$$D[x]=C[x-1]\oplus rot(C[x+1],1) \tag{1}$$
$$A[x,y]=A[x,y]\oplus D[x]$$

***ρ* and *π* steps:**

$$B[y,2\square x+3\square y]=rot(A[x,y],r[x,y]) \tag{2}$$

***χ* step:**

$$A[x,y]=B[x,y]\oplus((not B[x+1,y]) \text{ and } B[x+2,y]) \tag{3}$$

***1 Step:***

$$A[0,0]=A[0,0]\oplus RC \tag{4}$$

## B. Fault Injection Attacks

Among the techniques that can break the cryptographic algorithms, we find the fault injection attacks. This technique is to inject one or several faults during the hash process and to use the erroneous output to extract the secret information.

## III. KECCAK IMPLEMENTATION

## A. Implementation details of KECCAK

Figure 2 shows the block diagram of proposed KECCAK architecture. This architecture takes 1600-bit for the inputs data. Then it performs the padding operation and the hash process. The output data is 512-bit.

The architecture of KECCAK consists of four modules: (1) the Input/Output Interface, (2) the Control Unit, (3) the Padder Unit, and (4) the KECCAK Round.

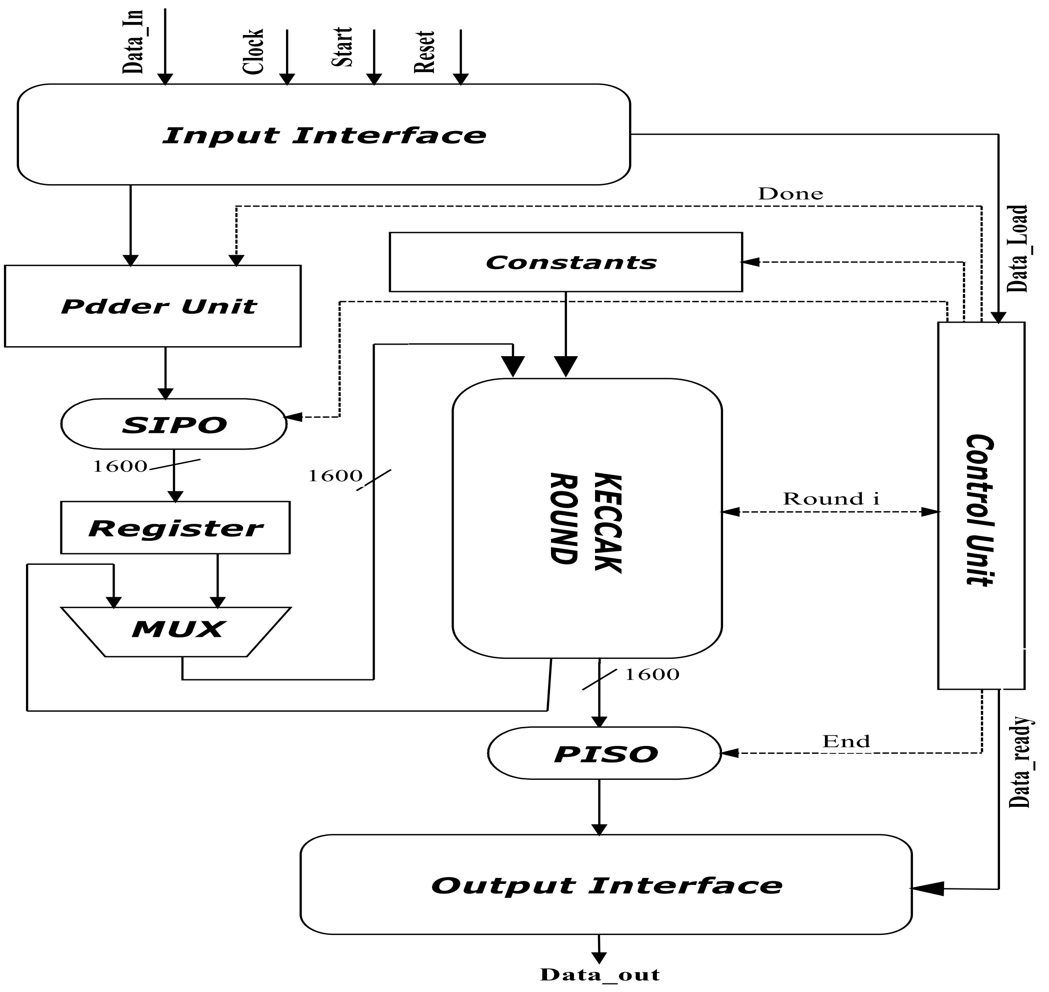

Fig. 2. Block Diagram KECCAK

- Input/Output Interface is the input blocks. The input data is 1600-bit length while the output is 512-bit wide. So the Input/output interface has to buffer the information data.
- Control Unit is used to ensuring the synchronisation between all modules.
- Padder Unit implements the padding operation and the inversions per byte procedure and has an output of 1600-bit which is the sponge function of KECCAK. Then a 2-to-1 multiplexer drives the output data from padder to the primary KECCAK components.
- KECCAK Round is the main component of proposed design. It requires 25 clock cycles to produce the 512-bit message digests where each clock cycle requires the previous round, as well as the constant value RC at the start of the each round.

The KECCAK round is composed of five components (Figure 3):

- Theta component θ: this operation is performed in three steps: the first step, it takes the input message bits and computes the addition modulo 2 between the lanes at each matrix column. The results are five xored columns. The second step, those columns are left rotated by one bit and xored again with the results of previous operations. Finally step, the results of the second step are driven to a finally XOR stage with the component θ input lanes.
- Rho component ρ: this operation performs rotations left each lane where the rotation number per lane is obtained from the remainder of the division between the fixed values and the length of the lanes.
- Pi component π: the Pi component is a simple operation was used instead of logic operations to modify the position between the lanes according to the specifications. In addition, logic operations (AND, XOR and NOT) between the lanes are used by the component. These functions are applied to entire rows of lanes for each row.
- Chi component χ: there are five rows of five lanes, the Chi component implement 25 NOT, 25 AND and 25 XOR of 64-bit logic gates.

- IOTA component ι: the final component realises an addition modulo 2 between the round constant value and the first lane (1599-1536).

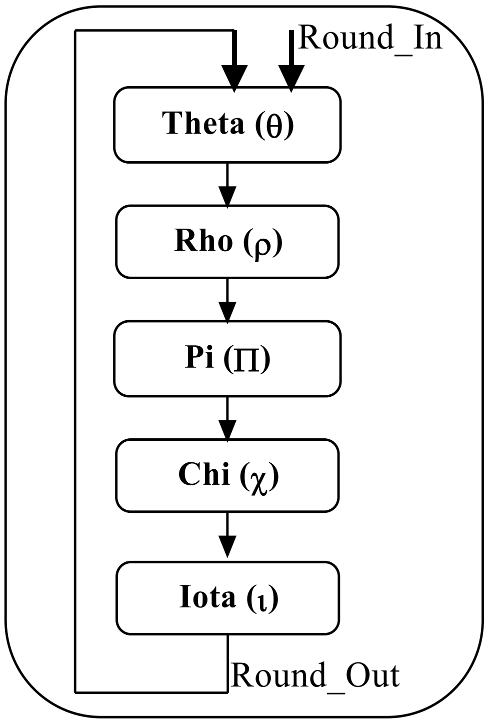

Fig. 3. The Structure of KACCAK Round

### B. *FPGA Implementation of KECCAK Architecture*

In this subsection, we present the hardware FPGA implementation of the proposed KECCAK architecture. The hardware description was performed via the VHDL language, simulated by ModelSim simulator and synthesised using ISE XILINX 14.1. The FPGA platform used is the Virtex-5.

Table 3 illustrated the occupied slices number; throughput (Gigabits per second), frequency (MegaHertz) and the efficiency (Gigabits per second per slices).

The data throughput and efficiency are calculated by equation 5 and equation 6 respectively.

$$Throughput = \frac{bit \times frequency}{clock\ cycles} \quad (5)$$

$$Efficiency = \frac{Throughput}{Area} \quad (6)$$

Table 3 shows that the proposed KECCAK architecture necessitates 1356 slices for 296.5 MHz working frequency and 11.86 Gbps throughput.

TABLE III. FPGA KECCAK IMPLEMENTATION: COMPARISON

| Design | Area (Slice) | Frequency (MHz) | Throughput (Gbps) | Efficiency (Mbps/slices) |
|---|---|---|---|---|
| [22] | 1414 | 271 | 12.3 | 8.68 |
| [23] | 2640 | 122 | 5.2 | - |
| Proposed | 1356 | 296.5 | 11,86 | 8.95 |

In addition, Table 3 presents a comparison between the proposed KECCAK designs and other previous works. Compared to [22] and [23], the proposed architecture has the lowest area and the highest working frequency. From hardware performances viewpoint, the proposed architecture requires 1356 slices for 296.5 MHz working frequency while the KECCAK design in [23] requires 2640 slices with 122 MHz working frequency. Although the design in [22] increases the throughput compared to our work, the proposed design is more efficient from area and frequency viewpoint. Therefore, our design realises a trade-off between the implementation hardware performances.

## IV. PROPOSED FAULT DETECTION SCHEME FOR THE KECCAK

In this section, we present the proposed scheme to protect the hardware KECCAK implantation against the fault injection attacks.

Duplicated the KECCAK hardware design means that the hash process data is duplicated. Therefore, two KECCAK round execute simultaneously. It is simple to scramble the KECCAK slices between two KECCAK rounds by using the hardware duplication technique.

We applied the scrambling technique at the end of each KECCAK operation. In other words, we applied this technique at the end of Theta, Rho, Pi, Chi and Iota.

Then, if a fault is injected into one data hash path, it causes faulty data process on the other data hash path.

The advantage of the proposed architecture is that this method avoids the fault injection attacks and does not modify the exact KECCAK Round process in the absence of attacks.

In this work, in order to increase the robustness against the fault attacks, we applied the scrambling at the bit level which means that each bit of the first data hash path is scrambling with the corresponding bit in the second data hash path

The proposed methodology is presented in Figure 4.

The slice KECCAK half (in data path 1) are scrambled with the KECCAK slice (in data path 2). The bit level scrambling technique causes a robust KECCAK design. In addition, in terms of hardware implementation, it is effortless to implement this technique. Also, it does not augment the implementation complexity level.

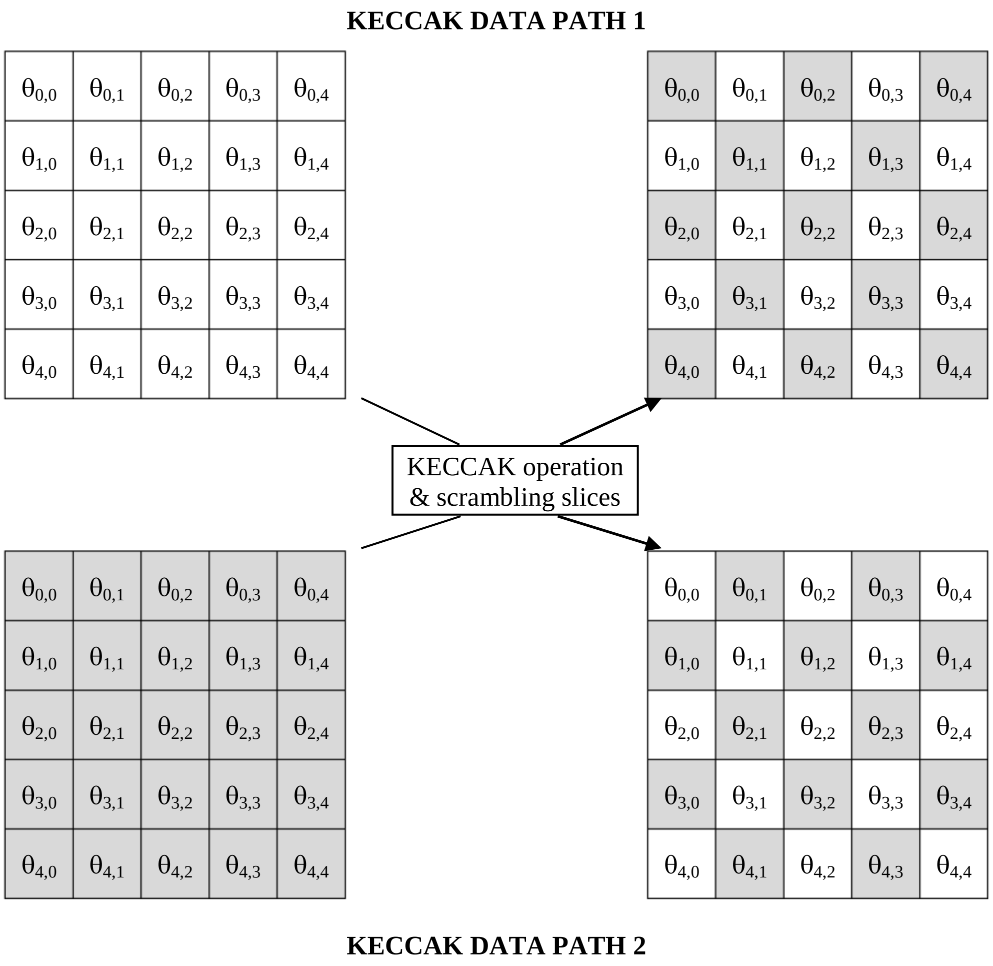

Fig. 4. Technique of scrambling in KECCAK operation

## V. Fault Detection Analysis

Many experiences of faults injection attacks were performed using the VHDL language to verify the robustness of the KECCAK architecture against the fault injection attacks. We considered two types of faults:

- Single-bit faults mean that one bit in the data hash path is changed.
- Multiple-bit faults mean that more than one bit in the data hash path is changed.

The single-bit and the multiple-bit faults are injected into all KECCAK operations where the erroneous bits number for the multiple-bit faults varies from 1 to 16. For this purpose, we developed a simulation fault model as shown in Figure 5.

The KECCAK detection scheme is tested using 17 tests different by fault multiplicity where each fault pattern is composed of 1000000 faulty vectors. The vector's length is 64 bits. The simulation faults attacks results are shown in Figure 6.

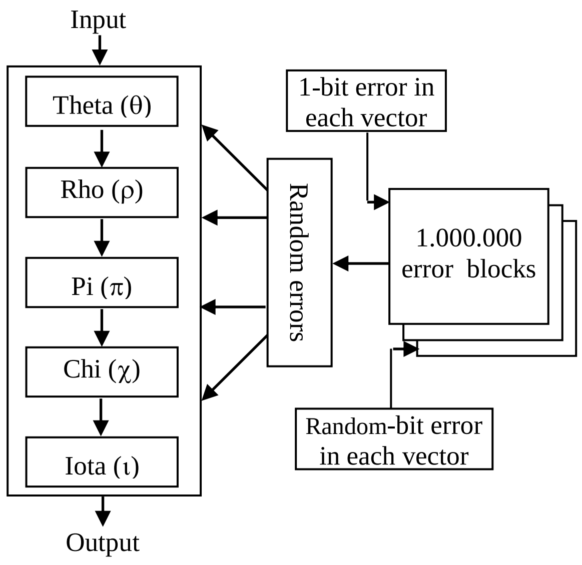

Fig. 5. Simulation model for fault attacks

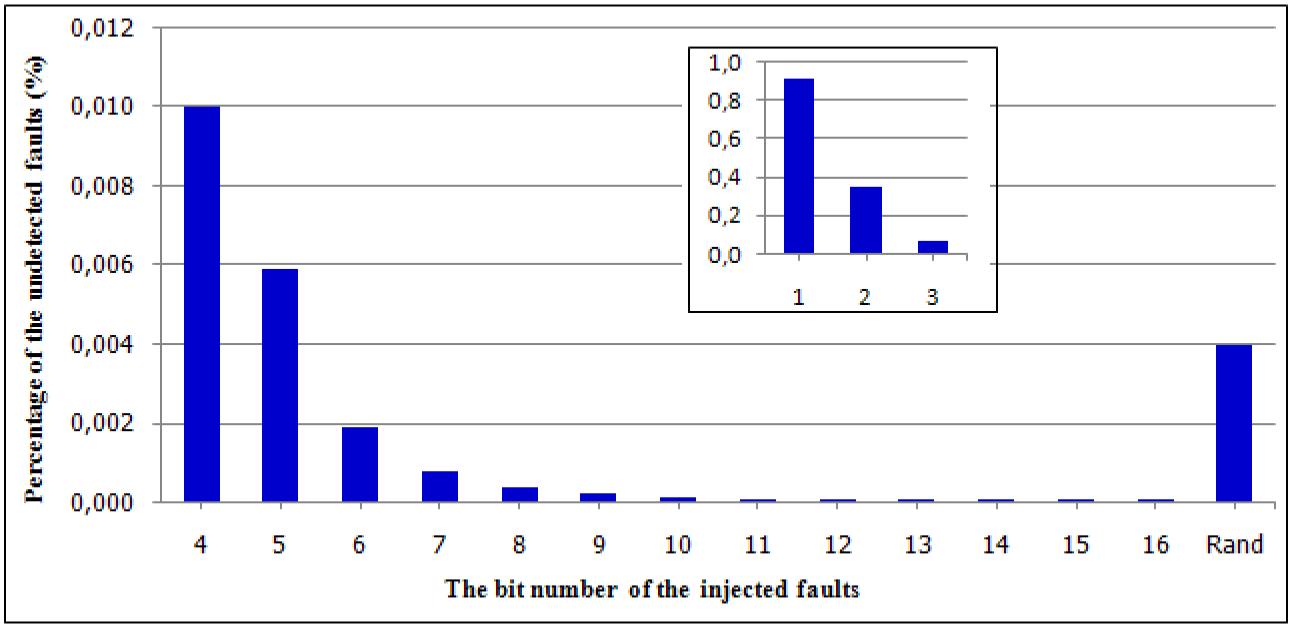

Fig. 6. Detection capability against fault attacks

As shown in Figure 6, the undetectable faults percentage decreases considerably when the fault multiplicity augmented. In the random faulty bit case, the percentage of the undetectable faults is about 0.004% which means that the detection capability percentage achieves 99.996%. Consequently, the proposed KECCAK detection scheme guarantees a high security level against fault attacks.

## VI. FPGA Implementation

In this section, we present the hardware FPGA implementation of the original KECCAK and the protected KECCAK designs. The hardware description was performed via the VHDL language the proposed architectures are simulated by ModelSim simulator and synthesised using ISE XILINX 14.1. The FPGA platform used is the Virtex-5.

Table 4 illustrated the occupied slices number; throughput (Gigabits per second), frequency (MegaHertz), the frequency and throughput degradations and the area overhead, for the protected and the unprotected KECCAK implementation.

TABLE IV. KECCAK FPGA Hardware Implementation: Results and Comparison

| Design | Area (Slice) *(Overhead)* | Frequency (MHz) *(Degradation)* | Throu. (Gbps) *(Degradation)* |
|---|---|---|---|
| Original KECCAK | 1356 | 296.5 | 11,86 |
| Protected KECCAK | 2260 *(66.66%)* | 291.3 *(1.75%)* | 11,65 *(1.77%)* |

As seen in Table 4, the original KECCAK hash function requires 1356 occupied slices for 296.5 MHz maximal frequency. However, the proposed protected KECCAK requires 66.66% more occupied slices and the maximal frequency decreased by 1.75% than the original KECCAK. Also, the proposed secured design causes 1.77% throughput degradation. Thus, our proposed KECCAK design realises a compromise between implementation cost and KECCAK robustness against fault attacks.

## VII. Conclusion

In this work, to improve the KECCAK safety, we proposed a new KECCAK fault detection scheme based on scrambling technique. We discuss the robustness of the proposed KECCAK architecture against fault attacks. We implemented the architectures: the original and the protected KECCAK on FPGA Virtex-5. Compared to the original implementation, the proposed KECCAK achieves 99.996% fault coverage and causes a very little frequency and throughput degradations. In the future works, we will try to protect the KECCAK architecture against the power attacks.

## References

[1] I. Ahmad and A. Das, “Analysis and detection of errors in implementation of SHA-512 algorithms on FPGAs”, The Computer Journal., vol 50( 6), pp. 728-738, 2007.

[2] M. Bahramali, J. Jiang, and A. Reyhani-Masoleh, “A fault detection scheme for the FPGA implementation of SHA-1 and SHA-512 round computations”, Journal of Electronic Testing, vol. 27, no. 4, pp. 517-530, 2011.

[3] Morris J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions", Federal Inf. Process. Stds. (NIST FIPS) - 202, August 2015.

[4] Fatma Kahri, Hassen Mestiri, Belgacem Bouallegue and Mohsen Machhout, "High Speed FPGA Implementation of Cryptographic KECCAK Hash Function Crypto-Processor", Journal of Circuits, Systems, and Computers, Vol.25(4), 2016.

[5] G. S. Athanasiou, G.-P. Makkas and G. Theodoridis, "High throughput piplined FPGA implementation of the new SHA-3 cryptographic hash algorithm", Int. Symp. Communications, Control and Signal Processing, pp. 538-541, May 2014.

[6] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "The KECCAK SHA-3 submission", Submission to NIST (Round3), http://keccak.noekeon.org/Keccak-submission-3.pdf, 2011.

[7] D. Barbara Nicholas and A. Sivasankar, "Design of FPGA based encryption algorithm using KECCAK hashing functions", International Journal of Engineering Trends and Technology, pp. 2438-2441, 2013.

[8] R. Karri, K. Wu, P. Mishra, and Y. Kim, “Concurrent error detection schemes of fault based side-channel cryptanalysis of symmetric block ciphers”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21(12), pp.1509-1517, 2002.

[9] S. Bayat-Sarmadi and M. A. Hasan,“On concurrent detection of errors in polynomial basis multiplication”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 15(4), pp. 413-426, Apr. 2007.

[10] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, and A. Szekely, “Uniform evaluation of hardware implementations of the round-two SHA-3 candidates”, in Proc. Conf. SHA-3 Candidate, pp. 1-16, 2010.

[11] M. Knezevic et al., “Fair and consistent hardware evaluation of fourteen round two SHA-3 candidates”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20(5), pp. 827-840, 2012.

[12] S. Bayat-sarmadi, M. Mozaffari-Kermani, and A. Reyhani-Masoleh, "Effcient and concurrent reliable realization of the secure cryptographic SHA-3 algorithm", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 33(7), July 2014.

[13] X. Guo and R. Karri, "Invariance-based concurrent error detection for Advanced Encryption Standard", In Proc. IEEE Design Automation Conference (DAC), 2012.

[14] X. Guo and R. Karri, "Recomputing with permuted operands: A concurrent error detection approach", IEEE Transactions on Computer-Aided Design of Integrated Circuits & Systems, vol. 32(10), pp. 1595-1608, 2013.

[15] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A lightweight high-performance fault detection scheme for the Advanced Encryption Standard using composite fields", IEEE Transactions on Very Large Scale Integration Systems, vol. 19(1), pp. 85-91, 2011.

[16] M. Karpovsky, K. Kulikowski, and A. Taubin, "Differential fault analysis attack resistant architectures for the Advanced Encryption Standard", In Smart Card Research and Advanced Applications VI, vol. 153, pp. 177-192, 2004.

[17] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers," In Proc. IEEE Design Automation Conference, pp. 579-584, 2001.

[18] P. Luo, Y. Fei, L. Zhang, and A. Ding, "Side-channel power analysis of di_erent protection schemes against fault attacks on AES", In Int. Conf. ReConFigurable Computing & FPGAs (ReConFig), 2014.

[19] P. Maistri and R. Leveugle, "Double-data-rate computation as a countermeasure against fault analysis. IEEE Trans. on Computers", vol. 57(11), pp.1528-1539, 2008.

[20] P. Luo, L. Zhang, Y. Fei, "Concurrent Error Detection for Reliable SHA-3 Design", IEEE International Great Lakes Symposium on VLSI, 2016.

[21] Hassen Mestiri, Fatma Kahri, Belgacem Bouallegue, Mohsen Machhout, "A high-speed AES design resistant to fault injection attacks", Microprocessors and Microsystems Journal, vol. 41, pp.47-55, 2016.

[22] J. Yaser, T. Lo’ai, T. Hala and M. Abidalrahman, “Hardware performance evaluation of SHA-3 candidate algorithms”, in the Journal of Information Security, vol. 3(2), pp. 69-76, 2012.

[23] F.D. Pereira, D. M. Ordonez, I. D. Sakai, A. M. de Souza, "Exploiting Parallelism on Keccak: FPGA and GPU Comparison", in the Parallel and Cloud Computing, 2013, vol. 2(1), p. 1-6.