An Improvement of Both Security and Reliability for Keccak Implementations on Smart Card

Pei Luo[1], Liwei Zhang[2], Yunsi Fei[1], and A. Adam Ding[2]

[1] Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115
silenceluo@coe.neu.edu, yfei@ece.neu.edu
[2] Department of Mathematics, Northeastern University, Boston, MA 02115
mathliwei@gmail.com, A.ding@neu.edu

Abstract. As the new SHA-3 standard, the security and reliability of Keccak have attracted a lot of attentions. Previous works already show that both software and hardware implementations of Keccak have strong side-channel power (electromagnetic) leakages, and these leakages can be easily used by attackers to recover secret key bits. Meanwhile, Keccak is vulnerable to random errors and injected faults, which will cause errors in the computation results. In this paper, we introduce a scheme based on the round rotation invariance property of Keccak to reduce the side-channel leakages while improve its reliability. The proposed scheme is resource friendly. Side-channel analysis results show that this method can efficiently reduce the side-channel leakages of Keccak implementations. Meanwhile, fault injection simulation results show that the proposed scheme can effectively improve the reliability of Keccak implementation, with error coverage almost 100%.

1 Introduction

Keccak will be widely used in cryptographic systems because it has been selected as the new SHA-3 standard recently. Keccak is a function family and can be easily used for regular hashing, salted hashing, stream encryption, pseudo-random sequence generator, thus it will be widely used in different kinds of cryptographic applications [1]. Effective methods are required to protect Keccak implementations against different kinds of attacks. In this paper, we focus on the protection of Keccak implementations on smart card platform against both side-channel power (electromagnetic) attacks and fault attacks.

Previous papers introduce different kinds of side-channel attack methods to conquer MAC-Keccak, both software [2,3] and hardware systems [4,5]. These attacks focus on either θ step [2,3,5] or the first round output [4], and the results show that attackers can efficiently recover key bits using the side-channel leakages of Keccak operations with very small number of power traces. Meanwhile, Keccak is vulnerable to random errors and injected faults when used for hashing and encryption.

Previous works protect Keccak against side-channel analysis and fault attacks separately. To protect Keccak against side-channel analysis, the designers proposed to

hide the leakages using secret sharing in [6]. This method introduces random numbers to mask the secret key bits and can effectively protect the systems against side-channel analysis. The deficiency of this method is that it has very high resource consumption overhead. For software implementations, two-share masking will cause two times resource consumption, while three times resource consumption will be introduced for hardware implementations with three-share masking. Besides secret sharing, some other methods such as random shuffling can reduce the leakages significantly while much lower resource overhead will be introduced [7,8]. While these methods can protect Keccak against side-channel analysis, they are still vulnerable to random errors and fault injection attacks. To protect Keccak against fault injection attacks, the authors in [9] proposed to protect Keccak on FPGAs using the round rotation invariance property. Other commonly used error detection schemes such as double redundancy, parity checking are also suitable for Keccak protection. The problem of these countermeasures is that they still have strong side-channel leakages and thus vulnerable to side-channel analysis attacks.

A simple method to solve this problem is to combine two different protection methods together directly, but the corresponding resource overhead will be very high. In this paper, we propose to use round rotation invariance property of Keccak sponge function to protect Keccak implementations on smart card platform against both side-channel attacks and fault attacks. The advantage of this scheme is that it can protect Keccak against both fault attacks and side-channel attacks, thus it will have much lower resource overhead than combining two protection methods together directly. Meanwhile, this scheme requires only minor modification of Keccak sponge function, thus it can be used together with with other protection schemes for even higher security and reliability level.

In this paper, we focus on the implementations for resource restricted platform, smart card, and test our scheme based on the implementation of AVR-Crypto-Lib [10]. Meanwhile, we run fault injection simulation at algorithm level for accurate error coverage results, which show that the proposed scheme can significantly reduce the side-channel leakages of Keccak implementations and improve its reliability against random errors and injected faults.

The rest of this paper is organized as follows. In Section 2, the preliminaries of Keccak which are needed in this paper will be introduced. In Section 3, the proposed scheme will be introduced, then attacks and simulation results will be given in Section 4. In the end, we will conclude this paper in Section 5.

2 Preliminaries of Keccak

Keccak is a hash function family based on the Sponge construction, as shown in Fig. 1 [11,12]. Keccak has two phases: 1) absorbing and 2) squeezing. In the absorbing phase, the message is broken into blocks (each block size is r bits, where r is the bit rate), which are absorbed iteratively by the permutation function f. Each f function works on a state at a fixed length $b = r + c$ (c is called capacity). In the squeezing phase, outputs are squeezed also by f functions and the length of the output is configurable (a multiple of r bits).

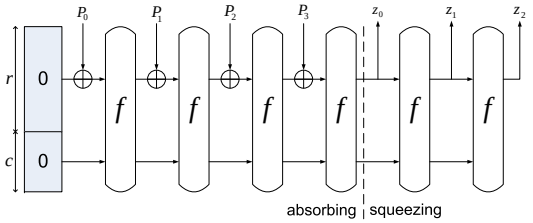

Fig. 1: The sponge construction

All of the 1600-bit states are organized in a 3-D array, as shown in Fig. 2. Each bit is addressed with three coordinates, written as $S(x, y, z),\ x, y \in \{0, 1, ..., 4\},\ z \in \{0, 1, ..., 63\}$. 2-D entities, *plane*, *sheet* and *slice*, and 1-D entities, *lane*, *column* and *row*, are also defined in Keccak and shown in Fig. 2.

The state S is composed of 25 lanes, denoted as:

$$S = \{L_{i,j}\}, i, j \in \{0, 1, 2, 3, 4\}, \tag{1}$$

and each lane $L_{i,j}$ contains 64 bits for Keccak-1600.

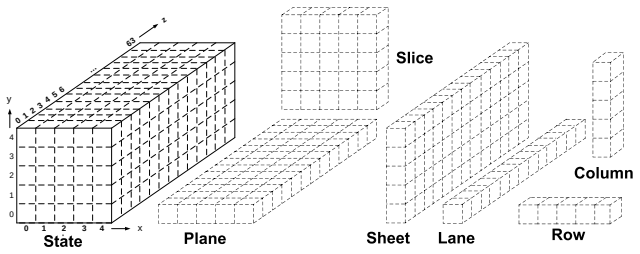

Fig. 2: Terminology used in Keccak

Notations: We note here that in this paper, we use this 3-D array method to denote the Keccak state and intermediate states. We use coordinates x, y and z to locate each bit, in which $x, y \in \{0, 1, ..., 4\}$, and $z \in \{0, 1, ..., 63\}$, we also define $X = [0 : 4]$, $Y = [0 : 4]$ and $Z = [0 : 63]$ to stand for the positions in each row, column and lane. Be aware that coordinates x, X and y, Y are modular 5 while z, Z are modular 64.

The f permutation function of Keccak-1600 consists of 24 rounds of operations, where each round has five sequential steps:

$$R_{i+1} = \iota \circ \chi \circ \pi \circ \rho \circ \theta(R_i),\ i \in \{0, 1, \cdots, 23\} \tag{2}$$

in which R_0 is the initial input. Details of each step are described below:

$-$ θ is a linear operation which involves 11 input bits and outputs a single bit. Each output state bit is the XOR between the input state bit and two intermediate bits

produced by its two neighbor columns. The operation is given as follows:

$$S'(x,y,z) = S(x,y,z) \oplus (\oplus_{i=0}^{4} S(x-1,i,z)) \\ \oplus (\oplus_{i=0}^{4} S(x+1,i,z-1)). \quad (3)$$

$-$ ρ is a permutation over the bits of the state along z-axis (in lanes).

$-$ π is a permutation over the bits of the state within slices, only the center bit ($x = 0,\ y = 0$) of the slice does not move. All other bits are permuted to other positions depending on their original coordinate.

$-$ χ is a non-linear step that contains mixed binary operations. Every bit of the output state is the result of an XOR between the corresponding input state bit and its two neighboring bits along the x-axis (in a row):

$$S'(x,y,z) = S(x,y,z) \oplus (\overline{S(x+1,y,z)} \cdot S(x+2,y,z)). \quad (4)$$

$-$ ι is a binary XOR with a round constant which is publicly known.

Further details of Keccak and Sponge construction can be found in [11,12,13].

3 Round Rotation Invariance Based Scheme

Previous papers proposed to used the structures of crypto algorithm for security and reliability enhancement. For example, in [14,15], the authors propose to use the invariance of AES for error detection. Such schemes make use of the structures of crypto algorithms for protections, thus they are usually efficient and effective. In this section, we introduce the round rotation invariance property of Keccak sponge function, and how to use it for both side-channel leakage reduction and fault detection on smart card platform.

3.1 Invariance of Keccak Permutation Function

Keccak reference manual explained that the mapping ι is added to disrupt the symmetry of Keccak operations (translation-invariant in the z direction), thus to avoid slide attacks [12]. Operations other than ι are translation-invariant in the z direction, which means the input can be rotated in z direction and then rotated back with the result unchanged:

$$g(in) = ROT^{-1}(g(ROT(in,\alpha)),\alpha). \quad (5)$$

In (5), ROT stands for the round rotation at z direction and α is a random number ($0 \leq \alpha \leq 63$), g stands for Keccak operations θ, ρ, π, and χ. While (5) holds for other four operations, it it not true for ι operation, because ι involves constant numbers $\iota_c = \{\iota_c[0], \iota_c[1], \cdots, \iota_c[23]\}$ besides the input in for each round. Thus we can rotate ι_c also α bits for operation invariance:

$$\iota(in,\iota_c) = ROT^{-1}(\iota(ROT(in,\alpha), ROT(\iota_c,\alpha)),\alpha). \quad (6)$$

Which means that we can denote Keccak sponge function over the first round input S_0 and ι_c as following:

$$f(in, \iota_c) = ROT^{-1}(f(ROT(in, \alpha), ROT(\iota_c, \alpha)), \alpha). \tag{7}$$

In this paper, we can also denote the right side of (7) as $f'(in, \alpha)$. It means the modified Keccak function has in and α as input, and round rotates each lane of in and ι_c α bits for Keccak operations, and rotate the final results back with the result unchanged. We give the details as follows.

For the first round input $S_0 = \{L_{i,j}\}, i, j \in \{0, 1, 2, 3, 4\}$, rotate each lane $L_{i,j}$ α bits at z direction:

$$L'_{i,j} = ROT(L_{i,j}, \alpha), i, j \in \{0, 1, 2, 3, 4\}, 0 \leq \alpha \leq 63. \tag{8}$$

Denote the new input composed of these rotated lanes as S'_0:

$$S'_0 = \{L'_{i,j}\}, i, j \in \{0, 1, 2, 3, 4\}. \tag{9}$$

We use ι'_c to stand for the rotated ι_c (the set of 24 constant numbers for ι operations in 24 rounds):

$$\iota'_c[i] = ROT(\iota_c[i], \alpha), i \in \{0, 1, \cdots, 23\}. \tag{10}$$

The Keccak operation results (S_{24} and S'_{24}) based on these two input (S_0, ι_c and S'_0, ι'_c) are as following:

$$\begin{cases} S_{24} = f(S_0, \iota_c) = f'(S_0, 0) \\ S'_{24} = f(S'_0, \iota'_c) = f'(S_0, \alpha) \end{cases}. \tag{11}$$

Then the following equation holds:

$$S_{24} = ROT^{-1}(S'_{24}, \alpha), \tag{12}$$

which means that Keccak result over the rotated input lanes and ι_c constant numbers can be rotated back to get the original result [9]. In the following section, we will demonstrate how to use this round rotation invariance property to reduce the side-channel leakages while increase reliability of Keccak implementations on smart card platform.

3.2 Invariance-Based Protection Scheme

The invariance property of Keccak permutation function can be used for side-channel leakage reduction because it can distribute leakages from one point to multiple points. Meanwhile, this property can also be used for fault detection by comparing the results of two rotated Keccak implementations with different α. We devise the structure in Fig. 3 for Keccak implementation on smart card for both side-channel leakages reduction and reliability enhancement.

It works like this, when the Keccak implementation receives the input message in, it starts working by generating one random number α_1, and use this random number for the rotated Keccak computation shown in Section 3.1. Then another random number α_2 will be generated and used for rotated Keccak to generate O_2. The output O_1 and O_2 are compared for fault detection.

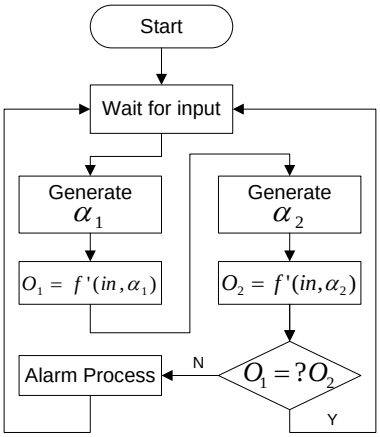

Fig. 3: The proposed invariance-based protection scheme

How it reduces side-channel leakages For Keccak implementations on smart card, if we randomly rotate each input lanes before Keccak operations, the leakage of one key bit will be distributed from one time point to multiple time points. We assume 64 bits in one lane ($L[0:63]$) are rotated by α bits ($\alpha \geq 1$) at z direction, and the rotated lane bits are $\{L[64-\alpha:63]\ L[0:63-\alpha]\}$. For 8-bit architectures (either microprocessor or FPGA), only 8 bits in each lane will be processed in one clock cycle. Take eight key bits at position $L[0:7]$ as example, after round rotation, these bits will be shifted α bits along z axis to be $L[\alpha:\alpha+7]$ ($\alpha+7$ is modular 64). Then the leakages of $L[0:7]$ will be distributed from one clock cycle to multiple clock cycles. Thus the leakages of these bits will be significantly reduced.

For differential power analysis (DPA), one bit may be moved into eight different bytes. For correlation power analysis (CPA), previous papers always attack eight bits in one lane together for higher signal-to-noise ratio (SNR). Thus:

- If α is a time of eight, these eight bits are still in one byte, the leakage of these eight key bits are distributed from one time point to another time point.
- If α is not a time of eight, these eight bits are distributed into two adjacent bytes, and leakages of other key bits in the same bytes will be working like noise and help to further reduce their leakages.

To evaluate the resilience of proposed scheme against side-channel analysis, we implement original Keccak and Keccak with the proposed scheme on a SASEBO-W board, which is designed specifically for side-channel evaluations [16]. Keccak has been implemented in 64-bit, 32-bit, 16-bit and 8-bit structures, examples are source code provided Keccak official site [17]. For compact platforms such as 8-bit smart cards, example implementations are like [18] (compact IC design) and AVR-Crypto-Lib [10] (for smart card software design). In this paper, we refer to AVR-Crypto-Lib [10] for implementation, and run side-channel analysis and fault injection on it.

How it improves the reliability For error detection methods like duplication or redundancy, they are incapable of detecting those faults that cause the same errors in both copies [14]. For such redundancy based schemes, attackers who have the ability to inject the same faults at the same positions of both copies will be able to bypass the error detection modules. But for the proposed scheme in Fig. 3, the positions of message bytes will be different for different random number α. Thus it will be impossible for attacker to inject faults at the same positions if he has no knowledge of two random numbers α_0 and α_1. To evaluate the error detection coverage of the proposed scheme, we run fault injection simulation at algorithm level, and details and results will be given in Section 4.

3.3 Advantages of the Proposed Scheme

As discussed in previous sections, the proposed scheme can significantly reduce side-channel leakages of Keccak, and also improve reliability of the Keccak implementations on smart card platform. Comparing with combining two different countermeasures together directly, the proposed scheme is resource friendly. This scheme has the following advantages:

- Comparing with countermeasures like secret sharing, the proposed scheme is easy to implement. The proposed scheme mainly operates on the input message lanes, and only needs to change the ι_c table for different random rotation number α, the sponge function needs no extra modifications.
- It can be easily combined with other countermeasures. The proposed scheme does not change the operations of sponge function, it can be easily combined with other countermeasures such as secret sharing, random permutation and random delay to improve the side-channel security. Error detection methods such as parity checking can be directly added on the proposed scheme to improve the error coverage for higher reliability.
- It is resource friendly and can be efficiently implemented on resource restricted applications. The proposed scheme is resistant to both fault attacks and side-channel attacks at the same time. Comparing with combining two different schemes together, the proposed scheme is resource efficient.

4 Side-Channel Power Analysis and Fault Injection Simulation of the Proposed Scheme

In this section, we attack the original Keccak implementation and the implementation protected with the proposed scheme. For power analysis, we use a LeCroy WaveRunner 640Zi oscilloscope to sample all power traces of the implementations on SASEBO-W board, and we use CPA attack as an example to demonstrate the results. Meanwhile, we simulate the error detection by injecting fault at algorithm level.

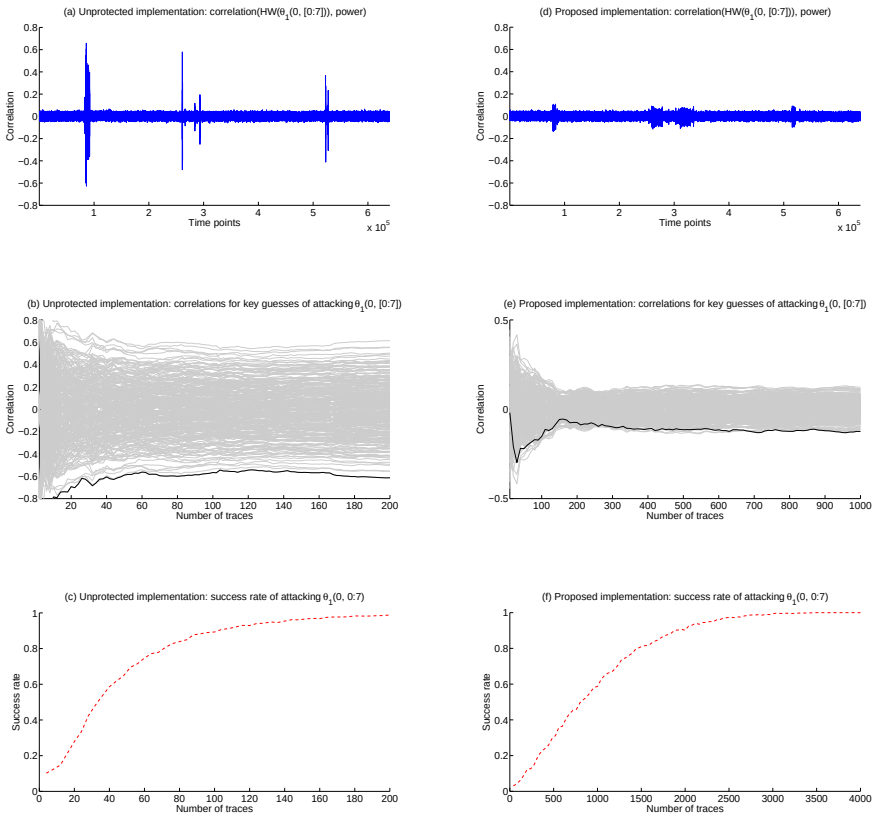

Fig. 4: Side-channel power analysis results: the left side are the results of attacking the original implementation, the right side are the results of attacking the implementation protected with the proposed scheme.

4.1 Side-Channel Analysis Results of Protected and Unprotected Implementations

For the original implementation, we sample 500 traces and use CPA to attack the first step of θ operation, the same as [2,3,5]. We attack eight bits each time for higher SNR, and use the result for the first eight bits of the first θ_1 lane ($\theta_1(0, [0:7])$) in this paper. The correlation between the power consumption and $HW(\theta_1(0, [0:7]))$ is shown in Fig. 4(a). Fig. 4(a) shows that the correlation is reaching 0.7 (or negative -0.7), which is very strong for cryptographic systems. This strong leakage can be easily used by attacker to retrieve key bits information.

Fig. 4(b) shows the correlation between different key assumptions and power consumption. The blackened trace is for the correct key and the gray traces are for false keys. It shows that the correlation between the correct key guess will stand out of the key guesses very quickly. Fig. 4(c) shows the success rate of CPA attacks. The attackers only need about 200 traces to recover the key bits with success rate 100%. Thus, without protection, the Keccak implementations on smart card are vulnerable to side-channel attacks, and a very small number of power traces are enough to successfully extract all the key bits.

For the proposed scheme, the leakages at one point are distributed to multiple points, and we anticipate that the proposed scheme can improve the side-channel security of Keccak implementations. For the proposed implementation, we sample 5,000 traces and run CPA on them, and show the correlation result in Fig. 4(d). It shows that for the implementation protected with the proposed countermeasure, the leakage decreases significantly. Comparing with the leakage of the original implementation, the correlation of the proposed scheme is much smaller, almost covered by noise in the system. Thus it's anticipated that the protected implementation will be much more difficult to conquer than the original implementation.

Fig. 4(e) shows the correlation between different key assumptions and power consumption. It shows that the right key's correlation is significantly reduced and it stands out of the key guesses very slowly. For the original implementation shown in Fig. 4(c), it only needs more than 100 traces for the correct key guess to stand out of the wrong key guesses. For the implementation protected with the proposed scheme, the leakages are reduced significantly and it needs many more traces for the right key to stand out of the wrong key guesses. Success rate result in Fig. 4(f) shows that attackers need about 4,000 traces to recover the key bits for this implementation, 20 times more than the original implementations.

Above results show that the proposed scheme can effectively protect Keccak implementation against side-channel analysis. Keccak implementation with the proposed countermeasure is much more difficult to conquer than the original implementation. Besides the reduction of side-channel leakages, the round rotation invariance of Keccak can also be used to protect Keccak against random errors and injected faults, and simulation results will be given in Section 4.2.

4.2 Fault Injection Simulation Result

In this paper, we do not differentiate faults and errors, which means that we only care about faults which generate errors at the output. We simulate fault injection and get the

error coverage of the proposed scheme at algorithm level. In this paper, we assume two attacker models, the weak attacker model and strong attacker model:

- Weak attacker model: the attackers can precisely inject faults at a given clock cycle, but have no control of the injected faults.
- Strong attacker model: the attackers can precisely inject faults at a given clock cycle, and have fully control of the injected faults.

For fault injection simulation, we target at four implementations, (a) the implementation protected with our proposed scheme, (c) the implementation protected with secret sharing, (c) the implementation protected with double copy redundancy and (d) implementation with parity checking error detection.

For both attacker models, we assume the attacker can distort one byte at a specific clock cycle, and he can inject either single or multiple byte faults into the system. For single byte fault model simulation, we randomly inject random faults (0-255) into one random byte of the input (200 bytes) of both copies. If the results of these two copies are equal while different from the original result, we think the injected errors are not detected. For multiple bytes fault injection, we inject random faults into from one to five bytes of both copies. Besides the fault positions and faults injected, the number of faults are also randomly generated.

For each implementation, we run the fault injection simulation for ~$2 * 10^8$ times for both single fault and multiple faults models. Simulation results show that for single fault model, the proposed scheme can detect 99.998% of the injected faults. For multiple faults model, the proposed scheme can detect 99.999975% of the injected faults. Thus the proposed scheme has a very high fault coverage for Keccak implementations on smart card platform. We summarize the single byte fault injection results of different protection schemes in Table 1.

Table 1: Comparison of countermeasures

	Leakage	Error coverage	
	Reduction[I]	Weak[II]	Strong[III]
Proposed scheme	85%	99.998%	99.868%
Secret sharing	100%	0%	0%
Double redundancy	0%	99.602%	0%
Parity checking	0%	99.606%	0%

[I] First order side-channel leakage reduction, the ratio of correlation;
[II] Weak fault injection model, attackers can inject faults into the same byte, but cannot control the injected fault values;
[III] Strong fault injection model, attackers have fully control of the fault injection positions and values.

From the fault injection simulation result, we can see that the proposed scheme can efficiently protect different applications of Keccak function, such as integrity checking,

hashing, stream encryption, etc. While advanced attacker can bypass some simple error detection methods, it will be almost impossible for him to bypass the error detection module in the proposed implementation without know ledges of the random numbers α_1 and α_2.

For double copy redundancy and parity checking, although they can detect most of the injected faults under weak attacker model, they will be easily bypassed by attackers under strong attacker model. What's more, such simple error detection schemes have no resilience to side-channel power analysis at all. Theoretically, secret sharing scheme can delete all the side-channel power leakages, but it has no effect against fault injection attacks. Thus, comparing with secret sharing, the advantage of the proposed scheme is that it can also improve the reliability of Keccak implementations.

5 conclusion

We present a method to reduce the side-channel leakages and improve the reliability of Keccak implementations simultaneously on smart card platforms. This method is easy to implement and can be effciently applied to both hardware and software Keccak implementations on source restricted platform. The proposed scheme only introduces minor modifications of the original algorithm so it can be combined easily with other countermeasures. Real attack results show that the proposed scheme can efficiently reduce the side-channel leakages of Keccak implementations, while it can detect almost 100% of the random errors and injected fault. Futher work will be different countermeasures of Keccak against side-channel analysis and fault attacks, on different platforms.

References

[1] G. Bertoni, J. Daemen, M. Peeters, and G. Assche, "Permutation-based encryption, authentication and authenticated encryption," *Directions in Authenticated Ciphers*, 2012.

[2] M. Taha and P. Schaumont, "Side-channel analysis of MAC-Keccak," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, June 2013, pp. 125–130.

[3] M. Taha and P. Schaumont, "Differential power analysis of MAC-Keccak at any key-length," in *Int. WkShp on Security*, Nov. 2013, pp. 68–82.

[4] P. Luo, Y. Fei, X. Fang, A. A. Ding, D. R. Kaeli, and M. Leeser, "Side-channel analysis of MAC-Keccak hardware implementations," in *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '15, 2015, pp. 1:1–1:8.

[5] P. Luo, Y. Fei, X. Fang, A. Ding, M. Leeser, and D. Kaeli, "Power analysis attack on hardware implementation of MAC-Keccak on FPGAs," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, Dec 2014, pp. 1–7.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Building power analysis resistant implementations of Keccak," in *Second SHA-3 candidate conference*, vol. 142. Citeseer, 2010.

[7] T. Gneysu and A. Moradi, "Generic side-channel countermeasures for reconfigurable devices," in *Cryptographic Hardware and Embedded Systems CHES 2011*, 2011, vol. 6917, pp. 33–48.

[8] P. Luo, L. Zhang, Y. Fei, and A. Ding, "Towards secure cryptographic software implementation against side-channel power analysis attacks," in *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*, July 2015, pp. 144–148.

[9] S. Bayat-Sarmadi, M. Mozaffari-Kermani, and A. Reyhani-Masoleh, "Efficient and concurrent reliable realization of the secure cryptographic SHA-3 algorithm," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 7, pp. 1105–1109, July 2014.

[10] *AVR-Crypto-Lib*, 2015 (accessed Oct 5, 2015). [Online]. Available: http://avrcryptolib.das-labor.org/

[11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," *Submission to NIST (Round 2)*, 2009.

[12] G. Bertoni, J. Daemen, M. Peeters, and G. Assche, "The Keccak reference," *Submission to NIST (Round 3)*, January, 2011.

[13] N. F. Pub, "DRAFT FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," *Federal Information Processing Standards Publication*, 2014.

[14] X. Guo and R. Karri, "Recomputing with permuted operands: A concurrent error detection approach," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 10, pp. 1595–1608, Oct 2013.

[15] X. Guo and R. Karri, "Invariance-based concurrent error detection for Advanced Encryption Standard," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12, 2012, pp. 573–578.

[16] T. Katashita, Y. Hori, H. Sakane, and A. Satoh, "Side-channel attack standard evaluation board SASEBO-W for smartcard testing," *Power*, vol. 3, p. 400, 2012.

[17] *Reference and optimized code in C*, 2015 (accessed Oct 5, 2015). [Online]. Available: http://keccak.noekeon.org/KeccakReferenceAndOptimized-3.2.zip

[18] P. Pessl and M. Hutter, "Pushing the limits of SHA-3 hardware implementations to fit on RFID," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, 2013, vol. 8086, pp. 126–141.