An Efficient Scheme to Reduce Side-Channel Leakage of MAC-Keccak for 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
[2] Department of Mathematics, Northeastern University, Boston, MA 02115

Abstract. As the new SHA-3 standard, the side-channel security of Keccak has attracted a lot of attentions. Some works show that both software and hardware implementation of Keccak have strong side-channel leakages, and these leakages can be used easily by an attacker to recover secret key information. Secret sharing schemes are introduced to mask the leakages, while such schemes will incur large resource overhead. In this paper, we introduce another scheme based on the round rotation probability of Keccak to reduce the side-channel leakages. This scheme is easy to implement while it can efficiently help to reduce the side-channel leakages of Keccak.

Keywords: Keccak, MAC-Keccak, SHA-3, Side-channel attacks, Countermeasure

1 Introduction

Keccak will be widely used in cryptographic systems because it has been selected as the new SHA-3 standard recently. This requires thoroughly evalution of its security properties, and methods to protect it against different kinds of attacks. In this paper, we focus on the protection of MAC mode of Keccak, MAC-Keccak, against side-channel power or electromagnetic attacks.

Previous papers introduces different kinds of side-channel attack methods to conquer MAC-Keccak for both software and hardware systems [1,2,3,4]. These attacks focus on either θ step [1,2,4] or the first round output [3], and the results show that attackers can efficiently recover key bits using the side-channel leakages of Keccak operations.

Meanwhile, the designers of Keccak also designed some countermeasures to protect Keccak agaisnt side-channel analysis. In [5], the designers of Keccak proposed to protect Keccak using secret sharing to mask the leakages. This method can effectively hide the leakages, but the resource overhead is very high. 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 leakage cancellation, some other methods can reduce the leakages significantly while much lower resource overhead will be introduced. For example, by randomly shuffling the operations of cryptographic system, leakages could be distributed from one time points to multiple points [6].

For Keccak in crypto systems, there are several commonly used implementations, such as slice based and lane based. Meanwhile, Keccak has been implemented in 64-bit, 32-bit, 16-bit and 8-bit structures (examples are source code provided Keccak official site [7]). For compact platforms such as 8-bit smart cards, resource will be very limited and more efficient countermeasures should be developed for them, suitable for 8-bit implementations (examples are the work from Peter Pessl and Michael Hutter designed for compact IC design[8] and AVR-Crypto-Lib [9]). In this paper, we will show a method to reduce the leakages of Keccak on smart card platform using the mathematical properties of Keccak operations. Results show that this scheme can significantly reduce the leakages of Keccak while resource overhead is very small.

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 implementation and attack 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 [10,11]. 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).

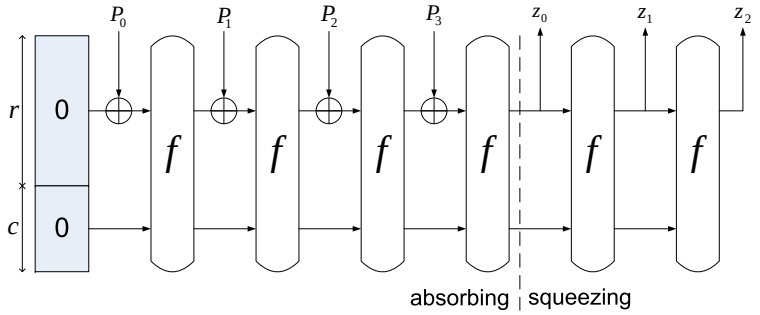

Fig. 1: The sponge construction

The default Keccak mode is Keccak-1600, with $r = 1024$ and $c = 576$ [10,11]. 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.

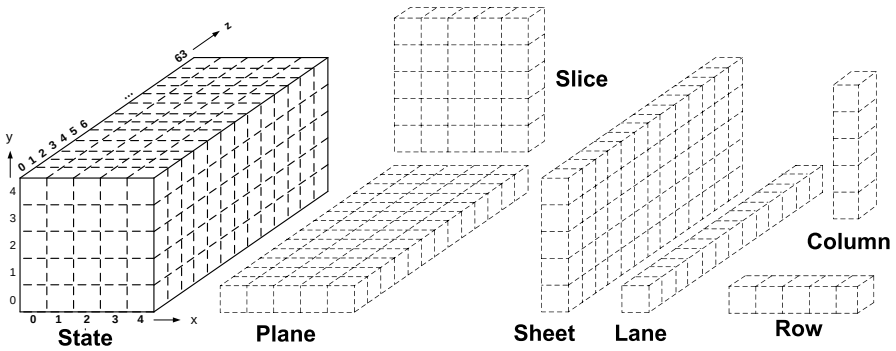

Fig. 2: Terminology used in Keccak

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:

$-\ \theta$ 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:

$$A(x, y, z) = a(x, y, z) \oplus (\oplus_{i=0}^{4} a(x-1, i, z)) \oplus (\oplus_{i=0}^{4} a(x+1, i, z-1)). \tag{3}$$

Here, we denote the input to each Keccak operation as a while the output as A. The two intermediate bits in this operation are the parity of two columns, $\oplus_{i=0}^{4} S(x-1, i, z)$ and $\oplus_{i=0}^{4} S(x+1, i, z-1)$, respectively.

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

$-\ \pi$ 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.

$-\ \chi$ 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):

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

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

Keccak is a function family and it can be easily used for regular hashing, salted hashing, stream encryption, pseudorandom sequence generator, thus it will be widely used in different kinds of cryptographic applications. So protection of Keccak against soft errors and injected faults is important for reliable cryptographic design. Further details of Keccak and Sponge construction can be found in [10,11,12].

3 Round Rotation Based Scheme

3.1 Theorem of the proposed scheme

For each run of Keccak, the cryptographic needs to generate a random number α between 0 and 63. This random number α is used for round rotation. For the first round input $S_0 = \{L_{i,j}\}, i, j \in \{0, 1, 2, 3, 4\}$, we can rotate each lane $L_{i,j}$ this α bits:

$$L'_{i,j} = ROT(L_{i,j}, \alpha), i, j \in \{0, 1, 2, 3, 4\}, 0 \leq \alpha \leq 63 \tag{5}$$

If we use these rotated lanes as the new input state S'_0, the new state can be denoted as:

$$S'_0 = \{L'_{i,j}\}, i, j \in \{0, 1, 2, 3, 4\}. \tag{6}$$

The Keccak operation results (S_{25} and S'_{25}) based on these two input (S_0 and S'_0) are as following:

$$\begin{cases} S_{25} = Keccak(S_0, \iota_c) \\ S'_{25} = Keccak(S'_0, \iota'_c) \end{cases}, \tag{7}$$

in which ι'_c stands for the rotated ι_c (the set of 25 constant numbers for ι operations in 25 rounds):

$$\iota'_c[i] = ROT(\iota_c[i], \alpha). \tag{8}$$

Here i is the index of round number and $0 \leq i \leq 24$.

Then the following equation holds:

$$S'_{25} = ROT(S_{25}, \alpha), \tag{9}$$

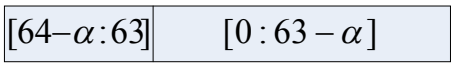

Fig. 3: Lane rotation

which means that after applying round rotation to the lanes of S_{25} with α bits, the result will be the lanes of S'_{25} respectively.

We denote one lane rotation as in Fig. 3. The 64 bits ([0:63]) are rotated by α bits, and the rotated lane bits are $\{[64-\alpha:63]\ [0:63-\alpha]\}$.

Take key bits at position $\{0,[0:7],0\}$ here as example, then these 8 bits will be shifted α bits along x-axis. Thus:

- These 8 bits are still in one one byte, but will not be operated at the same clock cycle as before.
- These 8 bits may be distributed to two bytes and these two bytes may be operated at different clock cycle as before. If one byte of them is executed in the same cycle as before (for example, $\alpha < 8$ or $\alpha > 56$), only part of them will be operated in the same cycle as before.

Thus the distribution of one key bit will be distributed from one time point to multiple point, thus the leakage will be reduced.

3.2 Implementation of the proposed scheme

Equation 9 shows that the result of f function (Keccak) for rotated lanes can be rotated back to form the correct result. The result can also be left rotated as input to next stage of f functions. For next stage f function, the input from last stage f function should not be rotated again while the input message bits should be rotated.

For cryptorgaphic systems, the host machine can generate the random number α and rotate the message bits before sending them to the smart card system, then smart card only needs to rotate the key bits to save time.

4 Side-Channel Power Analysis on the Proposed Scheme

4.1 Leakage of the original implementation

To evaluate the proposed scheme on smart card platform, we implement Keccak on SASEBO-W platform based on AVR-Crypto-Lib. For proposed scheme, we apply round rotation to each lane of input at the beginning of Keccak, and then rotate them back after Keccak operations. Meanwhile, for ι operation in each round, we also need to rotate the constant number α bits. We sample the power traces using a LeCroy WaveRunner 640Zi oscilloscope, and evaluate the side-channel power leakages using correlation power analysis method.

We first sample 500 traces for the original implementation, and target at the first byte of first lane of θ_1 result. The correlation between the power consumption and $HW(\theta_1(0,[0:7]))$ is shown in Fig. 4.

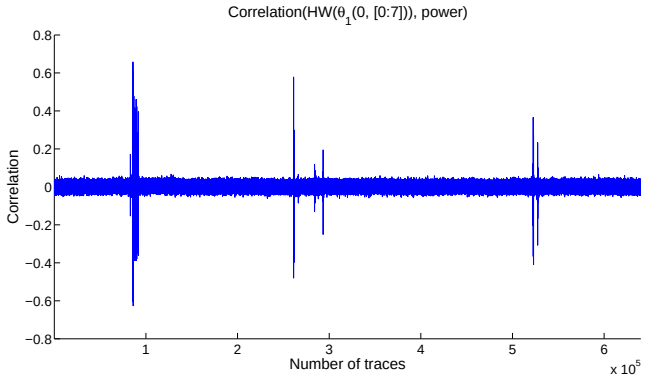

Fig. 4: CPA correlation for the original implementation

From Fig. 4. we can see the correlation between $HW(\theta_1(0,[0:7]))$ and power consumption is reaching 0.7 (-0.7), which is very large for cryptographic systems. This strong leakage can be used by attacker to retrieve key bits information. We present the attack results in Fig. 5(a).

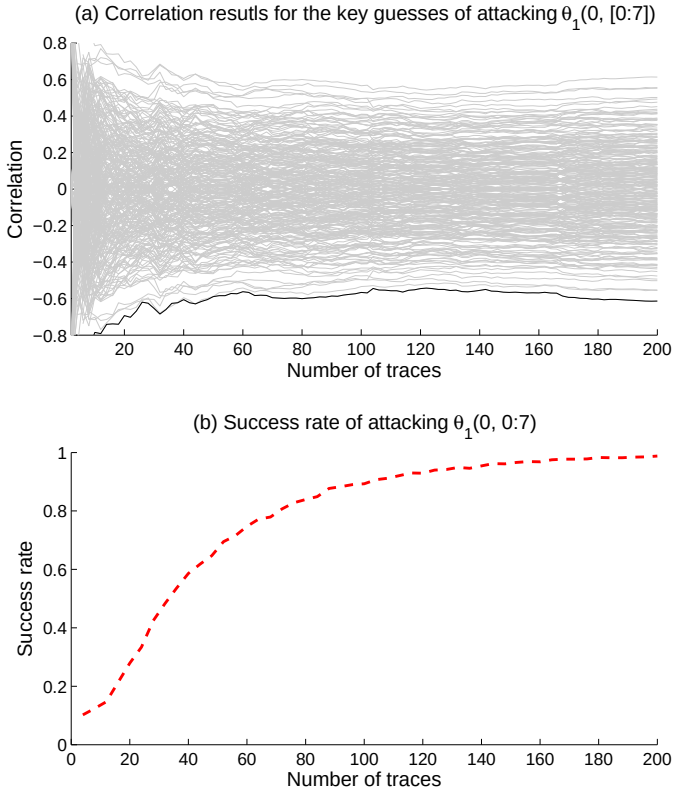

Fig. 5: CPA trace for all key guesses for original implementation

Fig. 5(a) shows that the right key will stand out of the key guesses very soon, and attackers only need about 200 traces to recover the key bits with success rate 1 according to the result shown in Fig. 5(b). Thus, without protection, the Keccak implemenation on smart card are vulnerable to side-channel attacks.

4.2 Leakage of the proposed implementation

For the proposed implementation, we sampled 5,000 traces and run CPA on them, we find that the correlation are hidden by random delay caused by the rotation operations. This is because for random α, the rotation time for each lane will be different for software system, and this difference will cause random delay in the system. Attacker cannot get key bits information if he has no knowledge of the random number α.

In this section, we assume that the attacker has full knowledge of the random number α. Which means the attack knows everything of the random delay caused by the random numbers. Then the correlation result is shown in Fig. 6(a).

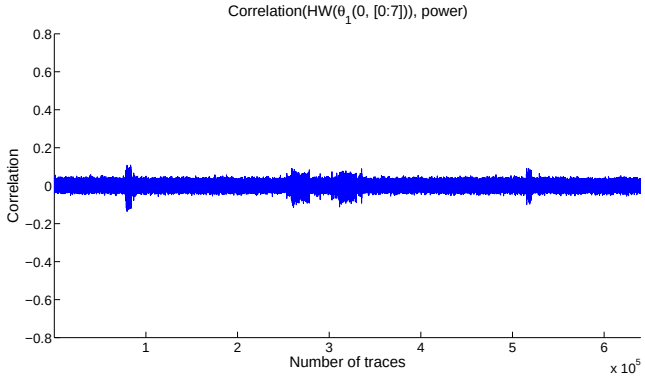

Fig. 6: CPA correlation for proposed implementation

It shows that with our countermeasure, the leakage decreases significantly, almost covered by noise for $5,000$ traces correlation. Thus it's anticipated that the protected implementation will be much more difficult to attack than the original version. The attack results are shown in Fig. 7.

In Fig. 7(a), it can be seen that the right key's correlation is not much higher than the incorrect keys as in original implementation. While the attacker will need almost 4,000 traces to recover the key bits for this implementation. Comparing with the original implementation, our proposed scheme will be much more difficult to attack. Taking the easy implementation and low resource overhead into consideration, this is a good countermeasure to reduce side-channel leakages for Keccak in smart card system.

5 conclusion

We present a method to reduce the leakages of Keccak on smart card platforms. This method can be effciently applied to both hardware and software implementations while very low resource overhead will be introduced. Real attacks results show that this scheme is easy to implement and it can significantly reduce the leakages.

References

[1] 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.

[2] 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.

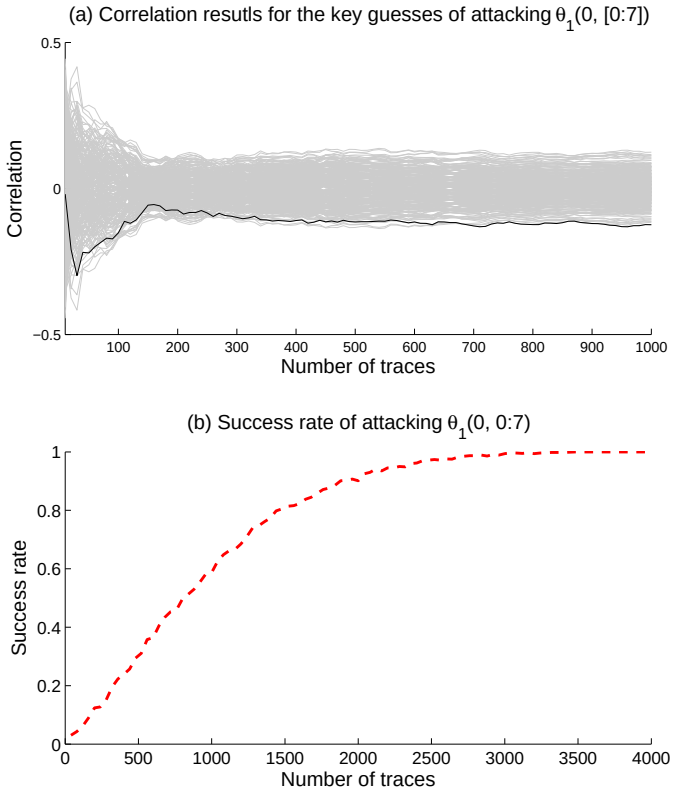

Fig. 7: CPA trace for all key guesses for proposed implementation

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

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

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

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