The analysis of the Keccak with the new method called parity

Ghanei yakhdan.mostafa, Noruzi, zynolabedin

Abstract:

The Keccak hash function is a chosen of the SHA-3 competition. This function has the appropriate structural and so far it has been resistant against collision attacks. In the last few years after efforts cryptonalysis, first, the collision in the second round and near collision in the third rounds, was found. and after adi Shamir and Et al succeeded find collision in fourth round and near collision in 5 round for keccak 224,256. In this paper we propose a method that it can used, for any desired text and finds, near collision, in the second round, regardless of any probability. Our attack technique using the find the text that have the same parity such original text in theta function. It is practical for all keccak varies and it finds near collision for second round in few minute. We have named this method " Parity Analyse”.

Introduction

The hash function is a one-way function that convert a string with arbitrary length to a string with constant length. A cryptographic hash function, denoted by h(x), must provide all of the following.

1- Compression: For any input x, the output y=h(x) is small. In practice, cryptographic hash functions produce a fixed size output, regardless of the length of the input, with typical output lengths being in the range of 128 to 512 bits.

2- Efficiency: It must be efficient to compute h(x) for any input x. Of course, the computational effort depends on the length of x, but the work should not grow too fast, as a function of the length of x.

3- Pri - image: It is computationally infeasible to invert the hash, that is, given y, we cannot find a value x. such that $h(x) = y$.

4- Collision resistance: Given x and $h(x)$, it is computationally infeasible to find any W, with $w \neq x$, such that $h(w) = h(x)$.

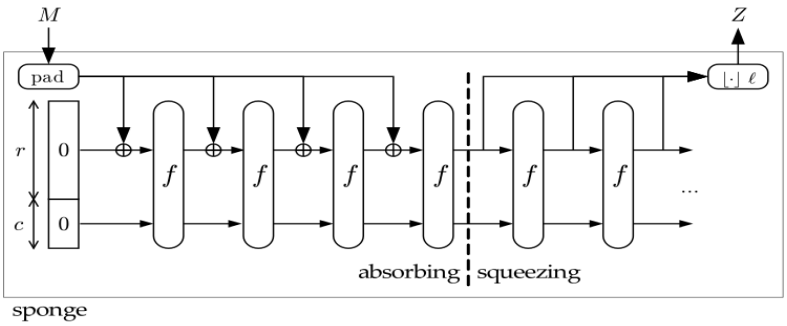
5- Second pri- image resistance: It is computationally infeasible to find any pair x and w, with x # w, such that h(x) = h(w).

Hash function used in wide application such as digital signature, authentication, and message authentication codes. So far, the numerous hash function been introduced. One of the most important these function is md4 that Was presented by rivest in 1991.This function, is base of for design functions such as MD5 and SHA. Given the importance security of hash functions, numerous papers published in analysis these functions. Theoretical and practical attacks on hash functions caused that NIST Institute in 2007 hold the competition for select the sha-3. Finally, in November of 2012, the winner of this competition was Keccak. The keccak hash function has used the sponge structure. On this function also, several attack has performed in reduced round. Including these attacks can noted to differential attack by Maria Nova and colleagues in 2011, which led to the collision in the third round and the Adi Shamir attack in 2011 using a combination of difference and algebraic methods which led to the collision in fourth round and near collision in the fifth round. in this paper, We are performing a new attack on it with use the internal function (θ).In this way, we Using one of the steps of θ function , where even parity is calculated for each message. We find the text that the output of it, is very similar to the output of the main text In this way, we find, collision in the second round regardless of any potential approximation. The advantages of the proposed method is that, its simplicity and ability to find near-collision for any text, regardless of the probability. In the second section we introduce the function Keccak, In the third section, the proposed method is described And the final section presents the conclusions of our discussion.

2-1 Introduction to the sponge structure

In cryptography, a sponge function or sponge construction is a class of algorithms with finite internal state that take an input bit stream of any length and produce an output bit stream of any desired length.

Sponge functions have both theoretical and practical uses. They can be used to model or implement most cryptographic primitives, including cryptographic hashes, message authentication codes, stream ciphers, block ciphers, and pseudo-random number generators. Figure 1 shows the overall structure of sponge function. A sponge function is built from of three components: a state memory, S, containing b bits, a function, f, of fixed length that permutes or transforms the state memory a padding function P. The state Memory is three-dimensional and it is in x, y, z axes. The state memory divided into two sections, R of size r bits and C of size c = b - r bits. The parameter r is called the bit rate and c is the



Figurer 1: sponge function

capacity. The padding function appends enough bits to the input string so that the length of the padded input is a whole multiple of the bit rate, r. The padded input can thus be broken into r-bit blocks. The sponge function is divided into two parts that called: 1- absorbed and 2- squeezing. The sponge function operates as follows:

a)Absorbed:

1-The state S is initialized to zero 2- The input string is padded 3- The first r-bit block of padded input is XORed with R. 4- S is replaced by f(S) 5- The next r-bit block of padded input (if any) is XORed with R. S is replaced by f(S),…. The process is repeated until all the blocks of the padded input string are used up. The sponge function output is now ready to be produced ("squeezed out") as follows:

b)Squeezed:

1- The R portion of the state memory is the first r bits of output. 2- If more output bits are desired, S is replaced by f(S). 3- The R portion of the state memory is the next r bits of output. The process is repeated until the desired number of output bits are produced.

Introducing keccak function

In Keccak function accepted as SHA-3 the state memory is 1600 bits, Capacity and bit rate to be selected accordance with Table 1. Each state bit is associated with 3 integer coordinates, a[x][y][z], where x and y are taken modulo 5, and z is taken modulo 64. The Keccak permutation consists of 24 rounds, which operate on the 1600 state bits. Each round of the permutation consists of five mappings. Keccak uses the following naming conventions, which are helpful in describing these mappings:

- A row is a set of 5 bits with constant y and z coordinates, i.e. a[*][y][z].

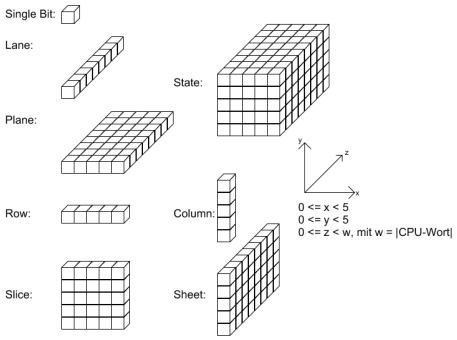


Figure 2: parts of sponge

- A column is a set of 5 bits with constant x and z coordinates, i.e. a[x][*][Z].

- A lane is a set of 64 bits with constant x and y coordinates, i.e. a[x][y][*].

- A slice is a set of 25 bits with constant z coordinate, i.e. a[*][*][z].

Figure 2 shows these parts and their names. At the continue we will be examined Keccak function in three parts, padding, mapped and outputs production.

a)Padding:

When the input string is applied to keccak function, at first, the padding, apply on it. In this process that is performed considering the bit rate Input bitstring expands to integer multiple of the bit rate. The padding runs According to the Equation 3.

$$P = M \,||\, 0X01 \,||\, 0X00 \,||\, \dots \,||\, 0X00 \qquad (1)$$

$$P = M \,||\, 0X01 \,||\, 0X00 \,||\, \dots \,||\, 0X00 \qquad P = P \; XOR \; (0X00 \,||\, \dots \,||\, 0X00 \,||\, 0X80) \qquad (2)$$

In the above equations M is the input message string P is extended string that the length of it is a multiple of the bit rate. The reason for using such method in padding is security considerations that described in [8].

b) Mapping f:

Table 1: the value of output keccak

Capacity(c)	Bit rate(r)	Output length(n)	Row
448	1152	224	1
512	1088	256	2
768	832	384	3
1024	576	512	4

The f mapping is the main part of keccak, consist of five function that have the task creation dispersion and diffusion. The five functions are given below, for each x, y, and z (where the state addition operations are over GF(2)):

1. θ is a linear map, which adds to each bit in a column, the parity of two other columns.

$$\theta: a[x][y][z] = a[x][y][z] + \sum_{y=0}^{4} a[x-1][y][z] + \sum_{y=0}^{4} a[x+1][y][z-1] \quad (3)$$

2. ρ rotates the bits within each lane by T(x,y), which is a predened constant for each lane.

$$\rho: a[x][y]=Rot(a[x][y], T(x,y) \qquad 0 \le x \le 4\ , 0 \le y \le 4 \quad (4)$$

3. π reorders the lanes.

$$\pi: a[x][y][z] = a[x'][y'][z] \text{ where } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}. \begin{pmatrix} x' \\ y' \end{pmatrix} \quad (5)$$

4. χ is the only non-linear mapping of Keccak, working on each of the 320 rows independently.

$$\chi: a[x][y][z] = a[x][y][z] + ((\sim a[x+1][y][z]) \wedge a[x+2][y][z]) \quad (6)$$

5. ι adds a round constant to the state.

$$\iota: a = a + RC[i] \quad (7)$$

Description of parity method:

In this section, we will introduce a new way to attack the Keccak. Our goal in this attack is that we find near collision in the second round. For attack, we examined the different parts of the map f. It is easy to show that keccak function, all the security condition has, when the input text is a bit smaller than r (It's only a block to be exercised). The analysis that we have done, show the weakness of keccak for the texts that have the same properties in low round. In the Keccak function, after padding, the extended input text, is applied to the mapping f. In the mapping f, θ is the first function that is applied to the input text then the function ρ, π applied and the output of this section apply to the function χ. Finally, the function ι applied to text and the first round ends. The π ,ρ,ι functions Are reversible and χ function is nonlinear. Therefore for attack, we will focus our attention on the θ function. θ function creates diffusion and confusion on the input text. This functions with a small number of gates(2 XOR Gates), cause diffusion in input bit sequence. Table 2 are given Signs and symbols used in this text.

Operations that θ function performs, we divided, operations that θ function performs, in three stages.

The first phase of this function, even parity in each column is computed .This calculation results in a two-dimensional array of 5 × 64 is stored. We show, this two-dimensional array with the matrix C[x].

C[x] = A[x][0]^A[x][1]^A[x][2]^A[x][3]^A[x][4] (8)

In the second stage, mixing operation be done on matrix C[x] and the result is set in a two-dimensional matrix D[x].

D[x] = C[(x-1)%5]^Rot(C[(x+1)%5],1) (9)

The bits mixed in ROT function in the following way:

def Rot(x,n) : return ((x>>(63))+(x<<n))%(1<<64) (10)

The result of this stage Xor with A[x] matrix for any x, y and the result is set on A[x]. This stage, the third stage is and in the relation (10) is shown.

```
for x in range(5):                                    (11)
    for y in range(5):

        A[x][y] = A[x][y]^D[x]
```

The operation completed and the A[x] matrix apply on ρ function.

Table 2: Symbols and description

description	application	symbol
Sum in modulo 2	A XOR b	A^B
Calculate A in module B	A mod B	A%B
Shift A to left (B bit)	Left shift	A<<B
Shift A to right (B bit)	Right shift	A>>B

We assume, the Input of θ function is, M string. All strings are in HEX. We show, output of the function with θ(M). Since our method is based on calculating the parity, for easier understanding, we choose a slice of the state memory and explain parity attack on it. The selected slice, is first slice of the state memory. the cause of use this slice is that calculations become easier .Suppose M as the main text of the first slice as Figure 3. The parity matrix for this slice is in equation (11).

C[x]: [0, 0, 1, 0, 0] (12)

If the second stage of function, performs on the c[x], The result is as follows.

D[x]: [0, 0, 0, 1, 0] (13)

The output of function for the M text is shown in Figure 4.

If we want, do the collision attack on the function we should, find the other text as M' that the output of it is such the M text(in this example the output should like the 3 figure)

We recall that the output of theta is obtained from sum D[x] with M. With little careful Can be seen that Always in the last row of the matrix θ(M), the matrix D [x] is. The reason for this is always the last row of the main matrix(M) is filled with zero in padding operation and when the D[x] matrix is XORed with it,

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0

Figure 3: a first slice of M

the result, D[x] is. Since the output of θ function from accumulating matrix D[x] and M text is obtained and D[x] with regard to θ(M), is specified, if we want find the collision in the third stage of θ The equation of (13) is applied. in (13) equation can be seen that the obtained text is the same as the main text and This means that, It's not possible find collisions in the θ function. So, our aim in this paper find the near collision in θ function is.

0	0	0	1	0
0	0	0	1	0
0	0	0	1	0
0	0	1	1	0
0	0	0	1	0

Figure 4: θ(M)

Main idea:

The main idea in the parity attack design is that we choose the texts which have the same parity. We select keccak (224) for explain our design, in this function and in any slice there are 200 different texts that the parity of them are same. In keccak per slice The last 7 bits are always zero (Reminded: Memory state consists of bit rates and capacity and Capacity will always be from zero filled. In keccak-224, the capacity in per slice, include the last 7 bits). For attack, we'll select the text that the maximum length of it is r bits .This text called The main text and will be represented by M, obtain parity of M; we are looking for the texts that they have the same parity, now. As mentioned, there are many of these texts, with experiment was determined that the texts that only in first slice is different with main text and make up the same parity, To find near-collisions are more suitable. So we should choose M' among the 200 texts, so that, firstly, it be have the same parity and secondly $\theta(M')$ be have most similar to $\theta(M)$.

With this method, we were able to reach the near collision in the 2 round. An example of the results is given below. In this trial for the main text, the first slice is shown in figure 3. For the slice in figure 3, the parity matrix is C[x]=[0,0,1,0,0]. The chosen text, is intended such that only difference is, in the first slice

0	1	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

Figure 5: a first slice of chosen text

with main text and generate the same parity matrix. The first slice of the chosen text is shown in Figure 5. As can be seen in the output, in the found text, output in the second round, only is different in six positions with the main output. Adi Shamir and colleagues using the method that combines differential and algebraic techniques is found near-collision in the fifth round, in a few days. But the probability of finding two texts that are very similar output of more than 2^{-30}

M (main text)

```
Main text
    ['0x1a', '0x0', '0x0', '0x0', '0x0']
     ['0x0', '0x0', '0x0', '0x0', '0x0']
     ['0x0', '0x0', '0x0', '0x0', '0x0']
     ['0x0', '0x0', '0x8000000000000000', '0x0', '0x0']
     ['0x0', '0x0', '0x0', '0x0', '0x0']
The parity in tetha step:
c[x]: [26, 0, 9223372036854775808, 0, 0]
the output of round 1:
     ['0x1b', '0x1b00000100000', '0xd0000', '0x10001a', '0x1b000000d0000']
     ['0x8000000', '0x3600003400000', '0x0', '0x3600008000000', '0x3400000']
     ['0x1000036', '0x3400', '0x1000000', '0x3436', '0x0']
     ['0x1a0006c00', '0x0', '0x80000000006c00', '0x1a0004000', '0x80000000000000']
     ['0x1a0000000000', '0x40000000000000', '0x1a000000006c', '0x0', '0x4000000000006c']
the output of round 1:

     ['0x4e4900205ede8e7', '0xeb5b62f5a2e03ae9', '0x13e2cc0209869a83', '0x4070f2e592694808',]
```

. In all of the analyses conducted on the Keccak, near collision in the next round of analysis, disappears. On our design this defect in the third round disappears.

Finded text

```
Finded text:
     ['0x1a', '0x8000000000000000', '0x0', '0x0', '0x0']
      ['0x0', '0x8000000000000000', '0x0', '0x0', '0x0']
```

['0x0', '0x0', '0x0', '0x0', '0x0']
['0x0', '0x0', '0x8000000000000000', '0x0', '0x0']
['0x0', '0x0', '0x0', '0x0', '0x0']
The parity in tetha step:
c[x]: [26, 0, 9223372036854775808, 0, 0]
the output of round 1:
['0x1b', '0x1b80000100000', '0xd0000', '0x10001a', '0x1b800000d0000']
['0x8000000', '0x3600003400000', '0x0', '0x3600008000000', '0x3400000']
['0x1000037', '0x3400', '0x1000000', '0x3437', '0x0']
['0x1a0006c00', '0x0', '0x80000000006c00', '0x1a0004000', '0x80000000000000']
['0x1a0000000000', '0x40000000000000', '0x1a000000006c', '0x0', '0x4000000000006c']
the output of round 2:
['0x4e4980205ede8e7', '0xeb5b62f5a2e03ae9', '0x13e2dc0209c65a83', '0x4070f2e5926908 0a',]

Conclusions

Hash function have important role in secure communications. Keccak hash function selected in November 2012 as SHA-3. It was shown that in keccak find collision for the text that length of it is less than r bit is almost impossible and hence, this function for such texts is a high security. So far, several attacks often with differential methods have been implemented on this function in reduced round. In this paper we offer the method that leading to near collision in the 2 round Without any approximations, and probability. To find the collision, with the proposed method can execute this method on keccak for the text that the length of it is bigger than r bit and work on this topic is continuing.

References :

[1] D.STINSON, *Cryptography Theory and Practice*, CRC, 2006

[2] Rivest, R. L, *The MD4 Message Digest Algorithm*, CRYPTO 1990. LNCS, vol. 537, 303–311, Springer, Heidelberg,1991.

[3] Chabaud, *Differential Collision in SHA-0*, Advancesin Cryptology CRYPTO 98, vol. 1462 of Lecture Notesin Computer Science, pages 56–71. Springer-Verlag, 1998.

[4] Pramstaller, Rechberger, , Rijmen, *Exploiting Coding Theory for Collision Attacks on SHA-1*, Cryptography and Coding 2005, LNCS 3796, 78-95, Springer- Verlag, 2005.

[5] http://www.nist.gov/itl/csd/sha-100212.cfm. Retrieved 2012-10-02.

[6] Adi Shamir, Itai Dinur, "Practical Analysis of Reduced-Round Keccak",spriger,2011

[7] Maria Naya-Plasencia, Andrea Rock "New attacks on Keccak-224 and Keccak-256",springer,2011

[8] http://keccak.noekeon.org/Keccak-reference-3.0.pdf