Keccak-f[25] and ElGamal

Chenglong Ma
s3629862

Abstract

This report briefly describes how Keccak-f[25] and ElGamal work.

1 Keccak

Keccak is an implementation of SHA-3 Cryptographic Hash Algorithm [1]. In this report, we discuss a lightweight version Keccak where we set state $b = 25 = 5 \times 5 \times 2^l$ in its *Sponge* stage. I.e. $l = 0$ and thus there is only **one** *slice* (5×5 bits) and 12 rounds as $n\_rounds = 12 + 2l$. Each round function consists of 5 steps to process the state [1]: $\{\theta \rightarrow \rho \rightarrow \pi \rightarrow \chi \rightarrow \iota\}$. The output of former step would be treated as the input of next, likewise, the output of ι step will be feed into θ step of next round. We implement 4 of them in this report (ρ step excluded). We will discuss these steps in detail in the following sections.

1.1 θ Step

The θ Step is defined as below:

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

where $A[x,y]$ indicates the *lane* in *column* x and *row* y; thus $C[x]$ is a bitwise XOR summation between all rows of column x. Then, $rot(C[x+1],1)$ rotates the bits within the lane $C[x]$. Particularly, as there is only slice in our implementation, the *rot* function makes no changes so that $rot(C[x+1],1) =$

$C[x+1]$. Thereafter, $D[x]$ combines $C[x-1]$ and $C[x+1]$ by XOR summation. Finally, the output of each single value in $A[]$ of this step obtains from the original $A[x, y]$ XOR $D[x]$.

An example of this step is shown below (Fig. 1), including input and all outputs of intermediate steps.

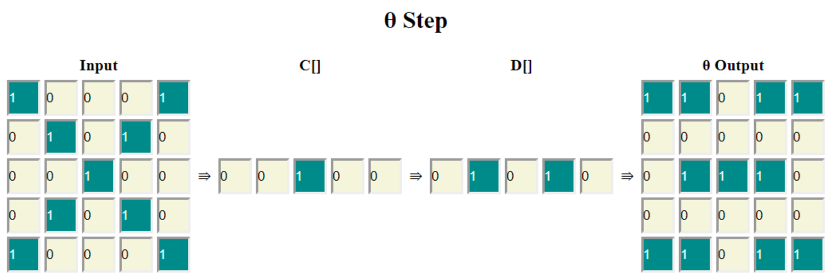

Figure 1: An Example of θ Step

The origin of coordinates in each matrix (i.e. $A[0, 0]$) locates at the left bottom. Take $A[2, 3]$ as an example, the input is 0 and the output of intermediate steps are:

$$\begin{aligned} C[2] &= A[2,0] \oplus A[2,1] \oplus A[2,2] \oplus A[2,3] \oplus A[2,4] \\ &= 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \\ &= 1 \\ D[2] &= C[1] \oplus C[3] \\ &= 0 \oplus 0 \\ &= 0 \\ A'[2,3] &= A[2,3] \oplus D[2] \\ &= 0 \oplus 0 \\ &= 0 \end{aligned} \tag{2}$$

1.2 π Step

Differ from the other 3 steps, which are all *substitution* functions, π step is a *permutation* function [2]. The output of θ step will put into π step as initial input. The π step is defined as:

$$A'[y, 2x + 3y] = rot(A[x, y], r[x, y]), x, y \in [0, 4], \tag{3}$$

likewise as θ step, x and y represent the index of *column* and *row* respectively (the same below). *rot* function would also make no effects.

Continued from the example of θ step 2, the π step is illustrated below (Fig. 2):

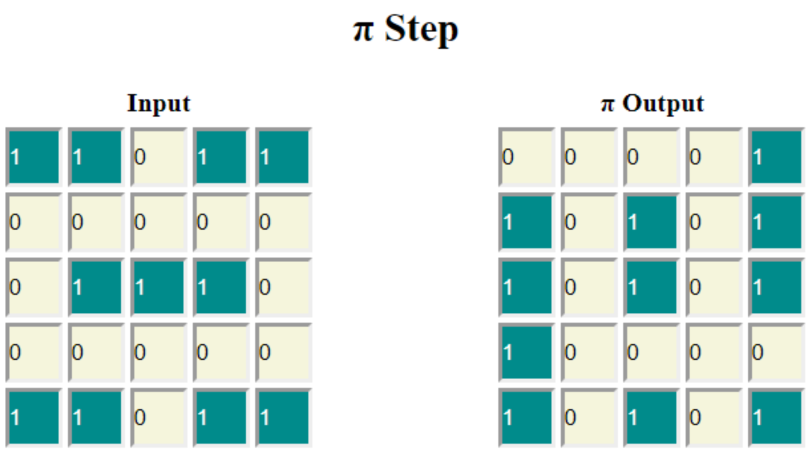

Figure 2: An Example of π Step

As shown above, the value of $A[2,3]$ would be assigned to $A'[3,(2\times 2+3\times 3) \mod 5] = A'[3,3]$, i.e., $A'[3,3] = A[2,3] = 0$.

1.3 χ Step

The output of $A'[x,y]$ in χ step is determined by $\bar{A}[x+1,y]$ and $A[x+2,y]$, the complete definition is:

$$A'[x,y] = A[x,y] \oplus (\bar{A}[x+1,y] \wedge A[x+2,y]), x,y \in [0,4], \tag{4}$$

where $\bar{A}[x+1,y]$ is the *NOT* operation of $A[x+1,y]$. This step can be illustrated as Fig. 3:

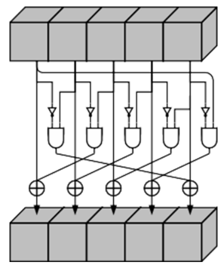

Figure 3: The illustration of χ step

Therefore, $A[2,3]$ would be substituted as:

$$\begin{aligned} A'[2,3] &= A[2,3] \oplus (\bar{A}[3,3] \wedge A[4,3]) \\ &= 1 \oplus (1 \wedge 1) \\ &= 0, \end{aligned} \tag{5}$$

The complete intermediate outputs are:

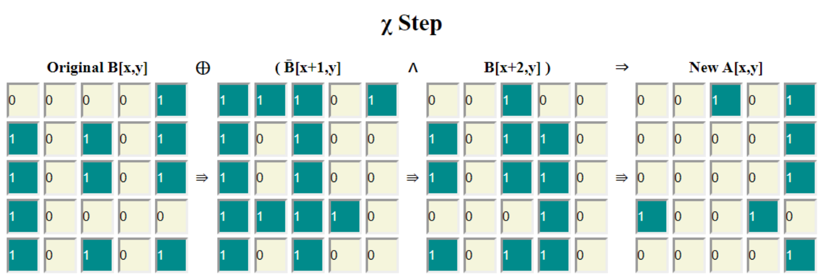

Figure 4: An Example of χ Step

1.4 ι Step

The ι step is the final step of one round in Keccak. It introduces a set of *round constants* and combines $A[0,0]$ with one of them (pick a different value for each round) by XOR summation [2], as shown below (Equ. 6). The rest of matrix would remain their values unchanged.

$$A'[0,0] = A[0,0] \oplus RC[i_r], \tag{6}$$

where $RC[]$ is the rounds constants array and i_r equals the current round index. It should be noted that each slice in the state of Keccak should XOR

the $slice^{th}$ bit in $RC[i_r]$ from the leftmost (e.g., in our implementation, only the *leftmost* bit of $RC[i_r]$ is used). Based on this, we get our final output of the first round:

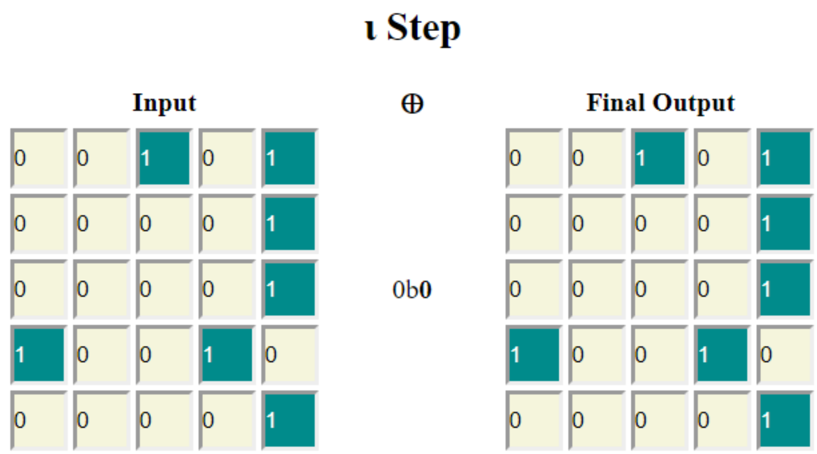

Figure 5: An Example of ι Step

1.5 Rounds

As mentioned at the beginning of this section, there are 12 rounds in our implementation. For each round, we feed the final output of the ι step into the θ step of next round and switch the round constants to next index. The last round of Keccak-$f[25]$ is shown below:

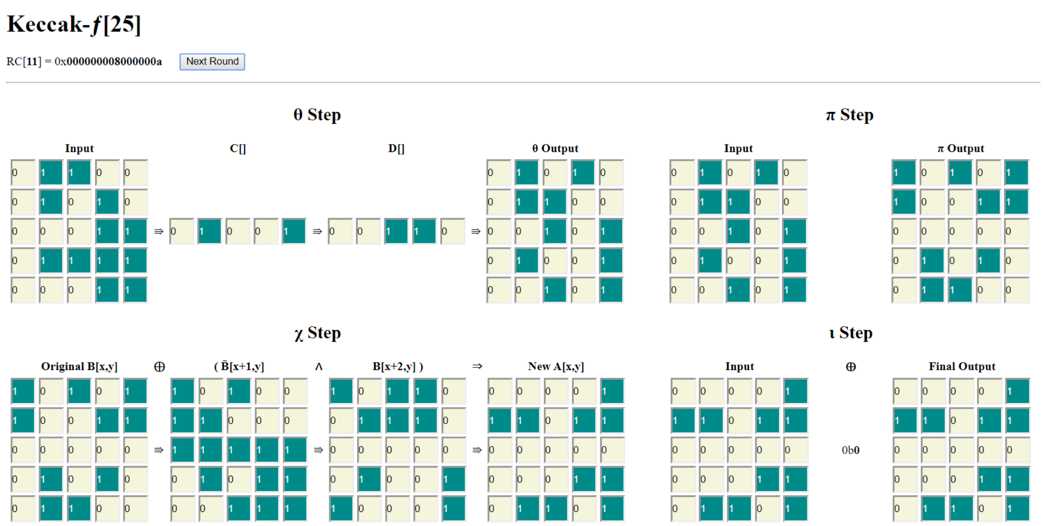

Figure 6: Final output of the 12^{th} round

2 Elgamal

The Elgamal encryption scheme is an asymmetric key encryption algorithm which consists of three phases (i.e. key generation, encryption, decryption) [3]. It is based on the *Diffie–Hellman* key exchange and defined over a cyclic group G. Therefore, its security relies heavily on Discrete-log based building blocks [3].

2.1 Configuration

The Elgamal algorithm is defined on a cyclic group so that it should be safe to select a prime p to generate a prime order group Z_p^*. Then we need to choose a prime q to define a unique subgroup G_q and $g \in Z_p^*$ is one of its generators where $p = \gamma q + 1$, γ is a specific integer.

2.2 Key Generation

To generate the *public* key of Elgamal, we firstly pick a random x as the *private* key where $x \in (0, q)$. Then we get the public key $\{p, g, y\}$:

$$\{p, g, y\} = \{p, g, g^x \mod p\}, \tag{7}$$

now we can pass this public key triplets to anyone who wants to send his/ her encrypted message to us. The following is an example of key generation phase:

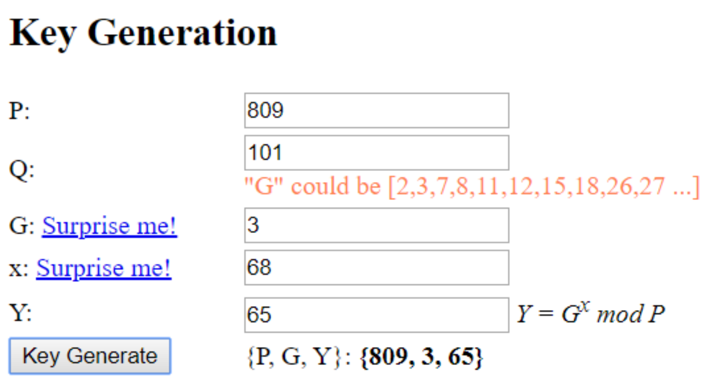

Figure 7: An Example of Elgamal Key Generation

It should be noted that y can be calculated by *Fast Exponentiation Methods* [4] whose pseudo-code is shown in Algo 1.

Algorithm 1 FastExponentiation(x, exp, n)

Input: An integer (x), an exponent (exp) and a modulus (n)
Output: res

if n == 1 **then**
  return 0
end if
$res \leftarrow 1$
for $i \leftarrow 0$ **to** $(exp - 1)$ **do**
  $res \leftarrow (res \times x) \mod n$
end for
return res

2.3 Encryption

Assume that we encrypt a message $m \in (0, p)$ using the above configured ElGamal. First, we need to choose an integer $k \in (0, q)$ randomly. Then, we get the cipher pair $\{c_1, c_2\}$:

$$\begin{cases} c_1 & = g^k \mod p \\ c_2 & = y^k \cdot m \mod p \end{cases} \tag{8}$$

Based on the example shown in Fig. 7 and setting $m = 100, k = 89$, we can easily get the cipher is

$$\begin{aligned} \{c_1, c_2\} &= \left\{g^k \mod p, y^k \cdot m \mod p\right\} \\ &= \left\{100^89 \mod 809, 65^89 \cdot 100 \mod 809\right\} \\ &= \{345, 517\}, \end{aligned} \tag{9}$$

the result is shown in Fig. 8.

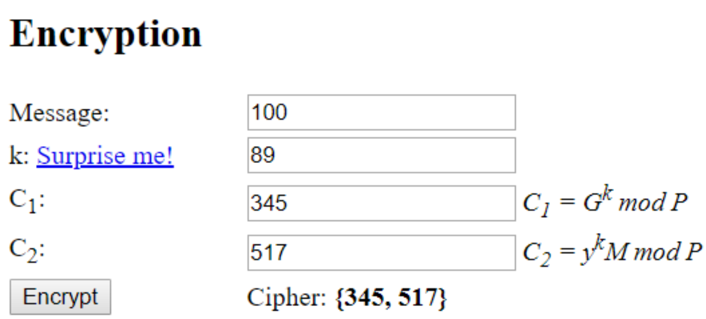

Figure 8: An Example of Elgamal Encryption

2.4 Decryption

With the private key x, we can decode the cipher pair $\{c_1, c_2\}$ to get the plaintext m:

$$m = \frac{C_2}{C_1^x} \mod p = C_2 \cdot C_1^{-x} \mod p \tag{10}$$

where $(C_1^{-x} \mod p)$ is the inverse of $(C_1^x \mod p)$ that $(C_1^{-x} \cdot C_1^x \equiv 1 \mod p)$. There are multiple methods to solve this equation. We implement an *Extended Euclidean algorithm* to get the inverse, see Algo 2 [1]:

Algorithm 2 inverse(a, n)

Input: An integer (a), a modulus (n)
Output: t

```
t ← 0; newt ← 1;
r ← n; newr ← a;
while newr ≠ 0 do
    quotient ← r ÷ newr
    (t, newt) ← (newt, t − quotient × newt)
    (r, newr) ← (newr, r − quotient × newr)
end while
if r ≥ 1 then
    return "a is not invertible."
end if
if t ≤ 0 then
    t ← t + n
end if
return t
```

Continued with the above example, we can get:

$$\begin{aligned} m &= \frac{C_2}{C_1^x} \mod p = C_2 \cdot C_1^{-x} \mod p \\ &= (517 \times 720) \mod 809 \\ &= 100 \end{aligned} \tag{11}$$

Finally, we complete all the tree phases of Elgamal:

Figure 9: An Example of Elgamal Decryption

[1] Refer to https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

2.5 Homomorphic Property of ElGamal

Homomorphic encryption allows users to perform calculations on the *ciphers* and bring the calculated encrypted results into correspondence with the results of operated on the *plaintexts* [5]. It enables the black-box operations on cipher messages and improves the security of the information transfer process. Elgamal is an implementation of homomorphic encryption. The multiplicative operation between two ciphers of m_1 and m_2 is as follows:

The encryptions of m_1 and m_2 are:

$$\begin{aligned} C_1 &= (c_1, c_2) = (g^k \mod p, y^k \cdot m_1 \mod p) \\ C_2 &= (c_1', c_2') = (g^{k'} \mod p, y^{k'} \cdot m_2 \mod p) \end{aligned} \tag{12}$$

The product of C_1 and C_2 is:

$$\begin{aligned} C_1 \cdot C_2 &= (c_1, c_2) \cdot (c_1', c_2') \\ &= (g^{k+k'} \mod p, y^{k+k'} \cdot (m_1 \cdot m_2) \mod p) \end{aligned} \tag{13}$$

Then, we get decryption of $C_1 \cdot C_2$ and verify if it equals the product of m_1 and m_2:

$$\begin{aligned} &D[(g^{k+k'}, y^{k+k'} \cdot (m_1 \cdot m_2))] = \frac{(y^{k+k'} \cdot (m_1 \cdot m_2))}{(g^{x \cdot (k+k')})} \mod p \\ &= m_1 \cdot m_2 \mod p \end{aligned} \tag{14}$$

Based on the deduction of Equ. 14, we have verified the homomorphic property of ElGamal, which is also demonstrated in our implementation.

We randomly choose 5 integer messages $[m_1, m_2, \cdots, m_5]$ and 5 random integer k $[k_1, k_2, \cdots, k_5]$, then encrypt them respectively. Finally, we calculate the products of each pair and the output has proved the homomorphic property of ElGamal as well, shown as Fig. 10.

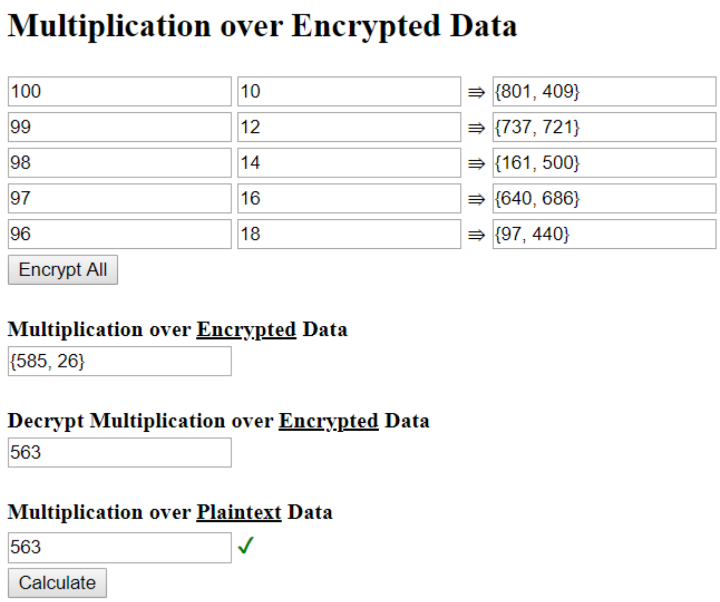

Figure 10: Homomorphic Property of ElGamal

References

[1] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak," in *Advances in Cryptology – EUROCRYPT 2013*, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds. Springer Berlin Heidelberg, 2013, pp. 313–314.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Pearson Upper Saddle River, 2017, 00382.

[3] Y. Tsiounis and M. Yung, "On the security of ElGamal based encryption," in *Public Key Cryptography*, ser. Lecture Notes in Computer Science, H. Imai and Y. Zheng, Eds. Springer Berlin Heidelberg, 1998, pp. 117–134, 00402.

[4] D. M. Gordon, "A Survey of Fast Exponentiation Methods," *Journal of Algorithms*, vol. 27, no. 1, pp. 129–146, Apr. 1998, 00676.

[5] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices." in *Stoc*, vol. 9, no. 2009, 2009, pp. 169–178, 04140.