Project:
Hash Functions

Joint Authors:
Gu Yichen 5143709162
Liu Yihao 515370910207
Peng Junda 5133
Wang Dayi 5133
(In Alphabetic Order)

Instructor:
Prof. Charlemagne Manuel

Course:
VE475
Introduction to Cryptography

Institute:
University of Michigan - Shanghai Jiao Tong University
Joint Institute

2017-August-8th

Abstract

The goal of this project is to perform some personal study in order to acquire the extra knowledge in cryptography while also improving major skills such as writing, collaboration and public presentation. In this project, we intend to have a preliminary understanding of a method of cryptography: Hash Functions.

In this project, we first . Then in the second chapter. The third chapter is mainly about. In the fourth chapter.

Cryptography is quite a complex and esoteric subject and there are so many mysteries in the world. We just do a simple and shallow survey on the aspect of Hash Functions. There still remains a large number of interesting and profound problems to be solved. The study of cryptography is still pending.

Contents

I Introduction

II KECCAK-p Permutations

The KECCAK-p permutations are specified with two parameters:

- One is the fixed length of the strings that are permuted, called the *width* of the permutation, denoted by b.
- The other is the number of iterations of an internal transformation, called a *round*, denoted by n_r.

The KECCAK-p permutation with n_r rounds and width b is denoted by KECCAK-$p[b, n_r]$ and the permutation is defined for any b in $\{25, 50, 100, 200, 400, 800, 1600\}$ and any positive integer n_r.

A round of a KECCAK-p permutation, denoted by Rnd, consists of a sequence of five transformations, which are called the *step mappings*. The permutation is specified in terms of an array of values for b bits that is repeatedly updated, called the *state*. The state is initially set to the input values of the permutation.

1 State

The state for the KECCAK-$p[b, n_r]$ permutation is comprised of b bits. There are two other quantities: $w = b/25$ and $l = \log_2(b/25)$. The seven possible values for these variables that are defined for the KECCAK-p permutations are given in the following table.

Table II.1: KECCAK-p permutation widths and related quantities

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6

It is convenient to represent the input and output states of the permutation as b-bit strings, and to represent the input and output states of the step mappings as 5-by-5-by-w arrays of bits. If S denotes a string that represents the state, then its bits are indexed from 0 to $b1$, so that

$$S = S[0] \,||\, S[1] \,||\, \cdots \,||\, S[b-2] \,||\, S[b-1]$$

If A denotes a 5-by-5-by-w array of bits that represents the state, then its indices are the integer triples (x, y, z) for which $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$. The bit corresponding to (x, y, z) is denoted by $A[x, y, z]$. A *state array* is a representation of the state by a three-dimensional array that is indexed in this manner.

1.1 State Array

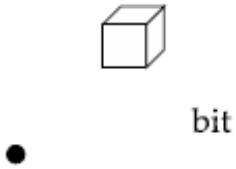

Figure II.1: Original Part of State Array

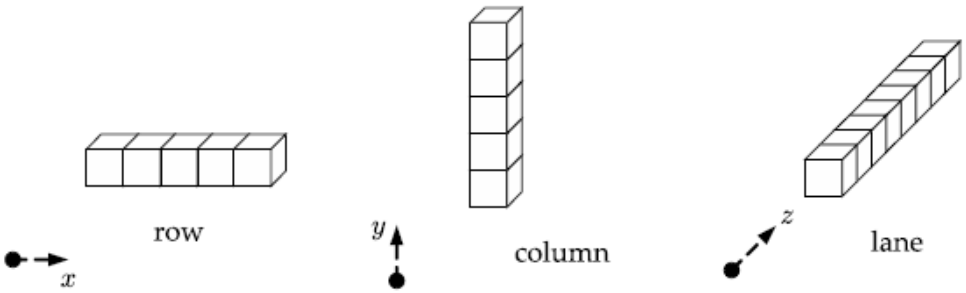

Figure II.2: 1-D Parts of State Array

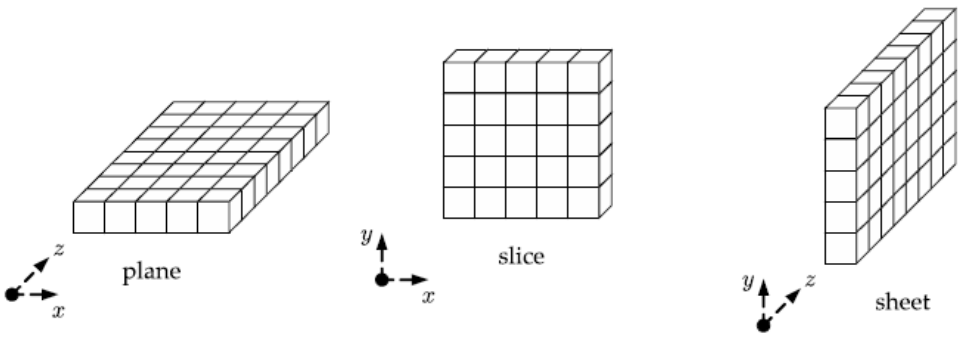

Figure II.3: 2-D Parts of State Array

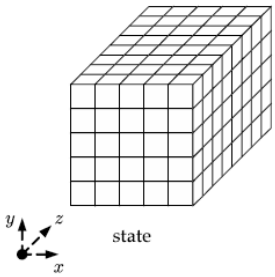

Figure II.4: 3-D Part of State Array

For $b = 200 (w = b/25 = 8)$, the state array for a KECCAK-p permutation and its lower-dimensional sub-arrays are illustrated in figures above. For a state, the single-dimensional sub-arrays are called *rows*, *columns* and *lanes*. The two-dimensional sub-arrays are called *sheets*, *planes* and *slices*. The original sub-array is called *bit*.

1.2 Converting Strings to State Arrays

S is a string of b bits that represents the state for the KECCAK-$p[b, n_r]$ permutation. The corresponding state array A is defined as follows:

$$\text{For all triples (x, y, z) such that } 0 \leq \text{x} < 5,\ 0 \leq \text{y} < 5 \text{ and } 0 \leq \text{z} < \text{w}$$

$$A[x, y, z] = S[w(5y + x) + z]$$

```
void sha3_convert_string_to_state_array(sha3_string S,
    sha3_state_array A, sha3_bits b, size_t r)
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            for (int k = 0; k < SHA3_WIDTH[b]; k++)
            {
                size_t index = SHA3_WIDTH[b] * (5 * j + i) + k;
                A[i][j][k] = index < r ? S[index] : (uint8_t) 0;
            }
        }
    }
}
```

1.3 Converting State Arrays to Strings

A is a state array and the corresponding string representation S can be constructed from the lanes and planes of A, as follows:

For each pair of integers (i, j) such that $0 \leq i < 5$ and $0 \leq j < 5$:

$$Lane(i, j) = A[i, j, 0] \mid\mid A[i, j, 1] \mid\mid \cdots \mid\mid A[i, j, w - 2] \mid\mid A[i, j, w - 1]$$

For each integer j such that $0 \leq j < 5$:

$$Plane(j) = Lane(0, j) \mid\mid Lane(1, j) \mid\mid Lane(2, j) \mid\mid Lane(3, j) \mid\mid Lane(4, j)$$

Then we can have the corresponding string representation S:

$$S = Plane(0) \mid\mid Plane(1) \mid\mid Plane(2) \mid\mid Plane(3) \mid\mid Plane(4)$$

```
void sha3_convert_state_array_to_string(sha3_state_array A,
 ↪  sha3_string S, sha3_bits b)
{
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            for (int k = 0; k < SHA3_WIDTH[b]; k++)
            {
                S[SHA3_WIDTH[b] * (5 * j + i) + k] = A[i][j][k];
            }
        }
    }
}
```

1.4 Labeling Convention for the State Array

In the diagrams of the state that accompany the specifications of the step mappings, the lane that corresponds to the coordinates $(x, y) = (0, 0)$ is depicted at the center of the slices. The complete labeling of the (x, y, z) coordinates for those diagrams is shown in the following figure.

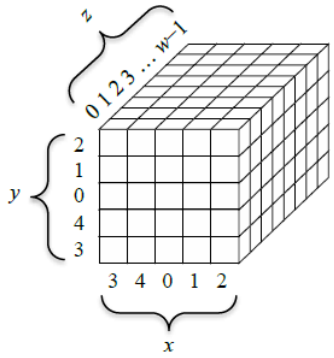

Figure II.5: The (x, y, z) coordinates for the diagrams of the step mappings

2 Step Mappings

The five step mappings that comprise a round of KECCAK-$p[b, n_r]$ are denoted by θ, ρ, π, χ and $\imath$. The algorithm for each step mapping takes a state array A as the input and returns an updated state array A' as the output. The size of the state is a parameter that is omitted from the notation, because b is always specified when the step mappings are invoked.

2.1 Specification of θ

Input:
state array A
Output:
state array A'
function $\theta(A)$
for all pairs (x, z) such that $0 \leq x < 5$ and $0 \leq z < w$ **do**
$C[x, z] = A[x, 0, z] \oplus A[x, 1, z] \oplus A[x, 2, z] \oplus A[x, 3, z] \oplus A[x, 4, z]$
$D[x, z] = C[(x - 1) \bmod 5, z] \oplus C[(x + 1) \bmod 5, (z - 1) \bmod w]$
end for
for all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$ **do**
$A'[x, y, z] = A[x, y, z] \oplus D[x, z]$
end for
return state array A'
end function

```
1  inline void sha3_step_theta(sha3_state_array A, sha3_bits b)
2  {
3      for (int i = 0; i < 5; i++)
4      {
5          for (int k = 0; k < SHA3_WIDTH[b]; k++)
6          {
7              sha3_temp[i][k] = A[i][0][k];
8              for (int j = 1; j < 5; j++)
9              {
10                 sha3_temp[i][k] ^= A[i][j][k];
11             }
12         }
13     }
14     for (int i = 0; i < 5; i++)
15     {
16         for (int k = 0; k < SHA3_WIDTH[b]; k++)
17         {
18             uint8_t D = sha3_temp[(i + 4) % 5][k] ^sha3_temp[(i + 1)
               ↪  % 5][(k - 1 + SHA3_WIDTH[b]) % SHA3_WIDTH[b]];
19             for (int j = 0; j < 5; j++)
20             {
21                 A[i][j][k] ^= D;
22             }
23         }
24     }
25 }
```

2.2 Specification of ρ

Input:
state array A
Output:
state array $A^{'}$
function $\rho(A)$
for all z such that $0 \leq z < w$ **do**
$A^{'}[0,0,z] = A[0,0,z]$
end for
$(x,y) = (1,0)$
for $t = 0$ to 23 **do**
for all z such that $0 \leq z < w$ **do**

$$A^{'}[x,y,z] = A[x,y,(z - \frac{(t+1)(t+2)}{2}) \bmod w]$$

end for
$(x,y) = (y,(2x+3y) \bmod 5)$
end for
return state array $A^{'}$
end function

```
1  inline void sha3_step_rho(sha3_state_array A, sha3_bits b)
2  {
3      uint8_t x = 1, y = 0;
4      uint8_t *B = sha3_temp[0];
5      for (int t = 0; t < 24; t++)
6      {
7          uint8_t temp = ((-(t + 1) * (t + 2) / 2) % SHA3_WIDTH[b] +
           ↪  SHA3_WIDTH[b]) % SHA3_WIDTH[b];
8          memcpy(B, A[x][y] + SHA3_WIDTH[b] - temp, sizeof(uint8_t) *
           ↪  temp);
9          memcpy(A[x][y] + temp, A[x][y], sizeof(uint8_t) *
           ↪  (SHA3_WIDTH[b] - temp));
10         memcpy(A[x][y], B, sizeof(uint8_t) * temp);
11         temp = (2 * x + 3 * y) % 5;
12         x = y;
13         y = temp;
14     }
15 }
```

2.3 Specification of π

Input:
state array A
Output:
state array A'
function $\pi(A)$
for all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$ **do**
$A'[x, y, z] = A[(x + 3y) \bmod 5, x, z]$
end for
return state array A'
end function

```
1  inline void sha3_step_pi(sha3_state_array A, sha3_bits b)
2  {
3      for (int k = 0; k < SHA3_WIDTH[b]; k++)
4      {
5          for (int i = 0; i < 5; i++)
6          {
7              for (int j = 0; j < 5; j++)
8              {
9                  sha3_temp[i][j] = A[(i + 3 * j) % 5][i][k];
10             }
11         }
12         for (int i = 0; i < 5; i++)
13         {
14             for (int j = 0; j < 5; j++)
15             {
16                 A[i][j][k] = sha3_temp[i][j];
17             }
18         }
19     }
20 }
```

2.4 Specification of χ

Input:
state array A
Output:
state array A'
function $\chi(A)$
for all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$ **do**
$A'[x, y, z] = A[x, y, z] \oplus ((A[(x+1) \bmod 5, x, z] \oplus 1) \cdot A[(x+2) \bmod 5, y, z])$
end for
return state array A'
end function

```
1  inline void sha3_step_chi(sha3_state_array A, sha3_bits b)
2  {
3      for (int k = 0; k < SHA3_WIDTH[b]; k++)
4      {
5          for (int j = 0; j < 5; j++)
6          {
7              for (int i = 0; i < 5; i++)
8              {
9                  sha3_temp[0][i] = A[i][j][k] ^ ((A[(i + 1) % 5][j][k]
                       ↪  ^ (uint8_t) 1) & A[(i + 2) % 5][j][k]);
10             }
11             for (int i = 0; i < 5; i++)
12             {
13                 A[i][j][k] = sha3_temp[0][i];
14             }
15         }
16     }
17 }
```

2.5 Specification of ι

Input:
integer t
Output:
bit $rc(t)$
function $rc(t)$
if $t \bmod 255 = 0$ **then**
return 1
end if
$R = 10000000$
for $i = 1$ to $t \bmod 255$ **do**
$R = 0 \,||\, R$
$R[0] = R[0] \oplus R[8]$
$R[4] = R[4] \oplus R[8]$
$R[5] = R[5] \oplus R[8]$
$R[6] = R[6] \oplus R[8]$
$R = \text{Trunc}_8[R]$
end for
return $R[0]$
end function

```
void sha3_generate_rc()
{
    for (int i = 0; i < 255; i++)
    {
        uint8_t R[9] = {1, 0, 0, 0, 0, 0, 0, 0, 0};
        for (int j = 1; j <= i; j++)
        {
            for (int k = 8; k > 0; k--) R[k] = R[k - 1];
            R[0] = 0;
            R[0] ^= R[8];
            R[4] ^= R[8];
            R[5] ^= R[8];
            R[6] ^= R[8];
        }
        printf("%d,", R[0]);
        if (i % 16 == 15)printf("\n");
    }
    printf("\n");
}
```

Input:
state array A round index i_r
Output:
state array A'
function $\chi(A)$
for all triples (x, y, z) such that $0 \leq x < 5$, $0 \leq y < 5$ and $0 \leq z < w$ **do**
$A'[x, y, z] = A[x, y, z]$
end for
$RC = 0^w$
for $j = 0$ to l **do**
$RC[2^j - 1] = rc(j + 7i_r)$
end for
for all z such that $0 \leq z < w$ **do**
$A'[0, 0, z] = A'[0, 0, z] \oplus RC[z]$
end for
return state array A'
end function

```
inline void sha3_step_iota(sha3_state_array A, sha3_bits b, size_t
 ↪  ir)
{
    for (int j = 0; j < b; j++)
    {
        A[0][0][SHA3_WIDTH[j] - 1] ^= SHA3_RC[(j + 7 * ir) % 0xFF];
    }
}
```

3 KECCAK-$p[b, n_r]$

III KECCAK

1 Specification of pad10*1

2 Specification of KECCAK$[c]$

IV Conclusion and Discussion

1 Discussion

In this project, we have searched a lot about the Hash Functions. In brief, our team learned a lot according to this project, and expand one's horizon on the vast field of cryptography.

2 Conclusion

In this project, we have made the study of Hash Functions in order to acquire the extra knowledge in cryptography. What's more, our major skills such as writing and collaboration haven been improved as well. We have had a preliminary understanding of xxx thanks for this subject. We have understood xxx. The superiority of xxx is represented throughout whole the project. Cryptography is quite a complex and esoteric subject and there are so many mysteries in the world. But we still have enough enthusiasm to explore the secrets in the world of cryptography. The study of cryptography is still pending.

V Bibliography

[1]