CS-641 Assignment 7

Team Bullshot

June 2020

Aman Tayal (180074) Rohit Ranjan (180629) Rishabh Kothary (180608)

1 Introduction

For this assignment we were given a weaker variant of KECCAK hash function called WECCAK where we assigned a number of tasks to follow. We try to be as consistent with the notation of [1] in the reference section as possible.

2 Inverse of θ

We use the notation of [2] in the reference section. We are dealing with states which are arrays of the size 5x5x8. To give it a more algebraic structure, we define monomials of the form $x^i y^j z^k$ where $0 \leq i, j \leq 4, 0 \leq k \leq 7$. Then we say that each state can be represented as a polynomial constructed out of these monomials and the coefficient of $x^i y^j z^k$ = A[i,j,k].

The state polynomial is part of polynomial quotient ring defined by the polynomial ring over GF(2)[x,y,z] modulo the ideal generated by
$< 1 + x^5, 1 + y^5, 1 + z^8 >$. We observe that translation can be represented by polynomial multiplication over this ring. Translation $\tau[u][v][w]$ can be represented performed my multiplying with the polynomial $x^u y^v z^w$ in the polynomial ring. The state of the transformation can be found by looking at the new polynomial after multiplication. As in [2] θ can be thought of as a state transformation which can be thought of as multiplication with the polynomial :

$$1 + \bar{y}(x + x^4 z) \text{ where } \bar{y} = 1 + y + ... + y^4$$

So we need to find θ^{-1} which is also a state transformation which can be represented as multiplication my a polynomial. We use the approach in [2], we let the inverse polynomial be of the form $1 + \bar{y}Q$, where Q is a polynomial of variables (x,y) and $\bar{y}$ is as defined before. So we need to solve the following equation :

$$(1 + \bar{y}(x + x^4z))(1 + \bar{y}Q) = 1 \ mod < 1 + x^5, 1 + y^5, 1 + z^8 >$$

Using the fact $\bar{y}^2 = \bar{y}$ this reduces to :

$$\text{Q} = 1 + (1 + x + x^4z)^{-1} \text{ mod} < 1 + x^5, 1 + z^8 >$$

Using Sage we found out Q = $x^4z^7 + x^4z^5 + x^4z^2 + x^4z + x^3z^7 + x^3z^6 + x^3z^5 + x^3z^4 + x^3z^3 + x^2z^7 + x^2z^5 + x^2z^4 + x^2z^2 + x^2 + xz^6 + xz^2 + z^5 + z^3 + z^2 + z + 1$.

Thus if we wish to obtain the values of states before transformation we would construct the transformed state polynomial and multiply with the polynomial $1 + \bar{y}$Q in the polynomial ring and get the value of A[i,j,k] by looking at the corresponding coefficient. Thus we have constructed θ^{-1}.

3 Inverse of χ

As the inverse of χ exists thus it must be a bijective map.The ruling equation is :

A'[x ,y ,z] = A[x ,y ,z] $\oplus$ ((A[(x-1) mod 5,y ,z] $\oplus$ 1)*A[(x+1) mod 5,y ,z])

Thus we see that it is a function of rows of the state matrix. The most important thing to notice is that if we know any two consecutive values of the row say A[x ,y ,z] and A[x+1 ,y ,z] then we can iteratively find the entire row A[* ,y ,z] Now consider the following possibilities:

• Let say A[x ,y ,z] = 1 then according to the χ equation A[x-1 ,y ,z] = A'[x-1 ,y ,z] and hence we know two consecutive elements of the row implying we know the entire row.

• Let say if A[x,y,z] = 0 then according to the χ equation A[x-2 ,y ,z] = A'[x-2 ,y ,z]. If A[x-2 ,y ,z] = 1 then we according to the previous possibility we are done. If not then we can iteratively find A[x-4 ,y ,z] until we get 1. If we dont get one then we get all zero row and hence we are done.

Thus effectively we have justified the fact that knowing one member of the row is enough to get the entire row. Therefore to find the inverse we can randomly guess a value for a member of the row and then compute the entire row and hence verify our solution. Now since this map is a function of rows therefore every row is independent and hence the total number of call required is 2*40 = 80 as there are (5*8) rows and every bit has 2 possibilities. Thus we have found a way to efficiently find the inverse of χ.

4 Security of WECCAK with $F = R \circ R$

4.1 Collision Attack

Consider an arbitrary message m, Let's say we get block B after applying transformation R twice on message m. If somehow it can be ensured that two messages m1, m2 (each of 184 bits) have same last 16 bits, then we can get two messages m1',m2' such that it will collide. Assume that u1 and u2 be the 184 bits sequences of m1 and m2. we can generate a pair of messages h1, h2 each of 184 bits such that,

$$\text{u1} \oplus \text{h1} = \text{u2} \oplus \text{h2}$$

This can be easily done if we take a random 184 bits string and get h2 appropriately.

Now consider the messages m1'= m1,h1, m2' = m2,h2. After xoring with h1, h2, the blocks will become identical and thus will give that same output block and hash.

By passing atmost $2^{16}+ 1$ messages, we are bound to get a collison among the last 16 bits for atleast one pair of messages, by the Pigenhole Principle. By birthday paradox, on passing approx 28 messages, we can get a collision among the last 16 bits atleast once with a high probability.

4.2 Second Pre-image attack

Again consider an arbitrary message m and it's hash $H(m)$. Our goal is to find a message m2 such that it's hash is also $H(m)$.

Consider, after applying round function R on m, we get output block $R(m)$ and $H(m)$ will just first 80 bits of $R(m)$.

If we're to find a message such that last 16 bits of it's hash is zero. then, we can find a second pre-image for any message . let this be m*.

Consider message m2 = m*, m',where following holds,

$$R(m*)[184 : 199] = \text{"}00 ::: 000\text{"}$$
$$m' = R(m*)[0 : 183] \oplus m$$

$$R(m*)[0:183] \oplus m0 = m$$
$$H(m*, m') = R((R(m*)[0:183] \oplus m0)jjR(m*)[184:199])[0:79]$$
$$H(m*, m') = R(m||"00 :: 00")[0:79]$$
$$H(m*, m0) = H(m)$$

Thus if m* can be found than for any m, a second pre-image can be easily found. To

find this m*, we tried hashing random input messages and checked if the output block has last 16 bits as 0. We wrote a code for this that can be found in code.py . We basically ran a brute force code to get this m*, and we were able to get it in around 100,000 iterations most of the times. The message we got in bitstring for $F = R \circ R$ is

$$m* = 00010110101011101101010001001100100000101011111$$

$$10011101001010000011010111001011001000001000011$$

$$01100101111100110010000000001110000110010010000$$

$$00110000000010101001100111010110001001000000100$$

4.3 Pre-image attack

we can observe that constiuent of F are invertible so given any string of 80 bits one can append it with random bits and invert to get an input string . This input can't always be a pre-image since it may or may not have last 16 bits are zero. using this constraint we concluded the follwoing,

Consider a message of length between 185 and (184*2) = 368 characters (inclusive).This message consists of two blocks. The

first block will be used to generate the last 16 bits and the second block will help us modify the digest of

first block to get the required input for second block. We take random bit-strings input for

first block and look at the digest obtained for the

first block. We create a set of these random inputs which map to a significant subset of all possible values of the last 16 bits. The size of this subset will be decided later. Now, we do the same from the other end. We consider the 80 bit digest and append 120 bit random strings to it. Passing it through F^{-1}, we get the input to second round of F. The second block of message is XORed with the

first 184 bits, so, we could obtain any of the required input for this 184 bits. But the last 16 bits can't be modified. These 16 bits are being carried over from the previous block. We iterate over random 120 bits strings to get

a set of inputs(to second round of F) which map to the 80 bit digest. The size of this set will also be decided soon. Currently, we have a set of inputs to first round of F and their corresponding output values for the last 16 bits. We also have a set of inputs to second round of F and we know that these map to the required digest. We will have a pre-image if we could

find a element in the

first set, whose output from F will map to some 16 bit suffix which is also present in the other set. This collision occurs with probability 50 Percent if the size of both these sets is nearly 28. Otherwise, we have iterated over the inputs to the

first round of F to generate all 216 sffixes. If one could assume that if the input to F is random, then all outputs bits are also random, then the total number of random inputs required to generate all 216 sffixes is 2^{16} ln 2^{16} (This many iterations will generate all suffixes with high probability). The simulation of above took 737803 iterations of random bit strings to generate all 216 suffixes for F. The code for this simulation is present in code.py. The value 2^{16} ln 2^{16} is approximately 729042, which is pretty close to our number of iterations.

5 References

(1)https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
(2)https://keccak.team/obsolete/Keccak-main-1.1.pdf