CS401

Term Project Phase IV

Mustafa Onuralp Devres 29310

Ege Kaan Özalp 2898

5.1.1 Inverse of S Function

```
#Inverse of S
S_inv: .byte 0xC, 0x3, 0x0, 0xA, 0xB, 0x4, 0x5, 0xF, 0x9, 0xE, 0x6, 0xD, 0x2, 0x7, 0x8, 0x1
   ,0xA, 0x7, 0x6, 0x9, 0x1, 0x2, 0xC, 0x5, 0x3, 0x4, 0x8, 0xF, 0xD, 0xE, 0xB, 0x0
   ,0x9, 0x2, 0xF, 0x8, 0x0, 0xC, 0x3, 0x6, 0x4, 0xD, 0x1, 0xE, 0x7, 0xB, 0xA, 0x5
   ,0xB, 0x5, 0x4, 0xF, 0xC, 0x6, 0x9, 0x0, 0xD, 0x3, 0xE, 0x8, 0x1, 0xA, 0x2, 0x7
```

Figure 1. Inverse of S Table

In the inverse of S function implementation, a similar approach to regular S function is used however instead of loading the address for the regular S Table the inverse of the S table is loaded. The input and output can be found in $t0 register.

```
inv_S: #The extract function needs to be called before calling this

    la $t5, S_inv #Base address of S_inv
```

Figure 2. Inverse S Table Address Loading

5.1.2 Inverse of L Function

The input is given through the $t0 register for the inverse of the L function. First, the input is circular shifted to left and then right for 10 bits. Then the input and the circular shifts are XOR'ed with each other to obtain Z. Lastly, Z is circular shifted to the left and right by 4 bits. All are XOR'ed and the inverse is saved to the $t0 register.

```
#Circular Shift Left 10
andi $t1, $t0, 0x3F #0000000000111111 First 6 digits for the left circular shift
sll $t1, $t1, 10
andi $t2, $t0, 0xFFC0 #1111111111000000 Last 10 digits for the left circular shift
srl $t2, $t2, 6

or $t1, $t1, $t2

#Circular Shift Right 10
andi $t3, $t0, 0x3FF #0000001111111111 First 10 digits for the right circular shift
sll $t3, $t3, 6
andi $t4, $t0, 0xFC00 #1111110000000000 Last 6 digits for the right circular shift
srl $t4, $t4, 10

or $t3, $t3, $t4

xor $t0, $t0, $t1
xor $t0, $t0, $t3 #Z
```

Figure 3. Calculation of Z in the Inverse of L Function

5.1.3 Inverse of P Function

Similar to inverse of S Function implementation the address for the inverse of p-box is loaded to the P function implementation. The input is given at $t0 register and the output is retrieved from the $t9 register.

```
# P-Box Inverse
pbox_inv: .byte 5, 7, 3, 4, 2, 6, 1, 0
```

Figure 4. Inverse of P-Box Table

```
inv_P:

    li $t9, 0
    la $t8, pbox_inv  #Load the address for the P-box-inv
```

Figure 5. Inverse of P-Box Address Load

5.1.4 Inverse of F Function

The input is gotten from and outputted to the $t0 register. First the inverse linear function of the input is calculated then the inverse S function is obtained, resulting in the value of Z. After Z is obtained it is manipulated in such a way that half of the bits are used for the inverse of P and the other are concatenated to the output of the inverse of P.

```
inv_F: #Input at $t0

    addi $sp, $sp, -4
    sw $ra, 0($sp)

    jal inv_L #Output at $t0
    jal extract
    jal inv_S #Output at $t0 = Z

    jal extract #To retrieve z0, z1, z2, z3
```

```
    sll $t2, $t2, 4 #Shift z2 left by 4 bits
    or $t0, $t2, $t1 #z2 or z3

    addi $sp, $sp, -8
    sw $t4, 0($sp)
    sw $t3, 4($sp)

    jal inv_P #output at $t9
```

Figure 6. Inverse L, S and P Calculations in the Inverse F Function

5.1.5 Inverse of W Function

The input for the inverse function for W is the same as the regular W function where the inputs are $s0, $s1 and $s2 registers for the X, A and B values respectively. First inverse F function is calculated for the $s0 register then XOR'ed with $s2 register. Lastly, the output is inversed with F function again and XOR'ed with $s1 register where it is saved to $t0 register.

```
move $t0, $s0
jal inv_F

xor $t0, $t0, $s2 #xor output and B
jal inv_F

xor $t0, $t0, $s1
```

Figure 7. Inverse of W Function Implementation

5.2 Decryption Algorithm

In the decryption algorithm a similar approach to the encryption algorithm is taken where first the addresses for R, K, P and C are loaded to $s registers.

```
#Load addresses
la $s3, R
la $s4, K
la $s5, P
la $s6, C

li $s7, 0 #initizlize i for the loop
```

Figure 8. Load Addresses

Firstly, to save the intermediate values space in the stack is opened to save them. Later, at each step the values are stored into the stack and then later loaded when needed.

```
beq $s7, 8, end_loop_decryption
addi $sp, $sp, -44 #make space for t0-2, T0-T7
```

Figure 9. Space in the Stack

For example, at the first step to call the inverse of W function, the input for the function is called and then moved to their respective $s0 registers.

```
lw $t1, 0($s3) #R[0]
sll $t2, $s7, 2 #Multiply the index by 4
add $t2, $t2, $s6
lw $t2, 0($t2) #C[i]

sub $t1, $t2, $t1
andi $t1, $t1, 0xFFFF #Get the lower 16-bits

move $s0, $t1 #C[i] - R[0] mod 2^16
```

Figure 10. Example Input Preparation

After the operations are completed, the value is saved to the stack.

```
jal inv_W
lw $t1, 12($s3) #R[3]

sub $t0, $t0, $t1
andi $t0, $t0, 0xFFFF #Get the lower 16-bits

sw $t0, 8($sp) #Save $t2
```

Figure 11. Save to the Stack

At the 4th step in the Decryption Algorithm described in the project document the value calculated saved to the Plain Text array as shown below.

```
jal inv_W
lw $t1, 0($s3) #R[0]

sub $t0, $t0, $t1
andi $t0, $t0, 0xFFFF #Get the lower 16-bits

sll $t2, $s7, 2 #Multiply the index by 4
add $t2, $t2, $s5 #Get the position of P

sw $t0, 0($t2) #Save to P
```

Figure 12. Value Saved to P

Updating the state vector R implementation is same as the encryption algorithm where it is similarly implemented.

When the ciphertext shown below given is given as input to the program the following output is retrieved which is the initial plaintext.

```
C[8] = {0x926a, 0xf9f8, 0x5bc5, 0xb575, 0x9707, 0x06a0, 0x3407, 0x33f2}
```

Figure 13. Input Ciphertext

```
Done decrypting C:
0x00001100
0x00003322
0x00005544
0x00007766
0x00009988
0x0000bbaa
0x0000ddcc
0x0000ffee
```

Figure 14. Decrypt Ciphertext

There is an option also to give plain text as decimal input from the keyboard if that is done the following encryption and decryption follows.

```
4352
13090
21828
30566
39304
48042
56780
65518
```

```
Initial Plain Text P:
0x00001100
0x00003322
0x00005544
0x00007766
0x00009988
0x0000bbaa
0x0000ddcc
0x0000ffee
```

Figure 15. Plain Text Input in the Form of Decimal and Hexadecimal

```
Done encrypting P:        Done decrypting C:
0x0000926a                0x00001100
0x0000f9f8                0x00003322
0x00005bc5                0x00005544
0x0000b575                0x00007766
0x00009707                0x00009988
0x000006a0                0x0000bbaa
0x00003407                0x0000ddcc
0x000033f2                0x0000ffee
```

Figure 16. Encryption and Decryption Results of the Plaintext

The initial value for the plaintext and the decrypted ciphertext resulted in the same output which shows that the encryption and the decryption functions, function correctly. Moreover, below details of the main function can be found.

```
    # Read 8 decimal values from the keyboard
    la $a0, P
    li $t0, 8 #8 values will be read
ead_loop:
    li $v0, 5
    syscall
    sw $v0, 0($a0)
    addi $a0, $a0, 4
    subi $t0, $t0, 1
    bnez $t0, read_loop

    la $t0, P
    li $t1, 0

    li $v0, 4
    la $a0, p_state_msg
    syscall

    jal print_P_loop
    jal initialization
    jal encryption

    jal initialization
    jal decryption
```

Figure 17. Main Function Calls and Input Reading