Experiment No. 3

Title: A5/1

Batch: A3 **Roll No.:16010421073** **Experiment No.:3**

Aim: To implement stream cipher A5/1

Resources needed: Windows/Linux

Theory: Pre Lab/ Prior Concepts:

A5/1 employs three linear feedback shift registers , or LFSRs, which are labeled X, Y, and Z. Register X holds 19 bits, ($x_0,x_1 \ldots x_{18}$).The register Y holds 22 bits, ($y_0,y_1 \ldots y_{21}$) and Z holds 23 bits, ($z_0,y_1 \ldots z_{22}$) ·Of course, all computer geeks love powers of two, so it's no accident that the three LFSRs hold a total of 64 bits.

Not coincidentally, the A5/1 key K is also 64 bits. The key is used as the initial fill of the three registers, that is, the key is used as the initial values in the three registers. After these three registers are filled with the key,1 we are ready to generate the keystream. But before we can describe how the keystream is generated, we need to say a little more about the registers X, Y, and Z.

When register X steps, the following series of operations occur:

$$t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$$
$$x_i = x_{i-1} \text{ for } i = 18, 17, 16, \ldots, 1$$
$$x_0 = t$$

Similarly, for registers Y and Z, each step consists of

$$t = y_{20} \oplus y_{21}$$
$$y_i = y_{i-1} \text{ for } i = 21, 20, 19 \ldots, 1$$
$$y_0 = t$$

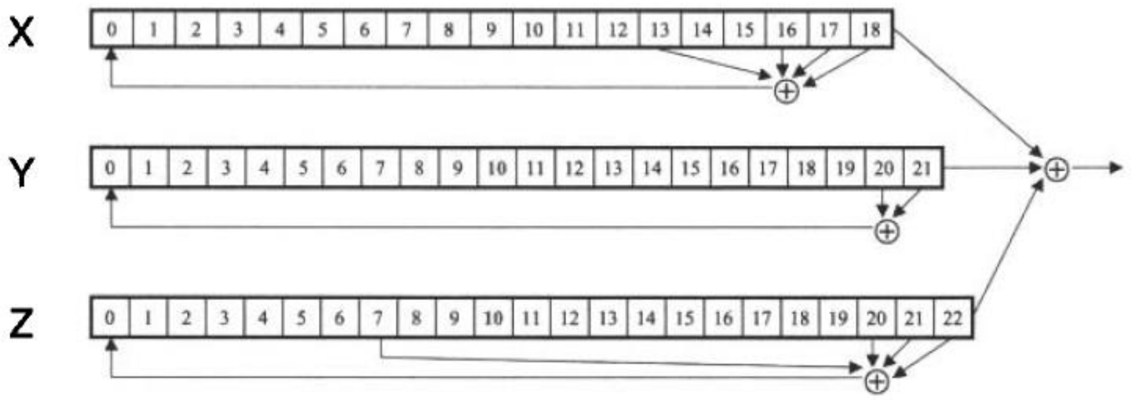
and

$$t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$$
$$z_i = z_{i-1} \text{ for } i = 22, 21, 20, \ldots, 1$$
$$z_0 = t$$

respectively.

Given three bits x, y, and z, define ma,](x,y, z) to be the majority vote function, that is, if the majority of x, y, and z are 0, the function returns 0; otherwise it returns 1. Since there are an odd number of bits, there cannot be a tie, so this function is well defined.

The wiring diagram for the A5/1 algorithm is illustrated below:



A5/1 Keystream Generator

Procedure / Approach /Algorithm / Activity Diagram:

A. Key Stream generation Algorithm:

At each step: m = maj(x_8, y_{10}, z_{10})

-Examples: maj(0,1,0) = 0 and maj(1,1,0) = 1

If x_8 = m then X steps

-$t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$

-$x_i = x_{i-1}$ for i = 18,17,…,1 and x_0 = t

If y_{10} = m then Y steps

-$t = y_{20} \oplus y_{21}$

-$y_i = y_{i-1}$ for i = 21,20,…,1 and y_0 = t

If z_{10} = m then Z steps

-$t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$

-$z_i = z_{i-1}$ for i = 22,21,…,1 and z_0 = t

Keystream bit is $x_{18} \oplus y_{21} \oplus z_{22}$

Implementation:

Implement the A5/1 algorithm. Encryption and decryption function should ask for key and a input and show the output to the user.

Results: (Program with output as per the format)

Code:

```
x = input("Enter the value of register X: ")
y = input("Enter the value of register Y: ")
z = input("Enter the value of register Z: ")

print("Initially ")
print("Register X: ", x)
print("Register Y: ", y)
print("Register Z: ", z)
```

```
print("\n")

keystream = []

for i in range(32):
    c = int(x[8]) + int(y[10]) + int(z[10])
    if c <= 1:
        m = 0
    else:
        m = 1

    if int(x[8]) == m:
        tx = int(x[13]) ^ int(x[16]) ^ int(x[17]) ^ int(x[18])
        x1 = x[0:18]
        x = str(tx) + x1

    if int(y[10]) == m:
        ty = int(y[20]) ^ int(y[21])
        y1 = y[0:21]
        y = str(ty) + y1

    if int(z[10]) == m:
        tz = int(z[7]) ^ int(z[20]) ^ int(z[21]) ^ int(z[22])
        z1 = z[0:22]
        z = str(tz) + z1

    print("X: ", x)
    print("Y: ", y)
    print("Z: ", z)

    k = int(x[18]) ^ int(y[21]) ^ int(z[22])
    print("For iteration ", i + 1, ", keystream bit: ", k)
    print("\n")
    keystream.append(k)

print("After all iterations: ")
print("Register X: ", x)
print("Register Y: ", y)
print("Register Z: ", z)
print("Keystream: ", keystream)

pt = input("Enter the plaintext: ")
ctr = 0
key = ''

for i in keystream:
    key = key + str(i)
    ctr = ctr + 1
    if ctr == len(pt):
        break
```

```python
ct = ''

for i, j in zip(key, pt):
    x = int(i) ^ int(j)
    ct = ct + str(x)

print("Ciphertext is: ", ct)
```

Output:

```
Enter the value of register X: 1010110010100010110
Enter the value of register Y: 1010110101010101011010
Enter the value of register Z: 10110010001011010010001
Initially
Register X:  1010110010100010110
Register Y:  1010110101010101011010
Register Z:  10110010001011010010001


X:  0101011001010001011
Y:  1010110101010101011010
Z:  11011001000101101001000
For iteration  1 , keystream bit:  1


X:  0010101100101000101
Y:  1101011010101010101101
Z:  11101100100010110100100
For iteration  2 , keystream bit:  0


X:  0001010110010100010
Y:  1101011010101010101101
Z:  11110110010001011010010
For iteration  3 , keystream bit:  1


X:  0000101011001010001
Y:  1110101101010101010110
Z:  11110110010001011010010
For iteration  4 , keystream bit:  1


X:  0000101011001010001
Y:  1111010110101010101011
Z:  11111011001000101101001
For iteration  5 , keystream bit:  1
```

```
X:  1000010101100101000
Y:  0111101011010101010101
Z:  01111101100100010110100
For iteration  6 , keystream bit:  1


X:  1100001010110010100
Y:  1011110101101010101010
Z:  00111110110010001011010
For iteration  7 , keystream bit:  0


X:  1110000101011001010
Y:  1101111010110101010101
Z:  00111110110010001011010
For iteration  8 , keystream bit:  1


X:  1111000010101100101
Y:  1101111010110101010101
Z:  10011111011001000101101
For iteration  9 , keystream bit:  1


X:  1111100001010110010
Y:  1110111101011010101010
Z:  11001111101100100010110
For iteration  10 , keystream bit:  0


X:  0111110000101011001
Y:  1111011110101101010101
Z:  11001111101100100010110
For iteration  11 , keystream bit:  0


X:  0111110000101011001
Y:  1111101111010110101010
Z:  11100111110110010001011
For iteration  12 , keystream bit:  0
```

```
X:  1011111000010101100
Y:  1111110111101011010101
Z:  11110011111011001000101
For iteration  13 , keystream bit:  0


X:  1011111000010101100
Y:  1111111011110101101010
Z:  11111001111101100100010
For iteration  14 , keystream bit:  0


X:  1011111000010101100
Y:  1111111101111010110101
Z:  01111100111110110010001
For iteration  15 , keystream bit:  0


X:  1011111000010101100
Y:  1111111110111101011010
Z:  10111110011111011001000
For iteration  16 , keystream bit:  0


X:  1011111000010101100
Y:  1111111111011110101101
Z:  01011111001111101100100
For iteration  17 , keystream bit:  1


X:  0101111100001010110
Y:  1111111111101111010110
Z:  01011111001111101100100
For iteration  18 , keystream bit:  0


X:  0101111100001010110
Y:  1111111111110111101011
Z:  00101111100111110110010
For iteration  19 , keystream bit:  1
```

```
X:  0010111110000101011
Y:  1111111111110111101011
Z:  00010111110011111011001
For iteration  20 , keystream bit:  1


X:  1001011111000010101
Y:  0111111111111011110101
Z:  00010111110011111011001
For iteration  21 , keystream bit:  1


X:  0100101111100001010
Y:  1011111111111101111010
Z:  00010111110011111011001
For iteration  22 , keystream bit:  1


X:  1010010111110000101
Y:  1101111111111110111101
Z:  00010111110011111011001
For iteration  23 , keystream bit:  1


X:  0101001011111000010
Y:  1110111111111111011110
Z:  00010111110011111011001
For iteration  24 , keystream bit:  1


X:  1010100101111100001
Y:  1111011111111111101111
Z:  00010111110011111011001
For iteration  25 , keystream bit:  1


X:  0101010010111110000
Y:  1111011111111111101111
Z:  00001011111001111101100
For iteration  26 , keystream bit:  1
```

```
X:  1010101001011111000
Y:  0111101111111111110111
Z:  00000101111100111110110
For iteration  27 , keystream bit:  1

X:  1010101001011111000
Y:  0011110111111111111011
Z:  10000010111110011111011
For iteration  28 , keystream bit:  0


X:  1010101001011111000
Y:  0001111011111111111101
Z:  01000001011111001111101
For iteration  29 , keystream bit:  0


X:  1010101001011111000
Y:  1000111101111111111110
Z:  10100000101111100111110
For iteration  30 , keystream bit:  0


X:  1010101001011111000
Y:  1100011110111111111111
Z:  01010000010111110011111
For iteration  31 , keystream bit:  0


X:  1101010100101111100
Y:  1100011110111111111111
Z:  10101000001011111001111
For iteration  32 , keystream bit:  0
```

```
After all iterations:
Register X:  1101010100101111100
Register Y:  1100011110111111111111
Register Z:  10101000001011111001111
Keystream:  [1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
Enter the plaintext: 10110101010011001001001001
Ciphertext is:  00001000110011000010110110
```

Questions:

1) List the stream cipher used in current date along with the name of applications in which those are used.

Ans:-

RC4 - RC4, which stands for Rivest Cipher 4, is the most widely used of all stream ciphers,particularly in software. It's also known as ARCFOUR or ARC4. RC4 steam chiphers have been used in various protocols like WEP and WPA (both security protocols for wireless networks) as well as in TLS. Unfortunately, recent studies have revealed vulnerabilities in RC4, prompting Mozilla and Microsoft to recommend that it be disabled where possible. Infact, RFC 7465 prohibits the use of RC4 in all versions of TLS.

These recent findings will surely allow other stream ciphers (e.g. SALSA, SOSEMANUK,PANAMA, and many others, which already exist but never gained the same popularity as RC4) to emerge and possibly take its place.

RC4 is used in various applications such as WEP from 1997 and WPA from 2003. We also find applications of RC4 in SSL from 1995 and it is a successor of TLS from 1999. RC4 is used in varied applications because of its simplicity, speed, and simplified implementation inboth software and hardware.

There are various types of RC4 such as Spritz, RC4A, VMPC, and RC4A.

1. SPRITZ: Spritz can be used to build a cryptographic hash function, a deterministic random bit generator (DRBG), n an encryption algorithm that supports authenticatedencryption with associated data (AEAD).
2. RC4A: Souraduyti Paul and Bart Preneel have proposed an RC4 variant, which they callRC4A, which is stronger than RC4.
3. VMPC: VMPC is another variant of RC4 which stands for Variably Modified Permutation Composition.
4. RC4A+: RC4A+ is a modified version of RC4 with a more complex three-phase key schedule which takes about three times as long as RC4 and a more complex output function which performs four additional lookups in the S array for each byte output,taking approximately 1.7 times as long as basic RC4.

Outcomes:

CO2: Illustrate different cryptographic algorithms for security

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

In This experiment, we learnt to successfully implement and execute stream

cipher A5/1.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References: Books/ Journals/ Websites:

1. Mark Stamp, “Information Security Principles and Practice”, Wiley.
2. Behrouz A. Forouzan, “Cryptography and Network Security”, Tata McGraw Hill
3. William Stalling, “Cryptography and Network Security”, Prentice Hall