Experiment No. 3

Title: Stream Cipher **-** A5/1

Batch: B-3 **Roll No.: 16010422234** **Experiment No.: 03**

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…x_{18}).The register Y holds 22 bits, (y_0,y_1…y_{21}) and Z holds 23 bits, (z_0,y_1…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

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

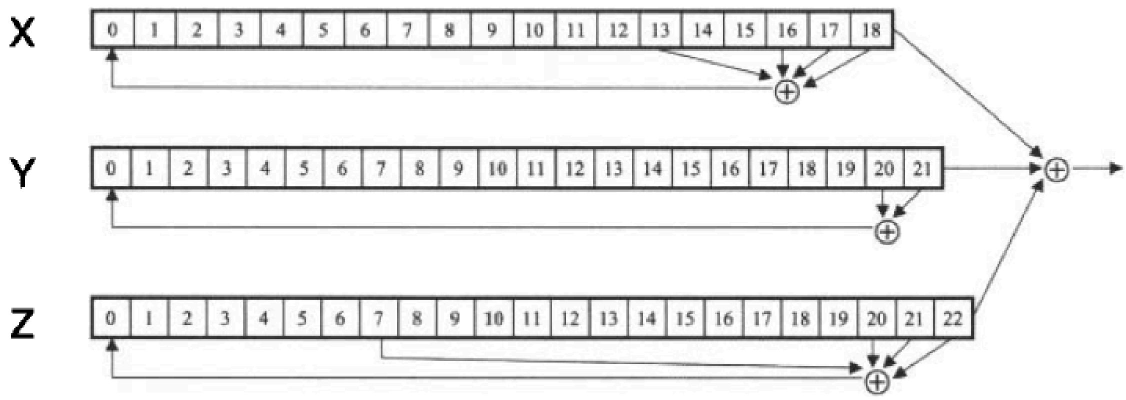
and

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

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 $i = 18,17,\ldots,1$ and $x_0 = t$

If $y_{10} = m$ then Y steps

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

$-y_i = y_{i-1}$ for $i = 21,20,\ldots,1$ 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 $i = 22,21,\ldots,1$ 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 a key and an input and show the output to the user.

Results: (Program with output as per the format)

```
class A51:
    def __init__(self, key):
        self.x = [int(b) for b in key[:19]]
        self.y = [int(b) for b in key[19:41]]
        self.z = [int(b) for b in key[41:]]

    def majority(self, x, y, z):
        return int(x + y + z > 1)
```

```
    def step_x(self):
        t = self.x[13] ^ self.x[16] ^ self.x[17] ^ self.x[18]
        self.x = [t] + self.x[:-1]

    def step_y(self):
        t = self.y[20] ^ self.y[21]
        self.y = [t] + self.y[:-1]

    def step_z(self):
        t = self.z[7] ^ self.z[20] ^ self.z[21] ^ self.z[22]
        self.z = [t] + self.z[:-1]

    def generate_keystream(self, length):
        keystream = []
        for _ in range(length):
            m = self.majority(self.x[8], self.y[10], self.z[10])
            if self.x[8] == m:
                self.step_x()
            if self.y[10] == m:
                self.step_y()
            if self.z[10] == m:
                self.step_z()

            keystream_bit = self.x[18] ^ self.y[21] ^ self.z[22]
            keystream.append(keystream_bit)

        return keystream

def xor_bitstrings(a, b):
    return [ai ^ bi for ai, bi in zip(a, b)]

def main():
    key = input("Enter 64-bit key as a binary string: ")
    if len(key) != 64 or not all(c in '01' for c in key):
        print("Invalid key! Key must be a 64-bit binary string.")
        return

    plaintext = input("Enter plaintext as a binary string: ")
    if len(plaintext) % 8 != 0 or not all(c in '01' for c in plaintext):
        print("Invalid plaintext! It must be a binary string and a
multiple of 8 bits.")
        return

    cipher = A51(key)
    plaintext_bits = [int(b) for b in plaintext]
    keystream = cipher.generate_keystream(len(plaintext_bits))
```

```
    ciphertext_bits = xor_bitstrings(plaintext_bits, keystream)
    ciphertext = ''.join(map(str, ciphertext_bits))
    print("Ciphertext (binary):", ciphertext)

    cipher = A51(key)
    ciphertext_bits = [int(b) for b in ciphertext]
    decrypted_bits = xor_bitstrings(ciphertext_bits,
cipher.generate_keystream(len(ciphertext_bits)))
    decrypted_text = ''.join(map(str, decrypted_bits))
    print("Decrypted text (binary):", decrypted_text)

if __name__ == "__main__":
    main()
```

```
PS C:\Users\chand\Downloads\V SEM\INS\EXP3> & C:/Users/chand/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/chand/D
ownloads/V SEM/INS/EXP3/exp3.py"
Enter 64-bit key as a binary string: 1010101010101010101010101010101010101010101010101010101010101010
Enter plaintext as a binary string: 11001010
Ciphertext (binary): 10011111
Decrypted text (binary): 11001010
PS C:\Users\chand\Downloads\V SEM\INS\EXP3>
```

Questions:

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

a) RC4: Used in protocols such as Secure Sockets Layer (SSL)/Transport Layer Security (TLS) and Wired Equivalent Privacy (WEP) for Wi-Fi security.
b) ChaCha20: Used in Google's QUIC protocol, and in TLS for HTTPS connections.
c) Salsa20: Employed in some versions of the SSH protocol.
d) Snow 3G: Used in the 3GPP standards for mobile communication.
e) Grain: Implemented in lightweight cryptographic protocols, particularly in resource-constrained environments like IoT devices.

Outcomes: Illustrate different cryptographic algorithms for security.

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

The experiment successfully demonstrated the implementation of the A5/1 stream cipher. By understanding and applying the key stream generation algorithm, we were able to encrypt and decrypt data using the A5/1 algorithm. The implementation highlighted the working mechanism of LFSRs and the majority function, providing insight into the functioning of stream ciphers in general. Overall, the objectives of implementing and understanding a basic stream cipher were achieved, and we gained practical knowledge of encryption and decryption processes using a specific algorithm.

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

---