Experiment No. 3

Title: A5/1

Batch: A3 **Roll No.: 16010421059** **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…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

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

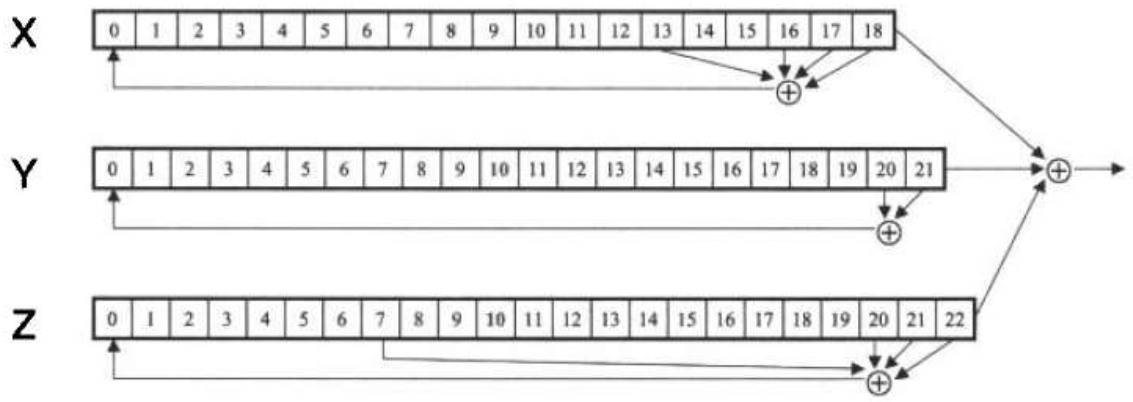
and

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

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,…,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)

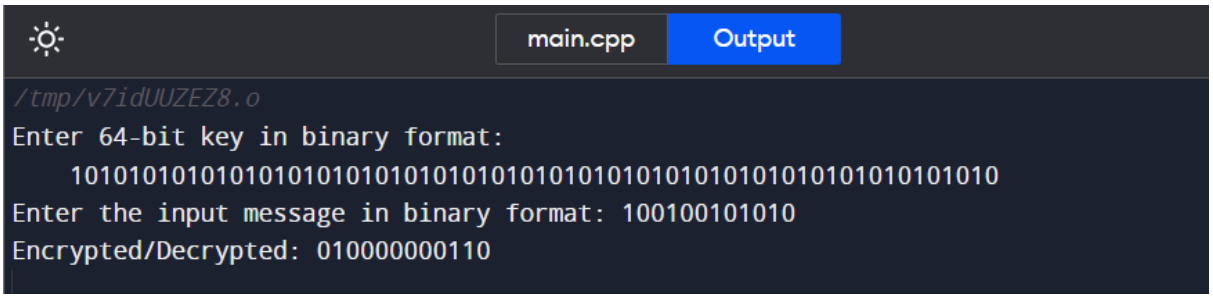
INPUT:

```
#include <iostream>
#include <bitset>
#include <vector>

using namespace std;

class A5_1 {
public:
   bitset<19> x;
   bitset<22> y;
   bitset<23> z;
```

```
    A5_1() {}

    void init(bitset<64> key) {
        for (int i = 0; i < 64; ++i) {
            x[i % 19] = x[i % 19] ^ key[i];
            y[i % 22] = y[i % 22] ^ key[i];
            z[i % 23] = z[i % 23] ^ key[i];
        }

        for (int i = 0; i < 100; ++i) {
            clock();
        }
    }

    void clock() {
        int m = majority(x[8], y[10], z[10]);

        if (x[8] == m) {
            int t = x[13] ^ x[16] ^ x[17] ^ x[18];
            x >>= 1;
            x[18] = t;
        } else {
            x >>= 1;
        }

        if (y[10] == m) {
            int t = y[20] ^ y[21];
            y >>= 1;
            y[21] = t;
        } else {
            y >>= 1;
        }

        if (z[10] == m) {
            int t = z[7] ^ z[20] ^ z[21] ^ z[22];
            z >>= 1;
            z[22] = t;
        } else {
            z >>= 1;
        }
    }

    int majority(int a, int b, int c) {
        return (a & b) ^ (a & c) ^ (b & c);
    }

    int generateKeystreamBit() {
        return x[18] ^ y[21] ^ z[22];
    }
};

string encryptDecrypt(const string& input, const bitset<64>& key) {
    A5_1 a51;
```

```
    a51.init(key);

    string output;
    for (char bit : input) {
        int keystreamBit = a51.generateKeystreamBit();
        int inputBit = bit - '0';
        int encryptedBit = inputBit ^ keystreamBit;
        output.push_back(char(encryptedBit + '0'));
        a51.clock();
    }

    return output;
}

int main() {
    string keyStr;
    cout << "Enter 64-bit key in binary format: ";
    cin >> keyStr;

    bitset<64> key(keyStr);

    string input;
    cout << "Enter the input message in binary format: ";
    cin >> input;

    string encrypted = encryptDecrypt(input, key);
    cout << "Encrypted/Decrypted: " << encrypted << endl;

    return 0;
}
```

OUTPUT:



Questions:

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

Ans: Few stream ciphers and potential applications:

1. RC4 (Rivest Cipher 4):
   - Applications: RC4 was widely used in various applications, including early versions of the SSL/TLS protocols for securing web traffic, wireless WEP encryption, and more. However, due to security weaknesses, its use has significantly diminished.

2. Salsa20/ChaCha:

- Applications: Salsa20 and ChaCha are considered modern and secure stream ciphers. They are often used in encryption for secure messaging apps like Signal and WhatsApp. They are also employed in various VPN protocols for secure communication.

3. GSM A5/1:
- Applications: GSM A5/1 is an older stream cipher used in the GSM (Global System for Mobile Communications) standard to encrypt voice and data traffic in 2G mobile networks. It's not considered secure today, and 3G and 4G networks use different encryption methods.

4. HC-128 and HC-256:
- Applications: HC-128 and HC-256 are stream ciphers designed for high-speed and security. They have been used in various applications, including secure wireless communication, disk encryption, and secure messaging.

5. E0 (Bluetooth Encryption):
- Applications: E0 is used for encryption in Bluetooth connections. While it's a stream cipher, it's considered weak, and more recent versions of Bluetooth (e.g., Bluetooth 4.0 and later) use stronger ciphers.

6. SNOW 3G:
- Applications: SNOW 3G is a stream cipher used in 3G mobile networks for encryption and integrity protection of user data. It's used in applications related to mobile communication.

7. Grain:
- Applications: Grain is a lightweight stream cipher used in applications requiring low power consumption and low memory usage, such as RFID tags and sensor networks.

8. Trivium:
- Applications: Trivium is another lightweight stream cipher suitable for resource-constrained devices and applications like RFID, secure key exchange, and hardware security modules.

Outcomes:

CO 2: Illustrate different cryptographic algorithms for security.

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

Implemented stream cipher A5/1 successfully.

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