Experiment No. 3 Title: A5/1

**Batch: B3 Roll No.: 16010420099 Experiment No.: 03**

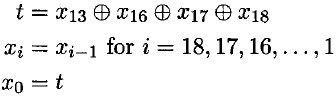
**Aim:** To implement stream cipher A5/1

**Resources needed:** Windows/Linux

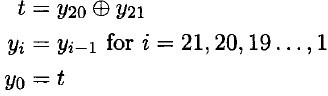
# Theory: Pre Lab/ Prior Concepts:

A5/1 employs three linear feedback shifl registers , or LFSRs, whichare labeled X, Y, and Z. Register X holds 19 bits, (x0,x1…x18)-The register Y holds 22 bits, (y0,y1…y21) and Z holds 23 bits, (z0,y1…z22)·Of course, all computer geeks love powers of two, so it's no accident that the three LFSRs hold a total of 64 bits.

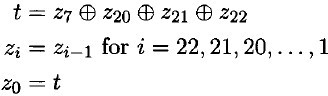
Not coincidentally, the A5/1 key K is also 64 bits. The key is used as the initial fill of the three registers, that is, the key is used as the initial values in the three registers. After these three registers are filled with the key,1 we are ready to generate the keystream. But before we can describe 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:

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



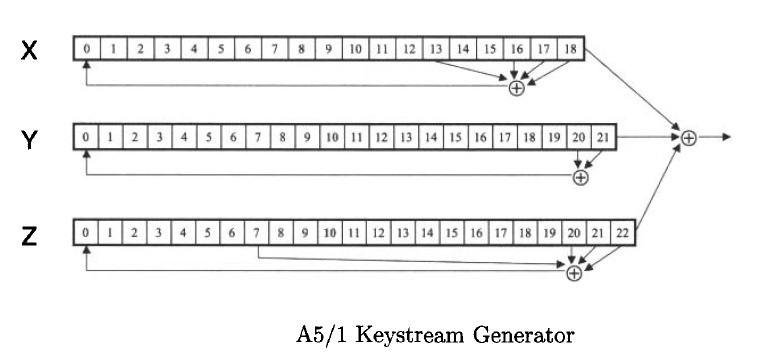
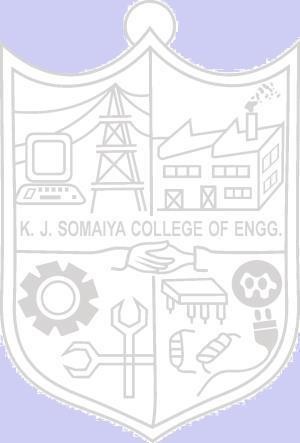
and



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:



# Procedure / Approach /Algorithm / Activity Diagram:

**A. Key Stream generation Algorithm:**

At each step: m = maj(x8, y10, z10)

-Examples: maj(0,1,0) = 0 and maj(1,1,0) = 1 If x8 = m then X steps

-t = x13x16x17x18

-xi = xi1 for i = 18,17,…,1 and x0 = t If y10 = m then Y steps

-t = y20y21

-yi = yi1 for i = 21,20,…,1 and y0 =t If z10 = m then Z steps

-t = z7z20z21z22

-zi = zi1 for i = 22,21,…,1 and z0 = t

# Keystreambit is *x*18*y*21*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)

public class Binary {

public static String convertTextToBinaryString(String text) { String binaryString = "";

for (char c : text.toCharArray()) binaryString += Integer.toBinaryString(c);

return binaryString;

}

public static String convertBinaryStringToText(String binaryString)

{

String text = "";

String[] bytes = binaryString.split("(?<=\\G.{7})"); for (String textByte : bytes) text += (char)

Integer.parseInt(textByte, 2);

return text;

}

public static char XOR(char a, char b) { if (a == b) return '0';

else return '1';

}

public static char maxBit(char a, char b) { return a > b ? a : b;

}

}

public class LFSR {

private char[] bits; // LSB is at the starting index, NOT at the

end

private int clockingBit; private int[] tappedBits;

LFSR(int \_numberOfBits, int \_clockingBit, int[] \_tappedBits) {

// Initialize with all zeroes bits = new char[\_numberOfBits];

for (int i = 0; i < \_numberOfBits; i++) bits[i] = '0'; clockingBit = \_clockingBit;

tappedBits = \_tappedBits;

}

LFSR(char[] \_bits, int \_clockingBit, int[] \_tappedBits) { bits = \_bits;

clockingBit = \_clockingBit; tappedBits = \_tappedBits;

}

void setBits(char[] \_bits) { bits = \_bits;

}

char getClockingBit() {

// Will return the clocking bit return bits[clockingBit];

}

char clock() {

// Will perform the clocking mechanism and

// return the output bit char inputBit = '0';

for (int bitIndex : tappedBits)

inputBit = XOR(inputBit, bits[bitIndex]);

char outputBit = bits[bits.length-1]; for (int i = bits.length - 1; i > 0; i--)

bits[i] = bits[i-1];

bits[0] = inputBit; return outputBit;

}

}

public class KeyGenerator {

private LFSR x = new LFSR(19, 8, new int[] {18, 17, 16, 13}); private LFSR y = new LFSR(22, 10, new int[] {20, 21}); private LFSR z = new LFSR(23, 10, new int[] {22, 21, 20, 7});

KeyGenerator(char[] key) { char[] xBits = new char[19]; char[] yBits = new char[22]; char[] zBits = new char[23];

System.arraycopy(key, 0, xBits, 0, 19);

System.arraycopy(key, 19, yBits, 0, 22);

System.arraycopy(key, 22, zBits, 0, 23); x.setBits(xBits);

y.setBits(yBits); z.setBits(zBits);

}

public char getNextKeyBit() {

char xClockingBit = x.getClockingBit(); char yClockingBit = y.getClockingBit(); char zClockingBit = z.getClockingBit();

char maxClockingBit = maxBit(xClockingBit, maxBit(yClockingBit, zClockingBit));

char outputBit = '0';

if (maxClockingBit == xClockingBit) outputBit = XOR(outputBit, x.clock());

if (maxClockingBit == yClockingBit) outputBit = XOR(outputBit, y.clock());

if (maxClockingBit == zClockingBit) outputBit = XOR(outputBit, z.clock());

return outputBit;

}

}

import java.util.Scanner; import static app.Binary.\*;

public class StreamCipher { private String keyBinary;

StreamCipher(String key) { keyBinary = "";

keyBinary = convertTextToBinaryString(key);

keyBinary = String.format("%64s", keyBinary).replace(' ', '0'); if (keyBinary.length() > 64) keyBinary = keyBinary.substring(0,

64);

}

public String encrypt(String plaintext) { KeyGenerator generator = new

KeyGenerator(keyBinary.toCharArray());

String plaintextBinary = convertTextToBinaryString(plaintext); char[] ciphertextBinary = plaintextBinary.toCharArray();

for (int i = 0; i < ciphertextBinary.length; i++) ciphertextBinary[i] = XOR(ciphertextBinary[i],

generator.getNextKeyBit());

return convertBinaryStringToText(new String(ciphertextBinary));

}

public String encryptBinary(String plaintext) { KeyGenerator generator = new

KeyGenerator(keyBinary.toCharArray());

char[] ciphertext = plaintext.toCharArray(); for (int i = 0; i < ciphertext.length; i++)

ciphertext[i] = XOR(ciphertext[i], generator.getNextKeyBit());

return new Str

i

}

ng(ciphertext);

public String decr KeyGenerator g KeyGenerator(keyBinary

y

e

.

e

String ciphert convertTextToBinaryStr

i

x

char[] plainte for (int i = 0 plaintextB generator.getNextKeyBi

;

i

t

B

return convert

pt(String ciphertext) { nerator = new toCharArray()); xtBinary = ng(ciphertext);

tBinary = ciphertextBinary.toCharArray(); i < plaintextBinary.length; i++)

nary[i] = XOR(plaintextBinary[i], ());

inaryStringToText(new String(plaintextBinary));

}

public String decr KeyGenerator g KeyGenerator(keyBinary

y

e

.

x

char[] plainte for (int i = 0 plaintext[ generator.getNextKeyBi

;

i

t

i

return new Str

ptBinary(String ciphertext) { nerator = new toCharArray());

t = ciphertext.toCharArray(); i < plaintext.length; i++)

] = XOR(plaintext[i], ());

ng(plaintext);

}

public static void

:

// Example key 0101001000011010110001

1

Scanner reader

main(String[] args) {

10001100100101001000000110111111010110111

= new Scanner(System.in);

System.out.pri Cipher");

n

n

System.out.pri String plainte System.out.pri String key = r StreamCipher c

x

n

e

i

tln("This program will demonstrate

tln("Enter your message:"); t = reader.nextLine(); tln("Enter your key:"); ader.nextLine();

pher = new StreamCipher(key);

Stream

String ciphert String decrypt

e

e

xt = cipher.encryptBinary(plaintext); dtext = cipher.decryptBinary(ciphertext);

System.out.pri System.out.pri System.out.pri

n

n

n

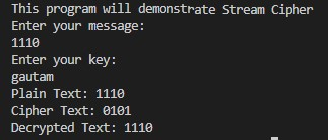
tln("Plain Text: " + plaintext); tln("Cipher Text: " + ciphertext); tln("Decrypted Text: " + decryptedtext);

reader.close()

;

}

}



# Questions:

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

ChaCha is the most widely used stream cipher in software due to its high speed in real-time applications. It is based on the Salsa20 cipher but it is improved. Google had selected ChaCha20 along with Bernstein's Poly1305 message authentication code in SPDY, which was intended as a replacement for TLS over TCP. ChaCha20 is also used for the arc4random random number generator in FreeBSD,[28] OpenBSD,[29] and NetBSD[30] operating systems. ChaCha20 usually offers better performance than the more prevalent Advanced Encryption Standard (AES) algorithm on systems where the CPU does not feature AES acceleration (such as the AES instruction set for x86 processors). As a result, ChaCha20 is sometimes preferred over AES in certain use cases involving mobile devices, which mostly use ARM-based CPUs.

# Outcomes: CO2- Illustrate different cryptographic algorithms for security Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

Understood the class of stream ciphers and the A5/1 keystream cipher. Wrote a small program to demonstrate the same.

# 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