# Experiment No : 10

**Aim :** Write a program to implement A3/A5/A8 GSM security algorithm.

**Theory :**

- GSM uses three different security algorithms called A3, A5, A8 are generally implemented together.
- An A3/A8 algorithm is implemented in SIM cards and in GSM networks authentication centres.
- It is used to authenticate customers and generate a key for encryption voice and data traffic.
- Development of A3 and A8 algorithms is considered a matter for individual GSM networks operators, although example implementations are available.
- An A5 encryption algorithm scrambles the user's voice and data traffic between the handset and the base station to provide privacy.
- An A5 algorithm is implemented in both the handset and the BSS.

**A3**

⇒

1) Authentication algorithm
2) Calculates SRES Based on the Kikey and RAND send by the MSC

3) Not standardized; can be chosen independently by each operator.

- A8

⇒

1) Key generation algorithm needed to calculate the session key Kc
2) Calculation of Kc depends on ki and RAND.
3) Not standardized; can be choosen independently by each operator

- A5

⇒

1) Stream cipher used to encrypt over the air transmissions.
2) Ciphering is based on Kc and the frame number.
3) Specified at international level to enable roaming.

- Conclusion:

A3/A5/A8 GSM security algorithms are implemented successfully.

Pair (A+)

**Code:**

A3

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;

public class Main {
    public static void main(String[] args) throws NoSuchAlgorithmException {
        long k = generateRandomKey();
        long m = generateRandomMessage();

        String sres = a3Algorithm(k, m);

        System.out.println("128-bit Secret Key (K in hexadecimal): " +
Long.toHexString(k));
        System.out.println("128-bit Random Message (M in hexadecimal): " +
Long.toHexString(m));
        System.out.println("RES/SRES (in hexadecimal): " + sres);
    }

    public static long generateRandomKey() {
        Random random = new Random();
        return random.nextLong();
    }

    public static long generateRandomMessage() {
        Random random = new Random();
        return random.nextLong();
    }

    public static String a3Algorithm(long k, long m) throws
NoSuchAlgorithmException {
        byte[] keyBytes = toByteArray(k);
        byte[] messageBytes = toByteArray(m);

        MessageDigest md = MessageDigest.getInstance("MD5");
        byte[] hash = md.digest(concatenateArrays(keyBytes, messageBytes));

        StringBuilder sres = new StringBuilder();
        for (byte b : hash) {
            sres.append(String.format("%02X", b));
        }
        return sres.toString();
    }

    public static byte[] toByteArray(long value) {
        byte[] result = new byte[16];
```

```
        for (int i = 0; i < 8; i++) {
            result[i] = (byte) (value >> (56 - i * 8));
        }
        return result;
    }

    public static byte[] concatenateArrays(byte[] a, byte[] b) {
        byte[] result = new byte[a.length + b.length];
        System.arraycopy(a, 0, result, 0, a.length);
        System.arraycopy(b, 0, result, a.length, b.length);
        return result;
    }
}
```

**Output:**

```
Run                                                    10s on 11:02:58, 03/19 ✓

128-bit Secret Key (K in hexadecimal): 75b0022a0ee84e0d
128-bit Random Message (M in hexadecimal): dc86fd9a7438fcba
RES/SRES (in hexadecimal): 098D641E383AA2B82780E37DD50A731D
```

**Code:**

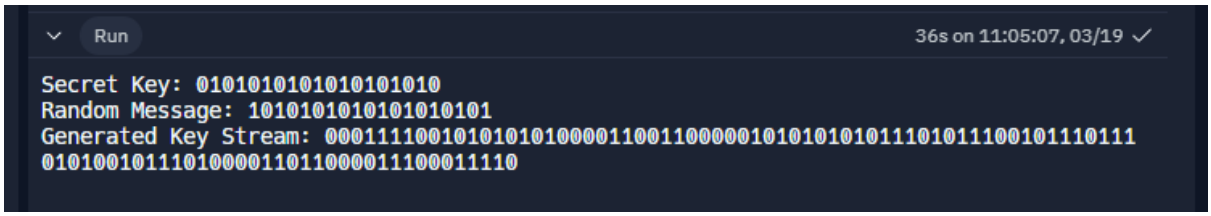
A5

```
public class Main {

  private int[] register1 = new int[19];
  private int[] register2 = new int[22];
  private int[] register3 = new int[23];

  public Main() {
    // Initialize registers with arbitrary values
    for (int i = 0; i < 19; i++) {
      register1[i] = 0;
    }
    for (int i = 0; i < 22; i++) {
      register2[i] = 0;
    }
    for (int i = 0; i < 23; i++) {
      register3[i] = 0;
    }
  }

  public void setKey(String key) {
    // Set the key for register 1
    for (int i = 0; i < 19; i++) {
```

```
      register1[i] = Character.getNumericValue(key.charAt(i % key.length()));
    }

    // Set the key for register 2
    for (int i = 0; i < 22; i++) {
      register2[i] = Character.getNumericValue(key.charAt((i + 19) %
key.length()));
    }

    // Set the key for register 3
    for (int i = 0; i < 23; i++) {
      register3[i] = Character.getNumericValue(key.charAt((i + 41) %
key.length()));
    }
  }

  public void generateKeyStream(int numBits) {
    for (int i = 0; i < numBits; i++) {
      int majority = (register1[8] & register2[10]) ^ (register1[8] &
register3[10]) ^ (register2[10] & register3[10]);
      int newBit = majority ^ register1[18] ^ register2[21] ^ register3[22];

      shiftRegister(register1);
      shiftRegister(register2);
      shiftRegister(register3);

      System.out.print(newBit);
    }
    System.out.println();
  }

  private void shiftRegister(int[] register) {
    int feedback = (register[13] ^ register[16] ^ register[17] ^ register[18])
& 0x01;
    for (int i = register.length - 1; i > 0; i--) {
      register[i] = register[i - 1];
    }
    register[0] = feedback;
  }

  public static void main(String[] args) {
    Main a5 = new Main();
    String key = "01010101010101010"; // Example key
    String randomMessage = "10101010101010101"; // Example random message
    a5.setKey(key);
    System.out.println("Secret Key: " + key);
    System.out.println("Random Message: " + randomMessage);
    System.out.print("Generated Key Stream: ");
```

```
    a5.generateKeyStream(100); // Generate 100 key bits
  }
}
```

**Output:**




**Code:**

A8

```
public class Main {

    // Constants
    private static final int LFSR_LENGTH = 22;
    private static final int[] INIT_STATE = {1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0};

    // Variables
    private int[] lfsr = new int[LFSR_LENGTH];

    // Constructor
    public Main() {
        // Initialize LFSR with initial state
        System.arraycopy(INIT_STATE, 0, lfsr, 0, LFSR_LENGTH);
    }

    // Clocking function
    private void clock() {
        int feedback = lfsr[0] ^ lfsr[1] ^ lfsr[2] ^ lfsr[8];
        for (int i = LFSR_LENGTH - 1; i > 0; i--) {
            lfsr[i] = lfsr[i - 1];
        }
        lfsr[0] = feedback;
    }

    // Generate key stream
    public int[] generateKeystream(int numBits) {
        int[] keystream = new int[numBits];
        for (int i = 0; i < numBits; i++) {
            keystream[i] = lfsr[LFSR_LENGTH - 1];
```

```
            clock();
        }
        return keystream;
    }

    public void printInitialState() {
        System.out.println("Initial State of LFSR:");
        for (int bit : lfsr) {
            System.out.print(bit);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Main a8 = new Main();
        a8.printInitialState();

        int[] keystream = a8.generateKeystream(100); // Generate 100 key bits
        System.out.println("Generated Key Stream:");
        for (int bit : keystream) {
            System.out.print(bit);
        }
    }
}
```

**Output:**

```
Initial State of LFSR:
10100100100100100010
Generated Key Stream:
0100100100100100100101100000111001100010100000011111111101110111000101110011111011000
1001001101011000
```