Experiment No 3: GSM Security

PART A

(PART A: TO BE REFFERED BY STUDENTS)

A.1 Aim: **To implement GSM security algorithms (A3/A5/A8)**

A.2 Objectives: To understand the security algorithms in mobile networks

A.3 Outcomes: Student will be able to implement security algorithms for mobile communication network. (LO-4)

A.4 Tools Used/programming language: Java, Python etc

A.5 Theory:

- Authentication verifies identity and validity of SIM card to the network and ensures that subscriber has access to the network.
- Term used
  - ✓ Ki= **individual subscriber authentication key**, it is 32 bit number and present only in SIM card and stored in authentication center.
  - ✓ RAND= **random 128 bit number generated by AUC** (authentication center) when network request to authenticate the subscribers.

- ✓ SRES (signed responses) = 32 bit crypto variable used in authentication process.
- ✓ Kc = 64 bit cipher key.

- MS is challenged by given RAND by the network.

▪ Security in GSM

- Three algorithms have been specified to provide security services in GSM. **Algorithm A3** is used for **authentication**, **A5** for **encryption**, and **A8** for the **generation of a cipher key**.

- In the GSM standard **only algorithm A5 was publicly available, whereas A3 and A8 were secret**, but standardized with open interfaces.

- **Network providers can use stronger algorithms for authentication**– or users can apply stronger end-to-end encryption.

- Algorithms A3 and A8 (or their replacements) are located on the SIM and in the AUC and can be proprietary.

- Only A5 which is implemented in the devices has to be identical for all providers.

Subscriber Authentication

For subscriber authentication algorithm used is A3

1. A3 algorithm is inbuilt inside SIM and AUC, Input for A3 is Ki and RAND
2. Ki=Stored inside SIM(kiis encrypted inside SIM card) and not share on network and also present in AUC of MSC.
3. Before a subscriber can use any service from the GSM network, he or she must be authenticated. Authentication is based on the SIM, which stores the individual **authentication key Ki**, the **user identification IMSI**, and the algorithm used for authentication **A3**.
4. When user want to access GSM network IMSI number from SIM send to MSC then HLR then to AUC.

5. Now AUC check IMSI number is present or not and identify associated Ki value (Ki is fixed), in this procedure AUC generate RAND number which is different for every new user request.

6. AUC using authentication algorithm A3(input to A3 are ki and RAND) calculate SRES as output of A3 and AUC using algorithm A8 of cipher generation (input to

   A8 are ki and RAND) calculate Kcand send these SRES, Kc and RAND to HLR then from HLR to MSC. These three terms SRES, Kc and RAND are called as triplet.

7. MSC now send only RAND value to MS

8. MS using algorithm A3 (input to A3 is Ki and RAND)calculate SRES and using algorithm A8 calculate Kc and send these SRES and kc to MSC

9. MSC check SRES receive from MS and Network are same or not. If both are same user is authenticated and connection is set up.

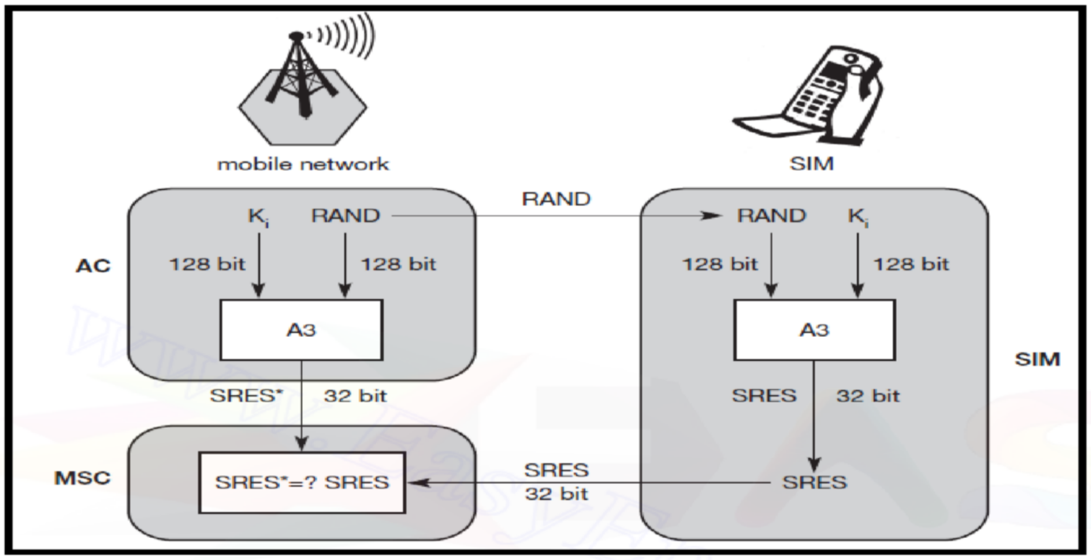


Figure: Subscriber Authentication

Encryption

1. To ensure privacy, all messages containing user-related information are encrypted in GSM over the air interface.

2. After authentication, MS and BSS can start using encryption by applying the cipher key Kc

3. Kc is generated using the individual key Ki and a random value by applying the algorithm A8. Note that the SIM in the MS and the network both calculate the same Kc based on the random value RAND. **The key Kc itself is not**

   transmitted over the air interface.
4. MS and BTS can now encrypt and decrypt data using the algorithm A5andthe cipher key Kc. As Figure shows, Kc should be a 64 bit key – which is not very strong, but is at least a good protection against simple eavesdropping. However, the publication of A3 and A8 on the internet showed that in certain implementations 10 of the 64 bits are always set to 0, **so that the real length of the key is thus only 54 consequently**, the encryption is much weaker.

5. Note: An **eavesdropping attack**, also known as a sniffing or snooping **attack**, is a theft of information as it is transmitted over a network by a computer, smart-phone, or another connected device. The **attack** takes advantage of unsecured network communications to access data as it is being sent or received by its user. **Eavesdropping** is the act of intercepting communications between two points.

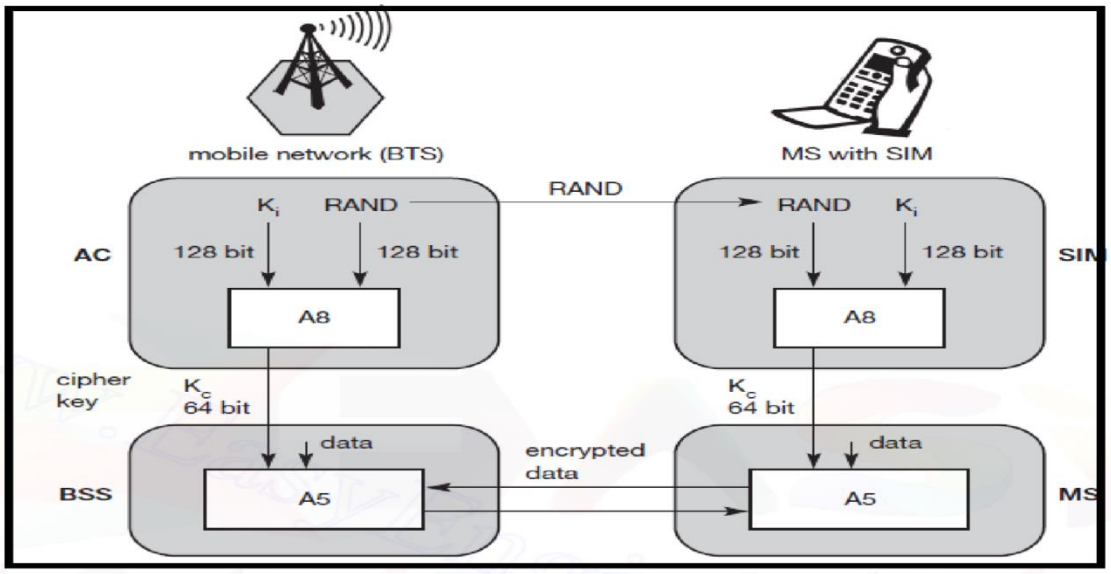


Figure: Data Encryption

A.6 Sample Source Code:

https://www.theprogrammingcodeswarehouse.com/2020/04/implementation-ofa3-security.html

```
import random
k=random.getrandbits(128)
m=random.getrandbits(128) kb=bin(k)[2:]
mb=bin(m)[2:] kbl=kb[0:64] kbr=kb[64:]
mbl=mb[0:64] mbr=mb[64:]
a1=int(kbl,2)^int(mbr,2)
a2=int(kbr,2)^int(mbl,2) a3=a1^a2
a4=bin(a3)[2:].zfill(64) a5=a4[0:32]
a6=a4[32:] a7=int(a5,2)^int(a6,2)
print("128 Bit Key = ",kb)
print("128 Random Bits Generated = ",mb) print("RES/SRES
= ",bin(a7)[2:].zfill(len(a5)))
```

A.6 Sample Output:

```
128 Bit Key
=11111011101001100100000100100110001001110011110100111010110100011110001110000001
111

011101110110111010100010110101000111010001


128 Random Bits Generated
=1100000100010001011000101110010011011010110011001000110101110001001000010100101
001

00000100111100000001000011001001111111000100


RES/SRES=11110110110100000010111110001101
```

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the ERP or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no ERP access available)

Roll No. B30	Name: Pranjal Bhatt
Class : TE COMPS B	Batch :B2
Date of Experiment:	Date of Submission
Grade :	

B.1 Question of Curiosity:

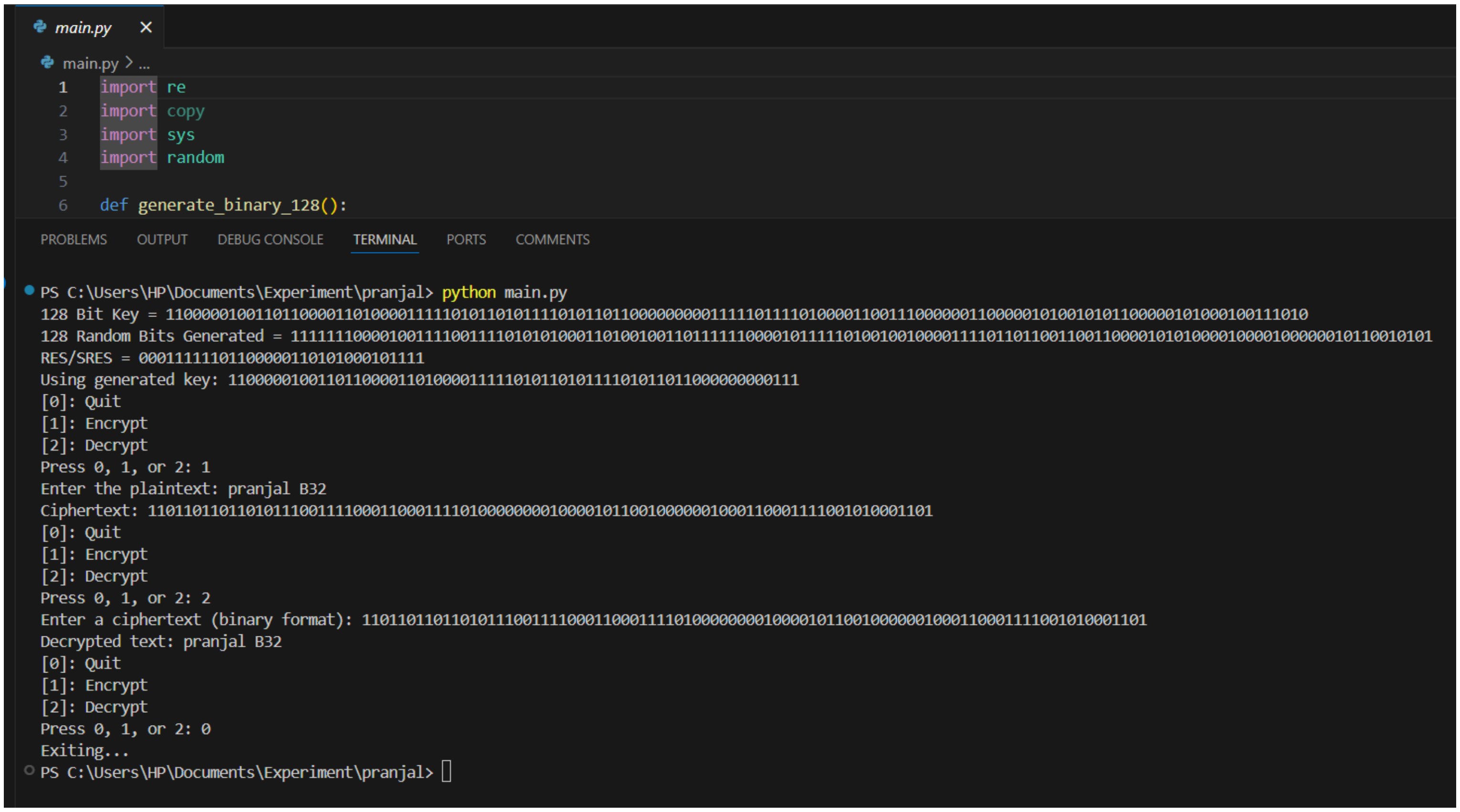
Q.1: Source Code (students need to implement GSM Security Algorithm using any programming language like Java, Python, etc)

```
import re
import copy
import sys
import random

def generate_binary_128():
   return bin(random.getrandbits(128))[2:].zfill(128)

# Generate random values
k = generate_binary_128()
m = generate_binary_128()

kbl, kbr = k[:64], k[64:]
mbl, mbr = m[:64], m[64:]
```

```
a1, a2 = int(kbl, 2), int(kbr, 2)
a3 = a1 ^ a2

a4 = bin(a3)[2:].zfill(64)
a5, a6 = a4[:32], a4[32:]
a7 = int(a5, 2)

print("128 Bit Key =", k)
print("128 Random Bits Generated =", m)
print("RES/SRES =", bin(a7)[2:].zfill(len(a5)))

# Register lengths
reg_x_length, reg_y_length, reg_z_length = 19, 22, 23

key_one = ""
reg_x, reg_y, reg_z = [], [], []

def loading_registers(key):
  global reg_x, reg_y, reg_z
  reg_x, reg_y, reg_z = [int(bit) for bit in key[:reg_x_length]], [int(bit) for bit in
key[reg_x_length:reg_x_length+reg_y_length]], [int(bit) for bit in key[reg_x_length+reg_y_length:]]

def set_key(key):
  global key_one
  if len(key) == 64 and re.match("^([01])+$", key):
    key_one = key
    loading_registers(key)
    return True
  return False

def get_majority(x, y, z):
  return 1 if (x + y + z) > 1 else 0

def get_keystream(length):
  reg_x_temp, reg_y_temp, reg_z_temp = reg_x[:], reg_y[:], reg_z[:]
  keystream = []

  for _ in range(length):
    majority = get_majority(reg_x_temp[8], reg_y_temp[10], reg_z_temp[10])
    if reg_x_temp[8] == majority:
      reg_x_temp.insert(0, reg_x_temp[13] ^ reg_x_temp[16] ^ reg_x_temp[17] ^ reg_x_temp[18])
      reg_x_temp.pop()
    if reg_y_temp[10] == majority:
      reg_y_temp.insert(0, reg_y_temp[20] ^ reg_y_temp[21])
```

```
            reg_y_temp.pop()
        if reg_z_temp[10] == majority:
            reg_z_temp.insert(0, reg_z_temp[7] ^ reg_z_temp[20] ^ reg_z_temp[21] ^ reg_z_temp[22])
            reg_z_temp.pop()
        keystream.append(reg_x_temp[18] ^ reg_y_temp[21] ^ reg_z_temp[22])

    return keystream

def to_binary(plain):
    return ''.join(format(ord(x), '08b') for x in plain)

def convert_binary_to_str(binary):
    return ''.join(chr(int(binary[i:i+8], 2)) for i in range(0, len(binary), 8))

def encrypt(plain):
    binary = to_binary(plain)
    keystream = get_keystream(len(binary))
    return ''.join(str(int(binary[i]) ^ keystream[i]) for i in range(len(binary)))

def decrypt(cipher):
    keystream = get_keystream(len(cipher))
    binary = ''.join(str(int(cipher[i]) ^ keystream[i]) for i in range(len(cipher)))
    return convert_binary_to_str(binary)

def main_fun():
    key = kbl
    set_key(key)
    print("Using generated key:", key)

    while True:
        choice = input("[0]: Quit\n[1]: Encrypt\n[2]: Decrypt\nPress 0, 1, or 2: ")
        if choice == '0':
            print("Exiting...")
            sys.exit(0)
        elif choice == '1':
            plaintext = input("Enter the plaintext: ")
            encrypted_text = encrypt(plaintext)
            print(f"Ciphertext: {encrypted_text}")
        elif choice == '2':
            while True:
                cipher = input("Enter a ciphertext (binary format): ")
                if re.match("^([01])+$", cipher):
                    decrypted_text = decrypt(cipher)
                    print(f"Decrypted text: {decrypted_text}")
                    break
                else:
                    print("Invalid ciphertext. Must be in binary format.")

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

Q.2: Output of GSM Security Algorithm



```
main.py
import re
import copy
import sys
import random

def generate_binary_128():
```

```
PS C:\Users\HP\Documents\Experiment\pranjal> python main.py
128 Bit Key = 11000001001101100001101000011111010110101111010110110000000000111110111101000011001110000001100000101001010110000010100010011101 0
128 Random Bits Generated = 11111110000100111100111101010100011010010011011111110000101111101001001000011110110110011001100001010100001000010000000101100101 01
RES/SRES = 00011111101100000110101000101111
Using generated key: 1100000100110110000110100001111101011010111101011011000000000111
[0]: Quit
[1]: Encrypt
[2]: Decrypt
Press 0, 1, or 2: 1
Enter the plaintext: pranjal B32
Ciphertext: 11011011011010111001111000110001111010000000001000010110010000000100011000111100101000110 1
[0]: Quit
[1]: Encrypt
[2]: Decrypt
Press 0, 1, or 2: 2
Enter a ciphertext (binary format): 11011011011010111001111000110001111010000000001000010110010000000100011000111100101000110 1
Decrypted text: pranjal B32
[0]: Quit
[1]: Encrypt
[2]: Decrypt
Press 0, 1, or 2: 0
Exiting...
PS C:\Users\HP\Documents\Experiment\pranjal> 
```

Q.3: List out various elements of GSM architecture and explain in brief function of each element.

The **Global System for Mobile Communications (GSM)** architecture consists of several key components categorized into different subsystems:

1. Mobile Station (MS)

- **Components**: Mobile handset (phone) and SIM (Subscriber Identity Module).
- **Function**:
    - The mobile device allows users to make and receive calls, send SMS, and use data services.

  - The SIM stores subscriber-related information, including IMSI (International Mobile Subscriber Identity), authentication key, and network-specific data.

2. Base Station Subsystem (BSS)

- **Components**: Base Transceiver Station (BTS) and Base Station Controller (BSC).
- **Functions**:
  - **BTS**:
    - Handles radio communication with mobile stations.
    - Manages frequency allocation and radio signal processing.
  - **BSC**:
    - Controls multiple BTSs.
    - Manages handovers, power control, and frequency hopping.

3. Network and Switching Subsystem (NSS)

- **Components**:
  - **Mobile Switching Center (MSC)**
  - **Home Location Register (HLR)**
  - **Visitor Location Register (VLR)**
  - **Authentication Center (AuC)**
  - **Equipment Identity Register (EIR)**
- **Functions**:
  - **MSC**:
    - Central switch for call setup, routing, and handovers.
    - Connects GSM network to the Public Switched Telephone Network (PSTN).
  - **HLR**:
    - Stores permanent subscriber information (IMSI, service plans, location, etc.).
  - **VLR**:
    - Temporary database storing subscriber information while in a specific location.
  - **AuC**:
    - Provides security by authenticating users and generating encryption keys.
  - **EIR**:

- Stores information about mobile devices to prevent unauthorized access (blacklist/whitelist database).

4. Operation and Support Subsystem (OSS)

- **Components**:
  - **Operation and Maintenance Center (OMC)**
- **Function**:
  - Manages network operations, including monitoring performance, fault detection, and configuration management.

B.2 Conclusion:

This experiment demonstrates the implementation of a **stream cipher encryption technique** using a **keystream generator** based on a **linear feedback shift register (LFSR)** approach. The process involves:

1. **Random Key Generation**: A 128-bit key is randomly generated and divided into two 64-bit halves, with bitwise XOR operations performed to derive intermediate values for encryption.
2. **Register Loading and Keystream Generation**: Three shift registers (X, Y, Z) are initialized and updated dynamically based on the majority function to produce a keystream.
3. **Encryption & Decryption**:
   1. The plaintext is converted into binary and XORed with the generated keystream to produce ciphertext.
   2. The decryption process reverses the encryption by XORing the ciphertext with the same keystream, successfully retrieving the original message.