Experiment No. 3

Aim: To implement security algorithm of GSM in python.

Requirements: Compatible version of python.

Theory:

GSM is the most secured cellular telecommunications system available today. GSM has its security methods standardized. GSM maintains end-to-end security by retaining the confidentiality of calls and anonymity of the GSM subscriber.

Temporary identification numbers are assigned to the subscriber’s number to maintain the privacy of the user. The privacy of the communication is maintained by applying encryption algorithms and frequency hopping that can be enabled using digital systems and signalling.

Mobile Station Authentication:

The GSM network authenticates the identity of the subscriber through the use of a challenge-response mechanism. A 128-bit Random Number (RAND) is sent to the MS. The MS computes the 32-bit Signed Response (SRES) based on the encryption of the RAND with the authentication algorithm (A3) using the individual subscriber authentication key (Ki). Upon receiving the SRES from the subscriber, the GSM network repeats the calculation to verify the identity of the subscriber.

The individual subscriber authentication key (Ki) is never transmitted over the radio channel, as it is present in the subscriber's SIM, as well as the AUC, HLR, and VLR databases. If the received SRES agrees with the calculated value, the MS has been successfully authenticated and may continue. If the values do not match, the connection is terminated and an authentication failure is indicated to the MS.

The calculation of the signed response is processed within the SIM. It provides enhanced security, as confidential subscriber information such as the IMSI or the individual subscriber authentication key (Ki) is never released from the SIM during the authentication process.

Signalling and Data Confidentiality:

The SIM contains the ciphering key generating algorithm (A8) that is used to produce the 64-bit ciphering key (Kc). This key is computed by applying the same random number (RAND) used in the authentication process to ciphering key generating algorithm (A8) with the individual subscriber authentication key (Ki).

GSM provides an additional level of security by having a way to change the ciphering key, making the system more resistant to eavesdropping. The ciphering key may be changed at regular intervals as required. As in case of the authentication process, the computation of the ciphering key (Kc) takes place internally within the SIM. Therefore, sensitive information such as the individual subscriber authentication key (Ki) is never revealed by the SIM.

Encrypted voice and data communications between the MS and the network is accomplished by using the ciphering algorithm A5. Encrypted communication is initiated by a ciphering mode request command from the GSM network. Upon receipt of this command, the mobile station begins encryption and decryption of data using the ciphering algorithm (A5) and the ciphering key (Kc).

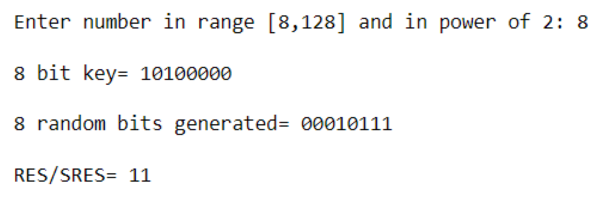
Subscriber Identity Confidentiality:

To ensure subscriber identity confidentiality, the Temporary Mobile Subscriber Identity (TMSI) is used. Once the authentication and encryption procedures are done, the TMSI is sent to the mobile station. After the receipt, the mobile station responds. The TMSI is valid in the location area in which it was issued. For communications outside the location area, the Location Area Identification (LAI) is necessary in addition to the TMSI.

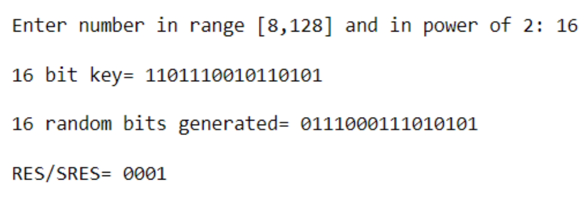
Code:

```
import random
import math

def encryption(data_size):
    assert 8<= data_size <= 128, "Data size should be in range [8,128]"
    assert data_size & (data_size-1) == 0, "Data size must be in power of two"

    key, data = [bin(random.getrandbits(data_size))[2:].zfill(data_size) for _ in range(2)]
    key_left, key_right = key[:data_size//2], key[data_size//2:]
    data_left, data_right = data[:data_size//2], data[data_size//2:]
    a1, a2 = int(key_left,2)^int(data_right,2),int(key_right,2)^int(data_left,2)
    a3 = a1^a2
    a4 = bin(a3)[2:].zfill(data_size//2)
    a5, a6 = a4[:data_size//4], a4[data_size//4:]
    a7 = int(a5,2)^int(a6,2)

    return key, data, bin(a7)[2:].zfill(data_size//4)

data_size = int(input("Enter number in range [8,128] and in power of 2: "))
print()
try:
    key, random_bits, res_sres_ratio = encryption(data_size)
    print(f"{data_size} bit key= {key}\n\n{data_size} random bits generated=
{random_bits}\n")
    print(f"RES/SRES= {res_sres_ratio}")

except AssertionError as error:
    print(error)
```

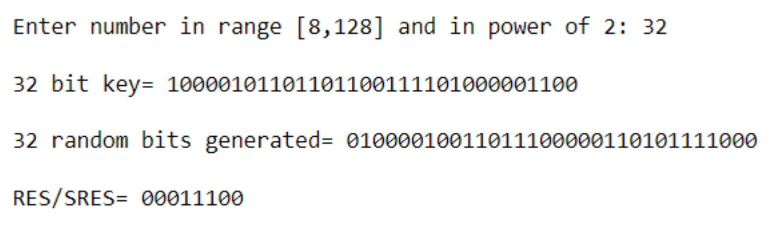
Output:



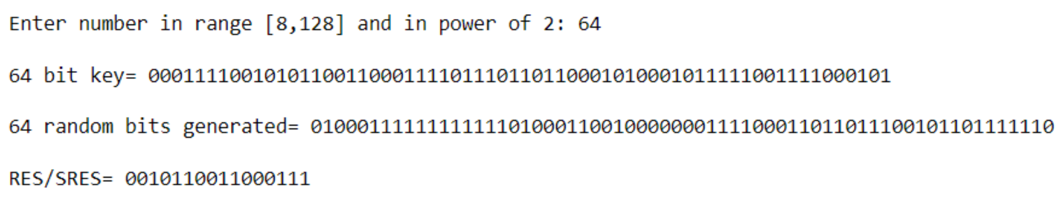
```
Enter number in range [8,128] and in power of 2: 8

8 bit key= 10100000

8 random bits generated= 00010111

RES/SRES= 11
```



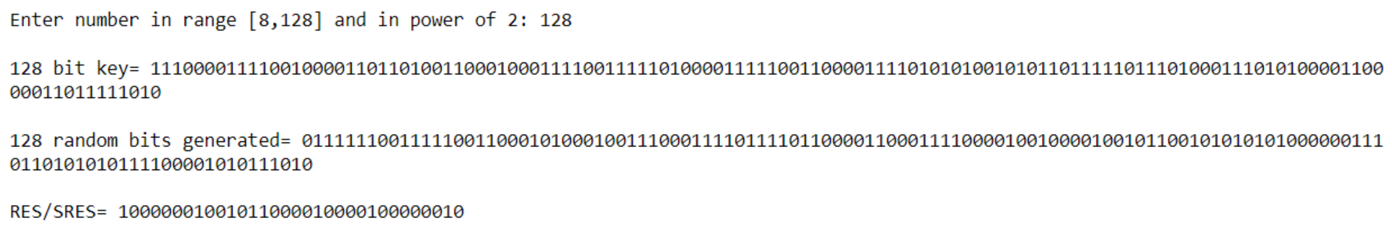
```
Enter number in range [8,128] and in power of 2: 16

16 bit key= 1101110010110101

16 random bits generated= 0111000111010101

RES/SRES= 0001
```



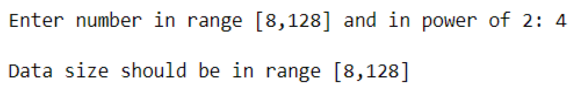
```
Enter number in range [8,128] and in power of 2: 32

32 bit key= 10000101101101100111101000001100

32 random bits generated= 01000010011011100000110101111000

RES/SRES= 00011100
```



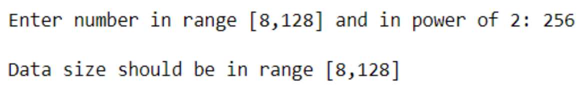
```
Enter number in range [8,128] and in power of 2: 64

64 bit key= 0001111001010110011000111101110110110001010001011111001111000101

64 random bits generated= 0100011111111111101000110010000000111100011011011100101101111110

RES/SRES= 0010110011000111
```



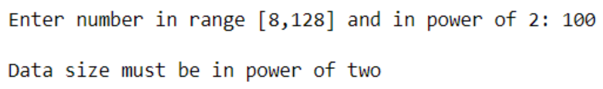
```
Enter number in range [8,128] and in power of 2: 128

128 bit key= 111000011110010000110110100110001000111100111110100001111100110000111101010100101011011111011101000111010100001100
00011011111010

128 random bits generated= 01111110011111100110001010001001110001111011110110000110001111000010010000100101100101010101000000111
0110101010111100001010111010

RES/SRES= 10000001001011000010000100000010
```



```
Enter number in range [8,128] and in power of 2: 4

Data size should be in range [8,128]
```



```
Enter number in range [8,128] and in power of 2: 256

Data size should be in range [8,128]
```



```
Enter number in range [8,128] and in power of 2: 100

Data size must be in power of two
```

Conclusion: We have successfully implemented GSM security algorithm in python.