**SNS Assignment 2, Execution Document**

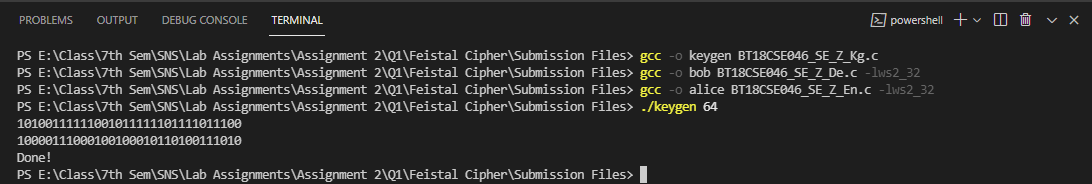
This document contains the commands required to run the program for each of the 5 questions present in the Practicals 2 document

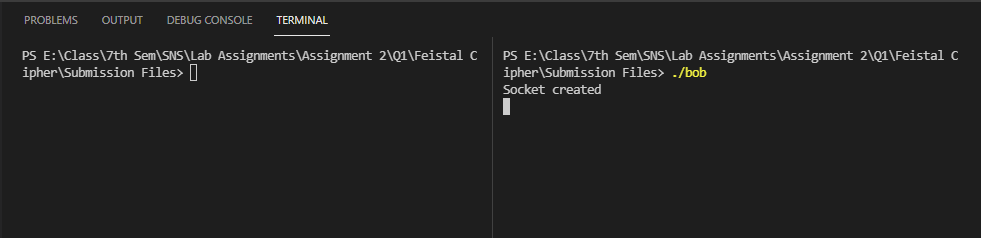
Roll Number: BT18CSE**046**

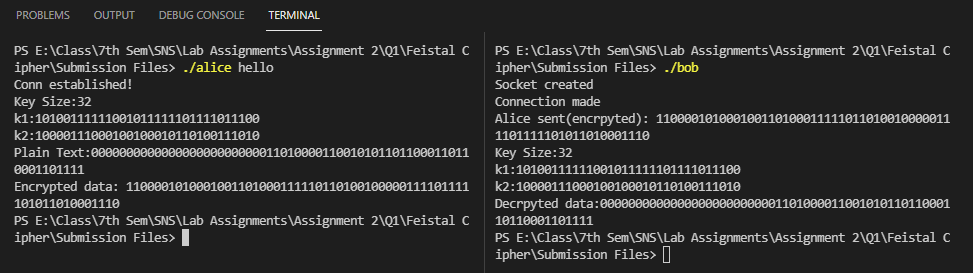
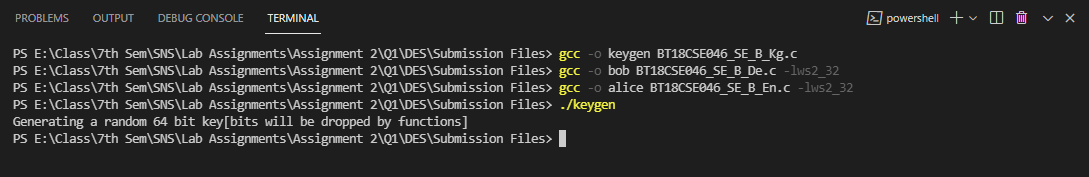
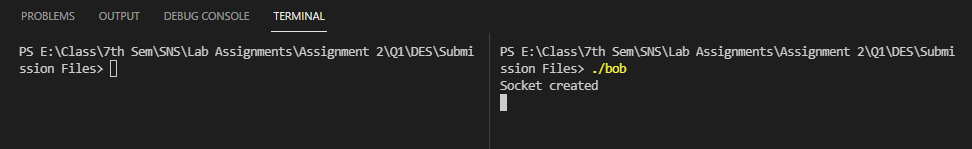
Question Set: B

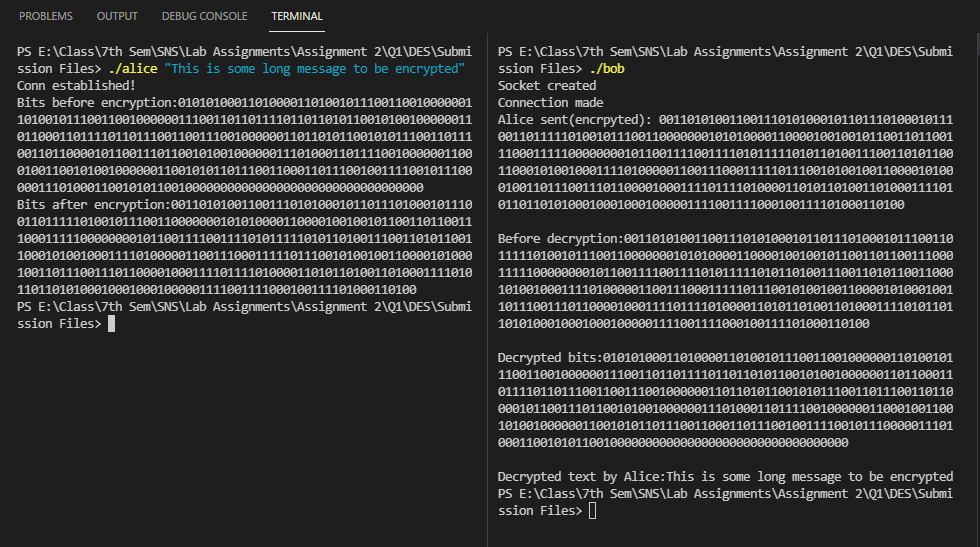
Socket programming is done in C, specifically using winsock2 library (will work on windows only)

**1.** **Symmetric Cryptosystem**

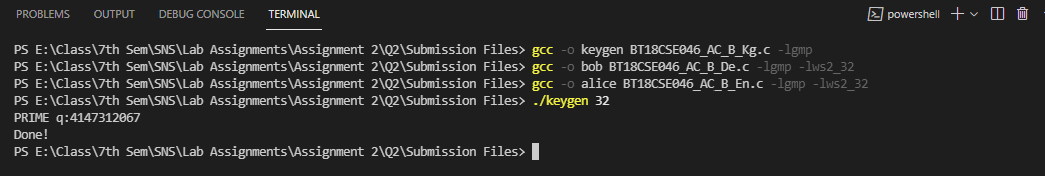
* **Z** : Implement 2-round Feistel cipher. Suitably, choose your own function f, such that f uses your roll no last two digits in its computation. (2.5 M)
  + **Files provided** : BT18CSE046\_SE\_Z\_Kg.c, BT18CSE046\_SE\_Z\_En.c, BT18CSE046\_SE\_Z\_De.c
  + **Compilation commands**:
    - Keygen program: gcc -o keygen BT18CSE046\_SE\_Z\_Kg.c
    - Encryption program: gcc -o alice BT18CSE046\_SE\_Z\_En.c -lws2\_32
    - Decryption program: gcc -o bob BT18CSE046\_SE\_Z\_De.c -lws2\_32
  + **Execution Steps:**
    - Generate the required key using : ./keygen.exe <key\_size>
      * Where the key\_size for the feistal cipher is assumed to be the sum of the 2 round key sizes, k1 and k2 (k1+k2)
      * keys.txt will be created
    - Start the receiver program(decryptor) in listening mode: ./bob
    - Start the encryption program with command line input of message to be encrypted: ./alice <message>
      * Where message is the message to be encrypted and sent over sockets to the decryption program.
  + **Sample Run:**
    - Compiling and key generation
    - As apparent above, 2 32 bit keys have been created when input was 64 bit key( design choice for keygen)
    - Execution

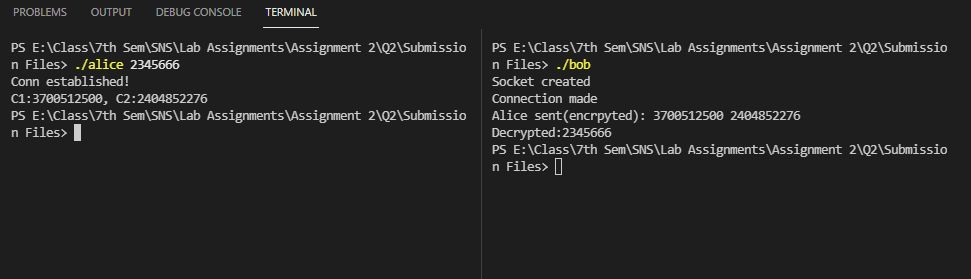
Start decryption program in listening mode before running the encryption program

* + - As we can see in the above ss, the plain text before and after encryption match ( bits output in ss )
    - The input data was padded to match 64 bits
* **B**: DES CFB mode (2.5 m)
  + **Files Provided**: BT18CSE046\_SE\_B\_Kg.c, BT18CSE046\_SE\_B\_En.c, BT18CSE046\_SE\_B\_De.c
  + **Compilation Instructions:**
    - Keygen: gcc -o keygen BT18CSE046\_SE\_B\_Kg.c
    - Decryption Program: gcc -o bob BT18CSE046\_SE\_B\_De.c -lws2\_32
    - Encryption Program: gcc -o alice BT18CSE046\_SE\_B\_En.c -lws2\_32
  + **Execution steps:**
    - Keygen: ./keygen
    - Run decryption program: ./bob
    - Run encryptor with command line args: ./alice <message>
      * Where message is to be encrypted
  + **Sample run:**
    - Compilation and key generation:
    - This generates the des 64 bit key( which will automatically be reduced to 56 by functions) in file called key.txt
    - Start the decryption program in listening mode
    -  Now finally run encryption program
    - As we can see, the output is correctly decrypted

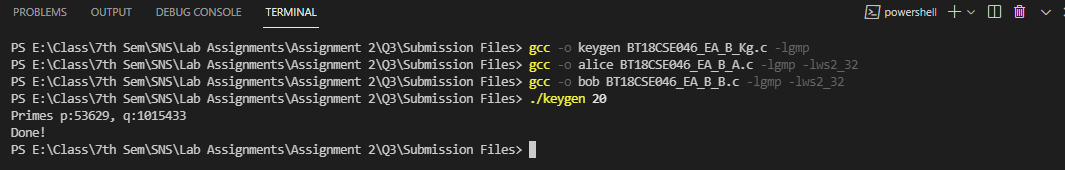
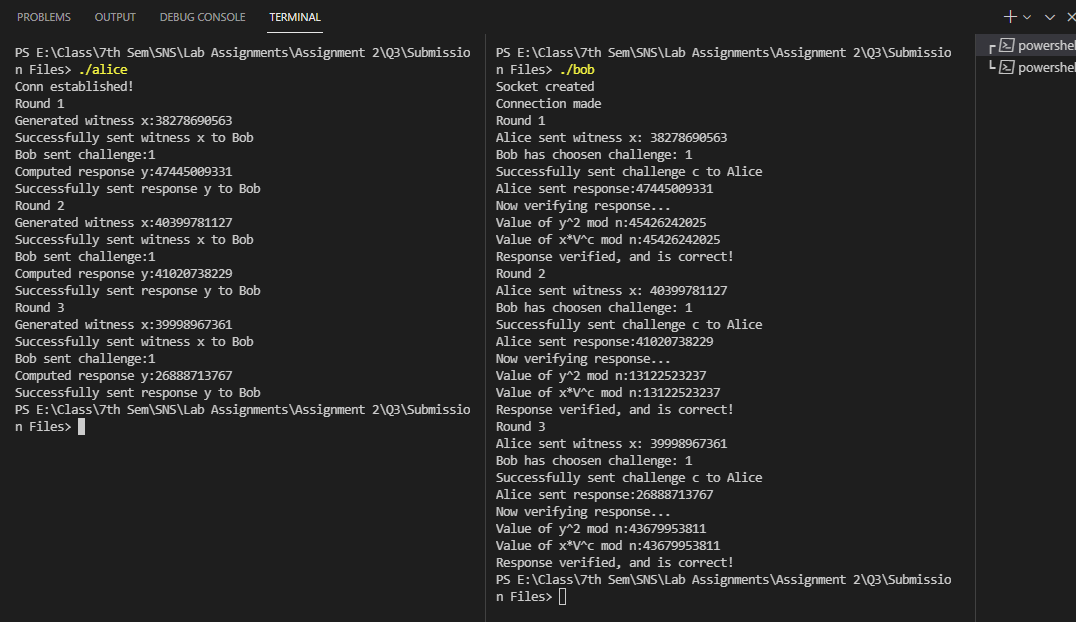


**2. Asymmetric Cryptosystem**

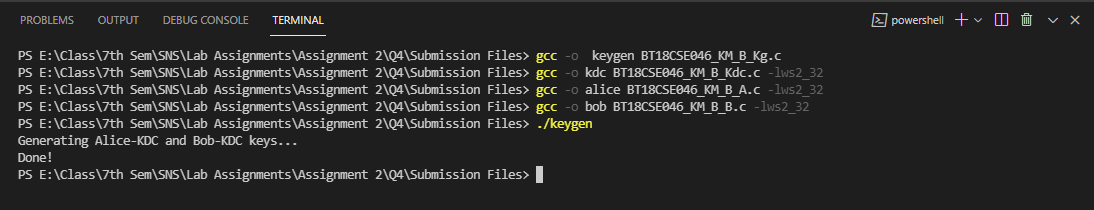
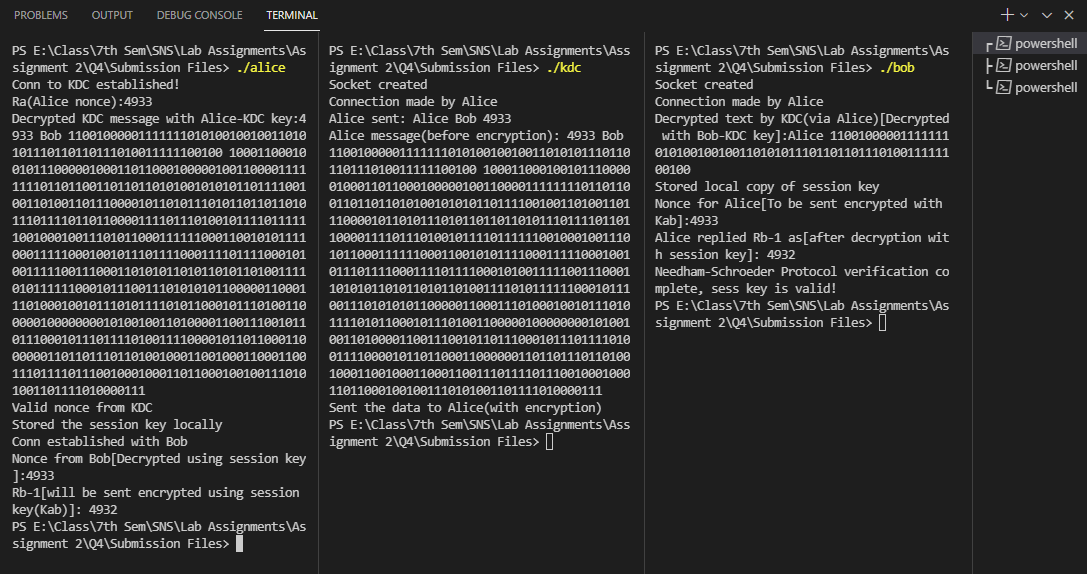
* **B**: Elgamal (5m)
  + **Files Provided**: BT18CSE046\_AC\_B\_Kg.c, BT18CSE046\_AC\_B\_De.c, BT18CSE046\_AC\_B\_En.c
  + **Compilation Instructions**:
    - **Keygen**: gcc -o keygen BT18CSE046\_AC\_B\_Kg.c -lgmp
    - Decryption Program: gcc -o bob BT18CSE046\_AC\_B\_De.c -lgmp -lws2\_32
    - Encryption Program: gcc -o alice BT18CSE046\_AC\_B\_En.c -lgmp -lws2\_32
  + **Execution steps**:
    - Generate keys: ./keygen <key\_size>
      * Where key\_size is the given as k, and the primes for elgamal will be chosen from 1^k ( 111…1 k times)
    - Run decryption program: ./bob
    - Run encryption program with command-line arguments: ./alice <message>
      * Where message is and integer base-10 from 1 to chosen prime q-1
  + **Samp**l**e run:**
    - Compiling and keygen:
    - Generates both public keys and secret keys( PK.txt and SK.txt resp.)
    - Run ./bob, wait for Socket created
    - Run ./alice <message>



**3. Entity Authentication**

* **B:** Fiat-Shamir Protocol ( 5m )
  + **Files Provided:** BT18CSE046\_EA\_B\_Kg.c, BT18CSE046\_EA\_B\_B.c(Bob), BT18CSE046\_EA\_B\_A.c(Alice)
  + **Compilation Instructions:**
    - Keygen: gcc -o keygen BT18CSE046\_EA\_B\_Kg.c -lgmp
    - Entity with secret(Alice): gcc -o alice BT18CSE046\_EA\_B\_A.c -lgmp -lws2\_32
    - Entity who wants to verify the authenticity(Bob): gcc -o bob BT18CSE046\_EA\_B\_B.c -lgmp -lws2\_32
  + **Execution Steps:**
    - Generate keys using : ./keygen.exe <key\_size>
      * Similar to elgamal
    - Run bob in listening mode: ./bob
    - Run alice: ./alice
      * We run the fiat-shamir protocol for 3 times ( in a loop, with different witness every time by Alice)
  + **Sample Run:**
    - Compilation and keygen:
    - Run ./bob, in listening mode
    - Run ./alice
    - As we can see in the above ss, all three rounds were completed successfully

**4. Key Management**

* **B:** Needham-Schroeder Protocol ( 5m )
  + **Files Provided:** BT18CSE046\_KM\_B\_Kg.c, BT18CSE046\_KM\_B\_Kdc.c, BT18CSE046\_KM\_B\_A.c, BT18CSE046\_KM\_B\_B.c
  + **Compilation Instructions:**
    - Keygen(for Alice-KDC and Bob-KDC keys): gcc -o keygen BT18CSE046\_KM\_B\_Kg.c
    - KDC: gcc -o kdc BT18CSE046\_KM\_B\_Kdc.c -lws2\_32
    - Alice: gcc -o alice BT18CSE046\_KM\_B\_A.c -lws2\_32
    - Bob: gcc -o bob BT18CSE046\_KM\_B\_B.c -lws2\_32
  + **Execution Steps:**
    - Run keygen( generates des keys ): ./keygen
    - Run bob in listening mode: ./bob
    - Run kdc in listening mode: ./kdc
    - Run alice to make session request: ./alice
  + **Sample Run:**
    - Compilation and keygen:
    - Run Bob on 1 terminal
    - Run kdc on another terminal, wait for Socket created
    - Run alice
    - Alice stores her key in a file called Kab\_Alice\_copy.txt
    - Bob stores his copy of the session key(generated by kdc, sent by alice, encrypted by kdc using kdc-bob key) as Kab\_Bob\_copy.txt
    - In the above ss, it is apparent that the protocol has been successfully executed