**MSA University** 

**Faculty of Computer Science**

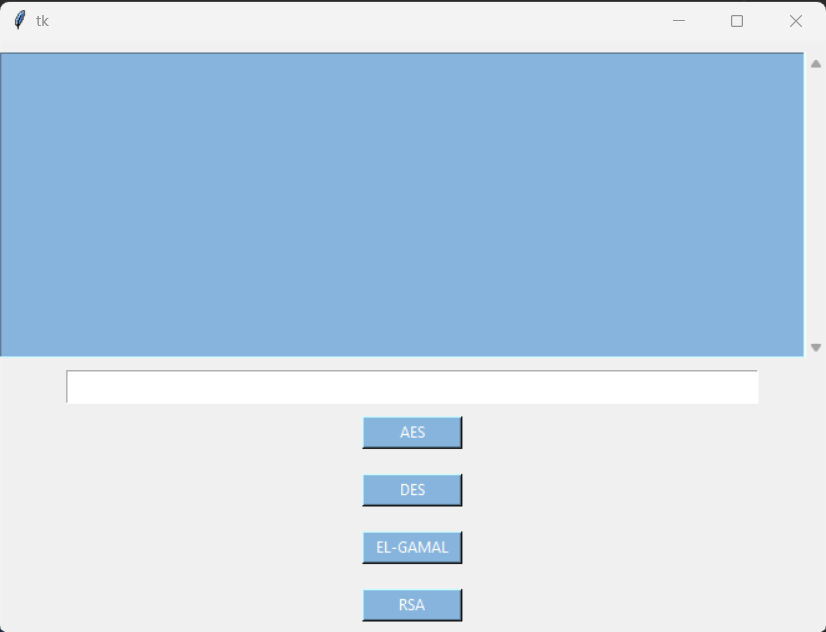
**CS401: Computer Security**

**Project**

**Delivered to: Dr. Ehab Emam**

|  |  |  |
| --- | --- | --- |
| **Name** | **ID** | **Roles** |
| Elshaymaa Atef | 203029 | RSA, GUI, Socket for RSA |
| Dema Mohamed | 200237 | ElGamal, Socket for ElGamal |
| Abdallah Mohamed Abdalla | 200879 | AES, Socket for AES, TTP |
| Kareem Sabry | 205027 | DES, Socket for DES |

### 



**ElGamal**

The prime\_gen() function generates a random prime number q between 2 and 1000. The function is\_prime() is used to check if the number is prime or not. If the number is prime, the function returns q.

The greatest\_comm\_divisor() function computes the greatest common divisor of two numbers using the Euclidean algorithm.

The generate\_key() function generates a random number key between 2 and q. It checks whether the greatest common divisor of key and q is equal to 1, and repeats the process until it finds a suitable value of key.

The power() function computes (a^b) % c using the binary exponentiation algorithm.

The encryption\_elgamal() function encrypts a given message msg using the ElGamal algorithm. It first calls the starter() function to generate the required parameters such as key, q, h and g. It generates a random number k using the generate\_key() function and computes s and p using the power() function. It then encrypts each character in the message by multiplying it with s. The encrypted message along with the other required parameters are combined into a string using the join() function, and returned as the encrypted ciphertext.

The decryption\_elgamal() function decrypts a given ciphertext string\_list using the ElGamal algorithm. It first splits the input string into its constituent parameters using the split() function. It then computes h using key, q and p, and uses it to decrypt each character in the ciphertext. The decrypted characters are combined into a string, which is returned as the original plaintext.

Finally, the starter() function generates the required parameters such as key, q, h and g for the ElGamal algorithm. It generates a random prime number q using the prime\_gen() function, and a random number g between 2 and q. It also generates a random number key between 2 and q, and computes h using g, key and q. The function returns key, q, h and g.

**RSA**

The program starts by defining two functions, `GCD(x,y)` and `invMod(e,eular)`, used in the RSA algorithm. The `GCD` function calculates the greatest common divisor of two numbers `x` and `y`, while the `invMod` function calculates the modular inverse of `e` with respect to `eular` using a brute-force approach.

The program then generates two random prime numbers `p` and `q` between 100 and 200. It uses a brute-force method to check if a number is prime or not. It checks if any number `x` from 2 to `z` (where `z` is the generated random number) divides `z`. If it does, then `z` is not a prime number, and it checks the next number. If no number from 2 to `z` divides `z`, then `z` is a prime number.

After generating the prime numbers `p` and `q`, the program uses them to calculate `n` as the product of `p` and `q`. It then calculates `eular` as `(p-1)\*(q-1)` and chooses a random number `e` between 1 and `eular` such that `e` is coprime with `eular`, i.e., `GCD(e, eular) = 1`. It then calculates the modular inverse `d` of `e` with respect to `eular` using the `invMod` function.

The program then defines two main functions `cipherRSA(msg)` and `decipherRSA(cipher)` that are used for encryption and decryption of messages using the RSA algorithm. The `cipherRSA(msg)` function takes a message `msg` as input and returns the cipher text as a string. It first calculates `C` as a list of integers, where each integer is the encrypted value of each character in the message `msg`. It then creates a list `biglist` that contains `C`, `d`, and `n` as elements and converts it to a string using the `join` function. It returns the string.

The `decipherRSA(cipher)` function takes a ciphertext `cipher` as input and returns the decrypted message as a string. It first splits the input string into a list `output\_list` and converts the first element of `output\_list` to a list of integers `ciphered`. It then extracts the values of `d` and `n` from the `output\_list`. It calculates the decrypted value of each integer in `ciphered` using the private key `d` and `n` and converts it to a character using the `chr` function. It finally joins all the characters to form the original message and returns it as a string.

**DES**

1. The input text is a binary string of length 64, and it is processed in 16 rounds.

2. The input key is also a binary string of length 64.

3. The input text is first permuted using a fixed permutation table ip, and then split into two halves of 32 bits each.

4. Each round consists of the following steps:

- The right half of the previous round becomes the left half of the current round.

- The right half of the previous round is permuted using another fixed permutation table ep.

- The result is XORed with the current round key, which is generated from the original key using a series of fixed permutation tables and shift operations.

- The result is then divided into eight 6-bit blocks, which are used to look up values in eight S-boxes.

- The output of the S-boxes is concatenated into a 32-bit block.

- The 32-bit block is permuted using another fixed permutation table pt.

- The result is XORed with the left half of the previous round to produce the right half of the current round.

5. After 16 rounds, the two halves are concatenated and permuted using a final fixed permutation table ivp to produce the final ciphertext.

**AES**

The `SubNib()` function performs a substitution operation on a nibble (four bits) by mapping it to another nibble using a fixed S-box.

The `RotNib()` function performs a rotation operation on a nibble by circularly shifting its bits to the left.

The `ShiftRows()` function performs a shifting operation on the rows of the state matrix, where each row is shifted cyclically by a certain number of bytes.

The `InverseSubNib()` function performs the inverse operation of the `SubNib()` function.

The `string\_to\_binary()` function converts a string into its binary representation by converting each character into its ASCII code and then converting the ASCII code into an 8-bit binary value.

The `binary\_to\_string()` function converts a binary string into its corresponding string representation by converting each 8-bit binary value into its corresponding ASCII character.

The `GF()` function performs multiplication in the Galois field GF(2^8), which is used in the MixColumns operation in the AES algorithm.

The main function `Decrypt()` takes two parameters, the cipher text as a list of bytes, and the key as a string of bits. The key is then split into 4 words of 32 bits each. These words are used to derive 3 subkeys, which are used for each of the 10 rounds of decryption.

The `Decrypt()` function then uses the derived subkeys to perform the rounds of decryption. It starts with the last subkey and performs the inverse operations used in the encryption, which are the inverse substitution, inverse permutation, and XOR operations. It then moves to the second-to-last subkey and repeats the same operations, and so on, until it reaches the first subkey, which is only used for the final XOR operation. Finally, the decrypted plaintext is returned as a list of bytes.

**Socket**

A socket server with two clients in Python works as follows:

1. The server is created and started by calling the socket() function from the socket module. This creates a socket object which is then bound to an address (IP address and port number) using the bind() method. Finally, the listen() method is called to make the server listen for incoming connections.
2. The two clients connect to the server by creating their own socket objects and calling the connect() method with the server's address. This establishes a TCP connection between the client and the server.
3. Once the connection is established, the clients can send and receive data to and from the server using the send() and recv() methods of the socket object. These methods send and receive data as bytes-like objects.
4. The server can handle multiple clients by creating a new thread or process for each incoming connection. When a client connects to the server, the server creates a new thread or process to handle the client's requests. This allows the server to handle multiple clients simultaneously.
5. The communication between the clients can be done by passing data through the server. For example, if client 1 wants to send a message to client 2, it can send the message to the server. The server can then forward the message to client 2. This can be done using a simple protocol where each message includes the recipient's address (IP address and port number) and the message data.
6. When the clients are finished communicating, they can close their connections by calling the close() method of their socket objects. The server can also be shut down by calling the close() method of its socket object.

In summary, a socket server with two clients works by establishing a TCP connection between the clients and the server, allowing them to send and receive data. The server can handle multiple clients by creating a new thread or process for each incoming connection. The communication between the clients can be done by passing data through the server.