CRYPTOGRAPHY

PROJECT REPORT

GROUP 12 | CHAT APPLICATION

SUMMARY

In this insecure digital generation, there is a lot of importance for individual privacy.

We will try to mold our system to cope up with existing integrity and encryption chat systems by improving our encryption standards, implementing new hashing algorithms for giving out enhanced performance from time to time.

Our chat application is Encrypted with AES and Data-integrity is maintained with CRC hash code there are 4 function used i.e.,
Encrypt (), Decrypt (), CRC (),stob (),btos ().

To start the conservation between Systems we used Socket programming which allow us to connect between two systems within a single Access-point or same network.

INTRODUCTION

OBJECTIVE

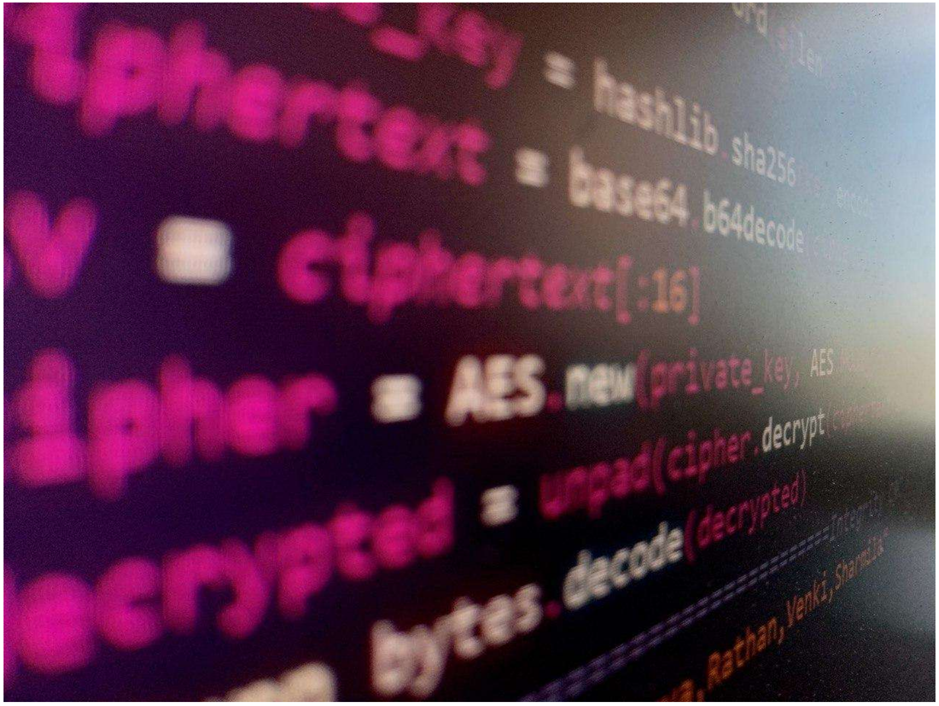
Our main objective is to provide “Data – Integrity” and “Data-Security” for the data that is transmitted between “Sender” and “Receiver”.

PROBLEM

Whenever, a sensitive information or data is transferred among sender and the receiver, the information sent can be Read, modified, or deleted completely by any malicious attackers. Also, the problem comes with the attackers who would be able to decode the message thereby, no data security.

SOLUTION

- ➔ To ensure “data-security” the message is encrypted using AES algorithm. AES -Encryption is considered as the most secure algorithm which requires very standards of computational power to decode the algorithm.
- ➔ To maintain “data integrity”, Simple technique called CRC (Cyclic Redundancy Check) is used. To check any active modifications the system computes CRC mathematically and append it to the data.



```
_key = hashlib.sha256
iphertext = base64.b64decode
v = ciphertext[:16]
ipher = AES.new(private_key, AES
ecrypted = unpad(cipher.decrypt(
bytes.decode(decrypted)
```

Figure 1

What we’ve Achieved

- Data Security
- Data Integrity
- Complexity

Libraries used

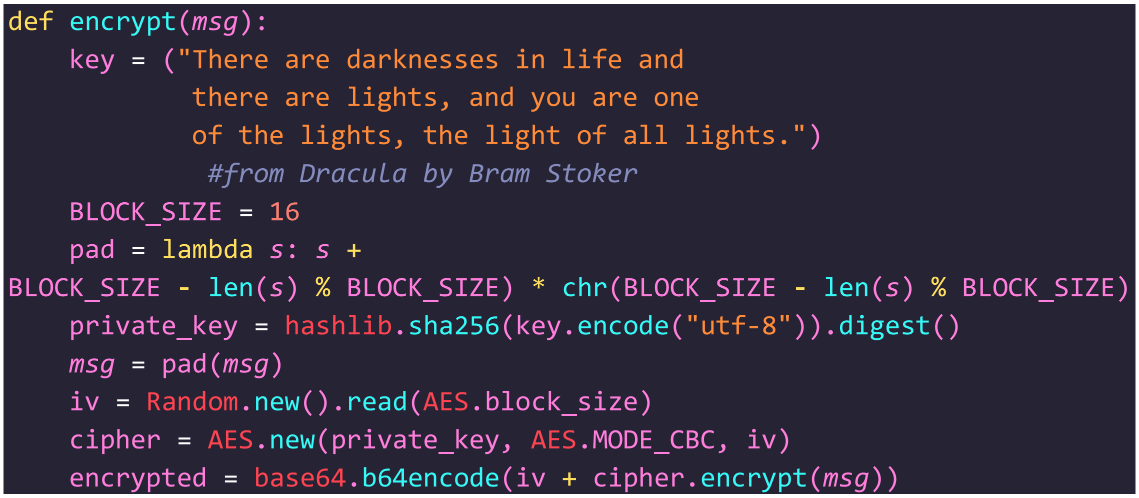
- Socket
- Cipher, AES

Functions used

- Encrypt ()
- Decrypt ()
- CRC ()
- String to Binary ()

CORE CODE

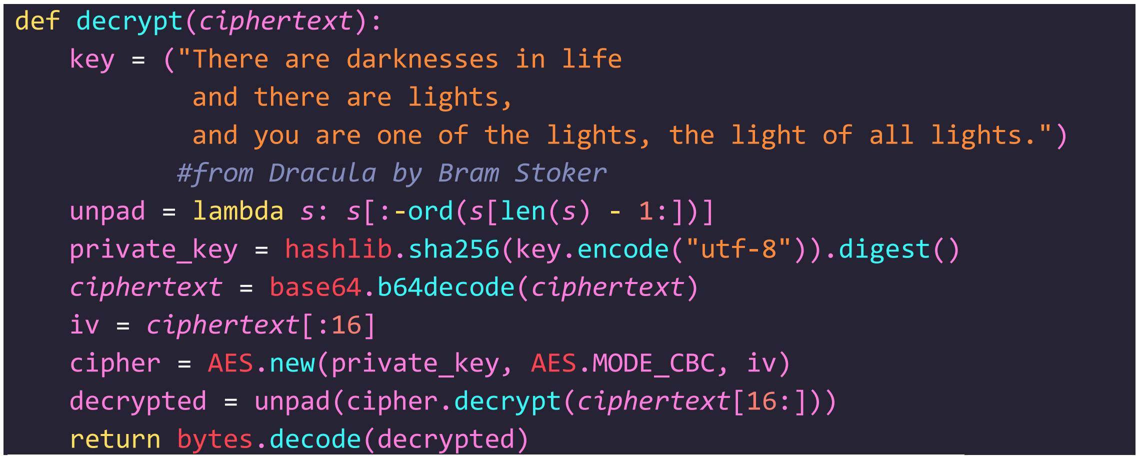
ENCRYPTION



```
def encrypt(msg):
    key = ("There are darknesses in life and
            there are lights, and you are one
            of the lights, the light of all lights.")
             #from Dracula by Bram Stoker
    BLOCK_SIZE = 16
    pad = lambda s: s +
BLOCK_SIZE - len(s) % BLOCK_SIZE) * chr(BLOCK_SIZE - len(s) % BLOCK_SIZE)
    private_key = hashlib.sha256(key.encode("utf-8")).digest()
    msg = pad(msg)
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(private_key, AES.MODE_CBC, iv)
    encrypted = base64.b64encode(iv + cipher.encrypt(msg))
```

we use **pycrypto** classes for AES 256 encryption and decryption. The program asks the user for a password (passphrase) for encrypting the data. This passphrase is converted to a hash value before using it as the key for encryption.

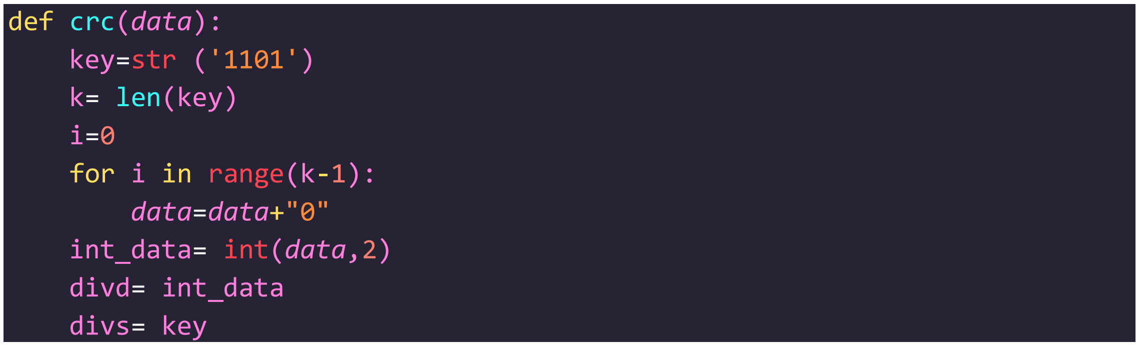
DECRYPTION



```
def decrypt(ciphertext):
    key = ("There are darknesses in life
            and there are lights,
            and you are one of the lights, the light of all lights.")
           #from Dracula by Bram Stoker
    unpad = lambda s: s[:-ord(s[len(s) - 1:])]
    private_key = hashlib.sha256(key.encode("utf-8")).digest()
    ciphertext = base64.b64decode(ciphertext)
    iv = ciphertext[:16]
    cipher = AES.new(private_key, AES.MODE_CBC, iv)
    decrypted = unpad(cipher.decrypt(ciphertext[16:]))
    return bytes.decode(decrypted)
```

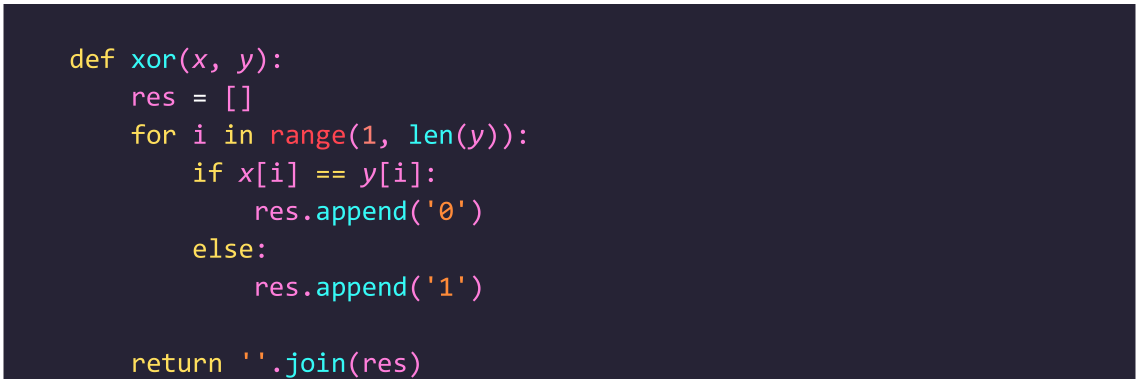
above program uses **SHA256** algorithm to generate the key from the passphrase.

CRC HASH



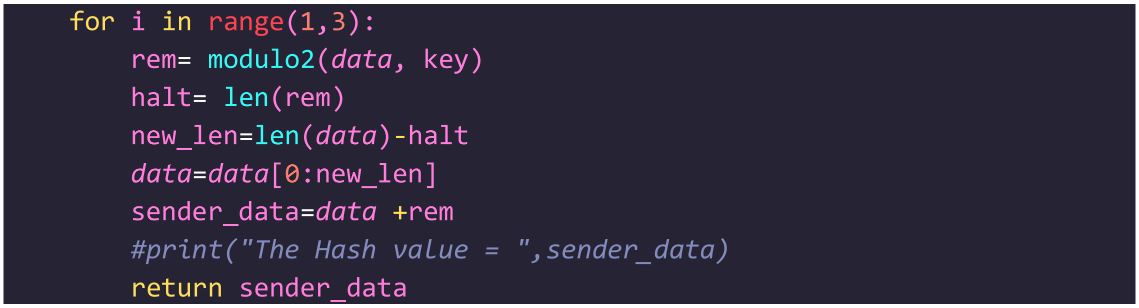
```
def crc(data):
    key=str ('1101')
    k= len(key)
    i=0
    for i in range(k-1):
        data=data+"0"
    int_data= int(data,2)
    divd= int_data
    divs= key
```

Here the data is taken as the argument and key is 1101 . Length of key minus one zero’s are appended at the end of the data.



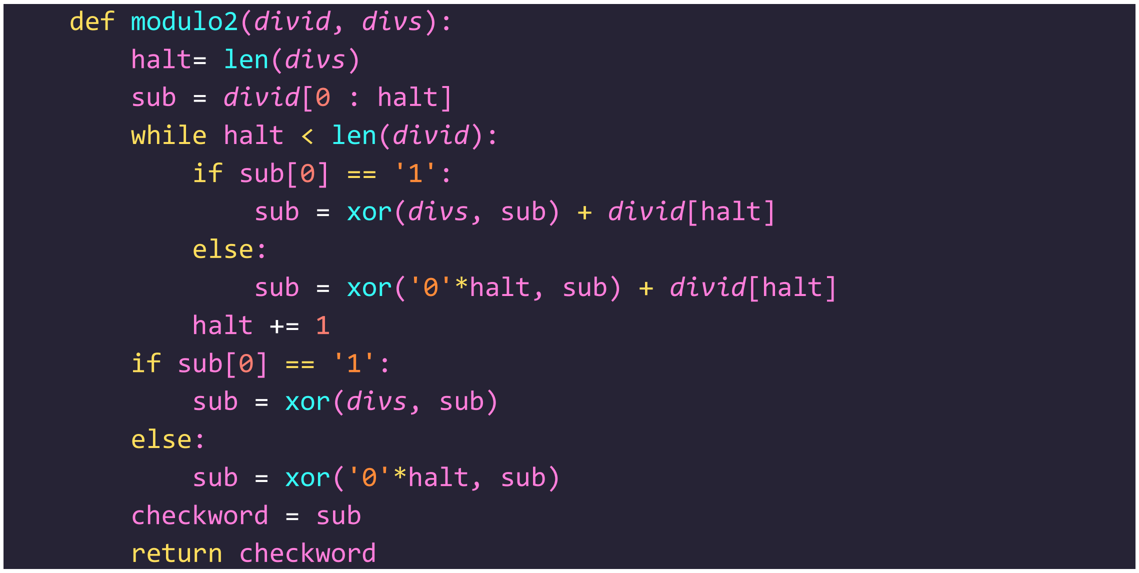
```

    def xor(x, y):
        res = []
        for i in range(1, len(y)):
            if x[i] == y[i]:
                res.append('0')
            else:
                res.append('1')

        return ''.join(res)
```

In xor method a list “res” is defined, if the bits in the division are equal then the value ‘0’ is appended to res if the bits are different then the value ‘1’ is appended to res. Call join method to join the bits in res and return the value.



```
    for i in range(1,3):
        rem= modulo2(data, key)
        halt= len(rem)
        new_len=len(data)-halt
        data=data[0:new_len]
        sender_data=data +rem
        #print("The Hash value = ",sender_data)
        return sender_data
```

Define a for loop with range(1,3) call modulo2 method and store the value in rem. Store the length of the remainder in halt variable. Define a variable new_len that stores the value of length of data minus length of the rem this gives the length of the initial data which is given as input. Define a variable “data” that stores the of substring from 0 to length of the new_len variable. Concatenate “data” with remainder and store the value in sender_data variable. This sender_data will the hash value of the give input

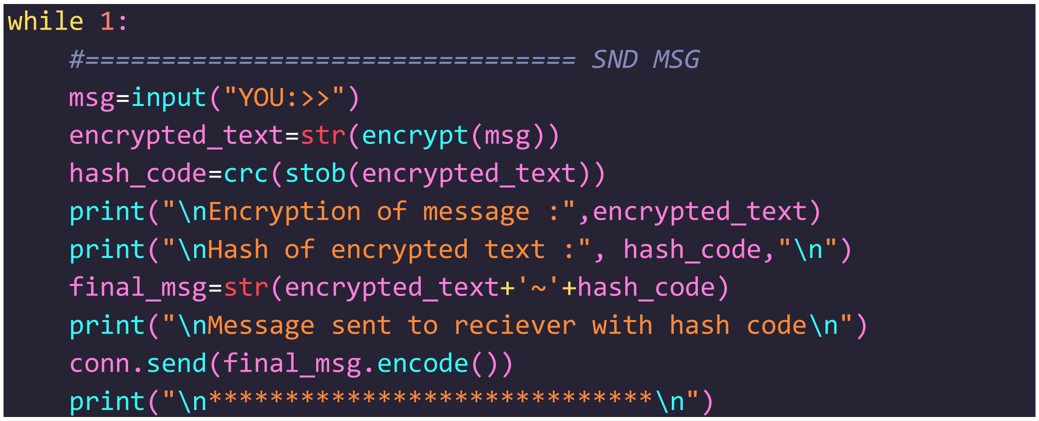


```
    def modulo2(divid, divs):
        halt= len(divs)
        sub = divid[0 : halt]
        while halt < len(divid):
            if sub[0] == '1':
                sub = xor(divs, sub) + divid[halt]
            else:
                sub = xor('0'*halt, sub) + divid[halt]
            halt += 1
        if sub[0] == '1':
            sub = xor(divs, sub)
        else:
            sub = xor('0'*halt, sub)
        checkword = sub
        return checkword
```

Define a variable halt that stores the value of the divisor to perform division , and define sub variable that contains a substring of dividend . And perform while loop with condition where length of divisor is less that the length of the dividend. If the initial bit of sub is 1 then perform xor operation between divisor and sub after performing xor operation add the next bit of the dividend to the result similarly If the initial bit of sub is 0 then take zeros equal to the length of the divisor and perform xor with sub after performing xor operation add the next bit of the dividend to the result and continue the process until the condition of the while loop fails. When halt becomes greater than length of the dividend, while condition fails and comes out of the loop, this will be the last step of the modulo2division now If the initial bit of sub is 1 then perform xor operation between divisor and sub similarly If the initial bit of sub is 0 then take length of divisor number of zeros and perform xor with sub. Store the final value of sub in checkword and return the value of checkword.

WHAT’S NEXT

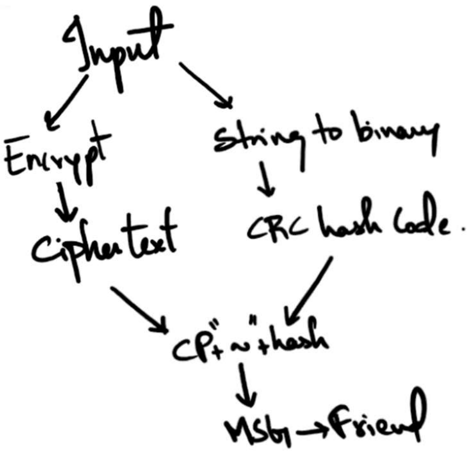
SEND MESSAGE



```
while 1:
    #================================= SND MSG
    msg=input("YOU:>>")
    encrypted_text=str(encrypt(msg))
    hash_code=crc(stob(encrypted_text))
    print("\nEncryption of message :",encrypted_text)
    print("\nHash of encrypted text :", hash_code,"\n")
    final_msg=str(encrypted_text+'~'+hash_code)
    print("\nMessage sent to reciever with hash code\n")
    conn.send(final_msg.encode())
    print("\n****************************\n")
```

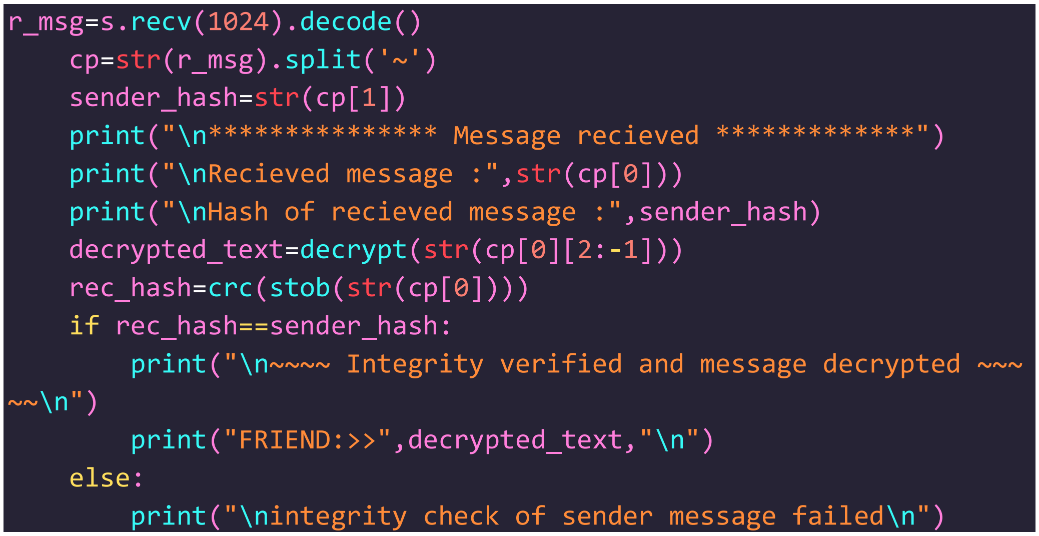
Understanding the Infinite loop, Since we need make the conservation as long the user requires we divided the code to two blocks RECV and SND

Now let’s breakdown SND-block, we asker the user to input the message and then it’s encrypted. After encryption the Ciphertext is converted to binary and then CRC hash value is generated later the final_msg contains both ciphertext and hash value separated by “~” and then sent to the Friend server using the socket instance.

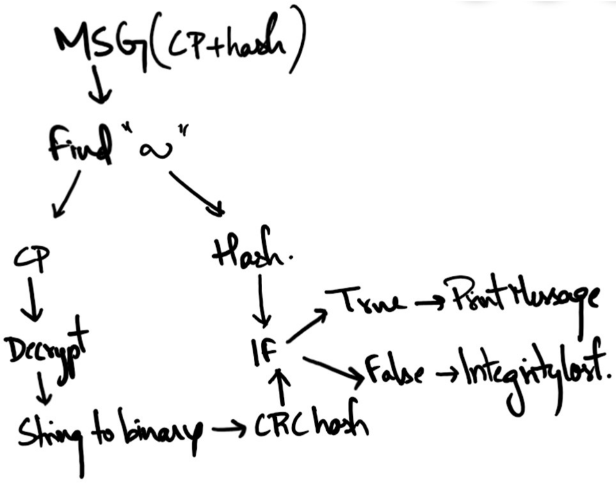


After combing the Ciphertext and hash code the message is send to the Friend who is connected using the system USER name using socket instance variable “conn” then the message is decodes to the machine language and then send to the Friend.

RECEIVE MESSAGE



```
r_msg=s.recv(1024).decode()
    cp=str(r_msg).split('~')
    sender_hash=str(cp[1])
    print("\n*************** Message recieved *************")
    print("\nRecieved message :",str(cp[0]))
    print("\nHash of recieved message :",sender_hash)
    decrypted_text=decrypt(str(cp[0][2:-1]))
    rec_hash=crc(stob(str(cp[0])))
    if rec_hash==sender_hash:
        print("\n~~~~ Integrity verified and message decrypted ~~~
~~\n")
        print("FRIEND:>>",decrypted_text,"\n")
    else:
        print("\nintegrity check of sender message failed\n")
```

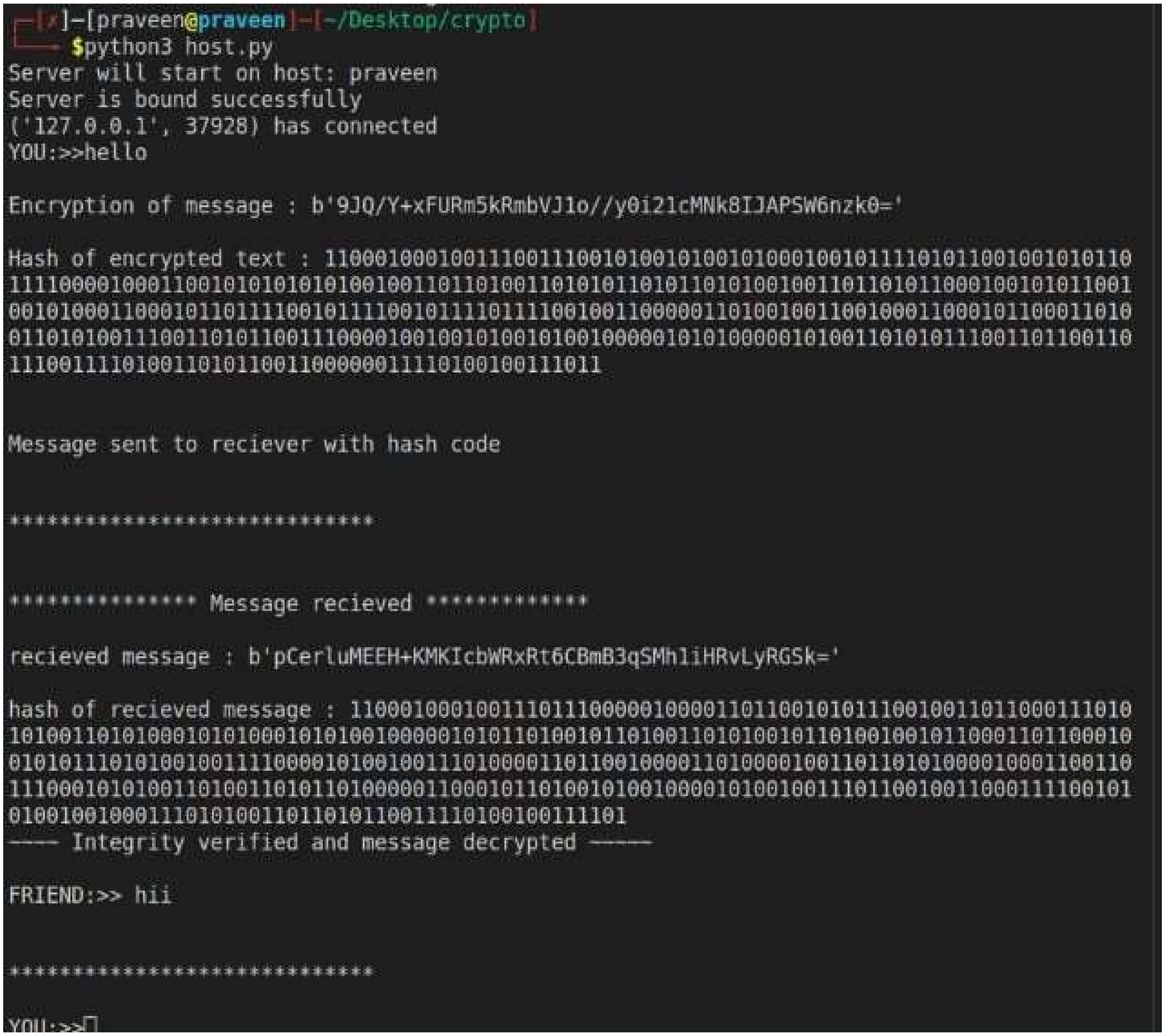


The message received is decoded to user-level language and stored in the r_msg now we need to separate the cipher text and hash code using "~" and 1st half is stored in sender hash and 2nd half is stored in the decrypted_text now after decryption its converted to binary and send to CRC to find the hash if the received hash and generated is TRUE then message is displayed else

```
Integrity check of sender message failed !!
```

RESULTS

PERSON 1

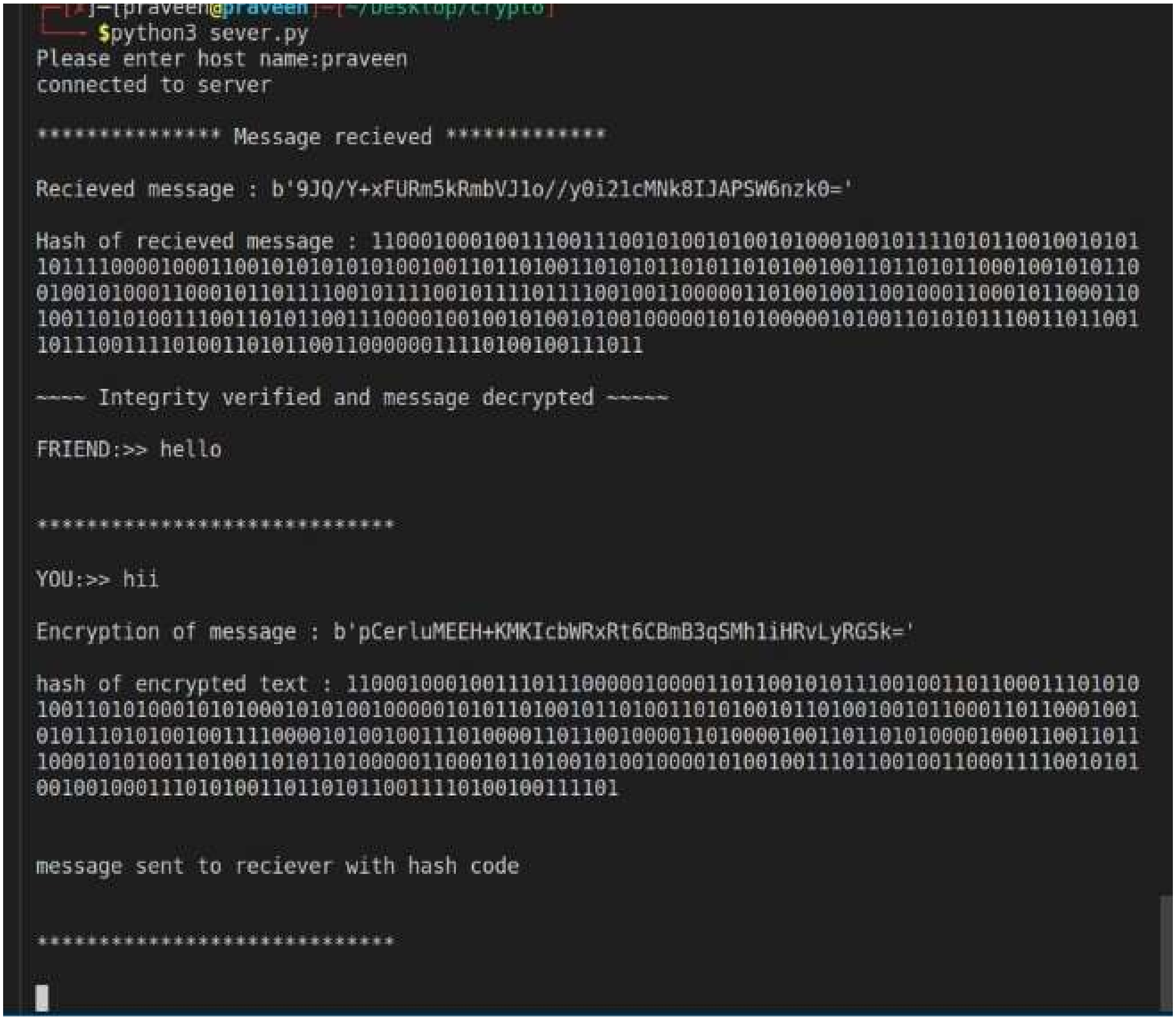


```
┌─[✗]─[praveen@praveen]─[~/Desktop/crypto]
└──╼ $python3 host.py
Server will start on host: praveen
Server is bound successfully
('127.0.0.1', 37928) has connected
YOU:>>hello

Encryption of message : b'9JQ/Y+xFURm5kRmbVJ1o//y0i21cMNk8IJAPSW6nzk0='

Hash of encrypted text : 1100010001001110011100101001010010100010010111101011001001010110
1111000010001100101010101010100100110110100110101011010110101001001101101011000100101011001
0010100011000101101111001011110010111101111001001100000110100100110010001100010110001101010
0110101001110011010110011100001001001010010100100000101010000010100110101011100110110011
0
11100111101001101011001100000001111010010011101 1
```

```
Message sent to reciever with hash code


******************************


*************** Message recieved *************

recieved message : b'pCerluMEEH+KMKIcbWRxRt6CBmB3qSMh1iHRvLyRGSk='

hash of recieved message : 110001000100111011100000100001101100101011100100110110001110 10
1010011010100010101000101010010000010101101001011010011010100101101001001011000110110001 0
010101110101001001111000010100100111010000110110010000110100001001101101010000100011001 10
1110001010100110100110101101000001100010110100101001000010100100111011001001100011110010 1
010010010001110101001101101011001111010010011110 1
---- Integrity verified and message decrypted -----

FRIEND:>> hii


******************************

YOU:>>▯
```

RESULTS

PERSON 2



```
┌─[✗]─[praveen@praveen]─[~/Desktop/crypto]
└──╼ $python3 sever.py
Please enter host name:praveen
connected to server

*************** Message recieved **************

Recieved message : b'9JQ/Y+xFURm5kRmbVJ1o//y0i21cMNk8IJAPSW6nzk0='

Hash of recieved message : 11000100010011100111001010010100101000100101111010110010010101
101111000001000110010101010101001001101101001101010110101101010010011011010110001001010110
01001010001100010110111100101111001011110111100100110000011010010011001000110001011000110
10011010100111001101011001110000100100101001010010000010101000001010011010101110011011001
1011100111101001101011001100000011110100100111011

~~~~ Integrity verified and message decrypted ~~~~~

FRIEND:>> hello


******************************

YOU:>> hii

Encryption of message : b'pCerluMEEH+KMKIcbWRxRt6CBmB3qSMh1iHRvLyRGSk='

hash of encrypted text : 11000100010011101110000001000011011001010111001001101100011101010
10011010100010101000101010010000010101101001011010011010100101101001001011000110110001001
01011101010010011110000101001001110100001101100100001101000010011011010100001000110011011
10001010100110100110101101000001100010110100101001000010100100111011001001100011110010101
0010010001110101001101101010110011110100100111101


message sent to reciever with hash code


******************************

█
```

ANNUAL REVIEW | 11