PROJECT REPORT

University at Buffalo - CSE 664

Advanced Cryptography and Computing Security

THRESHOLD ENCRYPTION SYSTEM

Project Mentor:

Professor Marina Blanton

Project Members:

Aditya Nalge (50207629)

Arti Gupta (50170010)

CONTENTS

1) Project Overview

1.a) Target Functionality

Our project, a threshold encryption system **targets the goal of adding a layer of security during decryption**. The target functionality of our project could be defined as creating an encryption system for a group of trusted users to use wherein each member would be able to encrypt data, but; for decrypting the encrypted data, a threshold number of users would have to work together. The value of threshold t would be less than or equal to the total number of users n **($n \geq 2t - 1$)**

This adds a layer of security to the encrypted data. Even if one of the users is compromised and wishes to access the data for malicious means, he/she cannot access the data alone. The user would require cooperation of the other members until he has the total number of users - required to obtain threshold, willing to cooperate.

To understand our project better, **consider this example**: Imagine a secure room which is locked from the outside. You have to open 3 doors, one after the other to enter the room. Each door has a different lock and uses a different key.

As a result, you would have to lock and unlock three doors to close and open the room respectively.

Alternatively, imagine the room has only one door which can be locked with a single key. However, to unlock the door, you can use five different keys but you need to insert three of those keys at the same time. This is more convenient as you only have to lock the door once. Also, even though you need three keys, you are unlocking a single time.

Moreover, it provides safety in case an unlocking key is lost because you could still use three of the remaining four keys to unlock the door. Thus, this approach is better than the previous approach.

Similarly, threshold encryption schemes can be used as a substitute of traditional symmetric encryption schemes in places where having the requirement of multiple parties for decryption adds an essential layer of security.

1.b) Security Model and Objectives

Information Protected From - The information is protected from any person who is outside the trusted group of members.

Adversary Capabilities - An adversary trying to break the encryption scheme can use known cipher text attack. The adversary will have to perform frequency analysis to analyze the frequency of characters in cipher text. For a single encryption round, the system is a mono alphabetic cryptographic system. Thus the adversary can only perform frequency analysis on an single cipher text. This is because, the random chosen during ElGamal encryption will result in the same input string being encrypted differently when encryption is called twice.

Trust Assumptions - For a threshold encryption scheme, the security model requires that **it is implemented amongst a group of trusted users**. Encryption is fairly simple since a public key is used whose value is known to everyone. However, **key generation is a critical component** since each user contributes to generate the public key. **It is a trust assumption that all the participating users keep their contribution secret and do not share it with each other.**

The information being protected and the capabilities of an adversary depend entirely on the security environment wherein the scheme is being used. It could be used in the financial sector at a bank which would only allow large transactions to go through when two or more managers approve it. In this case; the data being protected would be the financial transactions. It could also be used on a small scale where a user could encrypt data on a disk and distribute the decryption shares amongst his/her friends. In that case the data being protected would be personal data.

Security Objectives - A threshold encryption scheme should ideally achieve:

- **Confidentiality**
- **Correctness**
- **Integrity**
- **Authorization**
- **Non repudiation**

1.c) Project Goals

The goals of the project were similar to that of a normal data encryption decryption system. However, being a threshold encryption system, there were additional goals that had to be defined.

One of the main goals was to achieve an efficient way to divide a secret using **Shamir's Secret Sharing Algorithm**. The algorithm allowed us to generate shares for secrets possessed by each user and generate decryption shares.

Another goal was to ensure that, even though the users share decryption shares amongst themselves, **no information can be obtained about the original secret using these decryption shares**. This goal is fulfilled since recovering the original from the secret is a **discrete logarithm problem** and thus, is computationally infeasible for large enough values.

An **important goal** in establishing the threshold is that even if **threshold -1 number of users** work together; they learn nothing about the original data from the encrypted data. The only way to decrypt the encrypted data is by using a threshold number of decryption shares.

1.d) Cryptographic Design

There are five main steps to the Threshold Encryption Scheme. These steps are,

i. Creating the 'Public Key' using a secret value from each user.
ii. Creating and distributing 'Shares' of each user's secret.
iii. Encrypting a Message.
iv. Creating Decryption Shares for each user based on their secret.
v. Decrypting a Message.

Also, we are implementing **Digital Signatures** for non-repudiation.

Amongst the above steps, step i,ii have to be executed only once by each user. Step iii and v can be executed multiple times to encrypt and decrypt different messages. Step iv will have to be executed to obtain decryption shares, every time we want to decrypt a different message.

We will describe each step in detail:

i. **Creating the 'Public Key' using a secret value from each user**

We use a modified ElGamal Encryption in our project since decryption is different as the secret key is not known. However, encryption and public key are similar to standard ElGamal.

The ElGamal public key is (**p, q, g, h**). Here,

G_q is the unique sub group of Z^*_p

p - is a large prime number.

q - is a large prime number such that $q = (p - 1) / 2$

g - is a generator of cyclic group G_q

The p, q & g values can be predefined by all the users together. It is acceptable to use the same p, q & g values for different public keys as long as the value of h changes.

The value of h is determined by the following steps:

- Each user P_i chooses a secret value X_i such that $0 < X_i < q-1$ (i - each user's number)
- The user then computes h_i as $h_i = g^{X_i}$
- Every user shares this h_i value. It is important to note that no other user can determine Xi using hi because it will mean calculating $\log_g h$ which is a discrete log problem.
- Finally, h is determined as $h = (h_1 \times h_2 \times h_3 \times \dots \times h_n) \bmod p$ (n - total number of users)

Thus, we get the public key (p, q, g, h). The values of p, q, g & h are public and hence known to all the users.

ii. **Creating and Distributing 'Shares' of each user's secret**

In this step, the Shamir's Secret Sharing algorithm is used to create shares of the secret. Each user creates n shares of their secret and distributes it amongst the other users.

For ex: Assume 5 users and threshold value is 3. Say,

User 1 has a secret X1, User 2 has a secret X2, User 3 has a secret X3, User 4 has a secret X4 and User 5 has a secret X5.

User 1 will create 5 shares of X1 as S (1,1), S (1,2), S (1,3), S (1,4), S (1,5).

User 2 will create 5 shares of X2 as S (2,1), S (2,2), S (2,3), S (2,4), S (2,5).

User 3 will create 5 shares of X3 as S (3,1), S (3,2), S (3,3), S (3,4), S (3,5).

User 4 will create 5 shares of X4 as S (4,1), S (4,2), S (4,3), S (4,4), S (4,5).

User 5 will create 5 shares of X5 as S (5,1), S (5,2), S (5,3), S (5,4), S (5,5).

Every user will receive one share from all the remaining users. Thus,

User 1 now has S (1,1), S (2,1), S (3,1), S (4,1), S (5,1)

User 2 now has S (1,2), S (2,2), S (3,2), S (4,2), S (5,2)

User 3 now has S (1,3), S (2,3), S (3,3), S (4,3), S (5,3)

User 4 now has S (1,4), S (2,4), S (3,4), S (4,4), S (5,4)

User 5 now has S (1,5), S (2,5), S (3,5), S (4,5), S (5,5)

Each user adds the received shares to compute a share of the main secret X. However, no user knows X.

User 1 computes S (1,1) + S (2,1) + S (3,1) + S (4,1) + S (5,1) = S1

User 2 computes S (1,2) + S (2,2) + S (3,2) + S (4,2) + S (5,2) = S2

User 3 computes S (1,3) + S (2,3) + S (3,3) + S (4,3) + S (5,3) = S3

User 4 computes S (1,4) + S (2,4) + S (3,4) + S (4,4) + S (5,4) = S4

User 5 computes S (1,5) + S (2,5) + S (3,5) + S (4,5) + S (5,5) = S5

Thus, each user finally receives a share of the main secret without knowing what the main secret was.

iii. **Encrypting a Message**

Initially, we are signing the message using Digital Signatures where sk is the signing key,

sig = sk.sign (message) (**Signing Key** is different for each user separately)

where sk will be generated for each user and each will get its vk i.e. verification key which depends on the user's sk; to verify the message later. After, the message is signed, encryption process will start:

We use the public key $\mathbf{P_k}$ **- (p, q, g, h)** to encrypt a message.

A random value y is chosen, such that $y \in Z^*_q$.

To encrypt a message $m \in G$

- Compute, $c1 = g^y$
- Compute, $c2 = m \times h^y$
- Cipher Text c -

$$\mathbf{c = Enc_{pk}(m) = (c1,c2) = (g^y, m.h^y)}$$

iv. **Creating Decryption Shares for each user based on their secret**

Unlike standard ElGamal, there is no secret key for decryption.

Thus, decryption is modified and done using decryption shares.

Each user creates their own decryption share D_i as -

$$K_i = (S_i * ((u2*u3)/((u2-u1) * (u3-u1))))$$

$$D_i = (c1^{K_i}) \% p$$

Here,

- S_i is the users share of the main secret X.

- u1, u2,u3 are public values i.e. the user numbers of the current user and other users present during decryption.

- c1 is the cipher text obtained in step 3.

- D_i is the decryption share.

- K_i is an intermediate value

v. **Decrypting a Message**

If the total number of members that are a part of this encryption scheme is 'n' and if the threshold of the encryption system is 't', we require atleast '*t*' Decryption Shares to decrypt the message.

We ask the users to enter their decryption shares d_1, d_2,...d_t

Then multiply all these shares together as,

$$d = (d_1 \times d_2 \times d_3 \times .... \times d_t) \bmod p$$

Decryption is done to obtain the original message m as,

$$m = (c2 \times d^{-1}) \% p$$

Verification step of Digital Signatures takes place in this step i.e. Decryption.

During verification phase, the **verification key** is used to determine if the message was encrypted by an authorized user using;

assert vk.verify (sig, m)

Correctness can be proved as,

(Decryption using Decryption Shares (Encryption using Public Key(m))) = m

1.e) Security Objectives Achieved

Confidentiality - We were able to successfully encrypt the message.

Correctness - On decryption, we were able to retrieve the original message back.

Integrity - Hashing was used to ensure message integrity.

Authorization - Since, the algorithm is designed for a trusted group of users, it can be assumed that only the users who are a part of the algorithm hold the public key which they create together. As a result, a message which was encrypted using this key could be authorized as sent from one of the members. We are implementing digital signatures.

Non repudiation - Digital Signatures were used to achieve non-repudiation.

2) Project Outcome

2.a) Project Setup

The entire project was done using the programming language **Python 3**. We used 'Specific Python Development Environment' (Spyder) which is a powerful interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features.

No external databases were used. Project is being tested on both Windows 8.1 and Mac OS X El Capitan platform.

We are using some of the cryptography libraries:

- from ecdsa import SigningKey
- from ecdsa import VerifyingKey, BadSignatureError
- from Crypto import Random
- from Crypto.Util import number
- import pem (for storing signing key and verification key into a file)

Project can be run from Python Console.

2.b) Results

- Performance Analysis:

- We tested with different test cases, we are able to retrieve original message every time.
- Initially, we manually calculated power values in our code, but, then we used in-built **pow** function and now code is execution is faster.
- We tried to use library for various phases of project, but, since our project involves various concepts (decryption shares 5 times for 5 users etc), we are manually computing and using cryptography library for just digital signatures and random number generator.

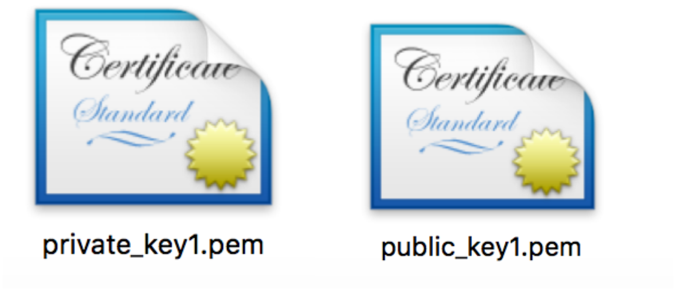
- Security Analysis:

- Instead of using any random function, we are generating random numbers using cryptography libraries to make it strongly secure.
- Our project provides security in many aspects, like we have implemented digital signatures to avoid non-repudiation, and, provides security against any adversary who may try to attack the system even if t-1 shares are known to him.

- Scalability Issues:

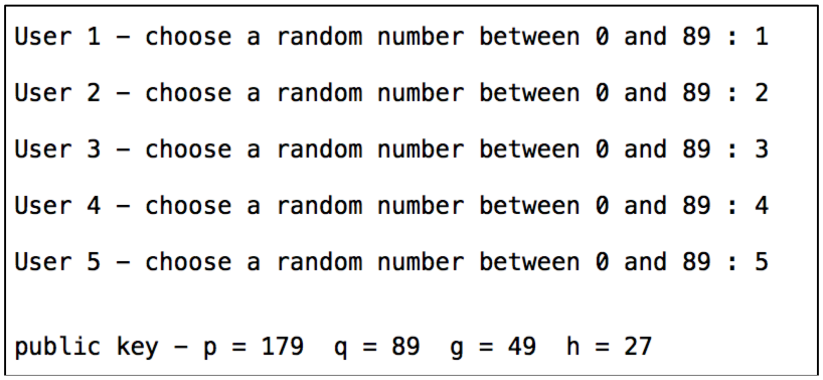
- When values of p, q are too large, there are computation issues due to restricted computation ability of testing environment.
- It is difficult to create a trusted environment among large number of users.
- Storage issues may arise due to large computations and large number of users involved since it will require us to store those many inputs/outputs.
- It will increase cost of computation.

For Digital Signatures, Signing key (private_key) and Verification key (public_key) are exported to a file and after decryption, user import his verification file (public_key) and verifies the signature. Following gets stored in file:



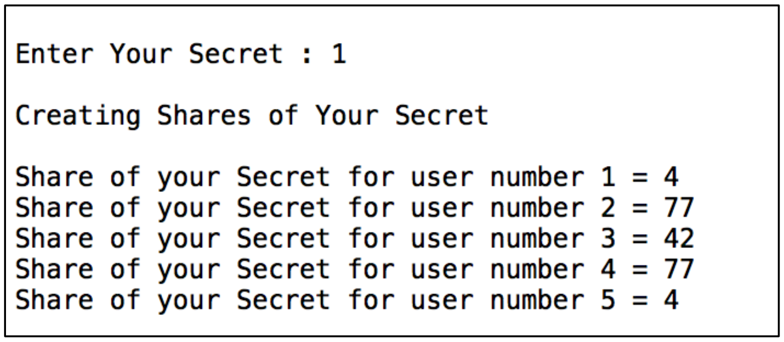
Output on Console: (Screen Shots)

Initially, users will choose their secret values and public key will be generated:



```
User 1 - choose a random number between 0 and 89 : 1

User 2 - choose a random number between 0 and 89 : 2

User 3 - choose a random number between 0 and 89 : 3

User 4 - choose a random number between 0 and 89 : 4

User 5 - choose a random number between 0 and 89 : 5


public key - p = 179  q = 89  g = 49  h = 27
```

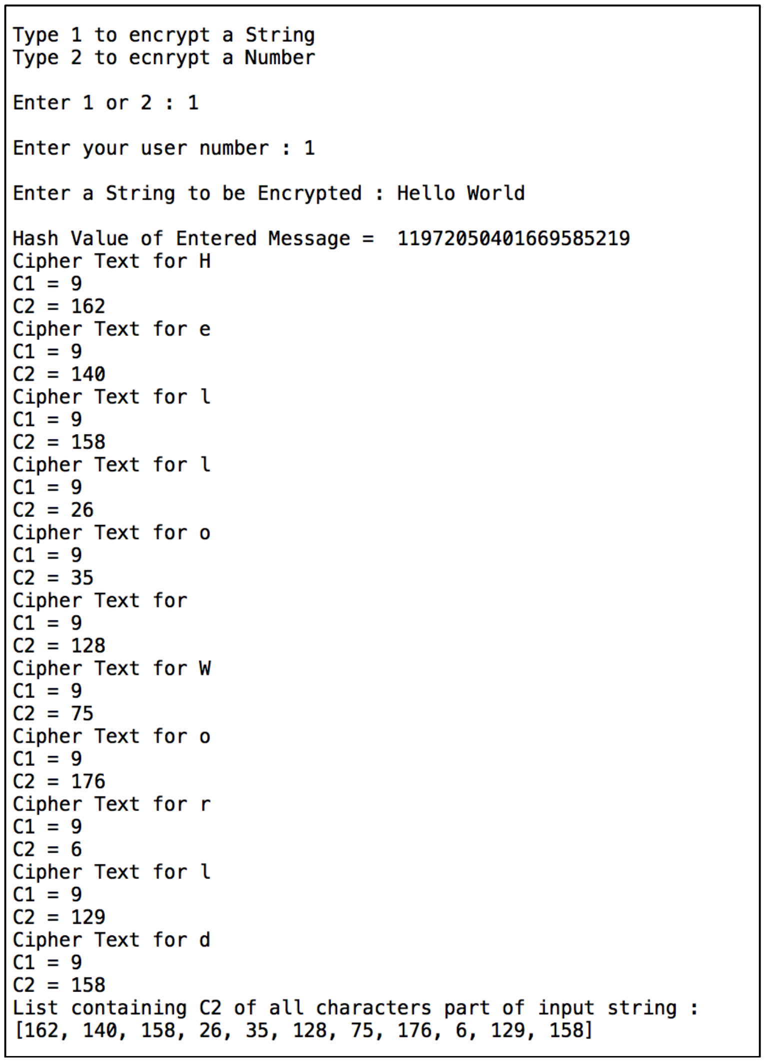
Then, each user will enter his secret and shares will be generated:



```
Enter Your Secret : 1

Creating Shares of Your Secret

Share of your Secret for user number 1 = 4
Share of your Secret for user number 2 = 77
Share of your Secret for user number 3 = 42
Share of your Secret for user number 4 = 77
Share of your Secret for user number 5 = 4
```

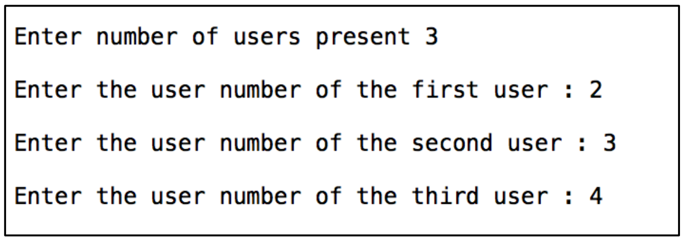
Similarly, other shares will be generated.

Now, we will start encryption:

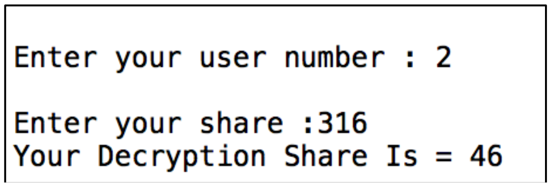


```
Type 1 to encrypt a String
Type 2 to ecnrypt a Number

Enter 1 or 2 : 1

Enter your user number : 1

Enter a String to be Encrypted : Hello World

Hash Value of Entered Message =  11972050401669585219
Cipher Text for H
C1 = 9
C2 = 162
Cipher Text for e
C1 = 9
C2 = 140
Cipher Text for l
C1 = 9
C2 = 158
Cipher Text for l
C1 = 9
C2 = 26
Cipher Text for o
C1 = 9
C2 = 35
Cipher Text for
C1 = 9
C2 = 128
Cipher Text for W
C1 = 9
C2 = 75
Cipher Text for o
C1 = 9
C2 = 176
Cipher Text for r
C1 = 9
C2 = 6
Cipher Text for l
C1 = 9
C2 = 129
Cipher Text for d
C1 = 9
C2 = 158
List containing C2 of all characters part of input string :
[162, 140, 158, 26, 35, 128, 75, 176, 6, 129, 158]
```

Total number of users present at the time of decryption:

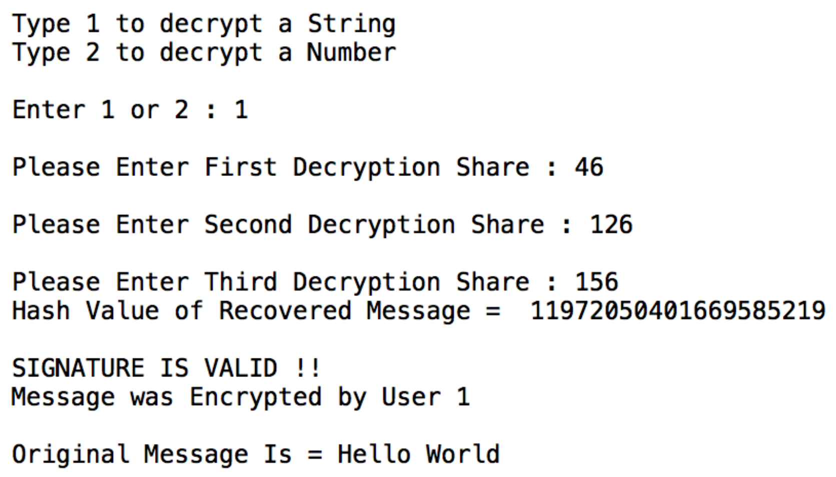


```
Enter number of users present 3

Enter the user number of the first user : 2

Enter the user number of the second user : 3

Enter the user number of the third user : 4
```



```
Enter your user number : 2

Enter your share :316
Your Decryption Share Is = 46
```

Similar process for other users. Now decryption will begin and verification of signature will be validated.



```
Type 1 to decrypt a String
Type 2 to decrypt a Number

Enter 1 or 2 : 1

Please Enter First Decryption Share : 46

Please Enter Second Decryption Share : 126

Please Enter Third Decryption Share : 156
Hash Value of Recovered Message =  11972050401669585219

SIGNATURE IS VALID !!
Message was Encrypted by User 1

Original Message Is = Hello World
```

2.c) Description of Achieved Project Goals

Steps in achieving our goals :

- Successful Key Generation – The key generation script of our project is successfully able to generate a key using contributions from all the trusted members.

- Successful Sharing of Secrets – Using Shamir's Secret Sharing, our script used to create secret shares is **able to successfully create shares for a secret** that can be used to recover the secret. However, we don't recover the secret in our project, we modify the shares obtained and use this modified shares in decryption.

- Successful Encryption using ElGamal Encryption - In our ElGamal Encryption, we were able to effectively use the standard form of ElGamal Encryption scheme to encrypt the input given by the user.

- Successful Creation of Decryption Shares - Perhaps the most crucial goal of our project was **successfully creating decryption shares** for the users such that a threshold number of users could decrypt the message. However **any number of users less than the threshold would not be able to learn anything about the message from the cipher text**.

- Successful Decryption of Original Message – The script for decryption enables us to use the decryption shares to decrypt the cipher text and obtain the original message back.

<u>Justification for adjusting goals:</u>

- Initially, we had plans to build a UI for our project. But, since our project requires many more inputs at run-time, so, instead of building a UI (20-21 screens: as per the structure of project), we decided to focus more on improvising cryptographic techniques in our project.

- For the value of p, q; ideally p and q would be 2000 bits long to because 2000 bits would be a acceptable security parameter. We were unable to implement that due to lack of computation power of the system environment that we are using.

2.d) Difficulties Encountered

A major difficulty was understanding the concept of the project. Figuring out how a single key could be used for encryption but a combination of different keys for decryption. It took us a lot of meetings with our Mentor along with reading multiple research paper to finally understand the flow of the security components of the project.

Another issue was testing our code. Due to the multiple variations in inputs that could be provided, it was difficult testing all combinations of inputs for our code. It would be a periodic occurrence where our scripts would successfully complete a couple test cases before we realized that there was an error. Gradually, we fixed the errors and did not encounter a single failed test case.

Implementing the project for small values was completed. However, modifying these values to hold up to current security standards was tough as the current security standards demands certain values to be longer than 2000 bits. This highly increased computations which our system was not able to handle.

2.e) Testing

Testing of our project was an important task because of the various required inputs and calculations. Also, being a threshold encryption system, the groups of users participating in each decryption test could be different. As a result, many combinations of values with different inputs had to be tested to verify correctness of our algorithm.

In the earlier stages, we followed a White Box Testing approach. We would develop a Python script to perform a specific function and test it accordingly. We would modify the script based on the tests. Once, all the different scripts were developed, we combined them to proceed to the latter part of our project.

In the latter part of our project, we relied on Black Box Testing wherein inputs were provided and the output message obtained after decrypting the cipher text was compared with the original message. If the output message and input message were the same, we would mark that test as successful.

Q4) Description of Data

We used a lot of test cases since our threshold encryption system was a (5,3) system, there were a lot of combinations of inputs possible. The test cases were a combination of strings and integers.

References:

1. http://www.cs.cornell.edu/courses/cs754/2001fa/307.PDF
2. https://www.dcl.hpi.uni-potsdam.de/teaching/cloudsec/presentations/threshold-cryptography.pdf
3. https://en.wikipedia.org/wiki/ElGamal_encryption
4. https://www.cryptoworkshop.com/ximix/lib/exe/fetch.php?media=pedersen.pdf
5. http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html
6. https://en.wikipedia.org/wiki/Lagrange_polynomial#Definition

SOURCE CODE

All the code mentioned below is written for this project. We are referring the syntax and use cases of cryptographic libraries from https://pypi.python.org/pypi/ecdsa/0.13. Any doubts about function syntax were solved after referring stack overflow.

Script 0 - Public Key Generation

```
#************************************KEY GENERATION**************************************

#max = 9223372036854775807
#p = 6744073709551523
#q = 3372036854775761
#p = 209459
#q = 104729

p = 179
q = 89
g = 49

while True:
   x1 = int(input("User 1 - choose a random number between 0 and " + str(q) + " : "))
   if (0 < x1 < q):
      break;
   else:
      print ("Error! Choose Again")
h1 = pow(g,x1,p)
#h1 = (g**x1) % p


while True:
   x2 = int(input("User 2 - choose a random number between 0 and " + str(q) + " : "))
   if (0 < x2 < q):
      break;
   else:
      print ("Error! Choose Again")
h2 = pow(g,x2,p)
#h2 = (g**x2) % p


while True:
   x3 = int(input("User 3 - choose a random number between 0 and " + str(q) + " : "))
   if (0 < x3 < q):
      break;
   else:
      print ("Error! Choose Again")
h3 = pow(g,x3,p)
#h3 = (g**x3) % p


while True:
   x4 = int(input("User 4 - choose a random number between 0 and " + str(q) + " : "))
   if (0 < x4 < q):
      break;
   else:
      print ("Error! Choose Again")
h4 = pow(g,x4,p)
#h4 = (g**x4) % p


while True:
   x5 = int(input("User 5 - choose a random number between 0 and " + str(q) + " : "))
   if (0 < x5 < q):
      break;
   else:
      print ("Error! Choose Again")
h5 = pow(g,x5,p)
#h5 = (g**x5) % p

h = (h1 * h2 * h3 * h4 * h5) % p

f = open('PublicKey.txt','w')
f.write("Public Key is -\n")
f.write("p =")
f.write(str(p))
f.write("\tq =")
f.write(str(q))
f.write("\tg =")
f.write(str(g))
f.write("\th =")
f.write(str(h))
f.write(' ')
f.close()

print ("\n\npublic key - p =",p, " q =",q," g =",g," h =",h)
```

Script 1 - Creating Shares

```
#***********************************CALCULATING SHARES**********************************

from Crypto.Util import number
from Crypto import Random

n = 5
t = 3


my_list = []

s = int(input("\nEnter Your Secret : "))
my_list.append(s)


for i in range(1,t):
   CryptoRandomNumber = number.getRandomRange(1, 88, Random.new().read)
   my_list.append(CryptoRandomNumber)

#my_list[2] = 15
#my_list[1] = 14

print("")
print("Creating Shares of Your Secret")
print("")

for i in range(1,n+1):
   share = ((my_list[2]*(i**2))+(my_list[1]*i)+(my_list[0]))%q
   print("Share of your Secret for user number",i,"=",share)
```

Script 3 - Encryption & Signing

```
#***********************************ENCRYPTION &SIGNING*************************************

from ecdsa import SigningKey
from Crypto import Random
from Crypto.Util import number
import pem

def int_to_bytes(x):
   return x.to_bytes((x.bit_length() + 7) // 8, 'big')

def int_from_bytes(xbytes):
   return int.from_bytes(xbytes, 'big')

my_list1 = []
my_list2 = []
list_c2 = []
z = 9223372036854775807
w = 0

y = number.getRandomRange(1, (q-1),
Random.new().read)
#y = 10
print ("Y chosen at random between 1 and ",q-1,"=",y)

#h = int(input("Enter Value of h : "))
print('')
print("\nType 1 to encrypt a String")
print("Type 2 to ecnrypt a Number")
k = int(input("Enter 1 or 2 : "))
f = int(input("Enter your user number : "))

if (k==1):

   string = input("Enter a String to be Encrypted : ")
   print('')

   for c in string:
      my_list1.append(c)

   for c in string:
      my_list2.append(ord(c)+w)
      w=w+1


   b = [str(x) for x in my_list2]
   r_int = ''.join(b)
   #r = int(r)

   r_enc = hash(r_int) % ((z + 1) * 2)

   data_integrity1 = hash(string) % ((z+1) * 2)
   print("Hash Value of Entered Message =
",data_integrity1)

   mbytes_enc = int_to_bytes(r_enc)


   if (f==1):
      sk1 = SigningKey.generate()
      vk1 = sk1.get_verifying_key()

open("private_key1.pem","wb").write(sk1.to_pem())

open("public_key1.pem","wb").write(vk1.to_pem())
      sk1 =
SigningKey.from_pem(open("private_key1.pem").read()
)
      sig = sk1.sign(mbytes_enc)


   elif (f==2):
      sk2 = SigningKey.generate()
      vk2 = sk2.get_verifying_key()
```

Script 3 - Total Users Present

```
#***************************************USERS PRESENT*****************************************

v=0

while v==0:
   num = int(input("Enter number of users present "))

   if (num <= 0 or num >5):
      print("Invalid Choice! Choose Again")

   if (0< num < 3):
      print("Not Enough Users ! Choose Again")

   elif num==3:
      v=1

      while True:
         u1 = int(input("Enter the user number of the first user : "))
         u2 = int(input("Enter the user number of the second user : "))
         u3 = int(input("Enter the user number of the third user : "))
         if((u1 <= 0) or (u1>5) or (u2<=0) or (u2>5) or (u3 <= 0) or (u3 > 5)):
            print("\nInvalid User Number! Re Enter")
         else:
            break;

   elif num==4:
      v=1

      while True:
         u1 = int(input("Enter the user number of the first user : "))
         u2 = int(input("Enter the user number of the second user : "))
         u3 = int(input("Enter the user number of the third user : "))
         u4 = int(input("Enter the user number of the fourth user : "))

         if((u1 <= 0) or (u1>5) or (u2<=0) or (u2>5) or (u3 <= 0) or (u3 > 5) or (u4 <= 0) or (u4 > 5)):
            print("\nInvalid User Number! Re Enter")
         else:
            break;

   elif num==5:
```

Script 4 - Create Decryption Shares

```
#*************************************DECRYPTION SHARES***************************************

def invmodp(a, b):

   for d in range(1, b):
      r = (d * a) % b
      if r == 1:
         break
   else:
      raise ValueError('%d has no inverse mod %d' % (a,
b))
   return d

if (num ==3):

   u = int(input("\nEnter your user number : "))

   if (u == 1):
      if(u == u1):
         t = int(input("Enter your share :"))
         s = (t * ((u2*u3) * invmodp(((u2-u1)*(u3-u1)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u2):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u3) * invmodp(((u1-u2)*(u3-u2)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u3):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u2) * invmodp(((u2-u3)*(u1-u3)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))
      else:
         print("User Not Present")

   elif (u == 2):
      if(u == u1):
         t = int(input("Enter your share :"))
         s = (t * ((u2*u3) * invmodp(((u2-u1)*(u3-u1)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u2):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u3) * invmodp(((u1-u2)*(u3-u2)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u3):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u2) * invmodp(((u2-u3)*(u1-u3)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))
      else:
         print("User Not Present")

   elif (u == 3):
      if(u == u1):
         t = int(input("Enter your share :"))
         s = (t * ((u2*u3) * invmodp(((u2-u1)*(u3-u1)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u2):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u3) * invmodp(((u1-u2)*(u3-u2)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u3):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u2) * invmodp(((u2-u3)*(u1-u3)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))
      else:
         print("User Not Present")

   elif (u == 4) :
      if(u == u1):
         t = int(input("Enter your share :"))
         s = (t * ((u2*u3) * invmodp(((u2-u1)*(u3-u1)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u2):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u3) * invmodp(((u1-u2)*(u3-u2)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u3):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u2) * invmodp(((u2-u3)*(u1-u3)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))
      else:
         print("User Not Present")

   elif (u == 5) :
      if(u == u1):
         t = int(input("Enter your share :"))
         s = (t * ((u2*u3) * invmodp(((u2-u1)*(u3-u1)),q)))
         print("Your Decryption Share Is =",(pow(c1,s,p)))

      elif(u == u2):
         t = int(input("Enter your share :"))
         s = (t * ((u1*u3) * invmodp(((u1-u2)*(u3-u2)),q)))
```

```
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u3):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2) * invmodp(((u2-u3)*(u1-u3)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        else:
            print("User Not Present")


elif(num ==4):

    u = int(input("\nEnter your user number :"))

    if (u == 1):
        if(u == u1):
            t = int(input("Enter your share :"))
            s = (t * ((u2*u3*u4) * invmodp(((u2-u1)*(u3-
u1)*(u4-u1)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u2):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u3*u4) * invmodp(((u1-u2)*(u3-
u2)*(u4-u2)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u3):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u4) * invmodp(((u1-u3)*(u2-
u3)*(u4-u3)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u4):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u3) * invmodp(((u1-u4)*(u2-
u4)*(u3-u4)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        else:
            print("User Not Present")

    if (u == 2):
        if(u == u1):
            t = int(input("Enter your share :"))
            s = (t * ((u2*u3*u4) * invmodp(((u2-u1)*(u3-
u1)*(u4-u1)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u2):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u3*u4) * invmodp(((u1-u2)*(u3-
u2)*(u4-u2)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u3):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u4) * invmodp(((u1-u3)*(u2-
u3)*(u4-u3)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u4):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u3) * invmodp(((u1-u4)*(u2-
u4)*(u3-u4)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        else:
            print("User Not Present")

    if (u == 3):
        if(u == u1):
            t = int(input("Enter your share :"))
            s = (t * ((u2*u3*u4) * invmodp(((u2-u1)*(u3-
u1)*(u4-u1)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u2):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u3*u4) * invmodp(((u1-u2)*(u3-
u2)*(u4-u2)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u3):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u4) * invmodp(((u1-u3)*(u2-
u3)*(u4-u3)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u4):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u3) * invmodp(((u1-u4)*(u2-
u4)*(u3-u4)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        else:
            print("User Not Present")

    if (u == 4):
        if(u == u1):
            t = int(input("Enter your share :"))
            s = (t * ((u2*u3*u4) * invmodp(((u2-u1)*(u3-
u1)*(u4-u1)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))
        elif(u == u2):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u3*u4) * invmodp(((u1-u2)*(u3-
u2)*(u4-u2)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u3):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u4) * invmodp(((u1-u3)*(u2-
u3)*(u4-u3)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        elif(u == u4):
            t = int(input("Enter your share :"))
            s = (t * ((u1*u2*u3) * invmodp(((u1-u4)*(u2-
u4)*(u3-u4)),q)))
            print("Your Decryption Share Is =",(pow(c1,s,p)))

        else:
            print("User Not Present")

elif(num ==5):
    u = int(input("\nEnter your user number :"))

    if (u == 1):
        t = int(input("Enter your share :"))
        s = (t * ((u2*u3*u4*u5) * invmodp(((u2-u1)*(u3-
u1))*(u4-u1)*(u5-u1),q)))
        print("Your Decryption Share Is =",(pow(c1,s,p)))

    elif (u == 2):
        t = int(input("Enter your share :"))
        s = (t * ((u1*u3*u4*u5) * invmodp(((u1-u2)*(u3-
u2))*(u4-u2)*(u5-u2),q)))
        print("Your Decryption Share Is =",(pow(c1,s,p)))

    elif (u == 3):
        t = int(input("Enter your share :"))
        s = (t * ((u1*u2*u4*u5) * invmodp(((u1-u3)*(u2-
u3))*(u4-u3)*(u5-u3),q)))
        print("Your Decryption Share Is =",(pow(c1,s,p)))

    elif (u == 4):
        t = int(input("Enter your share :"))
        s = (t * ((u1*u2*u3*u5) * invmodp(((u1-u4)*(u2-
u4))*(u3-u4)*(u5-u4),q)))
        print("Your Decryption Share Is =",(pow(c1,s,p)))

    elif (u == 5):
        t = int(input("Enter your share :"))
        s = (t * ((u1*u2*u3*u4) * invmodp(((u1-u5)*(u2-
u5))*(u3-u5)*(u4-u5),q)))
        print("Your Decryption Share Is =",(pow(c1,s,p)))

    else:
            print("User Not Present")
```

Script 5 - Decryption & Verification

```
#**********************************DECRYPTION & VERIFICATION**********************************

from ecdsa import VerifyingKey, BadSignatureError
import pem

message_list = []
message_list1 = []
my_list3 = []
my_list4 = []
a = 'a'
e = 0
w = 0

print("\nType 1 to decrypt a String")
print("Type 2 to decrypt a Number")

k = int(input("Enter 1 or 2 : "))

if (k==1):
   #list_c2 = input("Enter List c2 : ")

   if (num ==3):
      d1 = int(input("Please Enter First Decryption Share : 
"))
      d2 = int(input("Please Enter Second Decryption 
Share : "))
      d3 = int(input("Please Enter Third Decryption Share 
: "))

      d = (d1 * d2 * d3) % p

   if (num ==4):
      d1 = int(input("Please Enter First Decryption Share : 
"))
      d2 = int(input("Please Enter Second Decryption 
Share : "))
      d3 = int(input("Please Enter Third Decryption Share 
: "))
      d4 = int(input("Please Enter Fourth Decryption 
Share : "))

      d = (d1 * d2 * d3 * d4) % p

   if (num ==5):
      d1 = int(input("Please Enter First Decryption Share : 
"))
      d2 = int(input("Please Enter Second Decryption 
Share : "))
      d3 = int(input("Please Enter Third Decryption Share 
: "))
      d4 = int(input("Please Enter Fourth Decryption 
Share : "))
      d5 = int(input("Please Enter Fifth Decryption Share : 
"))

      d = (d1 * d2 * d3 * d4 * d5) % p

   for i in range(0,len(list_c2)):
      message_list1.append(a)

   for i in range (0,len(list_c2)):
      m = ( list_c2[i] * invmodp(d,p)) % p
      message_list.append(m-w)
      w=w+1

   for i in range(0,len(list_c2)):
      message_list1[i] = chr(message_list[i])

   r = ''.join(message_list1)
   #print("\nOriginal Message Is =", r)

   string = r

   data_integrity2 = hash(r) % ((z+1) * 2)
   print("Hash Value of Recovered Message = 
",data_integrity2)

   for c in string:
      my_list3.append(c)

   for c in string:
      my_list4.append(ord(c))

   b_dec = [str(x) for x in my_list4]
   r_dec_int = ''.join(b)
   #r = int(r)

   r_dec = hash(r_dec_int) % ((z + 1) * 2)



   mbytes_dec = int_to_bytes(r_dec)

   if (f==1):
      try:
         vk1 = 
VerifyingKey.from_pem(open("public_key1.pem").read(
))
```

```
            assert vk1.verify(sig, mbytes_dec)
            print("\nSIGNATURE IS VALID !! \nMessage was 
Encrypted by User 1")
            print("\nOriginal Message Is =", r)
            e=1
        except BadSignatureError:
            print ("\nNot a Valid Signature!!")
    elif (f==2):
        try:
            vk2 = 
VerifyingKey.from_pem(open("public_key2.pem").read(
))
            assert vk2.verify(sig, mbytes_dec)
            print("\nSIGNATURE IS VALID !! \nMessage was 
Encrypted by User 2")
            print("\nOriginal Message Is =", r)
            e=1
        except BadSignatureError:
            print ("\nNot a Valid Signature!!")
    elif (f==3):
        try:
            vk3 = 
VerifyingKey.from_pem(open("public_key3.pem").read(
))
            assert vk3.verify(sig, mbytes_dec)
            print("\nSIGNATURE IS VALID !! \nMessage was 
Encrypted by User 3")
            print("\nOriginal Message Is =", r)
            e=1
        except BadSignatureError:
            print ("\nNot a Valid Signature!!")
    elif (f==4):
        try:
            vk4 = 
VerifyingKey.from_pem(open("public_key4.pem").read(
))
            assert vk4.verify(sig, mbytes_dec)
            print("\nSIGNATURE IS VALID !! \nMessage was 
Encrypted by User 4")
            print("\nOriginal Message Is =", r)
            e=1
        except BadSignatureError:
            print ("\nNot a Valid Signature!!")
    elif (f==5):
        try:
            vk5 = 
VerifyingKey.from_pem(open("public_key5.pem").read(
))
            assert vk5.verify(sig, mbytes_dec)
            print("\nSIGNATURE IS VALID !! \nMessage was 
Encrypted by User 5")
            print("\nOriginal Message Is =", r)
            e=1
        except BadSignatureError:
            print ("\nNot a Valid Signature!!")

    if (e==0):
        print("Message sent from unauthorised source")

elif (k==2):

    c2 = int(input("\nEnter C2 : "))
    if (num ==3):
        d1 = int(input("Please Enter First Decryption Share : 
"))
        d2 = int(input("Please Enter Second Decryption 
Share : "))
        d3 = int(input("Please Enter Third Decryption Share 
: "))

        d = (d1 * d2 * d3) % p

    if (num ==4):
        d1 = int(input("Please Enter First Decryption Share : 
"))
        d2 = int(input("Please Enter Second Decryption 
Share : "))
        d3 = int(input("Please Enter Third Decryption Share 
: "))
        d4 = int(input("Please Enter Fourth Decryption 
Share : "))

        d = (d1 * d2 * d3 * d4) % p

    if (num ==5):
        d1 = int(input("Please Enter First Decryption Share : 
"))
        d2 = int(input("Please Enter Second Decryption 
Share : "))
        d3 = int(input("Please Enter Third Decryption Share 
: "))
        d4 = int(input("Please Enter Fourth Decryption 
Share : "))
        d5 = int(input("Please Enter Fifth Decryption Share : 
"))

        d = (d1 * d2 * d3 * d4 * d5) % p

    m = ( c2 * invmodp(d,p)) % p
    #print("\nOriginal Message Is =", m)

    r = str(m)

    r = hash(r) % ((z + 1) * 2)

    l=str(m)
```

```
   data_integrity2 = hash(l) % ((z+1) * 2)

   mbytes_dec = int_to_bytes(r)

   if f==1:
      try:
         vk1 =
VerifyingKey.from_pem(open("public_key1.pem").read(
))
         assert vk1.verify(sig, mbytes_dec)
         print("\nSIGNATURE IS VALID !! \nMessage was
Encrypted by User 1")
         print("\nOriginal Message Is =", m)
         if data_integrity1 == data_integrity2:
            print("Message Integrity is Not Compromised")
         else:
            print("Message Integrity is Compromised!!!")
         e=1
         print("Hash Value of Recovered Message =
",data_integrity2)
      except BadSignatureError:
         print ("\nNot a Valid Signature!!")
   elif f==2:
      try:
         vk2 =
VerifyingKey.from_pem(open("public_key2.pem").read(
))
         assert vk2.verify(sig, mbytes_dec)
         print("\nSIGNATURE IS VALID !! \nMessage was
Encrypted by User 2")
         print("\nOriginal Message Is =", m)
         if data_integrity1 == data_integrity2:
            print("Message Integrity is Not Compromised")
         else:
            print("Message Integrity is Compromised!!!")
         e=1
         print("Hash Value of Recovered Message =
",data_integrity2)
      except BadSignatureError:
         print ("\nNot a Valid Signature!!")
   elif f==3:
      try:
         vk3 =
VerifyingKey.from_pem(open("public_key3.pem").read(
))
         assert vk3.verify(sig, mbytes_dec)
         print("\nSIGNATURE IS VALID !! \nMessage was
Encrypted by User 3")
         print("\nOriginal Message Is =", m)
         if data_integrity1 == data_integrity2:
            print("Message Integrity is Not Compromised")
         else:
            print("Message Integrity is Compromised!!!")
         e=1
         print("Hash Value of Recovered Message =
",data_integrity2)
      except BadSignatureError:
         print ("\nNot a Valid Signature!!")
   elif f==4:
      try:
         vk4 =
VerifyingKey.from_pem(open("public_key4.pem").read(
))
         assert vk4.verify(sig, mbytes_dec)
         print("\nSIGNATURE IS VALID !! \nMessage was
Encrypted by User 4")
         print("\nOriginal Message Is =", m)
         if data_integrity1 == data_integrity2:
            print("Message Integrity is Not Compromised")
         else:
            print("Message Integrity is Compromised!!!")
         e=1
         print("Hash Value of Recovered Message =
",data_integrity2)
      except BadSignatureError:
         print ("\nNot a Valid Signature!!")
   elif f==5:
      try:
         vk5 =
VerifyingKey.from_pem(open("public_key5.pem").read(
))
         assert vk5.verify(sig, mbytes_dec)
         print("\nSIGNATURE IS VALID !! \nMessage was
Encrypted by User 5")
         print("\nOriginal Message Is =", m)
         if data_integrity1 == data_integrity2:
            print("Message Integrity is Not Compromised")
         else:
            print("Message Integrity is Compromised!!!")
         e=1
         print("Hash Value of Recovered Message =
",data_integrity2)
      except BadSignatureError:
         print ("\nNot a Valid Signature!!")

   if (e==0):
       print("\nMessage sent from unauthorised
source!!!")


else:
   print("Invalid Choice! Choose Again")
```