

Name - Aditya Patidar
Roll no. - 18075006
Dept. - CSE (B.Tech)
Network Security PA-5

Github Link -

<https://github.com/Aditya-Patidar/Network-Security-Assignments-18075006-/tree/main/Practical%20Assignment%205>

Objective Implement the El Gamal algorithm in Python

Basic Theory

The Gamal encryption system is an example of asymmetric key encryption algorithm for public-key cryptography. It is based on Diffie–Hellman key exchange. The Digital Signature Algorithm (DSA) is a variant of the El Gamal signature scheme, which should not be confused with El Gamal encryption

Uses:- Hybrid cryptosystem uses this algorithm.

Algorithm: El Gamal Encryption Algorithm has three parts

- A key generator
- The encryption algorithm
- The decryption algorithm

Three parts:

- **Key Generation:**

- The receiver chooses a very large number q and a cyclic group F_q .
- From the cyclic group F_q , he choose any element g and an element a such that $\gcd(a, q) = 1$
- Then computes $h = g^a$
- The receiver publishes F , $h = g^a$, q , and g as his public key and retain a as private key

- **Encryption:**

- Sender selects an element k from cyclic group F

such that $\gcd(k, q) = 1$. ○ Then computes $p = g^k$ and $s = h^k = g^{(a \cdot k)}$.

○ Multiply s with M

○ Then send the receiver $(p, M \cdot s) = (g^k, M \cdot s)$

● **Decryption:**

○ Receiver calculates $s' = p^a = g^{(a \cdot k)}$.

○ Receiver divides $M \cdot s$ by s' to obtain M as $s = s'$.

Source code:

```
# Python program to illustrate ElGamal encryption
```

```
import random
```

```
from math import pow
```

```
a = random.randint(2, 10)
```

```
def gcd(a, b):
```

```
    if a < b:
```

```
        return gcd(b, a)
```

```
    elif a % b == 0:
```

```
        return b;
```

```
    else:
```

```
        return gcd(b, a % b)
```

```
# Generating large random numbers
```

```
def gen_key(q):
```

```
    key = random.randint(pow(10, 20), q)
```

```
    while gcd(q, key) != 1:
```

```
        key = random.randint(pow(10, 20), q)
```

```
    return key
```

```
# Modular exponentiation
```

```
def power(a, b, c):
```

```
    x = 1
```

```
    y = a
```

```

while b > 0:
    if b % 2 != 0:
        x = (x * y) % c;
    y = (y * y) % c
    b = int(b / 2)

return x % c

# Asymmetric encryption
def encrypt(msg, q, h, g):

    en_msg = []

    k = gen_key(q)# Private key for sender
    s = power(h, k, q)
    p = power(g, k, q)

    for i in range(0, len(msg)):
        en_msg.append(msg[i])

    print("g^k used : ", p)
    print("g^ak used : ", s)
    for i in range(0, len(en_msg)):
        en_msg[i] = s * ord(en_msg[i])

    return en_msg, p

def decrypt(en_msg, p, key, q):

    dr_msg = []
    h = power(p, key, q)
    for i in range(0, len(en_msg)):
        dr_msg.append(chr(int(en_msg[i]/h)))

    return dr_msg

# Driver code
def main():

    msg = 'Implement the El Gaamal algorithm in Python'
    print("Message from senders side :", msg)

    q = random.randint(pow(10, 20), pow(10, 50))

```

```
g = random.randint(2, q)
```

```
key = gen_key(q) # Private key for receiver
```

```
h = power(g, key, q)
```

```
print("g used : ", g)
```

```
print("g^a used : ", h)
```

```
en_msg, p = encrypt(msg, q, h, g)
```

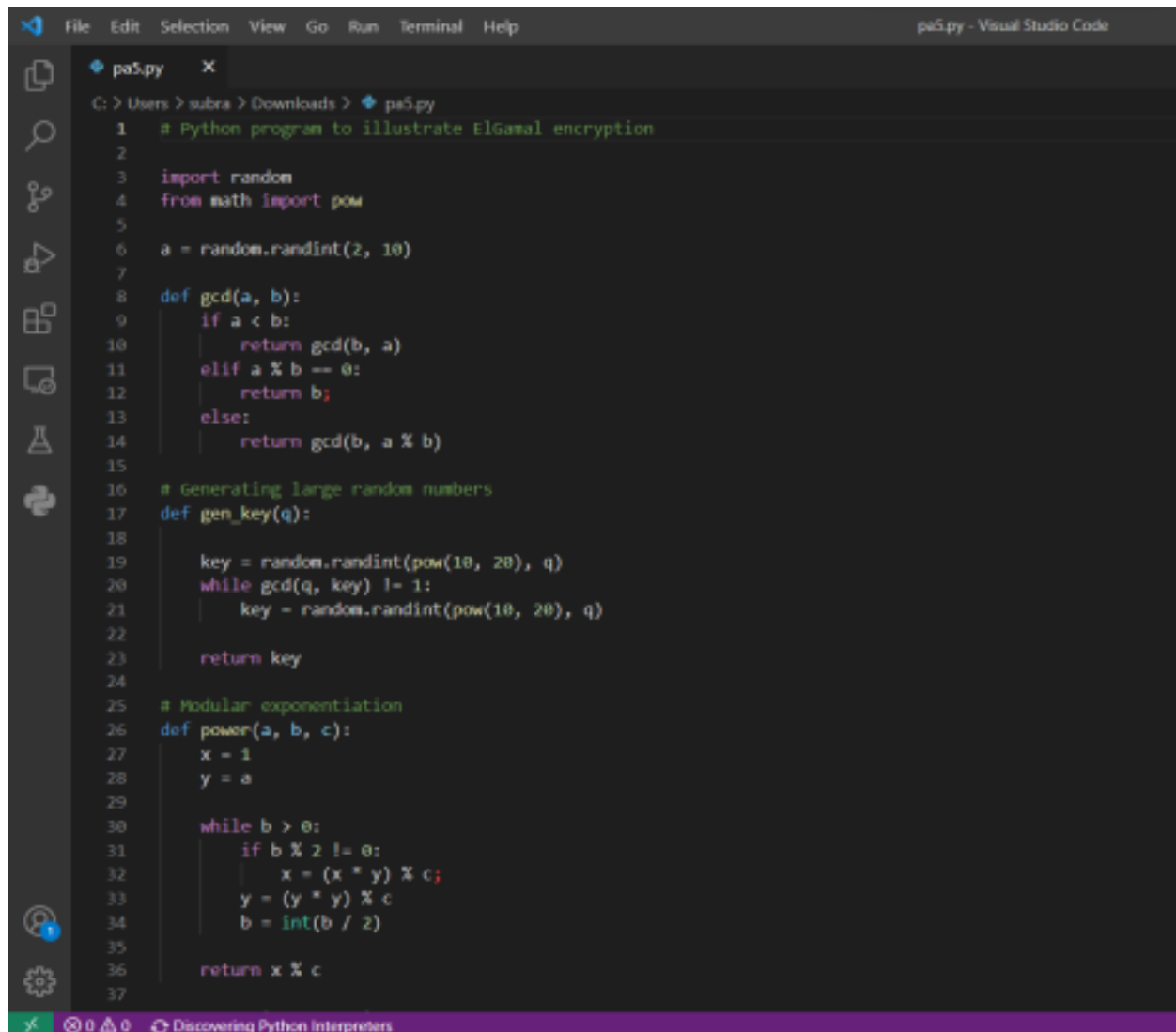
```
dr_msg = decrypt(en_msg, p, key, q)
```

```
dmsg = ".join(dr_msg)
```

```
print("Decrypted Message at receiver side :", dmsg);
```

```
if __name__ == '__main__':  
    main()
```

Screenshots



```
File Edit Selection View Go Run Terminal Help
pa5.py X
C: > Users > subra > Downloads > pa5.py
1  # Python program to illustrate ElGamal encryption
2
3  import random
4  from math import pow
5
6  a = random.randint(2, 10)
7
8  def gcd(a, b):
9      if a < b:
10         return gcd(b, a)
11     elif a % b == 0:
12         return b;
13     else:
14         return gcd(b, a % b)
15
16  # Generating large random numbers
17  def gen_key(q):
18
19     key = random.randint(pow(10, 20), q)
20     while gcd(q, key) != 1:
21         key = random.randint(pow(10, 20), q)
22
23     return key
24
25  # Modular exponentiation
26  def power(a, b, c):
27     x = 1
28     y = a
29
30     while b > 0:
31         if b % 2 != 0:
32             x = (x * y) % c;
33         y = (y * y) % c
34         b = int(b / 2)
35
36     return x % c
37
```

```

23     return key
24
25     # Modular exponentiation
26     def power(a, b, c):
27         x = 1
28         y = a
29
30         while b > 0:
31             if b % 2 != 0:
32                 x = (x * y) % c;
33             y = (y * y) % c
34             b = int(b / 2)
35
36         return x % c
37
38     # Asymmetric encryption
39     def encrypt(msg, q, h, g):
40
41         en_msg = []
42
43         k = gen_key(q) # Private key for sender
44         s = power(h, k, q)
45         p = power(g, k, q)
46
47         for i in range(0, len(msg)):
48             en_msg.append(msg[i])
49
50         print("g^k used : ", p)
51         print("g^ak used : ", s)
52         for i in range(0, len(en_msg)):
53             en_msg[i] = s * ord(en_msg[i])
54
55         return en_msg, p
56

```

```
en_msg[i] = chr((en_msg[i] + h))
```

```
return en_msg, p
```

```
def decrypt(en_msg, p, key, q):
```

```
dr_msg = []
```

```
h = power(p, key, q)
```

```
for i in range(0, len(en_msg)):
```

```
    dr_msg.append(chr(int(en_msg[i]/h)))
```

```
return dr_msg
```

```
# Driver code
```

```
def main():
```

```
msg = 'Implement the El Gaamal algorithm in Python'
```

```
print("Message from senders side :", msg)
```

```
q = random.randint(pow(10, 20), pow(10, 50))
```

```
g = random.randint(2, q)
```

```
key = gen_key(q)# Private key for receiver
```

```
h = power(g, key, q)
```

```
print("g used : ", g)
```

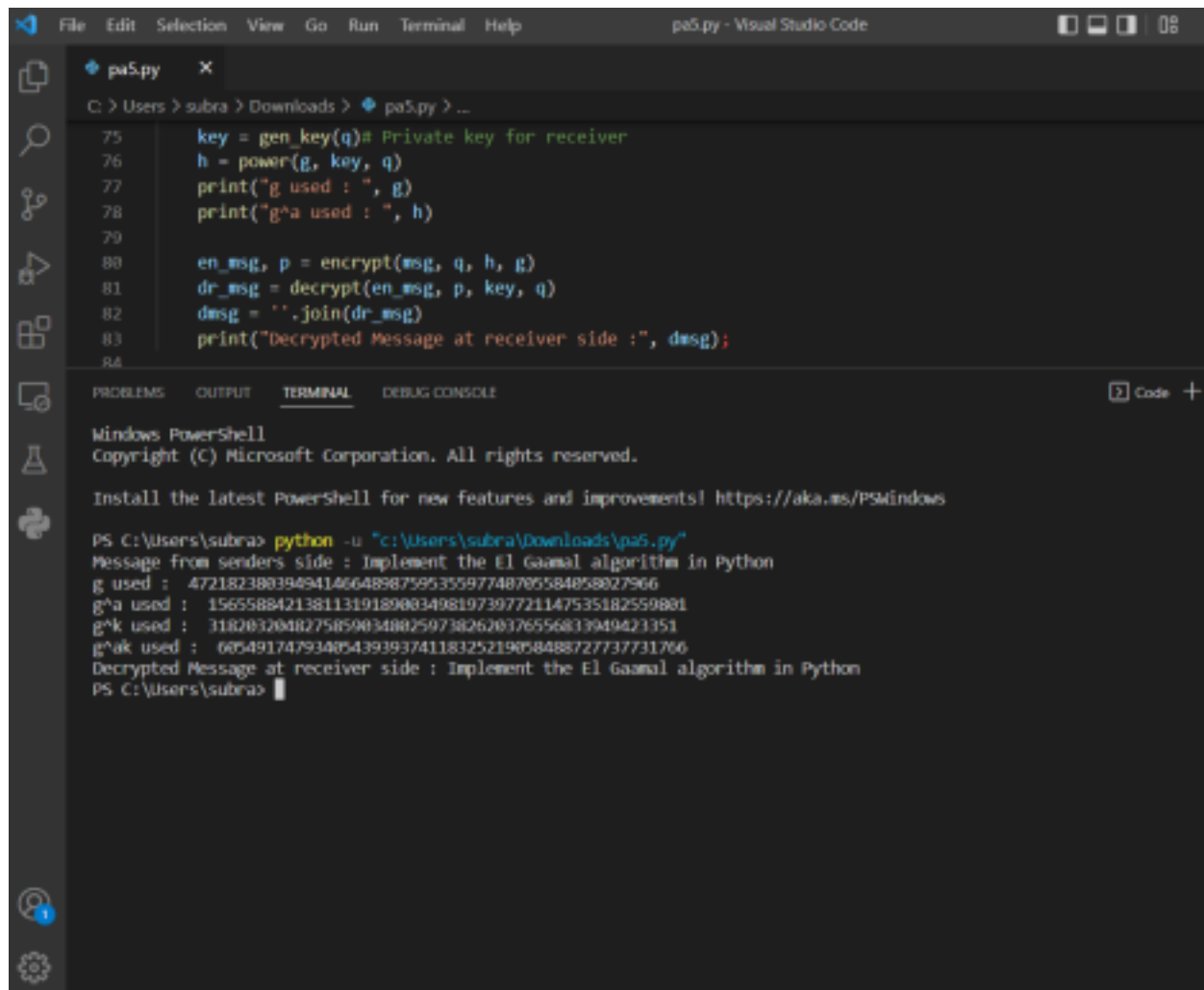
```
print("g^a used : ", h)
```

```
en_msg, p = encrypt(msg, q, h, g)
```

```
dr_msg = decrypt(en_msg, p, key, q)
```

```
dmsg = ''.join(dr_msg)
```

```
print("Decrypted Message at receiver side :", dmsg);
```



The image shows a Visual Studio Code window with a Python file named `pa5.py` open. The file is located at `C:\Users\subra\Downloads\pa5.py`. The code implements the El Gamal encryption and decryption algorithm. The terminal window shows the output of running the script, displaying the generated private key, the values of g and g^a , the encrypted message, and the decrypted message.

```
75     key = gen_key(q)# Private key for receiver
76     h = power(g, key, q)
77     print("g used : ", g)
78     print("g^a used : ", h)
79
80     en_msg, p = encrypt(msg, q, h, g)
81     dr_msg = decrypt(en_msg, p, key, q)
82     dmsg = ''.join(dr_msg)
83     print("Decrypted Message at receiver side :", dmsg);
84
```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS C:\Users\subra> python -u "c:\Users\subra\Downloads\pa5.py"
Message from senders side : Implement the El Gamal algorithm in Python
g used : 4721823883949414664898759535597748765584058027966
g^a used : 15655884213811319189003498197397721147535182559801
g^k used : 3182032048275059034802597382620376556833949423351
g^ak used : 6054917479340543939374118325219058488727737731766
Decrypted Message at receiver side : Implement the El Gamal algorithm in Python
PS C:\Users\subra>