
ELGAMAL: Report Project Hand-in 1

Lamberto Ragnolini

Fall Semester, September 27, 2024

1 OVERALL PREMISE

Given the freedom of choice in the implementation I decided to implement the structure as a Class named ELGAMAL so that all the implemented functions, that are part of the class, will be able to view the shared parameters (base, shared prime etc..). The function calls are at the end of the python script with the initialization of the class. This generalizations gives the user the ability of a ease change of parameters due to its implementation with a class, just by changing the three variables at the end of the script the user will be able to run the elgamal encryption with different numbers. **Also very important, in my implementation I assume that every time Alice sends a message a new exchange is started, that is why in the send function I generate a new random r every time. This part of the implementation was supervised and approved by the Teaching assistant.**

2 QUESTION 1

- You are Alice and want to send 2000 kr. to Bob through a confidential message. You decide to use the ElGamal public key method. The keying material you should use to send the message to Bob is as follows:
 - The shared base $g=666$
 - The shared prime $p=6661$
 - Bob's public key $P K = g \times \text{mod } p = 2227$

Send the message '2000' to Bob.

- code

```
1 # this function encrypts c1
2 def compute_c1(self, r):
3     return (self.g**r) % self.p
4
5 # this function encrypts c2
6 def compute_c2(self, M, r):
7     return (M*(self.PK**r)) % self.p # PK^k is the mask that later Bob will
8     be able to remove with his x
9
10 # this function is used to encrypt the message and send it
```

```

10 def send(self,message):
11     r = random.randint(1, self.p-1) #Alices secret key that is chose at
        random every time, remember it should be smaller than our p
12     c1 = self.compute_c1(r)
13     c2 = self.compute_c2(message,r)
14     cypher_pair = (c1,c2) #both numbers must be in the range 0...p-1
15     print(f"this is the encrypted sent pair : {cypher_pair}")
16     return cypher_pair

```

- Explanation

- The main function is the send(), this function takes as input parameters the message to send, in our case 2000. I create the secret key of Alice by selecting at random a number that is less than the prime number chosen for the encryption, once that is done I compute c1 and c2 for the encryption following the formulas seen in class and taken from the book. In order to handle the math and for more readability I create two auxilliary functions that compute our c1 and c2. Compress everything into a pair and return the encrypted pair.

3 QUESTION 2

- You are now Eve and intercept Alice's encrypted message. Find Bob's private key and reconstruct Alice's message.
- code

```

1 # this auxilliary function is used to decyrpt the message that Alice sendd to
  Bob
2 def decryption(self,c1,c2,x):
3     hide_msg = (c1**x) % self.p # this means doin -> (g^r)^x mod p
4     # To remove the hiding now we have to :
5     # compute modular inverse of hide msg
6     # compute mod p of the modular inverse
7     decryption = (pow(hide_msg,-1,self.p) * c2 ) % self.p
8     return decryption
9
10 # This function is used to intercept the message that ALice sendd and find
   Bob's private key
11 # returns the decrypted message
12 def interceptance(self,cypher_pair):
13     # explode the pair
14     c1 = cypher_pair[0]
15     c2 = cypher_pair[1]
16     # now we need to find x of Bob knowing p,g,PK
17     # we know PK = g^x mod p
18     # what we wanna do is find that x that given as exponent to g gives 2227
        when we apply mod p
19     # This method uses a brute force approach
20     x = 0
21     for i in range(self.p):
22         if ((self.g**i) % self.p) == self.PK:
23             x = i
24             print(f"Bobs secret key is {x}")
25             break
26

```

```

27     dec = self.decryption(c1,c2,x)
28     print(f"the decrypted message is {dec}")
29     return dec

```

- Explanation

- The main function is the interceptance() function, In this case Eve is able to intercept the encrypted message meaning the cypher pair sent before by Alice so this becomes the parameter to pass to our function. What I do is first of all explode the pair to retrieve c1 and c2, after that given that p is a small prime number we are able to exploit elgamal vulnerability by brute forcing with a loop to discover Bob's private key. This is pretty simple given that we know both the mechanism to create the public key and the value of the public key which are $P_K = g^x \mod p = 2227$, from here we simply need to find that x that elevated to $g=666 \mod p=6661$ that gives as output 2227. A loop will easily retrieve that given the small nature of our selected p. Once our algorithm has found Bob's private key we will simply use the math seen in the lectures and present also in the book to compute the decryption. As before for readability and for a better coding scheme I implemented an auxilliary function that takes care of that.

4 QUESTION 3

- You are now Mallory and intercept Alice's encrypted message. However, you run on a constrained device and are unable to find Bob's private key. Modify Alice's encrypted message so that when Bob decrypts it, he will get the message '6000'. (We assume that Mallory knows the message is 2000!!!!, this was told by Teaching assistant in exercise session)
- code

```

1  # This function is used to intercept the message Alice sent to Bob and change it
   # 's content
2  def Mallory_intercept(self,cypher_pair, new_message):
3      # explode the pair
4      c1 = cypher_pair[0]
5      new_c2 = cypher_pair[1] *new_message
6      new_cypher_pair = (c1,new_c2)
7      print(f"the new encrypted message is {new_cypher_pair}")
8      return new_cypher_pair

```

- Explanation

- Here a simply function is used to solve this problem. Due to the fact that Mallory knows that the sent message is 2000 as Teaching assistant told us, all that mallory needs to do is without decrypting the message, multiply the encrypted message c2 by a certain amount, in our case 3, so that when Bob will try do decypher the message he will decypher $c2=2000*3=6000$. I wanted to generalize the function giving the user the ability of deciding by how much to multiply the quantity so the multiplication parameter is a parameter that I pass to the function. We need to do this to make sure that the r used in the message exchange is not changed and that the pair is the same one. Once done with the multiplication we will reform the pair and send it back by return it.

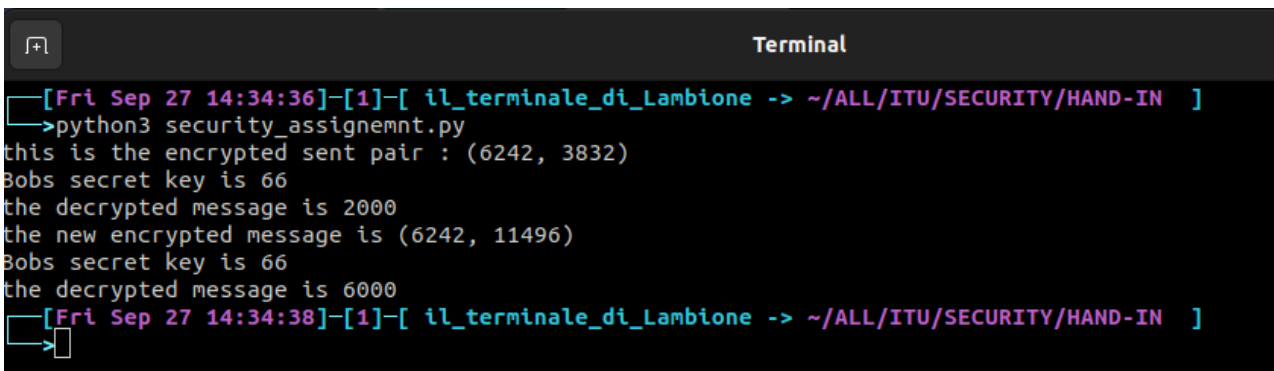
5 FUNCTION CALLS AND VALIDATION

- code with initialization of the class, and function calls

```
1 # initializations and function calls
2
3 # this are the parameters you can use for your ELGAMAL encryption
4 # change this for different outcomes
5 p = 6661
6 g = 666
7 PK = 2227
8
9 # payloads
10 Alice_msg = 2000
11 # ASSUMING THAT THE RECEIVED VALUE IS ALREADY 2000
12 Mallory_msg = 3
13
14 # instance of the class
15 E = ELGAMAL(p,g,PK)
16
17 # function calls for the exercises
18
19 # ex1
20 pair = E.send(Alice_msg)
21
22 # ex2
23 E.interceptance(pair)
24
25 # ex3
26 Mallory_encrypted_msg = E.Mallory_intercept(pair, Mallory_msg)
27 # this function call is done to show that when Bob will decrypt he will see
    Mallory payload (the changed message)
28 E.interceptance(Mallory_encrypted_msg)
```

The last function call is made to show that Mallory evil act was proficient and that Bob will effectively read 6000 when decrypting the message

- output when running the script



```
[Fri Sep 27 14:34:36]-[1]-[ il_terminale_di_Lambione -> ~/ALL/ITU/SECURITY/HAND-IN ]
python3 security_assignemnt.py
this is the encrypted sent pair : (6242, 3832)
Bobs secret key is 66
the decrypted message is 2000
the new encrypted message is (6242, 11496)
Bobs secret key is 66
the decrypted message is 6000
[Fri Sep 27 14:34:38]-[1]-[ il_terminale_di_Lambione -> ~/ALL/ITU/SECURITY/HAND-IN ]
```

Figure 1: output of the script