ElGamalRedis

ElGamal Encryption over Redis and Index Calculus Attack

Agnese Salutari

agneses92@gmail.com

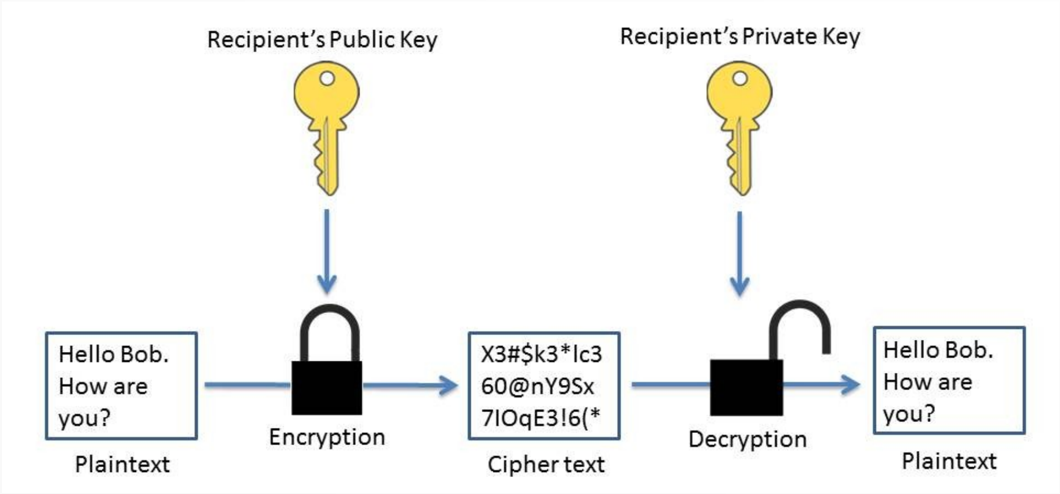
Table of Contents

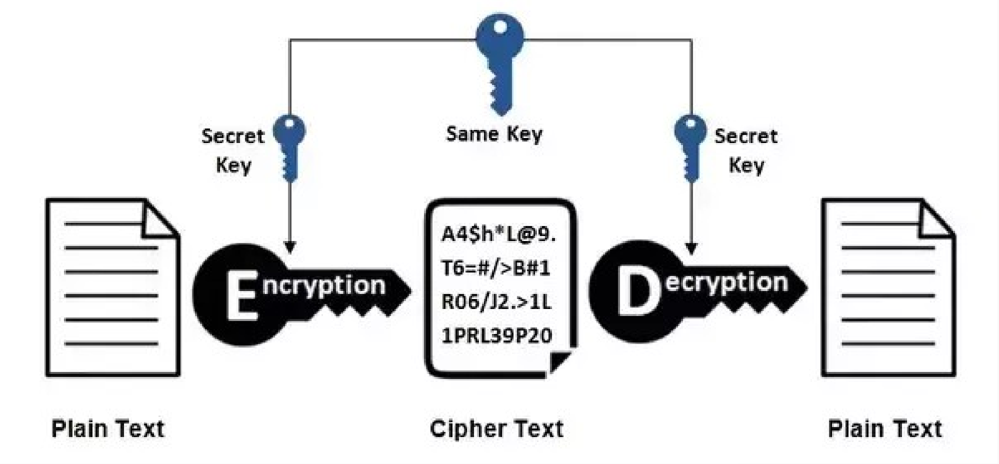
Encryption Methods

Encryption methods are divided in two categories:

Public Key Encryption

Symmetric Encryption





Encryption Methods - Sessions

Usually, these methods are combined together in Sessions:

Public Key Encryption is used by sender-receiver couples to share the Symmetric Encryption key, that will be used for message delivery.

Sessions allow us to use the best of each method:

- Public Key Encryption is expensive, but is secure to deliver data on a public channel (for the first time).
- Symmetric Encryption is cheap, but can’t be used on a public channel if the key has not been shared yet.

ElGamal Method

ElGamal Encryption is a Public Key Encryption method.

The security of Elgamal Encryption is based on Discrete Logarithm of big integers, because this function is:

- Easy and fast to use by sender during encryption and by receiver during decryption.
- Hard to compute for undesidered sniffers, who miss pieces of information.

Discrete Logarithm Problem

Given a prime number p, let a and b be non-zero integers modulo p.

Find x such that:

$$a^x \equiv b \pmod p$$

The solution of this problem, x, is the Discrete Logarithm of b with respect to a:

$$x = L_a(b)$$

ElGamal Encryption

Scenario: Alice wants to send a message, m, to Bob.

- Bob chooses a big prime number, p, a primitive root of that prime, a, and an integer, e.

Then, he computes: $a^e \equiv b \left(mod\ p \right)$

He publishes his Public Key, (p, a, b).

Only Bob will know his Private Key, e.

ElGamal Encryption

Scenario: Alice wants to send a message, m, to Bob.

- Alice takes Bob's Public Key, (p, a, b), and chooses an integer, k.

Then she computes:

- $a^k \equiv r \pmod{p}$
- $b^k m \equiv t \pmod{p}$; m < p

She sends (r, t) to Bob.

Only Alice knows k.

ElGamal Encryption

Scenario: Alice wants to send a message, m, to Bob.

- Bob receives (r, t) and decrypt the message by computing:

$$t\,r^{-e} \equiv m \left(mod\ p \right)$$

This works because:

$$t\,r^{-e} \equiv b^{k}\, m\, a^{-ek} \equiv a^{ek}\, m\, a^{-ek} \equiv m \left(mod\ p \right)$$

A spy, Eve, could take Bob’s Public Key and listen to the channel, sniffing what goes through, that is (r, t):

The message is encrypted and Eve doesn’t know e or k.

To find such an exponent is a Discrete Logarithm Problem, computationally too hard to compute (if the numbers involved in the Logarithm are big), so she can’t decrypt the message.

Index Calculus Algorithm

Input: A Discrete Logarithm Problem $a^e \equiv b \,(mod\; p)$

Output: The solution, e.

- relations = [] is a matrix.
- Base = $[p_0, p_1, \ldots, p_r]$ is a base of r+1 prime factors.
- For r = 1, 2, …
  - Try to factor $a^r \,(mod\; p)$ using the base.
  - If a factorization is found, form a vector with the exponents used for each factor and k:

    $$[f_0, f_1, \ldots, f_r, k]$$

  - If this vector is Linear Independent with relations rows: append it as a row to relations.

  Exit loop when relations contains r+1 rows.
- Obtain the Reduced Echelon Form of the relations matrix and keep the last column. This column contains the discrete logarithms (to the base a) of the base factors:

$$[l_0, l_1, \ldots, l_r]$$

Index Calculus Algorithm

- For s = 1, 2, …
  - Try to factor $a^s b \pmod p$ with the base.
  - When a factorization is found:
    - Keep the exponents used:

$$[f_0, f_1, \ldots, f_r, k]$$

    - The solution is:

$$e = f_0 l_0 + f_1 l_1 + \ldots + f_r l_r - s$$

ElGamalRedis - Project

ElGamalRedis is a modular project containing:

- An ElGamal Cryptosystem (implemented in Python 3)
  - On a real public communication channel (Python 3 and Redis)
  - Used by a distributed system (composed of Python 3 daemons)
- An Index Calculus Attack:
  - Performed by a real sniffer on the channel (a Python 3 daemon)

ElGamalRedis - Technologies

- Why Python 3?

  Python 3 is the last Python release. Python is a powerful language and is widely used.

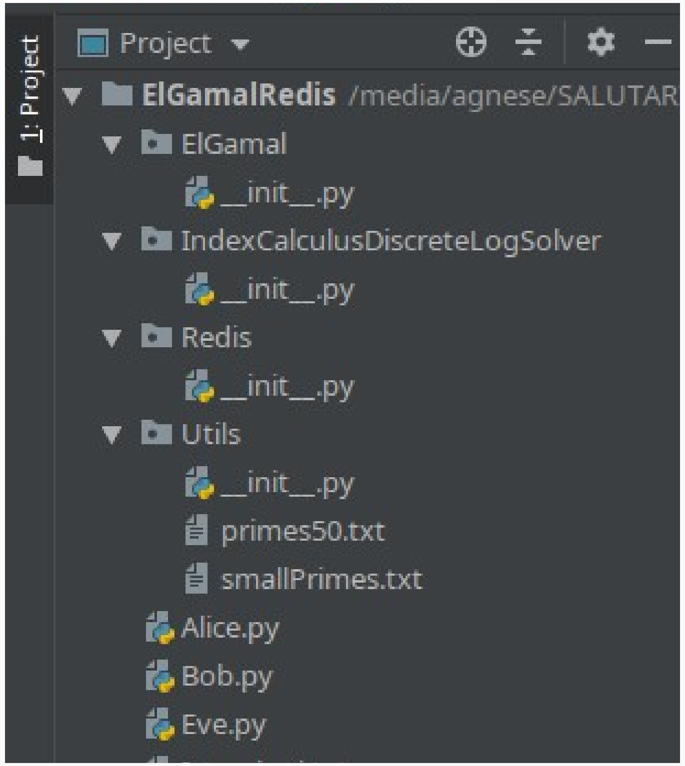
- Why Redis?

  Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker.

  https://redis.io/

ElGamalRedis - Structure

- To study the problem at a deep level, I didn't use Python 3 libraries for Cryptography. So I made the following libraries:
  - Utils/ModularArithmetics:
    containing Modular Arithmetics functions.
  - Redis:
    to manage the communication channel.
  - ElGamal:
    to create and manage ElGamal Cryptosystems.
  - IndexCalculusDiscreteLogSolver:
    to create Index Calculus Attackers.
- Alice.py, Bob.py and Eve.py are daemons:
  they use my libraries to work.

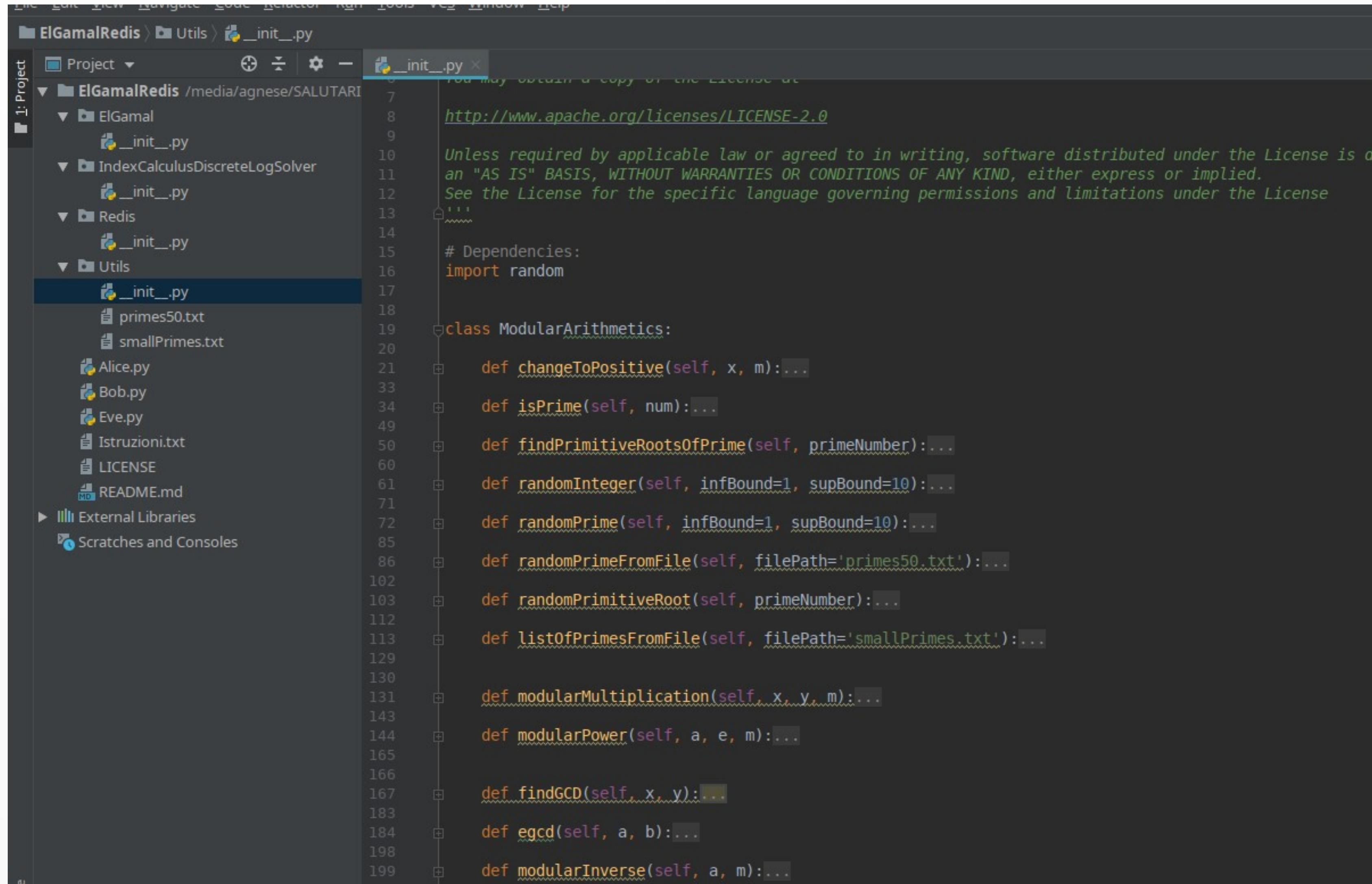


ElGamalRedis - Features

- ElGamalRedis libraries can extract prime numbers contained in files (if the path is setted):
  This is useful, because finding big primes is time-consuming.
  Daemons use this method for key generation.
- ElGamalRedis libraries can compute primitive roots of a prime number or simply initialize a as a random integer between 2 and p – 1 (depending on settings):
  I recommend the second option (used by my daemons too), because primitive big prime roots computation is hard.
- ElGamalRedis libraries allow to delete matrix all-zeros columns (and update its base of factors):
  For Index Calculus attacks, the dimension of the base can be choosen during the initialization, but can be difficult to choose the best value a priori, so there could be a lot of all-zeroes columns.
  Deleting them is helpful for computation: the less the columns involved, the less the LI rows needed.
- The number of rounds to perform during Index Calculus attacks can be setted too.
- ElGamalRedis system works on localhost by default, but every module can be setted to work on a different machine, so:
  - Parallel computation, useful for Index Calculus Attack, can be easily performed.
  - You can see the dynamics of a real network.

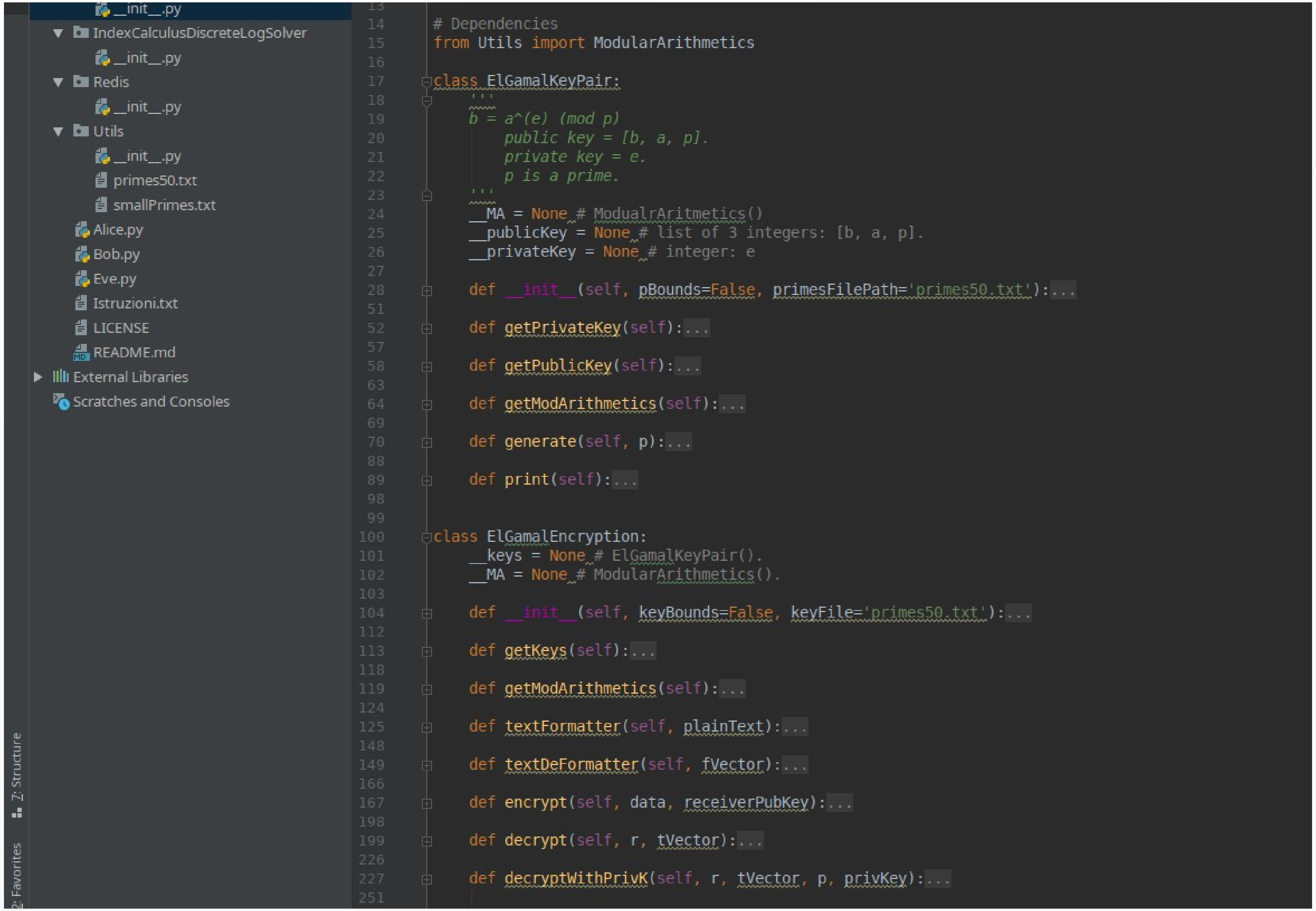
ElGamalRedis - Structure

Modular Arithmetics library



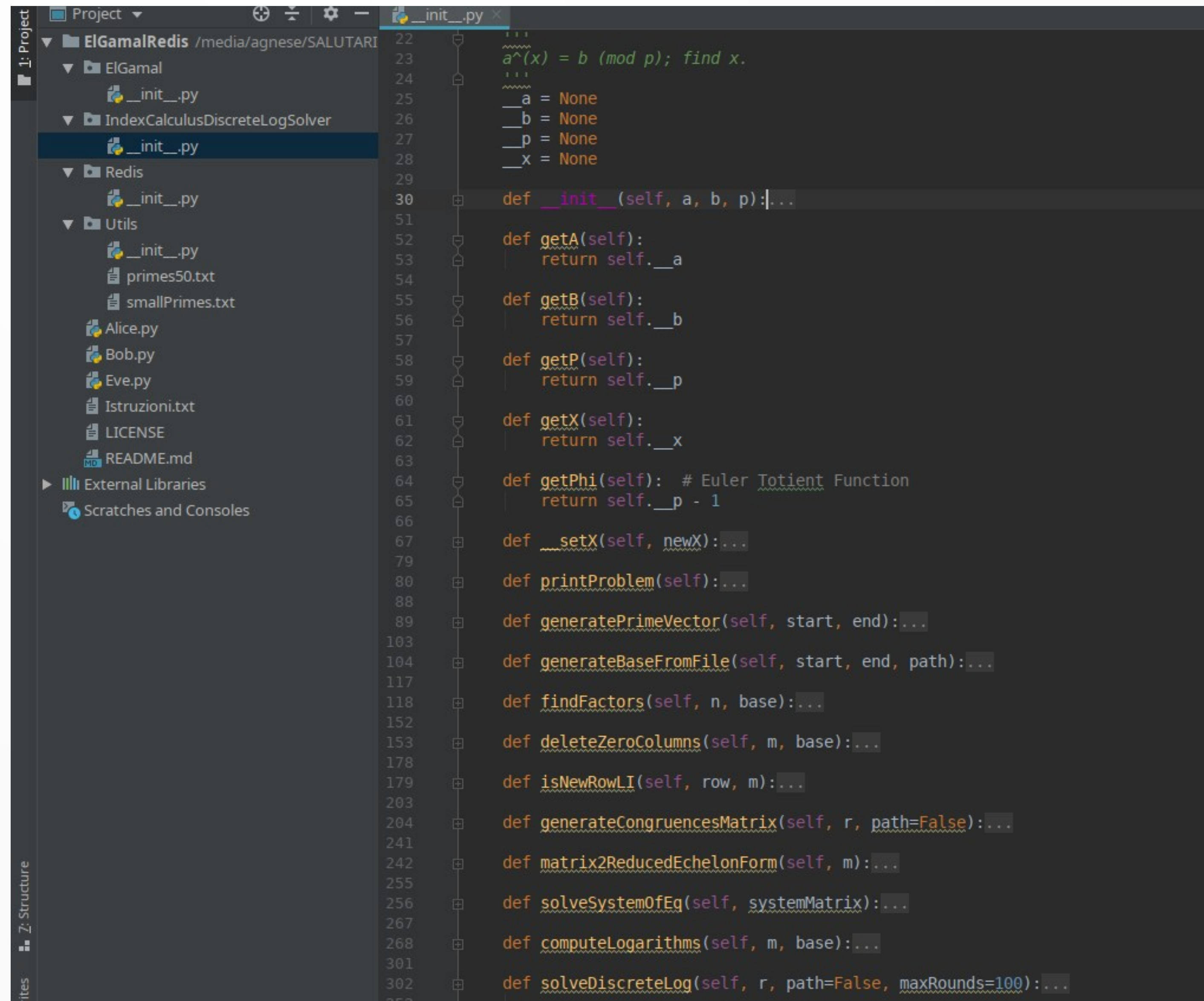
ElGamalRedis - Structure

ElGamal library



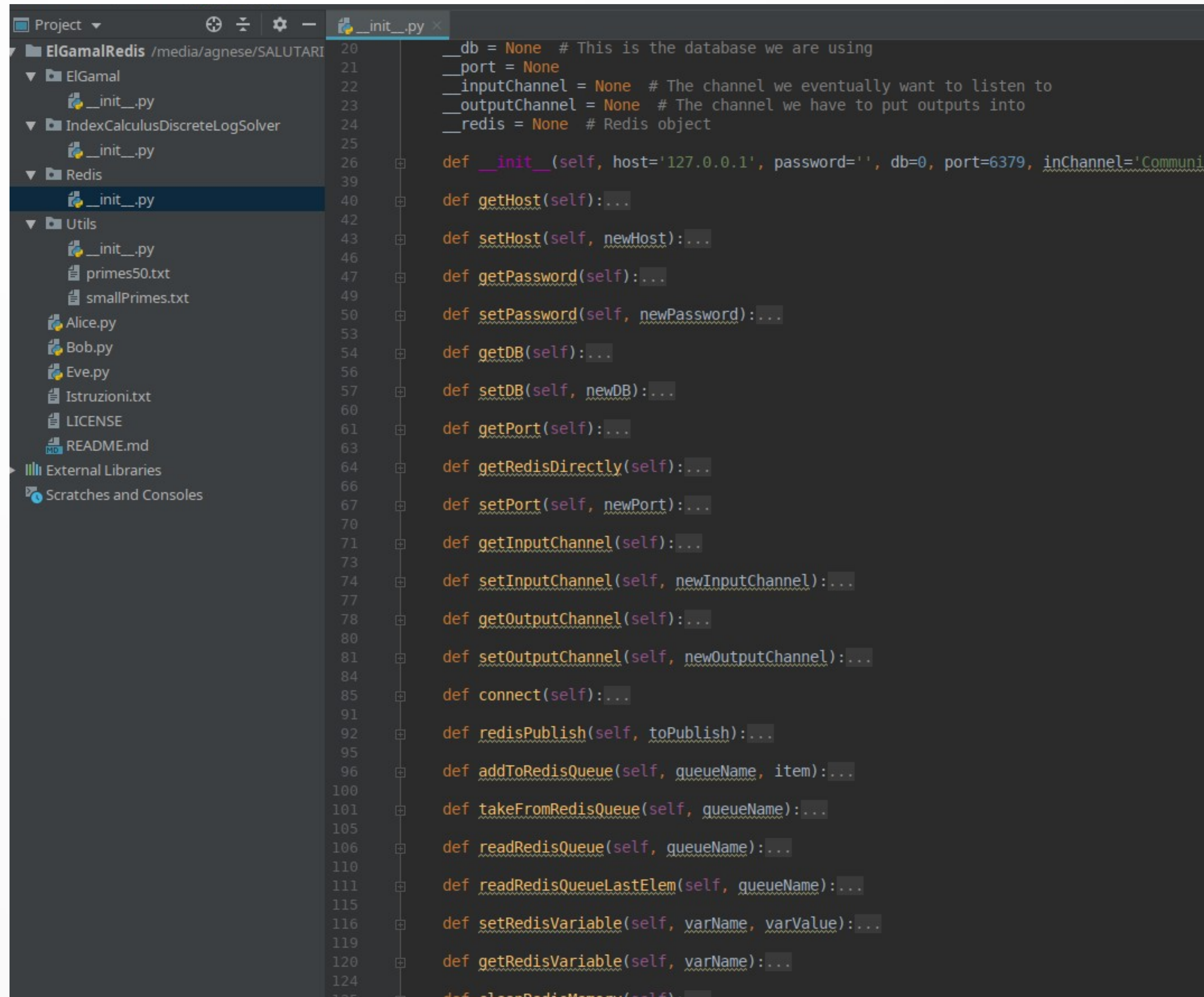
ElGamalRedis - Structure

Index Calculus library



ElGamalRedis - Structure

Redis library



ElGamalRedis - Structure

Bob

```
RCh = Redis.RedisChannel()
print("Bob creates his ElGamal Keys: ")
BobElGamal = eg.ElGamalEncryption(False, keyFile='Utils/primes50.txt')
print("Bob connects and registers his Public Key on the channel:")
RCh.connect()
RCh.setRedisVariable(varName='BobPublicKey', varValue=str(BobElGamal.getKeys().getPublicKey()))
print("Bob waits for Alice's messages from the channel:")
pubsub = RCh.getRedisDirectly().pubsub()
channelName = 'CommunicationChannel'
pubsub.subscribe(channelName)
for item in pubsub.listen():
    if item['type'] == 'message':
        msg = ast.literal_eval(item['data'].decode('utf-8'))
        print('Message arrived: ' + str(msg))
        r = int(msg[0])
        tVector = msg[1]
        decodedText = BobElGamal.decrypt(r=r, tVector=tVector)
        print('Decoded Text: ' + decodedText)
```

Alice

```
RCh = Redis.RedisChannel()
print("Alice creates her ElGamal Keys: ")
AliceElGamal = eg.ElGamalEncryption(False, keyFile='Utils/primes50.txt')
print("Alice connects and reads Bob's Public key from the channel:")
RCh.connect()
BobPubKey = ast.literal_eval(RCh.getRedisVariable('BobPublicKey').decode('utf-8'))
print('Bob Key is: ' + str(BobPubKey))
plainText = 'ciao!'
print('Plain Text: ' + str(plainText))
print('Alice encrypts her message:')
encrypted = AliceElGamal.encrypt(data=plainText, receiverPubKey=BobPubKey)
print(encrypted)
print('Alice sends her message.')
RCh.redisPublish(str(encrypted))
```

ElGamalRedis - Structure

Eve

```python
print('TEST: x = 30, a = 1520  ##########################################################')
ic = IC.IndexCalculus(1520, 15203215, 15485863)
res = ic.solveDiscreteLog(r=20, maxRounds=500)

print('TEST: x = 100, a = 16720  ##########################################################')
ic = IC.IndexCalculus(16720, 5263484, 15485863)
res = ic.solveDiscreteLog(r=20, maxRounds=1000)



print('Eve connects and reads Bob Public Key from the channel: ')
RCh = Redis.RedisChannel()
RCh.connect()
BobPubKey = ast.literal_eval(RCh.getRedisVariable('BobPublicKey').decode('utf-8'))
print('Bob Key is: ' + str(BobPubKey))
print('Eve sniffs the channel, waiting for messages addressed to Bob...')
pubsub = RCh.getRedisDirectly().pubsub()
channelName = 'CommunicationChannel'
pubsub.subscribe(channelName)
for item in pubsub.listen():
    if item['type'] == 'message':
        msg = ast.literal_eval(item['data'].decode('utf-8'))
        print('Message sniffed: ' + str(msg))
        r = int(msg[0])
        tVector = msg[1]
        ICProblem = IC.IndexCalculus(a=BobPubKey[1], b=BobPubKey[2], p=BobPubKey[0])
        print("Eve tries to calculate Bob's private key via Index Calculus...")
        BobPrivKey = ICProblem.solveDiscreteLog(r=100, maxRounds=1000)
        print("But she can't, because it's too big to compute!")
```

ElGamalRedis – How it works

Alice.py, Bob.py and Eve.py are daemon processes that interact via Redis only:

- Bob.py is a process that creates its own Public and Private Keys, publishes the public one, waits for Alice's message and decrypts it.
- Eve.py is a sniffer process, that uses Index Calculus Algorithm to try to solve Discrete Logarithm problems (obviously, it can solve little factors numbers problems only).
- Alice.py is a process that reads Bob's Public Key, encrypts a message and sends it to Bob.py.

ElGamalRedis – Environment Setup

- Install Redis (https://redis.io/):
  - sudo apt update
  - sudo apt install redis-server
- The supervised directive is set to no by default. If you are running Ubuntu, which uses the systemd init system, find it and change it to systemd:
  - sudo nano /etc/redis/redis.conf
- Restart Redis:
  - sudo systemctl restart redis.service
- If you don't want Redis to be a startup program:
  - sudo systemctl disable redis

ElGamalRedis – Let’s run it

- You have to run Redis first:
  - redis-server
- (Optional) If you want to see what's happening on Redis:
  - redis-cli monitor
- You have to run
  - Bob.py
  - Eve.py
  - Alice.py

  in that order, because Alice and Eve need to read Bob's Public Key from Redis and because Eve needs to listen to the channel waiting for Alice's messages.

ElGamalRedis – Let's run it

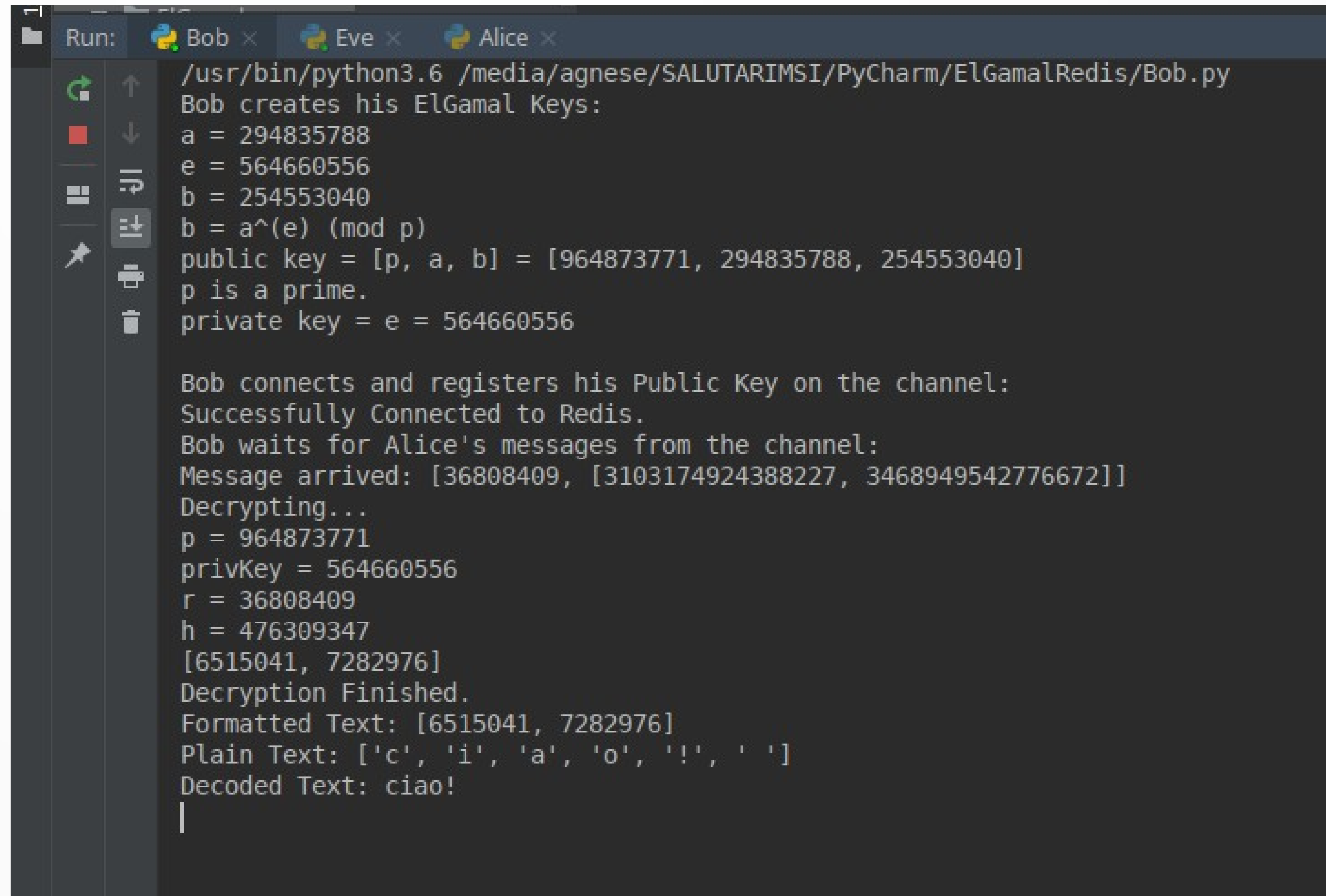
redis-server

File Modifica Visualizza Segnalibri Impostazioni Aiuto

```
agnese@agnese-PS63-Modern-8RC:~$ redis-server
32515:C 24 Apr 19:40:31.950 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
32515:C 24 Apr 19:40:31.950 # Redis version=4.0.9, bits=64, commit=00000000, modified=0, pid=32515, just started
32515:C 24 Apr 19:40:31.950 # Warning: no config file specified, using the default config. In order to specify a
config file use redis-server /path/to/redis.conf
32515:M 24 Apr 19:40:31.953 * Increased maximum number of open files to 10032 (it was originally set to 1024).

                                  Redis 4.0.9 (00000000/0) 64 bit

                                  Running in standalone mode
                                  Port: 6379
                                  PID: 32515

                                        http://redis.io

32515:M 24 Apr 19:40:31.955 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/co
re/somaxconn is set to the lower value of 128.
32515:M 24 Apr 19:40:31.955 # Server initialized
32515:M 24 Apr 19:40:31.955 # WARNING overcommit_memory is set to 0! Background save may fail under low memory co
ndition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command
'sysctl vm.overcommit_memory=1' for this to take effect.
32515:M 24 Apr 19:40:31.955 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This
will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kern
el/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting afte
r a reboot. Redis must be restarted after THP is disabled.
32515:M 24 Apr 19:40:31.955 * DB loaded from disk: 0.000 seconds
32515:M 24 Apr 19:40:31.956 * Ready to accept connections
```

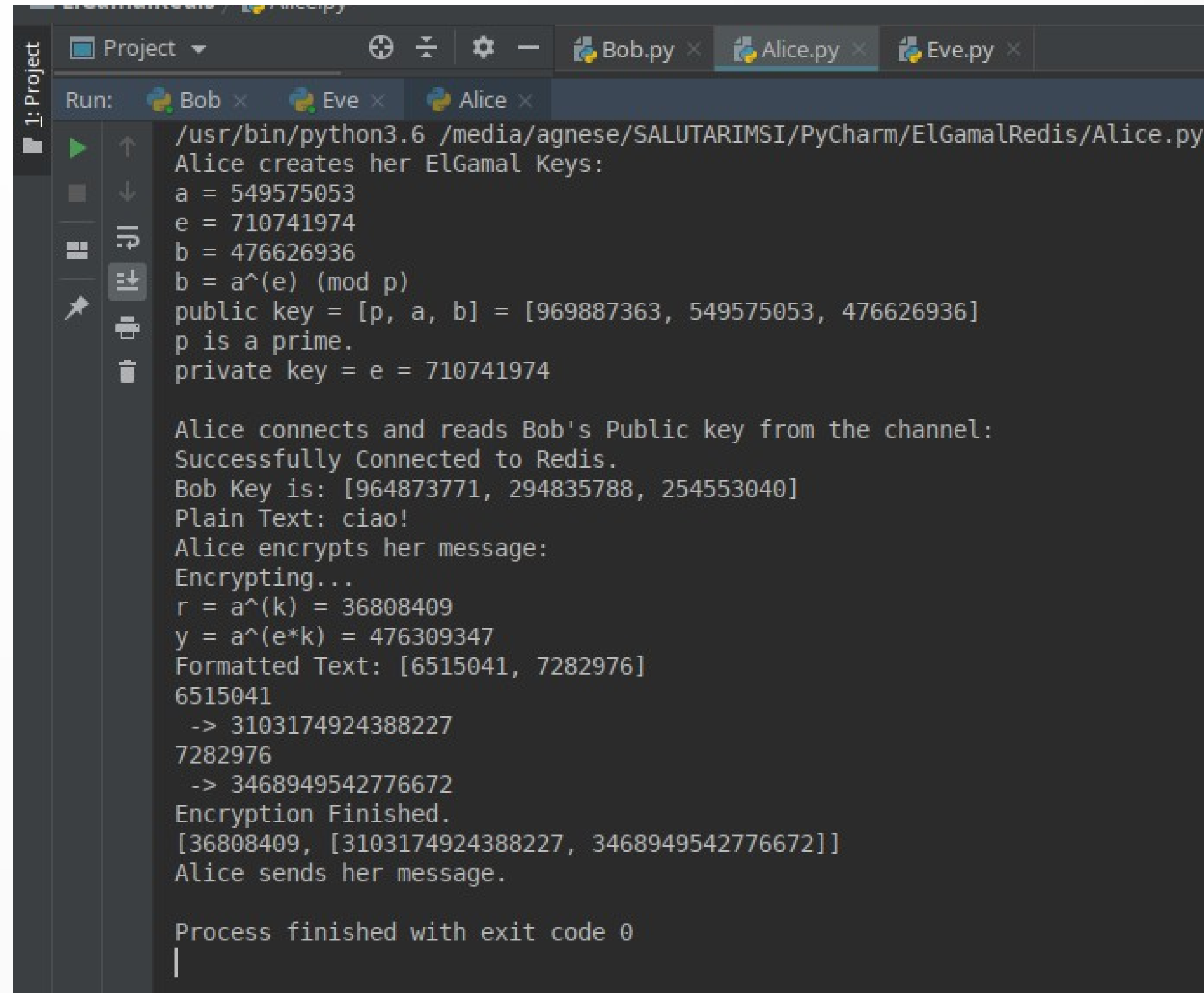
ElGamalRedis – Let's run it

Bob.py



ElGamalRedis – Let's run it

Alice.py



ElGamalRedis – Let's run it

Eve.py

```
Bob × Eve × Alice ×
Products: [-75050/73, 0, -33039/365, -3841046/365, 0, 2796326/3
Final Result = x = 30
TEST: x = 100, a = 16720  ####################################
16720^(x) = 5263484 (mod 15485863).
p is prime.
x = ?
m: [[    4     0     1     0     1     0     0     1     1]
 [    1     1     1     2     0     0     1     2   686]
 [    1     2     3     0     0     0     0     2   895]
 [    0     1     3     1     0     1     2     0   939]
 [    4     2     0     2     0     0     0     2  2082]
 [    1     1     0     0     2     0     1     0  2337]
 [    7     0     1     1     0     0     1     1  4858]
 [    0     0     0     1     1     4     0     0 10655]]
base: [2, 3, 5, 7, 11, 13, 17, 19]
Base of primes: [2, 3, 5, 7, 11, 13, 17, 19]
Congruence Matrix: M
[[    4     0     1     0     1     0     0     1     1]
 [    1     1     1     2     0     0     1     2   686]
 [    1     2     3     0     0     0     0     2   895]
 [    0     1     3     1     0     1     2     0   939]
 [    4     2     0     2     0     0     0     2  2082]
 [    1     1     0     0     2     0     1     0  2337]
 [    7     0     1     1     0     0     1     1  4858]
 [    0     0     0     1     1     4     0     0 10655]]
M in Reduced Row Echelon Form: RM
[[1 0 0 0 0 0 0 0 5213/8]
 [0 1 0 0 0 0 0 0 52647/32]
 [0 0 1 0 0 0 0 0 -993369/256]
 [0 0 0 1 0 0 0 0 -3176683/512]
 [0 0 0 0 1 0 0 0 -1547337/512]
 [0 0 0 0 0 1 0 0 2544845/512]
 [0 0 0 0 0 0 1 0 1557617/256]
 [0 0 0 0 0 0 0 1 2200059/512]]
Pivots: (0, 1, 2, 3, 4, 5, 6, 7)
Logarithms of Base elements: [5213/8, 52647/32, -993369/256, -3
16720^(x) = 5263484 (mod 15485863).
p is prime.
x = ?
powerA = a^(1) = 16720
b * powerA (mod p) = 14778914
Candidate: False
powerA = a^(2) = 812866
b * powerA (mod p) = 11012052
Candidate: False
```

```
Bob × Eve × Alice ×
x = 100
Match Case
Candidate: False
powerA = a^(584) = 11188169
b * powerA (mod p) = 6884450
Candidate: False
powerA = a^(585) = 12446503
b * powerA (mod p) = 1584321
Candidate: False
powerA = a^(586) = 6503166
b * powerA (mod p) = 9021390
Candidate: OrderedDict([(2, 1), (3, 1), (5, 1), (7, 2), (11, 0), (13, 0), (17, 1), (19, 2)])
Found: OrderedDict([(2, 1), (3, 1), (5, 1), (7, 2), (11, 0), (13, 0), (17, 1), (19, 2)]); l = 586
Exponents: [1, 1, 1, 2, 0, 0, 1, 2]
Products: [5213/8, 52647/32, -993369/256, -3176683/256, 0, 0, 1557617/256, 2200059/256]
Final Result = x = 100
Eve connects and reads Bob Public Key from the channel:
Successfully Connected to Redis.
Bob Key is: [964873771, 294835788, 254553040]
Eve sniffs the channel, waiting for messages addressed to Bob...
Message sniffed: [36808409, [3103174924388227, 3468949542776672]]
Eve tries to calculate Bob's private key via Index Calculus...
294835788^(x) = 254553040 (mod 964873771).
p is prime.
x = ?
m: [[   0    1    0    2    0    1    0    1    0    0    0    0    0    0
     0    0    0    0    0    0    0    1    0    1    0  149]
 [   0    0    1    2    0    1    0    1    1    0    0    0    0    0
     0    0    0    0    0    0    0    1    0    0    0  566]
 [   1    0    2    0    0    0    0    1    2    0    0    1    1    0
     0    0    0    0    0    0    0    0    0    0    0  911]
 [   4    0    1    2    0    1    1    1    0    0    0    0    0    1
     0    0    0    0    0    0    0    0    0    0    0  922]
 [   1    0    0    0    0    1    0    0    0    0    0    0    0    0
     0    0    1    0    0    1    0    0    0    1    0 1287]
 [   5    0    2    0    0    1    1    0    0    1    0    0    0    0
     0    1    0    0    0    0    0    0    0    0    0 1298]
 [   4    0    0    0    1    0    0    1    0    0    0    0    1    1
     0    0    0    0    0    0    0    0    1    0    0 2079]
 [   2    1    0    0    1    0    0    0    0    0    0    1    0    0
     0    0    0    0    0    0    0    0    1    0    0 2182]
 [   7    0    0    0    0    2    0    1    0    0    1    0    0    0
     0    0    0    1    0    0    0    0    0    0    0 2278]
 [   0    0    0    1    0    0    1    1    0    0    0    0    0    0
     0    1    1    0    0    0    0    0    1    0    0 2409]
 [   2    1    0    0    0    0    0    0    0    2    1    1    0    0
     0    0    0    0    0    0    1    0    0    0    0 2704]
 [   3    1    0    3    1    0    0    0    0    0    0    0    0    0
     0    0    1    0    0    0    0    0    0    0    0 3246]
 [   6    1    0    0    0    0    0    0    1    0    1    1    0    0
```

For this code and more information:

https://github.com/agnsal/ElGamalRedis

For another Discrete Logarithm Attack:

https://github.com/agnsal/SalutariDiscreteLogProblemSolver

Thank you for your attention