DAT 510: Assignment 2
Implement Secure Communications
Author: Sondre Tennø

Abstract
I solved this assignment by implementing three components for secure communication. For Key Exchange, I chose the Diffie Hellman key exchange, for CSPRNG I chose Blum Blum Shub generator and for Symmetric Cipher I chose to use DES. I created a client-server model to exchange messages using these three security components and successfully managed to get the right message across.

Introduction
In this assignment the goal was to implement a secure communication scenario, which utilized crypto-graphic primitives in a similar but rather simplified way as the real world-applications. The program is composed of three main components, Key Exchange, CSPRNG and Symmetric Cipher. We were to make a program based on this illustration;

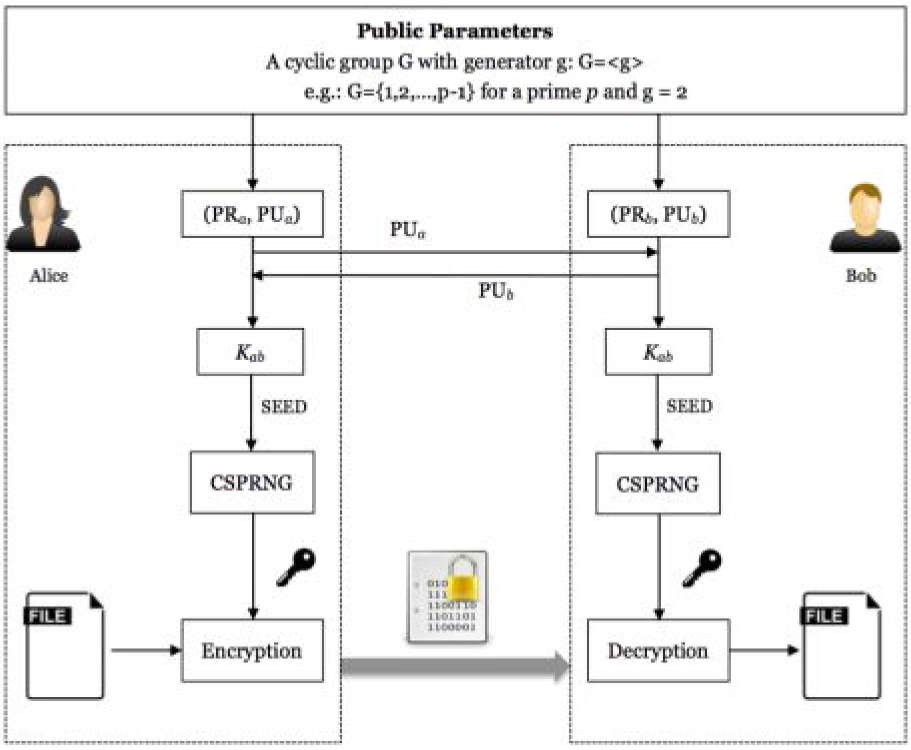
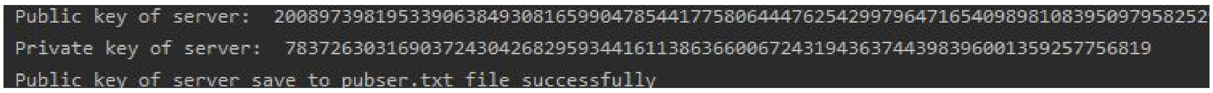


Figure 1: Secure Communications for Alice and Bob

Design and Implementation
Part 1

To tackle this problem, I have made a very simple client-server model to represent Alice and Bob. In Python I have programmed a client and a server that will send messages to each other based on these secure communication concepts.
When I start the server, the server will create a private key and a public key for itself, following Diffie-Hellman's key exchange scheme. Key pair(PRa, PUa).



```
Public key of server:  2008973981953390638493081659904785441775806444762542997964716540989810839509795825
Private key of server:  78372630316903724304268295934416113863660067243194363744398396001359257756819
Public key of server save to pubser.txt file successfully
```

Snippet of the logs when I start the server.

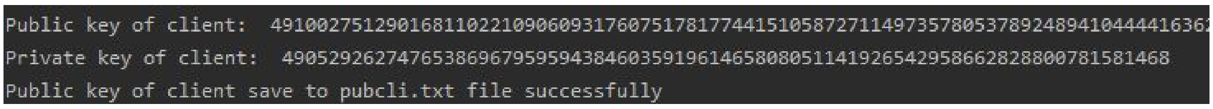
Public key of server (PUa):
2008973981953390638493081659904785441775806444762542997964716540989
8108395097958252925268978904918601129294169745249550668545459926249
4987064147298225990524018201495335970749170946753941999704676917439
5029183548638897975528477482859080381817813284413812124841364577266
4210929721050234286547887587115133357827837740948648823526231489813
4085735987438742675390761367405357138064163324809072891785185486935
2289169570918197137810663575644489510823302078366182894458535067005
8707542198398919042306709635805262589009289401517181462440913452164
8770893244614251845110397493054032884318406564215813543828145862304
79893899504384

Private key of server (PRa):
7837263031690372430426829593441611386366006724319436374439839600135
9257756819

I then save the public key to a text file called pubser.txt for the client to receive the public key later.

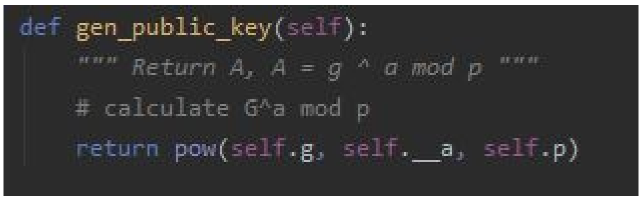
The server is now running and is open for connection from a client.

I then start the client, which will also create a private key and a public key for itself, following Diffie-Hellman's key exchange scheme. Key pair(PRb, PUb).



```
Public key of client:  491002751290168110221090609317607517817744151058727114973578053789248941044441636
Private key of client:  49052926274765386967959594384603591961465808051141926542958662828800781581468
Public key of client save to pubcli.txt file successfully
```

Snippet of logs when I start the client.

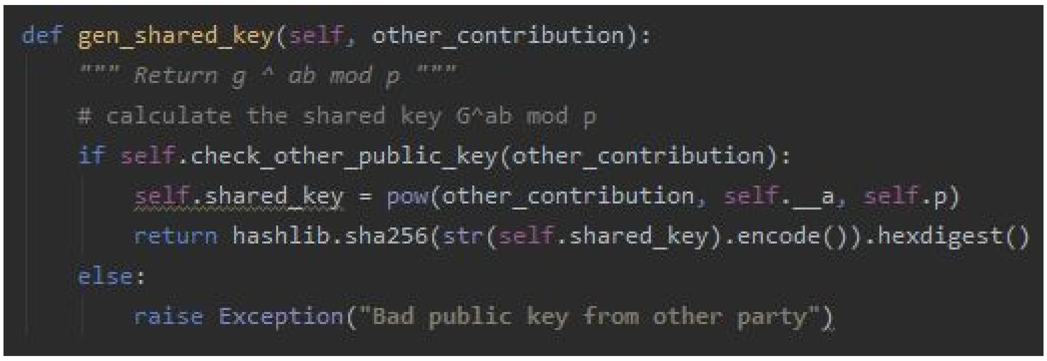


```
def gen_public_key(self):
    """ Return A, A = g ^ a mod p """
    # calculate G^a mod p
    return pow(self.g, self.__a, self.p)
```

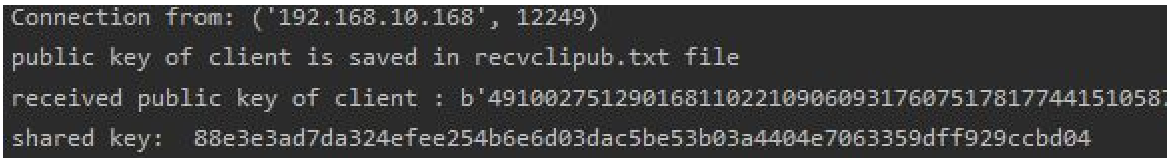
Public key of client (PUb):
4910027512901681102210906093176075178177441510587271149735780537892
4894104444163626262298478271588237594448252971041448943873100930712
3946048555378984549683186154638988435876275289252192550491334130729
9788745347136495015762055198421130051700590756429417533913004717859
1188168543462139562745977869205548999157411945304384669012262761316
5923714256399088421108115830652325138411024888395091842252099154553
6021488003682204888356022831180828581876495965092450883698749137978
0565747890976935629346929108470059977683314793631059732927661029838
4016083976920972921693442799777854457468702780523865181777660163768
4112140088003

Private key of client (PRb):
4905292627476538696795959438460359196146580805114192654295866282880
0781581468

The server and the client will now connect and receive each other's public key to create a shared key amongst them.

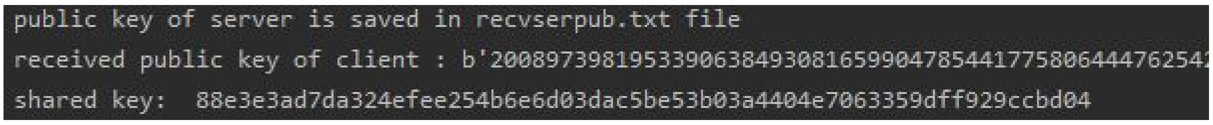


```python
def gen_shared_key(self, other_contribution):
    """ Return g ^ ab mod p """
    # calculate the shared key G^ab mod p
    if self.check_other_public_key(other_contribution):
        self.shared_key = pow(other_contribution, self.__a, self.p)
        return hashlib.sha256(str(self.shared_key).encode()).hexdigest()
    else:
        raise Exception("Bad public key from other party")
```



```
Connection from: ('192.168.10.168', 12249)
public key of client is saved in recvclipub.txt file
received public key of client : b'4910027512901681102210906093176075178177441510587
shared key:  88e3e3ad7da324efee254b6e6d03dac5be53b03a4404e7063359dff929ccbd04
```

Snippet from server.



```
public key of server is saved in recvserpub.txt file
received public key of client : b'200897398195339063849308165990478544177580644476254
shared key:  88e3e3ad7da324efee254b6e6d03dac5be53b03a4404e7063359dff929ccbd04
```

Snippet from client.

Shared key (Kab):
88e3e3ad7da324efee254b6e6d03dac5be53b03a4404e7063359dff929ccbd04

From the logs, we can see that the public keys generated were the same keys received on the other side. We can also confirm the shared keys generated are identical to each other. We can now start sending messages between the server and the client. But first we need to implement CSPRNG to strengthen the shared key. Here we will use Blum Blum Shub generator as a pseudo-random number generator which takes Kab as the seed, to generate a secret key K for subsequent encryption/decryption.

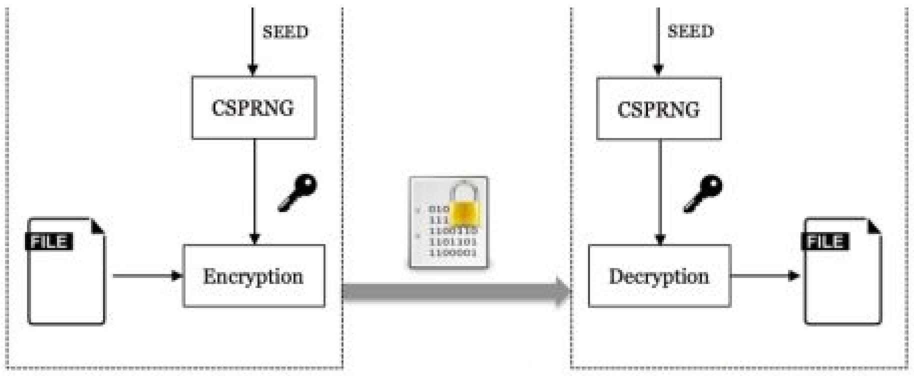


Figure 1: Secure Communications for Alice and Bob

This is how far we have gotten in the secure communication so far, we have CSPRNG and the Symmetric Cipher part left.

Blum Blum Shub is used as a pseudo-random number generator. Blum Blumb Shub is not truly random because it does depend on a seed for its randomisation. The formula for Blum Blum Shub goes like this;

$$x_{n+1} = x_n^2 \bmod M,$$

where M = pq, which is the product of two large primes p and q. Some output is derived from xn+1 the output is commonly either the bit parity of xn+1 or one or more of the least significant bits of xn+1. The seed x0 should be an integer that is co-prime to M and not 1 or 0.

The two primes p and q, should both be congruent to 3(mod4), and should be safe primes with a small greatest common divisor.((p-3)/2,(q-3)/2). These last 3 paragraphs describing Blum Blum Shub was taken from wikipedia. Blum Blum Slub generator is generally slow, but is the strongest proven pseudo-random number generator.

After we have strengthened our shared key using Blum Blum Shub, we can start sending messages which will be encrypted/decrypted by DES and by using the key generated by Blumb Blumb Shub. Since my implementation of SDES in the previous assignment wasn't 100%, I have used a finished DES library instead, which was stated to be allowed in this assignment.

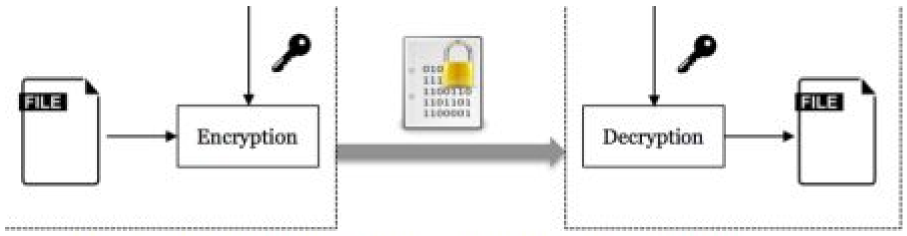
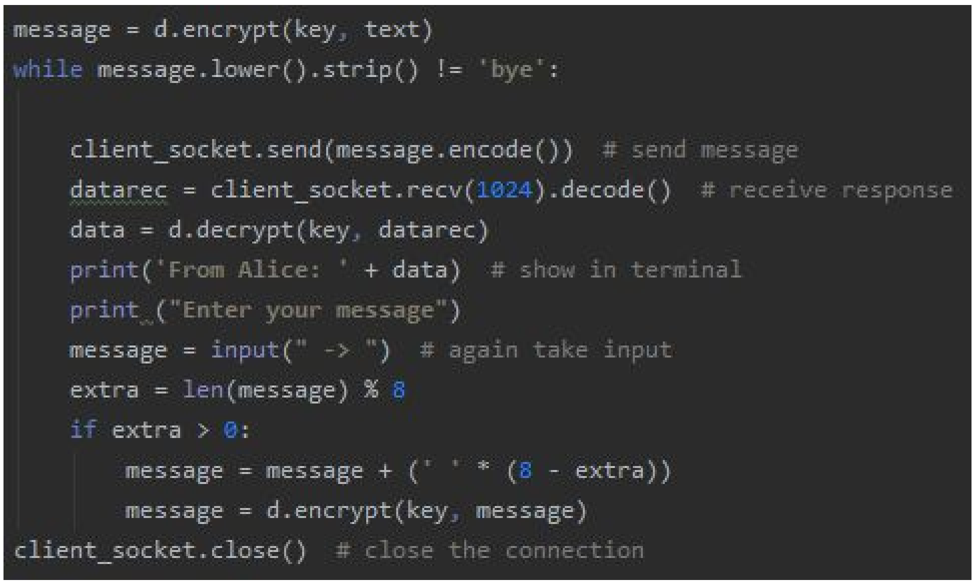


Figure 1: Secure Communications for Alice and Bob

Now we only have the Symmetric Cipher part left.

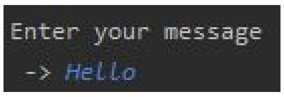


```
message = d.encrypt(key, text)
while message.lower().strip() != 'bye':

    client_socket.send(message.encode())  # send message
    datarec = client_socket.recv(1024).decode()  # receive response
    data = d.decrypt(key, datarec)
    print('From Alice: ' + data)  # show in terminal
    print ("Enter your message")
    message = input(" -> ")  # again take input
    extra = len(message) % 8
    if extra > 0:
        message = message + (' ' * (8 - extra))
        message = d.encrypt(key, message)
client_socket.close()  # close the connection
```

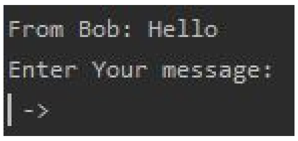
When we do send a message, the message will be encrypted with DES, sent to the other side, and decrypted to receive a plain text of the message.

So when I send the message “Hello” from the client to the server;



```
Enter your message
 -> Hello
```

Snippet from the client.



```
From Bob: Hello
Enter Your message:
 ->
```

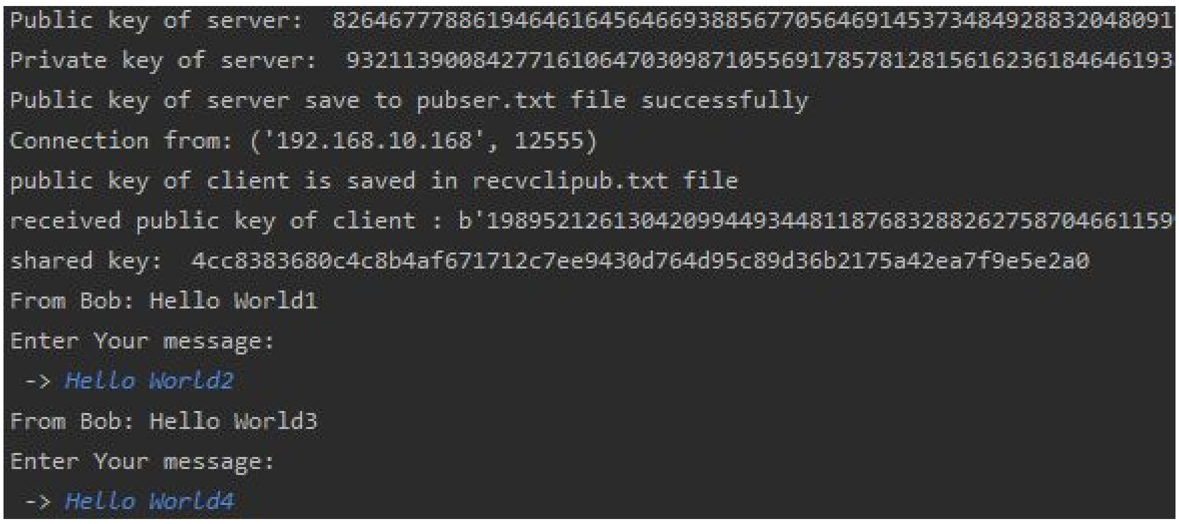
Snippet on the server side.

We can see that we receive the correct plaintext message on the other side.

Test Results
To test the results, I will run the program several times with different messages and showing all the output logs. We will see if the messages comes out correct, the program is stable, and see if the keys match up.

Test 1:

Logs from server:



```
Public key of server:  826467778861946461645646693885677056469145373484928832048091
Private key of server:  93211390084277161064703098710556917857812815616236184646193
Public key of server save to pubser.txt file successfully
Connection from: ('192.168.10.168', 12555)
public key of client is saved in recvclipub.txt file
received public key of client : b'1989521261304209944934481187683288262758704661159
shared key:  4cc8383680c4c8b4af671712c7ee9430d764d95c89d36b2175a42ea7f9e5e2a0
From Bob: Hello World1
Enter Your message:
 -> Hello World2
From Bob: Hello World3
Enter Your message:
 -> Hello World4
```

Private key:
93211390084277161064703098710556917857812815616236184646193891143521354159150

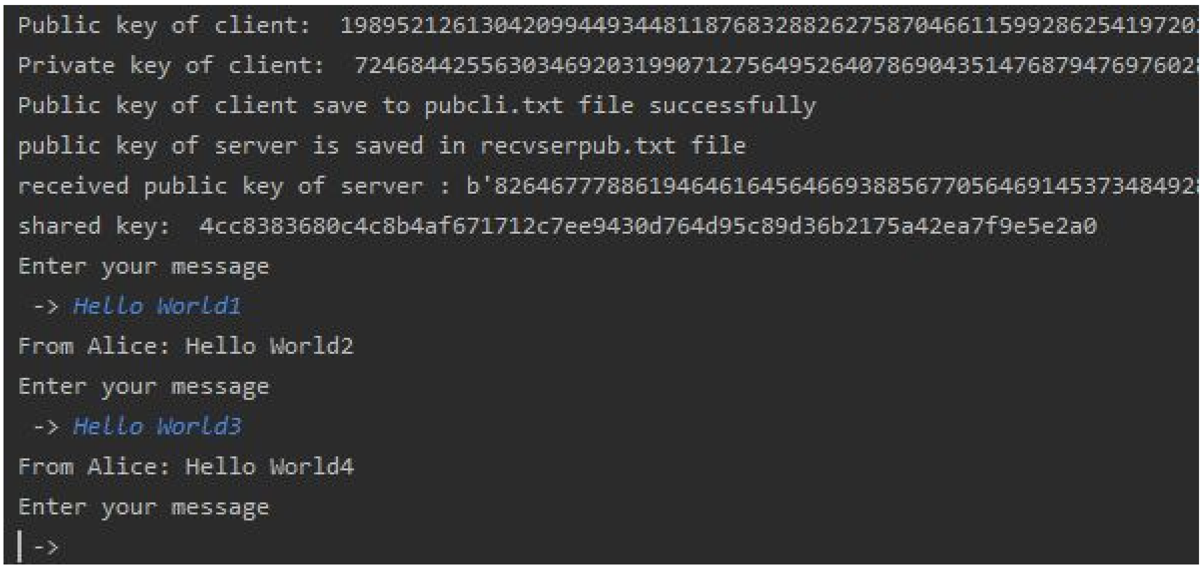
Public key:
826467778861946461645646693885677056469145373484928832048091727547467905390780700223811260223459326189422527501472200310359617266118243133368602148014575587142458987625834679044815728946133894383611016947205126666851576309998194222657841014387921795226371518203185079459515601350447659726615303858785188218352972351642574983756844143246160845187722444270791646813090956060563819740246162747543026848370827580120739108251298210323439202428983713791686797218313874122526709227541366713979075811537462081059994429906138924692442626894288546892597772214595150738218491444730863020496287117713420087647774989 0013628064012

Received public key:
1989521261304209944934481187683288262758704661159928625419720210803

3332543985989796543206325846913985569044014673038295626399113531730788529708714818978226792237308336979890630307605191332892563723553605262143472268495937003485472230996799102160943556709690414408186370816700044140196098080381916018952491155724719528746284065759544423365862765401709577073055671202657902647417381359040686498864952874077532173707476109517736572867531072982039990346683470396958479524984380709544207021737711923312594601220359150019013896962742028223244034275168031963733447721594812975335376186401861192104411840047386108956057099425

Shared key:
4cc8383680c4c8b4af671712c7ee9430d764d95c89d36b2175a42ea7f9e5e2a0

Logs from client:



```
Public key of client:  1989521261304209944934481187683288262758704661159928625419720
Private key of client:  72468442556303469203199071275649526407869043514768794769760
Public key of client save to pubcli.txt file successfully
public key of server is saved in recvserpub.txt file
received public key of server : b'82646777886194646164564669388567705646914537348492
shared key:  4cc8383680c4c8b4af671712c7ee9430d764d95c89d36b2175a42ea7f9e5e2a0
Enter your message
 -> Hello World1
From Alice: Hello World2
Enter your message
 -> Hello World3
From Alice: Hello World4
Enter your message
| ->
```

Private key:
72468442556303469203199071275649526407869043514768794769760285670107353114394

Public key:
19895212613042099449344811876832882627587046611599286254197202108033332543985989796543206325846913985569044014673038295626399113531730788529708714818978226792237308336979890630307605191332892563723553605262143472268495937003485472230996799102160943556709690414408186370816700044140196098080381916018952491155724719528746284065759544423365862765401709577073055671202657902647417381359040686498864952874077532173707476109517736572867531072982039990346683470396958479524984380709544207021737711923312594601220359150019013896962742028223244

034275168031963733447721594812975335376186401861192104411840047386108956057099425

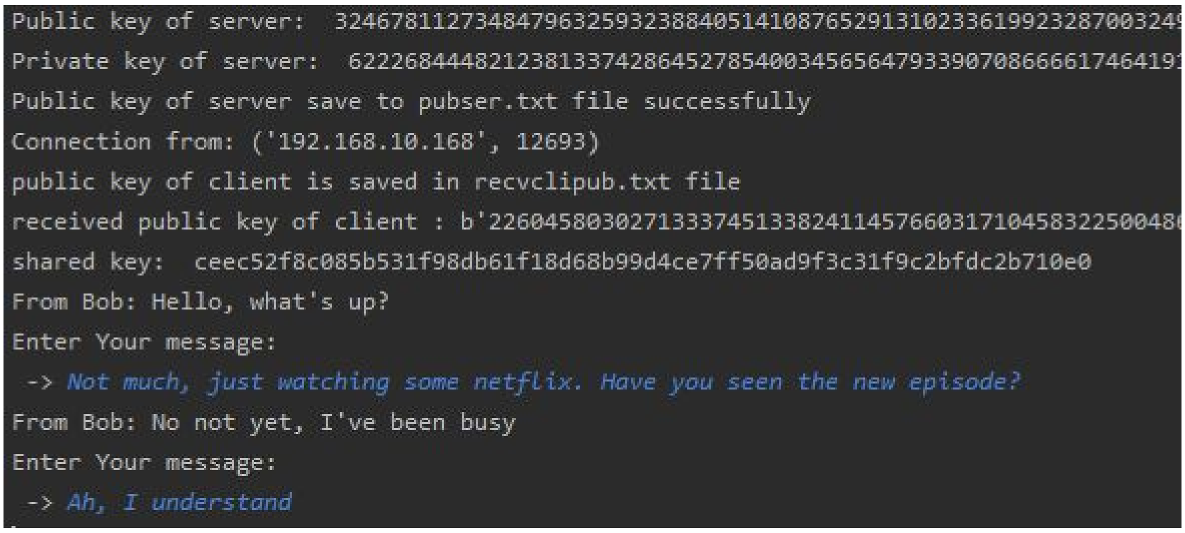
Received public key:
8264677788619464616456466938856770564691453734849288320480917275474679053907807002238112602234593261894225275014722003103596172661182431333686021480145755871424589876258346790448157289461338943836110169472051266668515763099981942226578410143879217952263715182031850794595156013504476597266153038587851882183529723516425749837568441432461608451877224442707916468130909560605638197402461627475430268483708275801207391082512982103234392024289837137916867972183138741225267092275413667139790758115374620810599944299061389246924426268942885468925977722145951507382184914447308630204962871177134200876477749890013628064012

Shared key:
4cc8383680c4c8b4af671712c7ee9430d764d95c89d36b2175a42ea7f9e5e2a0

Results of Test1:

We can see that all the messages received on both side are of the correct plain text.
We can see that the shared key is the same for both client and server.
We can see that the public keys and the received public keys do match.

Test2:

Logs from server:


```
Public key of server:  324678112734847963259323884051410876529131023361992328700324
Private key of server:  62226844482123813374286452785400345656479339070866661746419
Public key of server save to pubser.txt file successfully
Connection from: ('192.168.10.168', 12693)
public key of client is saved in recvclipub.txt file
received public key of client : b'2260458030271333745133824114576603171045832250048
shared key:  ceec52f8c085b531f98db61f18d68b99d4ce7ff50ad9f3c31f9c2bfdc2b710e0
From Bob: Hello, what's up?
Enter Your message:
 -> Not much, just watching some netflix. Have you seen the new episode?
From Bob: No not yet, I've been busy
Enter Your message:
 -> Ah, I understand
```

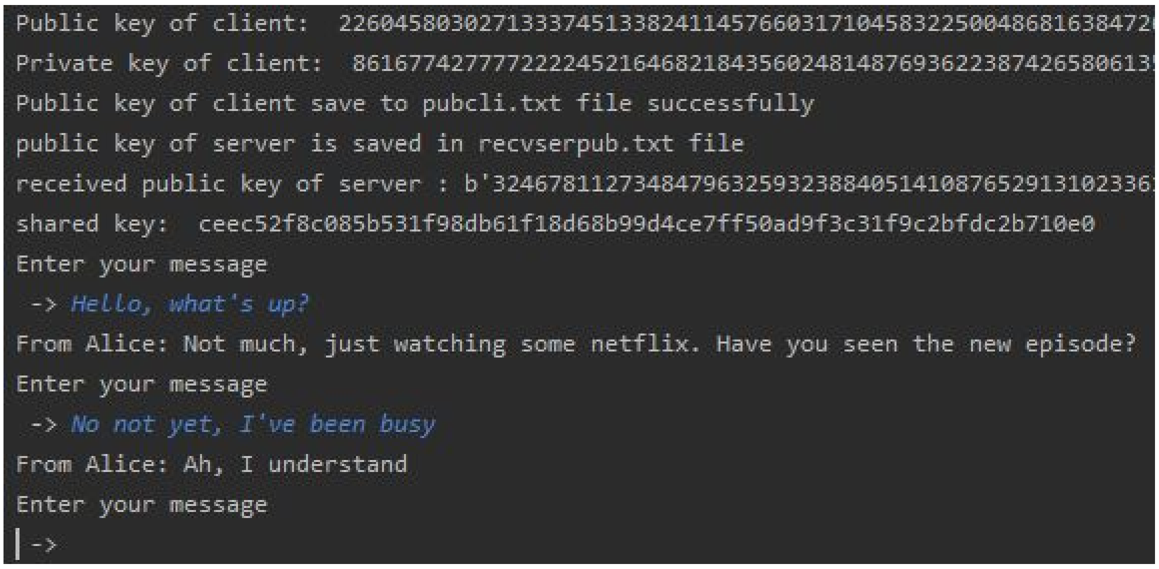
Private key:

6222684448212381337428645278540034565647933907086666174641912155648
9688730091
Public key:
3246781127348479632593238840514108765291310233619923287003249983516
9548343301475565245004252581833491808803143738864039466014362877955
1368401796716719637011060301160420846279921230844318257652403493859
6666571299664263455075223322402383859647920910740574594179332974258
3334684607026026984343986164420613106419847073121494754191374235277
5436292938984997905824888249814254413511951313865805036452470348928
7932875814814740036886271004288737021045923579732723051549164909946
7264805532549934348387216566430022152919508737564190640182863127891
1859784120966804094833084285993094987504606334752316182386098272506
215002658955

Received public
key:2260458030271333745133824114576603171045832250048681638472618442
0053030939775347479684321429860216838993676553890473150642550954609
4656016981538839390248012767507962198667541292030079882752340894164
0756693532220314374602218016198015047735979996862080877953682247082
4995711155133503160500665516164889524237873504774560755370138726808
2884313487177470856798182736836554213764559454772829529568467036178
0625344029163989631494589486700779572043915071261174057479196866321
1700038613736735129355750481073043271139271481519214108144962906481
3359761963916722268422545569063404962358359114761888725220052778683
24318830743156539

Shared key:
ceec52f8c085b531f98db61f18d68b99d4ce7ff50ad9f3c31f9c2bfdc2b710e0

Logs from client:



```
Public key of client:  22604580302713337451338241145766031710458322500486816384726
Private key of client:  8616774277772222452164682184356024814876936223874265806135
Public key of client save to pubcli.txt file successfully
public key of server is saved in recvserpub.txt file
received public key of server : b'324678112734847963259323884051410876529131023361
shared key:  ceec52f8c085b531f98db61f18d68b99d4ce7ff50ad9f3c31f9c2bfdc2b710e0
Enter your message
 -> Hello, what's up?
From Alice: Not much, just watching some netflix. Have you seen the new episode?
Enter your message
 -> No not yet, I've been busy
From Alice: Ah, I understand
Enter your message
|->
```

Private key:
8616774277772222452164682184356024814876936223874265806135103436911
1590628274

Public key:
2260458030271333745133824114576603171045832250048681638472618442005
3030939775347479684321429860216838993676553890473150642550954609465
6016981538839390248012767507962198667541292030079882752340894164075
6693532220314374602218016198015047735979996862080877953682247082499
5711155133503160500665516164889524237873504774560755370138726808288
4313487177470856798182736836554213764559454772829529568467036178062
5344029163989631494589486700779572043915071261174057479196866321170
0038613736735129355750481073043271139271481519214108144962906481335
9761963916722268422545569063404962358359114761888725220052778683243
18830743156539

Received public key:
3246781127348479632593238840514108765291310233619923287003249983516
9548343301475565245004252581833491808803143738864039466014362877955
1368401796716719637011060301160420846279921230844318257652403493859
6666571299664263455075223322402383859647920910740574594179332974258
3334684607026026984343986164420613106419847073121494754191374235277
5436292938984997905824888249814254413511951313865805036452470348928
7932875814814740036886271004288737021045923579732723051549164909946
7264805532549934348387216566430022152919508737564190640182863127891
1859784120966804094833084285993094987504606334752316182386098272506
215002658955

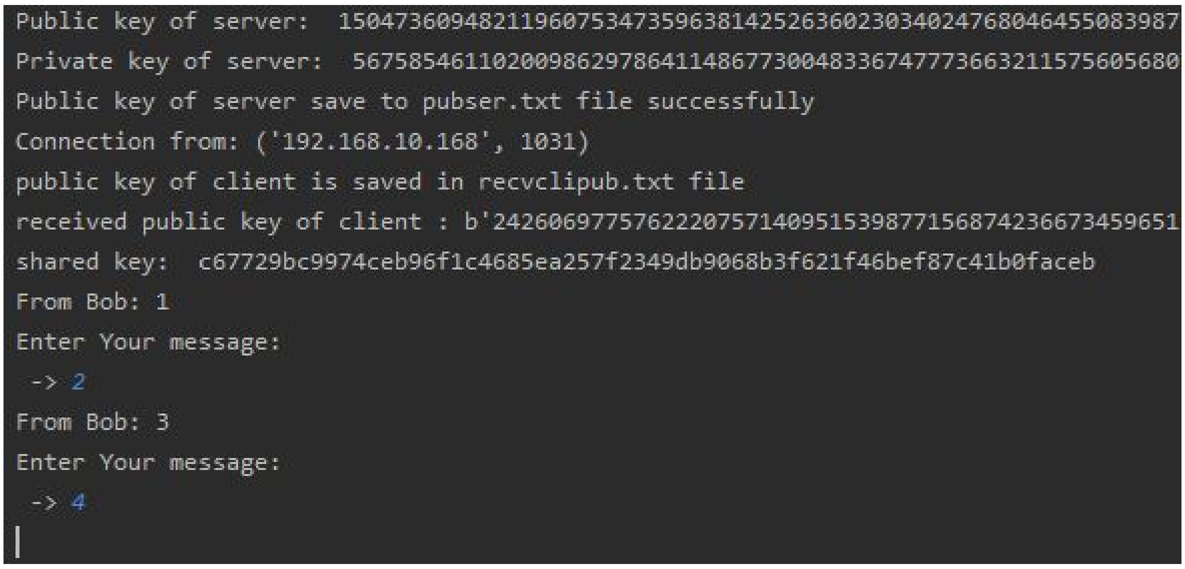
Shared key:
ceec52f8c085b531f98db61f18d68b99d4ce7ff50ad9f3c31f9c2bfdc2b710e0

Results of Test2:

We can see that all the messages received on both sides are of the correct plain text.
We can see that the shared key is the same for both client and server.
We can see that the public keys and the received public keys do match.

Test3:

Logs from server:



```
Public key of server:  150473609482119607534735963814252636023034024768046455083987
Private key of server:  567585461102009862978641148677300483367477736632115756056807
Public key of server save to pubser.txt file successfully
Connection from: ('192.168.10.168', 1031)
public key of client is saved in recvclipub.txt file
received public key of client : b'2426069775762220757140951539877156874236673459651
shared key:  c67729bc9974ceb96f1c4685ea257f2349db9068b3f621f46bef87c41b0faceb
From Bob: 1
Enter Your message:
 -> 2
From Bob: 3
Enter Your message:
 -> 4
```

Private key:
567585461102009862978641148677300483367477736632115756056807221936516230003060

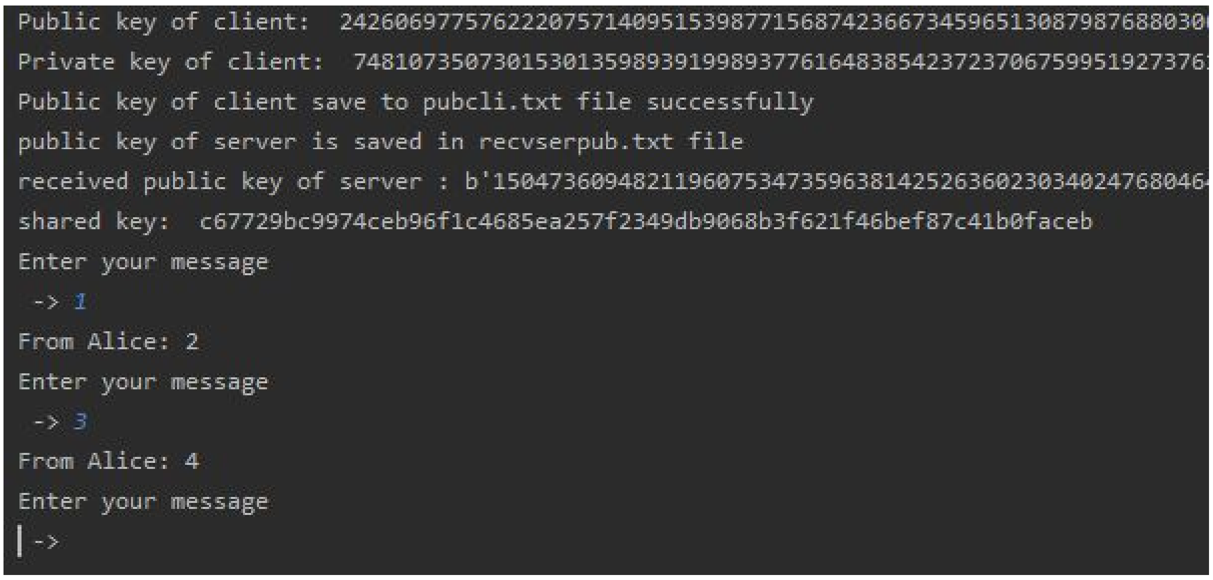
Public key:
1504736094821196075347359638142526360230340247680464550839872122018617993851425344898358194425846973042194924591041285881629286128505697745051646225262235947189240631609380718559706404973974573158875767516114282142029557089847983854574890807271210270574248278290546157773821951222880239747628667826427833567628115254056125452412288383274543017449368570833903429160922361954934590286285546411747944637647378582179035483113435336816231325744874888856622474147501267802647119352579661544424924273614537043811332325901163597802098113562701393880589350405984420847533737032738901772079503107926972091668114480172473732537 8

Received public key:
2426069775762220757140951539877156874236673459651308798768803067837488052206365894711488923989813442637606194344679871404575811384011860657591184313130678343940784778751867500080782213524328121930379444605267366638510748750952734813286365667170170993026074681650529054374801309691821993690548903599192175064957342586823939911200240533684583261079569356959154314020407133631837840887345931537033999043326590480063283882746611428470610584414381332766518844436492369207135792050217176787458626501294638219128320352228348760303528973204242151350622766361674599784416881753847918466548404834785659605254309963809614563726 3

Shared key:

c67729bc9974ceb96f1c4685ea257f2349db9068b3f621f46bef87c41b0faceb

Logs from client:



```
Public key of client:  24260697757622207571409515398771568742366734596513087987688030
Private key of client:  748107350730153013598939199893776164838542372370675995192737
Public key of client save to pubcli.txt file successfully
public key of server is saved in recvserpub.txt file
received public key of server : b'150473609482119607534735963814252636023034024768046
shared key:  c67729bc9974ceb96f1c4685ea257f2349db9068b3f621f46bef87c41b0faceb
Enter your message
 -> 1
From Alice: 2
Enter your message
 -> 3
From Alice: 4
Enter your message
|->
```

Private key:
74810735073015301359893919989377616483854237237067599519273761210083905108592

Public key:
2426069775762220757140951539877156874236673459651308798768803067837488052206365894711488923989813442637606194344679871404575811384011860657591184313130678343940784778751867500080782213524328121930379444605267366638510748750952734813286365667170170993026074681650529054374801309691821993690548903599192175064957342586823939911200240533684583261079569356959154314020407133631837840887345931537033999043326590480063283882746611428470610584414381332766518844436492369207135792050217176787458626501294638219128320352228348760303528973204242151350622766361674599784416881753847918466548404834785659605254309963809614563 7263

Received public key:
150473609482119607534735963814252636023034024768046455083987212201861799385142534489835819442584697304219492459104128588162928612850569774505164622526223594718924063160938071855970640497397457315887576751611428214202955708984798385457489080727121027057424827829054615777382195122288023974762866782642783356762811525405612545241228838327454301744936857083390342916092236195493459028628554641174794463764737858217903548311343533681623132574487488885662247414750126780264711935257966154442492427361453704381133232590116359780209811356270 13

938805893504059844208475337370327389017720795031079269720916681144801724737325378

Shared key:
c67729bc9974ceb96f1c4685ea257f2349db9068b3f621f46bef87c41b0faceb

Results of Test3:

We can see that all the messages received on both sides are of the correct plain text.
We can see that the shared key is the same for both client and server.
We can see that the public keys and the received public keys do match.

Discussion

In the test results we can see that all three tests gives us the correct message on the other side in plain text. All three tests match the shared keys. And all three tests have matching public keys sent and received on both sides.

Conclusion

In this assignment we learned about implementing secure communications. How a complete secure communication system is set up and how it works and all its components. We learned about Key Exchange, CSPRNG and Symmetric Cipher which we went through in the previous assignment. I did use a finished implementation of DES and Diffie Hellman Exchange. In the test results we can see that all three tests give us the correct message on the other side in plain text. All three tests match the shared keys. And all three tests have matching public keys sent and received on both sides.

Works Cited

https://www.asecuritysite.com/encryption/blum
Information I used about the Blum Blum Shub.

https://en.wikipedia.org/wiki/Blum_Blum_Shub
Information I used about the Blum Blum Shub.

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
Information I used concerning AES.

https://github.com/amiralis/pyDH/blob/master/pyDH/pyDH.py
The Diffie Hellman Exchange I used in this assignment.

https://github.com/RobinDavid/pydes/blob/master/pydes.py
The finished DES implementation I used in this assignment.

https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
Information I used concerning Diffie-Hellman key exchange.

Stallings, William. Cryptography and Network Security: Principles and Practice (7th Edition). 2016

https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator
Information I used concerning CSPRNG.