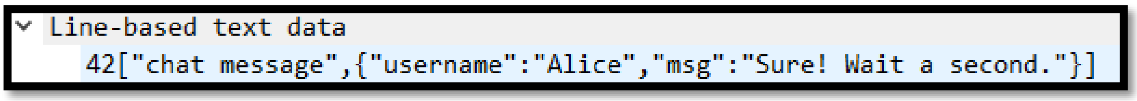
Project 2　　Report　　　　0216328 劉啟亦

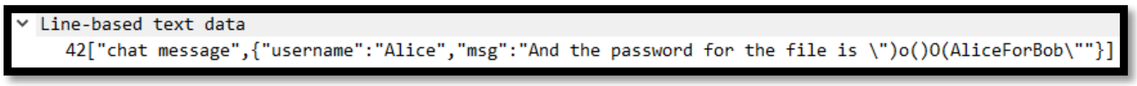
a. How to get the secret document step by step:

1. Use “wireshark” with keyword “websocket” to filter the packets in websocket file, and I find that there are some readable information in the fields, like:



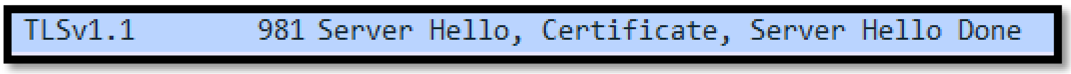
```
Line-based text data
    42["chat message",{"username":"Alice","msg":"Sure! Wait a second."}]
```

These packets not only show the communication messages, but also show the password:



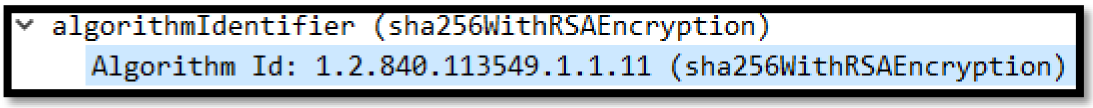
```
Line-based text data
    42["chat message",{"username":"Alice","msg":"And the password for the file is \")o()O(AliceForBob\""}]
```

2. Keep going to see other files, and the word “certificate” in the Info column catch my attentions:



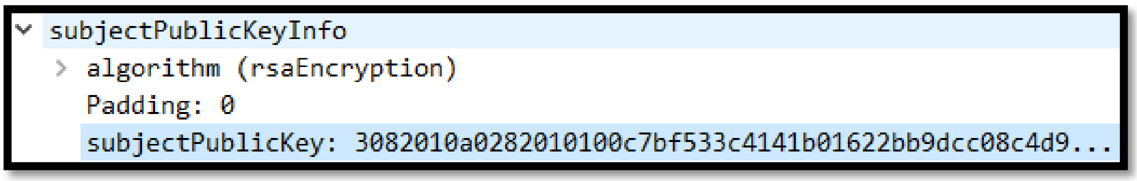
```
TLSv1.1        981 Server Hello, Certificate, Server Hello Done
```

Then I take the deeper look in this socket, and find out that Alice and Bob use the “RSA algorithm” to encrypt the data:



```
algorithmIdentifier (sha256WithRSAEncryption)
    Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
```

According to the knowledge I have learned in the class, I know RSA needs to do the “key exchange”, so this socket must carry someone’s public key somewhere:

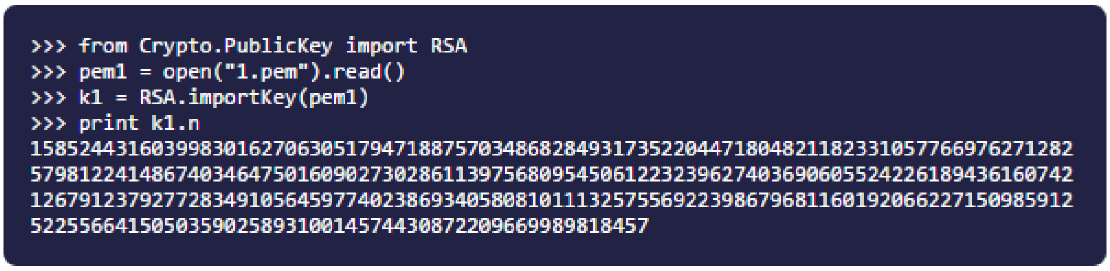


```
subjectPublicKeyInfo
  > algorithm (rsaEncryption)
    Padding: 0
    subjectPublicKey: 3082010a0282010100c7bf533c4141b01622bb9dcc08c4d9...
```

Next, I have to extract this public key to do some RSA cracking, then I click this:

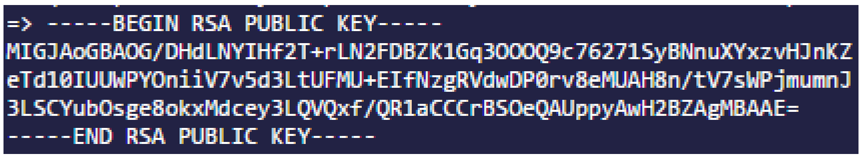
Will create a binary file with the key information. So, now I can have 12 different key binary files in this project.

3. Open the url in the hints, I realize how to get the “value of public key” and “n” which is the number generated by two keys’ production from the above key information files:



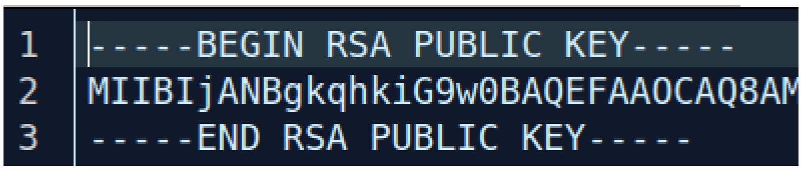
```
>>> from Crypto.PublicKey import RSA
>>> pem1 = open("1.pem").read()
>>> k1 = RSA.importKey(pem1)
>>> print k1.n
15852443160399830162706305179471887570348682849317352204471804821182331057766976271282
57981224148674034647501609027302861139756809545061223239627403690605524226189436160742
12679123792772834910564597740238693405808101113257556922398679681160192066227150985912
522556641505035902589310014574430872209669989818457
```

But, the key information files what I have created above are binary format with attached file name bin, they cannot suit for the code in pem file, so I need to do some process. In the url, it shows the correct format about the pem file:



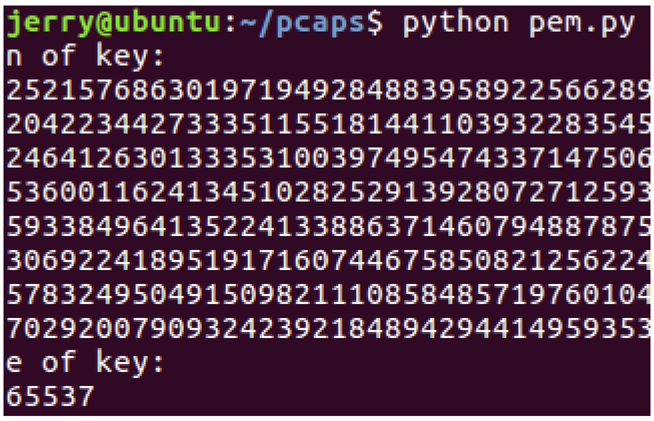
```
=> -----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAOG/DHdLNYIHf2T+rLN2FDBZK1Gq3OOOQ9c76271SyBNnuXYxzvHJnKZ
eTd10IUUWPYOniiV7v5d3LtUFMU+EIfNzgRVdwDP0rv8eMUAH8n/tV7sWPjmumnJ
3LSCYubOsge8okxMdcey3LQVQxf/QR1aCCCrBSOeQAUppyAwH2BZAgMBAAE=
-----END RSA PUBLIC KEY-----
```

With the experience in Project1, I quickly notice that the sequence with the "=" in the tail may be the base64 format, so I convert the code of my key information file from binary to base64, and then add the BEGIN line in the head and END line in the tail:



```
-----BEGIN RSA PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AM
-----END RSA PUBLIC KEY-----
```

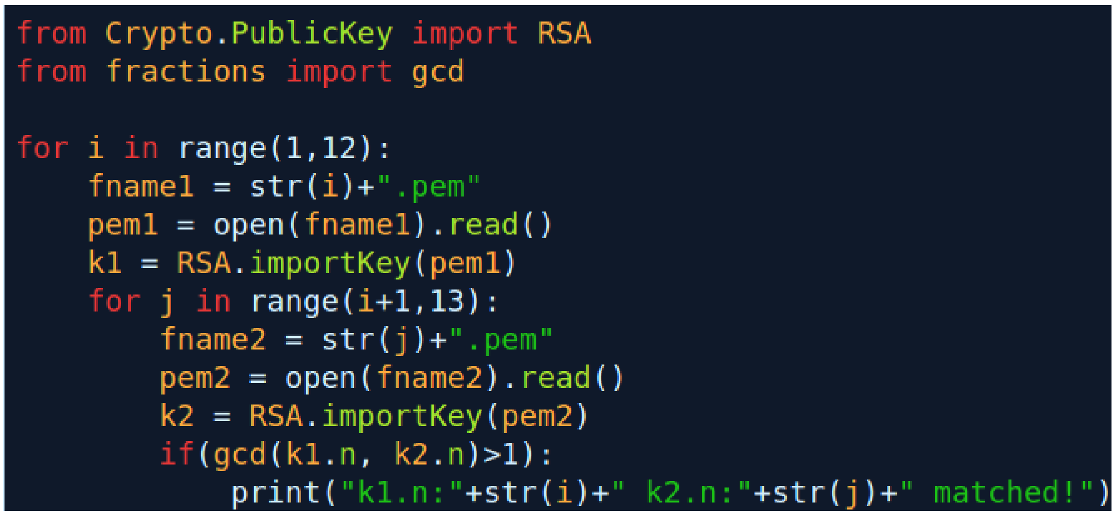
Now here are 12 correct key information pem files, and I can run the sample code to get the public key and n respectively:



```
jerry@ubuntu:~/pcaps$ python pem.py
n of key:
252157686301971949284883958922566289
204223442733351155181441103932283545
246412630133353100397495474337147506
536001162413451028252913928072712593
593384964135224133886371460794887875
306922418951917160744675850821256224
578324950491509821110858485719760104
702920079093242392184894294414959353
e of key:
65537
```

So, with these values, I just need to do the common factor attack, then to achieve the result I want.

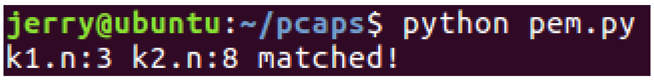
4. To do the common factor attack, I need to know whose value have the common factor first, so write a simple python code to compare all the n:



```python
from Crypto.PublicKey import RSA
from fractions import gcd

for i in range(1,12):
    fname1 = str(i)+".pem"
    pem1 = open(fname1).read()
    k1 = RSA.importKey(pem1)
    for j in range(i+1,13):
        fname2 = str(j)+".pem"
        pem2 = open(fname2).read()
        k2 = RSA.importKey(pem2)
        if(gcd(k1.n, k2.n)>1):
            print("k1.n:"+str(i)+" k2.n:"+str(j)+" matched!")
```

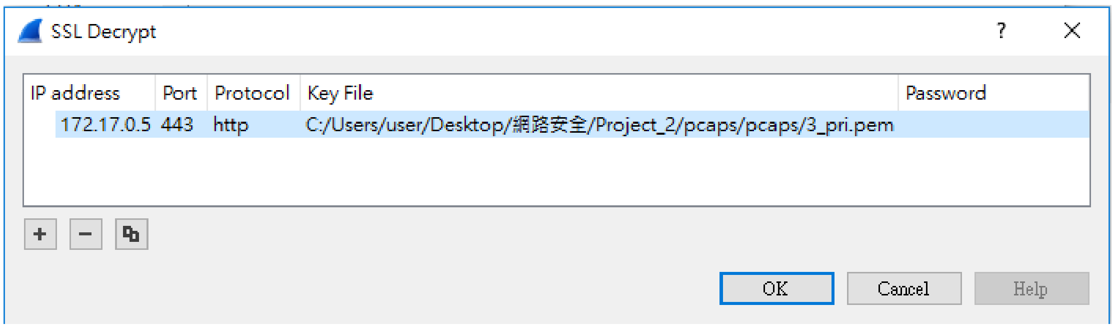
The result shows that:



```
jerry@ubuntu:~/pcaps$ python pem.py
k1.n:3 k2.n:8 matched!
```

Now, I have two targets: one is key no.3, the other is key no.8, and then I google the common factor attack for more details. In“ http://justrocketscience.com/post/rsa-common-prime”this website , it provides the sample python code to launch the attack, and I use it to get the “private key of no.3” and “private key of no.8”.

5. With the private key, I can go back the socket file to decrypt the RSA algorithm. Use wireshark to open the “https3” file and click “preference->protocol->SSL” to set the private key for decrypting:

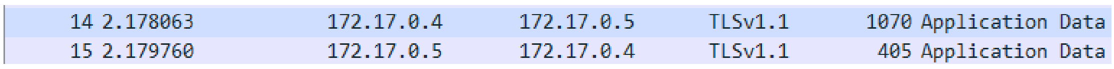


SSL Decrypt

IP address	Port	Protocol	Key File	Password
172.17.0.5	443	http	C:/Users/user/Desktop/網路安全/Project_2/pcaps/pcaps/3_pri.pem	

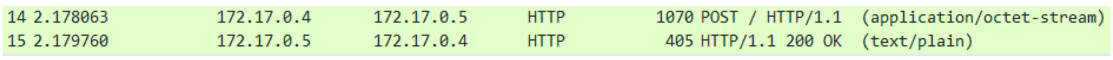
As result, some sockets have change:

<origin>



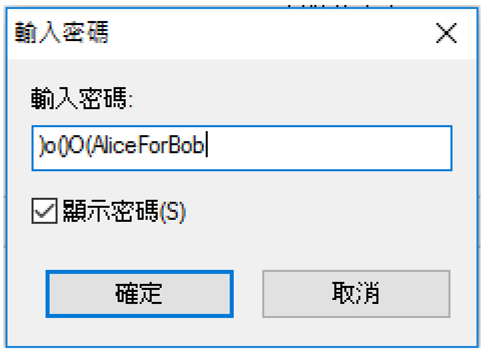
```
14 2.178063      172.17.0.4    172.17.0.5    TLSv1.1    1070 Application Data
15 2.179760      172.17.0.5    172.17.0.4    TLSv1.1     405 Application Data
```

<after private key decryption>



```
14 2.178063      172.17.0.4    172.17.0.5    HTTP    1070 POST / HTTP/1.1  (application/octet-stream)
15 2.179760      172.17.0.5    172.17.0.4    HTTP     405 HTTP/1.1 200 OK  (text/plain)
```

Then click ”export object HTTP” will get a file, and unzip it with password known in step1:



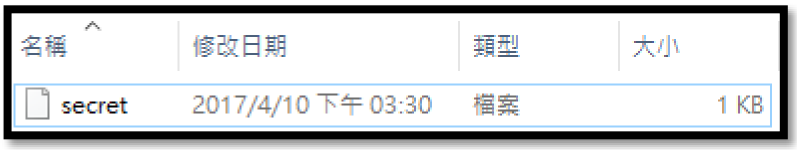
輸入密碼

輸入密碼:

)o()O(AliceForBob

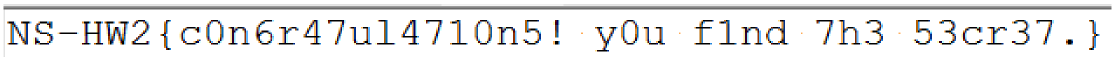
☑ 顯示密碼(S)

Finally, I get the secret file!!



名稱	修改日期	類型	大小
secret	2017/4/10 下午 03:30	檔案	1 KB

And the content of secret file is:



```
NS-HW2{c0n6r47ul4710n5! y0u f1nd 7h3 53cr37.}
```

b. What I have learned:

When doing data transmission on the Internet, I think it should choose the protocol carefully, especially for those sensitive data like password, they should not be easily readable for hacker, so try to make sure the channel be encrypted by some protocol.

Sooner or later, there is always weakness in security mechanism, so I think changing the encryption algorithm periodically or combine multiple encryption algorithm may be helpful.