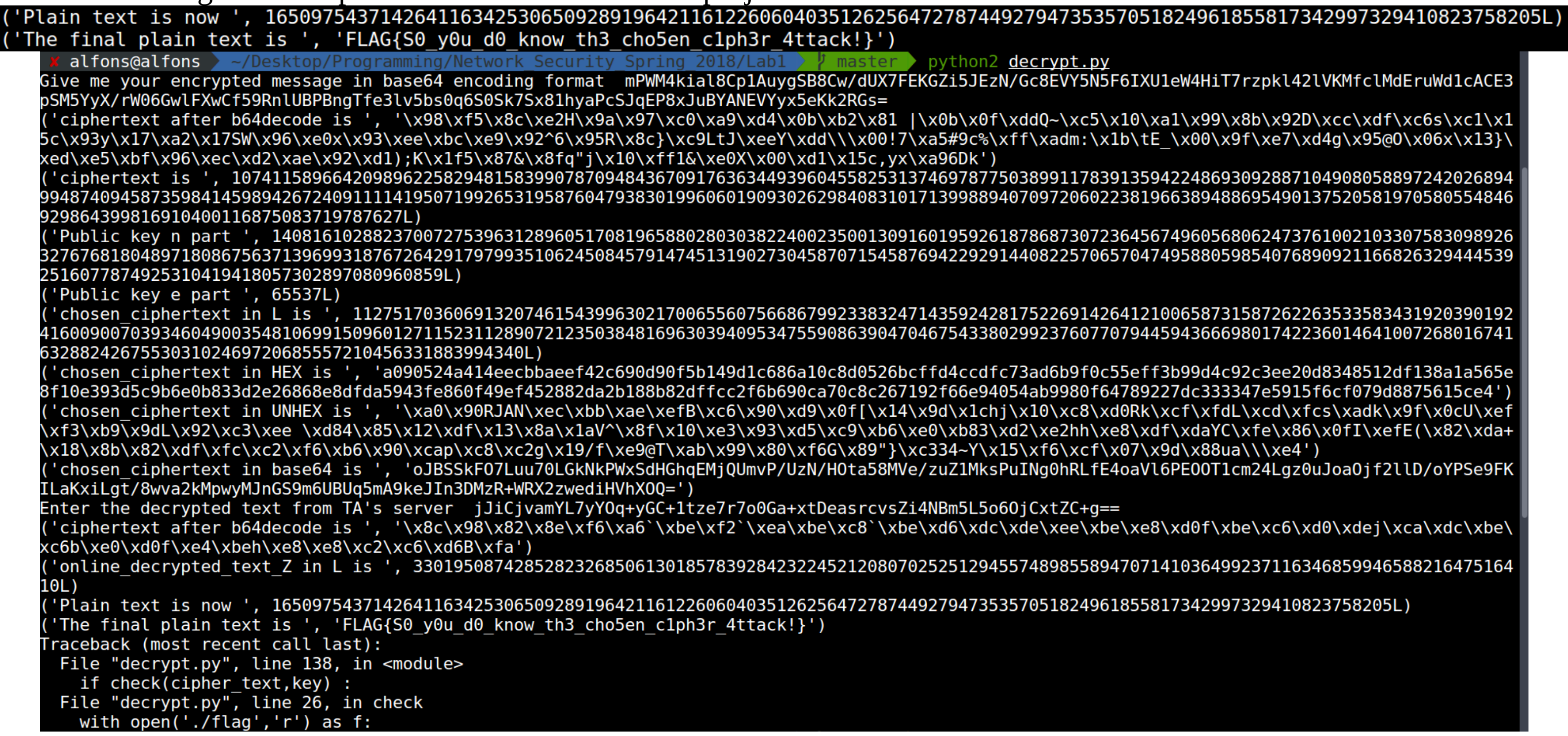
Network Security Project 1 Report

An-Fong Hwu 胡安鳳 0416324

* What is the “Chosen Ciphertext Attack?”

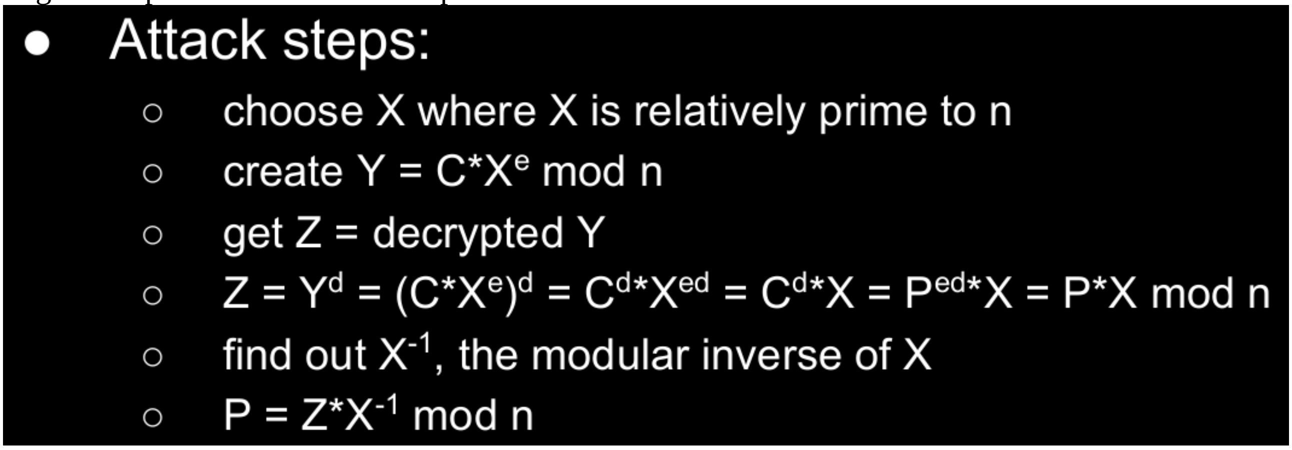
The chosen cipher attack is an attack where the cryptanalyst can gather information by obtaining the decryptions of chosen ciphertexts. From these pieces of information the adversary can attempt to recover the hidden secret key used for decryption.

*Algorithm implementation of CCA in this project



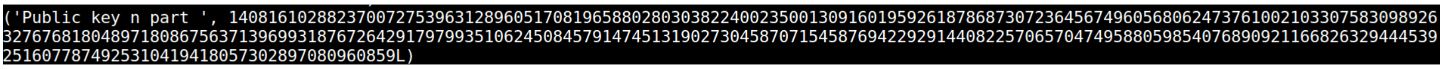
```
('Plain text is now ', 1650975437142641163425306509289196421161226060403512625647278744927947353570518249618558173429973294108237582O5L)
('The final plain text is ', 'FLAG{S0_y0u_d0_know_th3_cho5en_c1ph3r_4ttack!}')
 ✘ alfons@alfons  ~/Desktop/Programming/Network Security Spring 2018/Lab1   master  python2 decrypt.py
Give me your encrypted message in base64 encoding format  mPWM4kial8Cp1AuygSB8Cw/dUX7FEKGZi5JEzN/Gc8EVY5N5F6IXU1eW4HiT7rzpkl42lVKMfclMdEruWd1cACE3
pSM5YyX/rW06GwlFXwCf59RnlUBPBngTfe3lv5bs0q6S0Sk7Sx81hyaPcSJqEP8xJuBYANEVYyx5eKk2RGs=
('ciphertext after b64decode is ', '\x98\xf5\x8c\xe2H\x9a\x97\xc0\xa9\xd4\x0b\xb2\x81 |\x0b\x0f\xddQ~\xc5\x10\xa1\x99\x8b\x92D\xcc\xdf\xc6s\xc1\x1
5c\x93y\x17\xa2\x17SW\x96\xe0x\x93\xee\xbc\xe9\x92^6\x95R\x8c}\xc9LtJ\xeeY\xdd\\\x00!7\xa5#9c%\xff\xadm:\x1b\tE_\x00\x9f\xe7\xd4g\x95@O\x06x\x13}\
xed\xe5\xbf\x96\xec\xd2\xae\x92\xd1);K\x1f5\x87&\x8fq"j\x10\xff1&\xe0X\x00\xd1\x15c,yx\xa96Dk')
('ciphertext is ', 1074115896642098962258294815839907870948436709176363449396045582531374697877503899117839135942248693092887104908058897242026894
9948740945873598414598942672409111141950719926531958760479383019960601909302629840831017139988940709720602238196638948869549013752058197058O554846
9298643998169104001168750837197876 27L)
('Public key n part ', 140816102882370072753963128960517081965880280303822400235001309160195926187868730723645674960568062473761002103307583098926
3276768180489718086756371396993187672642917979935106245084579147451319027304587071545876942292914408225706570474958805985407689092116682632944 4539
2516077874925310419418057302897080960859L)
('Public key e part ', 65537L)
('chosen_ciphertext in L is ', 11275170360691320746154399630217006556075668679923383247143592428175226914264121006587315872622635335834319203 90192
4160090070393460490035481069915096012711523112890721235038481696303940953475590863904704675433802992376077079445943666980174223601464100726801 6741
63288242675530310246972068555721045633188399434OL)
('chosen_ciphertext in HEX is ', 'a090524a414eecbbaeef42c690d90f5b149d1c686a10c8d0526bcffd4ccdfc73ad6b9f0c55eff3b99d4c92c3ee20d8348512df138a1a565e
8f10e393d5c9b6e0b833d2e26868e8dfda5943fe860f49ef452882da2b188b82dffcc2f6b690ca70c8c267192f66e94054ab9980f64789227dc333347e5915f6cf079d8875615ce4')
('chosen_ciphertext in UNHEX is ', '\xa0\x90RJAN\xec\xbb\xae\xefB\xc6\x90\xd9\x0f[\x14\x9d\x1chj\x10\xc8\xd0Rk\xcf\xfdL\xcd\xfcs\xadk\x9f\x0cU\xef
\xf3\xb9\x9dL\x92\xc3\xee \xd84\x85\x12\xdf\x13\x8a\x1aV^\x8f\x10\xe3\x93\xd5\xc9\xb6\xe0\xb83\xd2\xe2hh\xe8\xdf\xdaYC\xfe\x86\x0fI\xefE(\x82\xda+
\x18\x8b\x82\xdf\xfc\xc2\xf6\xb6\x90\xcap\xc8\xc2g\x19/f\xe9@T\xab\x99\x80\xf6G\x89"}\xc334~Y\x15\xf6\xcf\x07\x9d\x88ua\\\xe4')
('chosen_ciphertext in base64 is ', 'oJBSSkFO7Luu70LGkNkPWxSdHGhqEMjQUmvP/UzN/HOta58MVe/zuZ1MksPuINg0hRLfE4oaVl6PEOOT1cm24Lgz0uJoaOjf2llD/oYPSe9FK
ILaKxiLgt/8wva2kMpwyMJnGS9m6UBUq5mA9keJIn3DMzR+WRX2zwediHVhXOQ=')
Enter the decrypted text from TA's server  jJiCjvamYL7yYOq+yGC+1tze7r7o0Ga+xtDeasrcvsZi4NBm5L5o6OjCxtZC+g==
('ciphertext after b64decode is ', '\x8c\x98\x82\x8e\xf6\xa6`\xbe\xf2`\xea\xbe\xc8`\xbe\xd6\xdc\xde\xee\xbe\xe8\xd0f\xbe\xc6\xd0\xdej\xca\xdc\xbe\
xc6b\xe0\xd0f\xe4\xbeh\xe8\xe8\xc2\xc6\xd6B\xfa')
('online_decrypted_text_Z in L is ', 3301950874285282326850613018578392842322452120807025251294557489855894707141036499237116346859946588216475164
10L)
('Plain text is now ', 1650975437142641163425306509289196421161226060403512625647278744927947353570518249618558173429973294108237582O5L)
('The final plain text is ', 'FLAG{S0_y0u_d0_know_th3_cho5en_c1ph3r_4ttack!}')
Traceback (most recent call last):
  File "decrypt.py", line 138, in <module>
    if check(cipher_text,key) :
  File "decrypt.py", line 26, in check
    with open('./flag','r') as f:
```

Algorithm procedure as shown in pdf from TA.



- Attack steps:
  - choose X where X is relatively prime to n
  - create $Y = C^*X^e \bmod n$
  - get Z = decrypted Y
  - $Z = Y^d = (C^*X^e)^d = C^{d*}X^{ed} = C^{d*}X = P^{ed*}X = P^*X \bmod n$
  - find out X^{-1}, the modular inverse of X
  - $P = Z^*X^{-1} \bmod n$

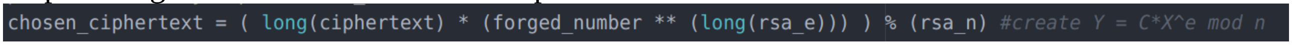
Step 1. As shown on above, the rsa-n is a huge integer, and X should be relatively prime to N, since N is odd number



```
('Public key n part ', 140816102882370072753963128960517081965880280303822400235001309160195926187868730723645674960568062473761002103307583098926
32767681804897180867563713969931876726429179799351062450845791474513190273045870715458769422929144082257065704749588059854076890921166826329444539
2516077874925310419418057302897080960859L)
```

and we have to choose an X for co-prime, the smallest select I will try 2(the forged number in source code).

Step 2. We get Y with the mathematical operation below.

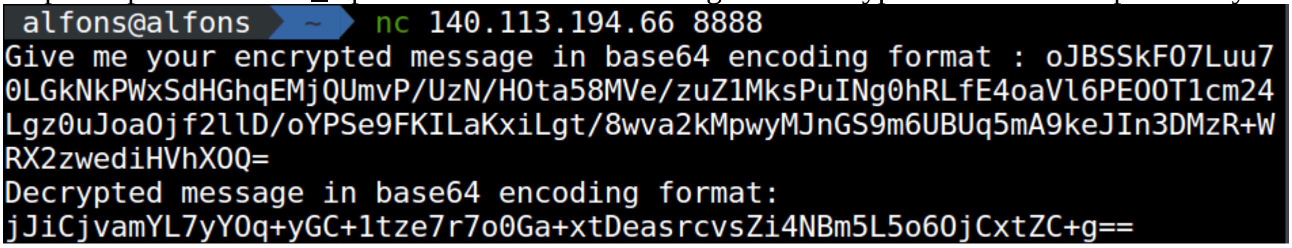


```
chosen_ciphertext = ( long(ciphertext) * (forged_number ** (long(rsa_e))) ) % (rsa_n) #create Y = C*X^e mod n
```

Since the encrypted data are written in the base64 form, hence the following step is needed:
a. decode with base64.b64decode (base64-->ASCII).
b. re-encode with hex, every byte(ASCII is encoded byte-wisely) can be convert to 2 hex number.
c. do chosen_ciphertext operation , that is chosen_ciphertext = ciphertext * (forged_number ^ rsa_e) % rsa_n
d. re-dncode with ASCII, every 2 hex can be converted to a byte,namely ASCII-encoded data.
e. encode with base64.b54encode (ASCII-->base64).

in that, the a b and d e are reverse operations to each other.

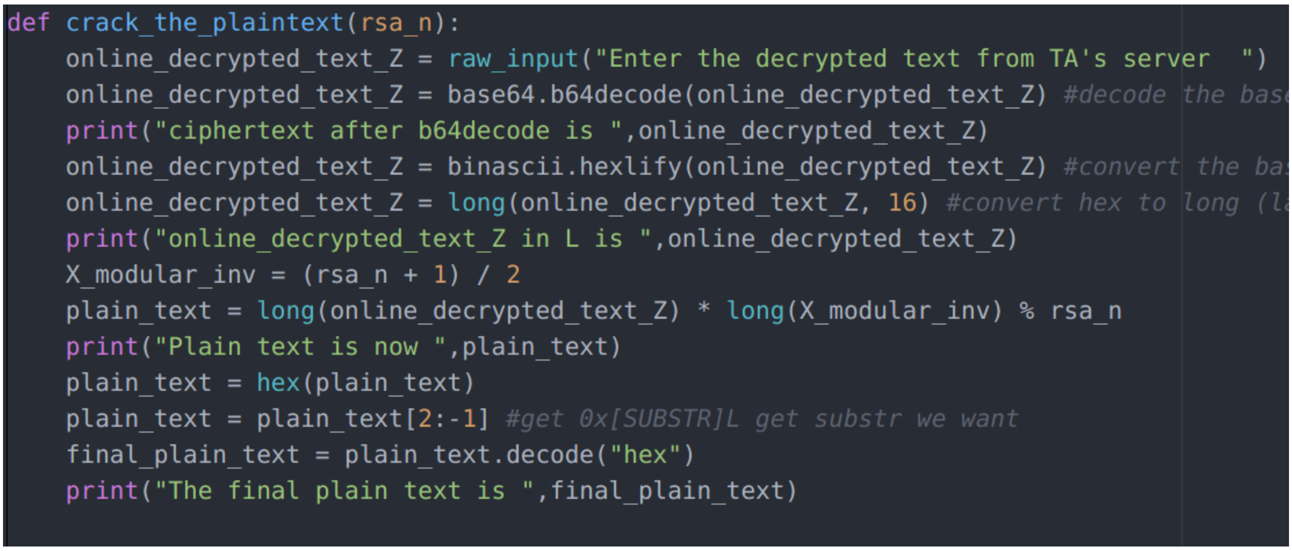
Step 3. Input the chosen_ciphertext into TA’s server to get Z = decrypted Y from TA’s public key.



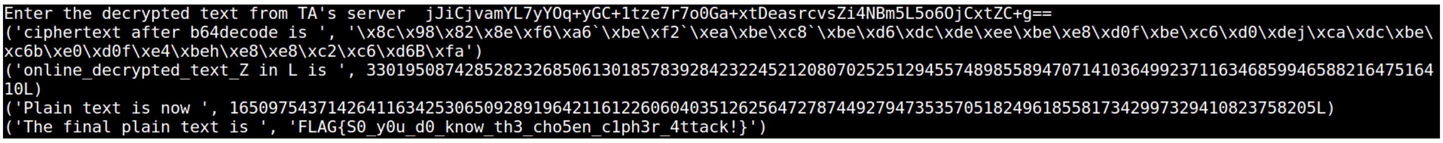
```
 alfons@alfons  ~  nc 140.113.194.66 8888
Give me your encrypted message in base64 encoding format : oJBSSkFO7Luu7
0LGkNkPWxSdHGhqEMjQUmvP/UzN/HOta58MVe/zuZ1MksPuINg0hRLfE4oaVl6PEOOT1cm24
Lgz0uJoaOjf2llD/oYPSe9FKILaKxiLgt/8wva2kMpwyMJnGS9m6UBUq5mA9keJIn3DMzR+W
RX2zwediHVhXOQ=
Decrypted message in base64 encoding format:
jJiCjvamYL7yYOq+yGC+1tze7r7o0Ga+xtDeasrcvsZi4NBm5L5o6OjCxtZC+g==
```

Z is jJiCjvam…...+g== (this is also encoded in base64)

Step 4.
P , the plaintext equals to Z*X^-1 mod n, where X is the modular inverse of X under N.
Since X = 2, then X^-1 under N will be (N+1)/2. So the final plaintext can be acquired.
We just need to be cautious about the encoding as well, the decoding is similar to that of above, but we decode with hex to ASCII for final FLAG information.



```
def crack_the_plaintext(rsa_n):
    online_decrypted_text_Z = raw_input("Enter the decrypted text from TA's server  ")
    online_decrypted_text_Z = base64.b64decode(online_decrypted_text_Z) #decode the bas
    print("ciphertext after b64decode is ",online_decrypted_text_Z)
    online_decrypted_text_Z = binascii.hexlify(online_decrypted_text_Z) #convert the ba
    online_decrypted_text_Z = long(online_decrypted_text_Z, 16) #convert hex to long (l
    print("online_decrypted_text_Z in L is ",online_decrypted_text_Z)
    X_modular_inv = (rsa_n + 1) / 2
    plain_text = long(online_decrypted_text_Z) * long(X_modular_inv) % rsa_n
    print("Plain text is now ",plain_text)
    plain_text = hex(plain_text)
    plain_text = plain_text[2:-1] #get 0x[SUBSTR]L get substr we want
    final_plain_text = plain_text.decode("hex")
    print("The final plain text is ",final_plain_text)
```



```
Enter the decrypted text from TA's server  jJiCjvamYL7yYOq+yGC+1tze7r7o0Ga+xtDeasrcvsZi4NBm5L5o6OjCxtZC+g==
('ciphertext after b64decode is ', '\x8c\x98\x82\x8e\xf6\xa6`\xbe\xf2`\xea\xbe\xc8`\xbe\xd6\xdc\xde\xee\xbe\xe8\xd0f\xbe\xc6\xd0\xdej\xca\xdc\xbe\
xc6b\xe0\xd0f\xe4\xbeh\xe8\xe8\xc2\xc6\xd6B\xfa')
('online_decrypted_text_Z in L is ', 3301950874285282326850613018578392842322452120807025251294557489855894707141036499237116346859946588216475164
10L)
('Plain text is now ', 165097543714264116342530650928919642116122606040351262564727874492794735357051824961855817342997329410823758205L)
('The final plain text is ', 'FLAG{S0_y0u_d0_know_th3_cho5en_c1ph3r_4ttack!}')
```

Note: the plain_text[2:-1] is used since 0x[HEX NUMBER]L will be parsed, we just want the HEX NUMBER in it.
*More thoughts about this project

1. Why do we use the base-64 encoding in this project?
To prevent data loss, which is quite important in Network Security since once a data is lost, a system vulnerability may be found.

Reference to:
https://stackoverflow.com/questions/201479/what-is-base-64-encoding-used-for

and

https://stackoverflow.com/questions/4070693/what-is-the-purpose-of-base-64-encoding-and-why-it-used-in-http-basic-authentica