Easy:

Sanity Check:

Just scroll down to the bottom of the assignment pdf and obtain the flag.

Something's off:

Since we know the first 4 characters of the flag are CS2107, we already know that "QgGFEL" represents "CS2107" character by character. We then know each character is shifted by 14. Assume [0:25] means A-Z, [26:51] means a-z and [52:61] means 0-9, we can obtain the actual characters.

```
mapping = {
    "E" : "0",  #confirmed
    "F" : "1",  #confirmed
    "G" : "2",  #confirmed
    "H" : "3",
    "I" : "4",
    "J" : "5",
    "K" : "6",
    "L" : "7",  #confirmed
    "M" : "8",
    "N" : "9",
    "o" : "a",
    "p" : "b",
    "q" : "c",
    "r" : "d",
    "s" : "e",
    "t" : "f",
    "u" : "g",
    "v" : "h",
    "w" : "i",
    "x" : "j",
    "y" : "k",
    "z" : "l",
    "0" : "m",
    "1" : "n",
    "2" : "o",
    "3" : "p",
    "4" : "q",
    "5" : "r",
    "6" : "s",
    "7" : "t",
    "8" : "u",
    "9" : "v",
    "A" : "w",
    "B" : "x",
    "C" : "y",
    "D" : "z",
    "O" : "A",
    "P" : "B",
    "Q" : "C",  #confirmed
    "R" : "D",
    "S" : "E",
    "T" : "F",
    "U" : "G",
```

```
    "V" : "H",
    "W" : "I",
    "X" : "J",
    "Y" : "K",
    "Z" : "L",
    "a" : "M",
    "b" : "N",
    "c" : "O",
    "d" : "P",
    "e" : "Q",
    "f" : "R",
    "g" : "S",  #confirmed
    "h" : "T",
    "i" : "U",
    "j" : "V",
    "k" : "W",
    "l" : "X",
    "m" : "Y",
    "n" : "Z"
}
#def decrypt(cipher):
def translate(cipher):
    plain = ""
    for char in cipher:
        if (char == '_' or char == '{' or char == '}'):
            plain = plain + str(char)
        else:
            plain = plain + mapping[char]
    return plain

print(translate(cipher))
```

MAC:

Go to the directory of the text file, and by Google Search, we know the syntax for openssl to generate the sha256 key of the HMAC is this command:

```
openssl dgst -sha256 -hmac "CS21072022" text.txt
```

Prime Time:

First, use online tool to factorize the modulus to obtain the value p and q, and then use them to compute the Euler totient. With this information, we can compute the private key (mi), which is the multiplicative inverse of the public exponent mod the modulus. After this, we already have a ciphertext, its private key and the modulus, using this three to obtain the plaintext, and then convert it to hexadecimal so that it can be decoded.

```
print("\nPrime Time Question")
cipher = 790720275960851803934057676432939180157671812730674675700641077222428
public_e = 65537
modulus = 1812872752217772943534763458716829296898731831681243593217411777434002 9
p = 126198501118389160989977983392586327
q = 14365247892422139769614670989751962 7
assert p*q == modulus
phi = (p - 1) * (q - 1)
mi = pow(public_e, -1, phi)
assert (mi*public_e)%phi == 1
plain = pow(cipher, mi, modulus)
print(plain)
plain = str(hex(plain))
print(plain)
print(bytes.fromhex(plain[2:len(plain)]).decode('utf-8', "ignore"))
```

```
CS2107{rSa_n33d5_g00d_pr1m3s}
```

Secret Penguin:

After Google Search, we know the syntax for openssl to encrypt the png file using AES CBC mode of operation is the following command:

```
openssl enc -aes-128-cbc -p -K 1234567890abcdef1234567890abcdef -iv
abcdef1234567890abcdef1234567890 -in tux.png -out tux.enc

Follow this command to get the sha256 of the encrypted file:
openssl dgst -sha256 tuc.enc
```

Medium:

Insecure OTP:

```
encrypted = "faa4a0ba8d435a2b2015c4625c80443e820c523a9ee190baa2504d20640cca2e6bc
msg = "Hey Grandma Susan'oo, I have told you not to play with my Photoshop! \
Why did you crop your head on the dragon... "
key = xor(msg.encode()[0:20], bytearray.fromhex(encrypted)[0:20])
print(key)

def decrypt(msg, key):
    res = b""
    for i in range(0, len(msg), 20):
        res += xor(key, msg[i:i+20])
    return res.decode('ascii', 'ignore')

print(decrypt(bytearray.fromhex(encrypted), key))
```

The rationale of this question is that a XOR (a XOR k) = k, hence by XOR-ing the first 20 characters of the message and the encrypted message (converted from hex to original string), we can obtain the key, and then by XOR-ing the key and the encrypted message, we can get back the original message.

Public Password:

According to the question, the social media is “blue bird”, which implies twitter, so we can obtain the password from that account actually…

Then we go to our Linux system, and enter the command `nc cs2107-ctfd-i.comp.nus.edu.sg 4003`

By entering the password, we obtain the flag from the server.

Substitution Cipher:

According to the question, the encrypted text is believed to be the terms and conditions of a software, hence by observing the encrypted text, we know that

`UT2107{TAQTHWAWHLD_UHSFYX_HT_LOWYD_TFLRD_HD_KLMHY_OLX_TLKY_XYBTLD}`

U → C, T → S

Then by further inspection, I realize:

(H)

(HH)

(HHH)

(HM) H → I and M → V

Also, the order BQUVYONFH represents ABCDEFGHI. Once I try to substitute the known characters, I obtain the flag, which can be read as English text already. Hence, I can know the flag.

Offline Password Cracking:

First, use “sudo apt-get install john” to download the password cracking tool. Then navigate to the directory of the text file. On that directory, type the command “john –-user=bob stolenshadow.txt” to crack the password. Then use the command “john –show stolenshadow.txt” to show the available password:

bob:abcd1234:1003:1003:Bob,,,:/home/bob:/bin/bash

```
tanruiyang@MSI:/mnt/c/Users/User/Documents/CS2107/Assignments/CTF1/offline-password-cracking/dist$ john --users=bob stolenshadow.txt
Created directory: /home/tanruiyang/.john
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Press 'q' or Ctrl-C to abort, almost any other key for status
abcd1234         (bob)
1g 0:00:00:06 100% 2/3 0.1466g/s 506.7p/s 506.7c/s 506.7C/s sniper..bigben
Use the "--show" option to display all of the cracked passwords reliably
Session completed
tanruiyang@MSI:/mnt/c/Users/User/Documents/CS2107/Assignments/CTF1/offline-password-cracking/dist$ john --show stolenshadow.txt
bob:abcd1234:1003:1003:Bob,,,:/home/bob:/bin/bash

1 password hash cracked, 0 left
```

According to the hint, we can only choose one password “abcd1234”.

Birthday Hash:

The byte to generate the SHA512 is given as the sum of the event + a random string. Inspired by the birthday paradox, we realized it is possible to have two hash values collide with each other (having exactly the same hash value although the byte strings are different). Hence, we only need to use computer to brute force generate some random hash values and check if got two hash values collide with one another. The code to generate the random hash value is as follow:

```
print("\nBirthday Hash")
import ...

event_hex = "2433076380ef23a5c534c195a663f809f2366947"
event = bytes.fromhex(event_hex)
lookup_table = dict()
seq = [str for str in string.printable if str not in string.whitespace]
for _ in range(2**48):
    random_byte = ''.join(random.sample(seq, 8)).encode()
    result = hashlib.sha512(event + random_byte).digest()[:6]
    if result in lookup_table and random_byte != lookup_table[result]:
        print("One of the possible combination is:")
        temp1 = random_byte.decode('utf-8', "ignore")
        temp2 = lookup_table[result].decode('utf-8', "ignore")
        if (sha512(event + temp1.encode()).digest()[:6] == sha512(event + temp2.encode()).digest()[:6]):
            print(temp1, temp2)
            break
    else:
        lookup_table[result] = random_byte
print("The flag is CS2107{No_h@sh_can_esc4pe_b1rthd@y_p@rad0x}")
```

Hard:

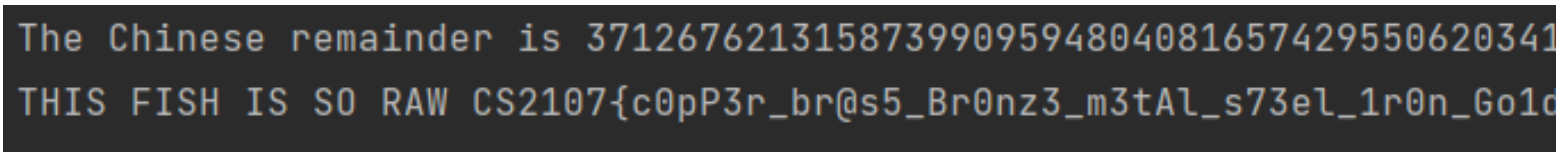
Copper-rsa

By searching information from wikepedia, I realized that a coppersmith-attack on rsa consists of these steps:

1. Have a list of x = c mod n (in this case, we have 5 c in c_arr, and 5 n in n_arr, the x is the plaintext)
2. Compute N = n1*n2*n3*…*ni (in this case, i = 5)
3. For each ni, compute Ni, where Ni = N / ni
4. Find multiplicative inverse of Ni mod ni
5. Compute c_arr[i] * Ni * n_arr[i]
6. Repeat this for all i in range(len(n_arr)), and add all the calculated value in step 5 together
7. Compute the summation mod N

After we have done all the steps mentioned above, we will know m^3, we then take cube root of this value, and this is the value of the plaintext in rsa.

Now, by solving the given quadratic equation with the help of the value calculated just now and do decoding, we will know the flag.



Rsa locked doors:

For door 1, since we have ciphertext c, private key d, and the Euler totient function phi, we can know the e by finding multiplicative inverse of d mod phi. We can also do brute force search to find all possible modulo n. We first obtain the factors of phi, and find all possible combination to get n:

```
def find_correct_modulo_n():
    i = 0
    next = 0
    for p_facs, q_facs in combinations:
        p1, q1 = 1, 1
        i += 1
        for pn in p_facs:
            p1 *= pn
        for qn in q_facs:
            q1 *= qn
        p, q = p1 + 1, q1 + 1
        plain = pow(c, d, p*q)
        try:
            print(Crypto.Util.number.long_to_bytes(plain).decode())
            break
        except:
            next += 1

find_correct_modulo_n()
password = "p00r{RSA_7z8o1}"
```

For door 2, by inspection, we first know that the public key of Bob and Alice are provided, the public exponent e is 65537 (in hexadecimal) and their modulo n are also given. Also, I realize that each data is 32 bytes, which is able to be decoded using base32 in Python.

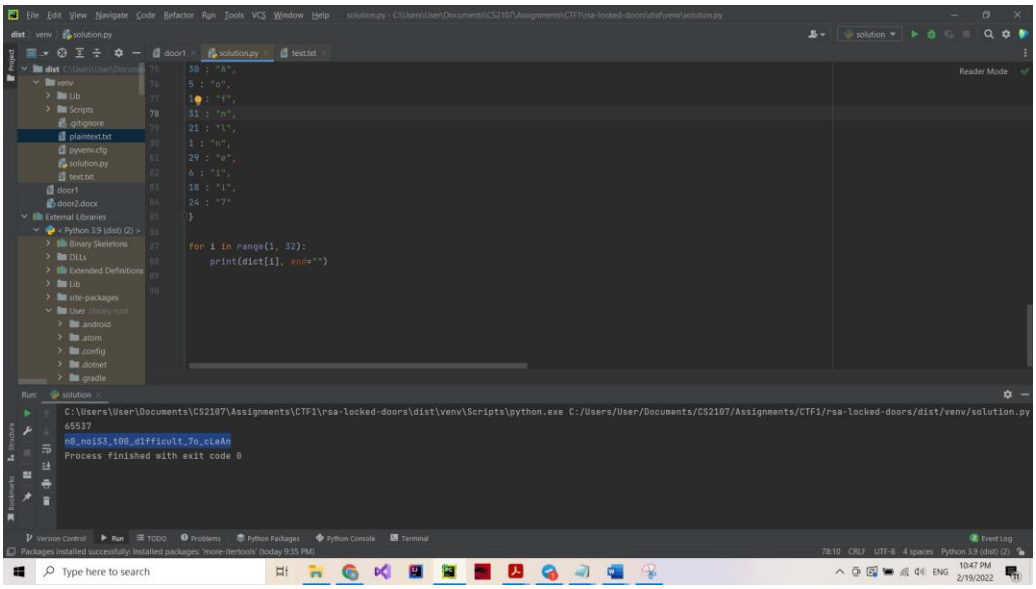
Once we start decoding, we will get the data, seq, and the signature like this:

```
SEQ: 29; DATA: 0x43d7799c50fd68141aaf6f688c6ace10; SIGNATURE: 0xd51639586a76318f36d2517fecd1f55

SEQ: 13; DATA: 0x79eca834168ce9bbee768d10b20bf03a; SIGNATURE: 0x1c652e27c4ef34c2acc7d7f28fcdc691

SEQ: 24; DATA: 0x7bcaf279a7e1415a2f7d0e1ecfcf52ab; SIGNATURE: 0x54614f571c50e65b699bc415632f96a6

SEQ: 31; DATA: 0x13849a92c41cb8b0b85b84073070476e; SIGNATURE: 0xf1d41d1a914d3347852dd1aeaa13d32

SEQ: 23; DATA: 0xcf4c800613c4a8bcada37b461a76036; SIGNATURE: 0x173f5129c2964e3c8e0c1be2b7a180ce

SEQ: 34; DATA: 0x757369ddb890ec4e6ce5dca8d133632a; SIGNATURE: 0x74a407bb5ecb9065131a214ec68ccd9

SEQ: 7; DATA: 0x3111e024fc44e82354b5885e7c3a49a7; SIGNATURE: 0x1a45f4408171c153efa60a4176a1a207

SEQ: 28; DATA: 0x76144142dc6e28df44c97d4187072b26; SIGNATURE: 0xe1d940948ff16b8664914477dba5e9a

SEQ: 27; DATA: 0x3111e024fc44e82354b5885e7c3a49a7; SIGNATURE: 0x1a45f4408171c153efa60a4176a1a207

SEQ: 30; DATA: 0x18c846c4f53523d5b471a1b1cfbda3a1; SIGNATURE: 0xe4573fef148c9192745ccec56c303fa

SEQ: 28; DATA: 0x757369ddb890ec4e6ce5dca8d133632a; SIGNATURE: 0x74a407bb5ecb9065131a214ec68ccd9

SEQ: 5; DATA: 0x3111e024fc44e82354b5885e7c3a49a7; SIGNATURE: 0x1a45f4408171c153efa60a4176a1a207

SEQ: 33; DATA: 0x6ef5eb53b51d10e4cff3e003a0eb3913; SIGNATURE: 0xefb578f66e6f21656c6911cd7c3dcce

SEQ: 29; DATA: 0x5e84afba3d0e47a9a38b68e8c18a3f55; SIGNATURE: 0x14c7ad6c2c1349682724a2664cb68976
```

Then we use the signature as the ciphertext to decrypt using the public exponent 65537 and the modulo of Bob. However, it contains a lot of incorrect result, hence we only gather the correct information:

```
7 : S
8 : 3
15 : 1
20 : u
25 : o
11 : 0
26 : _
28 : L
2 : 0
14 : d
3 : _
22 : t
13 : _
17 : f
23 : _
10 : t
27 : c
9 : _
19 : c
12 : 0
4 : n
30 : A
5 : o
16 : f
31 : n
21 : l
1 : n
29 : e
6 : i
18 : i
24 : 7
```

Then, by arranging them with their sequence given, we can get the flag:



Hence, the flag is CS2107{n0_noiS3_t00_d1fficult_7o_cLeAn}.