Q1.

Confidentiality：

訊息、資料不能有未經授權的洩漏或存取。例如在資料傳輸的過程中有第三者的監聽就會違反了 confidentiality requirement。

Integrity：

防止訊息有未經授權的篡改。例如資料傳輸的過程中有一個中間人攔截了傳送的訊息並且篡改後傳給另一方，就違反了 integrity requirement。

Availability：

確保 user 在需要的時候一定可以使用到服務。例如 DDOS 癱瘓網站讓使用者沒辦法使用服務與阻斷加密通訊的效果很類似，違反了 Availability requirement。

Q2.

One-wayness：

給定 output y，很難找到一個 x 滿足 Hash(x) = y。例如在資料庫存取大量的密碼 hash 值，Attacker 就算拿到了流出的 hash value 也難以推測可能的真實密碼。

Weak collision：

給定 x，很難找到 x' != x 使得 Hash(x') =Hash(x)。例如以往下載檔案的時候常有 MD5 做比對確認，背後的假設就是很難產生一個其他的檔案其擁有與原下載檔案一樣的 hash value。

Strong collision：

很難找到相異的 x、x' 使得 Hash(x')=Hash(x)。例如簽章的時候根據使用情境可能需要有沒辦法找到 strong collision 的性質。

Q3.

Setup：

Hash function: Hash(msg)

Large prime: q

Generator: g

Shamir's Polynomial = $S(x) = a_0 + a_1x + a_2x^2 \ldots + a_{t-1}x^{t-1}$

Secret key shares of n members : $sk_1 = S(1)$、$sk_2 = S(2)$、…、$sk_n = S(n)$

Public key of n members: $pk_1 = g^{S(1)} \pmod q$、…、$pk_n = g^{S(n)} \pmod q$

User Signature Signing for the i^{th} member:

Plaintext: m

Hash value: h = Hash(m)

Signature : $\sigma_1 = h^{ski} \pmod q$

User Signature Verification:

Valid: $e(\sigma , g) = e(h , g^{ski})$

Threshold Signature Signing:

By Lagrange Interpolation：

$S(x) := \sum_{i=0}^{m} S(x) l_i(x)$ which $l_i(x) = \prod_{\substack{0\le k\le m \\ k!=i}} \frac{x-x_k}{x_i-x_k}$

i.e. Secret key share $S(i) = \sum_{i=0}^{m} S(i) l_i(x)$

Plaintext: m

Hash value: h = hash(m)

Signature: $\omega = \sigma_1^{l_1(0)} * \sigma_2^{l_2(0)} * \dots * \sigma_m^{l_m(0)} \pmod q$

$= h^{S(1)l_1(0)+S(2)l_2(0)+\cdots+S(m)l_m(0)}$

$= h^{S(0)}$

Threshold Signature Verification:

Valid: $e(\omega , g) = e\ (h^{S(1)l_1(0)+S(2)l_2(0)+\cdots+S(m)l_m(0)},\ g)$

$= e\ (g\ ,\ g^{S(1)l_1(0)+S(2)l_2(0)+\cdots+S(m)l_m(0)})$

$= e\ (\ h\ ,\ g^{S(0)})$

Q4.

Round1：

就是一般的凱薩加密，key space 很小所以只要試過所有 shift value 就可以看出 plaintext。

Round2：

是一個 Vegenere Cipher，比較要注意的是加密的時候用的是一個短的 key 然後一直重複到跟 plaintext 一樣長後再加密，並且對應到空格的話不論 shift value 是多少都仍然會是空格。要找到 key 只要觀察 c1、m1 之間的 shift value 就可以得到了。

Round3：

是一個 Rail Fence Cipher，比較要注意的是每次連線的欄數都不同，必須先比對看 c1、m1 用的欄數是多少，解 c2 的時候才好解。

Round4：

就只是一個 base64 的解碼。

Q5.

Opt-1：

首先觀察到網路連線的傳輸速度夠快，所以 server 產生 time seed 的時候我們在 local 端幾乎可以拿到一模一樣的 time seed，就可以製造出相同的 randint 來做 XOR 的解密。

```
BALSN{7ime_Se3d_Cr4ck!n9}
```

Opt-2：XXXXXXXXXXXXXXXXXXXXXXX

Q6.XXXXXXXXXXXXXXXXXXXX

Q7.

首先我們必須取得”key={}&BALSN_Coin=1”的 hash value 命名為 original_digest。之後利用 python 的 hashpumpy package 來進行 Length Extended attack。使用的方法為：

```
# create Hash(key | original_data | data_to_add)
new_digest, new_data = hashpumpy.hashpump( original_digest , original_data , data_to_add , len(key) )
```

其中 original_data 設定為已知的參數部分也就是”1”，data_to_add 設定為要 append 的字串也就是”&BALSN_Coin=1000”，key 的長度就是未知的部分也就是”key={}&BALSN_Coin=”的長度。之所以將 append_data 這樣設定是因為觀察 chal.py 中後面的 BALSN_Coin 值會覆蓋住前面的，所以 balsn_coin 會從 1 被取代成 1000。

hashpump function 的 return 值為 tuple = (new_digest , original_data + padding + data_to_add)，只要把 new_data 的部分做 base64 Encode 後回答 ”How many BALSN coin you have?” 就可以偽造 1000 BALSB coin 的 hash。

(因為我們不知道 key length 所以必須用一個迴圈把每個 key length 對應的結果都丟丟看直到 valid。)

```
Here is your flag!
BALSN{L3ngTh_3xeT3n5iON_4tTacK_i5_34sY_w1tH_H4shPump}
```

Q8.

觀察到 RSA 的 public key e 都是 3，重複使用而且值很小，連線 3 次後就可以得到 N1、N2、N3、C1、C2、C3。

令 X = m^3，用 Chinese Remainder Theorem 解聯立：

$$X \equiv C1 \pmod{N1}$$
$$X \equiv C2 \pmod{N2}$$
$$X \equiv C3 \pmod{N3}$$

得出 m^3 後再開立方根就可以得到 plaintext m 再轉換成文字。

(會用到 python “gmpy2” package 進行比較精準的開根號運算。)

```
'BALSN{Therefore_We_5hould_Not_Choose_4_5mall_Public_Key...}'
```

Q9.

p-1 是這個 mod p 之 group 的 order，利用提示裡面的數字去除 p-1 發現 691829 是 p-1 的因數，而且 $g^{691829} \equiv 1 \pmod{p}$，所以 g 確實生的是 subgroup，它並不是這個 group 的 generator，這給我們一個很好的機會去暴力猜 private key (如果 g 是 generator 我們原先可能得暴力猜 p 次)，key space 直接縮小到 range(691829)內。

在這個範圍內只要 $g^a \equiv A \pmod{p}$ 就代表我們找到了 Alice 的 Private key a，可以製造出 s。且

$$Cipher \equiv m * s \pmod{p}$$

⇨ $m \equiv Cipher * s^{-1} \pmod{p}$

⇨ 解出 plaintext m 了。

(為了節省時間在找到 a 之後我把 for loop 註解掉了。)

```
'BALSN{black magic number}
```