# Spring 2019 Cryptography and Network Security

**Homework 1**
**DUE DATE: 4/9/2019, 23:59**

學號：R07944058　系級：網媒碩一　姓名：陳鵬宇

Collaborator：李吉昌

# Handwriting

## 1. CIA (10%)

**Confidentiality:** 訊息由 Alice 傳到 Bob 的過程中，不會被第三方 Eve 給竊聽。例如：登入個人銀行帳户，輸入密碼時，不會被第三方竊取帳密。

**Integrity:** 訊息傳送到對方時，內容不會在中途被竄改。例如：copy and paste 就能做到這件事，我們可以用 checksums、cryptographic checksums 去一定程度驗證 integrity。

**Availability:** 對任何提供服務的資訊系統，我們總是能在有人有需求時，提供他服務。例如：DDoS 就破壞了 availability。

## 2. Hash Function (10%)

**One-wayness:** 將 $X$ 丟進 hash 算出 $H(X) = Y$ 後，我們無法透過 $Y$ 找回 $X$。例如：資料庫存取的密碼是 hash value，而不是 plaintext，這樣一來即使資料庫被攻擊，也不會在第一時間就將密碼的明文拱手讓人竊取。

**Weak collision resistance:** 將 $X$ 丟進 hash 算出 $H(X) = Y$ 後，我們無法找到 $X' \neq X$ 且 $H(X') = H(X)$。例如：資料庫存取的密碼是 hash value，若攻擊者能找到某個 $X' \neq X$ 且 $H(X') = H(X)$，即使攻擊者不知道原本密碼的明文是什麼，還是可以登入我們的系統。

**Strong collision resistance:** 存在 $X \neq X'$，且 $H(X) = H(X')$ 這件事是很難發生的。$H(X)$。例如：我們希望能夠借助於「唯一的」ID 來查詢資料庫，可以透過計算資料的 hash value，而不是對原始資料查詢（由於資料大小可能無限，這通常會非常慢），若沒有 strong collision resistance，就無法達成。

## 3. Threshold Signature (15%)

**Setup:**

$$\text{hash function} : Hash(msg)$$
$$\text{large prime} : q$$
$$\text{generarotr} : g$$

若要達成 $(t, n)$ threshold signature 的 schema，考慮以下的多項式：

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}, \qquad (*)$$

secret key $sk = a_0$，我們希望將 $sk$ 分成 $n$ 份，而任意從這 $n$ 份取出 $t$ 份就足以重建出 $sk$。隨機取 $t-1$ 個數字當做多項式的係數 $a_1, a_2, \ldots, a_{t-1}$，再由此多項式 (*) 建立 $n$ 個點 $D_{x-1} = (x, A(x))$，即：

$$D_0 = (1, A(1)), D_1 = (2, A(2)), \ldots, D_{n-1} = (n, A(n)).$$

若要重建 $sk$，任意 $t$ 個點便能達成，例如考慮以下 $t$ 個點：

$$(x_0, y_0), (x_1, y_1), \ldots, (x_{t-1}, y_{t-1}).$$

我們可以算出 Lagrange basis polynomials:

$$\ell_j(x) = \prod_{m=0, m\neq j}^{t-1} \frac{x - x_m}{x_j - x_m}, 0 \leq j \leq t-1.$$

可得下式：

$$\begin{aligned} f(x) &= \sum_{j=0}^{t-1} y_j \cdot \ell_j(x) \\ &= A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \end{aligned}$$

我們知道 secret key $sk = a_0 = A(0) = \sum_{j=0}^{t-1} y_j \ell_j(0)$，令 $sk_i = y_i$ 可得 $sk = \sum_{i=0}^{t-1} sk_i \ell_i(0)$，因此我們有了以下新的 threshold setup，將本來的 $sk = x$, $pk \equiv g^{sk} (\text{mod } q)$ 分成以下 $n$ 份：

$$\begin{aligned} &\text{secret key} : sk_i = y_i \\ &\text{public key} : pk_i \equiv g^{sk_i} (\text{mod } q) \end{aligned}$$

## User Signature Signing:

$$\begin{aligned} &\text{plaintext} : m \\ &\text{hash value} : h = Hash(m) \\ &\text{signature} : \sigma_i \equiv h^{sk_i} (\text{mod } q) \end{aligned}$$

## User Signature Verification:

由 User Signature Signing (mod $q$ 不影響 pairing function 性質)，可以證得 User Signature Verification：

$$e(\sigma_i, g) = e(h^{sk_i}, g) = e(h, g)^{sk_i} = e(h, g^{sk_i})$$

因為每一個人都能被獨自驗證，因此預防了有心人是要製造假簽名的問題。

## Threshold Signature Signing:

$$\begin{aligned} pk = g^{sk} = g^{\sum_{i=0}^{t-1} sk_i \ell_i(0)} = \prod_{i=0}^{t-1} (g^{sk_i})^{\ell_i(0)} = \prod_{i=0}^{t-1} (pk_i)^{\ell_i(0)} \\ \sigma = h^{sk} = h^{\sum_{i=0}^{t-1} sk_i \ell_i(0)} = \prod_{i=0}^{t-1} (h^{sk_i})^{\ell_i(0)} = \prod_{i=0}^{t-1} (\sigma_i)^{\ell_i(0)} \end{aligned}$$

### Threshold Signature Verification:

$$
\begin{aligned}
e(\sigma, g) &= e(\prod_{i=0}^{t-1}(\sigma_i)^{\ell_i(0)}, g) \\
&= e(\prod_{i=0}^{t-1}(h^{sk_i})^{\ell_i(0)}, g) \\
&= e(h^{\sum_{i=0}^{t-1} sk_i\ell_i(0)}, g) \\
&= e(h^{sk}, g) \\
&= e(h, g)^{sk} \\
&= e(h, g^{sk})
\end{aligned}
$$

# Capture The Flag

## 4. Babe crypto (10%)

**warmup**

回傳 2。

**warmup again XD**

回傳 `text =` 後面的文字。

**round1**

事先向 server 發送 1000 次請求，拿回一個 `lines_seen` set，再將此 `lines_seen` set 預處理變成 `words_seen` set，接下來就是基本的凱薩加密，窮舉每個字元 $+i$ 再 mod 26，如果加工後的字有看到我們 `words_seen` set 的任何一個字，就回傳該句話。

**round2**

根據 $(c_{1i} - m_{1i})\text{mod } 26$ 找出 $salt_j$，再用 $c_{2i} - salt_j$ 找出明文 $m_2$。

**round3**

籬笆加密，先透過 $c_1, m_1$ 找出 $key$，再用該把 $key$ 解密 $c_2$ 找出明文 $m_2$。

**round4**

單純將 $c_1$ 做 base64 decode 即可得明文 $m_1$。
拿到 FLAG 如下：

```
BALSN{CRYPTO_1S_3ASY_XDD}
```

## 5. OTP (15%)

(a) 因為程式碼中有：

```
random.seed(int(time.time()))
```

種子已經被固定了，我們只要也用我們 local 端 `random.randint(0, 255)` 的值和 `cipher` 做 XOR 就能找出 FLAG。

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

(b) 跟 (a) 很像的是，程式碼當中有：

```
random.seed(int(time.time()))
```

種子一樣被固定了，因此我們可以拿到和 server 端一樣的

```
key_index = random.randint(0, (2**64) - 1)
```

`key_index` 共有 64 bits，代表的是，決定要拿這 64 把 keys 當中的哪幾把來做 XOR。示意圖如下：

$$\left[\texttt{key\_index}\right] \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ \vdots \\ k_{63} \end{bmatrix} \oplus m$$

透過腳本向 server 拿 $N$ 組 `ciphers`，加上我們同步算的 `key_index`，可得到：

$$\texttt{ciphers}^{N\times 48} = \texttt{key\_index}^{N\times(64+1)} \cdot \texttt{keys}^{(64+1)\times 48}$$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} \texttt{key\_index}_0 & 1 \\ \texttt{key\_index}_1 & 1 \\ \texttt{key\_index}_2 & 1 \\ & \vdots \\ \texttt{key\_index}_N & 1 \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ \vdots \\ k_{63} \\ m \end{bmatrix}$$

再透過 row reduce XOR 的方式，產生出一組 `[key_index 1] = [0 0 0 ··· 0 1]` 的形式，同時左側 $c$ 也會跟做 XOR，這樣一來 `[key_index 1] = [0 0 0 ··· 0 1]` 的該列對應的 $c_i$ 就是 $m$，即：

$$0 \times k_0 \oplus 0 \times k_1 \oplus 0 \times k_1 \oplus \cdots \oplus 0 \times k_{63} \oplus m = m.$$

```
BALSN{Tria1_4nd_3rr0r_And_Tri@l_AnD_Get_Fla9<3!}
```

## 6. MD5 Collsion (10% + Bonus 5%)

這題我使用了這個工具：python-md5-collision。

MD5 函數的 block 結構，是通過 Merkle-Damgård 方法的迭代方法完成的。首先填充給定的輸入文件，使其長度為 64 bytes 的倍數。然後將其分成單獨的 64 bytes block $M_0, M_1, \ldots, M_{n-1}$。根據以下規則計算 16 bytes 狀態序列 $s_0, \ldots, s_n$ 來計算 MD5 hash：$s_{i+1} = f(s_i, M_i)$，其中 $f$ 是某個固定的函數。這裡，初始狀態 $s_0$ 是固定的，即 IV，最終狀態 $s_n$ 是計算出的 MD5 hash value。

對於給定的 IV，Wang 和 Yu 的方法 可以找到兩對 pair $M, M'$ 和 $N, N'$，使得

$$f(f(s, M), M') = f(f(s, N), N').$$

由此可以找到任意長度的 pair，除了文件中間某處的 128 個 bytes 以外，它們是相同的，並具有相同的 MD5 hash value。我們將這兩個文件寫成 64 bytes block 的序列：

$$M_0, M_1, \ldots, M_{i-1}, M_i, M_{i+1}, M_{i+2}, \ldots, M_n,$$
$$M_0, M_1, \ldots, M_{i-1}, N_i, N_{i+1}, M_{i+2}, \ldots, M_n.$$

對作何的 IV $s_i$，我們試著找出一對 pair $M_i$, $M_{i+1}$ 和 $N_i$, $N_{i+1}$ 使得：

$$s_{i+2} = f(f(s_i, M_i), M_{i+1}) = f(f(s_i, N_i), N_{i+1}).$$

這時要達成題目要求的，給定兩個不同的 base64-encoded python2 codes，卻 print 出相同的 output 就簡單了，令 `data1` $= M_i, M_{i+1}$ 和 `data2` $= N_i, N_{i+1}$。

```
Program 1: if (data1 == data1) then { good_program } else { evil_program }
Program 2: if (data2 == data1) then { good_program } else { evil_program }
```

我的結果如下：

```
code1:
IyEvdXNyL2Jpbi9lbnYgcHl0aG9uMgojIC0qLSBjb2Rpbmc6IHV0Zi04IC0qLQojICAgICAKZGlmZiA9
ICcnJ7mShecrqiBt9ggCG6VBOOKpH3LO3n/N8ifaOfGNGZOK2BMHyODKFx5vDuFEzxu2RM/yo8HJds1j
ELKMYIUi4OkJdfl/D8LgF4+TIoTgPz2eVvXHpn5Kcu0eeZ3WS3sR4A3PcuDJMc6lhncNoorsMP7iHEdB
psqzTfbnkOz6ogdFJycnCnNhbWUgPSAnJye5koXnK6ogbfYIAhulQTtCqR9yzt5/zfIn2jnxjRmTitgT
B8jgyhcebw7hRM8btkTP8qPByXbNYxCyjGCFIuNJCXX5fw/C4BePkyKE4D89nlb1x6Z+SnLtHnmd1kt7
EeANz3LgyTHOpYZ3DaKK7DD+4hxHQabKs03255Ds+qIHRScnJwoKaWYgKHNhbWUgPTOgZGlmZik6CiAg
ICBwcmludCAiTUQ1IGlzIHNlY3VyZSEiCgplbHNlOgogICAgcHJpbnQgIkp1c3Qga2lkZGluZyEiCgo=

code2:
IyEvdXNyL2Jpbi9lbnYgcHl0aG9uMgojIC0qLSBjb2Rpbmc6IHV0Zi04IC0qLQojICAgICAKZGlmZiA9
ICcnJ7mShecrqiBt9ggCG6VBOOKpH3JO3n/N8ifaOfGNGZOK2BMHyODKFx5vDuFEz5u2RM/yo8HJds1j
ELKM4IUi4OkJdfl/D8LgF4+TIoTgPz2eVvXHJn5Kcu0eeZ3WS3sR4A3PcuDJMc6lhncNoopsMP7iHEdB
psqzTfbnkGz6ogdFJycnCnNhbWUgPSAnJye5koXnK6ogbfYIAhulQTtCqR9yzt5/zfIn2jnxjRmTitgT
B8jgyhcebw7hRM8btkTP8qPByXbNYxCyjGCFIuNJCXX5fw/C4BePkyKE4D89nlb1x6Z+SnLtHnmd1kt7
EeANz3LgyTHOpYZ3DaKK7DD+4hxHQabKs03255Ds+qIHRScnJwoKaWYgKHNhbWUgPTOgZGlmZik6CiAg
ICBwcmludCAiTUQ1IGlzIHNlY3VyZSEiCgplbHNlOgogICAgcHJpbnQgIkp1c3Qga2lkZGluZyEiCgo=
```

拿到 FLAG 如下：

```
BALSN{MD5_Ch3cK5Um_!5_Br0k3N}
```

## 7. Flag Market (10% + Bonus 5%)

這題用到的是 Length extension attacks，我用了 HashPump 這個工具，這題一開始因為沒有很清楚 padding 的方式，所以即使有了工具，還是不知該怎麼使用，感謝助教在這裡提供了這份 tutorial：Everything you need to know about hash length extension attacks。

HashPump 需要 4 個 arguments：

```
hashpump(hexdigest, original_data, data_to_add, key_length) -> (digest, message)
```

我在此題 2. Buy BALSN coin，How many BALSN coin you want? 回答了 1，要通過 BALSN_Coin > 1000 的測試，所以用以下的方式取得 (digest, message)，len(KEY) + 4 是因為 "key=" 長度為 4。

```
digest, message = hashpump(token, "&BALSN_Coin=1", "&BALSN_Coin=1001", len(KEY) + 4)
```

在 3. Buy Flag，How many BALSN coin you have? (input encode in base64) 回答 hashpump 回傳的 message 去掉 original_data 的部分，Show me your token. 則是回答 hashpump 回傳的 digest。

```
base64.b64encode('1\x80\x00\x00...&BALSN_Coin=1001')
```

拿到 FLAG 如下：

```
BALSN{L3ngTh_3xeT3n5i0N_4tTacK_i5_34sY_w1tH_H4shPump}
```

Hidden flag 的部分運用了 Python 在 format 的漏洞，觀察助教提供伺服器的程式碼，可以知道 format 後才會跑出 "!;P%*"，經過一翻 Google 後，發現在 Python 2 底下：

```
'{0.__ne__.__doc__[18]}'.format('') = '!'
'{0.__init__.__doc__[29]}'.format('') = ';'
'{0.rjust.__doc__[92]}'.format('') = 'P'
'{0.__mod__.__doc__[19]}'.format('') = '%'
'{0.format.__doc__[9]}'.format('') = '*'
```

同時也發現了長度剛好符合 hidden = data['hidded_flag'][:111]

```
len('{0.__ne__.__doc__[18]}{0.__init__.__doc__[29]}{0.rjust.__doc__[92]}
    {0.__mod__.__doc__[19]}{0.format.__doc__[9]}') = 111
```

因此只要傳給 server 以下就能順利產出 hidden flag：

```
digest, message = hashpump(
    token,
    '&BALSN_Coin=1',
    '&BALSN_Coin=2147483648&hidden_flag={0.__ne__.__doc__[18]}{0.__init__.__doc__[29]}
    {0.rjust.__doc__[92]}{0.__mod__.__doc__[19]}{0.format.__doc__[9]}',
    key_len)
```

順利拿到 HIDDEN FLAG：

```
BALSN{PyThOn_FOrM4t_5trInG_C4n_B3_daNG3rOuS}
```

## 8. RSA (10%)

RSA 在選定的 $e$ 太小時，容易受到攻擊，透過連上 server $e$ 次，我們拿到：

$$\begin{aligned} c_1 &\equiv m^e \mod N_1 \\ c_2 &\equiv m^e \mod N_2 \\ &\vdots \\ c_e &\equiv m^e \mod N_e \end{aligned}$$

根據 Haåstad's braodcast attack，若我們能拿到至少 $e = 3$ 組（每一組 tuple 都有一樣的 $e$）$(e, N_i, c_i), 1 \leq i \leq e$，這樣就能透過 CRT 找出 $m^e \mod N_1 N_2 \cdots N_e$，且因為 $m < \min\{N_1, N_2, \ldots, N_e\} \Rightarrow m^e < N_1 N_2 \cdots N_e$，所以我們可以找到 $m^e$ 並開 $e$ 次根找回明文 $m$。

以這題來說 $e = 3$，我們有

$$\begin{aligned} c_1 &\equiv m^3 \mod N_1 \\ c_2 &\equiv m^3 \mod N_2 \\ c_3 &\equiv m^3 \mod N_3 \end{aligned}$$

透過 CRT 可以找到 $m^3$，再開 3 次根號就能解出明文 $m$，拿到 FLAG 如下：

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

## 9. The Backdoor of Diffie-Hellman (10%)

Alice 在一開始送了 $A = g_{backdoor}^a$ 給 Bob，根據 Hint：

$$\begin{array}{rcll} g_{old}^{\frac{p-1}{691829}} & \equiv g_{backdoor} & \mod p & \\ g_{old}^{p-1} & \equiv g_{backdoor}^{691829} & \mod p & \\ g_{old}^{p-1} \equiv 1 & \equiv g_{backdoor}^{691829} & \mod p & (*) \end{array}$$

其中 (*) 式：在 $\mod p$ 群下，每 $p$ 次就會循環一次：

$$\begin{aligned} g_{old}^{p-1} &\equiv 1 && \mod p \\ g_{old}^{p} &\equiv g_{old} && \mod p \\ g_{backdoor}^{691829} &\equiv 1 && \mod p, \end{aligned}$$

這樣代表 $g_{backdoor}$ 在 $\mod p$ 群下，每 691829 次就會循環一次，因此 Bob 可以暴搜他從 Alice 拿收到的 $A = g_{backdoor}^a$ 去找出 Alice 的密鑰 $a$，找到 $a$ 後，就能取得 $B^a = (g_{old}^b)^a = g_{old}^{ab}$ 觀察原式，找出 $g_{old}^{ab}$ 在 $\mod p$ 群的乘法反元素，即可算出明文 $m$。

$$\begin{aligned} c &\equiv g_{old}^{ab} \times m \mod p \\ (g_{old}^{ab})^{-1} \times c &\equiv m \mod p \end{aligned}$$

拿到 FLAG 如下：

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