Handwriting

1. **CIA**

   (a) confidentiality: 保密性，確定資料不會被其他未經授權的人偷看、竊取。ex: 個資外洩

   (b) integrity: 完整性，確保資料不會被未經授權的人刪減、竄改。ex: 公用資料庫被駭客入侵、竄改

   (c) availability: 可用性，確保有意院的使用者有可以使用服務的權力。ex: Dos

2. **Hash function**

   (a) one-wayness: 確保攻擊者無法在合理的時間內從hash function的結果(y)推出原本的輸入(x)。ex: 使用者的密碼經由hash function變化之後儲存於資料庫上，不會被駭客反向取得原本的密碼。

   (b) weak collision resistance: 給定某一個輸入(x)，攻擊者無法在合理時間內找到另外一個輸入(x')，具有同樣的輸出。ex: 使用者的密碼經由hash function變化之後儲存於資料庫上，不會被攻擊者輕易取得另一組不同但是等效的密碼。

   (c) strong collision resistance: 攻擊者無法在合理時間內找到兩個不同的輸入($x \neq x'$)，對同一個hash function有相同的輸出。ex: 攻擊者在知悉hash function之後，不能同時創造出兩組不同但等效的密碼/簽證，以交互使用。

3. **Multi-prime RSA**

   a. Since we know that $\begin{cases} c = m^e \pmod N \\ m = c^d \pmod N \end{cases}$, we can prove the correctness by proving

   $$m \equiv m^{ed} \pmod N.$$

   Besides, because $ed \equiv 1 \pmod{\phi(N)}$, we can write

   $$ed = t \cdot \phi(N) + 1,\ t \in \mathbb{Z}.$$

   Then we can prove by showing $m \equiv m^{ed} \pmod{p_i}$ for all integer $i \in [1, r]$. There are 2 cases.

   case 1 $m \equiv 0 \pmod{p_i}$.
   If m is a multiple of p_i, then m^{ed} is a multiple of p_i. Thus, $m \equiv 0 \equiv m^{ed} \pmod{p_i}$.

   case 2 $m \not\equiv 0 \pmod{p_i}$. (prove with Fermat's Little Theorem)

   $$m^{ed} = m \cdot m^{t \cdot \frac{\phi(N)}{(p_i-1)}(p_i-1)} = m \cdot \left(m^{(p_i-1)}\right)^{t \cdot \frac{\phi(N)}{(p_i-1)}} = m \cdot (1)^{t \cdot \frac{\phi(N)}{(p_i-1)}} \equiv m \pmod{p_i}$$

   Since we have proved that $m \equiv m^{ed} \pmod{p_i}$ for all integer $i \in [1, r]$, we can conclude that

   $$m \equiv m^{ed} \pmod{\prod_{i=1}^{r} p_i} \Rightarrow m \equiv m^{ed} \pmod N$$

   b. There are two reasons why we should use distinct prime in RSA:

- The key in security of RSA, N, is the product of the primes. Thus, if we use 2 or more same prime, N will no longer be square-free and our system becomes more weakened.
- Although it isn't necessary, applying Chinese Remainder Theorem indeed can enhance the performance of RSA. However, it requires the number we used are pairwise co-prime.

c. We can apply it to speed up the decryption as RSA with following steps.

i. Compute private key. $(d,\ p_1, ...,\ p_r)$

ii. Compute d_{p_i} for each p_i as follow: $d_{p_i} = d \pmod{(p_i - 1)}$.

iii. Given ciphertext c, compute $c_i = c^{d_{p_i}} \pmod{p_i}$.

iv. Use CRT to compute the plaintext m as follow:

- Compute $M = \sum_{i=1}^{r} (c_i \cdot y_i \cdot N_i)$, where $N_i = \dfrac{N}{p_i}$ and $y_i \cdot N_i \equiv 1 \pmod{p_i}$.
- $m = M \pmod{N}$.

proof:

We first consider $c_i = c^{d_{p_i}} \pmod{p_i}$.

$$\begin{aligned} c_i &= c^{d_{p_i}} \pmod{p_i} = (m^e)^{d_{p_i}} \pmod{p_i} = m^{d_{p_i}(p_i - 1) + 1} \pmod{p_i} \\ &= m \bmod p_i \text{ by Fermat's Little Theorem.} \end{aligned}$$

The equation means that the plaintext m is uniquely determined modulo of each p_i.
Now we can consider M and apply above results to get that

$$M = m \cdot \sum_{i=1}^{r} (y_i \cdot N_i) = m \cdot \sum_{i=1}^{r} (y_i \cdot N_1 ... N_{i-1} \cdot N_{i+1} ... N_r)$$

Then we can simplify the equation with the fact that

$$N_1 ... N_{i-1} \cdot N_{i+1} ... N_r = N_i \cdot (N_i^{-1} \bmod p_i) = 1 \bmod p_i.$$

Thus, we can get that

$$M = m \cdot \sum_{i=1}^{r} 1 = mr.$$

Hence, we can conclude that our process can generate correct plaintext.

d. i. advantages:

- speed when using large key: Since we can use multiple primes, it is easier to generate large key, which could lead to lower computational complexity of modular exponentiation operations.
- resistance to factoring attack: The method with more prime used is harder to do factorization.

ii. disadvantages:

- complexity of implementation: Since multi-prime RSA is more complicated that 2-prime RSA, it is harder to implement and maintain. Besides, it could result in additional opportunities for errors and vulnerabilities.
- lower security level when using same key size: When the key size is the same, multi-prime RSA is more vulnerable because of smaller prime. This makes attackers easier to factorize N and get private key.

4. **Fun with Semantic Security**

In all sub-problems, I assume that $\mathcal{E}'$ isn't semantically secure. That is, $\epsilon' = |\mathbb{P}[Exp^{\mathcal{E}'}(0) = 1] - \mathbb{P}[Exp^{\mathcal{E}'}(1)) = 1]|$ is non-negligible. To prove that $\mathcal{E}$ isn't semantically secure when $\mathcal{E}'$ isn't semantically secure, I use $Adv^{\mathcal{E}'}(A)$ as a subroutine to construct an adversary $Adv^{\mathcal{E}}(B)$ and draw the game graph.

a. With the figure, we can clearly know that the advantage of $\mathcal{E}$ is actually the same as that of $\mathcal{E}'$ because r is uniformly sampled and the output of A and B is identical. Following is the mathematical proof:

$$\begin{aligned}\text{Advantage of } \mathcal{E} &= |\mathbb{P}[Exp^{\mathcal{E}}(0) = 1] - \mathbb{P}[Exp^{\mathcal{E}}(1)) = 1]| \\ &= |\mathbb{P}[b'_{ss} = 1 | b_{ss} = 0] - \mathbb{P}[b'_{ss} = 1 | b_{ss} = 1]| \\ &= |\mathbb{P}[Exp^{\mathcal{E}'}(0) = 1] - \mathbb{P}[Exp^{\mathcal{E}'}(1)) = 1]|, \text{ which is known to be non-negligible.}\end{aligned}$$

Hence, I got advantage is: So far, I prove the advantage of $\mathcal{E}$ is non-negligible if $\mathcal{E}'$ isn't semantically secure. Thus, since we know that $\mathcal{E}$ is semantically secure, we can conclude that $\mathcal{E}'$ must be semantically secure.

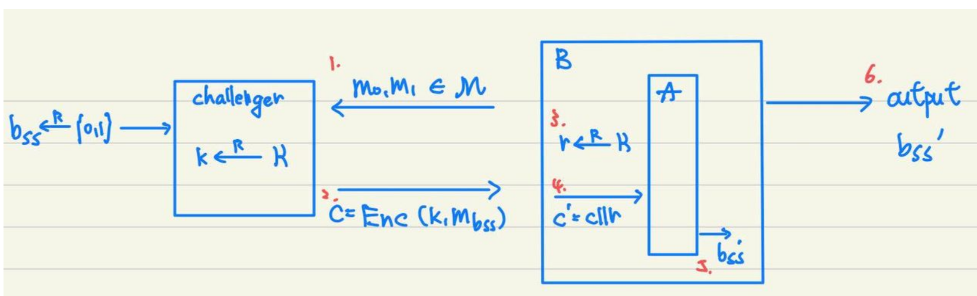

Figure 1: Game-based security definition of a.

b. Since $m_0, m_1, r \in \mathcal{M}$, $m_0 + rm_1 + r \in \mathcal{M}$. Thus, I constructed B with A and A output its guess of m'_0 or m'_1, where $(m'_0, m'_1) = (m_0 - r, m_1 - r)$ and r is randomly sampled from $\mathcal{M}$. In this way, B outputs what A outputs. Therefore, we can rewrite the advantage of $\mathcal{E}$ as:

$$|\mathbb{P}[Exp^{\mathcal{E}}(0) = 1] - \mathbb{P}[Exp^{\mathcal{E}}(1)) = 1]| = |\mathbb{P}[Exp^{\mathcal{E}'}(0) = 1] - \mathbb{P}[Exp^{\mathcal{E}'}(1)) = 1]|$$

, which is same as advantage of A and it is known to be non-negligible.
Hence, we prove that $\mathcal{E}'$ is semantically secure if $\mathcal{E}$ is semantically secure by contradiction.

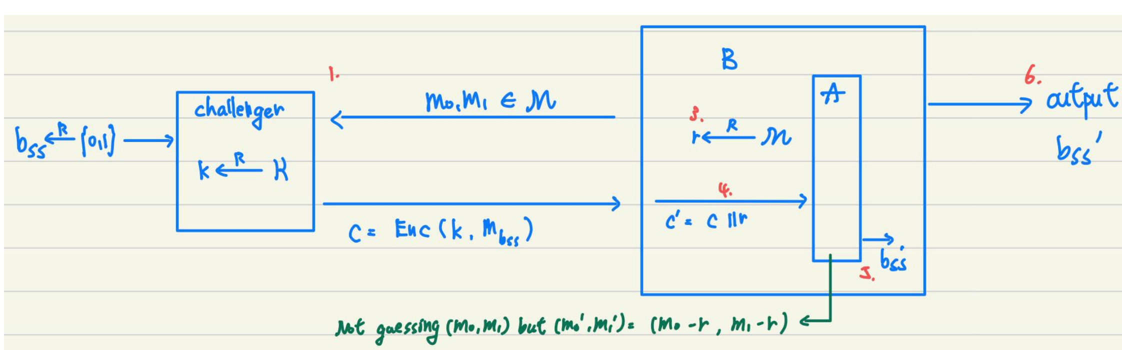

Figure 2: Game-based security definition of b.

c. Because $\mathcal{K}$ is a group with $+$, $k + r \in \mathcal{K}$. Thus, we can view this problem as:

$$A: \ Enc(k', m)||r, \text{ where } k' = r + k$$
$$B: \ Enc(k', m)$$

That is, the k is different for A and B from their definition. In this way, (c) is similar to (a). Hence, if A can distinguish $Enc(k+r, m_0)||r$ from $Enc(k+r, m_1)||r$ with non-negligible advantage, B isn't semantically secure. However, since we already know that $\mathcal{E}$ is semantically secure, $\mathcal{E}'$ should be semantically secure, too.

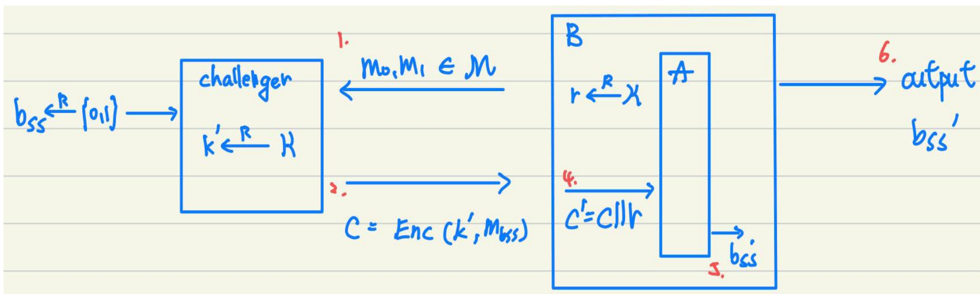

Figure 3: Game-based security definition of c.

Capture The Flag

1. **Simple Crypto:** CNS{5upeR_3asy_c1a55ic@l_cryp70!}

   - round 1: I enumerate all caesar cipher with key from 1 to 25 and choose the best answer.
   - round 2: I enumerate all rail fence cipher with key from 2 to 100 and choose the best answer.
   - round 3: With given c_1 and m_1, $c_1[i] \oplus m_1[i]$ gives us an array of corresponding keys. Then we just have to use the key to decrypt c_2.
   - round 4: First, I used Bacon's cipher to transform the sentence to A, B^*. Then, I apply rail fence cipher again.
   - final: apply base64 to get the flag.

2. **ElGamal Cryptosystem**

   a. CNS{n0_r3us3d_3ph3m3ra1_K3Y!}
   I encrypt my own message(m'), which gives me (c'_1, c'_2). Then, I can get original message(m) by $m = c_2 \cdot (c'_2)^{-1} \cdot m'$.

   c. CNS{l4gr4ng3_P0lyn0m14L_12_s0_34SY}
   First, I collected five partial decryption. Since partial decryption is $C_1^{f(i)\%p}$ not $f(i)$, we should find the power number of each partial decryption with Lagrange interpolation. Then, I simply multiply all of them to get $C_1^{sk\%p}$, which can be used as secret key to find the flag. However, because it isn't C_1^{sk}, we may need to retry several time to get the flag.

3. **Bank**

   a. CNS{ha$h_i5_m15used}

   b. CNS{$ha1_15_n0t_c0ll1510n_r3s1stnt}
   This problem requires us to find a pair of username with same value of sha1(). Thus, I use the 2 PDF files announced by google as collision. To summarize the method released together, it tries to find P and two block about to collide and concatenate them. Besides, no matter what we concatenate to this pair, they still serve as a pair of collision. Hence, I simply concatenated the 2 files with b'I love CNS'.

4. **Clandestine Operation**

   a. CNS{Aka_BIT_f1ipp1N9_atTaCk!}
   Nahida can serve as a padding oracle. Thus, all I have to do is implement padding oracle attack to get the plaintext except the first block.

   b. CNS{W15h_y0U_hav3_a_n1c3_d@y!}
   The first block of the previous problem can be found to be "job title:Grand ", and the content of the second block isn't concerned in this problem. Hence, I copy the ID and change the plaintext block, m_2, to the desired content. However, we also need to change ciphertext, c_1, which could result to invalid decoding. Therefore, I search through possible combination of the first two bytes to construct a valid ciphertext. Since the content of the first two bytes isn't concerned, too, I can randomly alter them.