Cryptography and Network Security   HW1 report   B07902116 資工三 陳富春

1. CIA

(1) Confidentiality :

Confidentiality is protection from unauthorized disclosure. That is, the data is inaccessible by anyone without authority if the confidentiality holds.

Example : Messenger offers a channel with confidentiality to text to friend. Only the user and people the user text to can access the messages.

(2) Integrity :

Intergity is protection from unauthorized changes. That is, the data is unchangeable by anyone without authority if the integrity holds.

Example : Only teacher or TAs can modify the grades on NTUCOOL. Unauthorized users such as students cannot change the grades.

(3) Availability :

Availability ensures intended users can access service. That is, users can access data or service normally if the availability holds.

Example : Google’s services such as search engine or mailbox are stable and thus achieve availability.

2. Hash Function

(1) One-wayness :

For a hash function H, given an output y, it is hard to find x such that $\mathrm{H(x) = y}$.

Example : For most systems that store users’ passwords, they hash users’ passwords to be more secure.

(2) Weak collision resistance :

For a hash function H, given an input x, it is hard to find another $\mathrm{x' \neq x}$ such that $\mathrm{H(x) = H(x')}$.

Example : When a user type his password to login, the system compares whether the hash value of the newly-typed password is the same as the stored hash value. The system allows the user to login if the two values are identical. This use the property of weak collision resistance.

(3) Strong collision resistance :

For a hash function H, it is hard to find two different inputs $\mathrm{x' \neq x}$ such that $\mathrm{H(x) = H(x')}$.

Example : When downloading file from the Internet, there can be a checksum to make sure that the file is complete. Supposed that the checksum comes from a hash function with strong collision resistance, then the property ensures that the file is exactly the correct file instead of a different file that happens to has the same checksum.

3. ElGamal Scheme

(a)

Alice uses the random value x as her private key and uses $c_1 \equiv g^x \pmod p$ as her public key. Then Alice sends her public key to Bob, whose private key is $sk_B = b$ and public key is $pk_B \equiv g^b \pmod p$. Then Alice can obtain a shared key $k_{AB} = (pk_B)^x$ and Bob can obtain $k_{AB} = (c_1)^b$ to build a symmetric key.

(b)

$$\mathrm{A(x)} = a_0 + a_1x + \cdots + a_{t-1}x^{t-1}$$

$$\mathrm{h(x)} = \sum_{i=1}^{t} L_i\,(x) \cdot y_i$$

$$L_i(x) = \prod_{j=1, j\neq i}^{t} \frac{x - x_j}{x_i - x_j}$$

$$L_i(x_i) = 1$$

$$L_i(x_j) = 0, j \in [1, t], j \neq i$$

$$L_i(0) = \prod_{j=1, j\neq i}^{t} \frac{-x_j}{x_i - x_j}$$

According to Shamir's Secret Sharing, the secret b is a_0, and the i-th user will receive their shared key $(i, A(i) \bmod p) = (x_i, y_i)$. $A(x) = h(x)$ because there are t different points (x_i, y_i) that satisfy $A(x) = h(x)$. Each user can compute their partial decryption result $(x_i, g^{y_i} \bmod p)$.

However, they cannot compute $A(0) = h(0)$ because they only know their own x_i. If t out of n users collaborate and thus share their (x_i, y_i) to each other, $h(0)$ can be computed by summing up $L_i(0) \cdot y_i$ and the secret $b = h(0)$ is recovered.

4. Simple Crypto

(1) Warmup

Send ‘2’ and send back what server says.

(2) Round 1 : Caesar cipher

I print out all the 26 possibilities and then pick the correct key.

(3) Round 2 : One-time pad

I get the answer by xor every two bytes in Alice’s words and Eve’s words.

(4) Round 3 : Bacon cipher

Create a binary string by turning lower letter to 0 and upper letter to 1 and ignoring non-letter word in cipher string. Then decode the binary string to plaintext by translating every 5 bits to upper letter. Value of the 5 bits means distance of the letter from ‘A’. At last, cut the ‘A’ tail to get the answer.

(5) Round 4 : Zig Zag cipher

Given a pair of cipher and plaintext, the key can be calculated by brute force search. The position of each letter in cipher will be determined according to the key. So by comparing the letter position between cipher and plaintext the key can be found. Then use the key to decrypt the second cipher to get the answer.

FLAG : CNS{ClA331CAl_CRYPT0_1S_3ASY_XDD}

5. RSA

(1) CNS{d1rect_cu6e_r00t}

The hint message says that the plaintext is too short. So I calculate the cubic root of the cipher and get the flag.

(2) CNS{Ch1nese_remaind3r_Theorem_1s_helpfu11111!!!!!}

Because the public key e is 3, collecting three ciphers and modulus is enough to recover the plaintext by Chinese Remainder Theorem.

(3) CNS{NOw_y0u_kNOw_Never_use_sma11_YEEEEEEEEE_again}

This time the modulus n is not given, but an encrypted number $c \equiv a^3 \pmod{n}$ is given where a is our input to the server. I choose the value of a such that a^3 will exceed 2^{1024} but not too much. Thus $c - a^3 = kn$ for small integer k. After dividing k, we can recover the modulus. Collect three different c and n and use CRT again the flag is decrypted.

(4) CNS{greaTEST_c0mm0n_divis0r_is_m0re_p0werfu1_than_y0u_think}

Let a_1 and a_2 be our two input numbers, c_1 and c_2 be the corresponding encrypted results, c be the encrypted flag. Because the public key e and private d always satisfy $ed \equiv 1 \pmod{\varphi(n)}$, $c_1^3 \equiv a_1 \pmod{n}$ and $c_2^3 \equiv a_2 \pmod{n}$. Then we can recover the modulus n by calculating the greatest common divisor of c_1^3 - a_1 and c_2^3 - a_2 and use the n to calculate c^3 mod n to get the flag.

6. Rainbow Table

(a)

$$
\begin{aligned}
\mathrm{E} &= \frac{|P|}{|P|} + \frac{|P|}{|P|-1} + \frac{|P|}{|P|-2} + \cdots + \frac{|P|}{|P| - \frac{|P|}{4} + 1} \\
&= |\mathrm{P}| \left(\frac{1}{|P|} + \frac{1}{|P|-1} + \frac{1}{|P|-2} + \cdots + \frac{1}{\frac{3}{4}|P| + 1} \right) \\
&= |\mathrm{P}| \left(\sum_{k=1}^{|P|} \frac{1}{k} - \sum_{m=1}^{\frac{3}{4}|P|} \frac{1}{m} \right) \\
&= |\mathrm{P}| \left(H_{|P|} - H_{\frac{3}{4}|P|} \right) \\
&\cong |\mathrm{P}| \left[(\ln|P| + \gamma) - \left(\ln\frac{3}{4}|P| + \gamma \right) \right] \\
&= |\mathrm{P}| * \ln\frac{4}{3} \\
&\cong 2.87 * 10^9
\end{aligned}
$$

(b)

For each combination of hash function and reduction function, there is $\frac{50000}{|P|}$ probability to collide. Once the collision happens, it generates the same chain and thus the same ending point. So the probability to generate a new ending point is $\left(1 - \frac{50000}{|P|}\right)^{10000} \cong 0.951.$

(c)

I construct a 10^7 chains rainbow table and the length of each chain is 150. For each password on the list, I generate 10 starting points by randomly adding tail in the range 0000 to 9999, like `mutate()` function in chal.py.

My reduction function $\mathrm{R_i}$ generates a possible password by (1) taking first half of the input hash value and add i, the iteration times of reduction function, and then module the number of passwords on the list as index to pick a password from the list (2) taking second half of the input hash value and then module 10000 as tail. Then concatenate the two part to form a possible password.

For each given hash value, run ($\mathrm{R_i} \bullet \mathrm{H_i}$) then $\mathrm{R_{chainlen-1}}$ to get a new endpoint E. Then find whether E is in the rainbow table. If it is, then E can be generated by the corresponding starting point, and the correct password is the password whose hash value is the same as the given hash value.

The result is about 130 correct passwords from 1000 hash values.

FLAG : CNS{sa1t_sh0uld_n3ver_bE_reu$ed!}

7. Padding Oracle

(1) Find the IV value :

Because the IV is constant, the first three cipher blocks c_1, c_2, c_3 are always the same. We can find IV by finding $\mathrm{D}(c_1)$ because $m_1 = \mathrm{D}(c_1) \oplus \mathrm{IV}$ where m_1 is the first block of plaintext, $\mathrm{D}(c_1)$ is value of c_1 after decryption. We can launch padding oracle attack to find $\mathrm{D}(c_1)$.

Every time we send two blocks message g_1 and c_1 to decryption server. In the beginning, g_1, the $\mathrm{D}(c_1)$ value we guess, is set all zero. Then we can launch padding oracle attack by modifying the value of g_1. After the attack, we know $\mathrm{D}(c_1)$ and the IV can be recovered.

The IV after decode is "your_everyday_iv".

(2) Generate cipher of chose plaintext :

Because we can use encryption in the server to encrypt a 16-bytes message, and IV and c_3 is the same, we can generate any cipher we want.

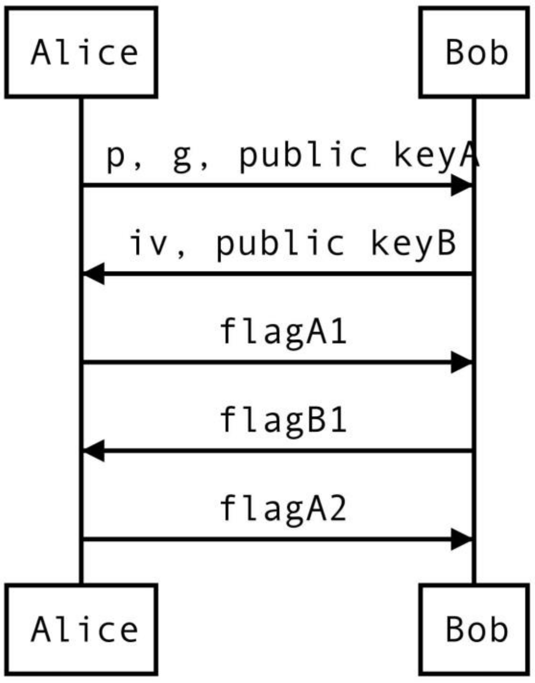
Let w_i be the i-th block of the wanted message, m'_i be the message to encrypt, c'_i be the encryption result of m'_i. We can set $m'_i = w_i \oplus c'_{i-1} \oplus c_3$ to generate a cipher. c'_0 is the IV. In this case the wanted message is "i_am_the_ta||act=printtheflag||your_message_is:". And the last block needs to pad to 16 bytes.

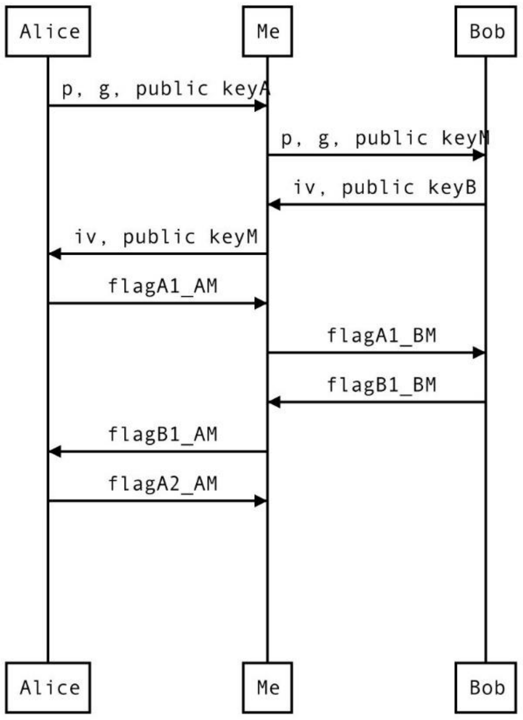
Then use the resultant cipher to update identity and print flag.

FLAG : CNS{(=^.x.^=)W15H Y0U H4V3 FUN}

8. Secret Exchange

(a)





The secret message : CNS{this_is_called_man_in_the_middle_attack.&_&}

The left graph shows how they exchange their secret. Alice uses her private key a to compute her public key g^a and send it to Bob. Bob uses his private key b to compute his public key g^b and send it and iv to Alice. Then Alice can generate the shared key by computing $(g^b)^a$ and use it as the key of AES. Bob can compute the shared key $(g^a)^b$ to decrypt the message Alice send.

The right graph shows how I attack their secret exchange using Man in the Middle attack. First I intercept what Alice sends to Bob to get her public key. Then replace her public key with mine and send to Bob. And do the same thing to what Bob send to me. By doing this, both Alice and Bob think that they connect to each other because they decrypt the messages correctly. However, they don't know that they share the key with me instead of Alice/Bob. Thus, for each message from Alice to Bob, I can intercept it and decrypt it with keyAM, the key shared by Alice and me, and then encrypt it using keyBM, and vice versa.

(b)

The key k that used to generate the signature (r, s) for two different messages is the same key. In the scenario, $r_1 = r_2$ and k can be computed. But the modulus $q = (p - 1)$ is not a prime and causes some trouble when I am trying find multiplicative inverse of $(s_1 - s_2)$. If the multiplicative inverse exists, k can be recovered by computing $(H(m_1) - H(m_2)) * (s_1 - s_2)^{-1} \pmod q$. Then I can pretend Bob and launch MitM again to get the flag.