CNS HW1

B03902082 資工四 江懿友

1. CIA

Confidentiality: 保密性，防止訊息被第三者竊聽。

例：使用 https 加密網站通訊內容。

Integrity: 完整性，防止訊息被第三者竄改。

例：傳送的檔案附上 MAC 來偵測訊息有沒有被竄改或是掉封包。

Availability: 可用性，確保使用者能使用服務。

例：網路服務使用防火牆阻擋惡意連線以防止 DDoS 攻擊，確保正常用戶的服務品質。

2. Hash Function

One-wayness: Given y, hard to find x s.t. y = H(x). 難以從某個 hash value 推算出能算出這個 hash value 的 input。

例：密碼資料庫只需要存密碼的 hash value，因為就算資料庫內容外洩，攻擊者也難以只從 hash value 推算出使用者的密碼或是能 hash 出相同的 value 的字串。

Weak collision resistance: Given x, hard to find x' =/= x s.t. H(x') = H(x). 難以根據某個明文 x 推算出另一個明文 x'，滿足 x 和 x' 的 hash value 相同。

例：Message digest 利用 weak-collision resistance 防止訊息被竄改。

Strong collision resistance: Hard to find x and x' with x =/= x' s.t. H(x') = H(X). 難以找出任意兩個明文，滿足兩個的 hash value 相同。

例：利用有 strong collision resistance 的 hash function，資料庫儲存資料時可以用資料的 hash value 當作 key (id)，如此就可以更有效率的查詢資料，因為不需要比對完整的資料內容而只需要比對 hash value。

3. ElGamal Threshold Decryption

setup

large prime: p, q

generator: g
secret key: $sk = b$
public key: $pk \equiv g^b \ (mod\, p)$
Generate a polynomial $F(x)$ of degree $t-1$ $s.t.$ $F(0) \equiv b \ (mod\, q)$.
For i = 1 … n, user i receives the partial secret $(x_i, y_i) = (i,\ F(i)\ mod\, q)$.

encrypt

Same as the original encryption method.

decrypt

plaintext: $m \equiv c_2 c_1^{-b} \ (mod\, p)$
where in order to obtain b, at least t partial secrets have to be gathered, then Lagrange interpolation can be used to reconstruct the polynomial $F(x)$ and secret key $b \equiv F(0) \ (mod\, q)$.

4. How2Crypto

FLAG: BALSN{You_Are_Crypto_Expert!!!^_^}

BALSN{You_Are_Crypto_Expert!!!^_^}

Round 1

This is a substitution cipher that works like 'a' -> '01', 'b' -> '02' with whitespace character unchanged.
In order to decipher just do the reverse substitution like '01' -> 'a', … , '26' -> 'z'.

Round 2

This is a Caesar cipher with whitespace character unchanged. Use c1 and m1 to calculate the displacement, and m2 can be deciphered from c2 easily.

Round 3

This is also a Caesar cipher, but this time the displacement is unknown. However we know that the plain text must be some meaningful English words, so the displacement can be obtained by looking at all the 25 possibilities and choose the displacement value that deciphers to meaningful text.

Round 4

This is a substitution cipher that each English letter is mapped to another English letter. Part of the mapping can be discover from c1 and m1. We can try to decipher c2 with this partial mapping. For the rest of the letters with unknown mapping, we can use human knowledge to solve it, as the deciphered message must be meaningful words. For example if c2 = 'abcde' and we know the partial mapping {a -> w, b -> o, d -> d, e -> s}, then we can infer that 'c' maps to 'r' because m2 = 'words' makes a valid English word.

Round 5

This is a transposition cipher that works like:
$c[i] = m[i \times k \ mod \ N] \forall i = 0...N-1 \ where \ N = length(m)$
Deciphering works like:
$m[i] = c[i \times k^{-1} mod \ N] \forall i = 0...N-1 \ where \ N = length(m)$

Round 6

This is a rail fence cipher (https://en.wikipedia.org/wiki/Rail_fence_cipher).
The "width" of the rail fence cipher can be determined from c1 and m1 by exhaustive search.

After completing every round, I get a base64 encoded png image. The image shows the flag.

5. Mersenne RSA

FLAG: BALSN{if_N_is_factorized_you_get_the_private_key}

The Mersenne primes used to generate N can be obtained by exhaustive search. After obtaining the two primes p and q, the secret key d can be calculated with the equation: $d \equiv e^{-1} \ (mod \ \Phi(N)) \ where \ \Phi(N) = (p-1) \times (q-1)$.

6. OTP

FLAG: BALSN{NeVer_U5e_0ne_7ime_PAd_7wIcE}

Because we know that the plaintext must be consists of printable characters and meaningful English sentences. I can infer the shortest possible key length to be 13 by exhaustive search. Then I can deduce the key byte by byte easily. The key is [169, 109, 201, 15, 92, 226, 255, 144, 212, 123, 223, 119, 148], and the plaintext is "Luigi wins by doing nothing" (http://knowyourmeme.com/memes/luigi-wins-by-doing-absolutely-nothing). The flag can be found embedded in the plaintext.

7. Double AES

FLAG: BALSN{so_2DES_is_not_used_today}

key0, key1 = 6298659, 4272711
I am no ACM champion but I know how to do the meet-in-the-middle attack. First I enumerate all possible values of key0 and encrypt the known-plaintext, and record all the (key, cipher_text) pair in a look-up table. Then I enumerate all possible values of key1 and decrypt the known-ciphertext, and look up the decrypted text in the look-up table. This way I can find out key0, key1 efficiently and decrypt the flag.

8. Time Machine

FLAG: BALSN{P0W_1s_4_w4st3_0f_t1m3_4nd_3n3rgy}

This problem requires me to find a collision of sha1. I use the "SHAttered" attack to generate colliding input pairs. The SHAttered attack provides two prefixes P1 and P2 with the property sha1(P1 | S) = sha1(P2 | S) where S can be an arbitrary suffix.
The problem first requires me to find X such that the lower 24 bits of sha1(X) equals some given bit sequence. This can be found by exhaustive searching all possible S using a counter, and test if sha1(P1 | S) matches the requirement. After I find a valid S, I construct X = (P1 | S) and Y = (P2 | S).

9. Future Oracle

FLAG: BALSN{Wh4t_1f_u_cou1d_s33_th3_futur3}

This problem is vulnerable to "length extension attack".
The login system itself works like a oracle that answers
"sha256(password || ID || Nc || login)" where ID and Nc is user input.
So when the login system asks for "sha256(ID || Ns || action) || sha256(password || ID || Ns || action)", I can ask the oracle the answer of sha256(password || admin || Ns || login), then I use length extension attack on sha256 to obtain sha256(password || admin || Ns || login[padding bytes] || printflag). By exhaustive search I can find that the length of password is 19. Thus I can forge the MAC without knowing the password and forge a "printflag" request.

10. Digital Saving Account

FLAG: BALSN{s3nd_m3_s0m3_b1tc01n_p13as3}

The login system is vulnerable to "cut-and-paste attack", because the request is encrypted by ECB mode.
By carefully choosing the length of name and pwd, I can align text to block boundaries and obtain:

A = encrypt(key, 'login=A&role=guest&pwd=B12345678')
B = encrypt(key, '&pwd=12345678901')
C = encrypt(key, 'login=AAAA&role=')
D = encrypt(key, 'admin' + padding)
and
A + B + C + D = encrypt(key,
'login=A&role=guest&pwd=B12345678&pwd=12345678901login=AAAA&role=admin' + padding)
Thus when I login with the token A+B+C+D, I can login with name=A, pwd=B12345678, and role=[guest, admin] and gain admin permission.

Then I need to forge a DSA (Digital Signature Algorithm). I know the public key (p, q, g, y) and two pairs of signature (m1, (r1, s1)), (m2, (r2, s2)).
The vulnerability here is that the nonce k used to sign m1 and m2 is reused,
thus r1 = r2 = $(g^k mod\, p)\; mod\, q$.
Thus by the equations of DSA (x is private key):
$s_1 \equiv k^{-1}(H(m_1) + xr_1)\ (mod\, q)$
$s_2 \equiv k^{-1}(H(m_2) + xr_2)\ (mod\, q)$
with r1 = r2, I can infer
$s_1 - s_2 \equiv k^{-1}(H(m_1) - H(m_2))\ (mod\, q)$
Since s1, s2, H(m1), H(m2) are all known, I can calculate k (and k^-1) by extended Euclidean algorithm, and I can calculate $xr_1 \equiv (s_1 k - H(m_1))\ (mod\, q)$.
At last I can forge "FLAG" 's signature (r, s) = $(r_1, k^{-1}(H("FLAG") + xr_1)\ mod\, q)$ without knowing the private key x.