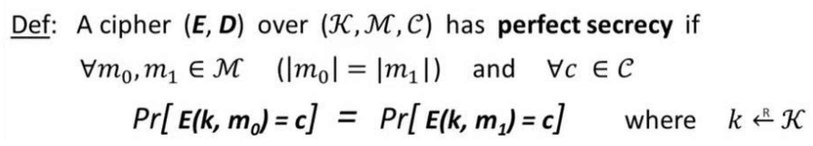
Cryptography Engineering Midterm 310555004 呂佳柔

**Problem 1**

1. That us choose two encryption key “a1 and a2” in $Z_p^*$, that has the following property: m*a1 mod p = c = m*a2 mod p
Since, every element b in $Z_p$* has an inverse $b^{-1}$ $Z_p^*$ such that b*$b^{-1}$ = 1 mod p
And a1 and a2   $Z_p^*$. Hence, a1 must equal to a2, which also means that:
P(E(k1, m) = c) = P(E(k2, m) = c)

2. The definition of perfect secrecy is:



Def: A cipher $(\boldsymbol{E}, \boldsymbol{D})$ over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ has **perfect secrecy** if
$\forall m_0, m_1 \in \mathcal{M}$ $(|m_0| = |m_1|)$ and $\forall c \in \mathcal{C}$
$$Pr[\, \boldsymbol{E(k, m_0) = c} \,] = Pr[\, \boldsymbol{E(k, m_1) = c} \,] \quad \text{where} \quad k \xleftarrow{R} \mathcal{K}$$

For OTP, E(k, m) = c = k XOR m => c XOR m = k XOR m XOR m = k , k=1.
Hence, it is one-to-one. Thus, we can prove that OTP is perfect secrecy.

3. One-time pad’s key is generated by random number generator, and the key will only be used once. So, there is no statistic relation between plaintext and ciphertext.

4. The threshold for a perfectly secure system is that a computationally unbounded adversary can’t conclude anything about the plaintext from the ciphertext. With a public-key system, the attacker can try to encrypt messages with the real public key. What the attacker can do is to try all one-bit messages, then all two-bit messages, then all three-bit messages, etc., until the expected matching ciphertext come out.

**Problem 2**

1. Starting with the relation $x_2$-$x_1$ = a($x_1$-$x_0$)(mod p), we immediately get (assuming for the moment that $x_1$-$x_0$ and m are coprime) a = ($x_2$-$x_1$)($x_1$-$x_0$) mod p where the division is mod m (using the extended Euclidean algorithm). The increment b is given by
b = ($x_1$-a$x_0$) mod p. So we found the defining formula and may predict the complete sequence.

2. Is not secure to use congruential generator as the keystream generator for a streamer cipher.

3. Only need two successive output sequence to predict the complete sequence.

4. By Problem 2-1, we prove that if an attacker knows p, attacker only need to have 3 successive outputs to predict the complete sequence.

**Problem 3**

Prove:

Prob(G(t) = $x_2$(t)) = Prob($x_1$(t)=0) + Prob($x_1$(t) = 1) * Prob($x_3$(t) = $x_2$(t)) = 1/2 + 1/4 = 3/4

**Problem 4**

Password design rule: use password length to achieve zero knowledge protocol. To logging into the system, users will need to set two passwords with different lengths (ex: 123 & 4567). The rule for entering a password when logging in to the system is <frist password> + <[0-9]{n}> + <second password > (ex:1233105550044567).

The system only needs to compare whether the prefix matches the first password and whether the suffix matches the second password. Since the adversary does not know the length of the correct password, it cannot be cracked in a limited number of times, which can achieve the effect of defense.

**Problem 5**

Using Trivium algorithm and cipher makes use of 80-bit key and 80-bit initialisation vector (IV); its secret state has 288 bits, consisting of 3 interconnected non-linear feedback shift registers of length 93, 84 and 111 bits, respectively. The cipher operation consists of two phases: the key and IV set-up and the keystream generation. Initialisation is very similar to keystream generation and requires 1152 steps of the clocking procedure of Trivium. The keystream is generated by repeatedly clocking the cipher, where in each clock cycle three state bits are updated using a non-linear feedback function, and one bit of keystream is produced and output. The cipher specification states that 264 keystream bits can be generated from each key/IV pair.

**Problem 6**

1. For an Affine Cipher mx + n (mod 26), we must have gcd(26, m) = 1, and we always take $1 \le n \le 26$. $\phi(26) = \phi(2*13) = 12$, hence we have 12*26=312 possible keys.

2. $P(26,26) = 26! \approx 2^{88}$

3. 26

**Problem 7**

(a) An attacker who knows the correct message-tag (i.e. CBC-MAC) pairs for two messages (m, t) and (m', t') can generate a third message m'' whose CBC-MAC will also be t'. This is simply done by XORing the first block of m' with t and concatenating m with this modified m'. i.e., by making $m'' = m \,||\, [(m_1' \oplus t) \,||\, m_2 \,||\, \ldots \,||\, mx']$. When computing the MAC for the message m'', it follows that we compute the MAC for m in the usual manner as t, but when value is chained forwards to the stage computing $MAC_k(m_1' \oplus t)$ we will perform an exclusive OR operation with the value derived for the MAC of the first message. The presence of that tag in the new message means it will cancel, leaving no contribution to the MAC from the blocks of plain text in the first messagem: $MAC_k(m_1' \oplus t \oplus t) = MAC_k(m_1')$ and thus the tag for m'' is t'.

(b) We define $f_a^* (x_1 \ldots x_m) = f_a^{(m+1)}(x_1 \ldots x_m)$, where m is the length of message. We show that $f^*$ is not a secure MAC. Take arbitrary l-bit words b, b', and c, where $b \neq b'$. It is easy to check that given ① $t_b = f^*(b)$, ② $t_{b'} = f^*(b')$, ③ $t_{b|c} = f^*(b \,||\, 1 \,||\, c)$
the adversary has in hand $t_{b|c} = f^*(b' \,||\, 1 \,||\, t_b \oplus t_{b'} \oplus c)$ — the authentication tag of a string someone has not asked about before since this is precisely $t_{b|c.}$

**Problem 8**

Only partially done, but i tried to do it. ☹