**2.2 network essential**

**2.2 a. Let –X be the additive inverse of X. That is –X**

**€**

**+ X = 0. Then:**

**P = (C**

**€**

**+ –K1) ⊕ K0 b. First, calculate –C'. Then –C' = (P' ⊕ K0)**

**€**

**+ (–K1). We then have:**

**C**

**€**

**+ –C' = (P ⊕ K0)**

**€**

**+ (P' ⊕ K0) However, the operations**

**€**

**+ and ⊕ are not associative or distributive with one another, so it is not possible to solve this equation for K0.**

**ALICE AND BOB**

**6.12 a.** By taking the first 80 bits of *v* || *c*, we obtain the initialization vector, *v*. Since *v*, *c*, *k* are known, the message can be recovered (i.e., decrypted) by computing RC4(*v* || *k*) ⊕ *c*.

**b.** If the adversary observes that *vi* = *vj* for distinct *i*, *j* then he/she knows that the same key stream was used to encrypt both *mi* and *mj*. In this case, the messages *mi* and *mj* may be vulnerable to the type of cryptanalysis carried out in part (a).

**c.** Since the key is fixed, the key stream varies with the choice of the 80-bit *v*, which is selected randomly. Thus, after approximately  messages are sent, we expect the same *v*, and hence the same key stream, to be used more than once.

**d.** The key *k* should be changed sometime before 240 messages are sent.

<http://code.activestate.com/recipes/266586-simple-xor-keyword-encryption/>

<https://pythonprogramming.net/encryption-and-decryption-in-python-code-example-with-explanation/>

https://code.sololearn.com/cK267fizDS8H/#py