Week 5 Homework due on Friday Oct 12, 22:00 Hour

Group 5

1) What is the security level? You need to calculate it in bits

Wong Ann Yi (1004000)
Liu Bowen (1004028)
Tan Chin Leong Leonard (1004041)

2) How many keys each user will have?

5) Incomplete code

Exercise 1

Estimate the security level of your credit card (assume that an adversary knows your name).

Answers;
Even if an adversary knows my name, there are still some security in the card which will prevent the adversary to steal the money from the card.

The credit number has 16 digits and an adversary will need to use brute force to find the number. He will need to compute 10^16 combination of numbers to guess the right credit card number.

In addition, there is 3 digit CVC security code at the back of the card. This will require the adversary to use 10^3 combination to guess the numbers.

Then there is an expiry date of the card which an adversary would need to guess the month (out of 12) and the year which a credit would have a validity of up to 3 years from the date of issue.

Last but not least, all credit cards bear the unique signature of the owner and an adversary will not be able to find and copy the signature. Some credit cards may have an additional PIN access which the adversary will again need to use brute force to guess.

Exercise 2

Consider a group of 30 users who wish to establish pair-wise secure communications using symmetric-key cryptography. How many keys each user will have? How many keys in total are needed for 30 users?

Answers:

In symmetric-key cryptography, the pair of users will use the same secret key K. Based on Kerckhoffs' Principle, the security of this encryption depends on the secrecy of the key and not on the algorithm.

We can use the combination method to solve it: nCr = n! / r! (n - r)! where n is the number of users and r is the number of people communicating at the same time. Therefore, the solution is ${}_{30}C_2 = 30! / 2! (30 - 2)! = 435$

Exercise 3

Suppose a chosen-cyphertext attacker cannot recover the secret decryption key for an encryption scheme. Does this mean the encryption scheme is secure? Why?
Answers:

In a chosen-cyphertext attack, the attacker will try to gather the information by obtaining the decryptions of the chosen cyphertexts. From the information, the attacker can attempt to recover the secret keys used for decryption. If the attacker cannot recover the secret key, he will have failed and he will use the other less powerful methods such as ciphertext-only, know-plaintext, chosen-plaintext attack methods to discover the key. However, these other methods are less powerful compared to the chosen-ciphertext attack. This does not mean that the encryption is secure, even without the secret key, an attacker may decrypt a specific other message. It will reveal some partial information about the message. For example, if an attacker has 10 chosen plaintexts and their corresponding ciphertexts, it is possible for an attacker to learn the least significant of the 11^{th} plaintext. This bit of information is valuable for the attackers to form new clues.

Exercise 4

Use the aes_enc() function (see below) to encrypt the BLK.BMP file from http://www.fileformat.info/format/bmp/sample/index.htm. Do you see any patterns in the ciphertext?

```
from Crypto.Cipher import AES

def aes_enc(key, msg):
        cipher = AES.new(key, AES.MODE_ECB)
        return cipher.encrypt(msg)
```

Answers:

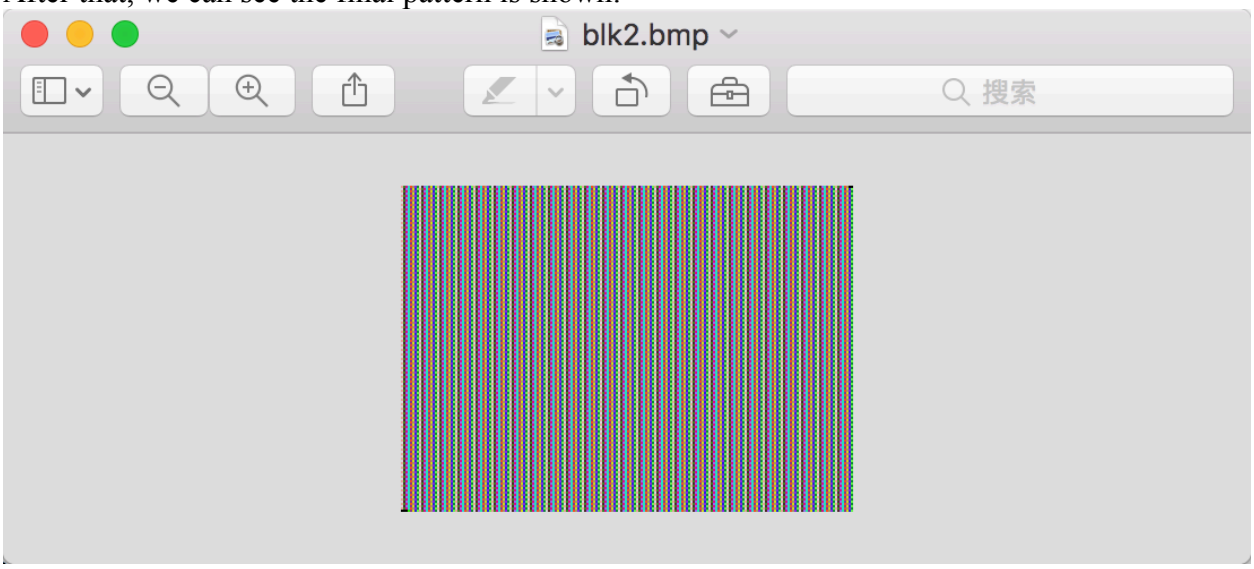
We changed the algorithm as follows:

```
from Crypto.Cipher import AES
key = "aaaabbbbccccdddd"
cipher = AES.new(key, AES.MODE_ECB)
with open("BLK.BMP", "rb") as f:
  clear = f.read()

print len(clear)
print len(clear)%16
# -6 is generated by len(clear)%16
clear_trimmed = clear[64:-6]
ciphertext = cipher.encrypt(clear_trimmed)
```

```
# -6 is generated by len(clear)%16
ciphertext = clear[0:64] + ciphertext + clear[-6:]
with open("blk2.bmp", "w") as f:
  f.write(ciphertext)
```

After that, we can see the final pattern is shown:



Exercise 5

Use aes_enc() to implement the encryption function of AES-CBC. Verify your implementation against the official test vectors (https://tools.ietf.org/html/ rfc3602#section-4, Case#4).

Answers:

We need to decode the key, IV to ‘hex’ format. Also, we divide msg into 4 parts and decode() to ‘hex’ and then append them. Then, AES.new(key, AES.MODE_CBC, IV) should pass IV as third arguments. And finally, use cipher.encrypt(whole_msg) to generate ciphertext.

```
from Crypto.Cipher import AES

key = ('56e47a38c5598974bc46903dba290349').decode('hex')
IV = ('8ce82eefbea0da3c44699ed7db51b7d9').decode('hex')
plaintext1 = ('a0a1a2a3a4a5a6a7a8a9aaabacadaeaf').decode('hex')
plaintext2 = ('b0b1b2b3b4b5b6b7b8b9babbbcbdbebf').decode('hex')
plaintext3 = ('c0c1c2c3c4c5c6c7c8c9cacbcccdcecf').decode('hex')
plaintext4 = ('d0d1d2d3d4d5d6d7d8d9dadbdcdddedf').decode('hex')
cipher = AES.new(key, AES.MODE_CBC, IV)
ciphertext = cipher.encrypt(plaintext1 + plaintext2 + plaintext3 + plaintext4)
```