CMPSC443 Lab Crypto 1

Submission

You need to write a **report** which includes:

1. A description of your findings for tasks.
2. Answer all the questions given in the tasks.
3. The python scripts you write to accomplish the task.

Links:

- Pycrypto API docs: https://www.dlitz.net/software/pycrypto/api/current/
- http://eli.thegreenplace.net/2010/06/25/aes-encryption-of-files-in-python-with-pycrypto/
- http://www.dreamincode.net/forums/topic/240431-how-to-work-with-bytes-in-python/
- http://pymotw.com/2/struct/

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

Setup - installing pycrypto 2.6.1 **[Kali]**

1. Download the package
https://pypi.python.org/packages/source/p/pycrypto/pycrypto-2.6.1.tar.gz
2. unpack and install
    - 2.1. cd to the download directory
    - 2.2. unpack it using the tar command: `tar -xzvf pycrypto-2.6.1.tar.gz`
    - 2.3. cd to the unpacked dir `cd pycrypto-2.6.1`
    - 2.4. build and install the python library
        - 2.4.1. `sudo python setup.py build`
        - 2.4.2. `sudo python setup.py install`
    - 2.5. test it - take a while to finish
        - 2.5.1. `sudo python setup.py test`

Task 1 - Encryption using different ciphers and modes

In this task, we will play with various encryption algorithms and modes.

1. The first example, create a python script using the following code, and run it

```
# Example 1: test_aes.py
# run the script: python test_aes.py

from Crypto.Cipher import AES

key = 'mysecretpassword'
plaintext = 'Secret Message A'
```

```
encobj = AES.new(key, AES.MODE_ECB)
ciphertext = encobj.encrypt(plaintext)

# Resulting ciphertext in hex
print ciphertext.encode('hex')
```

2. Go to http://pythonhosted.org//pycrypto/ the official library document.
    a. Modify the above python script and try the following encryption algorithms:
        i. Blowfish
        ii. Des
        iii. Des3
        iv. ARC4 (i.e., RC4)
        v. XOR

Task 2 - Encryption Mode – ECB vs. CBC

The file pic `original.bmp` contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please write a python script to encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, for the .bmp file, the first **54** bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. You can either do this manually after encryption or program it as part of your python script.
2. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

In the case of AES, the IV has fixed length of **16** bytes. You may choose the values of the encryption key and the IV.

You may use the following code snippet to construct your program

```
def encrypt_bmp(key, iv, mode, in_filename, out_filename=None, chunksize=1024):
    if not out_filename:
        out_filename = in_filename + '.enc'

    if mode == AES.MODE_ECB:
        # no iv is needed for ECB!
        encryptor = AES.new(key, AES.MODE_ECB)
    elif mode == AES.MODE_CBC:
        encryptor = AES.new(key, AES.MODE_CBC, iv)
    else:
        print "invalid mode, program stopped"
        return

    with open(in_filename, 'rb') as infile:
        # extract bmp header
        header = infile.read(54)
        # open output file
        with open(out_filename, 'wb') as outfile:
            outfile.write(header)
            while True:
                chunk = infile.read(chunksize)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += ' ' * (16 - len(chunk) % 16) #pad the last block with
spaces

                outfile.write(encryptor.encrypt(chunk))
```

Task 3 - Corrupted Ciphertext

To understand the properties of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 64 bytes long.
2. Encrypt the file using the AES-128 cipher. (use a python script)
3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. (you can manually do this)
4. Decrypt the corrupted file (encrypted) using the correct key and IV.

Please answer the following questions:

1. How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC respectively?
2. Please explain why.

Task 4 - Find the key

You are given a plaintext and a ciphertext, and you know that aes-128-cbc is used to generate the ciphertext from the plaintext, and you also know that the numbers in the IV are all zeros (not the ASCII character ‘0’). Another clue that you have learned is that the key used to encrypt the plaintext is a word from a typical English dictionary, words.txt. Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x20) are appended to the end of the word to form a key of 128 bits. Your goal is to write a program to find out this key. Some key info:

Plaintext (total 21 characters):
```
This is a top secret.
```
Ciphertext (in **hex** format):
```
3f814d00c3f1047f1dfa879115970472472a17eabdd9ba4fcd667743e1e03674
```

You should be aware that AES is a block cipher that operates on 128-bit blocks. You need to pad with last block using the PKCS5 padding.
Your job is to write a python script to find the **encryption key**!