Homework 3 – Secret-Key Encryption

1 Overview

The learning objective of this homework is for students to get familiar with the concepts and principles in the secret-key encryption. To achieve the objective, the students will do both theory (paper-and-pencil problems) and practice (programming and lab). After finishing the homework, students should be able to understand better and gain a first-hand experience on encryption algorithms and encryption modes. Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

2 Paper-and-Pencil Problem [46 pts +7 bonus pts]

There are a total of 9 paper-and-pencil problems (include one bonus question) from the KPS textbook ($X.Y.$ means number Y homework in Chapter X). Please make sure to read KPS textbook closely before answering these questions.

- 2.2. (3 pts)
- 2.3. (4 pts)
- 2.4. (4 pts)
- 3.2. (7 pts)
- 3.3. (7 pts)
- 3.5. (7 pts)
- 4.2. (7 pts)
- 4.4. (Bonus question, 7 pts)
- 4.6. (7 pts)

3 Lab and Programming Tasks [54 pts]

There are a total of 5 lab/programming tasks.

3.1 Lab Environment

Installing OpenSSL. In this lab, we will use `openssl` commands and libraries.

If your VM does not have it, you should first install `openssl` package using the following command:

```
% sudo apt-get install openssl
```

It should be noted that the above command only install the `openssl` binaries. If you want to use `openssl` libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. You can use the following command to do this:

```
  % apt-get source openssl

Untar the tar ball, and run the following commands.
You should read the INSTALL file first:

  % ./config
  % make
  % make test
  % sudo make install
```

Installing `GHex`**.** In this lab, we need to be able to view and modify files of binary format.`GHex` is a hex editor for GNOME, it allows the user to load data from any file, view and edit it in either hex or ascii.

You can use the following command to install it (it should be noted that the name of the command is called `/usr/bin/ghex2` at the time of writing). You can call `ghex2` to use it.

```
% sudo apt-get install ghex.
```

Fixing Apt-get You may encounter problems with apt-get, which is because the resource repository is outdated. You can fix it as follows:

```
1. % sudo vi /etc/apt/sources.list
(and please replace all "http://us.archive.ubuntu.com/ubuntu" links to
"http://old-releases.ubuntu.com/ubuntu/")

2. % sudo apt-get update
```

3.2 Task 1: Encryption using different ciphers and modes [6 pts]

In this task, we will play with various encryption algorithms and modes. You can use the following `openssl enc` command to encrypt/decrypt a file. To see the manuals, you can type `man openssl` and `man enc`.

```
  % openssl enc ciphertype -e  -in plain.txt -out cipher.bin \
                -K  00112233445566778889aabbccddeeff \
                -iv 0102030405060708
```

Please replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-aes-128-cfb`, `-des`, etc. In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing "`man enc`". We include some common options for the `openssl enc` command in the following:

```
  -in <file>     input file
  -out <file>    output file
  -e             encrypt
  -d             decrypt
  -K/-iv         key/iv in hex is the next argument
  -[pP]          print the iv/key (then exit if -P)
```

3.3 Task 2: Encryption Mode – ECB vs. CBC [8 pts]

The file `pic_original.bmp` (available at `http://courses.cse.tamu.edu/guofei/csce465/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 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 0x36 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 use the `ghex` tool to directly modify binary files.

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.

3.4 Task 3: Encryption Mode – Corrupted Cipher Text [10 pts]

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.

3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using `ghex`.

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, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. (2) Please explain why. (3) What are the implications of these differences?

3.5 Task 4: Programming using the Crypto Library [15 pts]

So far, we have learned how to use the tools provided by `openssl` to encrypt and decrypt messages. In this task, we will learn how to use `openssl`'s crypto library to encrypt/descrypt messages in programs.

OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library provides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface. A sample code is given in `http://www.openssl.org/docs/crypto/EVP_EncryptInit.html`. Please get yourself familiar with this program, and then do the following exercise.

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 this plaintext is an English word shorter than 16 characters; the word that can be found from a typical English dictionary. 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. You

can download a English word list from the Internet. We have also provided one here (`http://courses.cse.tamu.edu/guofei/csce465/words.txt`). The plaintext and ciphertext is in the following:

```
Plaintext (total 21 characters): This is a top secret.
Ciphertext (in hex format): 8d20e5056a8d24d0462ce74e4904c1b5
                            13e10d1df4a2ef2ad4540fae1ca0aaf9
```

Note 1: If you choose to store the plaintex message in a file, and feed the file to your program, you need to check whether the file length is 21. Some editors may add a special character to the end of the file. If that happens, you can use the `ghex` tool to remove the special character.

Note 2: In this task, you are supposed to write your own program to invoke the crypto library. No credit will be given if you simply use the `openssl` commands to do this task.

Note 3: To compile your code, you may need to include the header files in `openssl`, and link to `openssl` libraries. To do that, you need to tell your compiler where those files are. In your `Makefile`, you may want to specify the following:

```
INC=/usr/local/ssl/include/
LIB=/usr/local/ssl/lib/

all:
       gcc -I$(INC) -L$(LIB) -o enc yourcode.c -lcrypto
```

3.6 Task 5: Write your own DES encryption code (one round) [15 pts]

Implement a single round of DES in C(++) or Java. Test with the round keys 0x123456789abc for DES. Measure the execution speed of the implementation (using the time service). (You may need to execute the function more than once to get meaningful results.) The S boxes for DES are provided in the KPS textbook or `http://courses.cse.tamu.edu/guofei/csce465/sbox.htm` for your convenience.

4 Submission

You need to submit a detailed homework report to describe what you have done and what you have observed (also including screen shots if possible); you also need to provide explanation to the observations that are interesting or surprising. In your report, you need to answer all the questions listed in this homework. You need to send all required programs too (together with the report).