Assessed Coursework 1 — Secret-Key Encryption

1 Overview

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

2 Lab Environment

The software needed for this lab is installed on all machines in the Linux Lab (EM 2.50). If you want to use your own laptop, you will need to install several packages. This section describes which packages you need and how to test the installations. On the lab machines, first download the file f21cn.sh and then type `source f21cn.sh` to get access to all tools needed for this lab.

Testing OpenSSL. In this lab, we will use `openssl` commands and libraries, which are installed on all Linux lab machines. To test the command line shell do

```
   %  openssl version
OpenSSL 1.0.1e-fips 11 Feb 2013
```

To check that the libraries for `openssl` are installed do the following

```
  % rpm -qa | fgrep openssl
openssl-1.0.1e-16.el6_5.15.x86_64
openssl098e-0.9.8e-18.el6_5.2.x86_64
openssl-devel-1.0.1e-16.el6_5.15.x86_64
```

You should see one package `openssl-devel`, which contains all necessary libraries.

In Task 3.2 you will have to edit a hex file (a picture). Below are three different alternatives of doing this. Pick the **one** most suitable for you.

Using Emacs to edit hex-files: To edit hex-files, you can use the installed version of the `emacs` editor. Start it like this:

```
% emacs
```

In the editor type `M-x hexl-find-file` and enter the file name. You will see a screen with the hex-contents of the file. Use your cursor to move to a position that you want to edit. To insert a hex value do `C-M-x <value>` or use the `Hexl menu`. Use `C-x C-s` to save the file and `C-x C-c` to exit the editor.

Using `ghex2` to edit hex-files: An easy-to-use graphical hex editor, `ghex2`, is available on all Linux machines running the gnome desktop. You can start it by typing `ghex2` from the command-line or selecting it from the graphics section of the system menus.

Using `shed` to edit hex-files: A simple hex editor, `shed`, is available on the Linux machines, if you have run `source f21cn.sh`. Call it with the filename to edit as argument, e.g. `shed pic_cipher.bmp`. Use your cursor to move to a file position and the SPACE key insert a value. Prefix with a number to edit that many bytes. Documentation is available via a man-page, i.e. `man shed`. The full path is `/usr/bin/shed`.

GNU/Linux coreutils. Alternatively, you can also use the `coreutils` on any GNU/Linux system. This way you can avoid any manual editing of the files. Type `info coreutils` to get information about the entire toolset, browse the index and look for the tools for outputting parts of a file.

3 Lab Tasks

3.1 Task 1: Encryption using different ciphers and modes

In this task, we will use 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`, `-bf-cbc`, 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)
```

3.2 Task 2: Encryption Mode — ECB vs. CBC

The file pic_original.bmp contains a simple picture. You can download it using `wget` or using your web browser. We would like to encrypt this picture, so that people without the encryption key cannot view its contents. 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. On a Linux machine, just type `display pic_original.bmp` to view 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 use the `shed` or the `ghex2` hex-editor, or any of the other tools described above, to directly modify binary files. If you are fluent in shell scripting and use `coreutils`, you can automate the entire process.

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.

3.3 Task 3: Encryption Mode — Corrupted Cipher Text

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 `shed` or `ghex2`, modifying the 30th byte in this file.
4. Decrypt the corrupted file (encrypted) using the correct key and initialisation vector (IV).

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.

Please explain your observations.

3.4 Task 4: Reflections on ciphers and modes (F21CN/Level 11)

In this task, you should interpret the results of Task 2 and 3 in more detail, and more generally derive conclusions on the quality of the ciphers and modes tested.

Specifically, answer the following questions, related to Task 2:

- Which of the modes hides the highest degree of information?
- Which of the modes would be best suited to encrypt large volumes of data?
- Which of the modes would be best suited for encrypting streams of data?

Please answer the following questions related to Task 3:

- Based on your observations in Task 3, what are the implications on the usage for these modes in case of faults during data transmission?
- Describe a scenario that exemplifies the advantages and disadvantages of the modes studied in Task 3.

3.5 Task 5: Checksums

As preparation for Task 6 download the following files:

- English word list SHA1 checksum: `b3470280f84575a3db3ec3a6b9df2681ee0f5a18`
- Plain text file SHA1 checksum: `0b6f3556e8773a3e7c0ed31c634b9fd2a108adcc`
- Cipher text file SHA1 checksum: `92ce63d9f3495ca005237eb6cca47302b74c574f`

Use openssl and SHA checksums to answer the following questions (justify each answer): (1) Have the the downloaded files been tampered with? Give the openssl commands and responses to do this check. (2) Which cryptographic concept is covered by this check? (3) Does this check guarantee that the files are from the alleged originator?

3.6 Task 6: Known-plaintext attack

Now, consider the plaintext and the ciphertext that you downloaded in Task 5. You know that `aes-128-cbc` was used to generate the ciphertext from the plaintext. No salting was used during encryption. You also know that the numbers in the initialisation vector (IV) are all zeros (not the ASCII character '0'). Another clue 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. Your goal is to write a program to find out this key, using the files that you downloaded in Task 5.

There are 2 options of implementing this task. Choose **one** of these two options, based on your programming background. Option A allows you to use the commandline tools provided by `openssl` to encrypt and decrypt messages. Option B, which is more challenging and earns you extra points, will teach you how to use `openssl`'s crypto library from a C program to encrypt/descrypt messages in programs.

Test the implementation: Run your program with the input above. Use a screenshot to demonstrate that you have successfully inferred the key that was used for encryption.

Reflections on programabillity and performance: Give a critical analysis of the advantages and disadvantages of your implementation in the chosen option (A or B), and compare it with the alternative implementation using the other option. Comment on the programmability in each of the two options, e.g. how easy is it to adapt the program to use a different dictionary or use external software for guessing passwords, and on performance, i.e. which of the two options do you believe to be faster and explain why.

3.6.1 Option A: Using shell scripting and command line openssl

Use the `openssl` commands for en-/de-cryption in a shell script to determine the key that was used to encode the file. As an example of how to process the word list file, check this line count script, which reads text from a file and counts the number of lines. You will have to replace the body of the loop by calls to the appropriate `openssl` command and checking the result file.

Hint: to speed up development, test your script on a self encrypted, short text, and use a much shorter word list; once your solution works for your own text, run it on the given text and the full word list.

3.6.2 Option B: C Programming using the Crypto Library

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 C code is given here and can be downloaded here. Start from this C code and extend it to do Task 6.

Hint: A useful function for this task is `EVP_BytesToKey`, which transforms a password (string) into a key (binary).

Note: 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. For example, 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
```

4 Submission

You need to submit a detailed report to describe what you have done and what you have observed. Use either screenshots or just cut-and-paste the commandline with the responses, documenting the steps taken on each of the tasks above. You also need to explain the observations that are interesting or surprising. Document any difficulties that you met while doing this exercise.

In your report, you need to answer all the questions listed in this lab. Structure the report in sections, addressing each of the tasks. Include an introduction section, discussing what you expect to learn from the assignment and clarifying which assumptions you made in performing the tasks. Include a summary section, where you discuss whether your expectations were met, highlighting issues of particular importance, and suggesting further work, i.e. what kind of exercise could be done next building on the material from this assignment. If you didn't finish the complete task, clearly describe how far you got and what still would need to be done. In writing-up the work you performed on the above tasks, make sure to assess the strengths and weaknesses of the ciphers and modes used in for this assignment.

The **deadline for submitting the report is 3:30pm on Friday 10^{th} of October, 2014**. You should submit an electronic version via Vision (one file for the report and one file for the program sources). Also post a hardcopy with a cover sheet in the appropriate drop-box in front of the School Office 1.24.