Lab 2 – Implementation of DES/RSA and SHA-1 (Final version)

A. Introduction

1. The purpose of the lab

The notion of cryptography consists of hiding secret information from non-trusted peers by mangling messages into unintelligible text that only trusted peers can rearrange. In this lab, we will use and compare three different techniques commonly employed to hide or encrypt information: secret key cryptography (DES), public key cryptography (RSA) and message digests (SHA-1).

There is a group of built in functions that will help you complete this assignment. These functions are included in Openssl. Openssl is an open source toolkit that implements security protocols such as SSL but also includes a general-purpose cryptography library which you will use. Openssl is usually part of most of the recent Linux distributions.

This lab will be done in 4 sessions, including one lab lecture and one session for report preparation.

You should solve the problems in the presented order, i.e. solve the DES encryption/decryption problem, benchmark the DES, RSA and SHA-1 algorithms second. The last session is reserved for report preparation.

2. Sample codes and results

We are giving a group of files (folder name: "**Provided_Code"**) as a reference on how to use certain built in functions that we will mention in this handout and that you must use in your implementations.

There are two sub-folders: "**Provided_Code/Prob1"** which contains code skeletons and data for **Problem1** and "**Provided_Code/Prob2"** which contains code skeletons and data for **Problem 2**

These files are just skeletons that you should modify in order to obtain the expected results.

Included in the folder is also a file (*func_desc.txt*) with general descriptions of some built in functions.

B. Problems

Problem 1: DES encryption/decryption (50 points)

In this part of the lab, we will be coding a tool to encrypt and decrypt files using DES in mode CBC (Cipher Block Chaining). "tempdes_simple.c" is an implementation that encrypts/decrypts a fixed 64-bit (8 bytes) block.

In this assignment, you will extend this code to take an input file whose length (in bytes) is a multiple of 8 and encrypt/decrypt it, by implementing the Cipher Block Chaining DES mode of operation. You must actually implement the CBC mode, and you are not allowed to use any built-in function besides what is present in tempdes_simple.c. You can find information about DES-CBC in your text book.

The code skeleton names tempdes_cbc.c. To make your program interact with external files and adopt keys it should take following format:

./tempdes_cbc iv key inputfile outputfile

where the parameter iv is the initialization vector, and the parameter key is the key. Both must be represented as a string comprised only of hexadecimal digits. If any of the arguments is invalid, your code should return an appropriate message to the user. Be sure to consider the case when the keys are invalid.

In the skeleton code the macros c2l and l2c convert a 4-byte unsigned char array to a DES_LONG value in the correct format and vice versa. This is useful in your DES-CBC encryption and decryption methods.

Verification of your program

You may want to check your work against the input file "test.txt" which is 64-byte long. If you have implemented the algorithm correctly, you should get the output in "test.des".

To do this you should use following command:

```
./tempdes_cbc fedcba9876543210 40fedf386da13d57 test.txt my_test.des
```

and compare my_test.des to test.des. Furthermore, you can verify your DES in CBC mode against the built-in function DES_cbc_encrypt(input, output, length, ks, ivec, enc); the output of the built-in function and your DES-CBC should be similar. To see the content of a .des file, use the following command

```
$ less filename.des
```

Press ‘Q’ to quit.

- Provided codes (folder: Prob1)
  - **tempdes_cbc.c**: a skeleton that you need to complete to encrypt an input file whose length (in bytes) is a multiple of 8 using CBC (64 bytes as for our input “test.txt”)

- Compile your code
  - SunOS & Ubuntu:
    ```
    $ gcc -o tempdes_cbc tempdes_cbc.c -lssl -lcrypto
    $ ./tempdes_cbc iv key inputfile outputfile
    ```
  - MacOS:
    ```
    $ gcc -o tempdes_cbc tempdes_cbc.c -lssl -lcrypto -L/usr/local/opt/openssl/lib
      -I/usr/local/opt/openssl/include
    $ ./tempdes_cbc iv key inputfile outputfile
    ```

Problem 2: Benchmark DES, RSA and SHA-1 (50 points)

This part of the lab consists of measuring the time DES, RSA and SHA-1 take to process files of different sizes (from 8 bytes, 2 Mbytes).

The folder containing the lab also has text files with 8 different file sizes specialized for each encryption and decryption method (SHA-1 cannot be decrypted though).

Furthermore, the files associated with the lab contains a skeleton code for each algorithm, which demonstrates the different function calls, but you will need to augment each skeleton to allow for reading from arbitrary files and for measuring the execution time.

To measure the processing time you can use the gettimeofday() function located in the <sys/time.h> library. This snippet provides the functionality:

```c
struct timeval t_start, t_end;
long elapsed;
gettimeofday(&t_start, 0);
// The execution routine
gettimeofday(&t_end, 0);
elapsed = (t_end.tv_sec-t_start.tv_sec)*1000000 + (t_end.tv_usec - t_start.tv_usec);
```

Implement the DES, RSA and SHA-1 using the built-in functions in the OpenSSL library and enable the programs to output processing time, like:

```
$ ./temprsa ./data/test_512.txt
File containes 512 bytes
Encryption time: 55 us
Decryption time: 2026 us
```

- Provided codes (folder: Prob2)
  - **tempdes.c**: code skeleton for DES
  - **temprsa.c**: code skeleton for RSA
  - **tempsha1.c**: code skeleton for SHA1

Run each program 5 times and compute the averaged processing time for DES, RSA and SHA-1

C. Report Preparation

Report, including two parts: typed report and source code.

Content requirements

For typed report: The report should contain

(1) Problem 1: (1) proof of your code run actually (copy & paste of the console screen)
(2) Problem 2: (1) proof of your code run actually (copy & paste of the console screen) (2) answer of the following questions

1. Make graphs of the time measurements [µs] as function of file sizes [bytes] that show: (20 points)
   (a) DES encryption/decryption times.
   (b) RSA encryption times.
   (c) RSA decryption times.
   (d) SHA-1 digests generation times.
2. Answer the following questions:
   (e) Compare DES encryption and RSA encryption. Explain your observations. (10 points)
   (f) Compare DES encryption and SHA-1 digest generation. Explain your observations.
   (g) Compare RSA encryption and decryption times. Can you explain your observations?

For source code: All source codes must be submitted.

Format requirements

Both parts must be compressed (zip) and submitted together

- Name the zipped file as <Year.Month.Date>.<Student_ID>.<LabXX>.zip
- For example: 2019.07.11.s1222222.Lab2.zip

Requirements for typed report

- Must be prepared using a Word processor (e.g. MS Word or Latex)
- Must be converted into PDF for submission

- Submitted individually
- Also name the file as <Year.Month.Date><Student_ID><LabXX-report>.PDF
- Example 2019.07.11.s122222.Lab2-report.pdf

Requirements for source code

- Also name the file as <Year.Month.Date><Student_ID><file_name>
- For <file_name>, **tempdes_cbc.c** for Problem 1; **tempdes.c**, **temprsa.c**, **tempsha1.c** for Problem 2.

Lab submission

- By email (send the zip file to d8202101, m5222108, and CC to pham). The subject of your mail should be: **[CN02]Lab2**
- Due date: by the mid-night of the last-session day (report preparation)
- Late submission is subject to penalty