COMP4337/9337

Securing Wireless Networks (SWN)

Lab #1
INTRODUCTION TO BASIC CRYPTOGRAPHIC MECHANISMS

S1, 2019 – V:1 (Monday, 25.02.2019, 16:00)

Use eng.cse.COMP4337@unsw.edu.au *for class communications*
Students may use the Technical Questions Forum on Moodle to discuss this lab

---

1. General Description

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 lab. 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.

2. File with sample codes and results

We are giving a group of files (compressed in zip format) 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. These files are just skeletons that you should modify to obtain the expected results. Included in the compressed file is also a test case for your DES-CBC implementation (test.txt and test.des) and a file with general descriptions of some built-in functions. Please download the file skeletoncode.zip.

3. DES encryption / decryption

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.py" or "tempdec.c" (for C programmers) is a skeleton file that encrypts/decrypts a fixed 64-bit block. In this lab, you will extend this skeleton code to take an arbitrarily sized input file and encrypt/decrypt it, by implementing the Cipher Block Chaining DES mode of operation. You must implement the CBC mode. You may only use the built-in functions in "tempdes.py" or "tempdec.c". You can find information about DES-CBC in textbooks and online.

You may want to check your work against the input file "test.txt." If you have implemented the algorithm correctly, and follow 3.1 instructions you should get the output in "mytest.des." Check this against “test.des” provided to see if it matches. These files are part of skeletoncode.zip.

<u>NOTE: If you want to cocde in C, you will find “C” folder with corresponding skeleton code written in C.</u>

3.1 Requirements

(For Python)

a. Use the built-in functions that appear in tempdes.py.

b. Your code should result in a python script of the following form:

```
python tempdes.py   iv   key  inputfile outputfile
```

The parameters description is as follows:

- **`iv`**: the actual IV to use: this must be represented as a string comprised only of hexadecimal digits.
- **`key`**: the actual key to use: this must be represented as a string comprised only of hexadecimal digits.
- **`inputfile`**: input file name
- **`outputfile`**: output file name

For example:

```
python  tempdes.py  fecdba9876543210   0123456789abcdef
     test.txt       mytest.des
```

(For C)

a. Use the built-in functions that appear in tempdes.c

b. To compile the code use lcrypto library, for example:

```
gcc  tempdes.c -o tempdes -lcrypto
```

where, tempdes.c and tepmdes are your source and executable codes, respectively.

c. Your code should result in an executable of the following form:

```
./tempdes iv   key  inputfile outputfile
```

The parameters description is as follows:

- **`iv`**: the actual IV to use: this must be represented as a string comprised only of hexadecimal digits.
- **`key`**: the actual key to use: this must be represented as a string comprised only of hexadecimal digits.
- **`inputfile`**: input file name
- **`outputfile`**: output file name

For example:

```
./tempdes fecdba9876543210    0123456789abcdef
     test.txt       mytest.des
```

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.

4. **Performance measures for DES, AES, RSA, HMAC and SHA1**
   The final part of this lab consists of measuring the time DES, AES, RSA, HMAC and SHA-1 take to process files of different sizes.
   a. Generate text files with the following sizes:
      - For DES (in bytes): 8, 64, 512, 4096, 32768, 262144, 2047152
      - For AES (in bytes): 8, 64, 512, 4096, 32768, 262144, 2047152
      - For HMACs (in bytes): 8, 64, 512, 4096, 32768, 262144, 2047152
      - For SHA-1 (in bytes): 8, 64, 512, 4096, 32768, 262144, 2047152
      - For RSA (in bytes): 2, 4, 8, 16, 32, 64, 128
   b. Encrypt and decrypt all these files using the DES function that you wrote in part 3. Measure the time it takes to encrypt and decrypt each of the files. To do this, you might want to use the python/C timing functions. Add these timing functions to your implementation of part 3.
   c. Repeat the above but instead of DES use AES.
   d. Measure the time for RSA encryption and decryption for the file sizes listed in part a. To do this, make appropriate changes to the file ”temprsa.py”(or temprsa.c). This skeleton code shows how to use built-in RSA encryption and decryption functions, but you will need to augment it to allow for reading from arbitrary files, and to insert necessary instrumentation code for timing purposes.
   e. Measure the time for SHA-1 hash generation for the file sizes listed in part a. To do this, make appropriate changes to the file ”tempsha1.py” (or “tempsha1.c”). This skeleton code shows how to use built-in SHA-1 hashing functions, but you will need to augment it to allow for reading from arbitrary files, and to insert necessary instrumentation code for timing purposes. You are encourage to test other hash functions from the library.
   f. Measure the time for HMAC signature generation for the file sizes listed in part a. To do this, make appropriate changes to the file ”tempHMAC.py” (or “tempHMAC.c”). This skeleton code shows how to use built-in HMAC functions, but you will need to augment it to allow for reading from arbitrary files, and to insert necessary instrumentation code for timing purposes.

   g. Prepare a report of your observations in the format requested under “**Deliverables” (section 6).**

5. **Deliverables:**

   a. This Lab needs to be completed in groups.
      - All members will get the same mark.

   b. Create a folder named “Lab 1 Submission” containing two subdirectories: A) “Code”, B) “Report”.

c. In “Code” folder,

   - For **python** version, include a copy of the file “tempdes.py” .

     Name your file <groupname>.des.py, where group name is the name of your group on Moodle. For example: SWN19-A.des.py.
   - For **c** version, include a copy of the file “tempdes.c” .

     Name your file <groupname>.des.c, where group name is the name of your group on Moodle. For example: SWN19-A.des.c.

d. In “Report” folder, upload a **typed** report with the following information:

   - On the first page, include the name of the group, student names and ZIDs clearly on the first page.

   - Python or C code implemented in parts 3, 4.c and 4.d

   - Graphs showing: (i) DES encryption / decryption times; (ii) AES encryption / decryption times; (iii) RSA encryption / decryption times; (iv) SHA-1 digest generation times and (v) HMAC signature generation times. In each of these graphs, the X-axis should plot the file sizes in units of bytes, and the Y-axis should plot time measurements in units of microseconds (µs).

   - Answer the following questions:
     - Compare DES encryption and AES encryption. Explain your observations.
     - Compare DES encryption and RSA encryption. Explain your observations.
     - Compare DES encryption and SHA-1 digest generation. Explain your observations.
     - Compare HMAC signature generations and SHA-1 digest generation. Explain your observations.
     - Compare RSA encryption and decryption times. Can you explain your observations?

e. Now, ZIP the “Lab 1 Submission” file and upload it on Moodle using the designated link. Only ZIP files are allowed.

6. Marking & Deadline

Marks will be awarded by submission. Submissions will be awarded 2 marks (out of 100 for this course). Non-submission will result in losing 2 marks (out of 100 for this course). We will randomly check for complete submissions. Upto 10 marks will reduced from final marks for the subject if a deliberate attempt to fake submission is found.

Deadline for submission is Monday, 4th of March, 9:00 AM. Submissions after this deadline will be penalized as below:

7. Late Submission Policy

1 day after deadline: 10% reduction of the total mark for this component
2 days after deadline: 20% reduction of the total mark for this component
3 days after deadline: 30% reduction of the total mark for this component
4 days after deadline: 40% reduction of the total mark for this component
5 days or more: NOT accepted