COMP4337/9337 Securing Fixed and Wireless Network
Lab 1: Introduction to Basic Cryptographic Mechanisms

Due: Monday 28th February 2022 11.59PM AEDT

Objectives

This lab is designed to help the students to:

- learn the basics of cryptography for hiding secret information from non-trusted peers,
- implement DES algorithm using Python or C, and
- compare the performance of DES, AES, RSA, HMAC, and SHA-1 algorithms.

Lab Overview

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, AES), public key cryptography (RSA) and message digests (SHA-1).

We also give a group of files as a reference on how to use certain built-in functions that we will mention in this handout and that you can 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 refer to the `skeletoncode` folder provided by your tutors.

Assessment and Marking

The marks will be made available within 2 weeks of the submission date. The details of the marks are as follows:

- Total mark for Lab 1 is 100. This lab combined with marks for other labs will be scaled to 20 out of 100. The weight for this Lab is 0.3/1.
    - Lab Performance **(20),**
    - Report **(40)**
    - Code **(40)**
- Students who do not attend the lab will **lose ALL 100 marks.** Please discuss with the lab tutors if there is a valid reason for not attending.
- **Note:** Lab performance involves tutor asking question, feedback, and comments about the activity while the lab is in progress. Hence, if a group is found to be cheating or submitting a work that does not match what the tutor observes of the team performance, then NO MARK will be awarded for that group.
- The standard late penalty introduced under UNSW new assessment implementation procedure will be applied for this course.
    - 5% per day,
    - for all assessments where a penalty applies,

- capped at five days (120 hours) from the assessment deadline, after which a student cannot submit an assessment, and
- no permitted variation.

Submission

Please follow these instructions to prepare a submission for this lab:

1. You are given the option to do this lab either individually or in a pair. If you opt to do in a pair, both team members will get equal marks.
2. Create a folder named `Lab 1 Submission` containing two subdirectories: `code/` and `report`.
3. In `code/`:
   - For Python, include a copy of `tempdes.py` file. Name your file `<zid_1>_<zid_2>.des.py`, where zid_1 and zid_2 are the zids of group members. **For example:** `z1234567_z7654321.des.py`.
   - For C, include a copy of `tempdes.c` file. Name your file `<zid_1>_<zid_2>.des.c`, where zid_1 and zid_2 are the zids of group members. **For example:** `z1234567_z7654321.des.c`.
   - For students who do not have a strong preference and an experience in using C, we recommend using Python.
4. In `report/`, upload a written report with the following information:
   - On the first page, include the name of the group, student names and ZIDs clearly.
   - Python or C code implemented.
   - Graphs showing:
     - DES encryption/decryption time,
     - AES encryption/decryption time,
     - RSA encryption/decryption time,
     - SHA-1 digest generation times, and
     - HMAC signature generation times.

     Note: 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 and AES. Explain your observations.
     - Compare DES and RSA. Explain your observations.
     - Compare DES and SHA-1. Explain your observations.
     - Compare HMAC and SHA-1. Explain your observations.
     - Compare RSA encryption and decryption time. Can you explain your observations?
5. Compress `Lab 1 Submission` and its content into a `.zip` file and upload that file on Moodle using the link: https://moodle.telt.unsw.edu.au/mod/assign/view.php?id=4496656&rownum=0&useridlistid=62133f8d29b22693776532&action . Only `.zip` file format is allowed.

Part A: DES encryption and decryption

In this part of the lab, we will be coding a tool to encrypt and decrypt files using DES in Cipher Block Chaining (CBC) mode. `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 the 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 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 followed the instructions below you should get the output in `mytest.des`. Check the output against `test.des` provided to see if it matches. You can find these files in `skeletoncode/`.

Technical Requirements

- Use the built-in functions that appear in `tempdes.py` or `tempdes.c`, depending on your chosen language.
- For C code, use `lcrypto` library to compile the source code, for example:

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

  where, `tempdes.c` and `tempdes` are your source and executable, respectively.

- Your result should take the following arguments:

  For Python:

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

  For C:

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

- The parameters description is as follows:
  - `iv`: the actual IV to use, represented as a string comprised only of hexadecimal digits.
  - `key`: the actual key to use, represented as a string comprised only of hexadecimal digits.
  - `inputfile`: input file name.
  - `outputfile`: output file name.
- Example command to run the resulting code:

  For Python:

  ```
  $ python3 tempdes.py fecdba9876543210 0123456789abcdef
  test.txt mytest.des
  ```

  For C:

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

- **Please note that to get the sample output files you need to use the key and IV given in the README file.**
- If any of the arguments is invalid, your code should return an error message to the user. Be sure to consider the case when the keys are invalid.

Part B: Performance measures for various algorithms

The final part of this lab consists of measuring the time taken by DES, AES, PRESENT, RSA, HMAC and SHA-1 algorithms to process files of different sizes. Please follow the following steps:

1. Generate text files with the following sizes:
   - For DES, AES, PRESENT, HMAC, and SHA-1 (in bytes):

     `16, 64, 512, 4096, 32768, 262144, 2047152`

   - For RSA (in bytes):

     `2, 4, 8, 16, 32, 64`

2. Encrypt and decrypt all the files using the **DES** function that you wrote. 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 specific implementation.
3. Repeat the above but instead of DES use **AES**.
4. Measure the time for **RSA** encryption and decryption for the file sizes listed in `1`. 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.
5. Measure the time for **SHA-1** hash generation for the file sizes listed in `1`. 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 encouraged to test other hash functions from the library.
6. Measure the time for **HMAC** signature generation for the file sizes listed in `1`. 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.
7. Prepare a report of your observations in the format requested under Submission.

Part C: PRESENT Algorithm (Optional)

Any student interested in testing more algorithms, we provided the skeleton code for PRESENT, a lightweight encryption algorithm. You can follow the following instructions to test the performance of that algorithm. For more details about PRESENT algorithm, please refer to the paper here.

Measure the encryption/decryption times for **PRESENT** lightweight encryption and decryption for the file sizes listed in `1`. To do this, make appropriate changes to the file `temppresent.py` (or `temppresent.cpp`). This skeleton code shows how to use the implemented PRESENT 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.

Note: PRESENT Python implementation may only be able to take hexadecimal string as the input (0-9, a-f).