CS 5158/6058 Data Security and Privacy, Spring 2018
Project 1: One-Time Pad

Instructor: Dr. Boyang Wang

Due Date: 02/06/2018 (Tuesday), 11:59pm.
Format: Please submit a zip file of your code in Blackboard.
Total Points: 10 points

1 Project Description

In this project, you will need to implement one-time pad encryption, including encryption function Enc, decryption function Dec, and key generation function KeyGen. More specifically,

- For the encryption function, given a plaintext m and a secret key sk, your program need to compute a ciphertext of this plaintext $c \leftarrow m \oplus sk$, print it in the terminal, and write this ciphertext to a file.
- For the decryption function, given a ciphertext c and a secret key sk, your program need to output the plaintext of this ciphertext $m \leftarrow c \oplus sk$, print it in the terminal, and write this plaintext to a file.
- For the key generation function, given a security parameter λ, your program need to output a secret key sk, where the length of sk is λ. In addition, you also need to write this secret key to a file.

2 Basic Requirements

Programming Language: You can use either C/C++, Python or Java. If you choose to use C/C++, CMake is recommended. You can choose any IDE you like, but the code you submit must allow me (or the grader) to compile and run from the terminal in Linux.

Program Directory: Please name your project folder as `otp_m123456`, where `otp` is the name of this project and `m123456` is your UCID. The recommended directories of your program should be organized as follows:

```
./otp_m123456/src
./otp_m123456/build
./otp_m123456/data
./otp_m123456/Readme.txt
```

Normally, folder `src` should include all the source files and your own header files, e.g., `.cpp` and `.h` files. All the object files and executable files, e.g., `.o` files, should be under folder `build`. Folder `data` has all the given files and data, and also includes all the files and results generated by the program. In `Readme.txt` file, you should write a description of your code, show which language and version you use, and illustrate how to compile, run and use your code.

3 Project Details

For ease of implementation, assume the security parameter (i.e., the length of the secret key) λ is fixed for encryption and decryption, and it is 32 bits. The (default) secret key is

$$sk = \underline{01010101}10101010\underline{11110000}00001111$$

it should be stored in "`../data/key.txt`".

A (default) plaintext is `bear`, and is stored in file "`../data/plaintext.txt`". Each character in plaintext can be represented as an ASCII code (e.g., `b` $\leftrightarrow$ 98), which can be further represented with 8 bits (e.g., 98 $\leftrightarrow$ 01100010).

1. Encryption Function:

(a) Read a secret key sk from file "`../data/key.txt`", and read a plaintext m from file "`../data/plaintext.txt`";
(b) Represent plaintext m in binary, if its length is the same as the length of secret key sk, compute its ciphertext using one-time pad, print the ciphertext in the terminal, and write this ciphertext (in binary) to a file "`../data/ciphertext.txt`";
(c) If the length of a plaintext is different from the length of secret key, return and display an error message in the terminal (e.g.,"`error: length is incorrect!`"), and file "`../data/ciphertext.txt`" should be an empty file in this case;
(d) Your encryption function should be able to encrypt any plaintext with four characters, e.g., `eecs`, `cats`, `bike`, `rice`, etc.

2. Decryption Function:

(a) Read a secret key sk from file "`../data/key.txt`" and a ciphertext c from file "`../data/ciphertext.txt`";
(b) Decrypt this ciphertext c, output its plaintext, and write this plaintext (as a string, e.g., `bear`) to file "`../data/result.txt`". Similar as the encryption function, if the length of secret key is different from the length of ciphertext, return and display an error message.

3. Key Generation Function:

(a) Given a security parameter λ as a command-line argument, where $1 \leq \lambda \leq 128$, write a function to randomly generate a secret key with λ bits, print this secret key in the terminal, and write this secret key to file "`../data/newkey.txt`".

4. (CS 6058 Only) Distribution of Keys:

(a) Given a security parameter $\lambda = 3$, repeat your key generation function for at least 5000 times. Record all the unique 3-bit keys you program generated, calculate the frequency of each one, and prove that those keys are (almost) uniformly distributed. You should have a function to automatically collect this frequency distribution. In addition, you also need to evaluate the average running time of your encryption function with $\lambda = 128$.
(b) You need to submit one-page report (in pdf) to explain and show your results in terms of key distribution and encryption time. In your report, you should describe details of your implementation, such as OS, programming language, crypto libraries, encryption parameters, etc. You can use tables, figures or screenshots to help you present your results in your report. Please put this report under your project folder and submit it together with your code.

4 Evaluation

Your project will be evaluated in three aspects.

1. **Correctness of Functions (75%):** Your program should be able to correctly run all the functions described in this project. If for some reason, your code cannot be compiled but the logic of your code is correct, you will still get partial credits.

2. **Comments and Descriptions (15%):** Write comments and explain each function in your code, such as inputs, outputs, etc. You may need some of the functions in other projects. Detailed comments on each function can save your time in other projects. In addition, please clearly explain how to compile and run your code in `Readme.txt`.
3. **Coding Style (10%):** A good coding style is always important, especially for large projects. Please keep each function simple, try to avoid long functions, and create multiple `.h` and `.cpp` files if needed. For example, it is not a good idea to put everything in the main function.

5 Examples

This section provides some examples, which can help you understand the functions of your project.

Example 1: The following command calls the encryption function

```
otp  enc  ../data/key.txt ../data/plaintext.txt ../data/ciphertext.txt
```

where `otp` is the name of your executable file, `enc` is argument for encryption function, `../data/key.txt` is the key file, `../data/plaintext.txt` is the plaintext file, and `../data/ciphertext.txt` is the ciphertext file. Note that, depending on where your executable file is, the paths of your input and output files might be different.

Example 2: The following command calls the decryption function

```
otp  dec  ../data/key.txt ../data/ciphertext.txt ../data/result.txt
```

where the decryption of a ciphertext will be written to file `../data/result.txt`.

Example 3: The following command calls the key generation function

```
otp  keygen  100 ../data/newkey.txt
```

where `100` is the length of a new key and this new key will be written to file `../data/newkey.txt`.