CS 4640: Computer Security **Spring 2021**

Homework 3 (Individual Programming Assignment)

Total 100 Points

This homework will contribute to 15% of your final grades.

Due date & time: 11:59 pm on April 14th, 2021.

Late Policy: You have three extra days in total for all your homeworks and projects. Any portion of a day used counts as one day; that is, you have to use integer number of late days each time. If you exhaust your three late days, any late homework or project won't be graded.

Submission: Please submit a zip file of the 'src' directory of your homework package containing only the Java source file in ICON.

No points will be awarded for code that does not compile.
You should use Java for the assignment except one individual who I explicitly granted permission to use Python. C/C++ is not allowed for this assignment.
If you open any of the binary files (discussed in the problems below) in a text editor, you will see a bunch of garbage bytes. This is normal. You can optionally use hexdump to view the binary files if you want.

1 Problems

In this homework, you will be needed to solve 6 problems in Java. In your application package, you will see a directory called '`src`'. Inside that directory, you will see 6 more folders; one for each problem. Inside each of problem folder, you will see a '`*.java` file. When you open up the Java file, you will see that you have to fill up one or more functions.

Compilation: For compiling a java source, say for example, named '`Problem1.java`', you would need to type '`javac Problem1.java`'. After successful compilation, to execute the program you will type 'java Problem1 ...'. The '...' here refers to one or more command line arguments. **Note that, you would need to have Java installed in your machine for this assignment.**

1.1 Problem 1 (10 points) — BINARY FILE TO HEXADECIMAL

In this problem, you are supposed write a program that takes two command line arguments. The first argument is the name of an input binary file. The second argument is the name of an output file. You are supposed to read in the contents of the binary file, convert them to their hexadecimal equivalent, and output the hexadecimal representation of the binary context (in the ASCII format) to the output file.

Allowed Hexadecimal Digits: '01234567890ABCDEF'.

Testing: For testing purposes you can use any non-textual format file (e.g., *.pdf, *.doc). Your output file size should be exactly double the size of the input file. For your convenience, you are given an input file (named, '`input_hw2_CS4640.pdf`') and its corresponding correct output file (named, '`output_hw2_CS4640.hex`')

1.2 Problem 2 (10 points) — SECURE SYMMETRIC KEY GENERATION

In this problem, you will write a program to securely generate a random symmetric key of a particular length in **BYTES** and store it in a key file.

Your program will take two command line arguments. The first argument is the size of the key K (in **BYTES**) and the second argument is the name of an output file f. In this program, you are supposed to generate a cryptographically secure key of size K bytes and write the key to the output file f in the binary format.

There is a hint in the corresponding Java source file regarding how to generate a cryptographically secure key.

Testing: It is difficult to automatically test whether your key generation program is working. One possibility is to check that the output key file is indeed the right size. We will grade it based on the actual code instead of its output.

1.3 Problem 3 (15 points) — RANDOM ONE-TIME PAD

In this problem, you are supposed to write a program that encrypts the content of a given plaintext file using a given key and output the ciphertext in another file.

Your program will take the following three command line arguments.

Argument 1 The first one is the name of a binary file which you would like to encrypt/decrypt[1].

Argument 2 The second argument is a binary file containing the secret key to be used for encryption and decryption.

Argument 3 The third argument is the name of the output file in which you are supposed to write the ciphertext/plaintext after the encryption/decryption operation.

You can safely assume the input file size and the key file size will be same.

Testing: You can test random one-time pad easily due to its reversible nature. For generating a key, you can use your solution to the problem 2. Suppose you want to encrypt the contents of a plaintext file named '`test.in`' with the key contained in the file '`key.in`', and write the ciphertext into the file called '`test.cipher`', then you will invoke your program in the following way: "`java Problem3 test.in key.in test.cipher`". Then you can invoke your program again for decryption in the following way: "`java Problem3 test.cipher key.in test.plain`". The contents of the files `test.plain` and `test.in` should be identical. If they are not then your program does not work correctly.

[1]Note that, the encryption and decryption function for Random one-time pad is same. You just have to change the position of the plaintext and ciphertext to do encryption or decryption.

Sample Input-Output Files: Plaintext file (`input_hw2_CS4640.pdf`); Key file (`key_hw2_CS4640`); Ciphertext file (`hw2_CS4640.cipher`).

1.4 Problem 4 (20 points) — AES Encryption in the CBC mode (BLOCK SIZE=128 BITS)

In this problem, you are supposed to generate a random 128-bit key and a random 128-bit IV, use them to encrypt a plaintext file.

Your program will take the following four command line arguments.

Argument 1. This is the name of the plaintext file whose binary contents you want to encrypt using the AES CBC mode.

Argument 2. This is the name of the output key file where you will store the randomly generate 128 bit AES key. (needed for decryption in Problem 5)

Argument 3. This is the name of the output IV file where you will store the randomly generated IV. (needed for decryption in Problem 5)

Argument 4. This is the name of the output ciphertext file where you will store the ciphertext resulting from encrypting the plaintext file. (needed for decryption in Problem 5)

There is a hint in the corresponding Java source file.

Testing: For testing this implementation, you would have to rely on the solution to your problem 5. You will encrypt a file with your Problem 5 program and then will try to decrypt using your program for Problem 6. If after decryption, you get the original file back, then both your encryption and decryption are working.

1.5 Problem 5 (15 points) — AES Decryption in the CBC mode (BLOCK SIZE=128 BITS)

In this problem, you are supposed to write a program that decrypts an encrypted binary file using a given key and IV.

Your program will take the following four command line arguments.

Argument 1. This is the name of the ciphertext file whose binary contents you want to decrypt using the AES CBC mode.

Argument 2. This is the name of the input key file which will contain the 128 bit key that you are going to use for decryption.

Argument 3. This is the name of the input IV file which will contain the 128 bit IV that you are going to use for decryption.

Argument 4. This is the name of the output plaintext file where you will save the plaintext after decryption.

There is a hint in the corresponding Java source file.

1.6 Problem 6 (30 points) — Sorting Binary Records (Hardest Problem — Try Last)

In this problem, you are supposed to write a program that sorts (in the ascending order) some binary records. A binary record is 100 bytes long. The integer corresponding to the first 4 bytes of the record is the *key*. The remaining 96 bytes correspond to the record's *value*. See Figure 1 for the components of a record.

Your task is to write a program that takes a file containing one or more records as command line argument, sorts the records in the ascending order of the record's key value (i.e., lower to higher values of key), and output the sorted records into another file given to you as the second command line argument. Note that, I am not asking you to just output the keys in the output file. You should consider a record to be an atomic construct.

Note that, when interpreting the first 4 bytes of a record as an integer, you should consider them to be stored in the little endian format (`https://en.wikipedia.org/wiki/Endianness`).

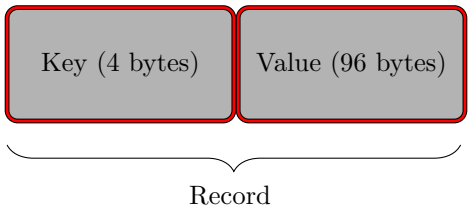

Figure 1: Figure: The binary record

Your program should take the following command line arguments:

Argument 1. Name of the binary file containing one or more binary records as discussed above.

Argument 2. Name of the output file in which you will save the binary records after you have sorted them in the ascending order of the key values.

1.6.1 Testing Facilities.

The following facilities have been given to you for testing your program.

Input generation. For generating inputs, please consult the `README` file inside the '`InputGenerator`' folder.

Viewing record files. For viewing a generated input file or your output file, please consult the `README` file inside the '`Viewer`' folder.

Reference implementation. For comparing your outputs, I have given a solution written in C. Please consult the `README` file inside the "`Reference-Solution`" folder.

1.6.2 Suggested Testing Approach.

1. You should first generate an input with the input generation program given in your homework package.

2. You will then run the input through the C reference solution I have given you.

3. You will then run the input through your own program.

4. You will then compare the contents of your output with the reference solution's output. You can use the '`diff`' utility in most Unix and Mac OS operating systems.

5. If they match, your implementation generated the right output. It is likely working and then try with larger number of records.

6. If they do not match, use the viewing program to see whether you are interpreting the keys correctly. For this, try with an input file containing 2/3 records. Check the keys in your output file appears in the input file. If they do not, then you are likely interpreting the keys wrongly.

7. You are in this step means all the keys in the input file also appears in your output file. This could mean you may not be sorting them correctly. If they are not sorted correctly, it's likely your comparator function used for sorting that is wrong.

8. You are in this step means all the keys in the input file appears in your output file and also they appear in the ascending order. Then, the most likely reason is that