Design and Implementation of Symmetric Encryption

Real-time Update:

* The original file should have an entropy of 5.151873619963597. Using my own implementation (which should be strong enough), the entropy of the encrypted file is 7.9964121590600977. You may use this as a reference. It does not necessarily mean your algorithm is weak if you get an entropy lower than mine. In general, the closer the entropy value is to 8.0, the stronger the encryption.
* The third bug was identified. A patch has been pushed to the GitHub repo at <https://github.com/junxzm1990/CS-4440-2022-Spring/tree/main/assignment-1>. Please pull again before you do the testing.
* Another small bug was identified. A patch has been pushed to the GitHub repo at <https://github.com/junxzm1990/CS-4440-2022-Spring/tree/main/assignment-1>. Please pull again before you do the testing.
* There is a small bug in the code we provided. A patch has been pushed to the GitHub repo at <https://github.com/junxzm1990/CS-4440-2022-Spring/tree/main/assignment-1>. Please pull again before you do the testing.

## Description

The goal of this assignment is to get familiar with symmetric encryption, including how it works, what properties it should carry, and how to use it for protecting data.

In this assignment, you are required to design a symmetric encryption algorithm that can provide a block cipher. The key size is 8-bit and the block size is 8-byte. You are free to design the algorithm as you would like but you are suggested to follow the design of standard algorithms like DES/AES (e.g., you are suggested to include a key-derivation scheme and multiple rounds of encrypting operations). By following the standard algorithm, your algorithm is more likely to be correct and provide stronger encryption, which will bring you a higher score. It is completely fine that you just follow the scheme of DES/AES if you can adapt their algorithms into an 8-bit key for 8-byte blocks; no points will be deducted if you do so.

When you finish your design of the algorithm, you will be required to implement two functions:

* public static byte[] cs4440Encrypt(byte[] data, byte key): this function will take two arguments: (i) data representing a block of 8-byte plain data and (ii) key representing an 8-bit key. Its code will need to encrypt the data using the given key and your encryption algorithm. The encrypted data, also 8-byte long, should be returned via the return value.
* public static byte[] cs4440Decrypt(byte[] data, byte key): this function will take two arguments: (i) data representing a block of 8-byte cipher data and (ii) key representing an 8-bit key. Its code will need to decrypt the data using the given key and your decryption algorithm. The decrypted data, also 8-byte long, should be returned via the return value.

When you implement the above two functions, you will need to create two more functions that respectively leverage cs4440Encrypt and cs4440Decrypt to encrypt and decrypt a given file.

* public static int encryptDoc(String plainfilepath, String cipherfilepath, byte key): this will take three arguments: (i) plainfilepath representing the path to the file that will need to be encrypted (ii) cipherfilepath representing the path to the file that will save the encrypted data and (iii) key representing the key to do the encryption. Its code will need to encrypt the data in the file at plainfilepath using the key and the above cs4440Encrypt function, and then save the encrypted data to the file at cipherfilepath.
  + You will need to use the Cipher Block Chaining mode when multiple data blocks need to be encrypted
  + If the data in the last block is less than 8-byte long, you will need to pad the data following the PKCS5 padding scheme (it has been both discussed during the lectures and explained on the slides; but a reference is here: <https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html>). Note: if the last block contains exactly 8 bytes, you still need to do padding; Please see the reference for details.
* public static int decryptDoc(String cipherfilepath, String plainfilepath, byte key): this will take three arguments: (i) cipherfilepath representing the path to the file that contains the encrypted data (ii) plainfilepath representing the path to the file that will save the decrypted data and (iii) key representing the key to do the decryption. Its code will need to decrypt the data in the file at cipherfilepath using the key and the above cs4440Decrypt function, and then save the encrypted data to the file at plainfilepath.
  + You will need to use the same Cipher Block Chaining mode when multiple data blocks need to be decrypted.
  + Padding bytes at the last block must be removed before that block is saved to the file. Note: using the PKCS5 scheme, the last block may contain only padding bytes.

## Specifics

Note: You will need to use **Java** to complete this assignment. The choice of Java is based on the survey result and, more importantly, the pre-requisite courses. If you cannot do Java, then you will need to explain to me how you satisfied the pre-requisite courses.

To help your implementation, we have provided a wrapper program: CryptUtil.java. In the program, we have created the code skeleton. What you need to do is to complete the code of the above four functions (API wrappers have been provided in the program for those functions).

When you complete the code, you can run CryptUtil.java, which will automatically launch some tests of your code.

* You will need to run the program with three arguments:
  + the first argument gives a path of a file that you want to encrypt;
  + the second argument gives a path of a file where you want to save the encrypted data;
  + the third argument gives a path of a file where you want to save the data decrypted from the encrypted data.
* A reference file (to-be-encrypted), cs4440-a1-testcase1.html, is provided to help you do the test, but it is highly suggested you also try other large files. We will use different files to test your submission.
* If your code runs OK (no bugs), you will see a set of messages from CryptUtil.java
  + It will tell you whether a simple test of cs4440Encrypt and cs4440Decrypt works.
  + It will tell you the entropy of the original file and the encrypted file. The higher the entropy of the encrypted file is, the stronger your encryption algorithm is. At least, you should see the entropy of the encrypted file is way higher than the original file (unless you give the program a file with random data to encrypt)
  + It will tell you whether decrypting the encrypted file will reproduce the original file. This will tell you whether your encryption and decryption are correctly working.
* All materials related to this assignment are available at: [https://github.com/junxzm1990/CS-4440-2022-Spring/blob/main/assignment-1/](https://github.com/junxzm1990/CS-4440-2022-Spring/blob/main/assignment-1)

## Submission

* You just need to submit CryptUtil.java with your code inside to Gradescope: <https://www.gradescope.com/courses/347861>

## Grading Policy:

The grading will be based on whether your code can run or not in each step:

**Step 1:** testingthe **cs4440Encrypt** and **cs4440Decrypt** interfaces:

* If the testing the program with **cs4440-a1-testcase1.html** and **cs4440-a1-testcase2.txt** both print “**It works!**” and the functions contain meaningful encryption/decryption logic: **+30**
* If the testing does not work but the functions contain meaningful encryption/decryption logic: **+20**
* If the testing does not work but the functions contain code showing efforts: **+10**

**Step 2:** testingthe **encryptDoc** and **decryptDoc** interfaces:

* If the testing the program with **cs4440-a1-testcase1.html** and **cs4440-a1-testcase2.txt** both print “**The decrypted file is the same as the source file**” and the functions contain meaningful logic: **+30**
* If the testing does not work but the functions contain meaningful operation logic: **+20**
* If the testing does not work but the functions contain code showing efforts: **+10**

**Step 3:** testingthe entropy:

* If the entropy with encrypting **cs4440-a1-testcase1.html** is over 7.5: **+30**
* If the entropy with encrypting **cs4440-a1-testcase1.html** is between 7.0 and 7.5: **+27**
* If the entropy with encrypting **cs4440-a1-testcase1.html** is between 6.5 and 7.0: **+24**
* If the entropy with encrypting **cs4440-a1-testcase1.html** is between 6.0 and 6.5: **+21**
* If the entropy with encrypting **cs4440-a1-testcase1.html** is between 5.5 and 6.0: **+18**
* If the entropy with encrypting **cs4440-a1-testcase1.html** is below 5.5 and above 5.15: **+15**
* If step 1 or step 2 does not work but points are given, **+10** for the entroyp

**Step 4:** checking the code for padding:

* If the code is correctly implemented, **+10**