Web Security
Homework Assignment 2
COSC 4371
2018 Spring

Problem 0:

In this homework assignment, you will use the `javax.crypto` package. To get familiar with the most important classes and interfaces, read the "*Java security: Java security, Part 1: Crypto basics*" article at http://www.ibm.com/developerworks/java/tutorials/j-sec1/j-sec1.html, focusing on sections "*Keeping a message confidential*" and "*Ensuring the integrity of a message*."

Please solve the following problems by completing the attached Java source file. For each problem, replace the code between `// BEGIN SOLUTION` and `// END SOLUTION` with your solution. The submission uploaded to Blackboard should include the completed Java source file. Please make sure that the uploaded source file can be compiled and executed without unhandled exceptions.

Problem 1 (2 points): Decryption

Decrypt the ciphertext stored in the file `P1_cipher.txt`:

- plaintext was encrypted using `AES` in `CBC` mode with `ISO10126Padding` padding
- use the 16-byte key from the file `P1_key` and 16 zero bytes as the IV (see source code)
- use the class `SecretKeySpec` to set the key for the cipher (https://docs.oracle.com/javase/7/docs/api/javax/crypto/spec/SecretKeySpec.html)
- use the class `IvParameterSpec` to set the IV for the cipher (https://docs.oracle.com/javase/7/docs/api/javax/crypto/spec/IvParameterSpec.html).

The correct plaintext is an English sentence.

Problem 2 (2 points): Hash Function

1. Compute the `MD5` hash of the plaintext from the first problem:
   - use the `MessageDigest` class with `MD5`.

   The correct hash is 16-bytes long, and the value of the first byte is -28.

2. Decrypt the ciphertext from the file `P2_cipher.bmp`:
   - plaintext was encrypted using `AES` in **`ECB`** mode with `ISO10126Padding` padding
   - use the `MD5` hash of the plaintext from the first problem as the key.

   The correct plaintext is an ordinary BMP image file, which you should be able to open with any image viewing application.

Problem 3 (2 points): Information Leakage with ECB

The bitmap file `P3_cipher.bmp` was encrypted in ECB mode. Consequently, patterns in the image are not hidden and you should be able to view them easily.[1]

1. Try to open the file `P3_cipher.bmp`. Realize that since the header of the file is encrypted, your image viewer will not be able to display it, so this problem is going to be a bit harder than you thought.
2. Restore the header of the encrypted BMP file by overwriting it with the plaintext header of similar BMP file (`System.arraycopy` will be a useful function for this). Do you happen to have a similar BMP file somewhere?[2]
3. Open the modified file. You should see five numbers.

Problem 4 (2 points): Brute-Forcing a Non-Random Key

The file `P4_cipher.bmp` was encrypted with a 16-byte key using `AES` in `CBC` mode with `ISO10126Padding` padding, which is a secure encryption… but only if the key is secret and random. You have the following information about the key:

- The first five bytes were chosen in secret and at random. However, they were distributed using an encrypted bitmap file, which you might have seen already. Did you happen to see five numbers recently?
- The following three bytes are the hour (0 – 23), minute (0 – 59), and second (0 – 59) at which the key was generated. Unfortunately, that is all you know about them.
- The remaining bytes are all zeroes.

Furthermore, you know that the plaintext is a BMP file with same width and height as the previous bitmap files that you have encountered. Consequently, the first 6 bytes of the plaintext are the same as in Problem 2.

Using the above information, try to recover the correct key. Note that when you decrypt the ciphertext with an incorrect key, you **might** encounter a `BadPaddingException` exception. You can ignore these by simply catching them, and then continuing with the next key candidate.[3]

---

[1] See Lecture 5 – Block cipher modes of operation.

[2] Hint: you obtained a plaintext BMP file in the previous problem, which happens to have the same dimensions, color resolution, etc. as the BMP that was encrypted for this problem.

[3] You should surround your decryption with try-catch, e.g.,

```
for (key in possible_keys) {
  …
  try {
    … // test key
  }
  catch (BadPaddingException e) {
    // decryption with wrong key, ignore this
  }
}
```