CS4440: Exercise 1 – Cryptography in Java

Deadline: Friday, January 31 @ 11:59PM

1 Introduction

In this course, the purpose of lectures is to introduce you to concepts at a high level, the purpose of quizzes is to test your understanding of some of those concepts, the purpose of the homeworks is to work on more open-ended/freeform tasks, and the purpose of exercises is to give you hands-on experience with how some of the lecture concepts translate into practice.

The lectures don't translate into a how-to for the exercises. (However, not understanding the lecture material will make the exercises much, much harder.) You may need to look up additional information online (e.g., read linked documentation). This is intentional and realistic.

2 Setup

- This exercise uses Java. You may need to download and install the Java development toolkit (JDK) on your computer.
- Essentially, Java gives a standard set of APIs for performing cryptographic operations but lets developers/the installation environment choose who provides the actual crypto implementations. See the Notes for more information. This exercise is using the Bouncy Castle Crypto provider. You can find the files to download here. Go ahead and grab the latest release (`bcprov-jdk15on-164.jar`).
- You will also need to download the latest Apache Commons codec (`commons-codec-1.14`).
- The following files are provided:
  - `EncryptionTool.java` (this is where you complete part 3.1)
  - `AlterCTREncryptedFileWithoutKey.java` (this is where you complete part 3.2)
  - The following are files to help you test that your code is working:
    - `encrypted-with-AES-CTR-mode`
    - `encrypted-with-AES-CBC-mode`
    - `encrypted-with-AES-GCM-mode`
    - `boringfilenothingtoseehere`
    - `encrypted-with-hybrid-RSA`

    * `original-salary-plaintext`
    * `target-salary-plaintext`

- You are welcome to use an IDE for this course; however, it's overkill. (I would, however, recommend an editor that can perform syntax highlighting.) Please be aware that you may or may not make configuration harder on yourself down the road if you use an IDE e.g. when you have to run a `.class` file that you did not compile from code.
- You'll need to add the BC and Apache Commons codec jar files to the classpath when you compile and when you run the program, e.g. if they're in the same directory:

```
javac -classpath "./*" EncryptionTool.java

java -classpath "./*":. EncryptionTool <options>
```

  ...or, depending on your system:

```
javac -classpath "./*" EncryptionTool.java

java -classpath "./*";. EncryptionTool <options>
```

- You may want to download and use a basic hex editor program (e.g., Hex Workshop or Hex Fiend).

3 The Task

You join Happy Happy Corporation to take over Alice's job. Alice won the lottery and decided to quit and travel the world. While this is great for Alice, her departure was sudden and the transition may be rocky.

Alice was in the process of writing an encryption/decryption tool for Happy Happy Corporation. She wrote comments while coding but she hadn't gotten around to finishing.

Before she left, Alice encrypted some files that are super secret and super important for the company (you have no idea why) using a variety of methods. The company insists that you finish writing the tool that she was writing and decrypt the encrypted files to recover the super secret data.

No one else has any idea what Alice was doing, and she won't be answering any phone calls or emails. Fortunately, she did leave a few notes on her computer:

```
CTR key:
67cd21b35eced6d3268c9fc47f512ad5436ca1843ab08988810ff7f3a3f4bb05

CBC key:
cc2bf484eca8a1b2fe8add4050a6c1112edfe82e21a1c6edbc48a996368c0913

GCM key:
f7160d8db15d646f6702aea77d7828a052cc8ed0b767bb838a6b147425b6bf9d

The RSA private key (encoded export) is in boringfilenothingtoseehere.
```

Happy Happy Corp. further tells you that Alice's machine has stored an encrypted copy of Alice's salary (encrypted via `encryptAESCTR()`) and that you'll make the same amount that's listed in the file. Alice never had the key to decrypt that file, but she kept an encrypted copy that she could pass to HR (who has the decryption key) as proof that they promised her that salary.

3.1 Implement Decryption Functions

(a) [**4 points**] Complete the method `decryptAESCTR()` so that the program can decrypt messages encrypted with `encryptAESCTR()`. You will want to write code that is similar to the example encryption code (but for decryption). Links to documentation are provided. DO NOT attempt to write your own crypto library here—it will take you a lot longer, and this course does not prepare you to securely implement cryptography libraries. You should be able to decrypt `encrypted-with-AES-CTR-mode` (the decrypted message will make it obvious when it's working).

(b) [**4 points**] Complete the method `decryptAESCBC()` so that the program can decrypt messages encrypted with `encryptAESCBC()`. You should be able to decrypt `encrypted-with-AES-CBC-mode`.

(c) [**2 points**] Complete the method `decryptAESGCM()` so that the program can decrypt messages encrypted with `encryptAESGCM()`. You should be able to decrypt `encrypted-with-AES-GCM-mode`.

(d) [**12 points**] Complete the method `decryptHybridRSA()` so that the program can decrypt messages encrypted with `encryptHybridRSA()`. You should be able to decrypt `encrypted-with-hybrid-RSA`.

3.2 Get a Higher Salary

(e) [**20 points**] You know that the plaintext information that was encrypted about Alice's salary was in `original-salary-plaintext`. You know that the plaintext was encrypted via `encryptAESCTR()`. You don't have the key, so you'd like to develop a way to alter a ciphertext encrypted via `encryptAESCTR()`—without knowing the key—so that it decrypts to a different plaintext (of the same length) of your choosing.

Implement the generic attack in `AlterCTREncryptedFileWithoutKey.java`. You can use `original-salary-plaintext` and `target-salary-plaintext` and any key you choose to test that your attack is working. Your attack must be generic (i.e. work with any key and any two plaintext files of equal length).

3.3 Further Writeup Questions

Answer the following questions and turn them in as a PDF file.

(f) **[1 points]** Change a few of the bytes in `encrypted-with-AES-GCM-mode` and then decrypt it. What happens? Why?

(g) **[2 points]** We're going to look at some of the file sizes produced via the different functions.

(a) Encrypt `original-salary-plaintext` using `encryptAESCTR()`. How many bytes is it?

(b) Encrypt `original-salary-plaintext` using `encryptAESCBC()`. How many bytes is it?

(c) Why is the file that was encrypted using CBC larger than the file that was encrypted using CTR?

(d) Encrypt `original-salary-plaintext` using `encryptAESGCM()`. How many bytes is it?

(e) Why is the file that was encrypted using GCM mode larger than the file that was encrypted using CTR mode?

(h) **[3 points]** Carol sends you messages that were supposedly padded with PKCS5/PKCS7 (discussed in lecture) padding prior to being encrypted with AES-CBC. Imagine that you decrypted the ciphertext using your own crypto implementation (not a library that automatically checks and strips the padding) and recovered the following last blocks of decrypted content (in hexadecimal). For each message, either indicate that the message was improperly padded or indicate how many bytes of padding were inserted after the actual message content ended:

(a)
```
[previous blocks]
0x AA AB BA 11 FF E3 21 50 D3 CD 33 43 10 10 06 01
[end of decrypted content]
```

(b)
```
[previous blocks]
0x 0A 10 0F 0E 0D 0C 0B 09 08 07 06 05 04 03 02 01
[end of decrypted content]
```

(c)
```
[previous blocks]
0x 10 12 A6 FF 11 B4 E3 23 50 1C 11 55 04 03 03 03
[end of decrypted content]
```

(d)
```
[previous blocks]
0x 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[end of decrypted content]
```

(e)
```
[previous blocks]
0x 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
[end of decrypted content]
```

(f)
```
[previous blocks]
0x 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 04
[end of decrypted content]
```

4 Turn-In

Turn in to Gradescope:

1. Your edited `EncryptionTool.java`. Make sure your code is commented and well formatted.
2. Your edited `AlterCTREncryptedFileWithoutKey.java`. Make sure your code is commented and well formatted.
3. A PDF with the answers to 3.3.

Autograder notes:

- You do not need to turn in the `.jar` files.
- The autograder will fail to run if the `.java` files do not compile.
- The autograder will fail to run if it does not find both of the `.java` files. You can turn in either of the files handed out to you if e.g. you want to see how you're doing as you're completing the assignment but haven't worked on both files. (The files we handed out to you will get 0 points, but the autograder will run on them.)
- You need to turn in **all files again if you resubmit, including your PDF**.
- Submit the files directly (e.g., not as a `.zip`).

5 Notes

1. We provide enough sample code that the Java syntax shouldn't trip you up too much. (Feel free to chat with us for additional guidance.) However, you might want to make sure that you understand the difference between the `javac` command (to compile a `.java` source code file) and the `java` command (to run a `.class` file with Java bytecode—notice that the extension `.class` isn't actually included in the command). If of interest, search for some basic tutorials explaining Java, Java bytecode, etc.

2. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/package-summary.html

3. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/package-summary.html

4. https://docs.oracle.com/en/java/javase/11/security/java-security-overview1.html

5. http://commons.apache.org/proper/commons-codec/apidocs/org/apache/commons/codec/binary/Hex.html

6. Excerpted from (3):

> The JDK defines a set of APIs spanning major security areas, including cryptography, public key infrastructure, authentication, secure communication, and access control. The APIs allow developers to easily integrate security into their application code.
>
> The APIs are designed around the following principles:
>
> **Implementation independence:** Applications do not need to implement security themselves. Rather, they can request security services from the JDK. Security services are implemented in providers (see the section Security Providers), which are plugged into the JDK via a standard interface. An application may rely on multiple independent providers for security functionality.
>
> **Implementation interoperability:** Providers are interoperable across applications. Specifically, an application is not bound to a specific provider if it does not rely on default values from the provider.
>
> **Algorithm extensibility:** The JDK includes a number of built-in providers that implement a basic set of security services that are widely used today. However, some applications may rely on emerging standards not yet implemented, or on proprietary services. The JDK supports the installation of custom providers that implement such services.

7. Remember when this came up in lecture? From the documentation:

> A transformation is of the form: "algorithm/mode/padding" or "algorithm". In the latter case, provider-specific default values for the mode and padding scheme are used.

That means that what code does could vary from system to system if the default provider used has different (e.g., block cipher mode / padding) defaults. Ouch.

From older documentation:

> It is recommended to use a transformation that fully specifies the algorithm, mode, and padding. By not doing so, the provider will use a default.

For example, the SunJCE and SunPKCS11 providers use ECB as the default mode, and PKCS5Padding as the default padding for many symmetric ciphers.

This means that in the case of the SunJCE provider:

```
Cipher c1 = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

and

```
Cipher c1 = Cipher.getInstance("AES");
```

are equivalent statements.

Note: ECB mode is the easiest block cipher mode to use and is the default in the JDK and JRE. ECB works for single blocks of data when using different keys, but it absolutely should not be used for multiple data blocks. Other cipher modes such as Cipher Block Chaining (CBC) or Galois/Counter Mode (GCM) are more appropriate.

8. We are using BC by pointing to it when we compile and explicitly loading BC as a provider. We could also install and configure it for the whole system, which would allow us to e.g. make BC the default crypto provider.

9. Some installations of the JDK—particularly on Windows—seem to run into problems where `JAVA_HOME` isn't correctly added to the environment variables and `%JAVA_HOME%\bin` isn't correctly added to the path/classpath. If you see this problem (i.e., can't run `java` on the command line after installation), I would suggest checking those details and/or coming to office hours.