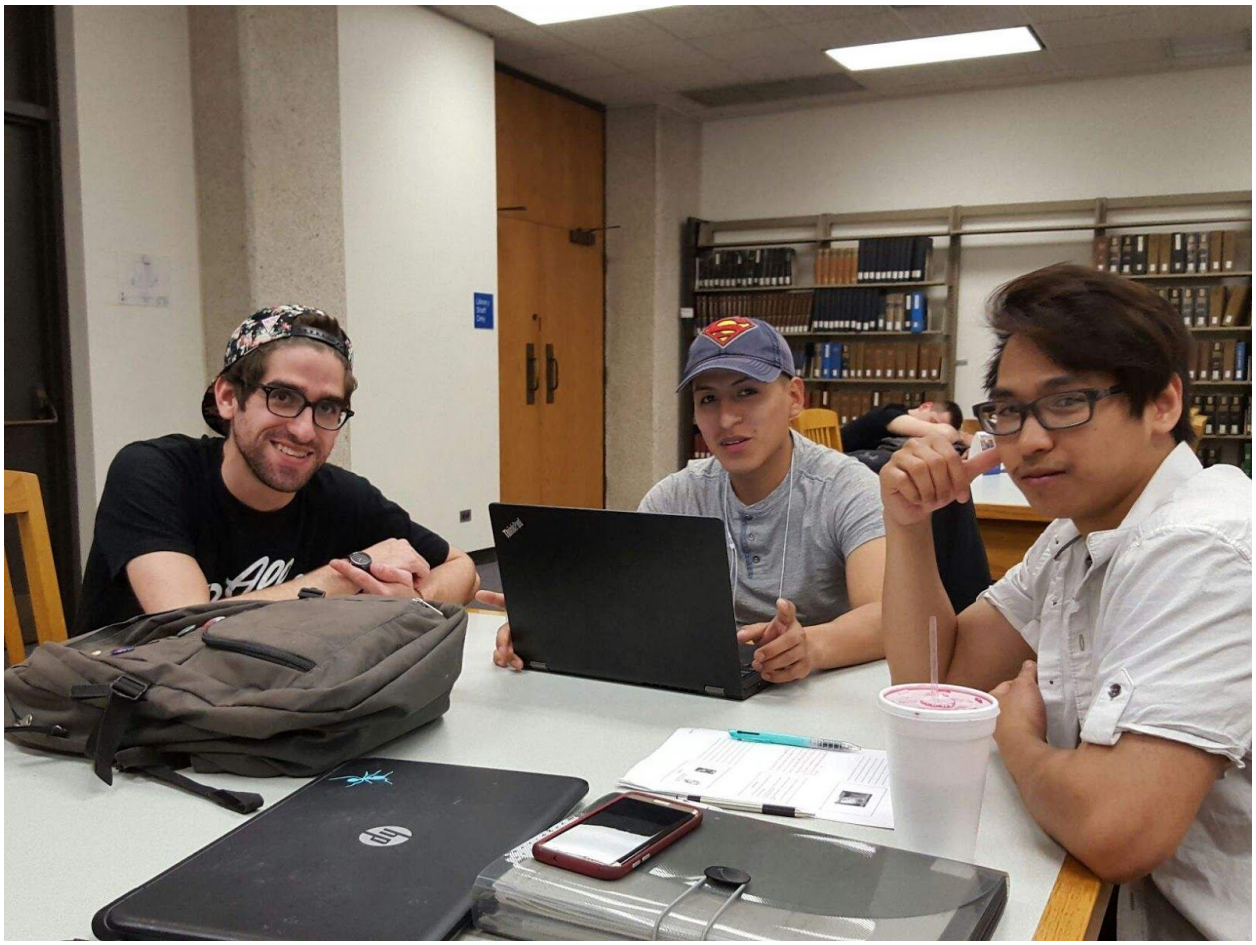
CS-207-1: Programming II Spring 2017 Northeastern Illinois University Research Lab: AES Encryption Due: Wednesday, 04/26/17 at 12:00 p.m.



Group Members: Carlos Quito, Stevie Scheid, Chy Sombath

Part A: Investigating the AES Encryption Algorithm

Use Google to investigate the AES encryption algorithm and answer the questions below.

Question #A.1

Who developed the AES Encryption algorithm? Provide some historical background on the AES Encryption algorithm.

A: In 1997 there was a public call for new kinds of data encryption by the National Institute of Standards and Technology(NIST). AES stands Advanced Encryption Standard, and the United States had a lot interest in this competition. Within the span of a year, or 1998, NIST had 15 algorithm submissions to choose from. Each submission was tested to see how effectively it encrypted data. In 1999 NIST narrowed down the decision to 5 finalist. However it took another 2 years before finally announcing on November 26, 2001 that the Rijndael algorithm was chosen as the AES.

Question #A.2

Describe some of the key aspects of the AES algorithm and its implementation.

A: Three key lengths 128(standard application 10 rounds)/192(12 iterations)/256(14 iterations). The higher the iterations the better the security. AES is used on almost everything that requires data encryption. It is also cost efficient

Question #A.3

Is AES encryption robust?

A: Yes it is an robust format. It is the most important symmetric algorithm currently used. It is also used by the NSA for classified data(usually used with longer key lengths and iterations).

Part B: Getting Started with AES in Java: Making an encrypt method

Question #B.1

Find the Cipher class in the Java 8 docs. In what package does it live? Import this package and create a class named AESEncryption. Create the main method, but leave it empty for now. Create a method named encrypt that does not return anything and does not take any parameters.

A: The Cipher class in the Java 8 docs lives in import javax.crypto.Cipher package so naturally we will import the javax.crypto.* package.

Question #B.2

The Cipher class refers to the different algorithms as transformations. However, the constructor to create a Cipher object is more complex than we would like. Instead, we can create a Cipher object using the static method getInstance. Click on the getInstance method that takes a String parameter. It provides you with a link to documentation for the String values you should use for different algorithms. What do you use for the AES algorithm? Use this static method with the correct parameter to create a Cipher object.

A: The String transformation for AES is “AES”. It will create an object using the AES algorithm, which takes 128-bit block cipher supporting keys 128,192,256 bits.

Question #B.3

Look at the getInstance method again. Does it throw any exceptions? What are they? Are they checked or unchecked? What happens if you try to compile your code at this point? Do you need to import any packages for these exceptions? Handle each exception by printing out a useful error message for the user. Compile your code.

A: Yes, it throws two exceptions which are not checked. They are NoSuchAlgorithmException and NoSuchProviderException. If we compile the code without handling the exception we get the message:

1)AESEncryption.java:15: error: <identifier> expected
catch(NoSuchAlgorithmException)

2) AESEncryption.java:19: error: <identifier> expected catch(NoSuchPaddingException)

In order to handle the exceptions we must import the special packages below:

import javax.crypto.NoSuchPaddingException, which has already been taken care of above. and

import java.security.NoSuchAlgorithmException.

Both packages extend from import.java.security.GeneralException, however this does not handle NosuchPaddingExcpetion which is why we imported 2 packages.

Question #B.4

Next, you need to initialize your Cipher object. In the Cipher class find an initialization method (haha, there are quite a few!!). All of them take an int as the first parameter. This int represents the operation mode (i.e. are you encrypting, decrypting, etc). Does the Cipher class have any constants that look like they might be useful? Pick the right one!

The Cipher class has several methods that uses an int as a parameter and initialize the cipher, this method would be the inti() method. Since we are at the stage where we

want to encrypt the data we have in hand, we will be using the constant, ENCRYPT_MODE.

Question #B.5

Next, regardless of which initialization method you use, you have to provide a Key object - so we have to generate a key before we can initialize the Cipher object. To help you with this, you have been provided with the code below. Look up SecretKey in the Java 8 docs - what is it? What does it inherit from?it (i.e. class, abstract class, etc). What does it inherit from? Do any of the methods in the code below throw exceptions? Where should this code go relative to the creation of your Cipher object? Then, call the initialization method that uses two parameters to initialize the Cipher object. Compile!!

A: The SecretKey interface is a secret (symmetric) key which inherits from the Key interface. This part of the code which generates a key object must be created before initializing the Cipher. The getInstance() method throws two exceptions, which we have previously pointed out. We also notice that the init() method throws an InvalidKeyException so we will handle that too.

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

Question #B.6

Which method in the Cipher class looks like it is used to encrypt or decrypt data? Pick the method that takes one parameter. What is the type of this parameter? What does the method return? In order to use this method, we will need to be able to read from the file that we are trying to encrypt and output an encrypted version of it (and save the key!).

A: The methods which are used to encrypt and decrypt data are doFinal() which take different parameters but the one we are interested in takes a byte[] input as the single parameter, and then returns a byte[].

Part C: Getting Started With Files Since you will be working with encrypting files (and storing a key in a file), it is important to research the necessary Java 8 API classes for working with files.

Question #C.1

You need one specific Java package imported in order to work with File objects. What is it? Import this package before your class definition. Each file that you read from needs

to be represented by a File object.

A: In order to read files we need to import the java.io.* package.

Question #C.2

Download the data.txt file from the NeededFiles.zip file and save it in the same folder as your Java file. Look at the data in this file so that you can see what it contains (for when you decrypt its encrypted version). Do you recognize this famous speech? Who gave this speech (also, when and where)?

A: Yes, it was Martin Luther King Jr. On August 28,1963 at Lincoln Memorial Washington D.C

Question #C.3

You will need to be able to read from your File objects. To do this, you will be using the FileInputStream class. Why is it necessary to use the FileInputStream class instead of the Scanner class (Hint: Look at the method that you have to use in the Cipher class to encrypt/decrypt data)? Find the constructor for theFileInputStream class in the Java 8 docs that takes a File object as a parameter. What type/name of the exception does it throw? Why do you need to know this before you create a FileInputStream object? Create a FileInputStream object for your File object. Make sure to handle the FileNotFoundException. Hint: Sure hope you're using multiple catch blocks as opposed to a bunch of separate try-catch statements. Compile.

A: We won't be using the Scanner class because we will be encrypting our data in form of bytes. Since the FileInputStream method creates an input byte[] from a file it is exceptionally useful when encrypting and decrypting.

The constructor FileInputStream(File file) throws a FileNotFoundException as well as a SecurityException if a security manager exist. It's important to know what kind of exception is being thrown because we are using multiple catch blocks and want to make sure we aren't creating extra catch blocks for a similar exception that are being handle from earlier.

Question #C.4

The read method of the FileInputStreamclass requires an initialized byte array for a parameter. Create and initialize a byte array. How do you know the length of the array? (Hint: Read the Java docs for the File class. Remember, in order to declare and create an array, you need an int for the size - you may find casting helpful!). Make sure to handle any exceptions and print out helpful error messages. Compile.

A: In order to get the length of text file we used the getlength() method within the File class. Then we cast it to an int. If the number is bigger than what an int can hold we

have will have a NumberFormatException.

Question #C.5

Coding Only Read from your data.txt file. Make sure to handle exceptions. Compile.

Question #C.6

Now that you have the data from the file, use the method from #B.6 to encrypt the data (make sure to assign it to the correct return type). Does this method throw any exceptions? What are they? Do they all inherit from the same Exception as any other Exception you have already caught? If so, find a way to use this superclass exception for these particular exceptions!

A: Yes it throws an IllegalBlockSizeException and a BadPaddingException, these all inherit from the GeneralSecurityException.

Part D: Writing the Encrypted Data

Question #D.1

You need to write the encrypted byte array to a file using a FileOutputStream object. Why is it necessary to use the FileOutputStream class instead of the Scanner class? Create a FileOutputStream object and write the encrypted data to a new file (yes, you will need a method from the FileOutputStream class to write the data). Compile.

A: For the same reason we use the FileInputStream, the Scanner class would not be capable of manipulating file inputs into a byte[] which is what we are using by encrypting.

Question #D.2

In order to decrypt, you will need the secret key that you generated - so you have to save it. Find the Key interface in the Java docs. Is there a method that returns the encoded key? What is the type that is returned from this method? Create a FileOutputStream object and use the appropriate method to write the key to a new file. Close all your files. Compile.

A: Yes, the method is getEncoded() and it returns the key in its primary encoding format. The return type of the returned key is a byte[].

Question #D.3

Coding Only: Run your code and call the encrypt method in the main method. Verify that

the contents of the encrypted file look like nonsense and that the contents of the file for the key look like nonsense.

Part E: The decrypt method Decrypting an encrypted file is very similar to encrypting, however you need to supply the unique key for decrypting (which is why we saved it in a file).

Question #E.1 - Coding Only

Create a method named decrypt that does not take any parameters and does not return anything. Using the same methodology as for Part D, read (and save using byte arrays) the encrypted file and the key. Since you already have a key, it is easy to create a key object using the following syntax (where keyBytes is the name of the byte array for your key):

SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");

Yes - compile. Compile frequently.

Question #E.2

Create a Cipher object the same way that as in the encrypt method. However, when initializing the Cipher object, which operational mode should you use? After initializing, use your Cipher object to perform the decryption (same method call as in the encrypt method). Create a FileOutputStream object and use the appropriate method to write the decrypted data to a new file. Close all your files. Compile. Comment out your encrypt method in main and call the

decrypt method.

A: We still use cipher.GetInstance(“AES”) however we will use static in DECRYPT_MODE to initialize cipher to decryption mode.

Question #E.3 - Coding Only

At this point, you have two methods: encrypt() and decrypt(). However, your code is not user-friendly. It is hard-coded to only read from specifically-named files. And you have to uncomment and comment the methods that you want to use. Write code to make this user-friendly by implementing the following changes:

1. Prompt the user to enter E for encryption and D for decryption.

2. If they enter E for encryption, use the encrypt() method and prompt them to enter the complete file name that is to be decrypted.

3. If they enter D for decryption, use the decrypt() method and prompt them to enter the complete file name that is to be decrypted, followed by a prompt to enter the complete file name for the key file.

Part F: Final Summary Whenever you complete a project, it is important to assess what you think went well and what you need to improve on.

Question #F.1

What was the most challenging part of this research lab for your group?

The most challenging part of this project was creating the encrypted method. There was a lot of times where the code would compile and work but the program wasn't doing what we necessarily wanted to do.

Question #F.2

What did your group learn/find the most useful by doing this research lab?

A: What was really useful about this research lab is that it help solidify Exception handling as well as inheritance whether it was from the File object class or the Interface Secret Keys.

Question #F.3

How was this lab different from the previous labs for your group.

A: The biggest difference about this lab then the previous Lab was that we really understood why we need to import packages. It also helped us see how we can really benefit from javadocs as long as we understand inheritance, casting, and object creation.

Question #F.4

What was the most fun aspect of doing this research lab?

A: The most fun aspect of this lab was finishing the lab and seeing the code compile, encrypt and decrypt our data.text file.