SE448: Information and Cyber Security	**Recitations 2–3**
Semester 2 5786	**11 March - 18 March 2026**
Lecturer: Michael J. May	**Kinneret College**

Using DES and AES in Java

In this recitation you will put together a simple application which will perform DES and AES encryption and decryption using the Java cryptographic libraries. The goal of this recitation is to understand how the Java cryptographic libraries work and try some experiments with CTR, CBC and initialization vectors.

1 Getting started using the Java crypto libraries

The tool we will build this week will use the Java cryptographic libraries which are built into the platform. In Java, all ciphers are implemented using a single `Cipher` class that provides implementations by name. To use the cipher, you use a static method in the `Cipher` class called `getInstance` that takes the name of the algorithm you want in the following formats:

- algorithm/mode/padding (*e.g.* DES/CBC/NoPadding)
- algorithm (*e.g.* DES)

The instance you get back first needs to be initialized using the `init` method. A sample call to initialize the cipher for decryption would be:

```
crypto.init(Cipher.DECRYPT_MODE, key);
```

For encryption, you'd write:

```
crypto.init(Cipher.ENCRYPT_MODE, key);
```

In both cases, you provide a key that must be of the type `Key`. We'll use `SecretKeySpec` for our key object. You define the key object by providing a set of bytes for the key and specifying the cipher that you're going to use it for. For instance, if we're going to use DES, we'd write:

```
SecretKeySpec key = new SecretKeySpec(keyBytes.array(), "DES");
```

Here `keyBytes` is a `ByteBuffer`, so we convert to a `byte[]` using `array`.

Since as of the first lecture we won't have finished learning about cipher modes, we'll start today by just using *Electronic Code Book (ECB) mode* for the cipher. As a next step, we'll move onto *Cipher Block Chaining (CBC)* mode, a much more secure alternative.

Note: ECB mode is highly insecure and should not be used for any cryptographic application.

1.1 What to do: Basic Java encryption

As a first step, build a Java command line program that performs the following operations:

1. Uses a fixed DES key to encrypt an 8-byte text string using ECB mode. The tool should print out the resulting cipher text to the command line.
2. Uses the same fixed DES key to decrypt the cipher text from the previous step. The tool should print out the resulting plain text to the command line.

2 Why not to use ECB

The tool we built in the previous section encrypts text using ECB in DES. ECB is not a good mode to use in real life for a number of reasons. To demonstrate one of the problems with ECB, I'll demonstrate an ECB GUI tool that I built (you will not need to built it). The tool is shown in Figure 1:

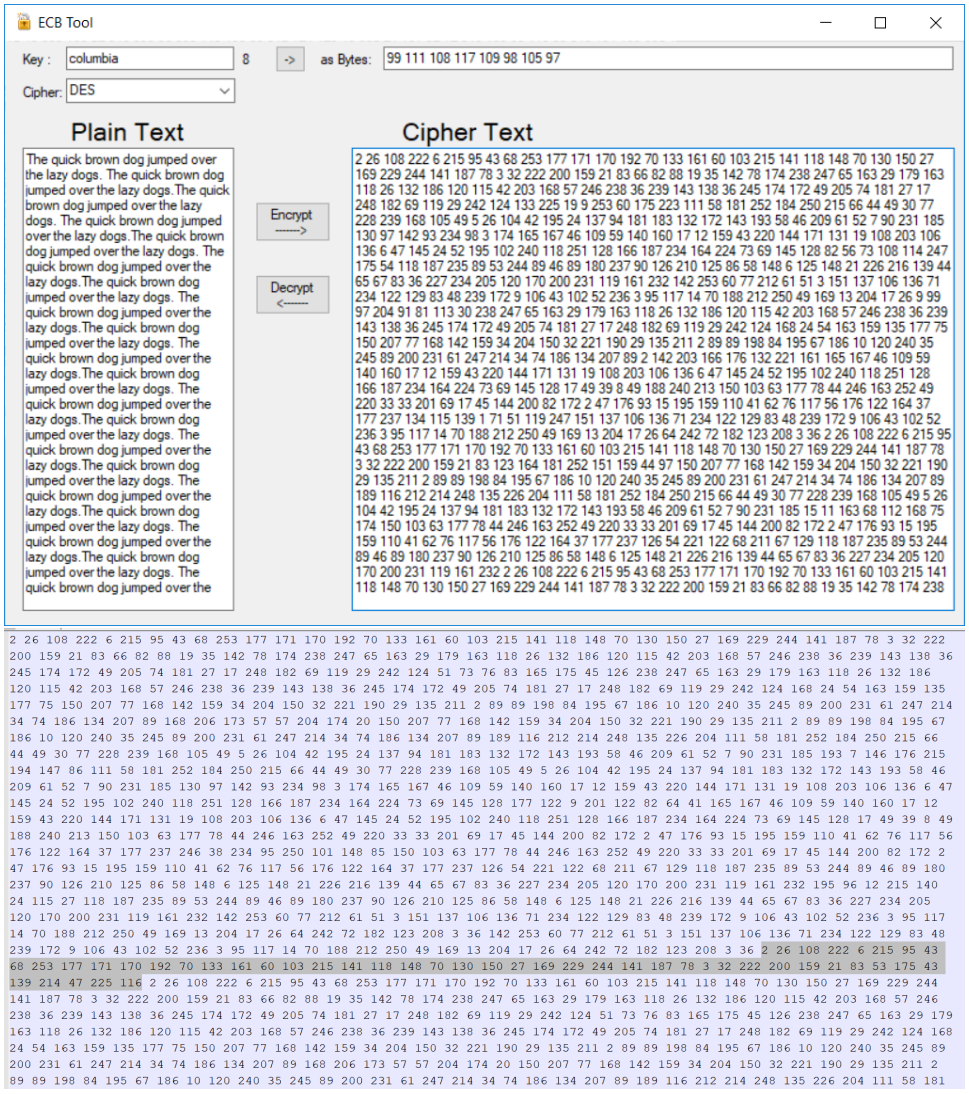

Figure 1: Screen shot of DES under ECB with ciphertext problem highlighted

The tool encrypts the text on the left side of the tool with the key shown in the top part ("columbia"). The cipher text is shown in decimal format (bytes) on the right side.

To understand some of the problems with ECB, let's examine the ciphertext shown in the figure above. It was produced using the following steps:

1. Secret key "columbia".
2. Repeating plaintext: "The quick brown dog jumped over the lazy dogs. The quick brown dog jumped over the lazy dogs."

The ciphertext output is shown in the lower half of Figure 1 with the problem highlighted.

The issue of repeating patterns holds for AES as well - it's an ECB problem, not a cipher problem. For example, Figure 2 shows a similar example with AES under ECB.

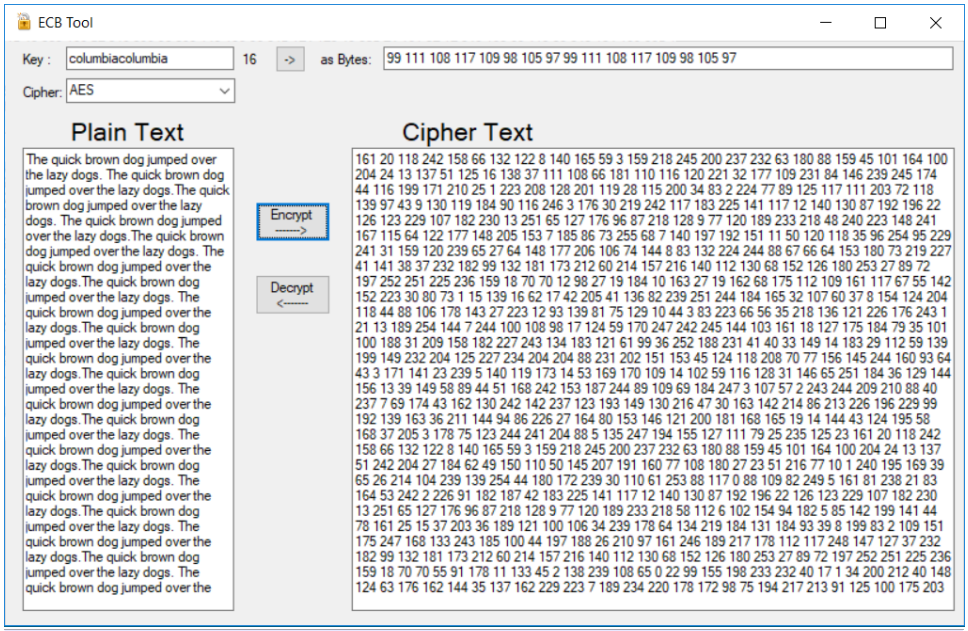

Figure 2: AES ciphertext example using AES

3 Next Step: DES Encryptor/Decryptor Tool

For our next task, we'll improve the code from the previous sections in a few ways:

1. We'll support Cipher Block Chaining (CBC) mode
2. We'll add support AES

Your job will be to develop a Java application that encrypts and decrypts files. This is meant as an exercise in using the cryptographic classes so you will get a good idea of how they work.

Since many of you have not used Java GUIs before, we'll use the command line only for this task. The command line application must take the following parameters:

1. The cipher suite to use. Support the following options:
   - AES/CBC/NoPadding
   - DES/CBC/NoPadding
   - AES/CBC/PKCS5Padding
   - DES/CBC/PKCS5Padding
2. The key to use, encoded as a hexadecimal string
3. The IV to use, encoded as a hexadecimal string
4. Whether to encrypt or decrypt
5. The name of the input file
6. The name of the output file.

DES uses a single key length (64 bits), but AES uses different key lengths (128, 192, or 256). You'll find that the cipher modes that don't have padding have specific requirements for input length. We'll work on that a bit more in the experiments below.

For convenience, output the cipher text in decimal bytes with spaces to delineate them, as is shown in the figure above.

4 Using CBC in Java

We will improve the security of the encryption by adding an initialization vector to the encryption and using the *Cipher Block Chaining (CBC)* mode of symmetric encryption. We set the mode for the cipher accordingly and add an array to the initialization vector property of the `Cipher` instance. Once that is done, we can use the tool as before to encrypt and decrypt.

5 What to do

1. Start with the "empty" tool provided and make it work with DES and CBC. You can use the code from the DESJava project to get things going quickly.
2. Perform the experiments below in Section 6.
3. Ensure the tool supports AES (use the `Cipher` class and "AES/CBC/PKCS5Padding". The change is not that large provided that you wrote the code correctly for DES. You'll just need to make sure that you change the cipher init and the key object definitions.
4. Perform the experiments below in Section 6. Since the key size and IV size are different in AES, you'll need to choose your own keys and IVs (the ones below are only 8 bytes long).

6 CBC Experiments

To help understand how DES and CBC work, let's perform the following experiments. For these experiments, use the DES/CBC/PKCS5Padding cipher suite.

6.1 Experiment 1: IV

1. Set the key to be: "746F7261626F7261". Set the IV to be: "756E74696C6F6E65".
2. Encrypt a file with the contents: "This is the song that doesn't end, yes it goes on and on my friend."
3. What do you get for cipher text?
4. Change the IV to be: "756E74696C74776F". Decrypt the cipher text. What happened? What part of the message got corrupted?
5. Change the IV to be: "71776565726B6965". Decrypt the cipher text. What happened? What part of the message got corrupted?
6. Return the IV to "756E74696C6F6E64". Note that the last number is 4 instead of 5 as it was originally. Decrypt the cipher text. What happened? What part of the message got corrupted?

Based on the above experiment, can you tell how many characters in the string are included in a single block?

6.2 Experiment 2: Keys

1. Set the key to "616E74696C657374". Set the IV to be: "7479706963616C79".
2. Encrypt a file with the string: "Come on along or go alone, he's come to take his children home."
3. What do you get for the cipher text?
4. Change the key to be: "766963746F726579". Decrypt the ciphertext file. What happened? Why?
5. Change the key to be: "616E74696C657375". Decrypt the message. What happened? Why? (Hint: Look at the last hexadecimal character)
6. Change the key to be: "616E74696C657376". Decrypt the message. What happened? Why? (Hint: Look at the last hexadecimal character)
7. Can you change any other characters in the key and still get the correct decryption?

Based on the above experiment, can you tell how Java converts from a 64 bit hexadecimal value to a 56 bit key?

6.3 Experiment 3: Plain text

1. Set the key to be "6E6F706576657279". Set the IV to be: "696E74657266616E".
2. Encrypt a file with the string: "Ripple in still water. There is no pebble tossed nor wind did blow"
3. What did you get for cipher text?
4. Change the last letter of the cipher text to be "v" (*i.e.,* it should read "blov"). Encrypt the file. How many bytes changed in the cipher text?
5. Change the last letter back to "w". Change the first character to be "S" (*i.e.,* is should be "Sipple"). Encrypt the file. How many bytes changed in the cipher text?

6. Change the first letter back to "R". Change the first "T" to be "U" (*i.e.,* the fifth word should be "Uhere"). Encrypt the file. How many bytes changed in the cipher text?

Why does this happen? How does this relate to how CBC works?

6.4 Experiment 4: Cipher text

1. Set the key to be "6265636B6F6C6E79". Set the IV to be: "6672756D706C6579". Convert the key and IV to be bytes.
2. Encrypt a file with the contents: "You enter a darkened room and sitting there in the gloom is Dracula.".
3. What did you get for the cipher text?
4. Change the last byte of the cipher text to be 19 (instead of 18). Decrypt the cipher text. What happened?
5. Change the last byte back to 18. Change the first byte to be -93 (instead of -94). Decrypt the cipher text. What happened?
6. Change the first byte to be -95. Decrypt the cipher text. What happened?

How sensitive is Java's DES decrypter to changes or errors in the cipher text? Can you figure out the block size for the message based on this?

6.5 Experiment 5: Padding

For the following experiments we're going to use the DES/CBC/NoPadding cipher suite.

1. Set the key to be "6265636B6F6C6E79". Set the IV to be: "6672756D706C6579".
2. Encrypt a file with the contents: "Balloons" What happens?
3. Encrypt a file with the contents: "Balloons bust" What happens?
4. Encrypt a file with the contents: "Balloons busting" What happens?

Retry the above steps with the DES/CBC/PKCS5Padding cipher suite. Try the above experiments again. What happens?

What can you tell about the role of the padding algorithm in encryption based on the results?

