AffineEncryptor.docx

Introduction

This assignment is about something that's been in the news off and on -- encryption. Don't worry the encryption that you will perform is trivial and can be broken instantly by the NSA. However, you will learn some math and another use of the term "hacking."

Ciphers

In cryptography, a cipher is a method of transforming a text in order to conceal its meaning. Possibly the simplest cipher is a shift cipher where the characters in the text are shifted (forward or backwards) to another position in the alphabet. There's just one sticking point: What happens when a character is shifted "off the end." For example, if there are 256 characters (as in the extended ASCII character set) what happens when character x is shifted to position 258? Answer: wrap back to the beginning, in this case to position 2. This is known as performing modulo arithmetic where the modulus is 256.

The Caesar Cipher

Julius Caesar is actually credited with using this cipher with a shift key of 3. To encrypt a letter, we first store its numeric value in an `int` type and then add the key value, and take the result modulo 256.

Here is the code for this:

```
public int encrypt(int b) {
   return (b + key) % 256;
}
```

The code appears in the textbook in Chapter 20.2 in the class CaesarCipher.java. The curious thing about this class is that there is no decrypt() method. The reason for this is the simple fact that the encrypt() algorithm can perform decryption if you simply use the negative value of the original encryption key.

Import AffineEncryptor.zip and run the Caesar Cipher by highlighting CaesarEncryptor.main(), right-click and Run As => Java Application. Bring focus to the Console window (just click on the window and enter the following:

```
Input file: original.txt
Output file: encrypted.txt
Encryption key (+/- 1..255): 13
```

The program creates a new output file that is not immediately visible. Highlight the project name (CaesarEncrypter) and right-click => Refresh. A new file named 'encrypted.txt' will appear. Its contents will look like:

zrr�-n�-zvq{vtu�-n�-�ur-ynxr;

Now reverse the process. Run the program again and at the prompts enter these values:

```
Input file: encrypted.txt
Output file: decrypted.txt
Encryption key (+/- 1..255): -13
```

Refresh the project files again and you will see 'decrypted.txt'. Its contents should be:

meet at midnight at the lake.

Note the "division of labor" between the two classes. CaesarEncryptor gets input from the user, creates input and output streams, and closes the files. CaesarCipher reads the contents of the input stream and writes to the output stream.

Hint: Make sure you can get the project to work. If you cannot, you should not advance to the Affine Cipher.

The Affine Cipher

The affine cipher is just a step above the shift cipher in sophistication. Instead of just shifting a letter by addition, let's apply two operations: multiplication and addition. Of course, we still must do this using modulo arithmetic to stay in the range 0 to 255.

In the affine cipher there are two values that constitute the encryption key: the multiplicative value (aKey) and the shift value (bKey). The encryption looks like this:

```
public int encrypt(int x) {
   int val = ((aKey * x) + bKey) % 256;
   return val;
}
```

Note: The aKey must NOT be a multiple of two. That's because the aKey must be relatively prime to the modulus used.

Now decryption is not so simple. It is only possible if we have chosen a good value for a. That's all because of that modulus operation. I won't go into the math, but the number a must be relatively prime (no common factors) to 256. Many numbers are relatively prime to 256, but let's stick with these: 1, 3, 5, 7, 11. In order to perform decryption with any of these numbers, you must know its multiplicative inverse modulo 256. This is easier to explain by example than by words.

Number	Inverse modulo 256	Explanation/Final result
1	1	1*1 = 1. 1 % 256 = 1.
3	171	3*171 = 513. 513 % 256 = 1.
5	205	5*205 = 1025. 1025 % 256 = 1.
7	183	7*183 = 1281. 1281 % 256 = 1.
11	163	11*163 = 1793. 1793 % 256 = 1.

The 1 as the final result indicates that we have found the unique multiplicative inverse for each number.

So mathematically if encryption goes like this:

y = (aKey*x + bKey) mod 256

Then decryption goes like this:

x = ((y - bKey) * (inverse of aKey)) mod 256 // that's multiplicative inverse of a mod 256

Hint: You must translate this into the decrypt() method.

Hacking

Hacking is an overloaded word in computer science. It can mean:

- Breaking the security of a system or network
- Getting some code to work by experimenting
- To write computer programs for enjoyment.

Let's concentrate on the last two meanings.

In this assignment, you will be given two classes that implement the Caesar cipher:

- CaesarCipher.java
- CaesarEncryptor.java.

Your job is to hack these files into their affine equivalent:

- AffineCipher.java
- AffineEncrpytor.java.

Because affine encryption and decryption are so different, the AffineEncryptor should start off by displaying the following to the user for encryption;

```
Enter 1 or 2:
 1 - Encrypt
 2 - Decrypt
1
Input file: original.txt
Output file: encrypted.txt
Key A (must be odd): 11
Key B (+/- 1..255): 27
```

For decryption, the program displays similar prompts, but the responses are different:

```
Enter 1 or 2:
 1 - Encrypt
 2 - Decrypt
2
Input file: encrypted.txt
Output file: decrypted.txt
Key A (must be odd): 11
Key B (+/- 1..255): 27
```

Notes: a and b values remain the same but for decryption the multiplicative inverse of a must be used.

My original.txt was:

```
meet at midnight at the lake.
```

The encrypted.txt is:

```
Êrr{F{ÊžgÕžˆ"{F{"r{¿F´r
```

The decrypted.txt is the same as the original.txt. When working with Eclipse you may have to refresh its list of files by right clicking on the project name and from the pop-up menu selecting "Refresh."

Hints:

- Don't forget to pass both the aKey and bKey values to the AffineCipher constructor.
- Validate that the aKey is odd and the bKey is in range (0 .. 255)
- Add a separate decrypt() method to AffineCipher
- Alter AffinceEncryptor.main() to issue the prompts shown.

Big Hint: Use BigInteger

There is a Java class that is designed to handle the huge numbers used for modern encryption algorithms. It is called 'BigInteger'. It has a nice method for calculating the multiplicative inverse of an integer named 'modInverse'. The invoking object is the one whose inverse is calculated as per the parameter value (the parameter is the modulus).

The tricky part is that the aKey and the modulus (256) must be "blow up" to BigInteger values and the return value "shrunk down" to an int value. Here are the lines in my AffineCipher constructor that do this:

```
BigInteger aKeyBig = new BigInteger(Integer.toString(aKey));
inverseOfaKey =  aKeyBig.modInverse(new BigInteger(Integer.toString(256))).intValueExact();
```

It is worth looking up the methods used in Integer and BigInteger classes.

Assignment

Download CaesarEncryptor.zip from Canvas and import the archive file into Eclipse. Run the program so that you can vouch that it performs encryption and decryption properly.

Make a copy of the project (in Eclipse just right click of the project and select "Copy" then "Paste") with the new name AffineEncryptor. Now change file and class names to AffineEncryptor.java and AffineCipher.java. Now hack (in a good way) the shift code into the affine code using the information in this document.

When finished attach AffineEncryptor.java and AffineCipher.java to your Canvas submission. Enter the "what I learned" observation.