Computer Security Assignment 1

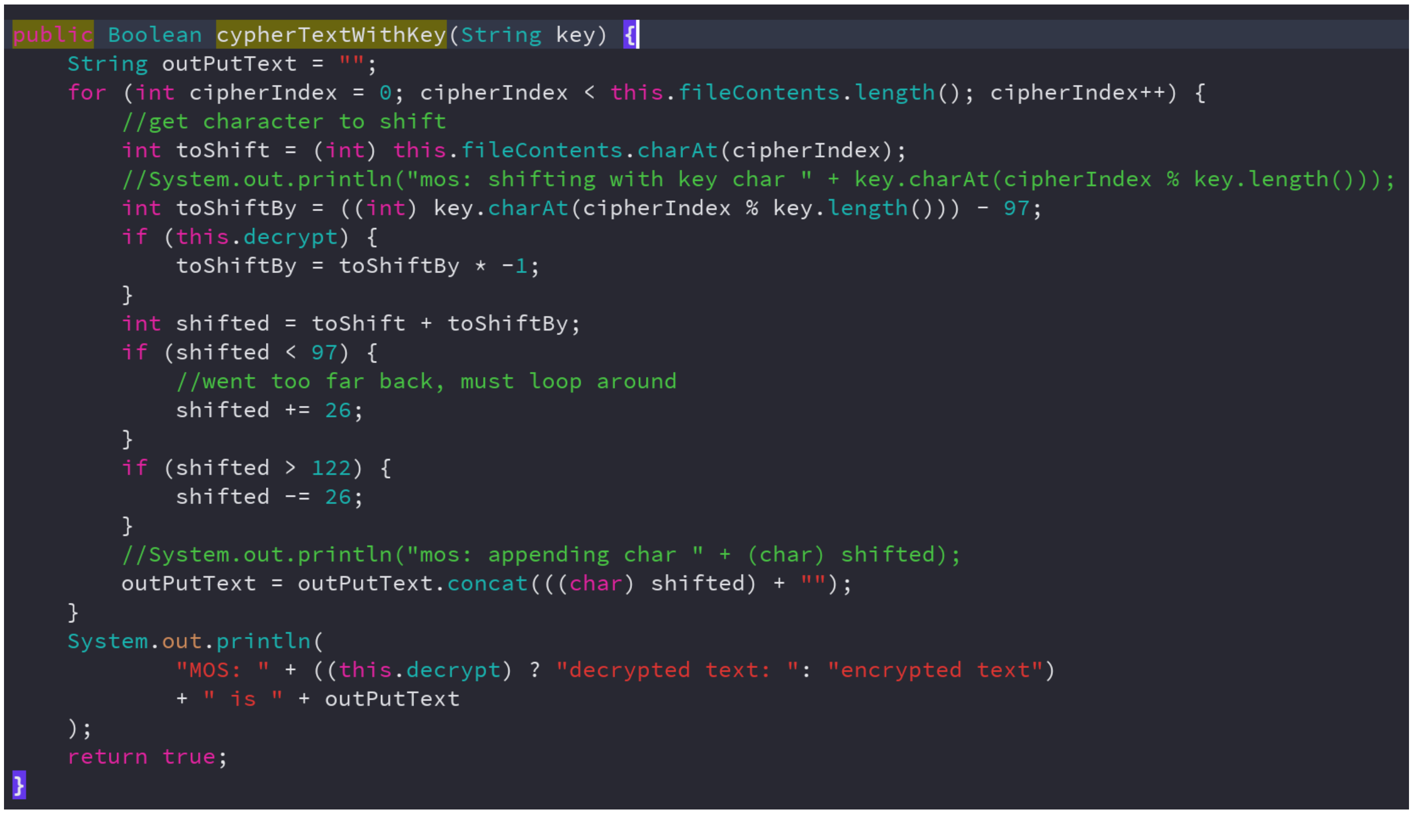
Mflaifel 20593396

Explanation of steps followed to encrypt plaintext with key

The java command line program can be used to both decrypt a plaintext message that has been encrypted with the vigenere cypher or to encrypt a plaintext message with a given key (passed as command line argument) using the vigenere cypher technique.

The main checks if we're given an encryption key, if we are then we encrypting some plaintext with the given key. To do this, the main function calls cypherTextWithKey in the Cypher class and passes it the key to encrypt with. The cipher text is then output to the command line. The plaintext is hardcoded to be located in plainTextToDecrypt.txt.

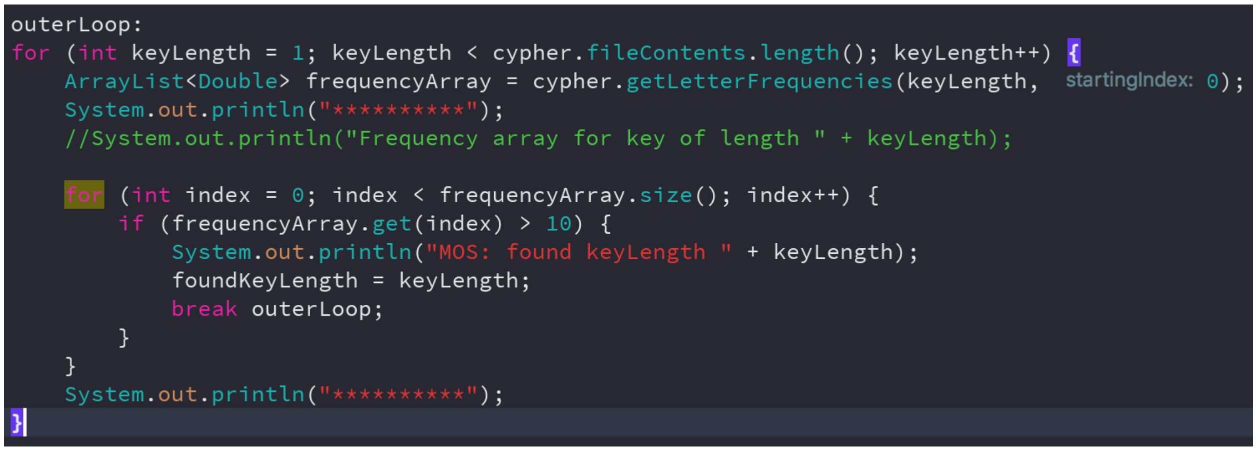
cypherTextWithKey is used for both encryption and decryption. It loops through the characters in the input file (to encrypt or decrypt). For each character it is shifted (forward or backwards depending on whether we are encrypting or decrypting) by the corresponding key character at the same position (repeating the key infinitely so we don'
Then after the shift if the character goes past 122 (z in asci), then we subtract 26 to keep it within the range of a-z. If after the shift the character goes below 97 (a in asci) then we add 26 to keep it within the a-z range.



```java
public Boolean cypherTextWithKey(String key) {
    String outPutText = "";
    for (int cipherIndex = 0; cipherIndex < this.fileContents.length(); cipherIndex++) {
        //get character to shift
        int toShift = (int) this.fileContents.charAt(cipherIndex);
        //System.out.println("mos: shifting with key char " + key.charAt(cipherIndex % key.length()));
        int toShiftBy = ((int) key.charAt(cipherIndex % key.length())) - 97;
        if (this.decrypt) {
            toShiftBy = toShiftBy * -1;
        }
        int shifted = toShift + toShiftBy;
        if (shifted < 97) {
            //went too far back, must loop around
            shifted += 26;
        }
        if (shifted > 122) {
            shifted -= 26;
        }
        //System.out.println("mos: appending char " + (char) shifted);
        outPutText = outPutText.concat(((char) shifted) + "");
    }
    System.out.println(
            "MOS: " + ((this.decrypt) ? "decrypted text: ": "encrypted text")
            + " is " + outPutText
    );
    return true;
}
```

Figure 1 cypherTextWithKey function

Explanation of steps followed to find key length and decrypt cipher text



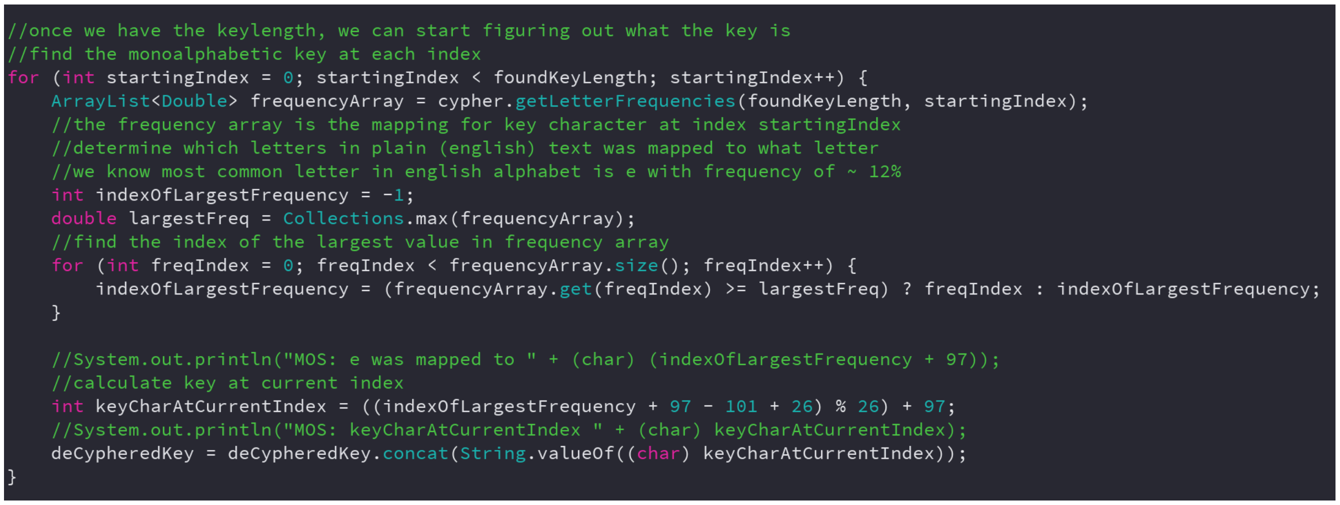
```java
outerLoop:
for (int keyLength = 1; keyLength < cypher.fileContents.length(); keyLength++) {
    ArrayList<Double> frequencyArray = cypher.getLetterFrequencies(keyLength,  startingIndex: 0);
    System.out.println("**********");
    //System.out.println("Frequency array for key of length " + keyLength);

    for (int index = 0; index < frequencyArray.size(); index++) {
        if (frequencyArray.get(index) > 10) {
            System.out.println("MOS: found keyLength " + keyLength);
            foundKeyLength = keyLength;
            break outerLoop;
        }
    }
    System.out.println("**********");
}
```

Figure 2 finding key length

To find the key length, I do a for loop for different key lengths, starting at key length 1 (smallest possible key length) and then looking at the statistics for letters if that was the assumed key

length. getLetterFrequencies loops through the file contents, jumping by keyLength and starting at startingIndex.
Figure 3 shows the statistics for keyLength of 1 starting at index 0.
Then we check for a character than has a frequency of more than 10%, which from the frequency distribution of the English letter alphabet means that that letter was mapped by the key character in this position from plain text character e to whatever that character is.

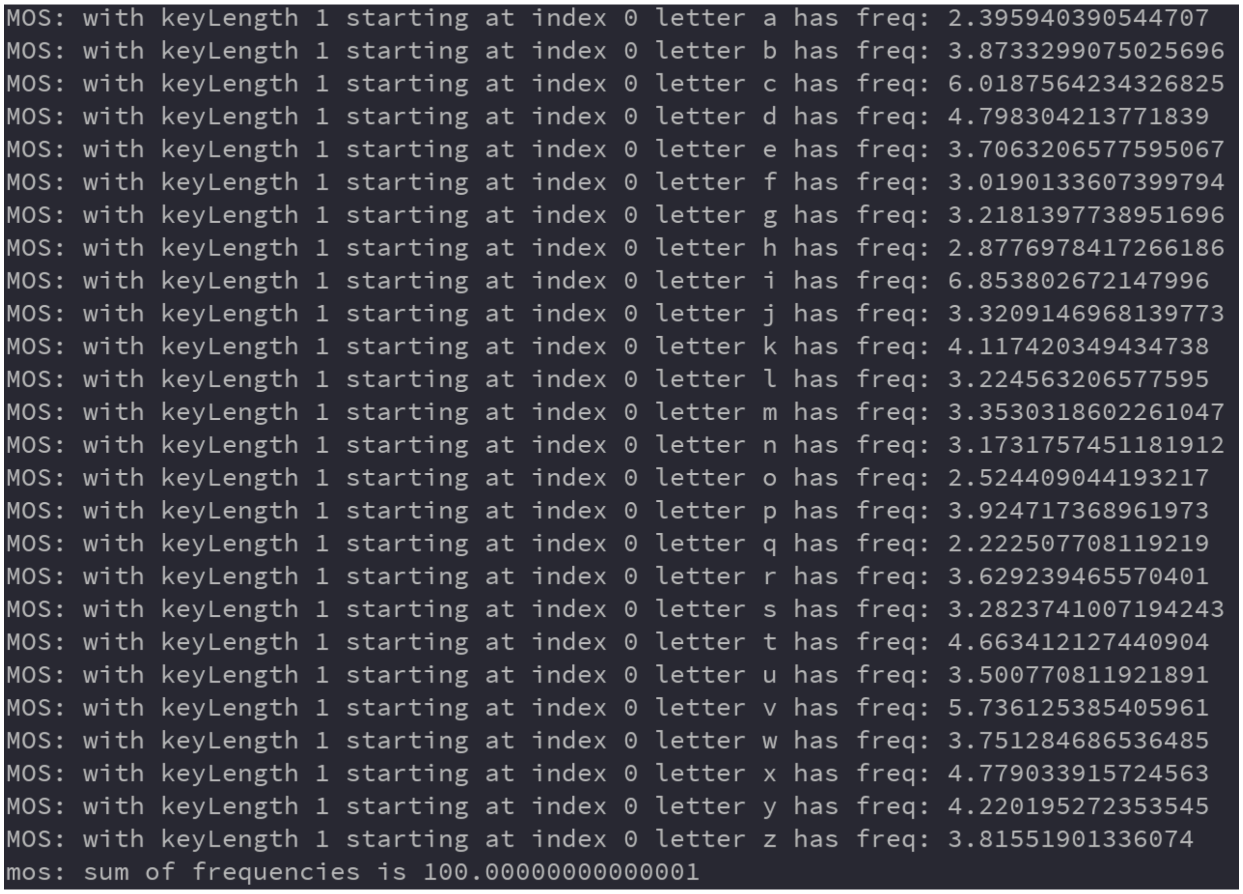
This assumes that when we get a character with a frequency of more than 10%, then we have found the keyLength. A more accurate way to find the keyLength would be to get the frequency distributions for all key lengths and use some statistical distance measure – such as the Bhattacharyya distance – to match the frequency distribution with the frequency distribution of the English alphabet, and the key length that produces a frequency distribution that is closest to the English alphabet is the keyLength. But for the purposes of this assignment checking if a character has more than 10% frequency distribution then it is sufficient to assume that it was mapped from plaintext e to whatever that character is.



```java
//once we have the keylength, we can start figuring out what the key is
//find the monoalphabetic key at each index
for (int startingIndex = 0; startingIndex < foundKeyLength; startingIndex++) {
    ArrayList<Double> frequencyArray = cypher.getLetterFrequencies(foundKeyLength, startingIndex);
    //the frequency array is the mapping for key character at index startingIndex
    //determine which letters in plain (english) text was mapped to what letter
    //we know most common letter in english alphabet is e with frequency of ~ 12%
    int indexOfLargestFrequency = -1;
    double largestFreq = Collections.max(frequencyArray);
    //find the index of the largest value in frequency array
    for (int freqIndex = 0; freqIndex < frequencyArray.size(); freqIndex++) {
        indexOfLargestFrequency = (frequencyArray.get(freqIndex) >= largestFreq) ? freqIndex : indexOfLargestFrequency;
    }

    //System.out.println("MOS: e was mapped to " + (char) (indexOfLargestFrequency + 97));
    //calculate key at current index
    int keyCharAtCurrentIndex = ((indexOfLargestFrequency + 97 - 101 + 26) % 26) + 97;
    //System.out.println("MOS: keyCharAtCurrentIndex " + (char) keyCharAtCurrentIndex);
    deCypheredKey = deCypheredKey.concat(String.valueOf((char) keyCharAtCurrentIndex));
}
```

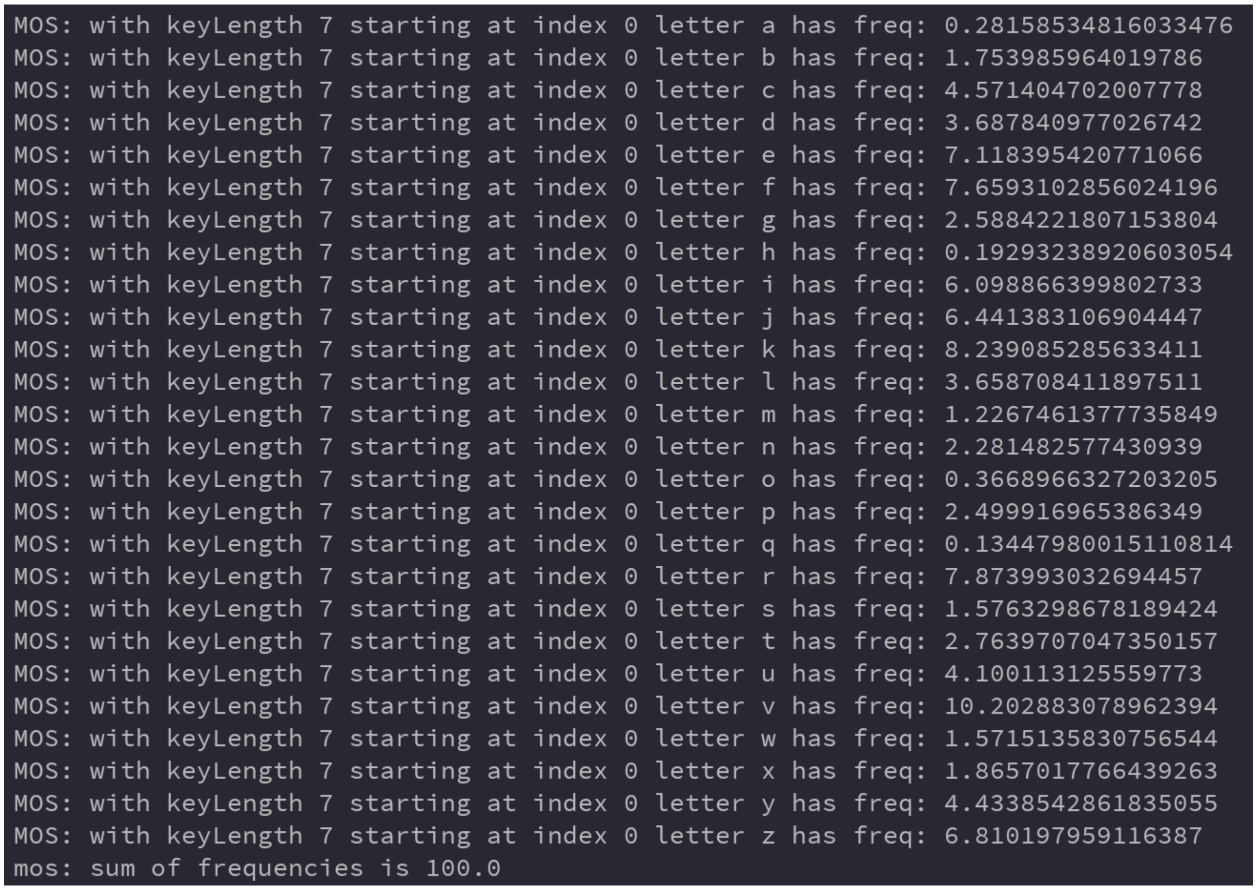
Figure 3 finding each character in the key

Figure 3 shows how we find individual letters in the key. We assume that the character with the highest frequency is the character that was e in the plaintext and so we calculate what key character would have mapped it to whatever character it currently is. We do this for every character in the key until we have found all the key characters.



```
MOS: with keyLength 1 starting at index 0 letter a has freq: 2.395940390544707
MOS: with keyLength 1 starting at index 0 letter b has freq: 3.8733299075025696
MOS: with keyLength 1 starting at index 0 letter c has freq: 6.0187564234326825
MOS: with keyLength 1 starting at index 0 letter d has freq: 4.798304213771839
MOS: with keyLength 1 starting at index 0 letter e has freq: 3.7063206577595067
MOS: with keyLength 1 starting at index 0 letter f has freq: 3.0190133607399794
MOS: with keyLength 1 starting at index 0 letter g has freq: 3.2181397738951696
MOS: with keyLength 1 starting at index 0 letter h has freq: 2.8776978417266186
MOS: with keyLength 1 starting at index 0 letter i has freq: 6.853802672147996
MOS: with keyLength 1 starting at index 0 letter j has freq: 3.3209146968139773
MOS: with keyLength 1 starting at index 0 letter k has freq: 4.117420349434738
MOS: with keyLength 1 starting at index 0 letter l has freq: 3.224563206577595
MOS: with keyLength 1 starting at index 0 letter m has freq: 3.3530318602261047
MOS: with keyLength 1 starting at index 0 letter n has freq: 3.1731757451181912
MOS: with keyLength 1 starting at index 0 letter o has freq: 2.524409044193217
MOS: with keyLength 1 starting at index 0 letter p has freq: 3.924717368961973
MOS: with keyLength 1 starting at index 0 letter q has freq: 2.222507708119219
MOS: with keyLength 1 starting at index 0 letter r has freq: 3.629239465570401
MOS: with keyLength 1 starting at index 0 letter s has freq: 3.2823741007194243
MOS: with keyLength 1 starting at index 0 letter t has freq: 4.663412127440904
MOS: with keyLength 1 starting at index 0 letter u has freq: 3.500770811921891
MOS: with keyLength 1 starting at index 0 letter v has freq: 5.736125385405961
MOS: with keyLength 1 starting at index 0 letter w has freq: 3.751284686536485
MOS: with keyLength 1 starting at index 0 letter x has freq: 4.779033915724563
MOS: with keyLength 1 starting at index 0 letter y has freq: 4.220195272353545
MOS: with keyLength 1 starting at index 0 letter z has freq: 3.81551901336074
mos: sum of frequencies is 100.00000000000001
```

Figure 4 letter statistics for key length 1



```
MOS: with keyLength 7 starting at index 0 letter a has freq: 0.28158534816033476
MOS: with keyLength 7 starting at index 0 letter b has freq: 1.753985964019786
MOS: with keyLength 7 starting at index 0 letter c has freq: 4.571404702007778
MOS: with keyLength 7 starting at index 0 letter d has freq: 3.687840977026742
MOS: with keyLength 7 starting at index 0 letter e has freq: 7.118395420771066
MOS: with keyLength 7 starting at index 0 letter f has freq: 7.6593102856024196
MOS: with keyLength 7 starting at index 0 letter g has freq: 2.5884221807153804
MOS: with keyLength 7 starting at index 0 letter h has freq: 0.19293238920603054
MOS: with keyLength 7 starting at index 0 letter i has freq: 6.098866399802733
MOS: with keyLength 7 starting at index 0 letter j has freq: 6.441383106904447
MOS: with keyLength 7 starting at index 0 letter k has freq: 8.239085285633411
MOS: with keyLength 7 starting at index 0 letter l has freq: 3.658708411897511
MOS: with keyLength 7 starting at index 0 letter m has freq: 1.2267461377735849
MOS: with keyLength 7 starting at index 0 letter n has freq: 2.281482577430939
MOS: with keyLength 7 starting at index 0 letter o has freq: 0.3668966327203205
MOS: with keyLength 7 starting at index 0 letter p has freq: 2.499916965386349
MOS: with keyLength 7 starting at index 0 letter q has freq: 0.13447980015110814
MOS: with keyLength 7 starting at index 0 letter r has freq: 7.873993032694457
MOS: with keyLength 7 starting at index 0 letter s has freq: 1.5763298678189424
MOS: with keyLength 7 starting at index 0 letter t has freq: 2.7639707047350157
MOS: with keyLength 7 starting at index 0 letter u has freq: 4.100113125559773
MOS: with keyLength 7 starting at index 0 letter v has freq: 10.202883078962394
MOS: with keyLength 7 starting at index 0 letter w has freq: 1.5715135830756544
MOS: with keyLength 7 starting at index 0 letter x has freq: 1.8657017766439263
MOS: with keyLength 7 starting at index 0 letter y has freq: 4.4338542861835055
MOS: with keyLength 7 starting at index 0 letter z has freq: 6.810197959116387
mos: sum of frequencies is 100.0
```

Figure 5 letter statistics for key length 7