CSC3621 Cryptography - Exercise 2

Aim - To understand the working of a polyalphabetic cipher and its weakness. In particular, the Vigenère cipher will be studied.

Introduction:

A Vigenere cipher is distribution of characters to a key in a given text. I have created a programme that takes a key and encrypts/decrypts a given text to produce the plain text/ cipher text.

Each letter in the alphabet has a numeric value related to it. The value is used to complete a calculation within the encryption algorithm to encrypt/decrypt the text.

The encryption takes the value of each key letter and shifts the plain text characters to the values i.e.:

‘c’ = index 2
‘e’ = index 4 if the key was ‘e’ then ‘c’ would become ‘g’ as it shifts 4 places.

If the message length is greater than that of the key then you simply use the key again until the message is filled. If the key fits over the message length then simply cut off the unneeded letters. i.e.

key = ncl
m = attackatdawn
nclnclnclncln

To decrypt Vigenere cipher you need the length of the key. This is what prevents it from being secure.

Implementation:

To implement such an algorithm, you must first understand what the system can and can’t read. To achieve the task I was required to take a variety of steps:

Read in

I first had to read in the text from the Plain Text and place in into a String. I could then complete the encryption method using a bespoke key.

Encrypt

To encrypt the plain text I made the function so that it would take in a String of the plain text and also a key to encrypt it with.

The function then places the the text into a character array and iterates through them performing the calculation to shift them to the positions of the key values.

```
String encrypt(String text, final String key) {
    /*
    *
    *Take plain text and key, then use vigenere cipher to encrypt
    *Take each letter and complete calculation to change character
    *
    */
    char letters[] = text.toLowerCase().toCharArray();

    char c;
    for (int i = 0, j = 0; i < letters.length; i++) {
        c = letters[i];
        if (c < 'a' || c > 'z') continue;

        letters[i] = (char) ((c + key.charAt(j) - (2 * 'a')) % 26 + 'a');
        j = ++j % key.length();
    }
    return new String(letters);
}
```

It then returns the array as a String(encrypted).

Decrypt

The decrypt function does the same as the encryption function but in reverse.

```
String decrypt(String text, final String key) {
    /*
    *
    *Take cipher text and key, then use vigenere cipher to decrypt
    *Take each letter and complete calculation to change character
    *
    */
    char letters[] = text.toLowerCase().toCharArray();
    char c;


    for (int i = 0, j = 0; i < text.length(); i++) {
        c = letters[i];
        if (c < 'a' || c > 'z') continue;

        letters[i] = (char) ((c - key.charAt(j) + 26) % 26 + 'a');
        j = ++j % key.length();
    }
    return new String(letters);
}
```

Finding the Key and Index of Coincidence

Both encryption and decryption work assume we know the key and its length. However if we want to decrypt a Vigenere cipher without the Key then we will have to find the key.

To do this we need to find the length of the key then we can work out what the key may be.

Understanding the length of the key is brute force and requires trial and error and learning at each test. This requires the splitting of the cipher text into columns for the computer to read. We can then test different lengths to find out the Index of Coincidence in each column. This will then return the most common frequency divided by the total number of

$$\text{Index of coincidence: } \sum_{i=0}^{25} {p_i}^2 = 0.065$$

characters in the text. We then take an average IOC of the overall and if it is close to 0.06 then we assume that it is the correct key length.

I did this by splitting the text into an array and adding each array to a list. I then found the IOC of each column and then calculated an average IOC.

```
int findKeyLength(String cipherText) {
    /*
    *
    *Finds the length of the key by getting the average IOC
    * and comparing it to 0.06
    *
    */
    int keyLength = 0;
    List<String[]> splitTexts = new ArrayList<>();

    for (int i = 0; i < 20; i++) {
        splitTexts.add(i, textSplit(cipherText, (i + 1)));
    }

    double avgIOC[] = new double[20];
```

```
IOC of cipher text: 0.0451
```

The number shown above is the general IOC of the cipher text for vigenere cipher.

```
1: 0.0451
2: 0.0448 0.0455
3: 0.0450 0.0451 0.0454
4: 0.0449 0.0451 0.0446 0.0458
5: 0.0665 0.0676 0.0663 0.0640 0.0667
```

This shows the required length of the key by finding the average IOC of each length.

Find the Key

Once the key length is established we can find the key by completing a frequency analysis on each column int the program.

I wrote a function that could find the most frequent character in each column compared to the plain text in “pg1661.txt” and then returns is as the best possible key.

```
Cipher Key is: plato
```

I then used the key returned to attempt to decrypt the cipher text. This succeeded and produced a decrypted file with an extract shown above.

Testing

As part of the testing process I completed the encryption of “newcastleuniversity” using the key “ncl”.

```
String cT = "newcastleuniversity";
```

```
//Encrypts newcastleuniversity with key ncl
vC.writeToFile(vC.encrypt(cT,key2),"nclencryption.txt");
```

Output: `aghpcdgnphptigcfkel`

I then compared the results to classmates who also produced the same result. I have written it in lowercase. I do understand that in cryptography practice it is better to encrypt to uppercase however for the purposes of this program I thought this would be suitable.

If I then try to find the key for the encrypted get from this, the result is not accurate. This is because the cipher text is too short and my program is not able to find the key for it.

```
newcastleuniversity Key is: ]cac_
```

You can see from this output that the key is not correct it should in fact be “ncl”. However this does not give it perfect secrecy as the shorter text is easier to brute force.

Conclusion

My analysis of Vigenere cipher is that the algorithm is not very secure and has many weaknesses. We can see from the tests that the algorithm can be broken using brute force in most circumstances. It therefore does not provide perfect secrecy and suffers in practicality. It is more secure than that of the shift/caesar cipher however equally has its weaknesses. We have seen how finding the index of coincidence gives us the ability to find the key length and therefore we can use frequency analysis on each column to find the key. This is brute force and depending on the length of the message can be a long process but is “crackable”.

References

Feng Hao. CSC3621_classical(2).pptx