# Coursework for CSC3621 Cryptography

## Part 1

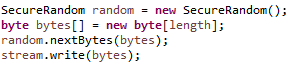
### Exercise 3

The main method of this exercise can be found within the OneTimePad.java file. Within this method I call three other classes. The first is a KeyGeneration.java class which is responsible for generating a key via a random set of bytes or by passing it in a String hex value. The second class that is called is the OneTimePadEncryption.java class which is responsible for encrypting and decrypting the given file and then writing the encrypted file to a new file. The final class that is called by the main method is the given OTPAttack.java. This is just used to show the output of this class and no code has been added.

#### Encrypting using One Time Pad

In One Time Pad it is important that the randomly generated key is only used once. If more than one message is encrypted with the same key then the resultant XOR between the encrypted files is equal to the XOR of the two plaintext files. This leads to vulnerabilities which will be discussed later. Therefore at the start of each run of the program I am generating a brand new key to be used for every encryption.

It is important that the key is as random as possible so I used SecureRandom to generate a large amount of bytes which were then written to a file to be used as a key.

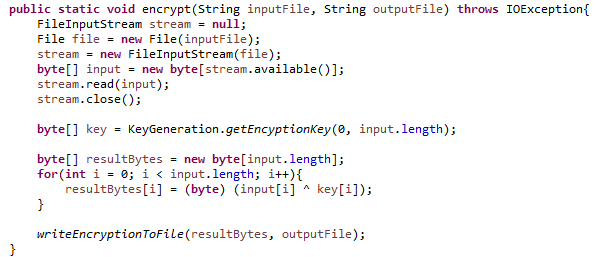


A snapshot of part of the key looks like the following:



Once the random string has been generated the encryption can then take place. The method encrypt found is OneTimePadEncryption.java is then called with a given inputFile and a outFile which to write to. This method reads in the file given as well as reading in the random generated key. Both of which are in the form of a byte[] array. The resultant encryption is then calculated by XOR’ing the two arrays together one byte at a time. The point of using XOR instead of AND or OR is that XOR has an equal chance of producing a 0 or a 1 unlike that of AND or OR.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AND | | | OR | | | XOR | | |
| 0 | **0** | 0 | **0** | **0** | 0 | **0** | **0** | 0 |
| 0 | **1** | 0 | **0** | **1** | 1 | **0** | **1** | 1 |
| 1 | **0** | 0 | **1** | **0** | 1 | **1** | **0** | 1 |
| 1 | **1** | 1 | **1** | **1** | 1 | **1** | **1** | 0 |



The resultant file of the encryption can be found in exercise3/onetimepadEncryption.txt. A sample output of that file looks like the following:



After the encryption process the main method is also able to reverse the encryption or decrypt the code using exactly the same method and key with no changes except that the file that is passed in is that of the onetimepadEncryption.txt. The resultant decryption of the file can be found in exercise3/onetimepadDecryption.txt and will be the same as the originally encrypted file.

#### Trading with a Friend

I was able to send a given key and an encoded lorem ipsum example to a friend and they were able to produce a correct decryption of the code. I was also sent an encrypted Text and an OTP pad key found within the friend folder. One the correct files were used as a parameter to the decryption method and the method for pulling out the generated secret was pointing at friend/otp instead of my generated key I was able to fully decrypt their file as seen in friend/Decryption.txt

#### Cryptanalyze

From the lectures I remembered that when encrypting two different method with the same key the XOR of the two encryptions is equal to that as the XOR of the two plaintext messages such that for is m1 XOR key = c1 and m2 XOR key = c2 then m1 XOR m2 = c1 XOR c2.

The next step that I took was to copy and paste the given class into eclipse and run the code. By looking at the code and at the output the class loops round each ciphered message and XOR’s it with every other one. When the result is a character in the range [a-zA-Z] then it prints it to the screen else it will print a dash. I knew that the full output of XORing the two messages is the same as XORing the two original plain text messages and therefore there must be something important about these plaintext characters.

The first tip of the exercise is to see what happens when you XOR a character in the range [a-zA-Z] with a space character. I could have written a small program to test the output but from a quick search I found out that the result of the tip is that the case of the character is changed from down case to up case or vice versa.

Say that we have two messages. Where an underscore(\_) is equivalent to a space and the asterisk is equivalent to a random other character.

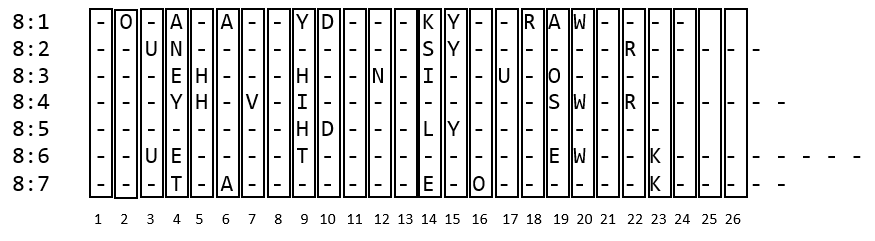
M1 = abcdefg

M2 = \_A\_B\_C\_

M1 XOR M2 ≈ A\*C\*E\*G

This logic means that for every capital alphabet character shown in the classes output is equal to the shown character or a space character.

The exercise is only interested in deciphering the 8th message and therefore I looked at the following output from the given class:



By looking at a column at a time we can decipher the message into plain text.

Column 2: Assumed that the letter for column 2 is an ‘O’. If we look at other messages that are XOR’ed with message 1 there is always a different capital letter suggesting that the second letter in message 1 is a space.

Column 3: Letter ‘U’. It appears twice within the column and no other letters do. So where there are spaces within the other text, message 8’s ‘u’ is up cased.

Column 4: Space. As each character is a different capital letter we can assume that the character within message 8 is a space which is up casing all the other messages. When XOR’ing 8:5 we get a -. This is due to both characters in column 4 being a space

Column 5,6,7: H,A,V Respectively. They are the only letters appearing in their respective columns.

Column 8: Cannot deduce any letter from the columns although judging that the previous three letters were HAV and the next letter is a space we can assume that the letter is an E creating the word HAVE.

By following the same rules for each column we eventually get the message.

\*OU\_HAV\*\_D\*N\*\_YOUR\_W\*RK

Where \* is a random character and \_ is a space and the column 8 is still represented by a random character.

The final decrypted message can be easily worked out from the output above. The message reads:

YOU HAVE DONE YOUR WORK