Project 1 - Vigenère Cipher

Marcielly Nascimento (7995)

Section 01

CIS 4367 Computer Security

Fall 2021

**How to run code**

To run the code, I opened the project “Project\_1” in the CodeBlocks C++ IDE and used the “build and run” feature on the “FinalVersion” build to run it, which runs using “main.cpp”. This version of the code is the final project version; it asked the user to input their choices on things like how many characters the language has, if they want to encrypt or decrypt, what frequencies they want to use for each character in the language, etc.,

To run the test build version that **tests the encryption**, I changed the build option to the “TestEncryption” build and then run it. This build is run from “mainEncrypt.cpp”.

To run the test build version that **tests the decryption**, I changed the build option to the “TestDecryption” build and then run it. This build is run from “mainDecrypt.cpp”.

To run the test build version that **tests the frequency analysis**, I changed the build option to the “TestProject1” build and then run it. This build is run from “mainTest.cpp”.

**How the code works**

In the version of the code that takes user input (“FinalVersion” build), the code starts by asking the user if they want to encrypt or decrypt. If the user chooses to encrypt, the code will then ask what file the user wants to input the plaintext from. After the user enters a valid file that is in the same location as the code, the code lets the user know the file opened successfully and then prints a preview of the code to the console in order to assure the user the code has opened the right file. The code then asks the user how many characters are in the language they want to encrypt in. After the user inputs the number of letters, the code will then ask if the language they are using uses the characters A-Z. If the user says it does, the code will then fill in the character array with the characters A-Z (this is only possible if the user input a number equal to or greater than 26 for the number of letters in the language). If the user input a number higher than 26 for the number of characters in the language, the code will then ask the user to input any remaining characters in the language. In this case, the user would then input the ASCII number values for “+”, “=”, and “\*”. Once the code has the plaintext and characters of the language, it formats the plaintext by capitalizing and removing everything that is not a character in the language, like spaces.

The program will then ask the user to input a key using only capitalized characters of the language. This key will be used to encrypt the plaintext using the equation (plaintext character + key = ciphertext character). This equation goes through each character in the plaintext with the key value cycling through each character in the key. This will output the ciphertext, which is sent to the file “output.txt”.

If instead the user chooses to decrypt, the code will ask what file the user wants to input the ciphertext from. After the user enters a valid file that is in the same location as the code, the code lets the user know the file opened successfully and then prints a preview of the code to the console in order to assure the user the code has opened the right file. The code then asks the user how many characters are in the language they want to decrypt. After the user inputs the number of letters, the code will then ask if the language they are using uses the characters A-Z. If the user says it does, the code will then fill in the character array with the characters A-Z (this is only possible if the user input a number equal to or greater than 26 for the number of letters in the language). If the user input a number higher than 26 for the number of characters in the language, the code will then ask the user to input any remaining characters in the language. In this case, the user would then input the ASCII number values for “+”, “=”, and “\*”. Once the code has the ciphertext and characters of the language, it formats the ciphertext by capitalizing and removing everything that is not a character in the language, like spaces. The code will then ask the user if they want to use language frequencies for English, the 29-character language used in this project, or manually input frequencies for each character in the language.

The code will then find the key length. To do so, the code will ask the user to input a number to represent the number of times (out of 100) that a coincidence must occur in order for that coincidence to become significant. This is important because to find the key length, the code shifts the ciphertext to the right once. The shifted string will be compared to the ciphertext string. If the shifted string and the ciphertext string have a character that matches at the same index then the variable “m” gets incremented each time this occurs until the string ends. After a the string reaches an end, the coincidence variable will be incremented if “(m/onlyAlpha.length())\*100 >= percent”. The percent variable is the number the user entered to represent the number of times a coincidence must occur in order for that coincidence to become significant. This means that if the number of matches/rows(100) times 10 is greater than or equal to the user inputted number, then the number of matches in that row of shifts can be considered significant, so the “coincidence” variable is incremented. The reason the number of times a coincidence must occur to become significant is a user input variable is because the number varies based on each ciphertext and is more of a process of trial and error than a static number that is known. So, it is important to allow the user to set this number and change it if it does not output the correct key length (though it should only be changed after the key contents are found since a key with repeating letters can often be shortened to a shorter key and function the same). This process is then repeated 100 times with each row shifting right by one. After it has run 100 times, the code will calculate the key length by taking the rows(100) and dividing it by the coincidences variable, then rounding to the nearest integer.

Now that the code knows the key length, it will use it to find the contents of the key. For each character in a key, the program will find the frequency of each corresponding letter in the ciphertext. If key length is 3 then to find the first character in the key, the program will only look at the first character in the ciphertext and every character 3 positions away from the previous one. If key length is 2, it will look at the second character and every character 3 positions away from the previous one. And so on. For each character in the key, the program will count how many of each letter it finds and the total number of characters it looks at, by using the mentioned method. It takes this information and finds the frequencies of the ciphertext that corresponds to each key content by dividing the count of each character in the language’s alphabet by the total number of characters that correspond to the key content/ the number of characters that were looked at. After the program finds the ciphertext frequencies for the first key content character, it will take that and the frequencies of the language that the user inputted previously, it will multiply the frequency of the ciphertext with the language frequencies for each character in the language and add all of those multiplications for each character up into a sum variable. The program will then shift the frequencies of the ciphertext to the right and calculate the sum of (language frequencies \* ciphertext frequencies) for each character in the language again. Then shift again. This cycle will repeat for the number of characters in the language. After all of the sums have been calculated for each possible shift, the program will have found the largest sum. The number of shifts corresponding to the largest sum represents the correct combination of amount of shifts that were used to encrypt the plaintext. For example, in the English language, if the largest sum happened at shift 5, the character of the key would be “F”. This then repeats for the number of characters in the key / key length. After each key character is found, the entire key is known and can be used to decrypt the ciphertext.

The program will then take the key and implement the equation ((ciphertext – key) MOD number of characters in the language). This equation occurs for each character in the ciphertext and the key cycles through each character in the key. This will output the plaintext, which is sent to the file “output.txt”.

**In-depth Examples**

In the test build “TestEncryption”, the total letters in the language is pre-set to 29. The ASCII numbers for the characters “+”,” =”, “\*”, and “A” through “Z” are pre-set as the characters for the language. And the language frequencies for each character are also pre-set. The program then opens the file “We.txt”. The first couple characters in this file are “WE THE PEOPLE OF THE UNITED STATES”. The program takes the entire text from the file and formats it by removing any characters that are not from the language, in this case the spaces are removed and everything else is kept. The program also converts any alphabet characters to upper case, though the plaintext in this case is already capitalized.

The program then moves onto the actual encryption. It asks the user what key should be used to encrypt it. In this case, we will use the key “USA”. The program, takes this key to encrypt the plaintext using the equation (plaintext character + key = ciphertext character). This equation goes through each character in the plaintext with the key value cycling through each character in the key. For example, the program will take the plaintext character “W” and the key “U” then add them and output the ciphertext character “Q”. For the next plaintext character “E” the code will add it and the next key character “S”, to output “Z”. After going through the entire plaintext all while cycling though the character in the key, the program writes the output ciphertext to the file “output.txt”.

In the test build “TestDecryption”, the total letters in the language is pre-set to 29. The ASCII numbers for the characters “+”,” =”, “\*”, and “A” through “Z” are pre-set as the characters for the language. And the language frequencies for each character are also pre-set. The program then opens the file “salad.txt”. The first couple characters in this file are “AUVPHNGLYFCHI=+B”. The program takes the entire text from the file and formats it by removing any characters that are not from the language. The program also converts any alphabet characters to upper case.

The program then asks the user about the number until coincidences are significant, which is set as the “percent” variable. In this case, we will use the number 5, which means that when shifting the rows to see how many characters match the original string, if 5% or more of a shift matches with the original string, then those matches become significant enough to become more than just a coincidence. The number of times a shift row coincidence becomes significant is tallied, and in this case there are 19 significant coincidences out of 100. 100/19 is 5.26, which rounded to the nearest integer is 5. Which means the key length is 5. Then to find out what each character of the key contains, the method of adding up the sum of all (text frequency \* language frequency) for each possible combination and then using the shift number that corresponds to the largest sum to find the key content, which is described in more detail in the “How the code works” section. The program then finds the key to be “\*SUN\*” and uses this key for the decryption. The equation ((ciphertext – key) MOD number of characters in the language) is used. For example, the ciphertext character “A” minus the key character “\*” outputs the plaintext “=”. And just like the encryption this equation occurs for each character in the ciphertext and the key cycles through each character in the key. This will output the plaintext, which is sent to the file “output.txt”.

**References**

[Vigenère Cipher - Decryption (Unknown Key)](https://youtu.be/LaWp_Kq0cKs) - This video was used as a reference on how to decrypt the Vigenère Cipher without knowing the key.