# CIS7 Project Documentation Guide – THE VIGENERE CIPHER.

A close up of a sign

Description automatically generated

K!smet Industries .

The author of the project is K!smet Industries. K!smet is the pseudo name for the app designer Kristen San Martin. This name was chosen decades ago. The word Kismet means “fate or destiny”, and it also happens to only contain letters that are in either her first or last name.

The problem to be solved by this program is to offer an easy way to encode and decode encrypted using a key. Therefore, unless the receiving party has the key code, it is mathematically impossible to intercept and decrypt. Just as any technology helps complete tasks that humans can do faster and more efficiently, so does this program. What would probably take a person two minutes a word to decrypt and encrypt (not to mention keeping track of where you are in the dizzying lines of the table) this program will do entire lines of message accurately and within milliseconds.

The objective of the program is to be easy to use, accurate and fun. It was designed not only to solve the problem of encoding and decoding messages using a key but to also seem a bit like a game. It starts off as if war is about to begin, then it asks, “What are Your orders”. It allows the user to choose either encryption or decryption, it allows the user to view the Vigenere Table, or to Stand Down (i.e. Quit). When the user quits it displays a little like about “Being at Peace…for now.” This was simply to add an element of fun to the cipher program. Color was added for a splash of fun as well as to help the user to know where they are at within the program simply by color. Encryption is purple and Decryption is blue. To aid in the ease of use, when asking to restart, although it is not obvious, the user can either directly enter the menu key without pressing Y first, or they can press Y and be prompted for their menu choice. The menu is also available to be viewed again, just in case the user forgot what their options were.

While I tried to make the program as complete as possible there are still some known limitations of the program. The first and most major of the limitations is the inability for the message to contain symbols or punctuation. The second limitation, although the ability to use case is not terribly important in this case, everything is converted to uppercase in order to limit the data validation needed. The third limitation is, as far as I am concerned, the UI. I did my best to spruce up the user interface as much as possible using different colors and highlights, however by today’s standards it’s just a DOS command prompt and is archaic looking.

If someone else or I were to try to upgrade the program to patch its known limitations the first should obviously enable an ability to use symbols and punctuation. “Woman without her man is nothing.” and “Woman, without her, man is nothing.” represents a significant difference in the way a sentence is interpreted due to punctuation and the lack there off. Punctuation is an important part of communication in the English language. I would suggest simple pass through punctuation the same way the spaces are pass through spaces from the user text into the encoded text and vice-versa. As for the UI, since I am very new to C++ I am not sure what sort of UI tools are available to make this program look more like what windows users are used to. All that I would know how to do is to rebuild it in JAVA and perhaps use JAVAFX or Android Studio to give it a more modern look.

Discrete Structures were used in this program mostly in the form of sets, recursion, reflexive relationships, and traversing through strings. Several functional algorithms were implemented in this program but a few of the main ones that drive the programs primary function are listed below. Each represents the use of discrete structures and these features are discussed in more detail.

**NextLetterInAlphabet()** is simpler form of recursion. It works by increasing a static variable by one, every time the function is called. This variable is essentially the ascii key of the next letter in the alphabet, when this static variable labeled “total” gets to the Z ascii code it returns to 65 which is A.

**addNextLetter\_Key()**is similar but works differently. It takes in a string which is essentially becomes its ‘local alphabet’ of letters. Every time the function is called it returns the next letter in the key. When it gets to the end of the string and returns the last letter of the word, it loops back around and then returns the first letter of the word and the recursively goes letter by letter from there.

**CreateMatchingKey**() uses a higher form of recursion as it iterates through intervals of i++ as many times as it needs to in order to match the length of the message it is encoding or decoding. Every time it loops it calls on addNextLetter\_Key() to provide the letters and then appends them to a larger string that will ultimately match the length of the text input. This function builds a reflexive relationship between the matching key and the user input.

**createVigenereSquare()** essentially uses recursion and calls on nextLetterInAlphabet() to fill the set for a 2D array. This algorithm uses a recursive loop and iterates through the entire 2D array pulling letters from the alphabet and placing them in the location according to the ‘i’ and ‘j’ variables in the embedded for loops.

**generateEncodedCipher()** this function transverses through the text input and the matching key in parallel to find the location that holds the letter that represents the encoded word. The text input letter represents the location on the column and the letter in the key represents the location in the row. Where they intersect is the encoded letter. As it transverses these strings in parallel it is appending these letters into a string. This is an example of using symbolic logic to model real life situations. Where the last function createMatchingKey() built the reflexive relationship, this function uses it to accurately find the location of the correct letter within the Vigenere table.

**PATCH 1 VERSION 2**

**generateEncodedCipher2()** Encoding is now done without the use of the physical Vigenere Table. Although the program still allows it to be used, the updated version of this algorithm now uses the addition method in the same way the decryptCipher() function uses the subtraction method. The algorithm transverses through the plaintext and key. It finds the symmetrical location of each and gets the ascii value. It then adds these together and uses mod 26 to loop recursively through the alphabet. It then adds 65 to get the appropriate ascii code. This ascii character is then appended to a building string that becomes the cipher*. Even more than before this method uses symbolic logic to represent real life situations.*

**decryptCipher()** this function also transverses through the user input and its corresponding key. In this case however, it finds the ascii value of each letter in both and then subtracts these values. This subtracted value (more or less) is the ascii code for the decoded letter. It does this process recursively though the length of the user input until every letter is decoded.

**Vigenere Cipher - Pseudocode**

**Menu**

1 Prompt user to encrypt

2 Prompt user to decrypt

**1 Encrypt**

Prompt user for text

Take in text convert to upper

Prompt for key

Take key convert to uppercase

Match key length to text length by looping key text - making a long key

Search letter by letter and save the first letter of the text and the first letter of the long-key into int variables and subtracting 64 to find their location in the alphabet.

Use these numbers as index locations in the vigenere square array. The letter at this given location is saved into a string and appended letter by letter till there are no more letters.

Results are produced with key plaintext and new cipher.

Ask to restart and return to menu

N = quit

Y = return to menu

**2 Decryption**

Prompt for encoded text

Convert input to upper

Prompt for key

Convert to upper

Make long key out of user input key.

One by one convert each letter in both the long-key and the code to their location index in the alphabet.

Use these numbers and subtract them. If this number is pos add 65. If this subtraction is negative add 26.

This number is an index for a character on the first row of the Vigenere square.

And these chairs one by one to a string to get a decided word.

Display decoded word, key, and encrypted word.

**Ask to restart**

Y = Restart

Q = Quit

**Vigenere Cipher - Flow Chart**

