Implementation of Vigenère in Swift Code

This is a self created implementation of the Vigenère Cipher in Apple’s Swift language.

Code Synoptic Breakdown is below:

I first wrote the VigenereCipher class declaring the required attributes (Alphabet & key configs), Inside the initialiser of this class I simply hardcoded the Alphabet setup, and a cipher key is passed in upon instantiation. This initialiser sets the required attributes as normal.

On this class there are three functions, the first of which (getAlphabetPosition) I wrote is used to get a hold of the index of any given character passed in and cross checked against the alphabet.

Then there are two functions – encryptionFunc & decryptionFunc. The encryption function works with a swift style for loop, going char by char, getting the position of the character in the alphabet, then checking if this char should be encrypted or not – for example whitespace should be ignored etc.

Now the magic happens!

```swift
//select character from key to encrypt this character from the original message using modulo arithmetic
let charToEncryptWith: Character = cipherKey[cipherKey.index(firstIndex, offsetBy: index % keyLength)]

//get the alphabet position of the encryption character just selected
let encryptionCharIndex = getAlphabetPosition(character: charToEncryptWith)

//Get the Vigenere encryption chars index - i.e. Use the formula to get the characters index that will replace the
original character in the original msg [(originalCharIndex + encryptionCharIndex + 26) Modulo 26]
let encryptedLetterIndex = (originalCharIndex + encryptionCharIndex + ALPHABETSIZE) % ALPHABETSIZE

//append this encrypted letter to the returnMsg
returnMsg.append(ALPHABET[ALPHABET.index(firstIndex, offsetBy: encryptedLetterIndex)])
```

This happens just like the code comments express

1. Get the character in the key that we will use to encrypt the character in this loop iteration, using Swift style String notation to extract the required char using modulo.
2. Lookup the index location of this character found in step 1.
3. Use the Vigenere formula to grab the position of the character we’re actually interested in which is derived from adding the (original chars index and encryption chars index and 26) and performing a modulo 26 on the sum of this addition.
4. Now simply append the character associated with this index to the returnMsg string.

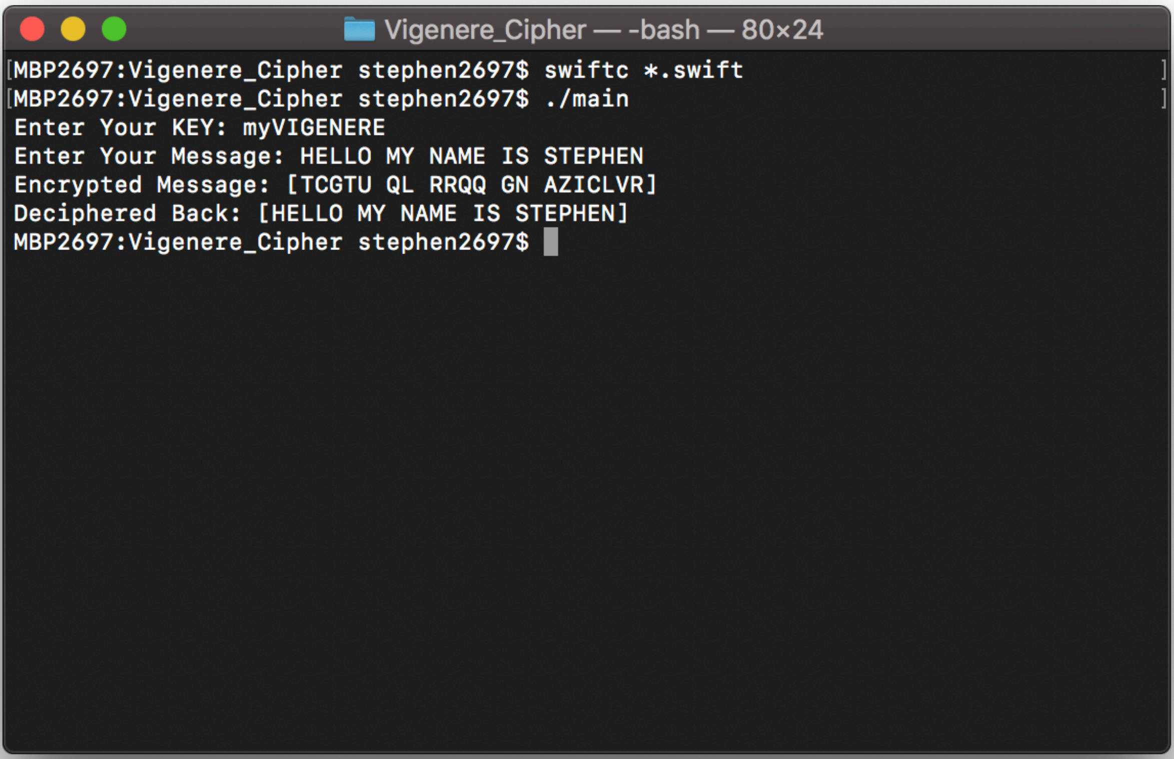
Decryption is handled in a likewise manner using a for loop and the main code lines are as follows:

```
//select character from key to decrypt character from the encrypted message using modulo arithmetic
let charToEncryptWith: Character = cipherKey[cipherKey.index(firstIndex, offsetBy: index % keyLength)]

//Get the decryption character index in alphabet
let decryptionCharIndex = getAlphabetPosition(character: charToEncryptWith)

//Get the decrypted letter's index in the alphabet
let decryptedLetterIndex = (originalCharIndex - decryptionCharIndex + ALPHABETSIZE) % ALPHABETSIZE

//append this decrypted letter to the returnMsg
returnMsg.append(ALPHABET[ALPHABET.index(firstIndex, offsetBy: decryptedLetterIndex)])
```

<u>To Run the program do as follows:</u>

Note I wrote a main swift file "main.swift" which acts as the entry point for the program - provides output to user and takes user input and instantiates the VigenereCipher class, passing the entered Key in the initialiser arguments – Nothing special.

To Run this program cd to the required directory and follow these steps:



```
MBP2697:Vigenere_Cipher stephen2697$ swiftc *.swift
MBP2697:Vigenere_Cipher stephen2697$ ./main
Enter Your KEY: myVIGENERE
Enter Your Message: HELLO MY NAME IS STEPHEN
Encrypted Message: [TCGTU QL RRQQ GN AZICLVR]
Deciphered Back: [HELLO MY NAME IS STEPHEN]
MBP2697:Vigenere_Cipher stephen2697$
```