Mini Project 1 - Vigenère Cipher & Brute Force Password Cracker

The Vigenère cipher is a method of encrypting alphabetic text, through the use of a simple form of polyalphabetic substitution. Furthermore, a polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. Specifically, the Vigenère cipher encrypts plaintext through use of the Vigenère square or Vigenère table. A Vigenère table consists of the alphabets written out 26 times in different rows, where each repetition of the alphabet is shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers. To encrypt, a table of alphabets can be used (i.e. tabula recta, Vigenère square, Vigenère table) that has several Caesar Ciphers in sequence with different shift values.

To begin Mini Project 1, a simple Vigenère cipher was implemented with the option to encrypt or decrypt a plaintext message with a given key. The cipher utilized the algorithm for *EK(m) = m + K mod 26* for encryption and the algorithm *DK(m) = m - K mod 26* for decryption. In respect to input for the cipher the plaintext (encrypt)/ciphertext (decrypt) only contains letters, has no spaces between words, and converts all characters to uppercase.

Code for Task 1 - Vigenère cipher:

```
import string

def Vigenère_cipher(text, key, mode):
   key_len = len(key)
   key_index = 0
   result = ""
   for char in text:
       char = char.upper()
       if char.isalpha():
           key_char = key[key_index % key_len].upper()
           key_index += 1
           if mode == '1':
               result += chr((ord(char) + ord(key_char) - 2 * ord('A')) % 26 +
ord('A'))
           else:
               result += chr((ord(char) - ord(key_char) + 26) % 26 + ord('A'))
   return result

print("-------------------------")
print("Audrey's Vigenère Cipher")
print("-------------------------")
```

```
print("\t1. Encrypt")
print("\t2. Decrypt")

mode = int(input("\n\tEnter your choice (1/2): "))
text = input("Enter the text to encrypt/decrypt: ")
key = input("Enter the key: ")

result = Vigenère_cipher(text, key, mode)

print("Result:", result)
```

Expanding on the Vigenère cipher (above), a Brute Force Password Cracker was implemented. The Brute Force Password Cracker takes in three parameters: (1) a string of ciphertext; (2) an integer keyLength that denotes the length of the key; and (3) an integer firstWordLength that denotes the length of the first word of the plaintext. The implemented password cracker tests all possible keys that have the length 'keyLength', consisting of characters from 'A' to 'Z' in sequence. For all of the possible keys, a plaintext is created and then compared to the given dictionary, in order to verify whether the first word, identified by firstWordLength, of the plaintext is in the given dictionary.

Code for Task 2 - Vigenère Cipher & Brute Force Password Cracker:

```
import string
import time

def Vigenère_cipher(text, key, mode):
    key_len = len(key)
    key_index = 0
    result = ""
    for char in text:
        char = char.upper()
        if char.isalpha():
            key_char = key[key_index % key_len].upper()
            key_index += 1
            if mode == 'encrypt':
                result += chr((ord(char) + ord(key_char) - 2 * ord('A')) % 26 +
ord('A'))
            else:
                result += chr((ord(char) - ord(key_char) + 26) % 26 + ord('A'))
```

```
    return result

def brute_force_cracker(ciphertext, key_length, first_word_length, dict_file):
    with open(dict_file, 'r') as f:
        words = set(word.strip().upper() for word in f)
    result = []
    for i in range(26**key_length):
        key = ""
        for j in range(key_length):
            key += chr(i // (26**j) % 26 + ord('A'))
        plaintext = Vigenère_cipher(ciphertext, key, 'decrypt')
        first_word = plaintext[:first_word_length].upper()
        if all(char in string.ascii_letters for char in first_word) and first_word in
words:
            result.append(plaintext)
    return result

print("------------------------------------------------------------")
print("Audrey's Vigenère Cipher & Brute Force Password Cracker")
print("------------------------------------------------------------")
print("\t1. Encrypt")
print("\t2. Decrypt")
print("\t3. Brute Force")

choice = int(input("\n\tEnter your choice (1/2/3): "))

if choice == 1:
    text = input("\nEnter the message: ")
    key = input("Enter the key: ")
    result = Vigenère_cipher(text, key, 'encrypt')
    print("\nEncrypted message:", result)
elif choice == 2:
    text = input("\nEnter the ciphertext: ")
    key = input("Enter the key: ")
    result = Vigenère_cipher(text, key, 'decrypt')
    print("\nDecrypted message:", result)
else:
    ciphertext = input("\nEnter the ciphertext: ")
    key_length = int(input("Enter the key length: "))
    first_word_length = int(input("Enter the length of the first word: "))
    dict_file = ("MP1_dict.txt")
```

```
    start_time = time.time()
    result = brute_force_cracker(ciphertext, key_length, first_word_length,
dict_file)
    end_time = time.time()

    print("\nTime taken to Brute Force Crack the Password:", end_time - start_time,
"seconds")
    for plaintext in result:
        print("Decrypted message:", plaintext)
```

The implemented Brute Force Password Cracker (above) calculated the time it took to successfully decrypt each of the following messages:

1. "MSOKKJCOSXOEEKDTOSLGFWCMCHSUSGX"
key length = 2; firstWordLength = 6
Time taken to Brute Force Crack the Password: 0.06850314140319824 seconds
Decrypted message: CAESARSWIFEMUSTBEABOVESUSPICION

2.
"PSPDYLOAFSGFREQKKPOERNIYVSDZSUOVGXSRRIPWERDIPCFSDIQZIASEJVCGXAYBGYXFPSREKF
MEXEBIYDGFKREOWGXEQSXSKXGYRRRVMEKFFIPIWJSKFDJMBGCC"
keyLength=3; firstWordLength = 7
Time taken to Brute Force Crack the Password: 0.5478341579437256 seconds
Decrypted message:
FORTUNEWHICHHASAGREATDEALOFPOWERINOTHERMATTERSBUTESPECIALLYINWARCANBRING
ABOUTGREATCHANGESINASITUATIONTHROUGHVERYSLIGHTFORCES

3. "MTZHZEOQKASVBDOWMWMKMNYIIHVWPEXJA"
keyLength=4; firstWordLength = 10
Time taken to Brute Force Crack the Password: 4.100910902023315 seconds
Decrypted message: EXPERIENCEISTHETEACHEROFALLTHINGS

4. "SQLIMXEEKSXMDOSBITOTYVECRDXSCRURZYPOHRG"
keyLength=5; firstWordLength = 11
Time taken to Brute Force Crack the Password: 121.23882699012756 seconds

Decrypted message: IMAGINATIONISMOREIMPORTANTTHANKNOWLEDGE

5. "LDWMEKPOPSWNOAVBIDHIPCEWAETYRVOAUPSINOVDIEDHCDSELHCCPVHRPOHZUSERSFS"
keyLength=6; firstWordLength = 9
Time taken to Brute Force Crack the Password: 5098.401610851288 seconds
Decrypted message:
EDUCATIONISWHATREMAINSAFTERONEHASFORGOTTENWHATONEHASLEARNEDINSCHOOL

6. "VVVLZWWPBWHZDKBTXLDCGOTGTGRWAQWZSDHEMXLBELUMO"
keyLength=7; firstWordLength = 13
Time taken to Brute Force Crack the Password: 98230.84074521065 seconds
Decrypted message: INTELLECTUALSSOLVEPROBLEMSGENIUSESPREVENTTHEM

As noted by the resulting times to decrypt each message, keyLength plays a significant role in determining the efficiency of password cracking. Regarding the brute force decryption of a message, the number of possible keys with keyLength, n, is 26^n, assuming a 26 character alphabet. Thus, resulting in a big-O time complexity of O(26^n) where the time it takes to decrypt a message grows exponentially in respect to the keyLength. For example, to decrypt a message with keyLength 4, the program would iterate through 26^4 combinations, resulting in a time complexity O(26^4).