APRIL 23, 2018

COMP 707

PROJECT REPORT

VEROSHA PILLAY
STUDENT NUMBER: 214539347

1. Ciphers Overview

1.1 Caesar Cipher

The Caesar cipher is a type of substitution cipher in which each character in the plaintext is shifted a number of spaces down the alphabet. E.g.: A shift of 2 will result in the letter “A” now being “C”.

<u>Encryption:</u> In order to encrypt a message using Caesar cipher, the first step is to translate all characters to numbers. Then pick a shift key value. Using the formula:

$$e(x) = (x + k)\ mod\ number\ of\ characters\ used$$

Where e(x) is the encryption function,

x is character that is being encrypted and k is the shift key value

NB: result is a number i.e.: position value of what the letter will be translated to.

<u>Decryption:</u> In order to decrypt a message, we need to move backwards by the number of the shift key value. Using the formula:

$$d(x) = (x - k)\ mod\ number\ of\ characters\ used$$

Where d(x) is the decryption function,

x is character that is being encrypted and k is the shift key value

NB: result is a number i.e.: position value of what the original letter.

<u>Caesar Cryptanalysis:</u> Cryptanalysis is the art of breaking codes and ciphers.

There are 2 methods which can be used to break the Caesar cipher:

1. Brute Force

   This involves cycling through every possible key i.e. 1 to the total number of characters/symbols used until one is found which allows the cipher text to be converted into plaintext. Decrypting the ciphertext using each key and determine the fitness of each decryption.

2. Frequency Analysis:

   This involves making use of the frequencies of each letter in the alphabet. Work out the frequencies of letters or symbols in the ciphertext and compare the results to the letter frequencies in the language. Then try to make guesses for words or letters e.g., a lone letter in English will be I or A therefore any lone letter in the ciphertext has a high chance of being either A or I.

1.2 Vigenere Cipher

This involves using two or more cipher alphabets to encrypt a message. The letters in the Vigenere cipher are shifted by different amounts, normally done using a word or phrase as the encryption key.

Encryption: Given a keyword, encrypt each letter of the message taken in the left-most column to the letter in the keyword-letter column. E.g.: Keyword=**SMURF**, Thus, the first five letters of the message use the alphabets corresponding the "S", "M", "U", "R", and "F" columns. So, the Vigenere code with this keyword is really five Caesar shifts used in a cyclical fashion.

Decryption: To decrypt, pick a letter in the ciphertext and its corresponding letter in the keyword, use the keyword letter to find the corresponding row, and the letter heading of the column that contains the ciphertext letter is the needed plaintext letter.

Vigenere Cryptanalysis: There are various methods which can be used to break the Vigenere cipher

1. Kasiski examination

   The Kasiski Test uses the occasional aligning of groups of letters with the keyword to determine the length of the keyword. This will produce repeated groups of letters in the ciphertext. By counting the number of letters between the beginnings of these repeated groups of letters and finding a number which is the multiple of those distances, we can estimate the length of the keyword. Then we can line up the ciphertext in n columns, where n is the length of the keyword. Each column can be treated as the ciphertext of a monoalphabetic substitution cipher. As such, each column can be attacked with frequency analysis.

   Steps Involved:

   - Determine the key length
   - The idea is that if two trigrams in the plaintext occur at a distance apart which is a multiple of key length, then they will encrypt to the same trigram in ciphertext.
   - We look for trigrams which occur more than once in the ciphertext, and speculate that their distances apart may be multiples of the key length.
   - For each trigram in the ciphertext that occurs more than once, we compute the GCD of the collection of all distances apart of its occurrences.

- Then guess that the keyword length is a divisor of at least one of these great common divisors.

2. Friedman Test

Friedman test is done by finding the Incidence of Coincidence for a sample of ciphertext which can indicate whether or not a polyalphabetic substitution has been used to encrypt a message. The Incidence of Coincidence is the probability that two randomly selected letters are the same.

Steps Involved:

- Determine the key length
- Find the Incidence of Coincide
- Given two streams of characters. If one (or both) of two streams are completely random, the index of coincidence would be expected to be 1/26 = 0.038 – if all the letters are random, then the probability is 1/26 that the two characters at the *i* - th place will match.

3. Frequency Analysis

Once the length of the key is known, the ciphertext can be rewritten into that many columns, with each column corresponding to a single letter of the key. Each column consists of plaintext that has been encrypted by a single Caesar cipher; the Caesar key (shift) is just the letter of the Vigenère key that was used for that column. Using methods similar to those used to break the Caesar cipher, the letters in the ciphertext can be discovered.

4. Key Elimination

The Vigenere cipher with normal alphabets essentially uses modulo arithmetic, which is commutative. So if the key length is known (or guessed) then subtracting the cipher text from itself, offset by the key length, will produce the plain text encrypted with itself. If any "probable word" in the plain text is known or can be guessed, then its self-encryption can be recognized, allowing recovery of the key by subtracting the known plaintext from the cipher text.

2. Code (with comments)

```
#student number 214539347
#Project_Main.py

import string
import math
import sys
from pyperclip import copy

#Global use
#List for Caesar cipher use
Assigned_List = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L",
"M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",
        "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n",
"o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
        "1", "2", "3", "4", "5", "6", "7", "8", "9", "0",
        " ", "!", "?", ":", ";", ",", ".", "-", "(", ")"]

#Vigenere cipher use
Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
shift = 1
matrix = [ Alphabet[(i + shift) % 26] for i in range(len(Alphabet)) ]
#["B","C","D","E".......“Z","A"]


#Vigenere Cryptanalysis use
vig_Alphabet = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L",
"M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

#dictionary with coressponding frequency for each letter in the Alphabet
relative_Frequencies=  { "a": 8.167, "b": 1.492, "c": 2.782, "d": 4.253, "e":
12.702, "f": 2.228, "g": 2.015, "h":6.094,
                         "i": 6.966, "j": 0.153, "k": 0.772, "l": 4.025, "m":
2.406, "n": 6.749, "o":7.507, "p":1.929,
                         "q": 0.095, "r": 5.987, "s": 6.327, "t": 9.056,  "u":
2.758, "v": 0.978, "w": 2.360, "x": 0.150,
                         "y": 1.974, "z": 0.074  }




def Caesar_Encryption(plaintext,key):
    ciphertext=""

    for letter in plaintext:

        if(letter in Assigned_List):
            char=Assigned_List.index(letter)+1 # get the index of the
character
            enc_Formula=((char+int(key))%(len(Assigned_List)))-1 #formula for
encryption
            ciphertext+=(Assigned_List[enc_Formula])

        else:
            ciphertext+=letter

    return ciphertext



def Caesar_Decryption(ciphertext,key):
    plaintext=""

    for letter in ciphertext:

        if(letter in Assigned_List):
            char=Assigned_List.index(letter)+1 # get the index of the
character
            dec_Formula=((char-int(key))%(len(Assigned_List)))-1 #formula for
decryption
            plaintext+=(Assigned_List[dec_Formula])

        else:
            plaintext+=letter  #Accounts for any characters that are not in
the Assigned_List


    return plaintext



def Vigenere_Encryption(key):
    ciphertext = []
    check = 0

    with open("Encryption.txt") as fileobj:
        for line in fileobj:
            print("\nPlainText:" ,line)
            for char in line.upper(): # each character value in the textfile
                if char not in Alphabet:
                    ciphertext.append(char)
                    continue
                else:
                    if char not in Alphabet:  #Accounts for any characters
which are not in the Vigenere Alphabet
                        ciphertext.append(char)
                        continue
                    else:
                        if (check % len(key) == 0):
                            check = 0
                        else:
                            check

                    resulting_Position = (Alphabet.find(char) +
matrix.index(key[check])) % 26 #Calculation of the position

#find() will determine if the letter occurs in Alphabet-index returned

                    ciphertext.append(matrix[resulting_Position]) #symbol
added to ciphertext
                                                                  #including
special symbols and numbers
                    check += 1

            return ciphertext



def Vigenere_Decryption(key):
    plaintext = []
    check = 0

    with open("Vigenere_decryption.txt") as fileobj:
        for line in fileobj:
            print("\nCipherText:" ,line)
            for char in line: #each character value in the textfile

                if char not in Alphabet:
                    plaintext.append(char)
                    continue
                else:

                    if (check % len(key) == 0):
                        check = 0
                    else:
                        check

                    resulting_Position = (matrix.index(char) -
matrix.index(key[check])) % 26 #Calculation of the position

                    plaintext.append(Alphabet[resulting_Position]) #symbol
added to plaintext
                    check += 1
            return plaintext


def Caesar_Cryptanalysis(plaintext):
    Lowercase_text=plaintext.lower()
    match_score=0
    Dictionary=""

    text=Lowercase_text.split(" ") #text split by space and added to a list

    with open("dictionary.txt") as fileobj: #textfile which contains 100 of
the most common words used in the english language
        for line in fileobj:
            Dictionary+=line
            Lowercase_Dictionary=Dictionary.lower()
            words=Lowercase_Dictionary.split("\n") #Dictionary words added to
a list called words

    for i in words: # Words in the plaintext will be checked if it's in the
dictionary(ie:list which dictionary words were added)
        if (i in text):
            match_score+=1  #A match_score is given for all matched words

    return match_score



#Functions related to Vignenere Cryptanalysis below
#Using Kasiski Examination

def sequence_shift(sequence, val): #takes the letters from vig_Alphabet and
it's index and apply specific shift
    val= val % len(sequence)
    return sequence[val:] + sequence[:val]

#sequence_shift output will look like this when called by
Vig_decrypt_Cryptanalysis,sample of some the first few lines:
'''Shift is: ['S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R']
   Shift is: ['M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
   Shift is: ['U', 'V', 'W', 'X', 'Y', 'Z', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T']
'''

def Word_Distance(ciphertext): # Goes through the ciphertext and returns a
list of the distance between matching words(trigrams and higher)
```

```
    distanceList = []

    while(len(distanceList)<10):

        for i in range(3,6): #Checks for matches from 3 to 6

            for j in range(len(ciphertext)-i):

                word = ciphertext[j:j+i]
                index = ciphertext.find(word,j+i)

                if(index!=-1):
                    distanceList.append(index-j)

    return distanceList



def Frequency(ciphertext):
    Frequency_Dict = {key:0  for key in relative_Frequencies.keys()}
    Sum = 0
    shift=0
    lower=0

    for letter in ciphertext:
        Frequency_Dict[letter.lower()] += 1
        Sum += 1

    for key in Frequency_Dict.keys():
        Frequency_Dict[key] = Frequency_Dict[key]/Sum


    score = [0 for i in range(26)] #A comparison is done between
Frequency_Dict to relative_Frequencies to determine number of shifts

    values = [0 for i in range(26)]

    while(shift < 26):
        index = 0
        for key in relative_Frequencies.keys():
            value = Frequency_Dict[vig_Alphabet[(shift+index)%26].lower()]

            values[index] = abs(value-relative_Frequencies[key])
            index += 1

        for counter in range(26):
            score[shift] +=
values[(counter+1)%26]+values[counter]+values[counter-1]
```

```
        if(score[shift] < score[lower]):
            lower = shift
        shift += 1

    return lower



#Form strings from the ciphertext of the letters that have been encrypted by
the same subkey
#Returns every nth letter for each keyLength set of letters in ciphertext
def nthLetters(factor, substring):
    result = []
    sub = ""
    for repeat in range (factor):
        index = repeat

        while index < len(substring):
            sub += substring[index]
            index += factor

        result.append(sub)
    return result



def Sub_Seq(l, s):
    m = s[l:]
    i = -1
    while(m):
        yield m #return a generator
        m = s[l:i]
        i -= 1



def resetFrequencies(): #Divides each letter frequency value by the sum of all
the frequency values, ie percentage contribution to total
    freq_total = 0

    for k in relative_Frequencies.keys(): #Totals values
        freq_total += relative_Frequencies[k]

    for key in relative_Frequencies.keys():
        relative_Frequencies[key] = relative_Frequencies[key]/freq_total # for
each key in the list-get the value of it to the value divided by the total
value
```

```python
def Key_Length(ciphertext): # Analysing patterns and distances between
patterns in ciphertext
    length = 3
    distanceList = Word_Distance(ciphertext)
    factors = {key:0 for key in range(3,27)} #Determine the factors

    for i in distanceList:

        for j in range(26,2,-1):

            if(i % j == 0):
                factors[j] += 1

    for keys in factors.keys(): #Determine key length

        if(factors[keys] > factors[length]): #if the frequency of the key in
the factors list is more than the current length, then length is updated
            length = keys

    GCD = math.gcd(distanceList[0] ,
math.gcd(distanceList[1],math.gcd(distanceList[2],
math.gcd(distanceList[3],distanceList[4])))) #Finds the GCD of the first, and
the combination of the next 3,

#than in turn is calculated the same as the gcd between itself

#and the next 2

    if(GCD<=26 and GCD>length):
        length = GCD

    return length



def Vig_Examination(ciphertext):
    key_value=""
    cipherNpSpace = ciphertext.replace(" ","")

    length = Key_Length(cipherNpSpace) #Get the Key length

    matrix = [[" " for l in range(length)]for k in
range(int(len(ciphertext)/length))]

    originalLength = len(cipherNpSpace)

    for v in range(int(originalLength/length)):
        for w in range(length):
                matrix[v][w] = cipherNpSpace[w] #sets this matrix value to the
character at the index of w of the ciphertext with no spaces
        cipherNpSpace = cipherNpSpace[length:]


    for item in range(length):
        s = ""
        for rows in range(int(originalLength/length)):
            s += matrix[rows][item]
        key_value += vig_Alphabet[Frequency(s)]

    return key_value



def Vig_decrypt_Cryptanalysis (ciphertext, key): #Decipher the Ciphertext and
return the message
    key = key.upper() #Key must be in all CAPS
    plaintext = ""
    count = 0

    for index in range(len(ciphertext)):

        if (ciphertext[index] == ' '): #need to check if ciphertext index is a
space and increment count
            plaintext += ' '
            count += 1

        else:
            col = ciphertext[index] #column is the index letter of the
ciphertext
            row = key[(index-count)%len(key)]  #row is the index of the col
letter % key length (character is part of the key)

            queueL = sequence_shift(vig_Alphabet, vig_Alphabet.index(row))

            plaintext += vig_Alphabet[queueL.index(col)] # gets the character
from the vig_Alphabet at that index
    print("\nKey : ", key)

    file1 = open("[214539347_[Vigenere_break].txt","w") #Plaintext is written
to a textfile along with the key
    file1.write("Key : ")
    file1.write(key)
    file1.write("\n")
    file1.write(plaintext)
    file1.close()
    return plaintext


def testKeys(possibleKeysList, ciphertext):
    keyTest = ""
    for key in possibleKeysList:
        keyTest += (Vig_decrypt_Cryptanalysis(ciphertext, key))

    return keyTest


#Kasiski used to break Vigenere Ciphertext
def Vigenere_Cryptanalysis (ciphertext):
    plaintext = ""
    Possible_Keys = []

    for i in ciphertext:
        if (ciphertext == ""):
            cipherNoSpace = ciphertext.replace(" ", "") #replace ciphertext
character with a space now with no space

            substring = longestRepetitiveSubstring(cipherNoSpace) #get the
longest substring found

            nthLetterList = nthLetters(substring[1], substring[0])

            return (nthLetterList)

        resetFrequencies()


        Possible_Keys.append(Vig_Examination(ciphertext))

        keyVal = testKeys(Possible_Keys, ciphertext) #get the Key used

        plaintext += keyVal #key is added to the plaintext and returned

        return (plaintext)



def processInput(screen, inputLine):

    if screen == 0:
        if inputLine == 1:
            print("\nCaesar Cipher Options: " + "\n----------------------")
            print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis")
            return 1
        elif inputLine == 2:
            print("\nVigenere Cipher Options: " + "\n------------------------
")
            print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis")
            return 2
        elif inputLine == 3:
            exitProgram()
    elif screen == 1:
        if inputLine == 1:
            plaintext=""
            print("\nKey being used : 17")
            key=17 #10 + 7 (214539347)

            with open("Encryption.txt") as fileobj: #Opening the textfile with
the plaintext

                for line in fileobj:
                    for char in line: #each character value in the textfile
                        plaintext+=char

                    print("\nPlaintext : ",plaintext,end="")
                    print("\n")
                    result=Caesar_Encryption(plaintext,key)
            print("Ciphertext :",result,end="")
            print("\n")

            file1 = open("[214539347]_[Caesar_encrypt].txt","w")
            file1.write(result)
            file1.close()

            copy(result) #copy to clipboard

            print("\nCaesar Cipher Options: " + "\n----------------------")
            print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis" +
"\n4.Back")
            return 1
        elif inputLine == 2:
            ciphertext=""
            print("\nKey being used: 14")
            key=14

            with open("Caear_decryption.txt") as fileobj: #Opening the
textfile with the ciphertext

                for line in fileobj:
                    for char in line: #each character value in the textfile
                        ciphertext+=char
                    print("\nCiphertext : ",ciphertext,end="")
                    print("\n")
```

```
          result=Caesar_Decryption(ciphertext,key)
        print("Plaintext :",result,end="")
        print("\n")

      #plaintext is written to a new file
      file1 = open("[214539347]_[Caesar_decrypt].txt","w")
      file1.write(result)
      file1.close()

      copy(result) #copy to clipboard
      print("\nCaesar Cipher Options: " + "\n----------------------")
      print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis" +
"\n4.Back")
      return 1
    elif inputLine == 3:
      ciphertext=""
      score_List=[]
      with open("Caesar_break.txt") as fileobj: #Opening the textfile
with the ciphertext

              for line in fileobj:
                for char in line: #each character value in the
textfile
                  ciphertext+=char
                print("\nCiphertext : ",ciphertext,end="")
                print("\n")
                for key in range(1,len(Assigned_List)):
                  result=Caesar_Decryption(ciphertext,key)
#Brute force,going through every key
                  word_match=Caesar_Cryptanalysis(result)
#and return a match score from the Caesar_Cryptanalysis()

                  score_List.append(word_match) #scores are
added to a list
                max_Score=max(score_List) #the highest score
ie: score which had the most matches
                if(max_Score>=10):
                  decoded=score_List.index(max_Score)+1
#index value of that score
                  print("Key : ", decoded)

message=Caesar_Decryption(ciphertext,decoded) #decrypt the message with the
key found
                  print("\nPlaintext :",message,end="")
                  print("\n")
```

```
                                    #plaintext and key are written to a new
file
                                    file1 =
open("[214539347_[Caesar_break].txt","w")
                                    file1.write(message)
                                    file1.write("\nKey:")
                                    file1.write(str(decoded))

                                    file1.close()

                                    copy(message) #copy to clipboard
                                    print("\nCaesar Cipher Options: " + "\n---
-------------------")
            print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis" +
"\n4.Back")
            return 1
        elif inputLine == 4:
            return(displayMenu())
    elif screen == 2:
        if inputLine == 1:
            print("\nKey being used: Verosha")
            key="Verosha"          #Key=Verosha
            key=key.upper()
            result=Vigenere_Encryption(key)
            r=''.join(result)
            print("\nCiphertext: {0}".format(''.join(result)))  #Letters from
the List are merged
            print("\n")

            f=''.join(result)
            file1 = open("[214539347]_[Vigenere_encrypt].txt","w")
            file1.write(f)
            file1.close()

            copy(r) #copy to clipboard
            print("\nVigenere Cipher Options: " + "\n------------------------
")
            print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis" +
"\n4.Back")
            return 2
        elif inputLine == 2:
            print("\nKey being used: Neuschwanstein")
            key="Neuschwanstein"
            key=key.upper()
            result=Vigenere_Decryption(key)
            r=''.join(result)
            print("\nPlaintext: {0}".format(''.join(result)))
            f=''.join(result)
            file1 = open("[214539347]_[Vigenere_decrypt].txt","w")
            file1.write(f)
            file1.close()

            copy(r) #copy to clipboard
            print("\nVigenere Cipher Options: " + "\n------------------------
")
            print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis" +
"\n4.Back")
            return 2
        elif inputLine == 3:
            ciphertext=""
            with open("Vigenere_break.txt") as fileobj: #Opening the textfile
with the ciphertext

                for line in fileobj:
                    for char in line: #each character value in the textfile
                        ciphertext+=char
            print("\nciphertext :",ciphertext)
            result=(Vigenere_Cryptanalysis (ciphertext))

            print("\nplaintext :" , result)
            print("\n")

            copy(result) #copy to clipboard
            print("\nVigenere Cipher Options: " + "\n------------------------
")
            print("\n1.Encrytion" + "\n2.Decryption" + "\n3.Cryptanalysis" +
"\n4.Back")
            return 2
        elif inputLine == 4:

            return(displayMenu())



def displayMenu():
    print("\nMenu Options " + "\n-------------")
    print("\n1.Caesar Cipher" + "\n2.Vigenere Cipher " + "\n3.Exit")
    return 0

def exitProgram():
    sys.exit()


def main():
    menuInput = -1
    screen = 0  #0: Menu, 1:Caesar, 2:Vigenere


    displayMenu()
    while (True):

        menuInput = int(input("\nPlease make a selection : "))

print("______________________________________________________________________________
______________________________________________________")

        screen = processInput(screen, menuInput)


main()
```

3. Results

Encryption Results:

1. **Caesar Cipher**

Text file: [214539347]_[Caesar_encrypt]

(key=17)

Ciphertext:

kyvH7r3rtvHt6473v;Hz0Hv5 v9vuH y96!xyH yvH0,44v 9ztr3HXr vy6!0vHw3r52vuHs,H :6H0
rz9H 6:v90NHkyvHvr0 :r9uO76z5 z5xHxr vHs!z3uz5xHz0H yvH653,H0 9!t !9vH6wH
yvH7r3rtvH:y60vH:r33Hr9vrHz0Hwr0yz65vuHz5HyzxyOt65 9r0 Ht636!90LH yvHv;
v9z69H:r330Hr9vHtr0vuH:z yH9vuHs9zt20MH yvHt6!9 Hw965 0H:z yH,v336:H3z4v0
65vNHkyvH966wHt695ztvHz0H0!996!5uvuHs,H7z55rt3v0NHkyvH!77v9Hw3669H6wH yvHXr
vy6!0vHz0H0!946!5 vuHs,HrHt96:O0 v77vuHxrs3vHr5uHyv3uHc!u:zxHZZ0Hwz90 H36uxz5xHr
Hev!0ty:r50 vz5MHw964H:yztyHyvH6ttr0z65r33,H6s0v9?vuH yvHs!z3uz5xH:692Hsvw69vH
yvHyr33H:r0Ht6473v vuNHkyvHx96!5uHw36690H6wH yvHXr vy6!0vH:v9vHz5 v5uvuH
6Hrtt6446ur vH yvH0 rs3v0NHkyvH7r00rxvH y96!xyH yvHXr vy6!0vMHt96:5vuH:z yH
yvH96,r3HSr?r9zr5Ht6r H6wHr940MH3vru0Huz9vt 3,Hz5 6H yvHt6!9 ,r9uNHkyvHt6!9
,r9uHyr0H :6H3v?v30MH yvH36:v9H65vHsvz5xHuvwz5vuH 6H yvHvr0 Hs,H yvHXr
vy6!0vHr5uH 6H yvH569 yHs,H yvHw6!5ur z650H6wH yvH06Otr33vuHivt
r5x!3r9Hk6:v9Hr5uHs,H yvHxr33v9,Hs!z3uz5xNHkyvH06! yv95Hv5uH6wH yvHt6!9
,r9uHz0H67v5MHz47r9 z5xHrH?zv:H6wH yvH0!996!5uz5xH46!5 rz5H0tv5v9,NHR Hz 0H:v0
v95Hv5uMH yvHt6!9 ,r9uHz0Huv3z4z vuHs,HrHs9zt2vuHv4sr524v5

MH:y60vH763,x65r33,H796 9rt z5xHs!3xvH4r920H yvHty6z9H6wH yvH69zxz5r33,H7961vt vuHtyr7v3LH yz0H y9vvO5r?vHty!9tyMH5v?v9Hs!z3 MH:r0Hz5 v5uvuH 6Hw694H yvHsr0vH6wHrHFGO4v 9vHP(FBOw QH2vv7MH yvH73r55vuHtv5 9v7zvtvH6wH yvHr9tyz vt !9r3Hv50v4s3vNHRHw3zxy H6wH0 v70Hr H yvH0zuvHxz?v0Hrttv00H 6H yvH!77v9H3v?v3N

2. **Vigenere Cipher**

Text file: [214539347]_[Vigenere_encrypt]

(key=”Verosha”)

Ciphertext:

OLV DSSAXI TCEWLZB ZG WUTZVVR LORJYXV LOE NCDAWARDGRZ YHTZLFIKL FGEEYWK BT XNC KAADV KCOLRN. XYS WHSOARFV-WODRKWFN GVXV PMPLYMEU AZ TCI FBDF SOVLQLBRZ SW HZL PVPRQW DHJWV KSSL VVVO AZ FVWYWGUEY ME VANH-XSEHJHSO GFZGBRN; XYS WETZVZCJ DAGPJ OJL CVWVR OPTC VVR TYIXOJ, HZL CJYIH XYOIXJ KAAH TICZGD LDQVGLVNZ. XYS JVOA GFFFPCZ MJ GMYRJYERWK BT TZBFHCGIJ. HZL UKTVF XSOJV FT LOE BEKSZVUNI ZG KBRHSLBLLD WC R QJVW-NXVDHLD BESZW HNY LVZV SUYAZU APS AMIGL SOYKZBY HT IILGUOWVRJHWPN, AVFA OOIXL YS GJCVWZCFHLGC FPKLRQIU HZL BPMCRAUG RSIY TLFJVV HZL HVPC KSZ CJQGZWAEY. XYS YYOPRU TDVOMW FT LOE BEKSZVUNI NSJL IIXVBVLD OS RQUVMHSUOLL TCI JHSILZW. KVW WANWRUW AHMSLUZ AHZ KRHWOOPWV, QJVWIIU KAAH OLV FGFAG FRJSYIVR TCSA OA EIAK, SEVHJ RAYEXXCM AUTJ XYS UVUMXPOJK. TCI TCMYTTEIR ZHS OAF ZWCEGW, KVW SORII CFL BZMEU VLFDRVR LV TCI VOKA BT XYS YHTZLFIKL AIH KC LOE ISIHZ IY OLV TGBNYEKWGUS JJ KVW ZO-XECZWK RZGKOFNUGEI HGDEM EER TF TCI XODSEMC SIASDDRX. HZL SJYKVWYN ZRU CX AHZ GFIJAYVVU WK VPZR, ZAHHROMEU S CIZA FT LOE NYIFGBNYMEU EVUIXRWF ZCZRVFQ. HT DXJ KWZTZVE SFK, TCI TCMYTTEIR AZ DZPZAAAEY FP O TYIXOVR WTBVRBAWUT, RLFGW WOGCXCFHLGC GFGARVGKWFN BPPXS EHRFW KVW JHJMI CX AHZ SIWYPNVPCM HYOEITHWK CCEGSD; AHDW KVJLE-IEMS UOUMGY, BWCEM FLWDA, WVW ZBLLNYIU HG MOMQ KVW IANI FT S 90-TEOVV (295-TL) REZT, KVW WLVRESV JEIXISHPEXI FT LOE VVTVAAEXXLFSS EIWVATSE. V JCWYOT JJ JHWWS VX KVW ZIYI XWNLS VGTSKZ TJ XYS MWPZV CSNLL.

Decryption Results:

1. **Caesar Cipher**

Text file: [214539347]_[Caesar_decrypt]

(key=14)

plaintext:

Theyre Dashing and daring, Courageous and caring, Faithful and friendly, With stories to share. They march through the forest, They sing out in chorus, Marching along, As their song fills the air. Gummi Bears, Bouncing here and there and everywhere. High adventure that’s beyond compare, They are the Gummi Bears. Well, Magic and mystery, Are part of their history, Along with the secret, Of gummiberry juice. Their legend is growing, They take pride in knowing, They'll do what is right, With whatever they do. Gummi Bears, Bouncing here and there and everywhere. High adventure that's beyond compare, They are the Gummi Bears. Well, my girl, dolores She march through the forest She takes one of each And sticks it down her pants? I like the red ones And I like the yellow ones We bite off their head And we give them to our friends Gummi Bears, Bouncing here and there and everywhere. High adventure that's beyond compare, They are the Gummi Bears. Gummi Bears, When a friend's in danger they'll be there, Lives and legends that we all can share, They are the Gummi Bears, They are the Gummi Bears, They are the Gummi Bears!

2. **Vigenere Cipher**

Text file: [214539347]_[Vigenere_decrypt]

(key="Neuschwanstein ")

plaintext:

IN A HOLE IN THE GROUND THERE LIVED A HOBBIT NOT A NASTY DIRTY WET HOLE FILLED WITH THE ENDS OF WORMS AND AN OOZY SMELL NOR YET A DRY BARE SANDY HOLE WITH NOTHING IN IT TO SIT DOWN ON OR TO EAT IT WAS A HOBBIT HOLE AND THAT MEANS COMFORT IT HAD A PERFECTLY ROUND DOOR LIKE A PORTHOLE PAINTED GREEN WITH A SHINY YELLOW BRASS KNOB IN THE EXACT MIDDLE THE DOOR OPENED ON TO A TUBE SHAPED HALL LIKE A TUNNEL A VERY COMFORTABLE TUNNEL WITHOUT SMOKE WITH PANELLED WALLS AND FLOORS TILED AND CARPETED PROVIDED WITH POLISHED CHAIRS AND LOTS AND LOTS OF PEGS FOR HATS AND COATS THE HOBBIT WAS FOND OF VISITORS THE TUNNEL WOUND ON AND ON GOING FAIRLY BUT NOT QUITE STRAIGHT INTO THE SIDE OF THE HILL THE HILL AS ALL THE PEOPLE FOR MANY MILES ROUND CALLED IT AND MANY LITTLE ROUND DOORS OPENED OUT OF IT FIRST ON ONE SIDE AND THEN ON ANOTHER NO GOING UPSTAIRS FOR THE HOBBIT BEDROOMS BATHROOMS CELLARS PANTRIES LOTS OF THESE WARDROBES HE HAD WHOLE ROOMS DEVOTED TO CLOTHES KITCHENS DINING ROOMS ALL WERE ON THE SAME FLOOR AND INDEED ON THE SAME PASSAGE THE BEST ROOMS WERE ALL ON THE LEFT HAND SIDE GOING IN FOR THESE WERE THE ONLY ONES TO HAVE WINDOWS DEEP SET ROUND WINDOWS LOOKING OVER HIS GARDEN AND MEADOWS BEYOND SLOPING DOWN TO THE RIVER

Cipher Cryptanalysis Results:

1. **Caesar Cipher break**

Text file: [214539347_[Caesar_break]

What is a hobbit? I suppose hobbits need some description nowadays, since they have become rare and shy of the Big People, as they call us. They are (or were) a little people, about half our height, and smaller than the bearded Dwarves. Hobbits have no beards. There is little or no magic about them, except the ordinary everyday sort which helps them to disappear quietly and quickly when large stupid folk like you and me come blundering along, making a noise like elephants which they can hear a mile off. They are inclined to be fat in the stomach; they dress in bright colours (chiefly green and yellow); wear no shoes, because their feet grow natural leathery soles and thick warm brown hair like the stuff on their heads (which is curly); have long clever brown fingers, good-natured faces, and laugh deep fruity laughs (especially after dinner, which they have twice a day when they can get it). Now you know enough to go on with.

Key:19

2. **Vigenere Cipher break**

Text file: [214539347_[Vigenere_break]

Key : SMURF

LA LA LA LA LA LA SING A HAPPY SONG LA LA LA LA LA LA SMURF IT ALL DAY LONG LA LA LA LA LA LA SMURF ALONG WITH ME LA LA LA LA LA LA SIMPLE AS CAN BE NEXT TIME YOURE FEELING BLUE JUST LET A SMILE BEGIN HAPPY THINGS WILL COME TO YOU SO SMURF YOURSELF A GRIN OOOOOO I HATE SMURFS ILL GET YOU ILL GET ALL OF YOU IF ITS THE LAST THING I EVER DO HEHEHEHE LA LA LA LA LA LA NOW YOU KNOW THE TUNE YOULL BE SMURFING SOON