Assignment 1

Monday, 4 March 2024 11:10 AM

Student Id: **a1824665**

Question 1:
(1 point) There is a secret passphrase embedded in the file called "test.txt" in the folder /home/student/linux_basics/q01. The secret can be found following the line that begins with the word "And" and ends with "it". Use grep with regular expression to locate this line and the line following it. What is the passphrase?

Answer:

```
student@hacklabvm:~/linux_basics/q01$ grep -A 1 '^And.*it$' test.txt
And give't Iago: what he will do with it
csf2024s1_{lanceteer-versify-phlogogenous}
student@hacklabvm:~/linux_basics/q01$ _
```

The secret Passphrase is "csf2024s1_{lanceteer-versify-phlogogeneous}" which can be found using the command "**grep -A 1 '^And.*it$' test.txt**" as shown in the screenshot above.

The grep command is used with the flag -A 1 which displays the specified amount of lines (in this case 1) after the specified pattern. The usual syntax for this is "**grep -A number-of-lines 'pattern' your-file-name.txt**".

The pattern search is further refined using more flags. The expression used is '**^And.*it$' test.txt**'. In this expression, '**^**' ensures that the string matches at the beginning of the line ('^And....' search for a line that starts with And). ' **.*** ' then matches any characters between 'And' and the next specified pattern 'it'. The '**$**' at the end of the pattern specifies that the string 'it' should be at the end of the line being searched for.

Question 2:
(1 point) In the folder /home/student/linux_basics/q02, there is a file called "here.txt" that contains passphrases. Find the passphrase that occur exactly 14 times.

Answer:

```
student@hacklabvm:~/linux_basics/q02$ grep -oE '\S+' here.txt | sort | uniq -c | awk '$1 == 14 {prin
t $2}'
csf2024s1_{lanceteer-versify-phlogogenous}
student@hacklabvm:~/linux_basics/q02$
```

To find the passphrase that occurs exactly 14 times, the command used is " **grep -oE '\S+' here.txt | sort | uniq -c | awk '$1 == 14 {print $2}'** ".

Where the **-o** flag (only matching) outputs only the matched parts of each line rather than the entire line. The **-E** flag enable basic regular expressions for more flexible searching. In the pattern being searched '**\S+**' represents any non whitespace character and the plus specifies one or more occurrences of the specified preceding element.

The **sort** command then arranges the passphrases found in alphabetical order which then makes it easier for **uniq -c** to count each unique occurrence.

The **awk** command then scans and process the sorted list where **$1** refers to the first field (the column of the sorted list passed on by sort command).

Then command **== 14** is used to find the passphrase that occurs 14 times by looking for the column with count = 14 in it.

The print command **{print $2}** is then used to print if count equals 14. i.e. The second column passed by sort contains the variable count.

Question 3:
(1 point) There are lots of files in /home/student/linux_basics/q03. What is the name of the file whose SHA256 sum is 389f0d2df51e5553118e2de48b40e1cc67ae2b477cf6d27ca1faf1c548f78f0c?

Answer:

```
student@hacklabvm:~/linux_basics/q03$ find -type f -exec sha256sum {} + | grep "389f0d2df51e5553118e
2de48b40e1cc67ae2b477cf6d27ca1faf1c548f78f0c"
389f0d2df51e5553118e2de48b40e1cc67ae2b477cf6d27ca1faf1c548f78f0c  ./csf2024s1_{undersense-consenting
-komondorok}
student@hacklabvm:~/linux_basics/q03$ _
```

To find the name of the file with the specified SHA256 sum, the command used is **find -type f -exec sha256sum {} + | grep "389f0d2df51e5553118e2de48b40e1cc67ae2b477cf6d27ca1faf1c548f78f0c"**

Using the **find** command and the **-f** flag (-f specifies files only, no directories), the command calculates the sha256 sum using **-exec sha256sum {}** (the brackets {} passes on all the files from the directory)**.**

The **grep** command then filters the output to include only the file that has the desired SHA256 sum "389f0d2df51e5553118e2de48b40e1cc67ae2b477cf6d27ca1faf1c548f78f0c".

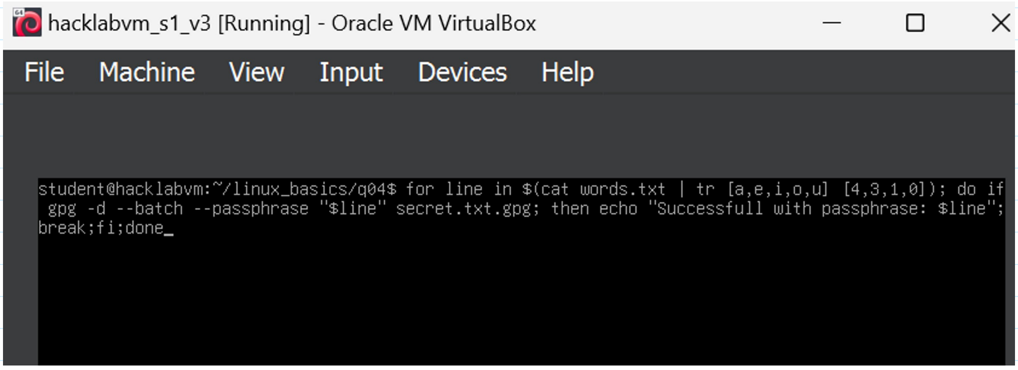
Question 4:
(1 point) Generate a list of passwords from the source file words.txt. Use "l33t" conversion so that a=>4, e=>3, i=>1, and o=>0. For example "hello world" becomes "h3ll0 w0rld". There is a file encrypted using gpg (Gnu Privacy Protection) under /home/student/linux_basics/q04, using the command gpg -c --batch --passphrase <pass>. Unfortunately, we have forgotten the password. Use the "l33t" converted password list to brute-force the encrypted file. This may take a few minutes. Please provide both the correct password and the content of the decrypted file.

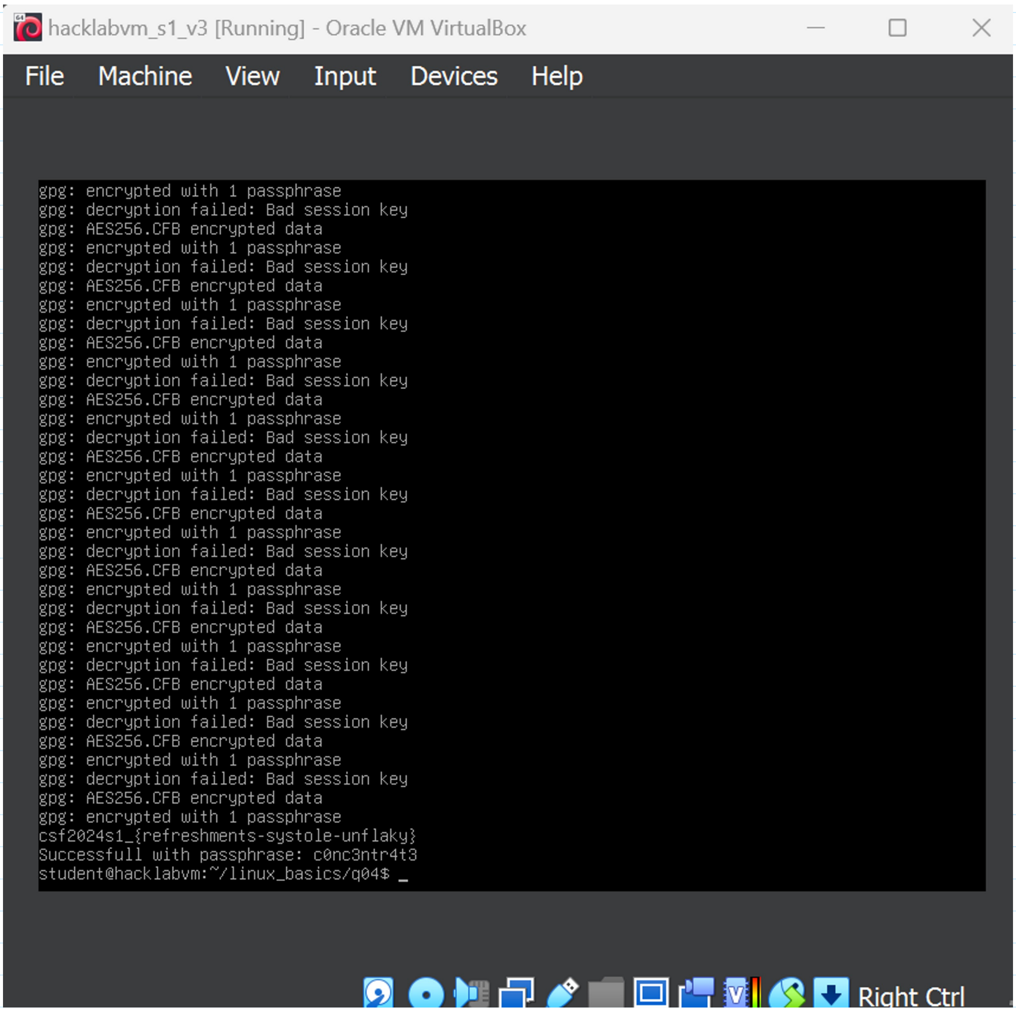
Answer:

A **for** loop is used to iterate over the output of the cat command (possible passphrases) after applying the **tr** command to perform l33t conversion.

The **if** condition then checks whether the gpg command can successfully decrypt the file using the passphrases which are stored in the variable line.

When a passphrase is successful in decrypting, a message is displayed using echo with the passphrase that worked. The break command then breaks out of the loop.



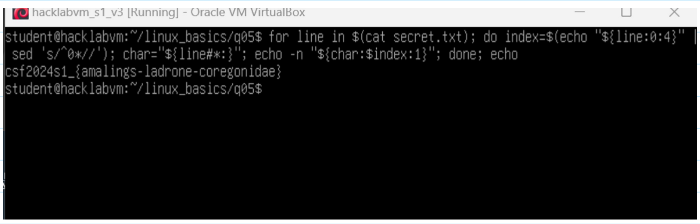


Question 5:
(1 point) Find the flag hidden (encoded?) in the file /home/student/linux_basics/q05/secret.txt. Looks like the cyber.py (Hint: the file cyber.py was used to generate the file...)

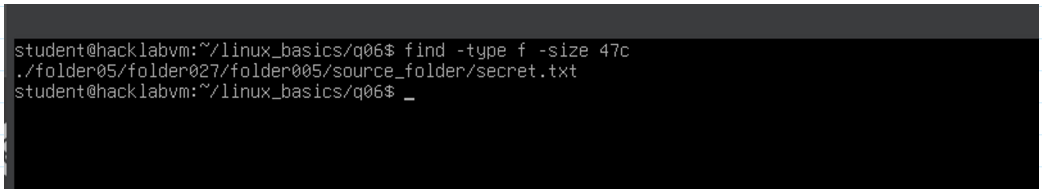
Answer:

A **for** loop is used to iterate over each line of secrect.txt. The "**${line:0:4}**" extracts the first four characters from each line of secret.txt and then applies the **sed 's/^0*//'** command on it which removes the leading zeros and stores it in a variable called line.
char="${line#*:}";** then extracts the substring after the colon from each line and stores it in variable char. The echo command (**echo -n "${char:$index:1}";) then prints the character at the position as specified y the value of 'index' from the 'char' variable. The word 'done' then ends the loop.

The flag generated by the cyber.py is csf2024s1_{amalings-ladrone-corefonidae}.

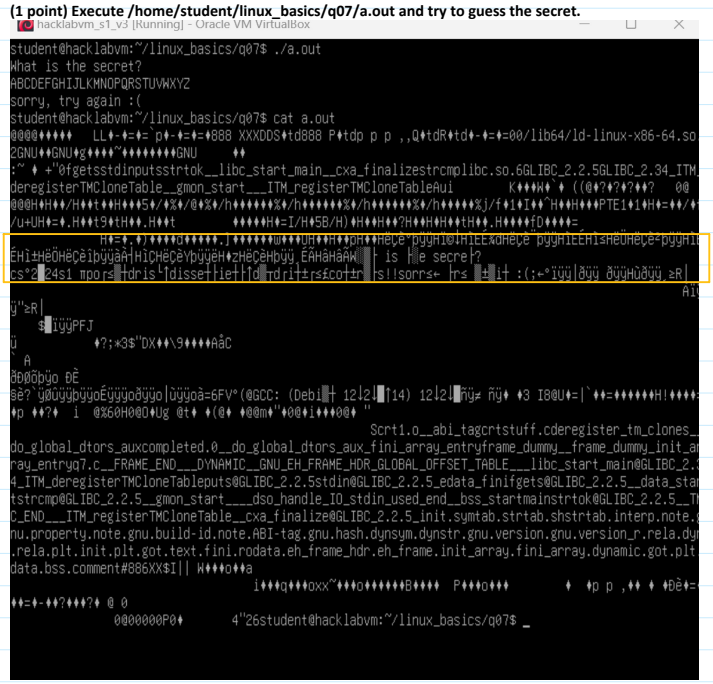


Question 6:
(1 point) There are lots of sub-directories and files under /home/student/linux_basics/q06. Find the file containing the secret. The file has size of exactly 47 bytes long.



To find a file of the exact size, we can use the **find** command. The search can be further narrowed down by using the **type** flag **(-type f for files only, no directories)**, and the **size** flag **(47c, for 47 bytes exactly).**

Question 7:
(1 point) Execute /home/student/linux_basics/q07/a.out and try to guess the secret.



The content of the a.out file is accessible through the **cat** command, as depicted in the provided screenshot.

Upon executing the cat command, the compiled code becomes visible, revealing the comparison made with my input (**csf2024s1_{polyandrism-dissentient-dawdlingly}**).

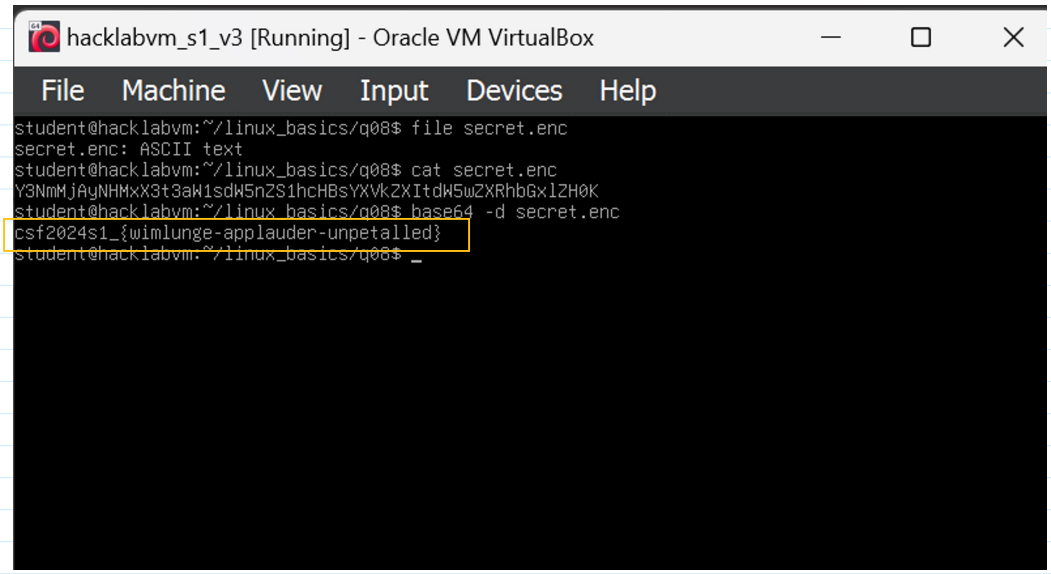
This comparison involves the stored variable and the text 'sorry, try again,' which is presented to the user when they are unable to accurately guess the secret.

Question 8:
(1 point) There is an encoded secret in /home/student/linux_basics/q08 folder. Decode to get the secret.

The file command can be used to find out what type of encoding is used. In this case us-ascii as shown in the screenshot below.

The next step is to decode it, by using the cat command and observing the output of the .enc file, it appears to be base64 encoded.

Then the file can be decoded by using the command **base64 -d secret.enc** which then outputs the secret flag as shown in the screenshot below.



Part 2:

Question 1:
(1 point) Go to /home/student/crypto/q01 on the HacklabVM (either via the graphical interface or via SSH). The file secret.txt.enc has been "encrypted" using this simple crypter code below (see mycrypto.py in the same directory). "Decrypt" the file and find the secret. Is this a good encryption? What is the key space?

Answer:
The command used to decrypt is in the format "**python3 decrytor_file.py encrypted_file_.txt.enc --seed xxxx**". The seed used is the same as specified in the decryptor file as

highlighted in the screenshot below.

```
student@hacklabvm:~/crypto/q01$ python3 mycrypto.py secret.txt.enc --seed 312024
student@hacklabvm:~/crypto/q01$ ls
mycrypto.py  secret.txt.enc  secret.txt.enc.enc
student@hacklabvm:~/crypto/q01$ cat secret.txt.enc.enc
csf2024s1_{outtravel-sargassumaishes-monolocular}
student@hacklabvm:~/crypto/q01$
```

```
#!/usr/bin/python3

import argparse
import os
import sys
import random

def mycrypto (filename):
    with open(filename, 'r') as f, open(filename + '.enc', 'w') as o:
        blob = f.read()
        for b in blob:
            key = random.randrange(255)
            x = ord(b) ^ key
            o.write(chr(x))

def main():
    parser = argparse.ArgumentParser(description='Encrypt (?) a file')
    parser.add_argument('filename', metavar='filename', type=str, help='file to encrypt')
    parser.add_argument('--seed', metavar='seed',type=int,default=312024,help='seed')
    args = parser.parse_args()

    if not os.path.isfile(args.filename):
        print('The file  does not exist')
        sys.exit()

    random.seed(args.seed)
    mycrypto(args.filename)

if __name__== "__main__":
    main()
~
~
~
~
~
~
"mycrypto.py" [readonly][noeol] 30L, 806B                              1,1           All
```

The decryptor used appears to be a XOR Cipher decryptor. XOR Cipher decryptors use the XOR (exclusive or) logical operations to encrypt data.

First, a randomly generated key is chosen. And then the key is used to apply the XOR logic to each byte of the data to encrypt. To decrypt, the same key and the XOR operation is used.

XOR Ciphers can be somewhat secure if a truly random key is chosen that is as long as the message itself. In this case, the key is only 8 bit long which makes it suspectable to brute force attacks. The key space is only 256 possibilities which is relatively small and easy to brute force.

Question 2:

. (1 point) A short, plaintext message has been encrypted using Textbook RSA (using the public exponent) with the same parameters used in the workshop. The encrypted message can be found in /home/student/crypto/q02 folder. Decrypt the secret.

```
n=0x9B51C20306EDE535C8FCAADBC3F3515E52A0D005703DD449BEC66B23E2932313
p=0xC5A047A7C52ED3A2875F7D76C47B555F
q=0xC93268355C09197BBF1659B5522FFACD
e=0x010001
d=0x0D067636BAC6088AD2281E4BFFCACFEFEF9BC1A69FB9E701063DFBAAB436E4C1
```

Answer:
The RSA encrypted secret was decoded using a Python decryptor, as illustrated in the screenshot below. The provided helper functions, as outlined in the questions, were employed for this process. Additionally, a new helper function named `hex_to_decimal` was used to covert the given RSA strings.

The RSA hexadecimal values were converted into a decimal format through the application of the `hex_to_decimal` helper function. Subsequently, the file content was read and stored in a variable named `encrypted_message`. This encrypted message was also converted into an integer representation.

Utilizing the modular operation, the ciphertext was decrypted with the RSA private key ('d', 'n'). The resulting decrypted message, represented as integers, was then transformed back into a string.

The final step involved printing the decrypted message to the terminal.

hacklabvm_s1_v3 [Running] - Oracle VM VirtualBox

```
import binascii

def string_to_int(string):
    return int.from_bytes(binascii.a2b_qp(string),byteorder='big')

def int_to_string(number):
    bin_data = number.to_bytes((number.bit_length() + 7)//8, byteorder='big')
    return binascii.b2a_qp(bin_data).decode("utf-8")

def hex_to_decimal(hex_string):
    return int(hex_string, 16)

n_hex = "0x9B51C20306EDE535C8FCAADBC3F3515E52A0D005703DD449BEC66B23E2932313"
p_hex = "0xC5A047A7C52ED3A2875F7D76C47B555F"
q_hex = "0xC93268355C09197BBF1659B5522FFACD"
e_hex = "0x010001"
d_hex = "0x0D067636BAC6088AD2281E4BFFCACFEFEF9BC1A69FB9E701063DFBAAB436E4C1"

n = hex_to_decimal(n_hex)
p = hex_to_decimal(p_hex)
q = hex_to_decimal(q_hex)
e = hex_to_decimal(e_hex)
d = hex_to_decimal(d_hex)

encrypted_file_path = 'rsa.encrypted'

with open(encrypted_file_path, 'rb') as file:
    encrypted_message = file.read()

enc = hex_to_decimal(encrypted_message)

m = pow(enc, d, n)
decrypted_message=int_to_string(m)

print("Message: ", decrypted_message)

"rsa_decryptor.py" 36L, 991B written
```

hacklabvm_s1_v3 [Running] - Oracle VM VirtualBox

```
student@hacklabvm:~/crypto/q02$ ls
rsa_decryptor.py  rsa.encrypted
student@hacklabvm:~/crypto/q02$ python3 rsa_decryptor.py
Message:  csf2024s1_{voteptarius}
student@hacklabvm:~/crypto/q02$ _
```

Question 3:
(1 point) Find the message hidden inside this encrypted bitmap image Download encrypted bitmap image. You don't need to decrypt.

The same code as workshop 2 was used to copy the header into a file called out.bmp.

```
student@hacklabvm:~/crypto/q03$ dd if=2000x2000_256-color.bmp.encrypted.zip count=54 ibs=1 > out.bmp
54+0 records in
0+1 records out
54 bytes copied, 0.00234083 s, 23.1 kB/s
student@hacklabvm:~/crypto/q03$ dd if=
2000x2000_256-color.bmp.encrypted.zip  out.bmp
student@hacklabvm:~/crypto/q03$ dd if=2000x2000_256-color.bmp.encrypted.zip skip=54 ibs=1 >> out.bmp
21055+0 records in
41+1 records out
21055 bytes (21 kB, 21 KiB) copied, 0.0109022 s, 1.9 MB/s
student@hacklabvm:~/crypto/q03$ _
```

Question 4:
(1 points) Go to /home/student/crypto/q04 and find the file ciphertext.txt, which was created using one of the classical ciphers. Find out what the original text is. What is the key used?

Answer:

A python script was used to perform a frequency analysis on the substitution cipher in order to find the key used to encrypt.

A frequency analysis function was defined and the counter class was used to count the occurrences of uppercase letters in the given text. This returned a string of letters sorted by frequency representing the most to least common letters.

Then another function find_key was defined which accepts the encrypted string and the common English letter list as inputs. It then calls the frequency analysis function to and generates all possible permutations of the sorted letters i.e. The potential keys. It then iterates through each permutation and maps the sorted letters.

The decryption function then takes the encrypted text and decrypts it using the provided key mapping.

```
import re
from collections import Counter

def frequency_analysis(text):
    letter_counts = Counter(re.findall('[A-Z]', text))
    sorted_letters= ''.join([item[0] for item in sorted(letter_counts.items(), key=lambda x: x[1], r
everse=True)])
    return sorted_letters

def find_key(encrypted_text, common_letters):
    encrypted_freq = frequency_analysis(encrypted_text)
    key_mapping = {encrypted: common for encrypted, common in zip(encrypted_freq, common_letters)}
    return key_mapping

def decrypt_with_key(encrypted_text, key):
    decrypted_text = ''.join(key.get(char, char) for char in encrypted_text)
    return decrypted_text

if __name__ == "__main__":
    file_path = "ciphertext.txt"

    with open(file_path, 'r') as file:
        encrypted_text = file.read()

        common_letters = "ETAOINSHRDLCUMWFGYPBVKJXQZ"

        key_mapping = find_key(encrypted_text, common_letters)

        decrypted_text = decrypt_with_key(encrypted_text, key_mapping)

        print("Decrypted Text:")
        print(decrypted_text)

        print("\nKey Mapping:")
        for encrypted, common in zip(frequency_analysis(encrypted_text),common_letters):
            print(f"{encrypted} -> {common}")
                                                                    1,4           Top
```

```
student@hacklabvm:~/crypto/q04$ python3 freq.py
Decrypted Text:
OA SHI ARE TEIA TL AOWEI, OA SHI ARE STDIA TL AOWEI, OA SHI ARE HCE TL SOINTW, OA SHI ARE HCE TL LTT
UOIRNEII, OA SHI ARE ESTMR TL TEUOEL, OA SHI ARE ESTMR TL ONMDENFUOAY, OA SHI ARE IEHITN TL UOCRA, O
A SHI ARE IEHITN TL NHDPNEII, OA SHI ARE ISDONC TL RTSE, OA SHI ARE SONAED TL NEISHOD, SE RHN EGEDYA
RONC TELTDE FI, SE RHN NTARONC TELTDE FI, SE SEDE HUU CTONC NODEMA AT REHGEN, SE SEDE HUU CTONC NODE
MA ARE TARED SHY - ON IRTDA, ARE SEDOTN SHI IT LHD UOPE ARE SDEIENA SEDOTN, ARHA ITWE TL OAI NTOIOEI
A HFARTDOAOEI ONIOIAEN TN OAI TEONC DEMEOGEN, LTD CTTN TD LTD EGOU, ON ARE IFSEDUHAOGE NECDEE TL MTW
SHDOITN TNUY.



Key Mapping:
T -> E
N -> T
X -> A
I -> O
S -> I
R -> N
C -> S
O -> H
U -> R
J -> D
Y -> L
Z -> C
L -> U
E -> M
M -> W
W -> F
V -> G
B -> Y
K -> P
student@hacklabvm:~/crypto/q04$
```

As seen in the screenshot above, this maps some of the letters correctly. To further narrow down the decryption key, another if statement is added to the previous find key function definition. This checks for common English words such as 'the' and 'and'.

```
def find_key(encrypted_text, common_letters):
    encrypted_frequency = frequency_analysis(encrypted_text)
    possible_keys = list(permutations(encrypted_frequency, len(common_letters)))

    for key in possible_keys:
        key_mapping = {encrypted: common for encrypted, common in zip(encrypted_freq, key)}
        decrypted_text = decrypt_with_key(encrypted_text, key_mapping)

        if any(word in decrypted_text for word in ['THE', 'AND', 'TO', 'IS', 'IN']):
            return key_mapping
    return {}
```

The key can then be observed as being

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
P	Y	W	D	C	B	J	Q	I	R	K	L	M	O	A	X	Z	N	S	E	H	V	U	T	F	G

After finding the key, the tr command was used to substitute the key and print out the secret message which appears to be a Charles Dickens 's poem.

```
student@hacklabvm:~/crypto/q04$ cat ciphertext.txt | tr "XUT" "the" |tr "NY" "of" | tr "COS" "was" |
tr "IX" "it" | tr [J,M,Z,L,R,E,W,B] [r,m,g,l,n,c,u,y]
it was the oest of times, it was the worst of times, it was the age of wisnom, it was the age of foo
lishness, it was the ewoch of oelief, it was the ewoch of increnulity, it was the season of light, i
t was the season of narkness, it was the suring of howe, it was the winter of neswair, we han everyt
hing oefore us, we han nothing oefore us, we were all going nirect to heaVen, we were all going nire
ct the other way - in short, the werion was so far like the wresent werion, that some of its noisies
t authorities insisten on its oeing receiVen, for goon or for eVil, in the suwerlatiVe negree of com
warison only.

student@hacklabvm:~/crypto/q04$ _
```

Question 5:
(1 point) Download this shadow file Download shadow file(/etc/shadow) from an old Linux system. Crack the password for the user account called "yoda" (the last entry).

Not attempted.