National University of Computer and Emerging Sciences, Karachi
FAST School of Computing

Network & Cyber Security I (CY 2001)
Fall 2022
Instructor: Dr. Aqsa Aslam
Email: aqsa.aslam@nu.edu.pk

Assignment-II
Date: October 23, 2022

Weightage: 2
Sections: BCY-3A

Assignment Instructions:

1. This is a INDIVIDUAL assignment.
2. The assignment must be submitted in on Google Classroom.
3. Write the roll no: and name on the front page of your assignment.
4. Plagiarism in any form will result in straight "F"

Deadline to submit the assignment no: Oct 28th, 2022

Objective:

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

This lab covers the following topics:

- Secret-key encryption
- Substitution cipher and frequency analysis
- Encryption
- Programming using the crypto library

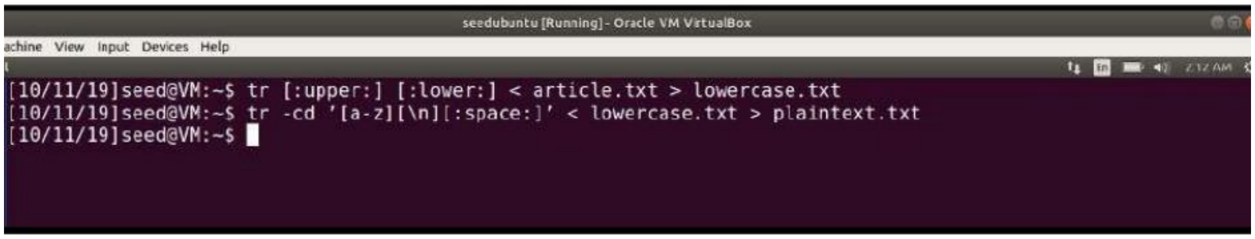
Task 1: Frequency Analysis Against Monoalphabetic Substitution Cipher

It is well-known that monoalphabetic substitution cipher (also known as monoalphabetic cipher) is not secure, because it can be subjected to frequency analysis. In this lab, you are given a cipher-text that is encrypted using a monoalphabetic cipher; namely, each letter in the original text is replaced by another letter, where the replacement does not vary (i.e., a letter is always replaced by the same letter during the encryption). Your job is to find out the original text using frequency analysis. It is known that the original text is an English article.

In the following, we describe how we encrypt the original article, and what simplification we have made. Instructors can use the same method to encrypt an article of their choices, instead of asking students to use the ciphertext made by us.

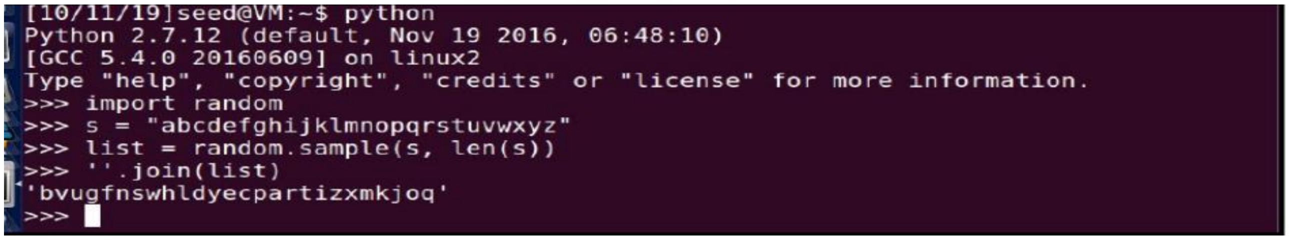
- **Step 1:** let us do some simplification to the original article. We convert all upper cases to lower cases, and then removed all the punctuations and numbers. We do keep the spaces between words, so you can still see the boundaries of the words in the ciphertext. In real encryption using monoalphabetic cipher, spaces will be removed. We keep the spaces to simplify the task. **We did this using the following**

(For simplification article (article.txt) upper cases has been converted into lower cases have been removed then by following commands.)
****tr: Translate, squeeze, and/or delete char**



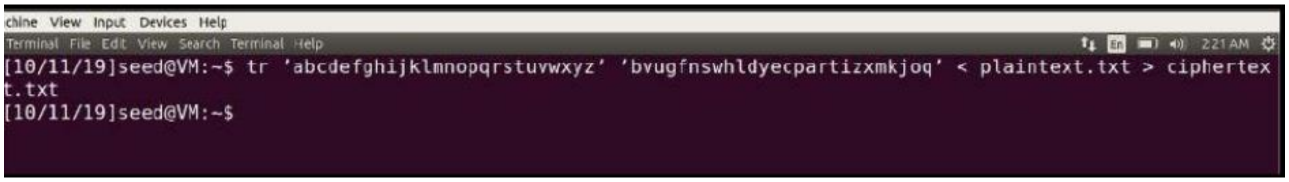
```
[10/11/19]seed@VM:~$ tr [:upper:] [:lower:] < article.txt > lowercase.txt
[10/11/19]seed@VM:~$ tr -cd '[a-z][\n][:space:]' < lowercase.txt > plaintext.txt
[10/11/19]seed@VM:~$ 
```

- **Step 2:** let us generate the encryption key, i.e., the substitution table. We will permute the alphabet from **a to z** using Python, and use the permuted alphabet as the key.
  See the following program.



```
[10/11/19]seed@VM:~$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> list = random.sample(s, len(s))
>>> ''.join(list)
'bvugfnswhldyecpartizxmkjoq'
>>> 
```

- **Step 3:** we use the **tr command** to do the encryption. We only encrypt letters, while leaving the space and return characters alone.



```
[10/11/19]seed@VM:~$ tr 'abcdefghijklmnopqrstuvwxyz' 'bvugfnswhldyecpartizxmkjoq' < plaintext.txt > ciphertex
t.txt
[10/11/19]seed@VM:~$
```

- **Step 4** Using frequency analysis to determine the key.
  Following code to calculate the frequency percentage of the occurrences of the characters in the ciphertext.

```
fp = open("ciphertext.txt","r")
s=fp.read()
fp.close()
count={}
freq={}
sum=0
for i in range(97,123):
    ch=chr(i)
    count[ch]=s.count(ch)
    sum+=count[ch]

for i in range(97,123):
    ch=chr(i)
    freq[ch]=(count[ch]/sum)*100
    print(ch+"\t:\t %.2f"%freq[ch])
```

Using the frequency analysis, you can find out the plaintext for some of the characters quite easily. For those characters, you may want to change them back to its plaintext, as you may be able to get more clues. It is better to use capital letters for plaintext, so for the same letter, we know which is plaintext and which is ciphertext. You can use the tr command to do this.

For example, in the following, we replace letters a, e, and t in in.txt with letters X, G, E, respectively; the results are saved in out.txt.

- **Step 5**

```
$ tr ’aet’ ’XGE’ < in.txt > out.txt
```

Submission:

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed in each step. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation.

Task 2: Encryption using Different Ciphers and Modes

In this task, we will play with various encryption algorithms and modes.
You can use the following openssl enc command to encrypt/decrypt a file. *To see the manuals, you can type man openssl and man enc.*

```
$ openssl enc -ciphertype -e  -in plain.txt -out cipher.bin \
              -K  00112233445566778889aabbccddeeff \
              -iv 0102030405060708
```

- ***Initialization vectors (iv): To randomize the ciphertext of the first block (and thus make each ciphertext unique, even if the plaintext message is repeated), an “Initialization Vector” (IV) is used. The IV is a random number known to both the encrypting and decrypting systems and should only be used once.***

- ***When encrypting multiple blocks of data using a block cipher, there are various encryption modes that may be employed, each having particular advantages and disadvantages. We will look at CBC***

<u>Step:1 DES Algorithm</u>
Command:
openssl enc -des -e -in plaintext.txt -out des_ciphertext.bin -k 34WVF7hI -iv 696e697469616c76

<u>Step:2 AES - Cipher block chaining:</u>
Command:
openssl enc -aes-128-cbc -e -in plaintext.txt -out aes_cbc_ciphertext.bin -k 00112233445566778889aabbccddeeff -iv 0102030405060708

Submission:

Describe what you have done and what you have observed in each step. You also need to provide explanation to the observations that are interesting or surprising. Explain the command use for encryption and decryption. Analysis the output. \

Task 3 (Optional): Those who want to explore more about encryption and decryption

In this task, you are given a plaintext and a ciphertext, and your job is to find the key that is used for the encryption. You do know the following facts:

- The aes-128-cbc cipher is used for the encryption.
- The key used to encrypt this plaintext is an English word shorter than 16 characters; the word can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), pound signs (#:hexadecimal value is 0x23) are appended to the end of the word to form a key of 128 bits.

Your goal is to write a program to find out the encryption key. You can download a English word list from the Internet.

You need to pay attention to the following issues:

- If you choose to store the plaintext message in a file, and feed the file to your program, you need to check whether the file length is 21. If you type the message in a text editor, you need to be aware that some editors may add a special character to the end of the file. The easiest way to store the message in a file is to use the following command (the -n flag tells echo not to add a trailing newline):

```
$ echo -n "This is a top secret." > file
```

- In this task, you are supposed to write your own program to invoke the crypto library. No credit will be given if you simply use the openssl commands to do this task. Sample code can be found from the following URL:
https://www.openssl.org/docs/man1.0.2/crypto/EVP_EncryptInit.html
- When you compile your code using gcc, do not forget to include the -lcrypto flag, because your code needs the crypto library. See the following example:

```
$ gcc -o myenc myenc.c -lcrypto
```

Best of Luck!