ISA 562 – Secure Programming and Systems

Instructor: Dr. Wassim Itani

Mini Project – 1

Topic: Experiments on Symmetric-Key Cryptography Principles.

Collaboration Model: This project can be completed individually or in a team of at most 2 students.

Submission Logistics:

- A **.zip file** : a folder which has all the submission files (refer below) included and later compressed, must be submitted by each member of the team (the same copy of .zip file) to secure an entry in the grade center.

- The .zip file must be named as follows:
  P1_[Your Last Name]_[Your GMUID].zip. For instance, if John Smith with GMUID: G12345678 were to submit this file, John would name it: P1_Smith_G12345678.zip
  If you are working in a team, the contents of the .zip file must be same, but the name of the .zip file must be changed for each student accordingly.

Submission Files (within .zip file):

1. Detailed lab report (.pdf file) must include:
   - Explanation (what you have done/ observed) for all the tasks.
   - Relevant screenshots (the more the better).
   - Contributions made by each group member.
2. A 'readme.txt' file containing names and GIDs of the group members.
3. Optionally you can also include any other files used or generated while executing any of the tasks.

NOTE: Submissions that do not meet the requirements given above or simply providing code/ screenshots without any sufficient explanation, will be penalized.

Lab Setup Instructions:

[WINDOWS USERS AND MAC WITH INTEL CHIPS ONLY]

- This lab has been tested on a pre-configured Ubuntu 20.04 VM, which can be downloaded from the SEED-Ubuntu20.04.zip link. When you unzip the file you will get a .vdi file representing the pre-configured Ubuntu image.
- Install the free VirtualBox software to be able to run the Ubuntu image on your computer. The VM image has been tested on Version 6.1.16. You can check the following VM Manual to stand on the detailed steps to import the VM to VirtualBox.

Start the VM and login into the account with the username **seed**, the password is **dees**.

[MAC USERS WITH APPLE SILICON (M1/M2/M3 MAC) ONLY]

- Students can work on Kali Linux running on UTM, or Virtual Box
- To download UTM, visit this link, and click on download.
- For Kali Linux, visit this link

For a detailed version of how to setup Kali Linux on UTM, refer to this video link.
Note: This project can also be completed on any other Linux flavor of your choice too.
Once the setup is completed, download the required Lab-setup files from here.

THE FOLLOWING PAGES CONTAIN THE STEPS TO BE PERFORMED AS PART OF THIS PROJECT.

Secret-Key Encryption Lab

Overview

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption and some common attacks on encryption. From this lab, students will 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.

Many common mistakes have been made by developers in using the encryption algorithms and modes. These mistakes weaken the strength of the encryption, and eventually lead to vulnerabilities. This lab exposes students to some of these mistakes, and ask students to launch attacks to exploit those vulnerabilities. This lab covers the following topics:

- Secret-key encryption
- Substitution cipher and frequency analysis
- Encryption modes, IV, and paddings
- Common mistakes in using encryption algorithms
- Programming using the crypto library

Readings. Detailed coverage of the secret-key encryption can be found in the following:

- Chapter 21 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.

Task 1: Frequency Analysis

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 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.

```
#!/bin/env python3

import random
s = "abcdefghijklmnopqrstuvwxyz"
list = random.sample(s, len(s))
key = ''.join(list)
print(key)
```

- Step 2: 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 command:

```
$ tr [:upper:] [:lower:] < article.txt > lowercase.txt
$ tr -cd '[a-z][\n][:space:]' < lowercase.txt > plaintext.txt
```

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

```
$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc' \
      < plaintext.txt > ciphertext.txt
```

We have created a ciphertext using a different encryption key (not the one described above). It is included in `Labsetup.zip` file, which can be downloaded from the lab's website. Your job is to use the frequency analysis to figure out the encryption key and the original plaintext.

We have also provided a Python program (`freq.py`) inside the `Labsetup/Files` folder. It reads the `ciphertext.txt` file, and produces the statistics for n-grams, including the single-letter frequencies, bigram frequencies (2-letter sequence), and trigram frequencies (3-letter sequence), etc.

```
$ ./freq.py
-------------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
...
-------------------------------------
2-gram (top 20):
yt: 115
tn: 89
mu: 74
...
-------------------------------------
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
...
```

Guidelines. 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`.

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

There are many online resources that you can use. We list some useful links in the following:

- `https://en.wikipedia.org/wiki/Frequency_analysis`: This Wikipedia page provides frequencies for a typical English plaintext.
- `https://en.wikipedia.org/wiki/Bigram`: Bigram frequency.
- `https://en.wikipedia.org/wiki/Trigram`: Trigram frequency.

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
```

Please replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-bf-cbc`, `-aes-128-cfb`, etc. In this task, you should try at least 3 different ciphers. You can find the meaning of the command-line options and all the supported cipher types by typing "`man enc`". We include some common options for the `openssl enc` command in the following:

```
  -in <file>     input file
  -out <file>    output file
  -e             encrypt
  -d             decrypt
  -K/-iv         key/iv in hex is the next argument
  -[pP]          print the iv/key (then exit if -P)
```

Task 3: Encryption Mode – ECB vs. CBC

The file `pic_original.bmp` is included in the `Labsetup.zip` file, and it is a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, For the `.bmp` file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate `.bmp` file. We will replace the header of the encrypted picture with that of the original picture. We can use the `bless` hex editor tool (already installed on our VM) to directly modify binary files. We can also use the following commands to get the header from `p1.bmp`, the data from `p2.bmp` (from offset 55 to the end of the file), and then combine the header and data together into a new file.

```
$ head -c 54 p1.bmp  > header
$ tail -c +55 p2.bmp > body
$ cat header body > new.bmp
```

2. Display the encrypted picture using a picture viewing program (we have installed an image viewer program called `eog` on our VM). Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

Select a picture of your choice, repeat the experiment above, and report your observations.

Task 4: Error Propagation – Corrupted Cipher Text

To understand the error propagation property of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 1000 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the `bless` hex editor.
4. Decrypt the corrupted ciphertext file using the correct key and IV.

Please answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. Please provide justification.

Expected Results:

- **Task 1:** Provide Screenshots (commands + outputs) of all the given commands/ code, with accompanying detailed explanation where necessary.

- **Task 2**: Provide Screenshots (commands + outputs) of all the given commands/ code, with accompanying detailed explanation where necessary for **at least 3 differentciphers**.

- **Task 3:** Provide Screenshots (commands + outputs) of all the given commands/ code, with accompanying detailed explanation where necessary. **Corruption of bits on bless editor must also be shown as screenshots**. For extra .bmp images visit this link.
  Note: For this task, **only .bmp images** should be used. Other image formats won't generate required outputs.

- **Task 4:** Provide Screenshots (commands + outputs) of all the given commands/ code, with accompanying detailed explanation where necessary. Please answer the following question: **How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively?** Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. Please provide justification.

Acknowledgment

We would like to acknowledge the contribution made by the following people and organizations:

- Jiamin Shen developed the following: the code running inside the container, and the container version of the task on predictable IV.
- The US National Science Foundation provided the funding for the SEED project from 2002 to 2020.
- Syracuse University provided the resources for the SEED project from 2001 onwards.