CSC 302 Computer Security

Secret-Key Encryption Lab

Tyler Prehl

1. 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 fist-hand experience on encryption algorithms, encryption modes, etc. 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 modes, IV, and paddings

Note: The demo lab was done in taz.cs.wcupa.edu server not badgerCTF.cs.wcupa.edu. So, all the screenshots and remote server for uploading files are baz.cs.wcupa.edu. When you do the lab, please connect to badgerCTF.cs.wcupa.edu.

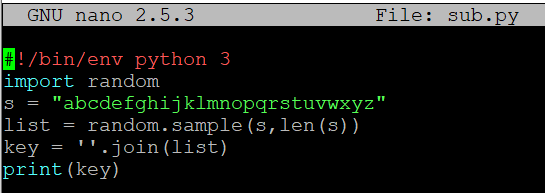
2. Task 1: Frequency Analysis

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.

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. To do it, please first connect to the Linux server (badgerctf.cs.wcupa.edu) through SSH. Then, open a text editor and create a Linux file.

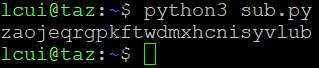
nano sub.py

Then, typing in the following code



After you are done with the code, please save and exit the text editor.

Running python script in Linux is simple. Please use the following commands to see the results. You may have different from mine, since the key is randomly generated each time you run the code.



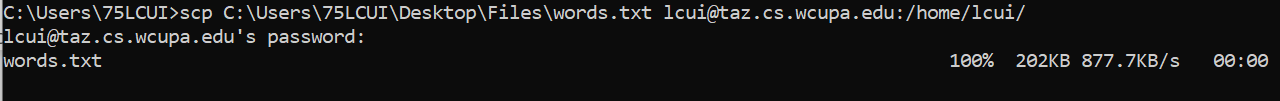
Step 2: let us do some simplification to the original article. We convert all upper cases to lower cases, and then removed all the punctuations and numbers. We do keep the spaces between words, so you can still see the boundaries of the words in the ciphertext. In real encryption using monoalphabetic cipher, spaces will be removed. We keep the spaces to simplify the task.

Step 2.1: upload the file from local machine to Linux server. SCP, Secure Copy Protocol, is a file transfer network protocol used to move files onto servers, and it fully supports encryption and authentication. SCP uses Secure Shell (SSH) mechanisms for data transfer and authentication to ensure the confidentiality of the data in transit.

The syntax of scp commdn is

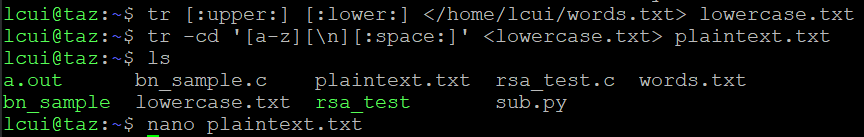
scp *location of file on local machine* *target location on remote server*

Here is an example. After you execute the scp command, it will ask you to enter the password for your account on the Linux server. If you enter the password correctly, it will copy the file to the server.



After it is done, please go to the ssh, and use command ls to check whether the file is under your folder or not.

Step 2.2: modify the file. The first command modify all uppercase to lowercaase. The second command removes punctuations and numbers. If you don’t know the directory of the file, please use the command pwd to see the current directory. After you execute the first two commands, please check whether you have the file “lowercase.txt” and “plaintext.txt” or not use command ls. Then, open the plaintext.txt using nano.



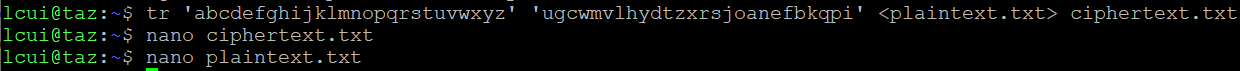
Please attach the screenshot of the first 10 lines of plaintext.txt.

Text

Description automatically generated

Step 3: we use the tr command to do the encryption. We only encrypt letters, while leaving the space and return characters alone. The first command encrypt ‘abcdefghijklmnopqrstuvwxyz’ by ‘ugcwmvlhydtzxrsjoanefbkqpi’, which was randomly generate by sub.py. The filename inside <> is the source file that you want to encrypt, which is plaintext.txt in this case. The ciphertext message is written to the filename after <>, which is ciphertext.txt in this case.

Then, you could use nano command to open both plaintext.txt and ciphertext.txt and see whether the letters are encrypted as you directed.



Step 4: Frequency analysis. 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 letter 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 ciphertext.txt with letter X, G, E, respectively; the results are saved in out.txt.



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

* <http://www.richkni.co.uk/php/crypta/freq.php>: This website can produce the statistics from a ciphertext, including the single-letter frequencies, bigram frequencies (2-letter sequence), and trigram frequencies (3-letter sequence), etc.
* <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.

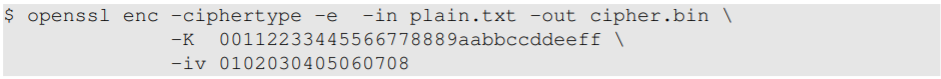
Please analyze the ciphertext message and attach a screenshot of the analysis.

Text

Description automatically generated

4. Task 2: Encryption using Different Ciphers and Modes

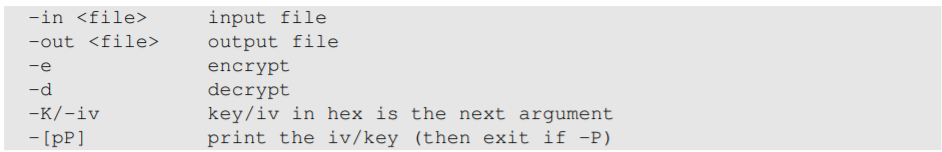
In this task, we will play with various encryption algorithms and modes. You can use the openssl enc command to encrypt/decrypt a file. To see the manuals, please type man openssl and man enc.



Please replace the *ciphertype* with a specific cipher type, such as -aes-128-cbc, -bf-cbc, -aes-128-cfb, etc. Please also replace the input file *plain.txt* to your input file name. If you use the same filename as I did in previous steps, the input file name is plaintext.txt. The output file is cipher.bin. The extension bin means it is a binary file. You could use following command to view the binary file in Linux.

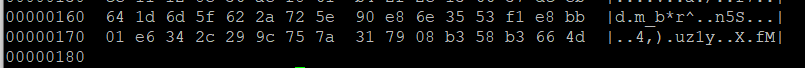
hexdump -C yourfile.bin

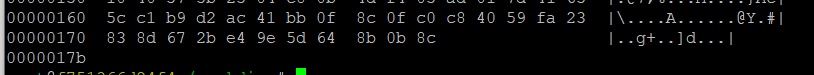
You can find the meaning of the command-line options and all the supported cipher types by typing “man enc”. We include some common options for the openssl enc command in the following:



Please encrypt your plaintext file using at least 3 different ciphers and take a screenshot of the last two lines of each ciphertext file you got.







5. Task 3: Encryption Mode – ECB vs. CBC

The file pic\_original.bmp is a picture file included in this lab. 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:

Step 5: download the picture “pic\_original.bmp” from D2L to your local machine.

Step 6: upload pic\_original.bmp from your local machine to remote server “badgerCTF.cs.wcupa.edu” using scp in the command prompt. Please find the scp syntax above.

Text

Description automatically generated

Then, check the file in the remote server badgerCTF.cs.wcupa.edu.

Text

Description automatically generated

Step 7: Please encrypt the picture using ECB mode and CBC mode

Text

Description automatically generated

Step 8: 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 must 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 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

Please create two viewable pictures, one for encryption in CBC mode and one for encryption in ECB mode using above syntax.

Step 9: download the two encrypted version from remote server badgerCTF.cs.wcupa.edu to your local machine using scp in the command prompt. You just need to change the source (first argument) to the directory of the file on the remote server, and the destination (second argument) to local machine directory. (You may need to first disconnect from the remote server and then download the files)

Please attach the screenshots of two encrypted images here and explain your observations.

cbc

A picture containing people, fabric

Description automatically generated

ecb

Background pattern

Description automatically generated

While the ecb sort of maintains the image with a lot of rgp noise/overlay, the cbc encryption COMPLETELY deranged the image into an undecipherable image.

6. Task 4: Padding

For block ciphers, when the size of a plaintext is not a multiple of the block size, padding may be required. The PKCS#5 padding scheme is widely used by many block ciphers. We will conduct the following experiments to understand how this type of padding works.

Step 10: Let us create three files, which contain 5 bytes, 10 bytes, and 16 bytes, respectively. We can use the statement “echo -n” command to create such files. For example:

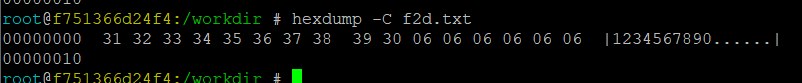
echo -n “12345” > f1.txt

After executing this command, f1.txt is a file with length 5 bytes (without the -n option, the length will be 6, because a newline character will be added by echo). We then use command “openssl enc -aes-128-cbc -e -in f1.txt -out f1e.txt” to encrypt this file. The file size can be displayed by command “ls -l” for the entire directory, or “ls -l filenameWithExtension” for a specific file.

Please create three files with length of 5 bytes, 10 bytes, and 16 bytes. Encrypt each file and described the size of the encrypted files.

Step 11: We would like to see what is added to the padding during the encryption. To achieve this goal, we will decrypt these files using command as “openssl enc -aes-128-cbc -d -in f1e.txt -out f1d.txt -nopad”. The option “nopad” means do not automatically remove the padding when decrypt the ciphertext message. If should be noted that padding data may not be printable, so you need to use a hex tool to display the content as “hexdump -C f1d.txt”.

Please show the padding information for file with length of 10 bytes and 16 bytes.



Text

Description automatically generated

7. Task 5: Initial Vector (IV)

Most of the encryption modes require an initial vector (IV). Properties of an IV depend on the cryptographic scheme used. If we are not careful in selecting IVs, the data encrypted by us may not be secure at all, even though we are using a secure encryption algorithm and mode.

Step 12: A basic requirement for IV is uniqueness, which means that no IV may be reused under the same key. To understand why, please encrypt the same plaintext (such as f1.txt) using

(1) two different IVs, and

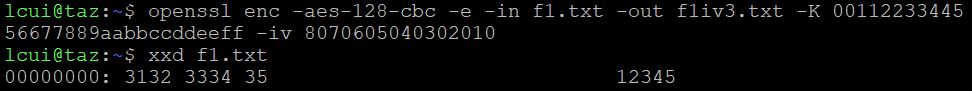
(2) the same IV.

The syntax to encrypt a file with a specific key and iv is as below:

Openssl enc -aes-128-cbc -e -in *inputfilename* -out *outputfilename* -K *encryptionKey* -iv *initialvectorvalue*

Then, the hex value of a file can be viewed by using command xxd as “xxd *filename*”.

Here is an example:



Please attach the screenshot of three encrypted files. Two use the same IV and one uses different IV. Then, describe your observation, based on which, explain why IV needs to be unique.