Secret-Key Encryption Lab

Telle Miller

Liberty University – CSIS 463

10/11/2021

**Introduction**

This lab was intended to familiarize me with the basic concepts of secret-key encryption, as well as some of the common attack vectors. In this lab, I gained first-hand experience with encryption algorithms, encryptions modes, paddings, and initial vectoring (IV). I was also able to use tools and write programs to encrypt/decrypt messages. Much of this lab utilized the OpenSSL API to perform the encryption and decryption.

**Setup**

This lab is completed using the resources provided by SEED labs. I completed it using their pre-built Ubuntu 20.04VM and the files outlined in the lab PDF. This lab also utilized containers, which was trivial thanks to the setup done by the pre-built VM.

**Tasks**

Task 1: Frequency Analysis

The goal of this task was to use frequency analysis on a monoalphabetic substitution cipher. This type of cipher is vulnerable to frequency analysis and is therefore not used in any large capacity. The lab environment provides the tools to generate and decrypt ciphertext, however the link provided to perform the frequency analysis was no longer available. I attempted to use other resources I found on the internet to perform the frequency analysis, but ultimately was unable to figure it out. I understand the principles in this task, but due to the Web Server 404 error in the link provided, was unable to carry it out fully.

Graphical user interface, text, application, email

Description automatically generated

Task 2: Encryption using Different Ciphers and Modes

The goal of this task was to utilize the libraries available in Ubuntu to perform a variety of encryptions. Using the *openssl* command, I was able to encrypt using three different cryptosystems very easily. This command made it very easy to perform this task. The three commands used to perform the encryption are displayed below.







Task 3: Encryption Mode – ECB vs. CBC

This task’s goal was to encrypt an image using the Electronic Code Book (ECB) and Cipher Block Chaining (CBC). Using these encryption algorithms, the goal is to encrypt an image so that those without the encryption key cannot decipher was the image contains. To maintain the legitimacy of the \*.bmp file type, the header information must be preserved. I used the *dd* command to remove the header information from the *pic\_original.bmp* file. This process is show below.

Text

Description automatically generated

I was then able to encrypt the image twice, first using the OpenSSL API and the CBC algorithm. For the second encryption, I used the OpenSSL API and the ECB algorithm.





Now that I had the image encrypted using both methods, I added the header information I saved earlier back to the images. Once again, I leveraged the *dd* command to accomplish this, which is shown below.





Now that both files had header information, the OS recognized them as \*.bmp files and I was able to open them. The first image displayed below is the result of the CBC algorithm. This encryption was very effective, as I cannot tell at all what the original image was. The image beneath that is the result of the ECB algorithm. While it succeeded in hiding the color of some of the objects, the overall shapes of the objects can be easily made out. The ECB algorithm did a much poorer job of fully encrypting the image.

A picture containing text

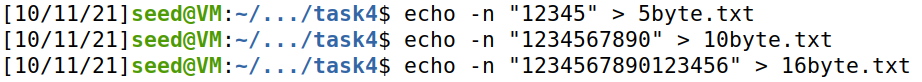
Description automatically generated

Background pattern

Description automatically generated

Task 4: Padding

The goal of this task was to view the level of padding added by each block cipher algorithm. Using block ciphers, when the size of a plaintext is not a multiple of the block size, the algorithm will add padding to the plaintext to fix it. To complete this task, I first created three files using the *echo* command. The file sizes are 5 bytes, 10 bytes, and 16 bytes.



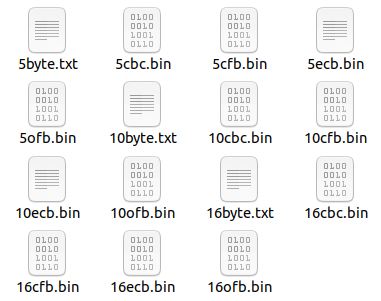
After I had created my files, we needed to encrypt them to add the padding. The images below show the process of encrypting each of the files with the CBC algorithm.







I repeated the steps shown above with three other algorithms: ECB, CFB, and OFB. The image below shows the ultimate result of all this encryption.



Now that all my files were encrypted, it was time to decrypt. By decrypting them, I could view the padding that is left over and determine the block size of the various encryption algorithms used. The image below shows the command to decrypt a given file.



I then used *hexdump* and *xxd* to view the hexadecimal version of each file. An example of this process is show below.

Text

Description automatically generated

Using this hexadecimal data, it is easy to see where the algorithm added padding before encryption. I repeated the step above with every file I had previously encrypted.

Task 5: Error Propagation – Corrupted Cipher Text

The goal of this task was to purposely corrupt encrypted text to see what the outcome of the decryption process would become. Initially, I created a large file of 1001 bytes using the python command show below.



I then encrypted the file using AES-128 and opened the output file using the *bless* command. Inside the hex editor application, I changed the 55th byte and saved the file. These steps are shown below.





Graphical user interface, application, table

Description automatically generated

I then attempted to decrypt the corrupted file using AES-128, but it returned a message of “bad decrypt”, as shown below. Using AES-128-CBC, a corrupted hex value made the contents of the file unsalvageable.

Text

Description automatically generated

Task 6: Initial Vector (IV) and Common Mistakes

The goal of this task was to demonstrate the issues that can be found when using inappropriate initial vectors (IVs). The basic requirement for an IV is uniqueness, meaning that no IV can be reused under the same key. This task had several subtasks, ranging from using the same IV twice to using predictable IVs. The conclusion of this task showed that a lot of information can be gleaned from encrypted text if an improper initial vector is used. I was unable to complete this task, as it required the use of Docker containers, which was an unfamiliar subject. In the end, I was happy to have read about IVs and learned about the importance of ensuring they are properly chosen.

**Conclusion**

Overall, this lab was challenging and gave great insight into the OpenSSL API and the most common attack vectors for cryptosystems. As with previous labs, the initial few tasks were relatively simple and not very time consuming, but by the end of the lab the tasks were difficult. The tasks covered a wide variety of topics including frequency analysis, various encryption algorithms available in OpenSSL, padding, error propagation, initial vectors, and ECB vs. CBC. My favorite section of this lab was the ECB vs. CBC, as I learned a lot about how variable algorithms in the same family, AES, can be. It was also interesting to work with images, as the ability to view a visual representation of encryption was enlightening. The issue of error propagation was also a key task for me. It is interesting that an algorithm can detect and exit when it notices even the most subtle change in the hexadecimal version of a file.

**References**

Delfs, Hans, and Helmut Knebl. *Introduction to Cryptography: Principles and Applications.* Springer, 2007.

*"In Our Time: Cryptography".* BBC Radio 4. Retrieved 11 October 2021.

Surender R. Chiluka. "*Public key Cryptography*". Retrieved from <https://www.cs.uri.edu/cryptography/publickeykidkrypto.htm>.

Bellovin, S. M. (2009, February 3). *Network Security: Modes of Operation*. Columbia University: Department of Computer Science. Retrieved October 11, 2021, from <https://www.cs.columbia.edu/~smb/classes/s09/l05.pdf>.

Margaret Cozzens and Steven J. Miller. "*The Mathematics of Encryption: An Elementary Introduction*". p. 180.