|  |
| --- |
|  |
| Capstone Experience  IST 894 |
| Lab 3 – Encryption Primitives and Block Cipher Modes of Operation |

Table of Contents

[1. General Context 3](#_Toc83329378)

[2. Technical Context 4](#_Toc83329379)

[3. Solution 6](#_Toc83329380)

[References 11](#_Toc83329381)

# General Context

In this lab we do some hands-on exercises with a few different types of block cipher modes of operation. A block cipher is an encryption method that applies a deterministic algorithm along with a symmetric key to encrypt a block of text, rather than encrypting one bit at a time as in stream ciphers (What Is a Block Cipher?, 2014). A block cipher mode of operation is an algorithm that utilizes a symmetric key block cipher to provide a service such an encryption (Computer Security Division, 2017). There are eight confidentiality modes of operation that are approved by NIST, and in this lab we look at four of them.

We will encrypt, decrypt, and compare the results of each action for plain text and image files. We start out with the simplest of the encryption modes, the electronic codebook (ECB). With ECB there isn’t an initialization vector, nor any additional variables add to the block before encryption which means that any blocks that contain identical plain text will all produce identical ciphertext. This may not seem like a big deal, but depending on what you are trying to encrypt patterns might present themselves which could potentially give clues as to what is encrypted. The other modes that we look at all have additional randomization integrated in to them to make the ciphertext unique no matter how many times you encrypt an identical block of data.

Finally, we will look at how resilient the various modes of operation are. Some times a byte of data is lost in transit, and the different modes have much different levels of dependency on the previous blocks being whole. We will show that, for ECB, if you change even a single byte of data the file basically becomes corrupted and cannot be decrypted, while the other modes have varying levels of success. Comparing how these actually function could help in a decision making process of when to use one mode of operation over another.

# Technical Context

The idea of using block cipher techniques is not necessarily new. Data Encryption Standard (DES), which we saw used in one of the password hashes in lab 2, was publicly released in 1976. In 2001 NIST chose the Rinjdael cipher to become the new Advanced Encryption Standard (AES) which could use key sizes of 128, 192, or 256 bits where DES used a fixed length key of 56 bits. In various products advertisements you might see a company claim that they have ‘military grade encryption’, and while some claim that this means AES-128 and above, others like Dashlane say it has to be at least AES-256. The truth of it all is that it’s largely a marketing term that has no precise meaning (Hoffman, 2020). We use and compare four different modes of operation of AES-128 and it ends up being pretty obvious that the mode of operation is just as important, if not more so, than the block cipher itself. NIST currently only has two approved block cipher algorithms, AES and Triple DES. There are fourteen approved modes of operations, but realistically only five of them are purely used for confidentiality.

We will look at Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB) modes. While all will use AES-128 in this lab, they will have drastically different results. ECB uses a process where the plaintext is broken up into blocks, and then a common encryption key is used across all of the blocks, and each block is independently encrypted and can also be decrypted independent of any of the other blocks (What Is Electronic Code Book (ECB) Encryption and How Does It Work?, n.d.). This means that identical blocks of plain text using the same encryption key will produce identical blocks of cipher text. If this was used to encrypt something with a lot of perceived patterns like an image file, the encrypted file would show what would appear to be a negative image but where the overall image would still be identifiable.

The other modes all have processes built into them to add randomization into each block so that the output of every block is unique. CBC, CFB, and OFB all use a similar process where the initial block is provided with an initialization vector to randomize the first block, but then each subsequent uses a process to combine all or part of that block with the next during encryption. This means that it acts like a true chain, meaning the blocks can not be decrypted individually or independently, you have to start at the one end and work your way to the other because each block builds upon the next.

Finally, to compare CBC, CFB, and OFB in a little more detail. CBC takes the plaintext, add in an initialization vector to use to help randomize the cipher text of the first block, and then from there it takes that cipher text and XORs the next block with it before it’s encrypted, and continues this pattern. This ensures that each message is unique. CFB works similarly to CBC, and any change that is made in a block will propagate for the entire length of the chain since the output from what changed is used to create each output after it. For both CBC and CFB encryption cannot take place in parallel, but decryption can. OFB works a little differently, even though it’s a block cipher, it acts like a stream cipher. OFB generates keystream blocks which are XORed with the plain text blocks to generate cipher text. For OFB, the encryption and decryption process are identical, and because of how it’s implemented if a portion gets lost or damaged, only that section becomes unreadable, the issue isn’t pushed down the entire length of the chain so OFB is the best bet if you’re in an environment where you are likely to lose packets.

# Solution

**To begin this lab, we log into our ‘Ubuntu with Snort and Other Tools’ VM in the cyber range. The first step we do is open a terminal window and install ‘Bless’ which is a hex editor for files. We then check out that ‘openssl’ is functioning. Figure 1 shows the output from ‘openssl’ which lists the various flags that can be passed and a large list of cipher types and modes of operation that are supported.**

Text

Description automatically generated

Figure 1

We then use the following command `openssl enc –aes-128-cbc –e –in test\_message.txt –out test\_aes\_128\_cbc.bin \-k 00112233445566778899aabbccddeeff \-iv 0102030405060708` to encrypt the ‘test\_message.txt’ file. To break down that command ‘enc’ specifies the command line tool for encrypting and decrypting, ‘-aes-128-cbc’ means to use AES with 128 bit keys and CBC as the mode of operation, ‘-e’ means encrypt the file, ‘-in’ specifies the file to encrypt, ‘-out’ specifies where to save the cipher text, ‘-k’ is the passphase used to encrypt, and ‘-iv’ is the initialization vector. Figure 2 shows what happens if we try to read the encrypted file. Finally, we decrypt the file with `openssl enc –aes-128-cbc –d –in test\_aes\_128\_cbc.bin –out test\_message2.txt \-k 00112233445566778899aabbccddeeff \-iv 0102030405060708` the changes in this command is we replace ‘-e’ with ‘-d’ because we are decrypting the file, and we make the ‘-in’ the encrypted file and ‘-out’ becomes where we want to save the resulting plain text. Figure 3 shows the decryption command followed by a print out of the resulting file. Figure 4 then shows that a diff between the original file and the decrypted file are the same.

Text

Description automatically generated

Figure 2

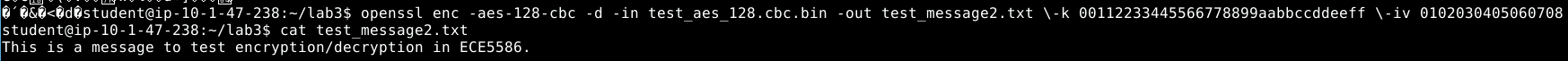


Figure 3

****

Figure 4

**Next, we use a similar process to encrypt a bitmap image file, but we’ll be using AES-128-ECB instead. Once the image file is encrypted, the file header is changed, so we have to use ‘bless’ to open the file and copy the first 54 bytes from the original image to the encrypted file as shown in Figure 5. Then Figure 6 shows the original image file, and Figure 7 shows why ECB isn’t a great solution for encrypting patterns, because identical plain text blocks output identical cipher text blocks, so an image basically looks like a negative of the original.**

Graphical user interface, text, application, email

Description automatically generated

Figure 5

Shape

Description automatically generated

Figure 6

Background pattern

Description automatically generated with medium confidence

Figure 7

**You can compare the output from the ECB encrypted Figure 7 with Figure 8 which is encrypted with AES-128-CBC and shows a much more random output. Both ECB and CBC are block ciphers, but ECB doesn’t randomize the blocks at all, so similar inputs have similar outputs where CBC uses an initialization vector to randomize the first block and then every subsequent block uses the previous output to ensure randomness instead of just inverting the image. So, between ECB and CBC, if you’re looking to actually protect your data, you’re going to use CBC.**

A close up of a rug

Description automatically generated with low confidence

Figure 8

**Finally, the last part of this lab looks at corrupted packets in a file that was transmitted over a noisy network. We encrypt the ‘plain.txt’ file with AES-128 ECB, CBC, CFB, and OFB and then use ‘bless’ to change the 40th byte in the file. This simulates what would happen if a packet got corrupted, and then we decrypted the manually edited files of each mode of operation.** Figure 9 shows the error that is output when ECB tries to decrypt a file with a corrupt packet, Figure 10 is the actual unreadable output of the ECB process. Figure 11 shows the outputs from CBC, CFB, and OFB in that order. As stated in the technical context section, CFB works similarly to CBC, and any change that is made in a block will propagate for the entire length of the chain since the output from what changed is used to create each output after it. So the single packet has a much larger effect. OFB, which ends up acting like a stream cipher, is only affected at the single packet that was corrupted. So if you’re going to be sending files over a noisy network, OFB is your best bet.

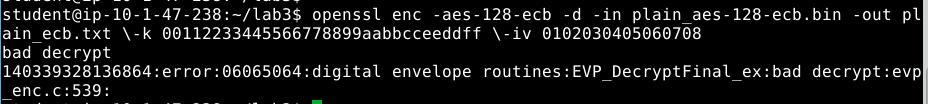


Figure 9

Graphical user interface, text

Description automatically generated

Figure 10

Text

Description automatically generated

Figure 11

# ****References****

Computer Security Division, I. T. L. (2017, January 4). *Block Cipher Modes—Block Cipher Techniques | CSRC | CSRC*. CSRC | NIST. <https://csrc.nist.gov/projects/block-cipher-techniques/bcm>

Hoffman, C. (2020, May 29). *What Is “Military-Grade Encryption”?* How-To Geek. <https://www.howtogeek.com/445096/what-does-military-grade-encryption-mean/>

*What is a Block Cipher? - WolfSSL*. (2014, December 19). <https://www.wolfssl.com/what-is-a-block-cipher/>

*What is Electronic Code Book (ECB) encryption and how does it work?* (n.d.). SearchSecurity. Retrieved September 22, 2021, from <https://searchsecurity.techtarget.com/definition/Electronic-Code-Book>