Exploring Symmetric Key Encryption Modes

Getting Started

Boot your Linux system or VM, log in, and then open a terminal window and start the lab:

```
cd labtainer/labtainer-student
start.py symkeylab
```

It may help to stretch the resulting bash terminal window to the right to provide for more output space.

Note the terminal displays the paths to two files on your Linux host:

1) This lab manual
2) The lab report template

On most Linux systems, these are links that you can right click on and select "Open Link". **If you chose to edit the lab report on a different system, you are responsible for copying the completed files back to the displayed path on your Linux system before using "./stop.py" to stop the lab for the last time.**

Familiarize yourself with questions in the lab report template before you start.

In this lab, you will explore the concept of encryption modes and their properties. To perform this exploration, you will be using an open source encryption product known as OpenSSL. You will explore the properties of these modes, seeing a visual representation of the state of the ciphertext, and exploring error propagation during decryption in these various modes.

Note: You can use the commands "`man openssl`" and "`man enc`" to get additional help on how to use the `openssl` command line tool. Appendix A contains a quick reference sheet of commonly used Unix commands and appendix B explains hexadecimal.

Use the "ll" command to list the content of the directory.

Task 1: Warm-up: Encrypt and then decrypt a file (any file)

In this task you are only getting used to the syntax of the `openssl` command.

Do the following:

1. Create a small text file “plain.txt”.
2. To encrypt the text file as “cipher.txt” type:

```
openssl CIPHER -e -in plain.txt -out cipher.txt -K KEY -iv IV
```

replacing:

- *CIPHER* with a specific cipher and CBC mode of operation, *e.g.* **aes-128-cbc**. (To see all the options use “man enc”).
- *plain.txt* is the name of the plain text file you just created
- *cipher.txt* is the name of the output file which will be the ciphertext
- *KEY* with a hexadecimal representation of a symmetric key (your choice)
- *IV* with a hexadecimal representation of an initialization vector (your choice)

Be sure to use the -K option, and not the -k option. The latter may not produce an error, but may cause issues later in the lab.

3. Observe the ciphertext you’ve created using `cat`, `more`, `less`
4. You should have observed that the encrypted file is gibberish that will not always display well. To see the actual hex values of the ciphertext, enter the following:
   `hexdump –C FILENAME`
5. List the contents of the directory using the *long* option (i.e., enter “`ls –l`”)to see the sizes for your original file and the encrypted file.
   Record the size of the plaintext file in item 1 of the report.

   Record the size of the corresponding ciphertext file in item 2 of the report.

   Note that item 3 asks a follow-up question. You can answer the question now or later (which is true for all the follow-up questions).

6. You can decrypt the ciphertext you’ve created using the following command. **Be sure to output to a new plaintext file**, and not overwrite the original plaintext.

```
openssl enc CIPHER -d -in cipher.txt -out plainmod.txt -K KEY -iv IV
```

7. Compare the decrypted plaintext with the original plaintext using the `diff` command, as shown below (replacing ORIGINAL and UNENCRYPTED with the file names you used):

   ```
   diff –a plain.txt plainmod.txt
   ```

   [Note: if the two files are different then you did something wrong. If `diff` returns nothing, then there were no differences, which is what you would expect.]

Task 2: Encryption Modes

In this task, you will explore the differences in security attained by several modes of encryption. You will use a web browser on your host Linux system (to view files that you create and modify on the "symkeylab" computer. You will have noticed that the symkeylab computer contains a "index.html"and "nps-logo.bmp" file. To see the page, start the firefox browser using this command:

./start_firefox

You will see the NPS logo, the (nps-logo.bmp file), and a broken link to a modified version of the logo, (a nps-logo_mod.bmp file that you will be creating).

1. ECB Mode

   a. Observe the NPS logo.

      When looking at the NPS logo with the eye of a cryptanalyst, what patterns do you see? Record your observations in item #4 of the report.

   b. Encrypt `nps-logo.bmp` using AES in **ECB** mode (with the option **aes-128-ecb**) to create a ciphertext (Name your ciphertext as "`nps-logo_mod.bmp`". [Because ECB mode does not require an IV, you do not need to provide one.]

   c. List the contents of the directory using the *long* option to see the sizes for the plaintext logo file and the encrypted file.

      Record the size of the plaintext logo file in item 5 of the report.
      Record the size of the encrypted logo file in item 6 of the report.

Note: When you refresh the page in the web browser, it fails to display the modified file because it does not recognize it as a valid image.

d. You can visualize the ciphertext by tricking an browser into thinking that the encrypted file is still a valid image file. To do this, you need to do a little "preprocessing" to make the file viewable.
BMP images have a 54-byte header that informs the image viewer about the image, such as the image size, and its dimensions. You need to replace the encrypted BMP header in the ciphertext you created with a valid BMP header from `nps-logo.bmp`.

To replace the 54-byte header with one command, do the following

```
dd if=nps-logo.bmp of=nps-logo_mod.bmp bs=1 count=54 conv=notrunc
```

e. After modifying the header, go back to the web browser and refresh the web page to view the encrypted image.

Using item 7 of the report, describe the visualization of the encrypted logo.

Note that item 8 of the report asks a follow-up question.

2. `CBC Mode`
   a. `Encrypt nps-logo.bmp again, but this time use` **`CBC`** `mode to create a ciphertext (with the` **`aes-128-cbc`** `option). [This time you need to provide an IV.]`
   b. List the contents of the directory using the *long* option to see the size of the ciphertext file you just created.

Record the size of the CBC-encrypted logo file in item 9 of the report.

   c. Using the `dd` command described above, modify the header of the new ciphertext to have the same 54 bytes as the original BMP file.
   d. View the encrypted ciphertext using the web browser refresh.

Using item 10 of the report, describe the visualization of the encrypted logo.

3. CFB Mode
    a. Encrypt the `nps-logo.bmp` file again, but this time use the **CFB** mode (with the **aes-128-cfb** option). [You need to provide an IV.]
    b. List the contents of the directory using the *long* option to see the size of the ciphertext file you just created.

       Record the size of the CFB-encrypted logo file in item 11 of the report.

    c. Using the `dd` command described above, modify the header of the new ciphertext to have the same 54 bytes as the original BMP file.
    d. View the encrypted ciphertext using the web browser refresh.

       Using item 12 of the report, describe the visualization of the encrypted logo.

4. OFB Mode
    a. Encrypt the nps-logo.bmp file again, but this time use the **OFB** mode (with the **aes-128-ofb** option). [You need to provide an IV.]
    b. List the contents of the directory using the *long* option to see the size of the ciphertext file you just created.

       Record the size of the OFB-encrypted logo file in item 13 of the report.

    c. Using the `dd` command described above, modify the header of the new ciphertext to have the same 54 bytes as the original BMP file.
    d. View the encrypted ciphertext using the web browser refresh.

       Using item 14 of the report, describe the visualization of the encrypted logo.

       Note that items 15, 16 and 17 of the report ask follow-up questions.

Task 3: Error Propagation During Decryption

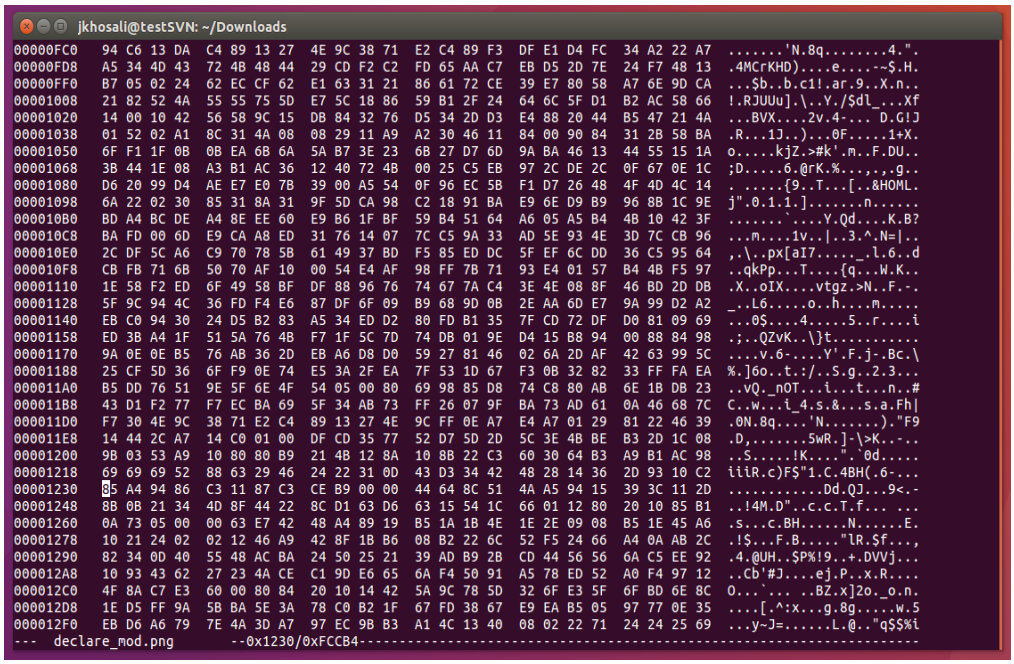
This task will help you develop an understanding of the ability of various cipher modes of operation to recover from corruption. You will encrypt a text file in two different modes, change a single bit in the middle of the encrypted file, decrypt the corrupted file, and then view the effects of the corrupted bit on the plaintext file.

1. Introduction
    a. List the contents of the directory using the *long* option so you can see the size of the `declare.txt` file (in bytes).
    Using item 18 of the report, record the size of `declare.txt`.

    Calculate the number of AES blocks it will take to encrypt `declare.txt`. Write your answer in item 19 of the report. Show your work.

    If each character in a text file is represented in ASCII format, where each character is represented in one byte, how many characters does it take to fill up an AES block? Write your answer in item 20 of the report.

2. ECB Mode
    a. Encrypt `declare.txt` using AES-128 in **ECB** mode (with the **aes-128-ecb** option).
    b. Open the encrypted file using hexedit, as shown below, replacing ENCRYPTEDFILE with the name you chose:
    `hexedit ENCRYPTEDFILE`
    c. Click <Enter> to select a byte to go to. When prompted with “New position? 0x”, enter `1230`, to take you to near the middle of the encrypted file. You should end up with “00001230” as an address on the left-hand, as shown in the next figure.



d. There will be a group of two hex digits to the right of “00001230”. Change the right-most digit of this pair such that only one bit is modified. **Refer to the table in Appendix C** to determine whether to change the hex digit up or down one to ensure that only one bit is modified in the ciphertext.

   Enter <Ctrl-X> to **Save the change** when you are done and then exit the hex editor.

e. Decrypt the ciphertext **without** overwriting the original file.

f. Use the `diff` command (with the “`-a`” option, as shown earlier) to show where the original file is different from the decrypted file. **Using item 21 of the report, describe the corruption of the decrypted file.**

   Note that item 22 is a follow-up question.

3. CBC Mode

    a. Encrypt declare.txt again using AES-128 in **CBC** mode (with option **aes-128-cbc**).

    b. Use hexedit again to modify the ciphertext in the same location such that only one bit is modified.

    c. Decrypt the ciphertext **without** overwriting the original file.
    d. Use the `diff` command (with the "`-a`" option, as shown earlier) to show where the original file is different from the decrypted file. **Using item 23 of the report, describe the corruption of the decrypted file.**

    Note that item 24 is a follow-up question.

Submission

After finishing the lab, go to the terminal on your Linux system that was used to start the lab and type:

```
stop.py symkeylab
```

If you modified the lab report on a different system, you must copy those completed files into the directory paths displayed when you started the lab, and you must do that before typing "./stop.py". When you stop the lab, the system will display a path to the zipped lab results on your Linux system. Provide that file to your instructor, e.g., via the Sakai site.

Appendix – Some Unix Commands

`cd` Change the current directory.

```
cd destination
```

With no “destination” your current directory will be changed to your home directory. If you “destination” is “..”, then your current directory will be changed to the parent of your current directory.

`cp` Copy a file.

```
cp source destination
```

This will copy the file with the “source” name to a copy with the “destination” name. The “destination” can also include the path to another directory.

`clear` Erase all the output on the current terminal and place the shell prompt at the top of the terminal.

`less` Display a page of a text file at a time in the terminal. (Also see `more`).

```
less file
```

To see another page press the space bar. To see one more line press the Enter key. To quit at any time press ‘q’ to quit.

`ls` List the contents and/or attributes of a directory or file

```
ls  location
ls  file
```

With no “location” or “file” it will display the contents of the current working directory.

`man` Manual

```
man command
```

Displays the manual page for the given “command”. To see another page press the space bar. To see one more line press the Enter key. To quit before reaching the end of the file enter ‘q’.

`more` Display a page of a text file at a time in the terminal. (Also see `less`).

```
more file
```

To see another page press the space bar. To see one more line press the Enter key. To quit at any time press ‘q’ to quit.

`mv` Move and/or Rename a file/directory

```
mv source destination
```

The “source” file will be moved and/or renamed to the given

“destination.”

`pwd` Display the present working directory

```
pwd
```

Appendix B – Hex Explained

Hex is short for hexadecimal. Numbers can be represented in many ways. We typically count in base-10, or decimal. Binary is counting in base-2. Hexadecimal is counting in base-16.

A standard way to describe these notations is to explain that each 'position' counts powers in these bases. Let $\mathbf{X_b}$ be a number whose value (**X**) should be interpreted as being written in base-**b**. The decimal number 13 might be written as 13_{10}, for clarity.

Counting by powers:

$1534_{10} = 4 \times 100 + 3 \times 101 + 5 \times 102 + 1 \times 103 = 4 + 30 + 500 + 1000$
$1011_{2} = 1 \times 20 + 1 \times 21 + 0 \times 22 + 1 \times 23 = 1 + 2 + 8 = 11_{10}$

The symbols we use to count by powers are all values less than base:

- Base 2 uses two symbols (0,1) to count.
- Base 10 uses ten symbols (0,1,2,3,4,5,6,7,8,9) to count.
- Base 16 uses symbols sixteen symbols: 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f.

$1534_{16} = 4 \times 160 + 3 \times 161 + 5 \times 162 + 1 \times 163 = 5428_{10}$
$1011_{16} = 1 \times 160 + 1 \times 161 + 0 \times 162 + 1 \times 163 = 4113_{10}$
$beef_{16} = 15 \times 160 + 14 \times 161 + 14 \times 162 + 11 \times 163 = 48879_{10}$

Sometimes we write hex digits using a leading '0x' to signal that the number is in hex, since its not always clear: 0x1534, 0x1011, 0xbeef, etc.

A single hex digit can represent any four-digit binary value, or 4 bits:

0000_2 = 0x0, 0001_2 = 0x1, ..., 1110_2 = 0xe, 1111_2 = 0xf

As an aside, just as 8 bits is a byte, we sometimes call 4 bits a nibble. Since a byte is eight bits, this means a byte is two nibbles, or two hex digits:

0x1f is the byte 00011111_2

This makes hex a convenient notation for writing bytes of random data, like keys.

Appendix C – Hex to Binary Translation

The following table provides the translation of hex to binary.

Hex digit	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111