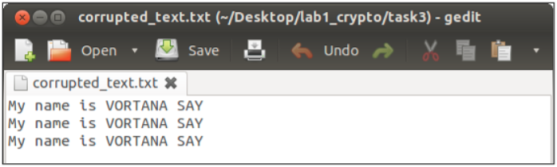
Crypto Lab – Secret-Key Encryption (Part 2)

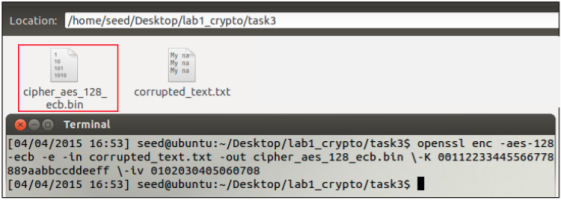
[MAY 30, 2015](https://vortana.wordpress.com/2015/05/30/crypto-lab-secret-key-encryption-part-2/) [VORTANA SAY](https://vortana.wordpress.com/author/vortanasay/)[LEAVE A COMMENT](https://vortana.wordpress.com/2015/05/30/crypto-lab-secret-key-encryption-part-2/#respond)

In the previous part, I showed the first two big parts of the crypto lab of the SEED Lab. In this part, I am going to discuss the rest of the tasks in the labs.

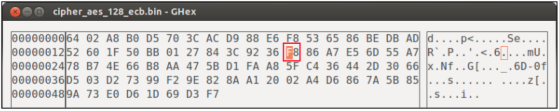
**3.  Encryption Mode – Corrupted Cipher Text**

  
I created a plain text file whose size is 69 bytes. I will encrypt this file using AES-128 and encryption mode, ECB, CBC, CFB, and OFB respectively. After that I will change a single bit of the 30th byte, 1E in hexadecimal value, so that I can get the corrupted encrypted file. Then I will decrypt the corrupted encrypted file using its encryption mode and explain the differences of the results.

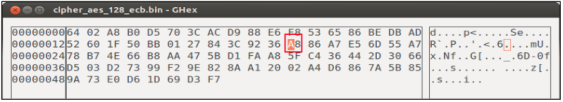
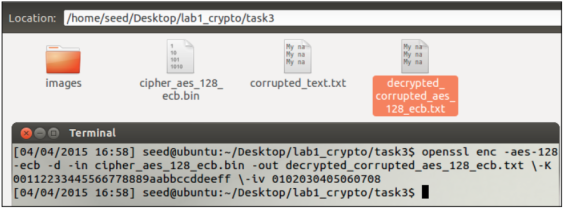
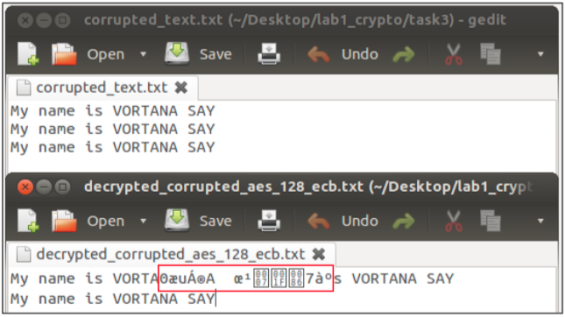
**Encryption mode – ECB**

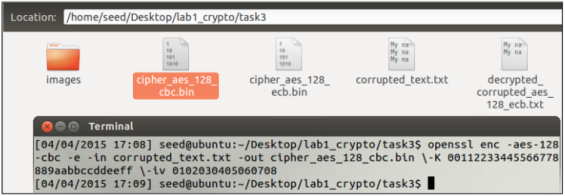
I encrypted the plain text, called corrupted\_text.txt.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-33-13.png)

* Original encrypted file

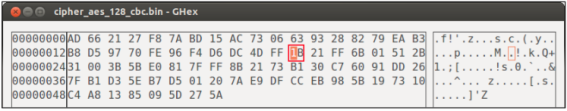
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-36-16.png)

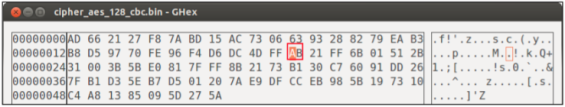
* Corrupted encrypted file

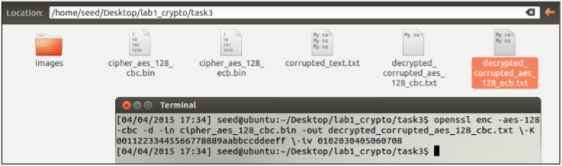
  
In the original encrypted file, the position 30th byte is 1E corresponds to the value F8, I changed the single bit of this value from F to A so the value of 30th byte now is A8 then I saved the file. Consequently, I got a corrupted encrypted file.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-37-58.png)  
I decrypted the encrypted file, cipher\_aes\_128\_ecb.bin which is became corrupted. I received a decrypted file, decrypted\_corrupted\_aes\_128\_ecb.txt.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-38-58.png)

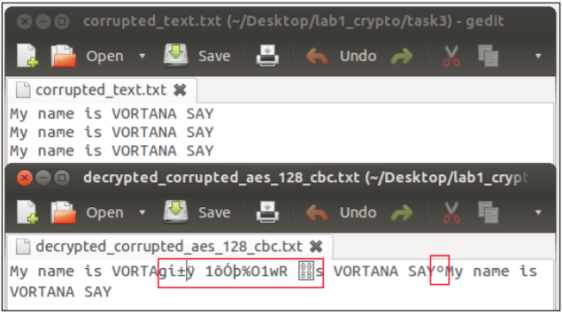
**Encryption mode – CBC**  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-39-37.png)

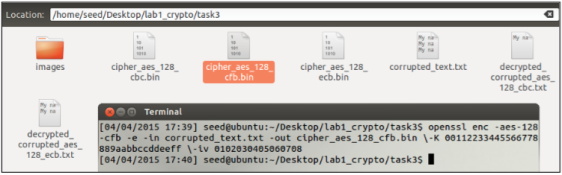
Original encrypted file

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-40-57.png)

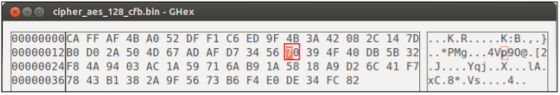
Corrupted encrypted file  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-41-35.png)

In the original encrypted file, the position 30th byte is 1E corresponds to the value 1B, I changed the single bit of this value from 1 to A so the value of 30th byte now is AB then I saved the file. Consequently, I got a corrupted encrypted file.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-42-10.png)

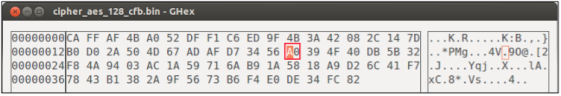
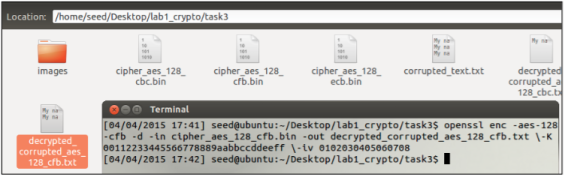
Here is the result:  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-42-47.png)

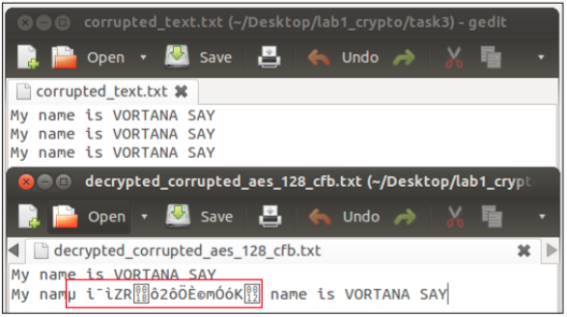
**Encryption mode – CFB**  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-43-35.png)

* Original encrypted file

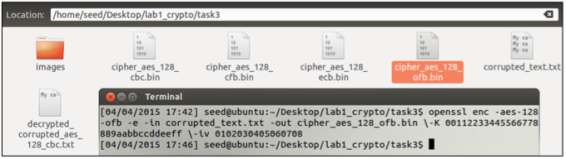
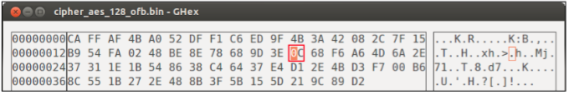
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-44-20.png)

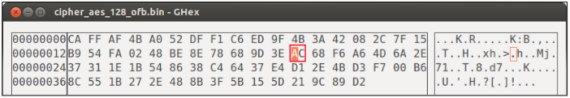
* Corrupted encrypted file

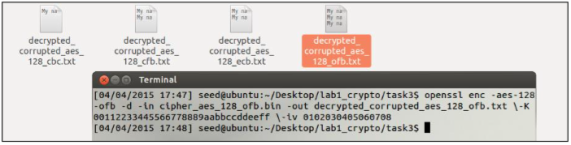
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-45-19.png)  
In the original encrypted file, the position 30th byte is 1E corresponds to the value 70, I changed the single bit of this value from 7 to A so the value of 30th byte now is A0 then I saved the file. Consequently, I got a corrupted encrypted file.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-46-06.png)

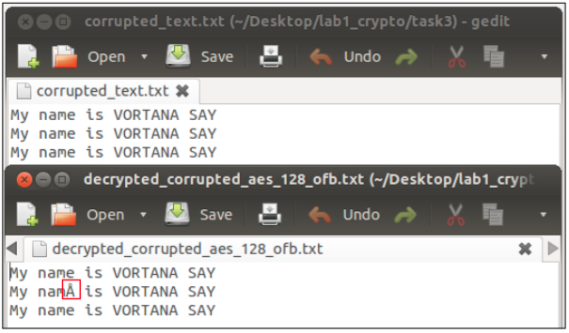
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-46-39.png)

**Encryption mode – OFB**

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-47-17.png)  
Original encrypted file  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-47-33.png)

Corrupted encrypted file  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-48-42.png)

In the original encrypted file, the position 30th byte is 1E corresponds to the value 0C, I changed the single bit of this value from 0 to A so the value of 30th byte now is AC then I saved the file. Consequently, I got a corrupted encrypted file.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-49-21.png)

Here is the result:  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-49-30.png)

**Conclusion**  
From the results above I can conclude that:

* In ECB mode, only one block is affected when any problem in a cipher text happens; furthermore; moreover, each block is decrypted independently. However, the corrupted bit of the 30th byte in cipher text block 8 bytes might spread to all n bits in plaintext block 8 bytes since we do the decryption one block at a time.
* In CBC mode, there was affect in two blocks.
* In CFB mode, there is problem in n / r number of blocks. Therefore, the error propagation criterion is poorer (Modes of Operation of Block Ciphers).
* In OFB mode, the feedback is only in the key-generation system. If the single digit of the 30th byte corrupted, then in plain text that only that byte or character is corrupted.**Thus, only OFB mode shows the most promising result and almost all the texts are recovered.**

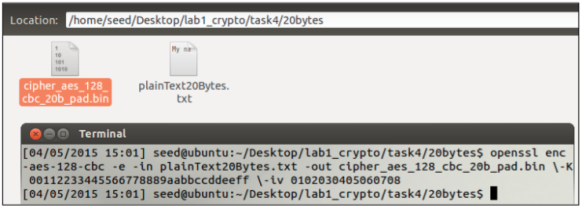
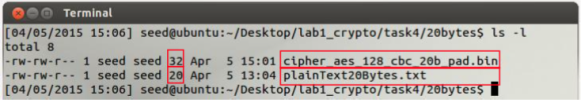
**4.  Padding**

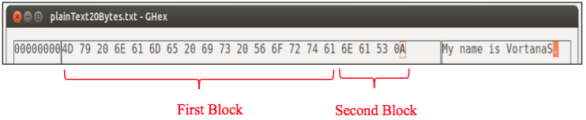
*Lab description: For block ciphers, when the size of the plaintex is not the multiple of the block size, padding may be required. In this task, we will study the padding schemes. Please do the following exercises:*

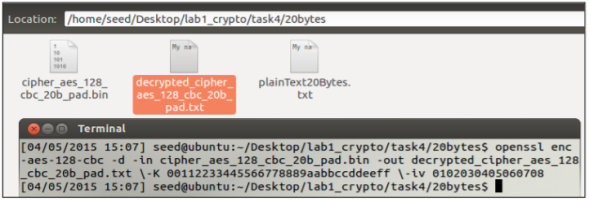
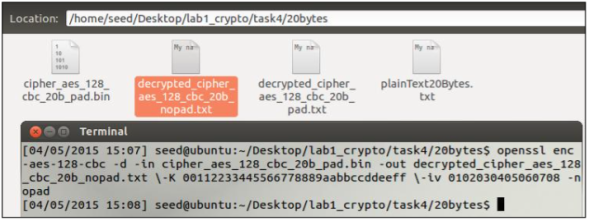
* *The openssl manual says that openssl uses PKCS5 standard for its padding. Please design an experiment to verify this. In particular, use your experiment to figure out the paddings in the AES encryption when the length of the plaintext is 20 octets and 32 octets.*
* *Please use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.*
* **Experiment to verify OpenSSL uses PKCS5**

Block cipher algorithms like AES and Triple DES in Electronic Code Book (ECB) and Cipher Block Chaining (CBC) mode require their input to be an exact multiple of the block size. If the plaintext to be encrypted is not an exact multiple, you need to pad before encrypting by adding a padding string. When decrypting, the receiving party needs to know how to remove the padding in an unambiguous manner (Block cipher mode of operation, 2015).

All the block ciphers normally use PKCS#5 padding also known as standard block padding and cipher block of 8 bytes. I created two files, 20 bytes and 32 bytes, and these files are encrypted by AES-128-CBC. The block size of 128 bits is equal to 128/8 = 16 bytes.

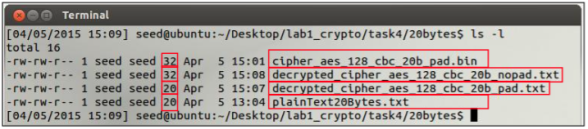
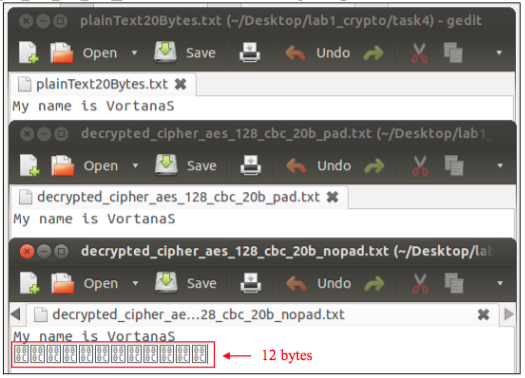
***Here is file whose size is 20 bytes.***  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-59-17.png)  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-22-59-25.png)

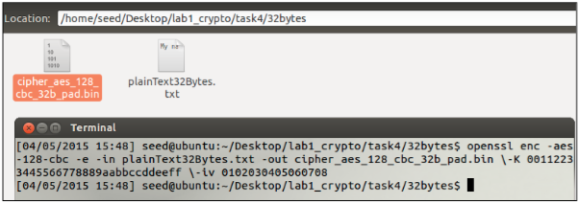
As can be seen, the size of the encrypted file is 32 bytes, 12 bytes more than the original file, 20 bytes. In other word, 12 bytes of padding is added to the encrypted file.  
Because I used AES-128-CBC, so each block is 16 bytes. The content of the files is 20 bytes, so the first block is 16 bytes, and the second block is 4 bytes. Thus, it needs 12 bytes to fill in 16 bytes.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-00-51.png)

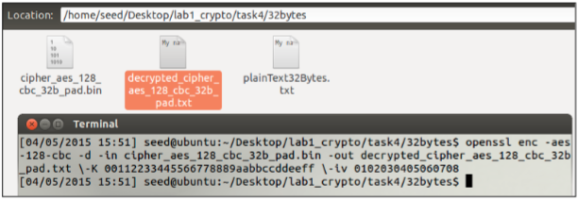
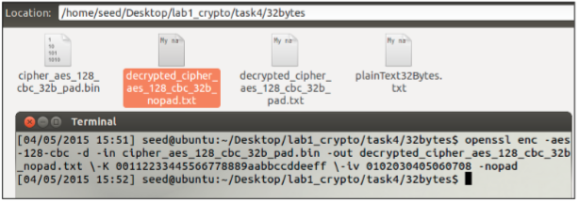
Another way to verify that openssl use the PKCS5 padding is to decrypt the encrypt file with option –nopad. This option indicates disable standard block padding. Normally during the encryption by default, it will include the padding, so when decrypt the file if I use the nopad option I can see the padding in the decrypted file.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-01-58.png)  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-02-05.png)

I decrypted two times from the encrypted file with different option, default and –nopad.

* decrypted\_cipher\_aes\_128\_20b\_pad.txt (option default)
* decrypted\_cipher\_aes\_128\_20b\_nopad.txt (option nopad)

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-03-48.png)  
The size of this file, decrypted\_cipher\_aes\_128\_20b\_pad.txt, 20bytes the same as plainText20Bytes.txt. The size of this file, decrypted\_cipher\_aes\_128\_20b\_nopad.txt, 32bytes the same as cipher\_aes\_128\_cbc\_20b\_pad.bin. Thus, it contains the padding.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-05-12.png)

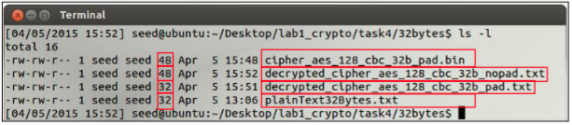
Here is file whose size is 32 bytes.  
Because I used AES-128-CBC, so each block is 16 bytes. The content of the files is 32 bytes, so the first block is 16 bytes, and the second block is 16 bytes. However, it needs padding so it needs 16 bytes more at the end of the encrypted file. The size of the encrypted file is 48 bytes, 16 bytes more than the original file, 32 bytes. In other word, 16 bytes of padding is added to the encrypted file.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-05-53.png)

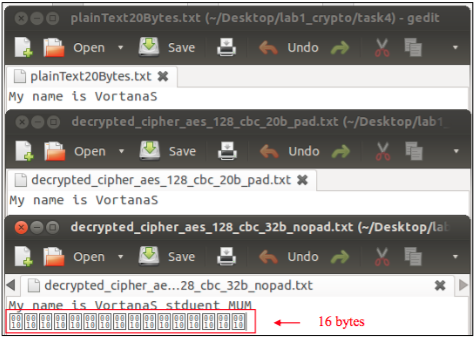
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-06-26.png)  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-06-33.png)

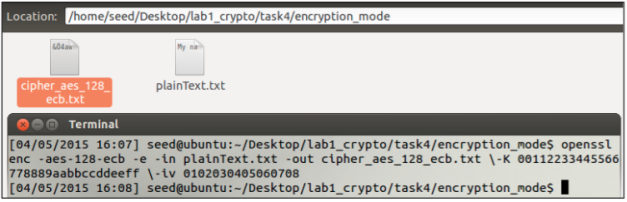
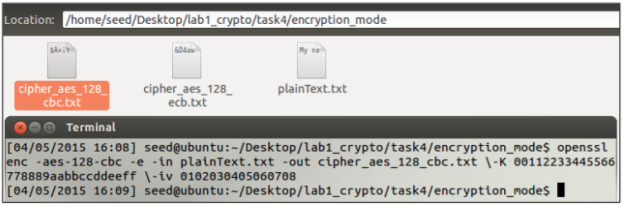
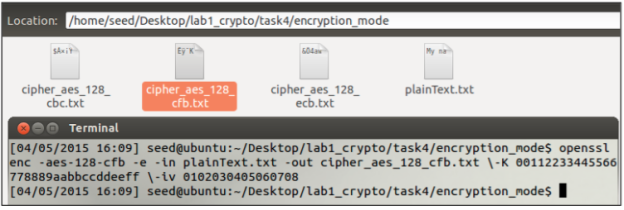
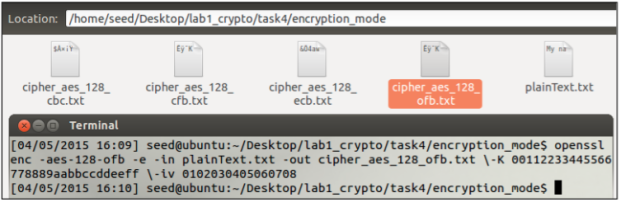
Again I decrypted two times from the encrypted file with different option, default and – nopad.

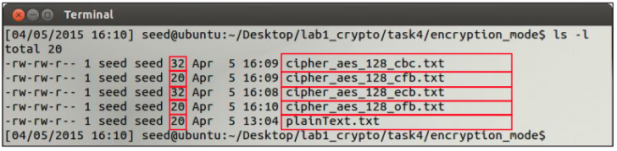
* decrypted\_cipher\_aes\_128\_32b\_pad.txt
* decrypted\_cipher\_aes\_128\_32b\_nopad.txt

(option default) (option nopad)

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-07-47.png)

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-08-14.png)

**Encryption modes and padding**  
In order to know which encryption mode needs padding, I created a plaintext file whose size 20 bytes. I encrypted this file with cipher AES-128 and different encryption mode, ECB, CBC, CFB and OFB respectively, then I will shows the size of each encrypted files to know if padding is added to any file.  
**ECB**  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-09-32.png)  
**CBC**  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-09-43.png)  
**CFB**  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-09-51.png)  
**OFB**  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-09-59.png)

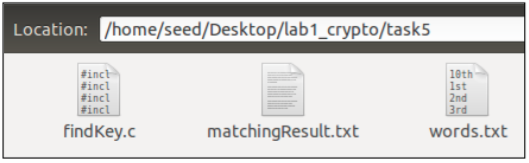
**Result**  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-12-16.png)

* Padding is needed for ECB and CBC encryption modes because their inputs contain number of blocks, thus padding could ensure that. Block size depends on the algorithm: AES uses 16 byte blocks while Blowfish and 3DES use 8-byte blocks.
* There is no need padding for encryption mode CFB and OFB because they are stream ciphers, in which the size of the block is usually fixed (one character).

**5.  Programming using the Crypto Library**

The goal of this task is to find the right key that was used to encrypt the given plain text, “This is a top secret.”.Given (refers to the lab description above):

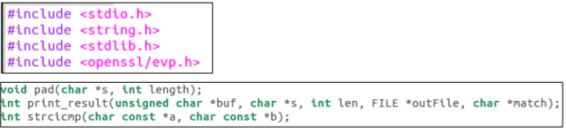
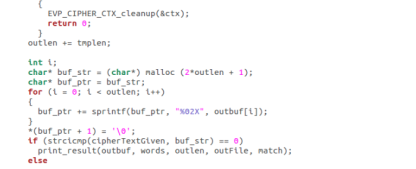
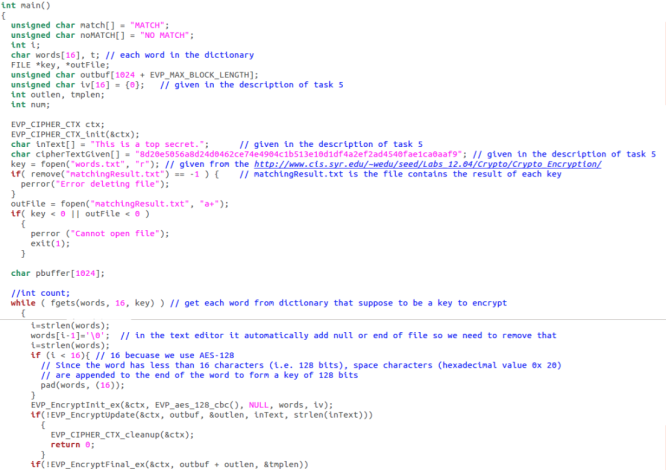
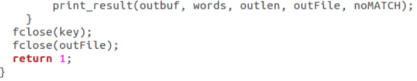
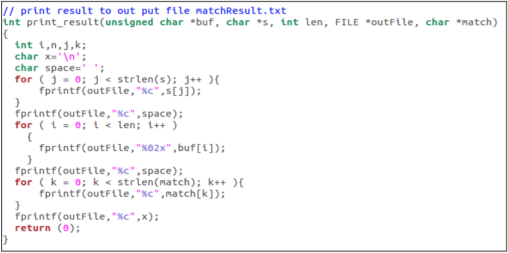
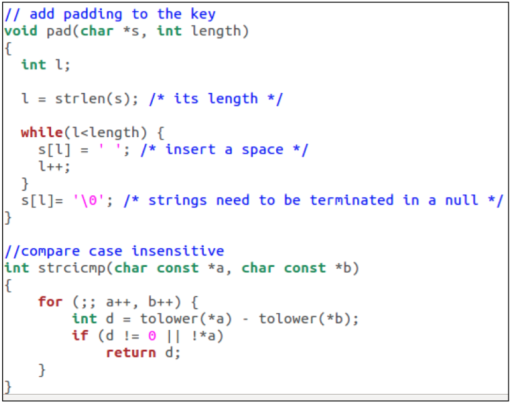
* Cipher was used is AES-128-CBC
* Numbers in the IV are all zeros
* Key that used an English word shorter than 16 characters
* Cipher text that resulted from the encryption from the plain text is8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9**Main condition:**  
  IF ( ChipherText = Encryption(Plain text, Key) )  
  Key is match;  
  ELSE  
  KEY is no match;

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-16-39.png)

I have three files:

* findKey.c is a c file that contains c code using the OpenSSL crypto library API EVP
* matchingResult.txt is a text file contains result from matching
* words.txt is a text file contains a English word list; it is given by the lab at <http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Crypto/Crypto_Encryption/>Here is some parts of the words.txt:

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-18-39.png)

Here is the code in fileKey.c; the images are ordered by the ordered of the code. I also uploaded code files to sakai lab1\_crypto folder. There are comments in the codes that help understand well.  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-27-29.png)  
[[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-28-06.png)](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-28-06.png)  
  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-33-33.png)  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-33-42.png)

In the code above, I read word line by line from the words.txt and do the encryption then using the main condition to test if the key is match. Then I save the results to the file matchResult.txt.

**Compile and run the code**

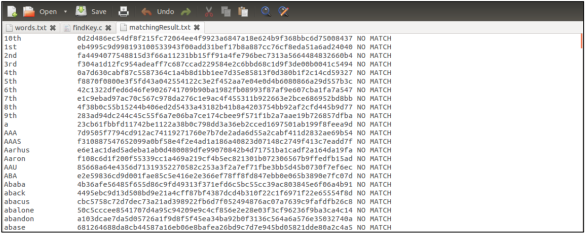
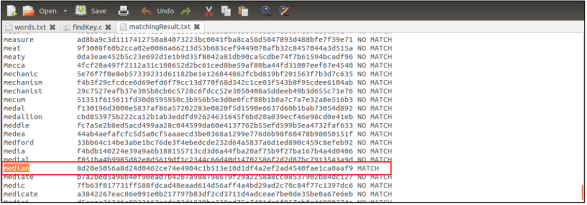
In the description of the lab, it states that we need to configure make file as follow:  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-35-29.png)

However, I don’t think we need to create makeFile for this exercise; we need to tell the gcc compiler where is the include folder and lib folder of the openSSL. Because I used SEEDUbuntu image provided by the lab, so the location of the include folder and lib folder are different from the instruction.  
Here is how I compile my code:

[Screen Shot 2015-05-29 at 23.36.53](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-36-53.png)

**Result**  
I saved the result in f matchingResult.txt in following format:  
[Screen Shot 2015-05-29 at 23.38.24](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-38-24.png)

Here is the contents of the matchingResult.txt:

[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-39-08.png)  
Here is the part that matched key found:  
[](https://vortana.files.wordpress.com/2015/05/screen-shot-2015-05-29-at-23-39-17.png)