

Workshop 2 – Symmetric Ciphers

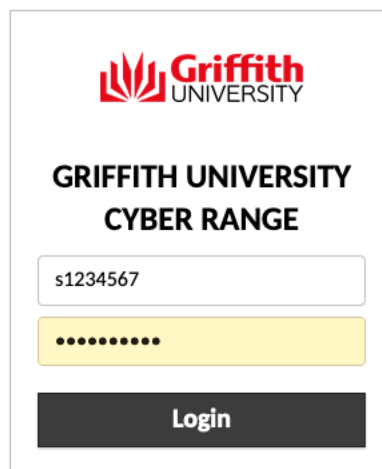
In this workshop, we will be reviewing some important concepts about symmetric ciphers. We will also see how to encrypt practically using symmetric ciphers implemented in OpenSSL and verify some security issues and properties of modern symmetric ciphers.

The practical lab environment can be accessed through Griffith Cyber Range. It is an Ubuntu Linux operation system (OS). Make sure you type all commands yourself if you are not familiar with Linux OS. Please note that what starts from the symbol “#” is not part of the command in our examples, i.e., it will be executed by the OS. These are called comments. They are meant to provide readable explanations of the commands or code.

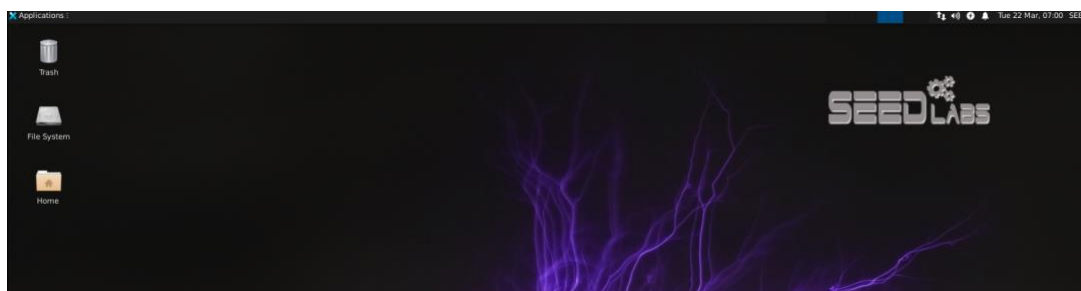
Preparation

1. Install Griffith University’s VPN software and connect. You can find the information about Griffith VPN from [here](#). (Please note, you will need VPN for the practical workshops and assignments.)

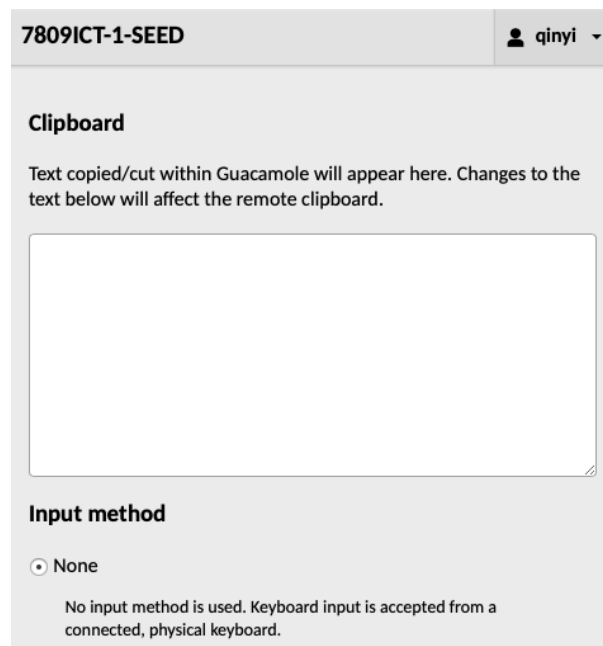
1. Go to cyber.ict.griffith.edu.au. Login using your s-Number and password: changeme.

A login form for Griffith University Cyber Range. At the top is the Griffith University logo. Below it, the text "GRIFFITH UNIVERSITY CYBER RANGE" is centered. There are two input fields: the first contains the s-number "s1234567" and the second is a password field with masked characters. Below the password field is a dark grey "Login" button.

2. You are now entered into the virtual machine (VM). For the workshops for the first 6 lectures, you will have access to a VM that has Ubuntu Linux OS installed.

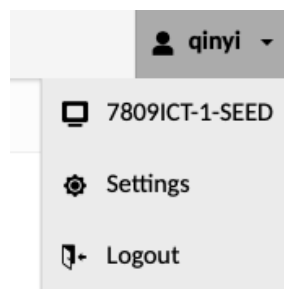


3. If this is your first-time login to Cyber Range, you should change your password as follows. Type “Ctrl+Alt+Shift” for Windows or “Ctrl+Command+Shift” for MAC to get the control panel.



Click on your username from and select “Settings”, then “Preferences”, input a new password and then click on “Update Password”. You will then be able to login using your now password in the future (you can always comeback and change your passwords).

4. To exit Cyber Range, make sure always use “Ctrl+Alt+Shift” for Windows or “Ctrl+Command+Shift” for MAC, click on the username, and select “Logout”. You can always reconnect after logout. Please DO NOT shut down your virtual machines.



Task 1. Review Important Concepts

1. What is the difference between a block cipher and a stream cipher?

Answer:

Stream ciphers are ciphers that perform a bitwise XOR operation on the plaintext input using a key, these two inputs are used to generate a ciphertext output. Usually the key is a pseudorandom bit stream deterministically generated using a PRNG algorithm that takes a 128 or 256 bit key, generated in a truly random manner, for reasons that will be discussed in part 2 of Task 1.

Block ciphers are ciphers that split the plain text into blocks of typically 64 or 128 bits data segment blocks, each of these segments are ciphered using stream cipher implementations.

Block ciphers have various implementation methods commonly referred to as modes, some of these modes use the cypher text output of preceding blocks and use this output as an input in the proceeding blocks stream cipher operation. This interaction between consecutive blocks in the encryption and decryption phases, how each block encrypts and decrypts in its owns stream, how keys and initialization vectors (IV) are used in each block (if an IV is used), and how block output if fed as an input into the next block (if it is fed into, that is), is what is referred to as operation modes of well known ciphers (i.e. aes,aria,des,rc2...etc.).

This implementation of block ciphers aims at causing “diffusion and confusion” where diffusion is the jumbling the statistical structure of plain text by distributing the pattern found in plain text and confusion is where relationships between ciphertexts and encryption keys are mystified to reduce any chances of attackers deducing correlations between keys and ciphertexts.

2. What are the issues that make OTP impractical?

Answer:

Although OTP is an unbreakable cipher that provides absolute security in the sense that is unconditionally secure, this ciphering technique also has a major practical disadvantage that makes it impractical to use on modern network communication protocols.

What makes the use of OTP encryption impractical is its key feature where the key used to encrypt the data is as long as the encoded data itself.

So a 5GB file would need a key that is also 5GB long (of truly random bits). This implementation of ideal security poses a major payload on network bandwidth. This makes the sharing of the symmetric key very difficult in practice.

The two main problems associated with using this encryption method is generating a truly random key string that is (bit-wise) as long as the plaintext, and the other problem is the distribution of such keys.

For this reason, the next evolution to this implementation is to use 128- or 256-bit (truly) random seeds to generate a pseudorandom key used to encrypt plain data.

The solution to the disadvantage OTP provides is using a seed that is a random bit stream fed into pseudorandom number generators that are put together in well-designed topologies to generate keys that are as random enough to be considered secure.

3. What is a product cipher?

Answer:

Product ciphers use mechanism that cause the “diffusion and confusion” phenomena when trying to decrypt ciphertext. Product ciphers take their name from the fact that they cause confusion by blurring the statistical structure found in plain text, by breaking and randomizing pattern between plaintext and cipher texts so there is no relationship between the ciphertext and the relative frequency of text found in plain texts. This is diffusion

The second dimension of this product is confusion where several rounds are used in the encryption process so an attacker is left confused when trying to establish a relationship between ciphertexts and possible keys, even if the attacker knows which encryption algorithm was used. This is best achieved with block ciphers that feed the output of one block into the other and utilise an IV.

The combination of these two qualities is what makes a cipher a product cipher.

4. What is the difference between diffusion and confusion?

Answer:

Diffusion is the jumbling of the statistical structure of plain text by distributing the patterns found in plain text, and confusion is where relationships between ciphertexts and encryption keys are mystified to reduce any chances of attackers deducing correlations between keys and ciphertexts as seen in autocorrelation analyses used against Vigenère ciphertexts.

The effects of diffusion as the ciphertext does not carry over the statistical relationships found in the original plain text. The effects of confusion is seen where the same plaintext produces a different cipher text with each encryption.

Task 2: Using symmetric ciphers implemented in OpenSSL to encrypt

In this part, we use the ciphers implemented in OpenSSL to perform encryption and decryption. OpenSSL is a software library for network security communication applications.

1. Create a plaintext file called plaintext.txt with a content. Encrypt plaintext.txt using the function 'enc' from OpenSSL

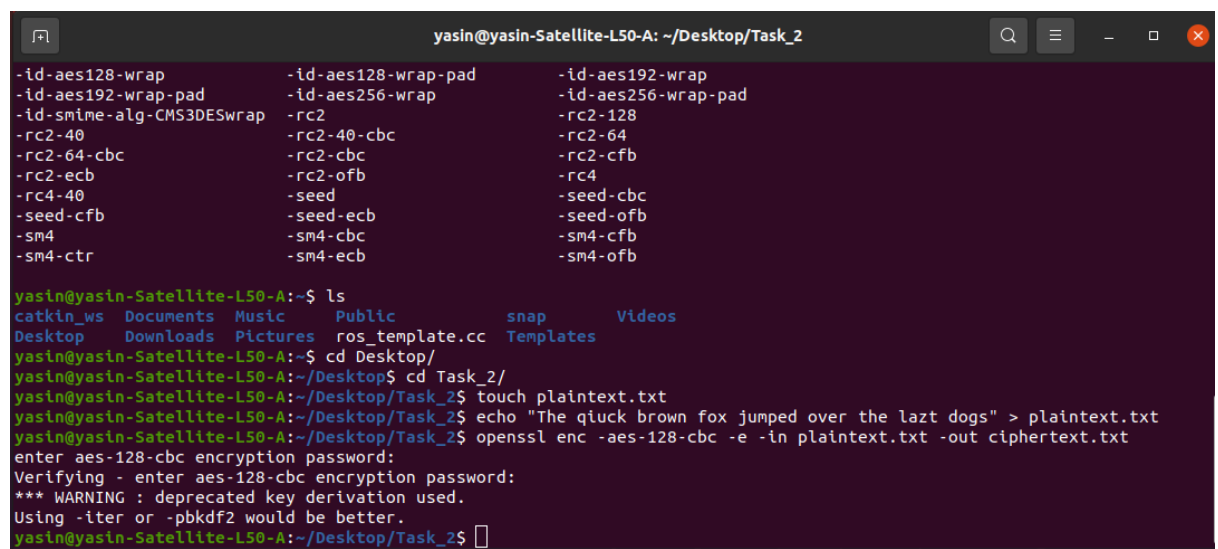
```
$ openssl enc -aes-128-cbc -e -in plaintext.txt -out  
ciphertext.txt -pbkdf2
```

You will be asked to provide encryption password. Choose any password that you remember, and press 'enter' (or 'return' if you are using Apple keyboard).

The option '-aes-128-cbc' specifies the cipher type to be AES algorithm with 128-bit-keys and CBC mode. Option '-e' stands for 'encryption'. The option 'in' and 'out' specifies the input file and the output file, respectively. Obviously, in the case, out input is plaintext.txt and the output is ciphertext.txt. Finally, pbkdf2 specifies using the key derivation function pbkdf2 for deriving keys.

You may notice that both encryption key and the initial vector (IV) which are required for CBC mode. Here they are generated using the pbkdf2.

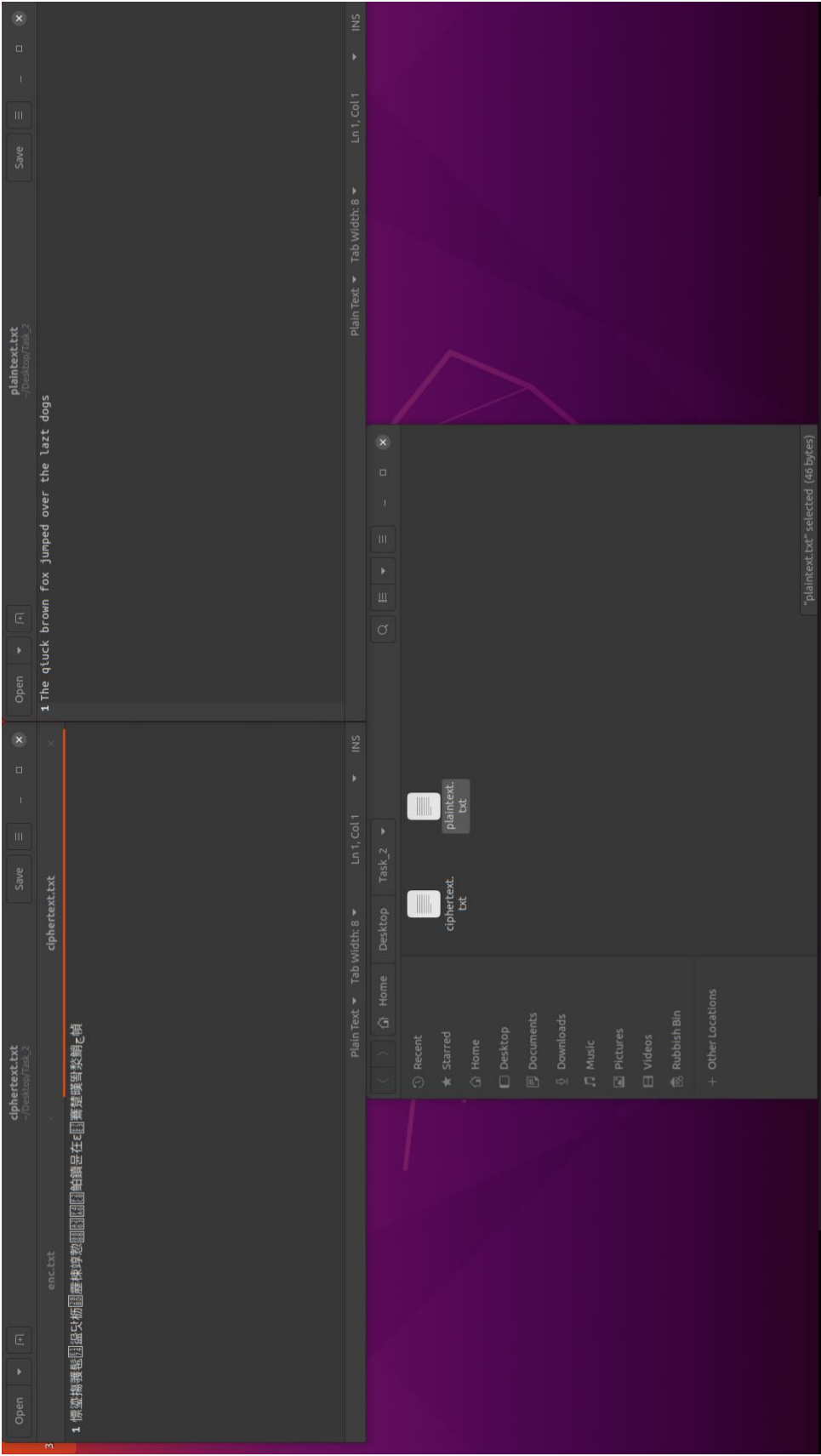
Answer:



```
yasin@yasin-Satellite-L50-A: ~/Desktop/Task_2
-id-aes128-wrap      -id-aes128-wrap-pad  -id-aes192-wrap
-id-aes192-wrap-pad -id-aes256-wrap      -id-aes256-wrap-pad
-id-smime-alg-CMS3DESwrap -rc2                -rc2-128
-rc2-40             -rc2-40-cbc         -rc2-64
-rc2-64-cbc         -rc2-cbc            -rc2-cfb
-rc2-ecb            -rc2-ofb            -rc4
-rc4-40             -seed               -seed-cbc
-seed-cfb           -seed-ecb           -seed-ofb
-sm4                -sm4-cbc            -sm4-cfb
-sm4-ctr            -sm4-ecb            -sm4-ofb

yasin@yasin-Satellite-L50-A:~$ ls
catkin_ws  Documents  Music      Public      snap        Videos
Desktop    Downloads  Pictures   ros_template.cc  Templates

yasin@yasin-Satellite-L50-A:~$ cd Desktop/
yasin@yasin-Satellite-L50-A:~/Desktop$ cd Task_2/
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ touch plaintext.txt
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ echo "The quick brown fox jumped over the lazy dogs" > plaintext.txt
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$
```



2. Display the content of ciphertext.txt to see if you can recognize the content. (You can use 'cat' or other commands)

Answer:

```
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ cat plaintext.txt
The quick brown fox jumped over the lazy dogs
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ cat ciphertext.txt
Salted__t@+B00hAac+000(W0H0fD0ln000@^yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$
```

3. Decrypt the ciphertext using your encryption password

```
$ openssl enc -aes-128-cbc -d -in ciphertext.txt -out
recovered.txt -pbkdf2
```

Here the option -d specifies it does decryption. Compare recovered.txt with plaintext.txt to see if the decryption correctly recovers the plaintext.

Answer:

```
Salted__t@+B00hAac+000(W0H0fD0ln000@^yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ ^C
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ openssl enc -aes-128-cbc -d -in ciphertext.txt -out recovered.txt
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ cat recovered.txt
The quick brown fox jumped over the lazy dogs
yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$ cat ciphertext.txt
Salted__t@+B00hAac+000(W0H0fD0ln000@^yasin@yasin-Satellite-L50-A:~/Desktop/Task_2$
```

4. OpenSSL supports many symmetric ciphers (and the corresponding operation modes that they support). The following command displays the supported ciphers

```
$ openssl enc -list
```

Try to use different ciphers to encrypt and decrypt different files other than text files (e.g., archive file .tgz) and show your teaching staff.

Answer:

```
yasin@yasin-Satellite-L50-A: ~/Desktop/Task_2
yasin@yasin-Satellite-L50-A:~$ openssl enc -list
Supported ciphers:
-aes-128-cbc      -aes-128-cfb      -aes-128-cfb1
-aes-128-cfb8     -aes-128-ctr      -aes-128-ecb
-aes-128-ofb      -aes-192-cbc      -aes-192-cfb
-aes-192-cfb1     -aes-192-cfb8     -aes-192-ctr
-aes-192-ecb      -aes-192-ofb      -aes-256-cbc
-aes-256-cfb      -aes-256-cfb1     -aes-256-cfb8
-aes-256-ctr      -aes-256-ecb      -aes-256-ofb
-aes128           -aes128-wrap      -aes192
-aes192-wrap      -aes256           -aes256-wrap
-aria-128-cbc     -aria-128-cfb     -aria-128-cfb1
-aria-128-cfb8    -aria-128-ctr     -aria-128-ecb
-aria-128-ofb     -aria-192-cbc     -aria-192-cfb
-aria-192-cfb1    -aria-192-cfb8    -aria-192-ctr
-aria-192-ecb     -aria-192-ofb     -aria-256-cbc
-aria-256-cfb     -aria-256-cfb1    -aria-256-cfb8
-aria-256-ctr     -aria-256-ecb     -aria-256-ofb
-aria128          -aria192          -aria256
-bf               -bf-cbc           -bf-cfb
-bf-ecb          -bf-ofb           -blowfish
-camellia-128-cbc -camellia-128-cfb -camellia-128-cfb1
-camellia-128-cfb8 -camellia-128-ctr -camellia-128-ecb
-camellia-128-ofb -camellia-192-cbc -camellia-192-cfb
-camellia-192-cfb1 -camellia-192-cfb8 -camellia-192-ctr
-camellia-192-ecb -camellia-192-ofb -camellia-256-cbc
-camellia-256-cfb -camellia-256-cfb1 -camellia-256-cfb8
-camellia-256-ctr -camellia-256-ecb -camellia-256-ofb
-camellia128      -camellia192      -camellia256
-cast             -cast-cbc         -cast5-cbc
-cast5-cfb       -cast5-ecb        -cast5-ofb
-chacha20         -des              -des-cbc
-des-cfb         -des-cfb1         -des-cfb8
-des-ecb         -des-ede          -des-ede-cbc
-des-ede-cfb     -des-ede-ecb      -des-ede-ofb
-des-ede3        -des-ede3-cbc     -des-ede3-cfb
-des-ede3-cfb1   -des-ede3-cfb8    -des-ede3-ecb
-des-ede3-ofb    -des-ofb          -des3
-des3-wrap       -desx             -desx-cbc
-idea128-wrap     -idea128-wrap-pad -idea192-wrap
-idea192-wrap-pad -idea256-wrap     -idea256-wrap-pad
-idea-cbc         -rc2              -rc2-128
-rc2-40           -rc2-40-cbc       -rc2-64
-rc2-64-cbc      -rc2-cbc          -rc2-cfb
-rc2-ecb         -rc2-ofb          -rc4
-rc4-40          -seed             -seed-cbc
-seed-cfb        -seed-ecb         -seed-ofb
-sm4             -sm4-cbc          -sm4-cfb
-sm4-ctr         -sm4-ecb          -sm4-ofb
yasin@yasin-Satellite-L50-A:~$ ls
catkin_ws  Documents  Music      Public     snap       Videos
```

openssl enc -aes-192-cfb was used on mp4 and docx files, the process of encryption and decryption worked just as well as it did with the test files.

Task 3: Verify the insecurity of ECB mode

We have discussed in the lecture that as the simplest operation mode for block ciphers, electronic cipher book (ECB) mode is only suitable for encrypting short message, e.g., a 256-bit random session key. In this task, we use ECB mode to encrypt an image to show the potential risk of encrypting long messages with ECB mode.

The image file, tux_clear.bmp can be accessed by going to <http://networksecurity.griffith.internal> from the web browser of the VM and then going to Week 3 folder.

1. We encrypt the file tux_clear.bmp and treat the encryption as a picture so we can view it. Recall a .bmp files uses the first 54 bytes to store some header information and the rest to store the information for the pixels. Thus, we avoid performing encryption to the first 54 bytes which may destroy the header information making the ciphertext not readable by results in a file not readable to the image viewer. We perform the followings to separate tux_clear.bmp into two parts and verify the concatenation of the of the two separated parts gives the original picture.

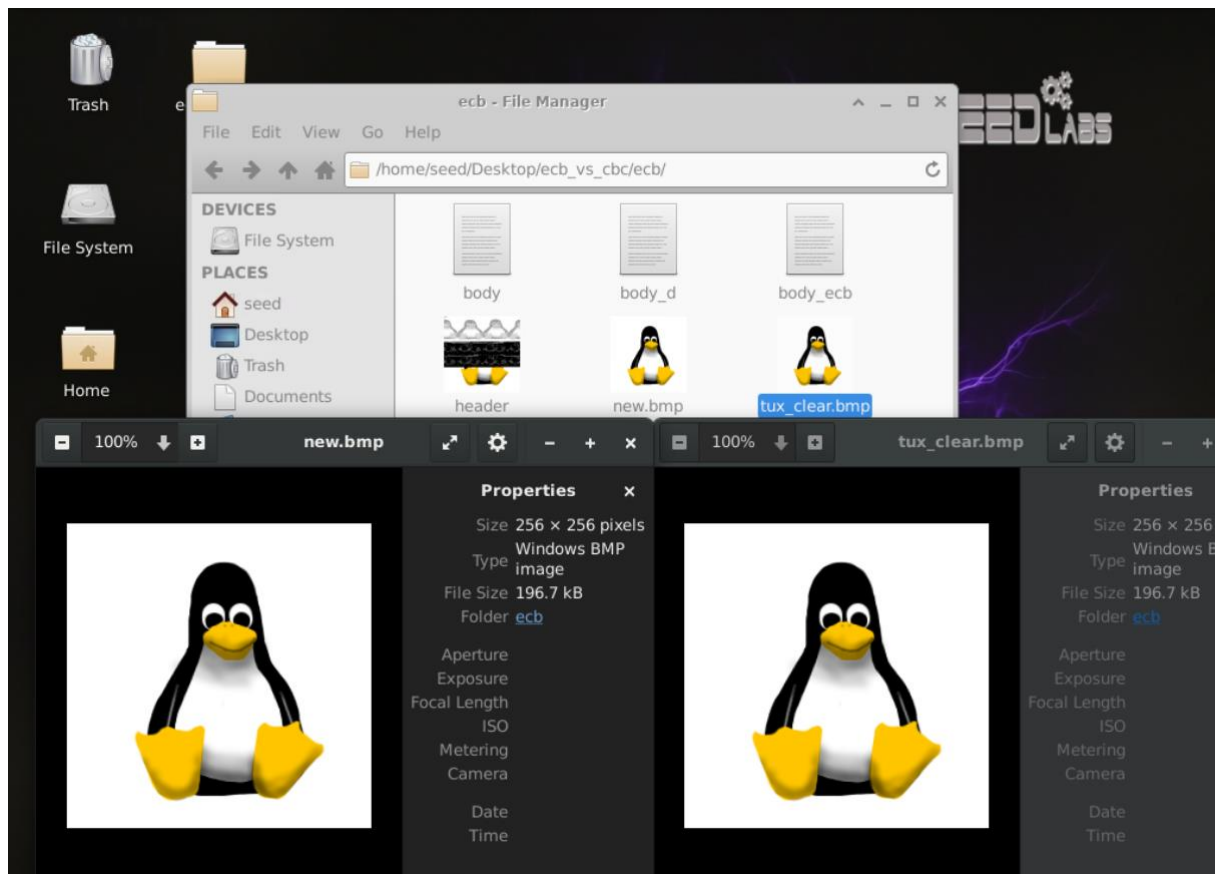
```
$ head -c 54 tux_clear.bmp > header
$ tail -c +55 tux_clear.bmp > body
```



```
$ cat header body > new.bmp
```

You can open new.bmp using the image viewer provided by the system to check if it shows the same image as tux_clear .bmp.

Answer:



Comments: When the head and the body of the file are combined the image editor is able to parse and process the image.

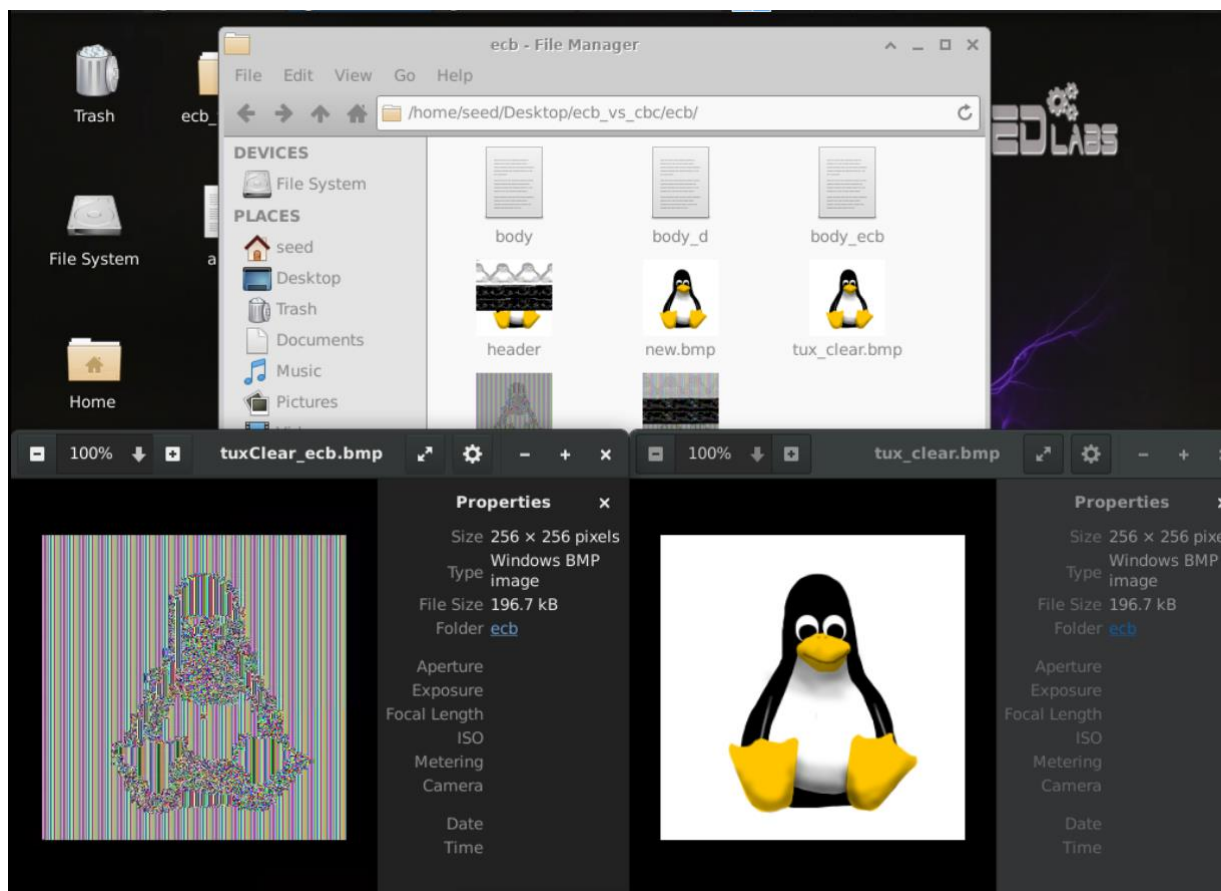
2. Using ECB mode to encrypt the file 'body'. Here we choose AES cipher with 128-bit keys. (You can choose any symmetric ciphers provided by OpenSSL with ECB mode.)

```
$ openssl enc -aes-128-ecb -e -in body -out body_ecb
# Encrypt the body of tux_clear.bmp to get a file body_ecb
# Choose your password for encryption when asked
```

```
$ cat header body_ecb > tux_ecb.bmp
# Concatenate the header and body_ecb
```

Check if you have obtained the encryption of the image tux_encrypted.bmp. Use an image viewer to open it. *What can you see? Can you explain the reason why we end up with such an image based on how ECB mode works?*

Answer:



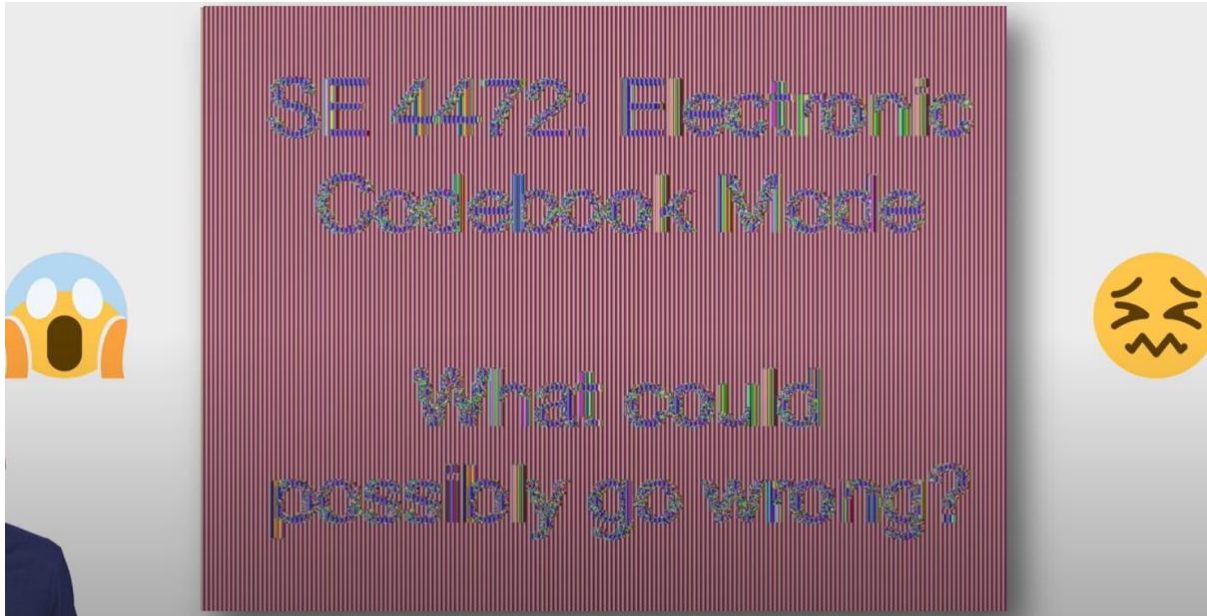
This experiment shows how ECB encryption is not very secure.

The reason for this is that ECB encryption is too serialised and deterministic, thus it lacks the design quality of confusion. The ECB block cipher lacks qualities of an ideal product cipher.

The randomisation ECB provides is restricted within the each block of the cipher. Although the key may provide some randomisation of plaintext elements within a block. The deterministic nature of this encryption method where each block is separately encoded only and the ciphertext of one block does not interact with the other. This type of separate block by block encryption preserves patterns between the blocks in ciphertext. In other words although the patterns within the blocks are diffused, the patterns of the plaintext are preserved as the cipher text displays the patterns across the ciphertext block elements. As a result the patterns within the block are jumbled, however the patterns across the cipher text blocks are preserved.

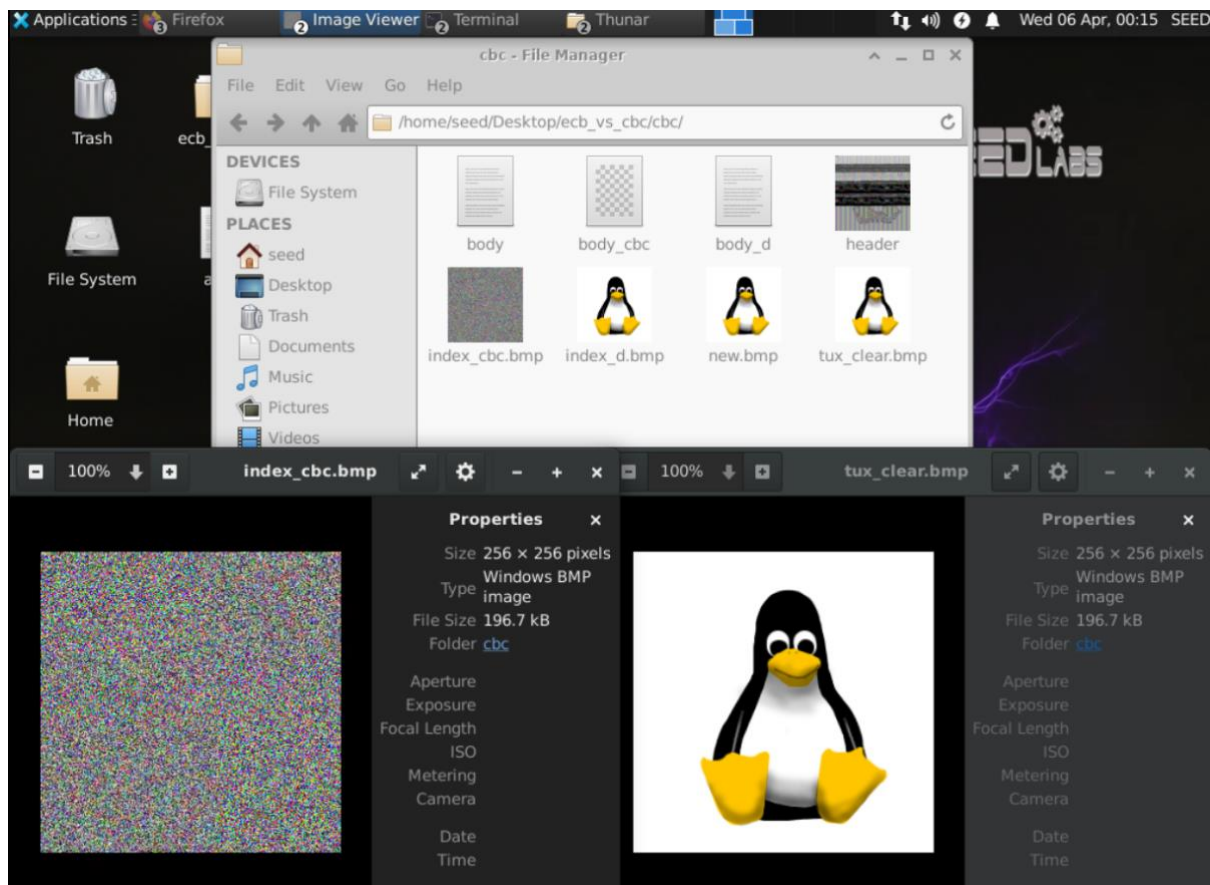
This pattern preservation across the ciphertext blocks is analogous to the patterns in the Vigenere Cipher showing up with autocorrelation analysis. That is the key size pattern across Vigenere cipher text is similar to the patterns showed across encrypted blocks in ECB encryptions. This is however only an analogy between a classic stream cipher and a very basic block cipher. The analogy is made with respect to how there is no confusion applied between ECB blocks as they encrypt like independent streams.

A more clear example of this is shown below where the background and foreground elements of an image are more contrasting, this example is taken from YouTube (found in this link <https://www.youtube.com/watch?v=oVCCXZfpu-w>) uploaded by a software engineer named Aleksander Essex:



3. Do step 2 again but with CBC mode (e.g., aes-128-cbc). Acquire the CBC-mode encryption of tux_clear.bmp, open it and compare it with tux_clear.bmp and tux_ecb.bmp. Explain your observation (e.g., based on how CBC mode works) to the teaching staff.

Answer:



A more secure encryption method would be to encrypt in CBC mode, which introduces the use of an initial vector (IV), which are essentially random numbers.

The initial vector is used in the first block to randomize the encryption in the block itself as well as all the remaining contiguous blocks. The ECB mode also feeds the ciphertext results into the next block which serves to hide the “boundaries” between consecutive blocks (be it 128, 192 or 256 bits in length). The ciphertext of each block is fed into the next block to be XORed with the plaintext of the proceeding block before being encrypted, this goes on for the remaining blocks.

The continuous feeding of preceding cipher texts in each successive block is how the CBC mode hides the patterns across the blocks. This is made even more unpredictable with the use of the IV bits in the initial block.

Task 4: Error Propagation

Let's verify how small error in ciphertext (e.g., due to noisy channels of communication networks) would affect recovering the original plaintext. We do so by create an encryption of an all-zero binary file, flip a single bit of the ciphertext, decrypt the flipped ciphertext and compare with the original all-zero file.

1. Encrypting the file a.txt using AES-128 with the different operation modes introduced in the class. For example:

```
$ openssl enc -aes-128-cbc -e -in a.txt -out cipher_cbc.bin -K  
00112233445566778899aabbccddeeff -iv  
01010202030304040505060607070808
```

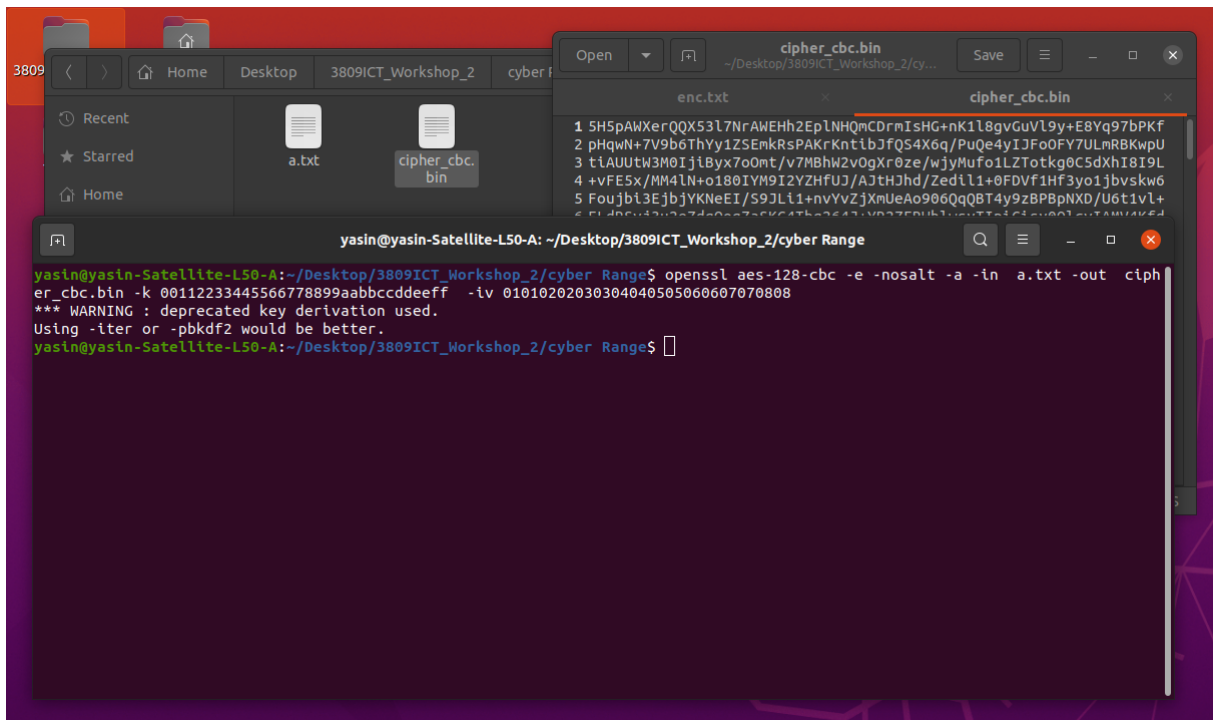
Here we explicitly give the key (option -K) and a initial vector (option -iv). You can do what we did before to use encryption passwords to generate pseudorandom keys and IVs. Doing that may create a salt string added before the result of encryption. So, when flipping a ciphertext bit, you should not flip a bit in the salt string, which will change the key value and IV value rendering incorrect decryption.

Answer:

The commands issued here did not work on ubuntu 20.04 LTS, the commands used in this workshop were:

```
openssl aes-128-cbc -e -nosalt -a -in a.txt -out cipher_cbc.bin -k  
00112233445566778899aabbccddeeff -iv  
01010202030304040505060607070808
```

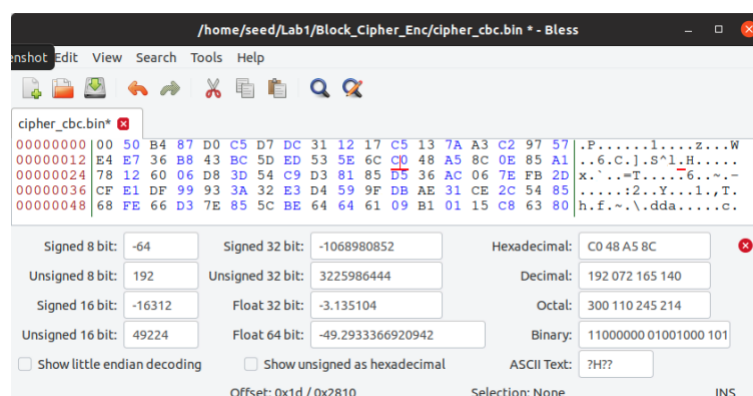
```
openssl aes-128-cbc -d -nosalt -a -in cipher_cbc.bin -out  
recovered_cfb.txt -k 00112233445566778899aabbccddeeff -iv  
01010202030304040505060607070808
```

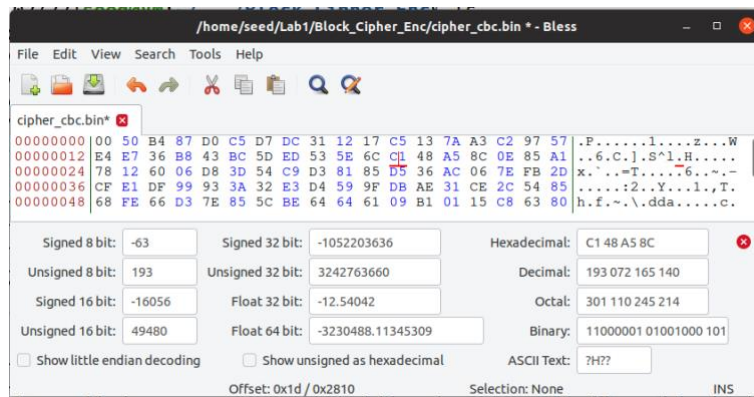


2. We use the hex editor called bless to edit cipher_cbc.bin.

```
$ bless cipher_cbc.bin
```

Note, bless displays bytes via hexadecimal numbers (0-f). So, you are editing hexadecimal numbers. So, just make you only flip one bit. For example, you can change 2 to 3 (in hexadecimal notations, you are changing 0010 to 0011). If you change 2 to 1, you are flipping two bits (i.e., 00).

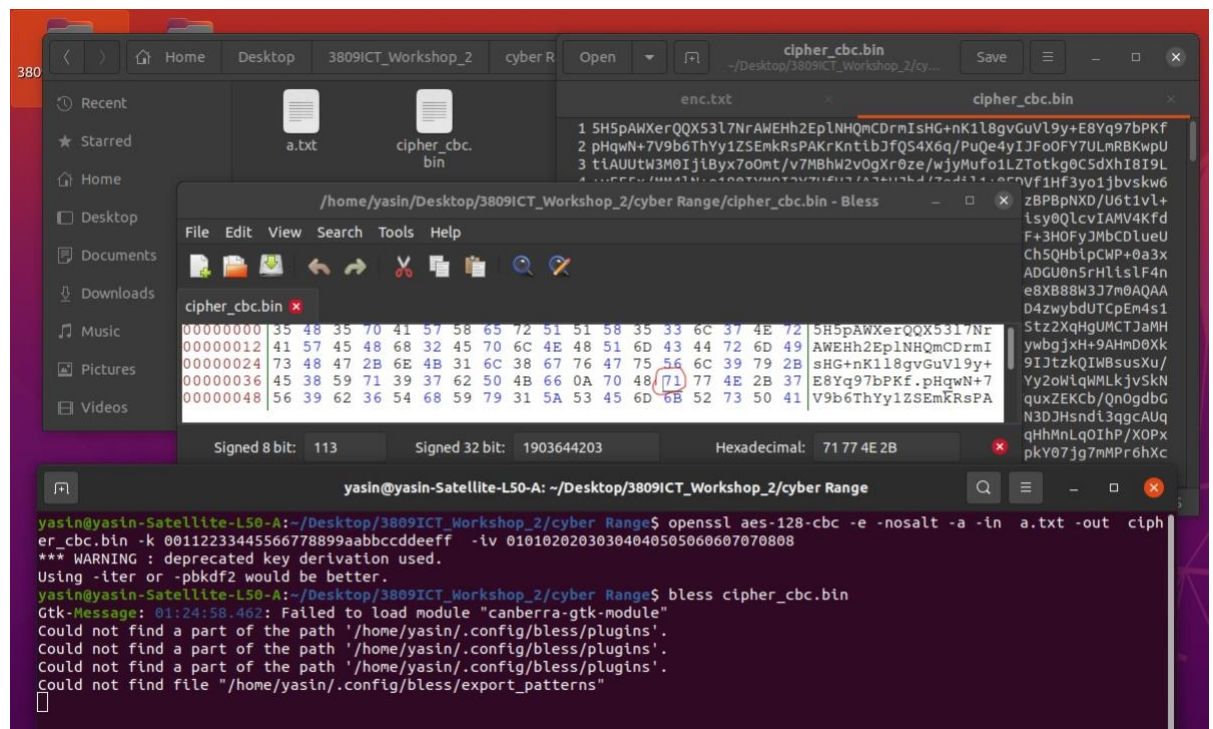




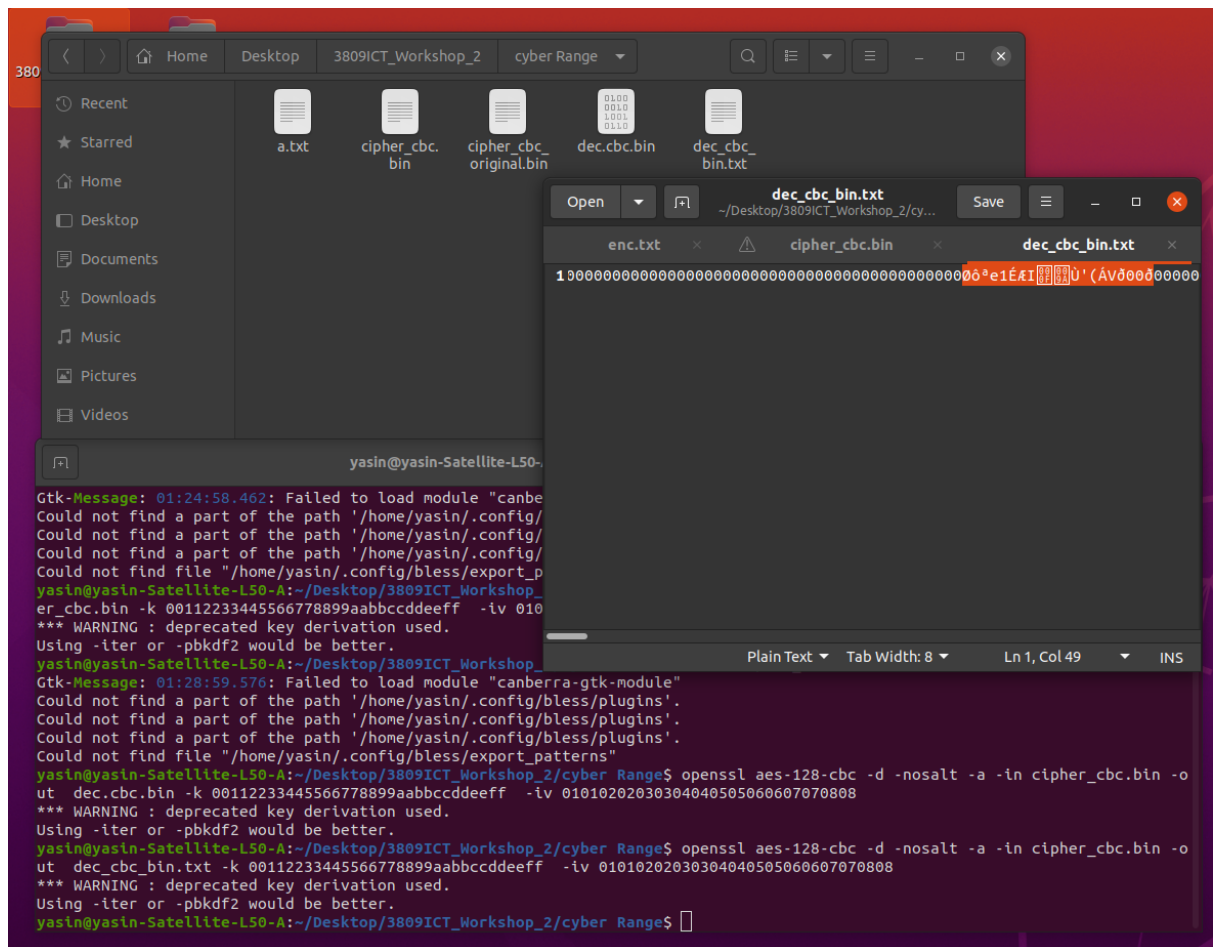
We flip “C0” in the second row to “C1”.

Answer:

The changes made in this assessment was in the 5th block different to the instructions here however only a single bit was altered.



Answer:



The result was a change as follows:

0000øø³e1ÉÆIšÙ'(ÁVð00ð000

We have an odd “ø” character among a stream of zero’s, the reason for this is when a single bit is flipped in a cipher block, this cipher text in this block feeds into the operation of the proceeding block that is used to decrypt using the key. So when one bit of ciphertext is flipped this bit will corrupt another bit in the next block since the error is propagated in a bitwise XOR operation to decrypt ciphertext into plain text.

When using the CBC mode changing one bit in ciphertext will only affect one ciphertext block (i.e the next block). If only one bit is corrupted in a block, this will only corrupt a single bit in the plain text of the next block, this will not be translated to the rest of the block up to the last padded block.

Task 5: PRNG with Weak Seeds

As a hacker, you have discovered a potential target Jack who always uses passwords with 4 decimal numbers. Also, you know Jack did some cryptography training and he always use AES-128 to encrypt important information. One day, you captured a ciphertext encrypted using

```
$ openssl enc -aes-128-cbc -e -in credit_card.txt -out  
top_secret.txt -pbkdf2  
# Encryption key and IV are generated using pbkdf2 as a PRNG  
and the encryption password is used as the seed
```

That is, you captured top_secret.txt and you know it was generated from the above command. You did some research and knew pbkdf2 is a highly secure PRNG. How can you recover credit_card.txt without a brute-force attack to AES-128-cbc? Explain your idea.

Answer:

The first possible method is to use a brute force attack with as many possibilities up to 10^4 , or by starting with 4-digit combinations that are most commonly used before exhausting as many as 10^4 combinations.

(Image below from <https://finance.yahoo.com/blogs/the-exchange/cracking-pin-code-easy-1-2-3-4-130143629.html>)

	PIN	Freq
#1	1234	10.713%
#2	1111	6.016%
#3	0000	1.881%
#4	1212	1.197%
#5	7777	0.745%
#6	1004	0.616%
#7	2000	0.613%
#8	4444	0.526%
#9	2222	0.516%
#10	6969	0.512%
#11	9999	0.451%
#12	3333	0.419%
#13	5555	0.395%
#14	6666	0.391%
#15	1122	0.366%
#16	1313	0.304%
#17	8888	0.303%
#18	4321	0.293%
#19	2001	0.290%
#20	1010	0.285%

However, this is not a practical approach a hacker or cryptanalyst would use.

Given the background knowledge that the seed key is a 4-digit string, this will be easy for a hacker as the 104 combination is easy for modern day computers (see the table below, taken from lecture 2 of 3809ICT, 2022)

Average Time Required for Exhaustive Key Search

Table 3.5 Average Time Required for Exhaustive Key Search

Key size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 decryptions/s	Time Required at 10^{13} decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	2^{55} ns = 1.125 years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	2^{127} ns = 5.3×10^{21} years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns = 5.8×10^{33} years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns = 9.8×10^{40} years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns = 1.8×10^{60} years	1.8×10^{56} years
26 characters (permutation)	Monoalphabetic	$26! = 4 \times 10^{26}$	2×10^{26} ns = 6.3×10^9 years	6.3×10^6 years

29

The process will be easy for the cryptanalyst as the seed password is a short range of numbers, so the attacker will have enough time to run through all possibilities to obtain a meaningful plaintext result.

Among the knowns to the cryptanalyst is the structure of credit card numbers. The cryptanalyst can use this knowledge as a condition to stop execution of the function once a meaningful plaintext result is obtained.

If the user had used a truly random password the resulting key and initialisation vector will be more truly random. In this case the password is chosen from a (very) short range of numbers. This is why the attacker will have enough time to run through possibilities obtain the original plain text data.

This is a practical example of using pseudo random number generators with weak seeds (i.e. passwords) as the key and initial vector output obtained from PRNG function is pseudo random, and thus all possibilities can be enumerated.

This weakness introduced by the user undermines the true strength of the CBC operation mode as a modern cipher.

The user either should generate a strong key and initial vector of truly random 128 bits or use a strong password as a seed that is both long enough as well as truly random.

Task 6 (Optional): Security of Double DES

We know that DES has key length that is too short to defend brute-force attack. One idea of improving DES was to use encrypt a message using DES twice with two different keys. Let E be the encryption algorithm of DES and k_1, k_2 be two DES keys. Such a double DES encryption performs like

$$c = E(k_2, E(k_1, m))$$

Now we might hope the total key length is 112 bits, enough for defending brute-force attack which should take 2^{112} steps. It turns out, such a system doesn't provide that level of brute-force security. Given one pair of plaintext-ciphertext pair, show how to brute-force the double encryption with complexity 2^{57} rather than 2^{112} .

This attack demonstrates that double symmetric ciphering doesn't increase the security much. In fact, to get 112 bits security, Triple-DES (a.k.a 3-DES) was introduced and standardised. You will see that in many security protocols, 3-DES remains an option

Acknowledgement: This lab instruction is partially based on the SEED labs from the SEED project led by Professor Wenliang Du, Syracuse University.