Ashlyn Goh 1002840 Cl02

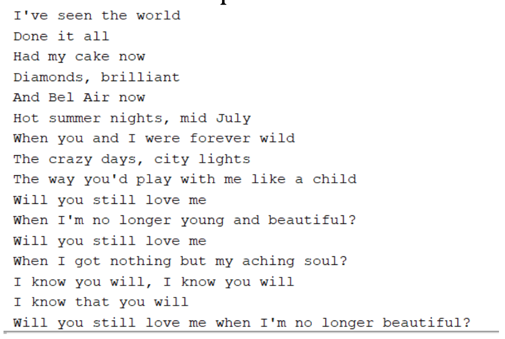
50.005 Lab #6 (NS Lab 2)

Part 1: Symmetric key encryption for a text file

Note: "printable" (in Questions 1 - 3) means human-readable (e.g., they are English letters or digits).

Question 1 (2pt): Try to print to your screen the content of the input files, i.e., the plaintexts, using *System.out.println()*. What do you see? Are the files printable?

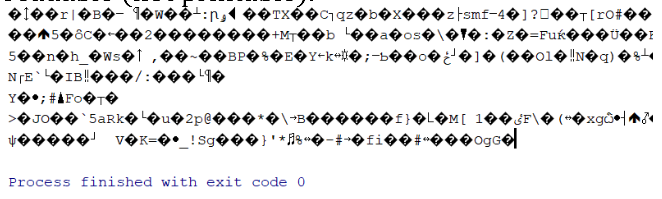
Using the BufferedReader to read the file and save the contents in a String, when we print this String, we are able to see the contents of the input file. The files are printable. The screenshot below is an excerpt of the contents of shorttext.txt.



```
I've seen the world
Done it all
Had my cake now
Diamonds, brilliant
And Bel Air now
Hot summer nights, mid July
When you and I were forever wild
The crazy days, city lights
The way you'd play with me like a child
Will you still love me
When I'm no longer young and beautiful?
Will you still love me
When I got nothing but my aching soul?
I know you will, I know you will
I know that you will
Will you still love me when I'm no longer beautiful?
```

Question 2 (3pt): Store the output ciphertext (in byte[] format) to a variable, say cipherBytes. Try to print the ciphertext of the smaller file using *System.out.println(new String(cipherBytes))*. What do you see? Is it printable?

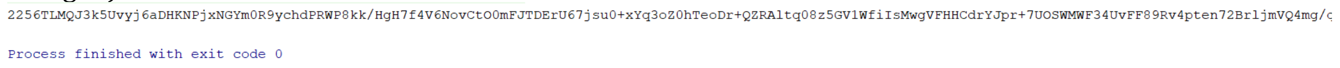
The result of printing the byte array without converting it to Base64 is shown below. It is not readable (not printable).



```
�↕��r|�B�- ¶�W��┴:ɲو◄ ��TX��C┐qz�b�X���z├smf-4�]?□���┬[rO#��
��🠉5�ðC�←��2����������+M┬��b └��a�os�\�▼�:�Z�=Fuḱ���Ü��
5���n�h_�Ws�↑ ,��~��BP�%�E�Y←k↔�;─ь���o�ج┘�]�(��Ol�‼N�q)�%┴
N┌E`└�IB‼���/:���└¶�
Y�•;#♟Fo�┬�
>�JO��`5aRk�└�u�2p@���*�\→B��������f}�└�M[ 1��ۥF\�(↔�xgὣ•┤🠉♂
ψ������┘  V�K=�•_!Sg���}'*♫%↔�-#→�fi��#↔����OgG�

Process finished with exit code 0
```

Question 3 (3pt): Now convert the ciphertext in Question 2 into Base64 format and print it to the screen. Is the Base64 encoded data generally printable?

The result of printing the Base64 encoded data is shown below. It is readable (in English letters or digits) but not understandable.



```
2256TLMQJ3k5Uvyj6aDHKNPjxNGYm0R9ychdPRWP8kk/HgH7f4V6NovCtO0mFJTDErU67jsu0+xYq3oZ0hTeoDr+QZRAltq08z5GV1WfiIsMwgVFHHCdrYJpr+7UOSWMWF34UvFF89Rv4pten72BrljmVQ4mg/

Process finished with exit code 0
```

Question 4 (3pt): Is Base64 encoding a cryptographic operation? Why or why not?

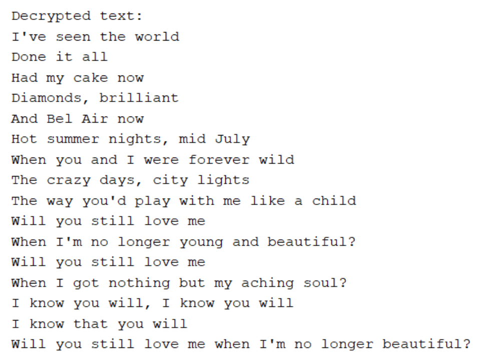
No, it is not.

Encoding is the conversion of data to a certain format. It is not transforming the data to keep it secret from others. In contrast, cryptography is a method of protecting information and communications using codes so that only those for whom the information is intended can read and process it.

Ashlyn Goh 1002840 CI02

Question 5 (3pt): Print out the decrypted ciphertext for the small file. Is the output the same as the output for question 1?

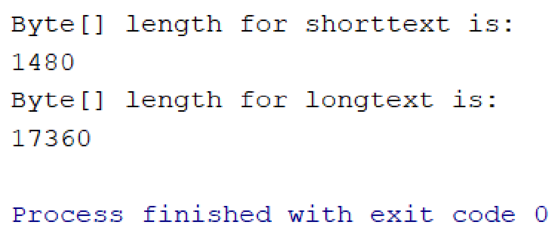
The screenshot below is an excerpt of the contents of decrypted ciphertext for the small file. The output is the same as that from question 1.



```
Decrypted text:
I've seen the world
Done it all
Had my cake now
Diamonds, brilliant
And Bel Air now
Hot summer nights, mid July
When you and I were forever wild
The crazy days, city lights
The way you'd play with me like a child
Will you still love me
When I'm no longer young and beautiful?
Will you still love me
When I got nothing but my aching soul?
I know you will, I know you will
I know that you will
Will you still love me when I'm no longer beautiful?
```

Question 6 (4pt): Compare the lengths of the encryption result (in byte[] format) for *smallFile.txt* and *largeFile.txt*. Does a larger file give a larger encrypted byte array? Why?

A larger file gives a larger encrypted byte array (as shown below).
This might be due to DES being a block cipher, meaning it operates on plaintext blocks of a given size (64-bits) and return ciphertext blocks of the same size. Hence, a larger file would be grouped into more 64-bit blocks, leading to a larger encrypted byte array.



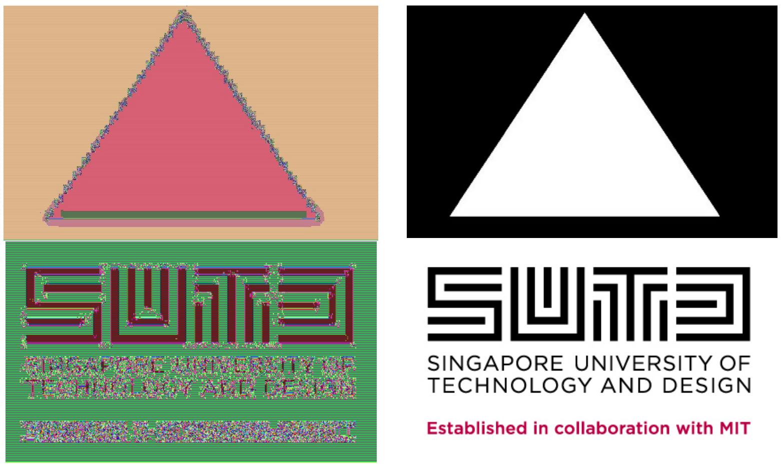
```
Byte[] length for shorttext is:
1480
Byte[] length for longtext is:
17360

Process finished with exit code 0
```

Ashlyn Goh 1002840 CI02

Part 2: Symmetric key encryption for an image file

Question 1 (4pt): Compare the original image with the encrypted image. What similarities between them do you observe? Can you identify the original image from the encrypted one?

Comparing the images, we can see that ECB mode encrypted version still clearly shows the outline of the original. Just like the original Triangle image, the encrypted one has the different colours for the background and the triangle, making it very easy to identify the original image. For the SUTD image, it is difficult to understand the words on the encrypted image but the main SUTD logo's outline can still be clearly identified.



Question 2 (5pt): Why do those similarities exist? Explain the reason based on what you find out about how the ECB mode works.

The ECB mode encrypts each 16-bytes block independently. Hence, blocks that were equal before encryption will remain equal after. This means that the group of pixels with the same colour will have the same (different colour from original) colour after encryption. We can see this from the images above where the whole triangle has the same pink colour because they were of the same colour before encryption. Repetitive areas in the input image result in repetitive patterns in the encrypted output, so many large-scale features of the image remain recognisable despite the encryption. This makes it easy for us to identify the original image after ECB mode encryption.

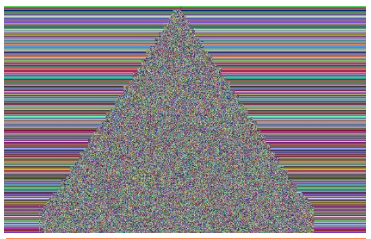
Ashlyn Goh 1002840 Cl02

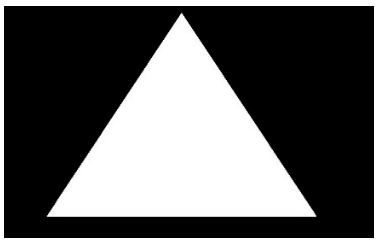
Question 3 (8pt): Now try to encrypt the image using the CBC mode instead (i.e., by specifying "DES/CBC/PKCS5Padding"). Compare the result with that obtained using ECB mode). What differences do you observe? Explain the differences based on what you find out about how CBC mode works.

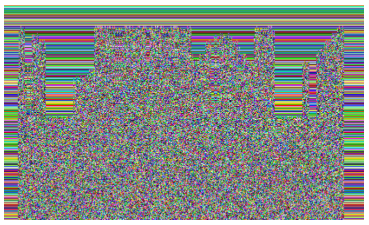
Comparing the images, we can see that it is harder to identify the original images from the encrypted ones that are done using CBC mode.

The main difference between the two set of encrypted images is the colour we get. The encrypted images using CBC mode have more variety in the colours, thus making them harder to identify. For the Triangle image, although the colours within the triangle are seemingly randomised, we can still see the outline of the triangle and thus difference between the ECB and CBC mode is not very significant. However, for the SUTD image, the difference between the two modes is huge. We are hardly able to identify the original image from the encrypted one using CBC mode.

The difference in colours is likely to be due to CBC using the results of encrypting the previous block to encrypt the current block. This chaining algorithm continues to the end of the file. Hence, this would ensure that every "#000000" hexadecimal colour will have a different output, thus causing colours to appear as random. This is why the encrypted images using CBC mode has their colours seemingly randomized and thus are harder to identify the original image.





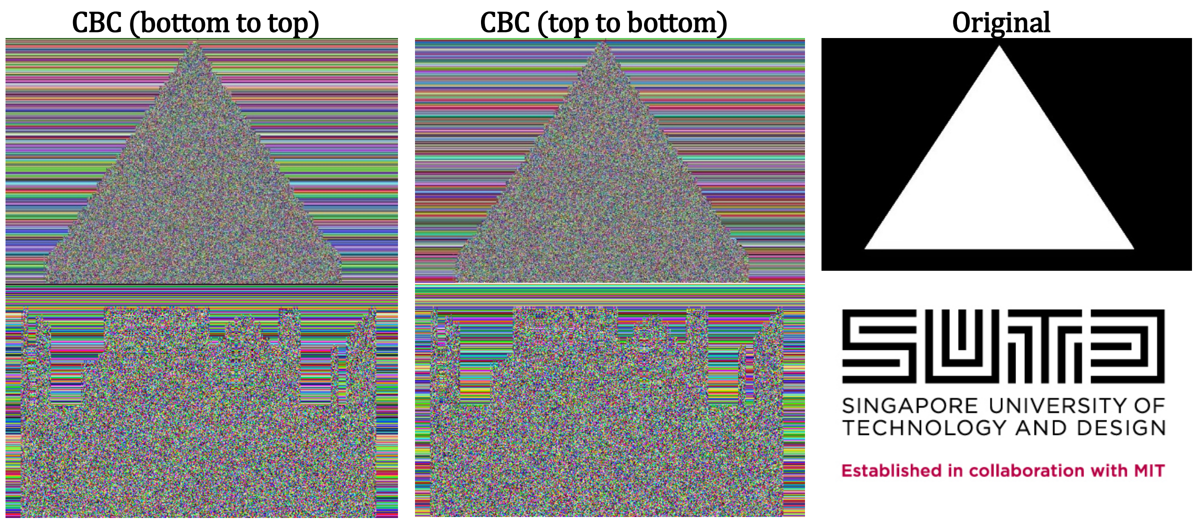


SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN
Established in collaboration with MIT

Question 4 (15pt): Do you observe any issue with image obtained from CBC mode encryption of "SUTD.bmp"? What is the reason of such observation? Can you explain and try on what would be result if data were taken from bottom to top along the columns of image? Can you try your new approach on ''traingle.bmp" and comment on observation?

The encrypted SUTD.bmp image using CBC mode still shows a rough outline of the pattern of the original image. The reason to this might be due to us initialising the Cipher object only once at the start. We should use a random IV for each block by initialising the Cipher again every block. Alternatively, we could have encrypted the whole image rather than go through it per column. We would then be able to get an encrypted image that is totally fuzzy, without any outline (which is especially obvious for the triangle image)

Even if the data were taken from the bottom to the top, the resulting encrypted image will be the same. This is because we are still encrypting the same image but filling each width pixel in a reversed order. The encrypted result of each pixel remains the same since the order of setting the pixel RGB does not matter. As a result, we observe that when data is taken from bottom to top along the columns of the triangle.bmp image, the encrypted result is similar to that we got in Part 2 Question 3.


CBC (bottom to top)
CBC (top to bottom)
Original
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN
Established in collaboration with MIT

Part 3: Signed message digests

Question 1 (7pt): What are the sizes of the message digests that you created for the two different files? Are they the same or different?

They are both size 16 and are the same. A larger file does not give a larger message digest.

A message digest is a hash function that takes in a variable length input and reduces it to a small value (typically 128 to 512 bits). In particular, we are using MD5 which produces a 128-bit digest. This matches the result we got which is 16 bytes (= 128 bits).

Ashlyn Goh 1002840 Cl02

Question 2 (8pt): Compare the sizes of the signed message digests (in byte[] format) for *smallFile.txt* and *largeFile.txt.* Does a larger file size give a longer signed message digest? Why or why not?

They are both of size 128. A larger file size does not give a longer signed message digest.

We initialised the key size to be 1024 in our codes. An RSA signature is a sequence of bytes of the same size of the modulus. Since our key uses a 1024-bit modulus n, the signature value is, numerically, an integer in the 1 to n-1 range, and the PKCS#1 standard specifies that this integer should be encoded as a sequence of bytes of same length as would be needed to encode the modulus (i.e. 128 bytes for a 1024-bit modulus). Hence, the length of signed message digest does not depend on the file size and we get 128 bytes for both files.