50005Lab5

Part 1: `DesSolution.java`

Question 1: 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?

Yes. Shorttext.txt holds song lyrics and and longtext.txt holds the contents of a book.

Question 2: 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?

It is printable but not readable as it is bytecode which results in series of unidentifiable/random symbols and characters being printed to the terminal.

Sample Output from my Terminal: `L�� ��jX┼)�1��Cm'1��)P#┤[L└♂ ����⌟_�PL┬���v��#訪B��↑┴L������r#`

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

Yes, and it is generally readable, however it is encrypted and does not look like the original text but a sequence of random letters put together.

Question 4: Is Base64 encoding a cryptographic operation? Why or why not?

No, although both may utilise the idea of reversible algorithmic operations (like in the case of symmetric keys) encoding is not used for security and does not provide any data confidentiality. It is simply used to format data so that it can be more easily manipulated/transferred etc. A cryptographic operation on the other hand requires some sort of key which changes the nature of the operation and prevents others from reversing the operation without it.

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

Both `shorttext.equals(decipherText1)` and `longtext.equals(decipherText2)` return `true`, thus they are both exactly the same.

Question 6: Compare the lengths of the encryption result (in `byte[]` format) for shorttext.txt and longtext.txt. Does a larger file give a larger encrypted byte array? Why?

Yes. The plaintext itself is being converted to a byte array for encryption, thus every character in the text will need to be converted into bytes and a longer text will consequently result in a larger byte array.

Part 2: `DesImageSolution.java`

Question 1: 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?

The images are very similar in terms of the lines and structure that compose the image, hence it is possible to identify the original image from the encrypted one. However, given a more detailed or real-life picture, this may be more difficult.

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

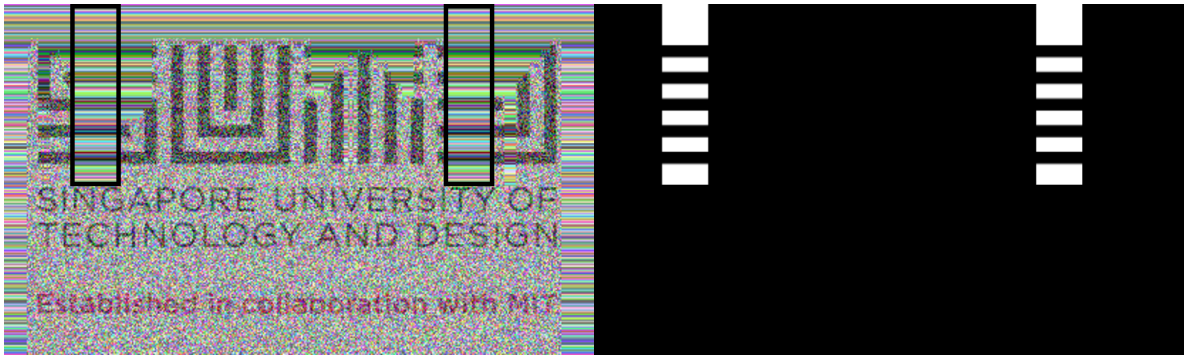
Electronic Codebook (ECB) is the first generation of AES and is fairly basic in that it encrypts blocks one by one without any reliance on other blocks. Hence, it is evident that the overall resulting encrypted data will look somewhat similar to the underlying data, which can manifest as similar visual patterns in the case of encrypted images.

Question 3 : 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.

Compared to EBC, Cipher Block Chaining (CBC) is a more advanced encryption method which relies on all the cipher blocks processed up to that point. In terms of the images, since we are starting from the top, we can see that horizontal lines form on the negative space (whitespace) which reveals the underlying encryption process that the path taken by the algorithm up to that point is roughly the same. This results in a much more 'noisy' result which looks random and makes it much more difficult to derive the unencrypted image from just by looking. However, as the path diverges due to the structure of the image from the top to the bottom, the strength of CBC becomes evident as the image more or less diverges to just pure noise.

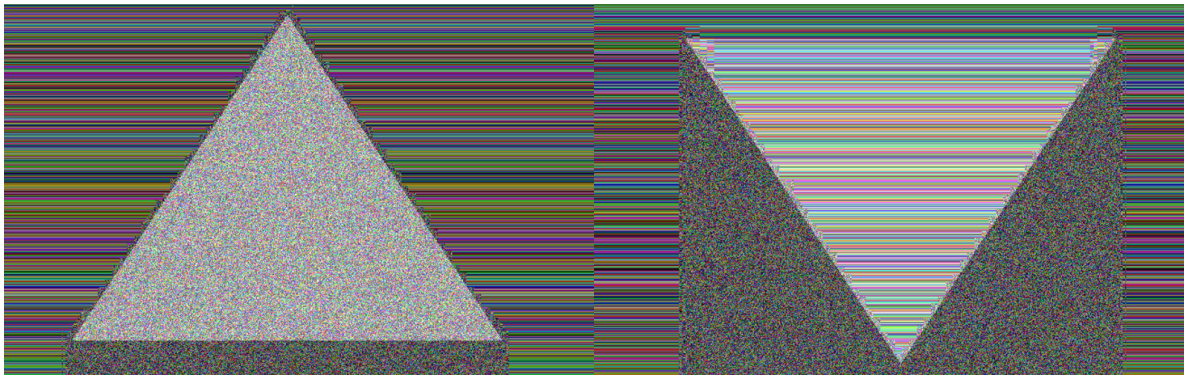
Question 4 : Do you observe any issue with image obtained from CBC mode encryption of “SUTD.bmp”? What is the reason for such observation? Can you explain and try on what would be the result if data were taken from bottom to top along the columns of the image? Can you try your new approach on “triangle.bmp”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

To be more detailed, the reason why we can see vertical lines is because we are processing each column of pixels using the same cipher instance which starts off with a similar initialization vector. This is determined by the generated key used as a parameter into the cipher encryption, resulting in similar encryption outputs from the top to the bottom if path taken is the same. Hence, while CBC is a stronger encryption method, it can still result in some underlying patterns making its way to the final encrypted image. Thus, what seems to be a strange 'tearing' effect for the CBC encrypted SUTD image can actually be explained rather easily if we just study the relationship between the encrypted and original image.



Of course, the algorithm becomes stronger in the case of a non-toy example where noise or greater pixel variation is introduced (such as the 'U' in SUTD - a higher definition image would instead have vertical lines extending into the stylised 'U' negative space).

We can see a stronger and more obvious version of this effect comparing triangle.bmp for both the upright and inverted version. For the upright version, the noise starts once the path hits the border of the triangle from the top (since the paths up to that border on each specific column are of different length) and the same goes for the inverted triangle (however the inside of the triangle is the same since we are still looking at the same path leading **down** to the border with respect to the other lines)



Part 3: `DesSignatureSolution.java`

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

The sizes are exactly the same. They are both strings of 24 characters which compile to a byte array of 16 bytes.

Question 2: Compare the sizes of the signed message digests (in `byte[] encryptedBytes = eCipher.doFinal(data.getBytes());` format) for shorttext.txt and longtext.txt. Does a larger file size give a longer signed message digest? Why or why not?

Larger files/text do not result in a longer message digest as the very definition of a message digest is a fixed numeric representation which is computed by a hash function for the purposes of message verification. Hence, even though shorttext.txt and longtext.txt are both strings of very different lengths, both of them will generate a digest of the same size.