# **1 Introduction**

In this task, a pseudo-AES algorithm will be implemented according to the coursework specification. In addition, this system provides a user interface so that the user can input the data conveniently and easily, and it is clearer for the output presentation. The system consists of 5 modes which are ECB, CBC, OFB, CFB and CTR. Thus, each group member is responsible for a different mode. Accordingly, the report will be organised to describe each mode in different chapter. Each chapter will cover the implementation details and result of testing.

## Test case

Each mode will use the same test case as follow:

The BBC has updated its cookie policy. We use cookies to ensure that we give you the best experience on our website. This includes cookies from third party social media websites if you visit a page which contains embedded content from social media. Such third party cookies may track your use of the BBC website. If you continue without changing your settings, well assume that you are happy to receive all cookies on the BBC website. However, you can change your cookie settings at any time.

* 1. General algorithm description

In this task, the encryption process consists of 5 rounds and each round consists of substitution, permutation and ‘XOR’ with key. Before the first round, the text need to do ‘XOR’ with the initial key, while in the last round, permutation is not needed.

As to ECB and CBC mode, a decryption process is needed, because, for recovering, the ciphertext need to do ‘XOR’, permutation and substitution, which is a reverse process of encryption. Hence, the difficulty of ECB and CBC is keeping the decryption follow the encryption as a reverse process. As for OFB, CFB and CTR, the decryption is the same as encryption, the main difference is the final ‘XOR’ operation takes different input.

# **3 CBC mode**

## 3.1 Algorithm description

In encryption part, the plaintext will first be transformed into bits and divided into several blocks which includes 64 bits of each. Each block presents as an 8x8 matrix. The first block will do ‘XOR’ with Initialization Vector(IV), which the others do with the former ciphertext, which is showed in figure 1. After that, the text will be sent to a repeated process that has 5 rounds. In the first round, the text is needed to do ‘XOR’ with key0 then do substitution, permutation and ‘XOR’ with key1 step by step. In other round, the text from former round does not need to do ‘XOR’ with key0. However, the last round, permutation is not needed.

In substitution, each row consisting of 8 bits is taken and replaced by the value in s-box. The first 4 bits in one row will be x, while the last 4 bits will be y. The value in position (x,y) of s-box will replace the row. As for permutation, each column shifts downward circularly by 0,1…,7.

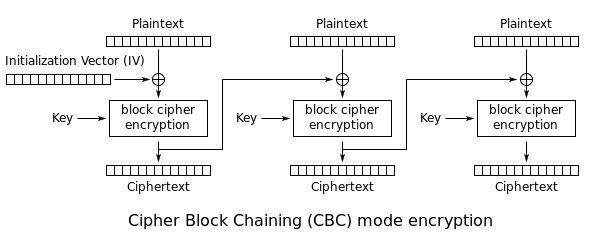


Figure 1 CBC encryption

Decryption part is the inverse process of encryption so that it is similar to implementing encryption. First, each ciphertext block consisting of 64 bits does the reverse operation described in the encryption part.



Figure 2 CBC decryption

## 3.2 Implementation description

* While implementing the algorithm, it is important to keep an eye on the format of text (string, binary, decimal or hexadecimal). In python bin(), int(), hex() and format() functions help to regularise the text. For example, a plaintext in type string can be divided into characters and for each character c, int(c, 16) can transform c into hexadecimal number.
* Considering the number of bits is not always multiple 64, hence, padding 0 is applied in the last block in CBC mode.
* As for substitution, it wastes much time in dividing each row in to ‘x’ and ‘y’ to locate the corresponding value in s-box. Hence, the s-box will be flattened as a one dimension array. Each row in block will be transformed into a decimal number which is the corresponding index in s-box, which is more efficient.
* Moreover, In permutation, a function named roll() in Numpy package in Python will be used to achieve this goal. Thus, no extra memory and variables are needed in this step.
* Furthermore ‘XOR’ operation, the result will be produced by adding each bit of key and corresponding bit of text, then modulo 2. Another efficient way is using ‘^’ operator in Python to achieve ‘XOR’. In CBC mode, the first method is used.
* In encryption process, ASCII is used to present a character, so one character has 8 bits. If using the right keys and IV for decryption, the decrypted text will not appear ‘00000000’ sequence. Hence, in decryption process, if ‘00000000’ is met, the decryption process will be terminated.

## 3.3 Result

When applying the plaintext of text case in 1.1, the result ciphertext in hexadecimal is showed as figure 3. The length of ciphertext keeps the same as that of plaintext in hexadecimal. Figure 4 shows the result of decryption which takes the content in figure 3 as input and recovers the plaintext.



Figure 3 Ciphertext in CBC mode

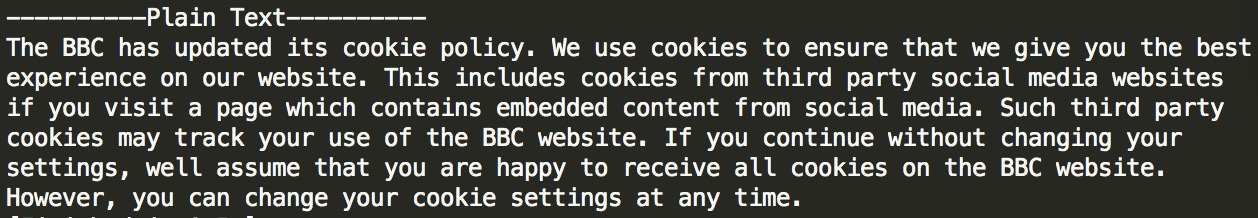


Figure 4 Recovered plaintext in CBC mode

3.4 Performance analysis

Firstly, if a wrong key or wrong IV is applied to the system, the decrypted text is wrong. Figure 5 shows the result that produced by a wrong key. Thus, when applying a wrong key, the whole decrypted text will be changed and no connections to plaintext.

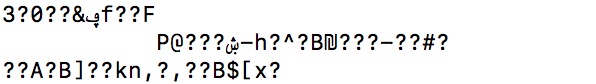


Figure 5 Decryption with wrong key

Time consuming is a vital factor in encryption and decryption processes. The test case in introduction section will be also applied. Figure 6 shows the time-consuming results of two rounds when encrypting the text of test case. Generally, substitution need less time than permutation and ‘XOR’, but sometimes permutation takes longer time than ‘XOR’ sometimes not. The plaintext of test case consists of 62 blocks (3968 bits) will take 0.129347085953 seconds. In decryption process, it takes 0.130770921707 seconds, which is quite close to the time consumed in encryption, simply because decryption process is the reverse process of encryption and no more operations are introduced.

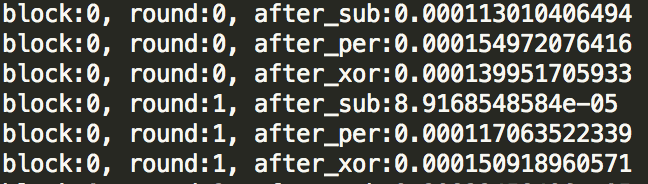


Figure 6 Time-consuming result of encryption

3.5 Security analysis

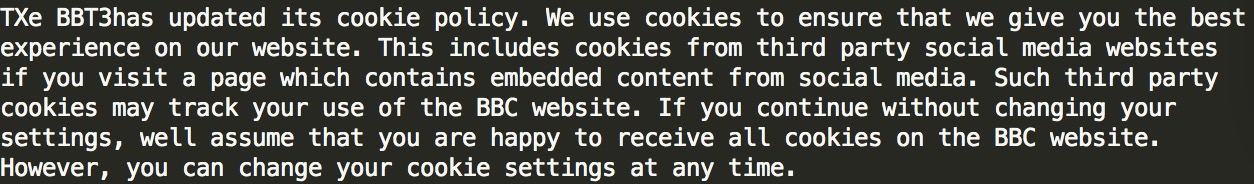
From figure 5, it can be seen that if using the wrong key, although one bit is different, the ciphertext cannot be recognised. However, if the keys leaks, in most situations, the IV cannot protect the plaintext. Figure 7 shows that if IV is not correct, it just influences the decrypted text. However, in some situation, if using a wrong IV which results a ‘00000000’ sequence appearing in advanced, the encrypted process will be terminated and protect the plaintext. 

Figure 7 Influences of using wrong IV