# 6. CTR

## 6.1 Encryption and decryption of CTR

The diagrams of the encryption and decryption of CTR are illustrated below. The keystream is generated by the counter and the key for encrypting the counter. Counter will add one automatically after dealing with the one block of the whole plaintext and the key will always stay same. The ciphertext will be generated by the operation of Xor between the keystream and plaintext. And similarly, plaintext can be recovered by the operation of Xor between the same keystream and ciphertext.

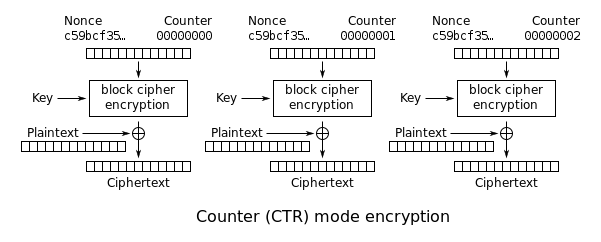


Figure 1. Encryption of CTR

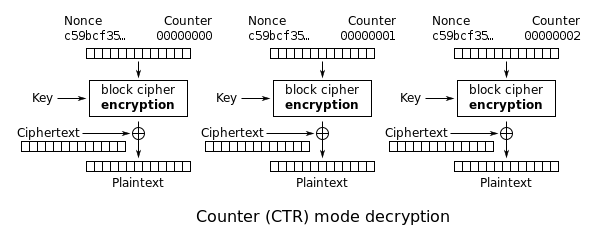


Figure 2. Decryption of CTR

## 6.2 Implementation & Problems

### 6.2.1 Encryption

In the processing of generating ciphertext, the inputs are key, counter and plaintext. The hexadecimal key need to be transformed to the binary key firstly and the hexadecimal counter need to be transformed to binary too for getting the 64 bits to get further encryption. Then, the plaintext need to be imported. Change Every character in the plaintext to ASCII code and then transform ASCII code to binary code. Therefore, every character will have 8 bits binary code. Put 8 characters into a batch, so a plaintext has several batches. If the last batch does not have 8 characters, add the paddings of 0 to make it be the 8 characters. For every batch, substitution, permutation and XOR operations are included to encrypt the keystream. Due to several other parts talk about the three operations, the chapter will not repeat them. After the encryption of the keystream, Xor is applied again between the current batch of the plaintext and the encrypted keystream to get the ciphertext for the current batch. In addition, one of most special place in the CTR mode is to add one for counter at the end of one iteration of the batch. So, for next batch, the counter will change 1. After the encryptions to all batches of plaintext, a list will output all the ciphertext generated by each batch.

### 6.2.2 Decryption

In the processing of decryption, ciphertext, counter and key will be inputted. Ciphertext is generated by above processing. Counter and key are same as the processing of generating ciphertext. Take the length of the ciphertext as new batch number and for every batch, do the same operations (substitution, permutation, Xor) to get the keystream which actually is the same as the one in previous processing of generating ciphertext but for better independence, I do it again. Finally, each batch of ciphertext Xor the keystream generated by corresponding batch of the ciphertext to get the original plaintext.

### 6.2.3 Problems

One of the biggest problems to implement the algorithm mentioned below is transformation between binary, hexadecimal and decimal. In addition, the size of a

list always need to be changed to meet the requirement of the operation. For example, size of some lists need to be 8\*8 but sometime, the 8\*8 list also need to be transformed to the string or other lists which have different size for display or other function. Sometimes, I will be dizzy about which size of the list now or whether it is binary. It is the most headache that I encounter and by solving it, I always debug to see the size of the variables, which is quite helpful but time-consuming too.

## 6.3 Test

The plaintext is showed as figure 3 and corresponding ciphertext generated by algorithm is showed as figure 4. The ciphertext is used to recover the plaintext and the truth is that the recovered string is exactly same as the plaintext in figure 3. Therefore, it can be seen good to go through the test.

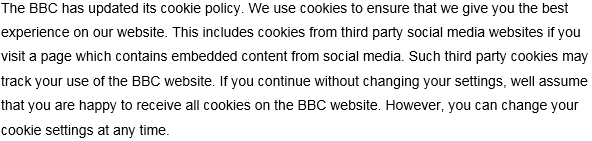


Figure 3. Text plaintext



Figure 4 ciphertext

## 6.4 Performance Discussion