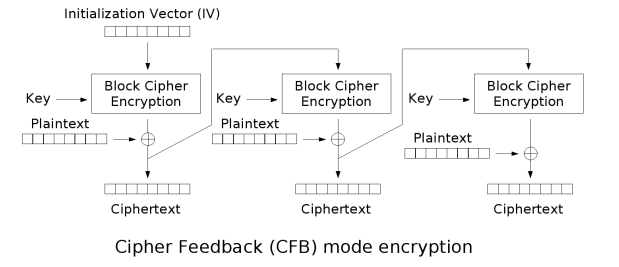
## CFB：

This part will explain and implement the CFB model. It is a pattern that transforms block ciphers into streams and hide plaintexts timely to transfer data less than packet. But it still has some deficiencies, first is it not conductive to parallel computing, the second one is one plaintext cell damage affects multiple cells, what’s more, IV should to be designated as unique.

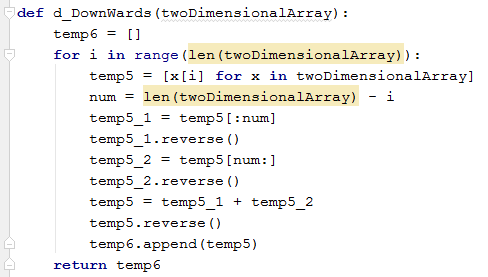
Encryption:



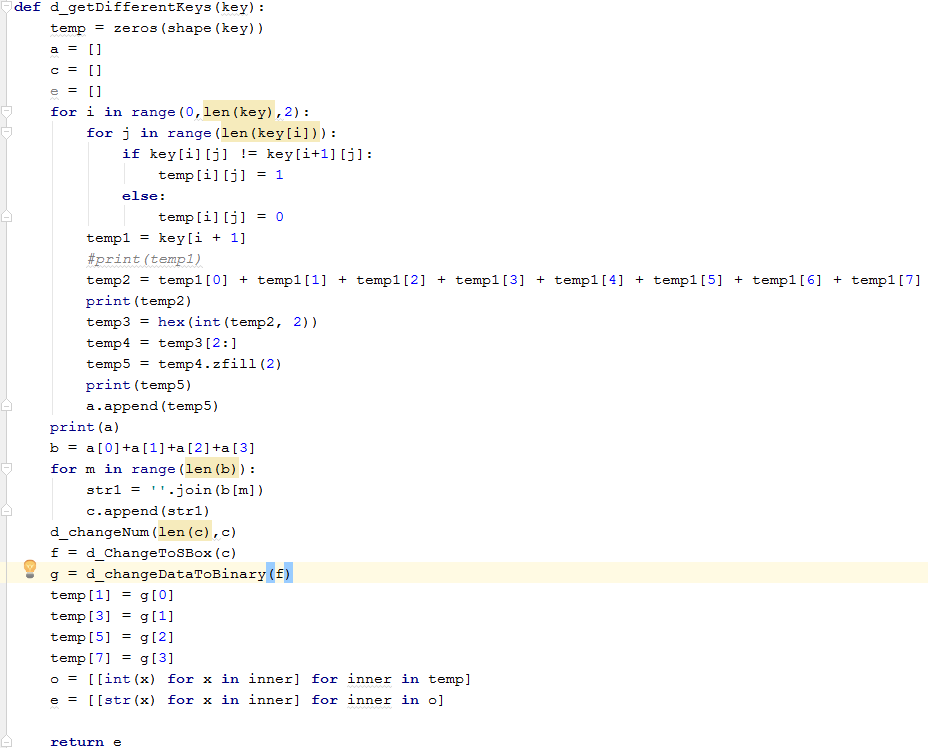
CFB encryption is divided into two parts. The one is encrypt the ciphertext obtained from the previous encryption step, the second one is XOR the encryped data with the current paragraph of the plaintext.

Here IV is a piece of data used for randomization and encryption. Therefore, different ciphertexts can be generated from the same plaintext and the same key without re-generating the key, avoiding a rather complicated process.

The first step of encryption is Substitution: IV need to be converted into hexadecimal value and compare with s-Box to get the corresponding value. The result of this step is an 8 \* 8 binary matrix. The second step is Permutation: For this step matrix will be transformed into columns, each column will be followed by a downward rotation (the zero column unchanged, the first column moves one. . . The seventh column moves seven). The code is as follows:

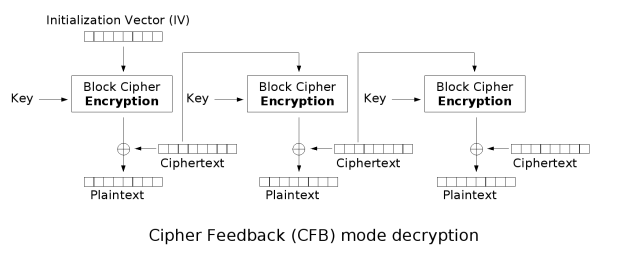


The third step is compute the XOR between the state and the key: The above three basic operations will be done five times and then get the value of k1.The usefulness of getting k1 is to XOR k1 with the first paragraph of the plaintext. It is worth mentioning that, In the above five rounds of operation, the key value of each round is not the same, a new round of key value determined by the previous round. Each line of new 8 \* 8key matrix will look like this: b0 XOR b1, Sbox(b1), b2 XOR b3, Sbox(b3), b4 XOR b5, Sbox(b5) b6 XOR b7, Sbox(b7). In order to achieve this goal, I create the python function show as below:



Next step is compute the bitwise XOR between the first block of plaintext and the k1: The result is an 8 \* 8 matrix, called m1. The next thing is to make m1 as a new round of IV, do all of the above operations and use the second paragraph plaintext XOR with the result to get m2. Loop this operation until all the plaintext are all encrypted.

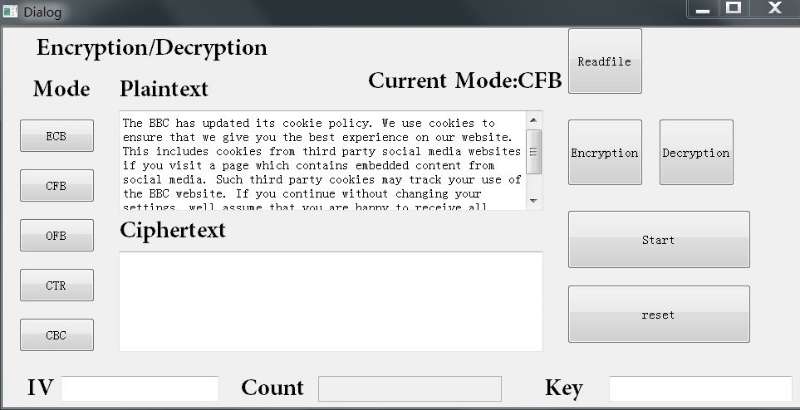
Decryption:



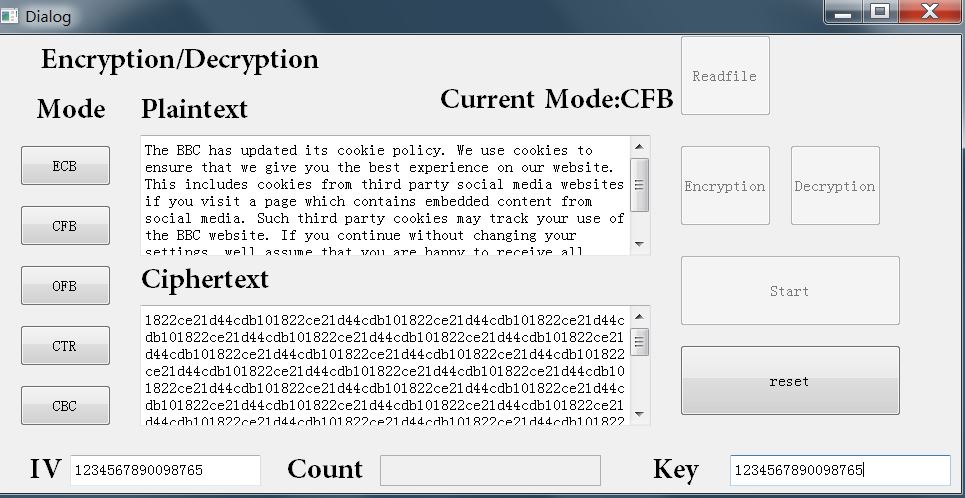
The decryption process of CFB mode is very simple, since the cipher block here is a self-synchronizing stream cipher, each section of the encrypted plaintext can be obtained directly by doing the XOR operation between the encrypted plaintext block and the ciphertext. Then, we should spliced the plaintext in a reverse direction. Since the data encrypted by the block encryptor in the encryption flow and the decryption flow is the previous one, the data length is not needed even if the length of the plaintext data is not an integral multiple of the encrypted block size, which ensures that the data length is the same before and after encryption .

Test and Evaluate:

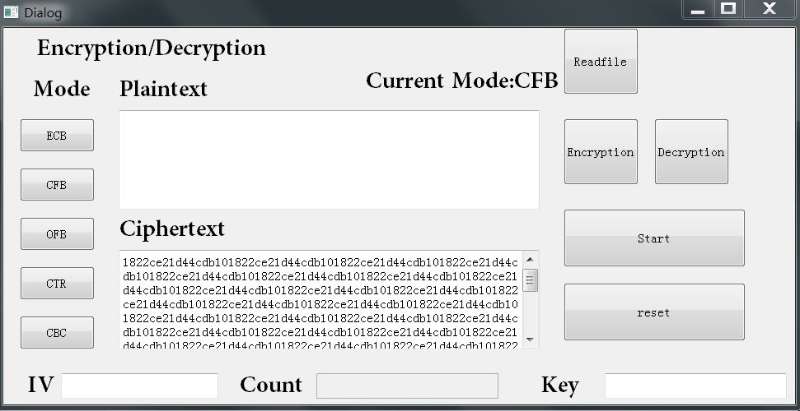
Encryption and decryption operation shown to the user by GUI. Here I input the test IV, plaintext and key into the interface. The test text is a long piece of text. The result is as follows:



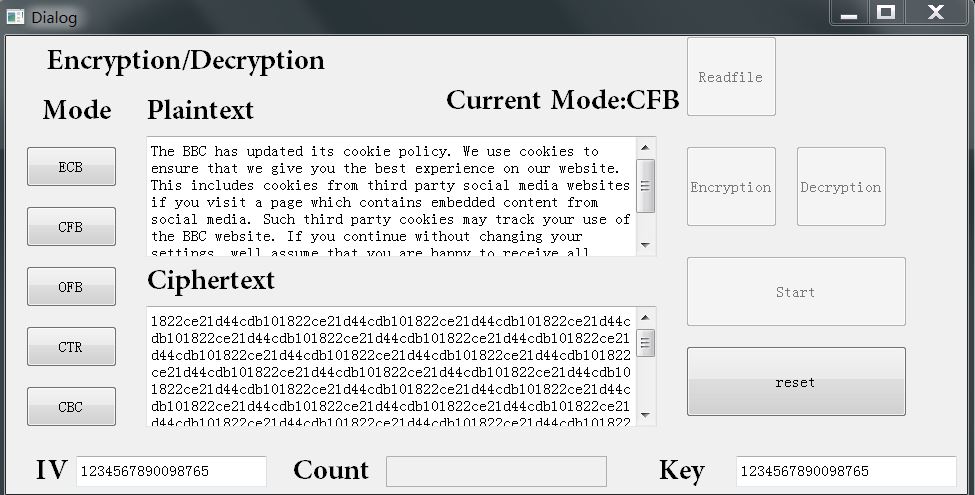
**Input the plaintext**



***Get the ciphertext***



**Input the ciphertex**



**Get the plaintext**

After that I have tested short text and numerical text and All the test results have proved to be correct. This model is completed. In conclusion, CFB is to encrypt first block at first, and then the result obtained is combined with plaintext to generate the current block, thereby effectively changing the key used to encrypt the current block.