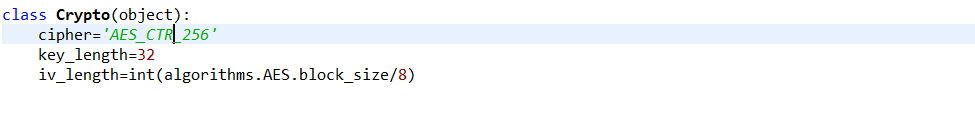
# Weekly goal

1Fix the md5 problem on the server client.

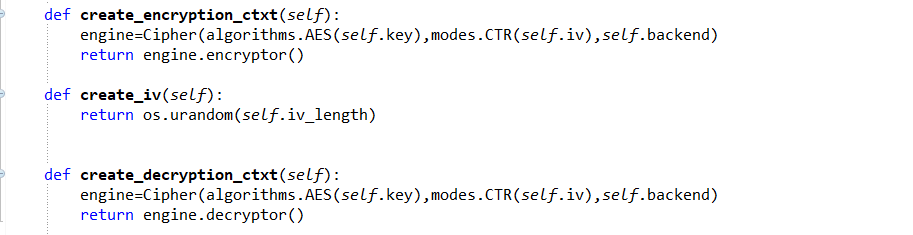
2Implement the encryption method.

## 1.Successfully implement the encryption method

The encryption algorithm is AES in CTR mode.



…………………………………………



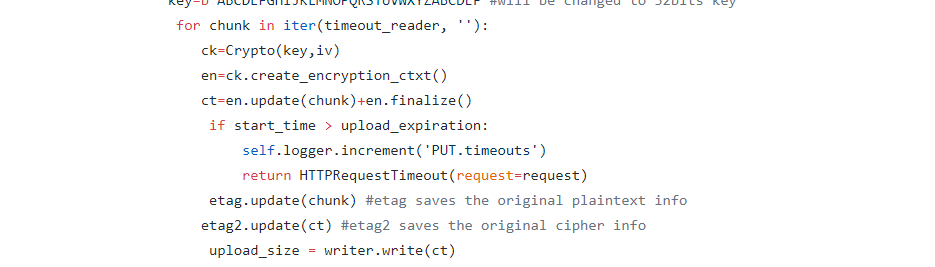
By disabling the decryption method when download, we will get a file encrypted by AES.

Then enabling the decryption method, we will get a file as the same as the original one. So it means the encryption method has been successfully implemented in this program.

## 2.Fix the md5 mismatch problem

The md5 problem is caused by wrong calculating order. The client compared the md5 of plaintext to the md5 of cipher text, which leaded the mismatch problem.

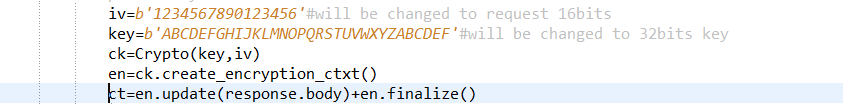
At first, I tried to solve the problem by calculating the md5 first then executing encryption code. So the md5 will be the hash of cipher text. However, this will raise another problem. The server will check the md5 and compare it to the file in storage node. Certainly, they are not equal, for the file in storage is already encrypted.



So at last I managed to solve it by calculating two md5 in uploading process. The code will send back the plaintext md5 to the client as a response, then save the ciphertext md5 in metadata in storage node, which solves two problems in one time.

## Next step

For now, the IV and KEY I use is initialized in script. So the next question is how to transform the IV (in plaintext?) and where to store the KEY. (Maybe they should be stored in a database?)



I think we can write the “IV” in the metadata list of a file like “metadata[‘IV’]=’xxxx’”. So when first time upload the file, the file will bring the IV with it all the time.

As for KEY storage, if using database, we can easily divide files into groups, and each group can have a key to it. But a new problem may raise. How to insure the security of the database?

## Problem needed to be fixed

Decryption error when downloading big-size file

In upload process the file is transforming in chunks (one chunk is 65536 bytes at most). So a file is encrypted chunk by chunk. But in download process, the server sends the file as a whole rather than by chunk. When downloaded, the file is decrypted in whole, which will cause problems when downloading file bigger than 65536 bytes(64K).

It’s a bit wired for this problem to occur…. For as far as I know, the AES will handle a block of 256bits(or 128) as a block in one time. No matter 128 or 256b, they are divisors of 65536B(a chunk size),so I am not sure how the problem occurs. Maybe this AES algorithm I am using is adjusted like changing the order of blocks?

Possible solution:

Because in write method, swift limit the chunk\_size to at most 65536 bytes, so it’s hard to solve the problem in upload process. Maybe it’s possible if we cut the file into chunks(65536B) when downloaded as well and decrypted chunk by chunk.(But we can’t be sure the chunk is 65536 bytes…)

## A bug found

If the uploaded file’s md5 not equal to the original one, the upload process won’t be revoked and the wrong file will exist in cloud storage. And if a new file shares the same name as the original one, after uploading, the previous file will be deleted without warning.