[Crypto] Flippin Bank

So, this was an easy crypto challenge even without the source code, you can deduce what you need to do from the challenge name. Anyways, we start by looking on the source code, and we see that to get the flag you need to provide ***username: admin, password: g0ld3n_b0y***

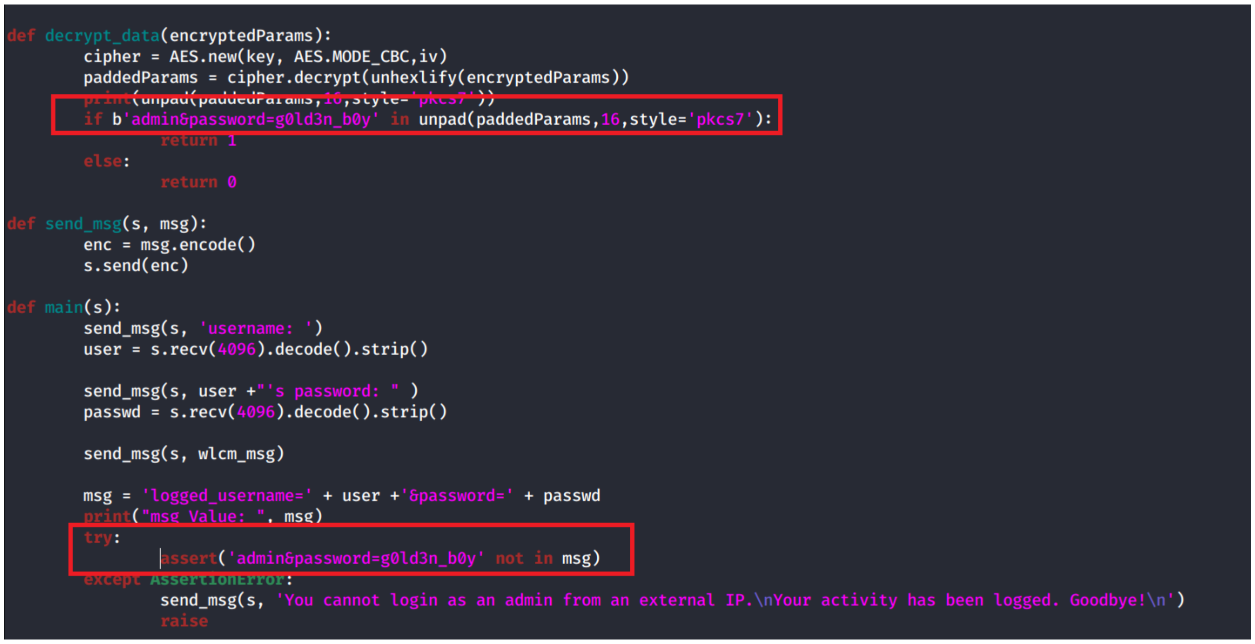
The server generates a message:

> ***msg = 'logged_username=' + user +'&password=' + passwd***

and then encrypts this message and sends you the cipher text and when you send it back again it decrypts it and checks to see if the string

admin,&password=g0ld3n_b0y is in the decrypted text, if so it'll generate the flag

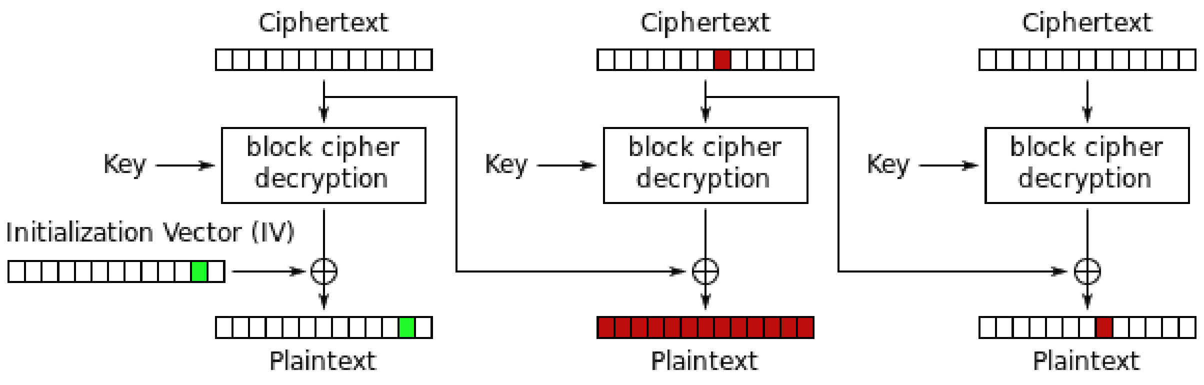
But because of the assert statement you cannot give the server the inputs required to produce the flag.



```
def decrypt_data(encryptedParams):
        cipher = AES.new(key, AES.MODE_CBC,iv)
        paddedParams = cipher.decrypt(unhexlify(encryptedParams))
        print(unpad(paddedParams,16,style='pkcs7'))
        if b'admin&password=g0ld3n_b0y' in unpad(paddedParams,16,style='pkcs7'):
                return 1
        else:
                return 0

def send_msg(s, msg):
        enc = msg.encode()
        s.send(enc)

def main(s):
        send_msg(s, 'username: ')
        user = s.recv(4096).decode().strip()

        send_msg(s, user +"'s password: " )
        passwd = s.recv(4096).decode().strip()

        send_msg(s, wlcm_msg)

        msg = 'logged_username=' + user +'&password=' + passwd
        print("msg Value: ", msg)
        try:
                assert('admin&password=g0ld3n_b0y' not in msg)
        except AssertionError:
                send_msg(s, 'You cannot login as an admin from an external IP.\nYour activity has been logged. Goodbye!\n')
                raise
```

We can see there is two function encrypt_data() and decrypt_data(). encrypt_data() take a message, randomly generated key and iv and encrypt it. In CBC encryption and decryption for first block there is no previous block so at that time iv is used.

Let us review how the CBC decryption process work:



Cipher Block Chaining (CBC) mode decryption

The image above illustrates the decryption process, notice that if we changed a byte in the cipher text then the entire plain text block generated from that cipher block will be corrupted.

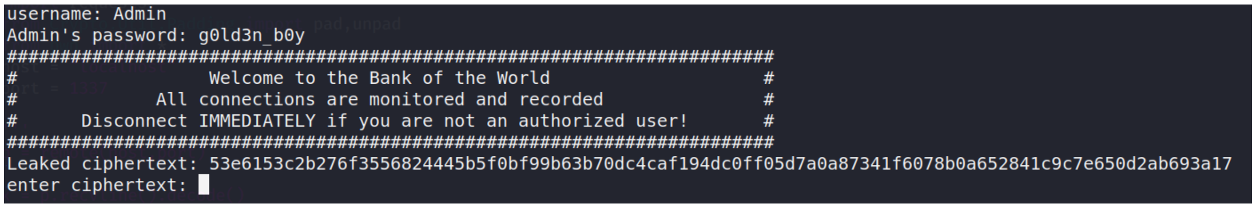
But still in the next block of plain text, the only thing that will change is the exact byte corresponding to the byte we changed before.

Let me explain first how we are going to approach this. So, if we gave the server username: ***Admin*** instead of ***admin*** that should bypass the assertion but will fail in the if statement in the decryption function. But if we managed to find a way to change the cipher text so that when it gets decrypted it will change the “A” in Admin to “a” that should do the trick.

Now let’s begin:

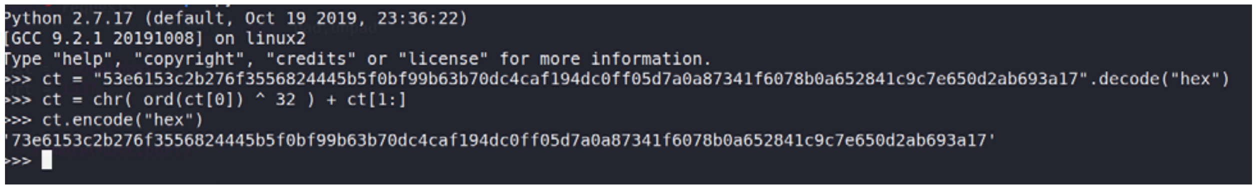
Message: logged_username=Admin&password=g0ld3n_b0y

Since this message get’s divided into blocks of 16, the “A” character will be the first in the second block, so we need to flip the first byte in the first block (XOR it with 32) to transform the “A” to “a”



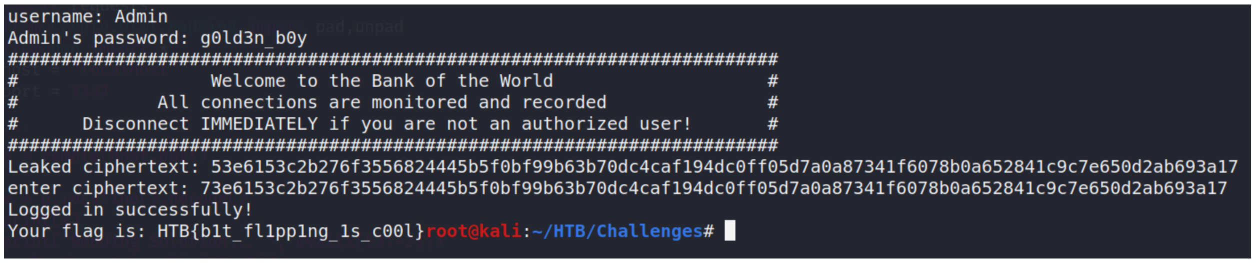
```
username: Admin
Admin's password: g0ld3n_b0y
########################################################################
#                  Welcome to the Bank of the World                    #
#             All connections are monitored and recorded               #
#      Disconnect IMMEDIATELY if you are not an authorized user!       #
########################################################################
Leaked ciphertext: 53e6153c2b276f3556824445b5f0bf99b63b70dc4caf194dc0ff05d7a0a87341f6078b0a652841c9c7e650d2ab693a17
enter ciphertext: 
```

So, we take this cipher text and perform the following



```
Python 2.7.17 (default, Oct 19 2019, 23:36:22)
[GCC 9.2.1 20191008] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ct = "53e6153c2b276f3556824445b5f0bf99b63b70dc4caf194dc0ff05d7a0a87341f6078b0a652841c9c7e650d2ab693a17".decode("hex")
>>> ct = chr( ord(ct[0]) ^ 32 ) + ct[1:]
>>> ct.encode("hex")
'73e6153c2b276f3556824445b5f0bf99b63b70dc4caf194dc0ff05d7a0a87341f6078b0a652841c9c7e650d2ab693a17'
>>> 
```

Now we take that new cipher text and send it And....



```
username: Admin
Admin's password: g0ld3n_b0y
########################################################################
#                  Welcome to the Bank of the World                    #
#             All connections are monitored and recorded               #
#      Disconnect IMMEDIATELY if you are not an authorized user!       #
########################################################################
Leaked ciphertext: 53e6153c2b276f3556824445b5f0bf99b63b70dc4caf194dc0ff05d7a0a87341f6078b0a652841c9c7e650d2ab693a17
enter ciphertext: 73e6153c2b276f3556824445b5f0bf99b63b70dc4caf194dc0ff05d7a0a87341f6078b0a652841c9c7e650d2ab693a17
Logged in successfully!
Your flag is: HTB{b1t_fl1pp1ng_1s_c00l}root@kali:~/HTB/Challenges# 
```

Flag: ***HTB{b1t_fl1pp1ng_1s_c00l}***