get_cookie returns a cipher text of plain text containing *"admin=False"*, notice that the IV is returned along with the cipher text, the first 16 bytes of the cipher text are IV used for encryption. In *check_admin* the input cipher text is decrypted with giving IV, it returns the flag if the decrypted text include string *"admin=True"*.

```
def check_admin(cookie, iv):
    cookie = bytes.fromhex(cookie)
    iv = bytes.fromhex(iv)

    try:
        cipher = AES.new(KEY, AES.MODE_CBC, iv)
        decrypted = cipher.decrypt(cookie)
        unpadded = unpad(decrypted, 16)
    except ValueError as e:
        return {"error": str(e)}

    if b"admin=True" in unpadded.split(b";"):
        return {"flag": FLAG}
    else:
        return {"error": "Only admin can read the flag"}


def get_cookie():
    expires_at = (datetime.today() + timedelta(days=1)).strftime("%s"
    cookie = f"admin=False;expiry={expires_at}".encode()

    iv = os.urandom(16)
    padded = pad(cookie, 16)
    cipher = AES.new(KEY, AES.MODE_CBC, iv)
    encrypted = cipher.encrypt(padded)
    ciphertext = iv.hex() + encrypted.hex()
```

```
        return {"cookie": ciphertext}
```

The encryption and decryption were both based on AES-CBC, decryption process is as follow, `pn` is the nth block of plain text, `cn` is the nth block of cipher text, `d()` is the decryption function.

```
p1  = c0 ^ d(c1)
=> d(c1) = p1 ^ c0
```

To make the server believe admin is requesting, needed to flip cipher text and IV so that the decrypted text contains *"admin=True"*. In the equation shown below, `pn'` is the plain text server should read, replaced `d()` according to equation above, .

```
p1' = c0' ^ d(c1)
    = c0' ^ p1 ^ c0
```

```
cipher ^ iv = plain
cipher = plain ^ iv
fake = cipher ^ iv'
=> iv' = fake ^ cipher = fake ^ plain ^ iv
```

After successfully decrypted, the server check splitted message if containing item *"admin=True"*, so the fake message should be *";admin=True;"*, the part to be changed was *"admin=False"*, located in the first block. Here is the flipping process.

```
def flip(cookie, plain):
    start = plain.find(b'admin=False')
    cookie = bytes.fromhex(cookie)
    iv = [0xff]*16
    cipher_fake = list(cookie)
    fake = b';admin=True;'
    for i in range(len(fake)):
       cipher_fake[16+i] = plain[16+i] ^ cookie[16+i] ^ fake[i]
       iv[start+i] = plain[start+i] ^ cookie[start+i] ^ fake[i]

    cipher_fake = bytes(cipher_fake).hex()
    iv = bytes(iv).hex()
    return cipher_fake, iv
```

There was one thing unknown here, the second block of original plain text containing an expiry, it could be different in every cookie genereated, that would impact the flipping result, so needed to made up a plain text that has expiry close enough to the one geneated by server, otherwise, there might be an padding issue.

```
expires_at = (datetime.today() + timedelta(days=1)).strftime("%s")
plain = f"admin=False;expiry={expires_at}".encode()
cookie = request_cookie()
cookie, iv = flip(cookie, plain)
print(request_check_admin(cookie, iv))
```

On success, the server responses the flag.

```
{'flag': 'crypto{4u7h3n71c4710n_15_3553n714l}'}
```

The full code is here.