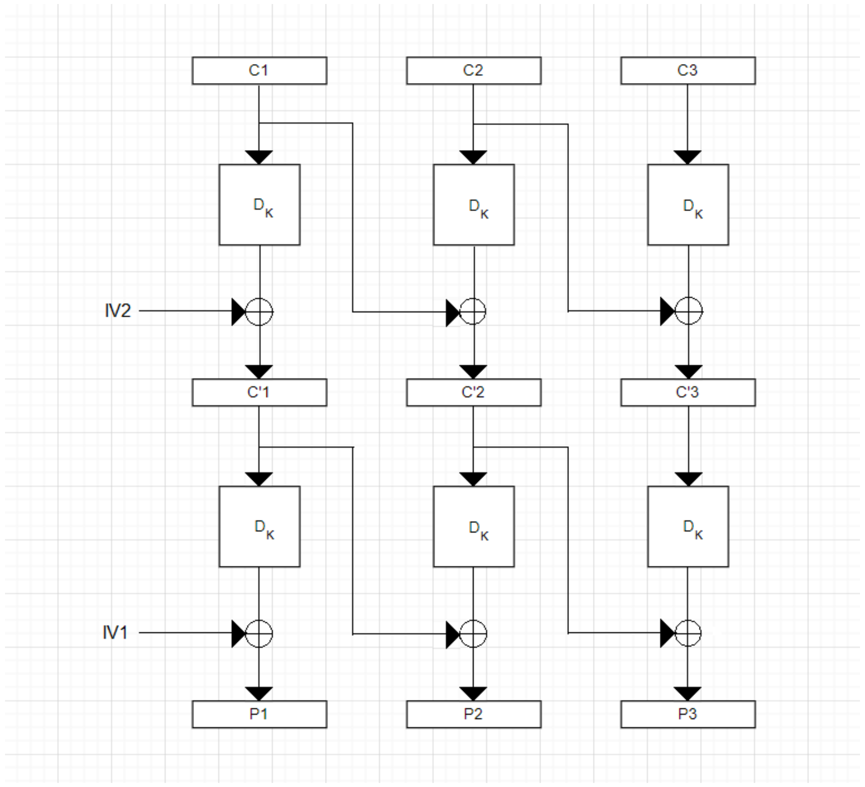
# CBCBC

```
def encrypt(msg):
    aes1 = AES.new(key, AES.MODE_CBC, iv1)
    aes2 = AES.new(key, AES.MODE_CBC, iv2)
    enc = aes2.encrypt(aes1.encrypt(pad(msg, 16)))
    return iv1 + iv2 + enc


def decrypt(msg):
    iv1, iv2, enc = msg[:16], msg[16:32], msg[32:]
    aes1 = AES.new(key, AES.MODE_CBC, iv1)
    aes2 = AES.new(key, AES.MODE_CBC, iv2)
    msg = unpad(aes1.decrypt(aes2.decrypt(enc)), 16)
    return msg


def create_user():
    username = input("Your username: ")
    if username:
        data = {"username": username, "is_admin": False}
    else:
        # Default token
        data = {"username": hidden_username, "is_admin": True}
    token = encrypt(json.dumps(data).encode())
    print("Your token: ")
  print(base64.b64encode(token).decode())
```

```
def login():
    username = input("Your username: ")
    token = input("Your token: ").encode()
    try:
        data_raw = decrypt(base64.b64decode(token))
    except:
        print("Failed to login! Check your token again")
        return None

    try:
        data = json.loads(data_raw.decode())
    except:
        print("Failed to login! Your token is malformed")
        return None

    if "username" not in data or data["username"] != username:
        print("Failed to login! Check your username again")
        return None

    return data
```



Server creates a token for each username in the form of encrypted json string. When login, user has to know his token and the username coresponding to that token. Consider when creating new user with empty string, we can receive an admin-role token by default. Admin token has 80 bytes in length, that means 5 AES blocks, two for IVs and three for ciphertext, and we also know admin’s name length is not greater than 13 bytes.

We need to know his name to get the flag, by using padding oracle attack with two different responses: “Your token is malformed” and “Check your token again”.

To recover the first plaintext block P1, we can modify IV1 as normal, check for correct padding. For the second block, things become more challenging, we need to know C’1 to recover P2, padding oracle attack on C2 can only recover D(C’2).

Fortunately, P3 can be guessed due to limited name length. Apply padding oracle attack on C3, we have D(C’3), 13 possibilities for P3 means 13 possibilities for C’2, same for D(C2).

C’1 = IV2 XOR D(C1), assume we know D(C2), we can send modified-IV || D(C2) XOR C1 || C2 to perform padding oracle attack and recover D(C1), finally: P2 = D(C’2) XOR C’1 = D(C’2) XOR IV2 XOR D(C1).

Summary, first we have to find P1 by using padding oracle attack. Then, find D(C’3) and D(C’2) with the same method.

P3 = D(C’3) XOR C’2, or P3 = D(C’3) XOR C1 XOR D(C2), try to find 13 possibilities of D(C2).

Next, we send modified-IV || D(C2) XOR C1 || C2 token for each possible value of D(C2) to find D(C1),

then C’1 = D(C1) XOR IV2.

Finally, P2 = C’1 XOR D(C’2).