CS 370
Introduction to Security

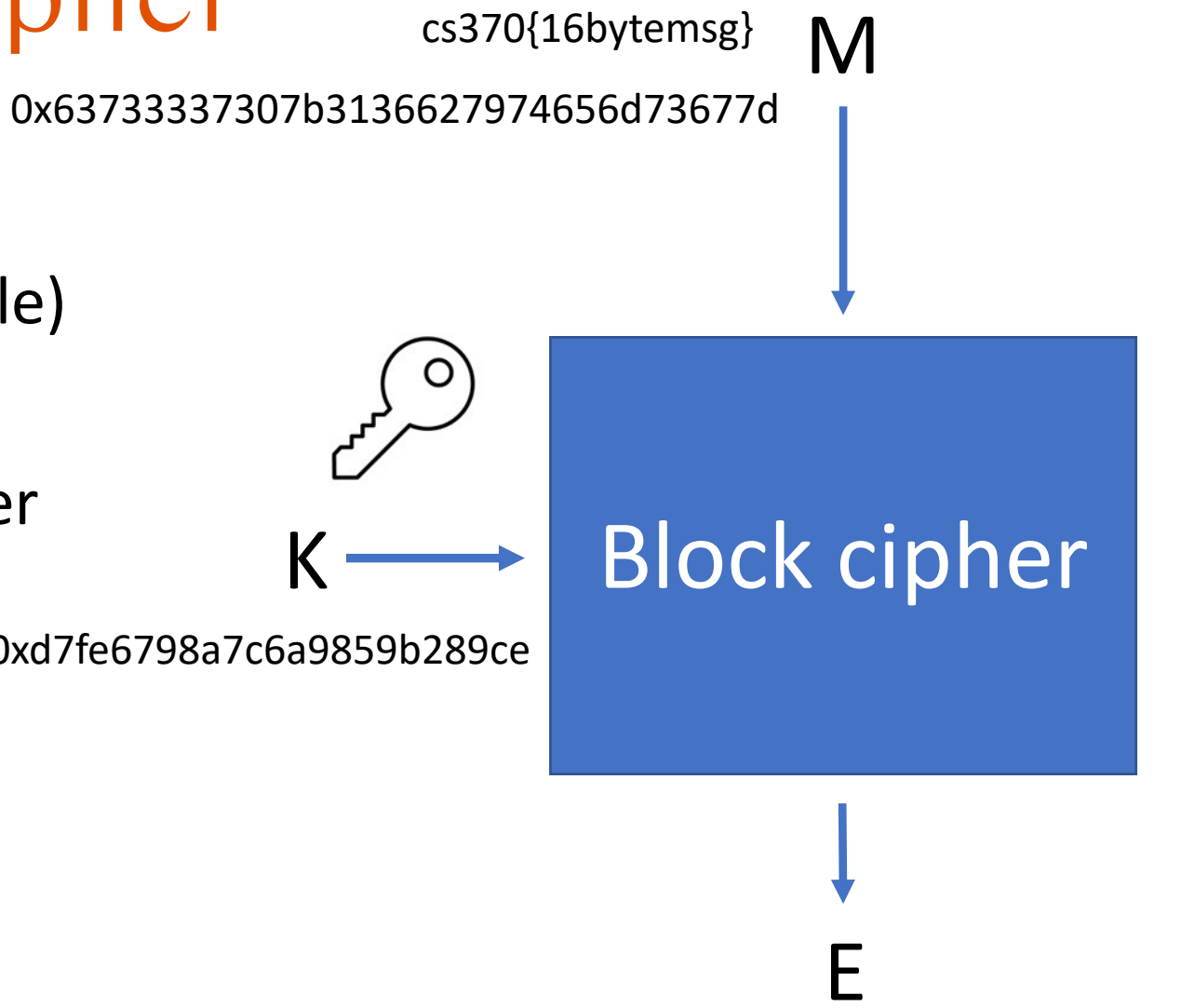
Block Cipher Modes

Yeongjin Jang

Oregon State University

Recap: Block Cipher

- Block: a fixed sized message (16-byte here for the example)
- Key: A secret between sender and receiver (short, 16-byte here for the example)
- M: Plaintext message
- E : Encrypted message



Recap: What Does the Block Cipher Do?

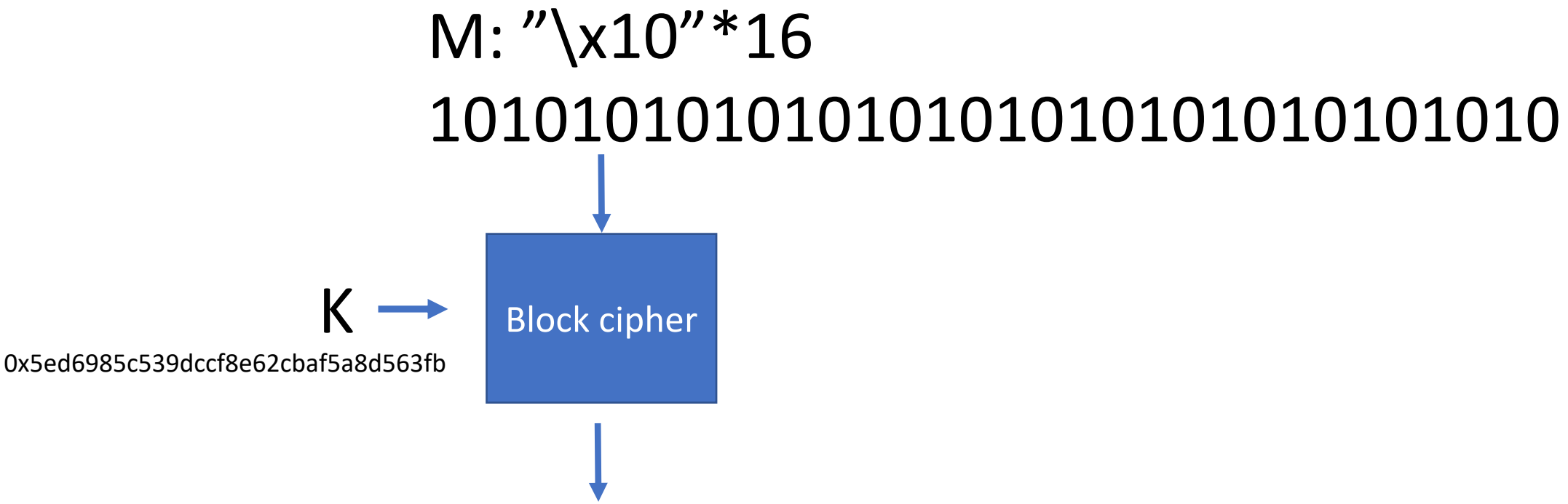
- Generating a permutation of the numbers in
  - $\{0,1\}^n$ -> $\{0,1\}^n$
  - It's permutation, so it must be one-to-one mapping

- It's like shuffling the card

msg	Ciphertext
0	0xaf531b0e1
1	0x14a986e7a
2	0xad738009d
3	0x5ed6985c5
4	0xf3b8aa2e8
5	0xad04ec00e
…	0x59fd94c21

Recap: Can encrypt arbitrary bytes..

- “0123456789ABCDE0x01” (16-byte) and “\x10\x10\x10…” (16-byte)



Recap: ECB is Insecure..

- Please take a look at the ECB encrypted Image
- https://cs370.unexploitable.systems/_static/encrypted.bmp

ECB is Insecure

20

I have securely encrypted the BMP (bitmap) image file that contains the flag to this challenge.

I used AES-ECB-128 with a super secure random key, so I bet you cannot get the flag.

The image is at here:
https://cs370.unexploitable.systems/_static/encrypted.bmp

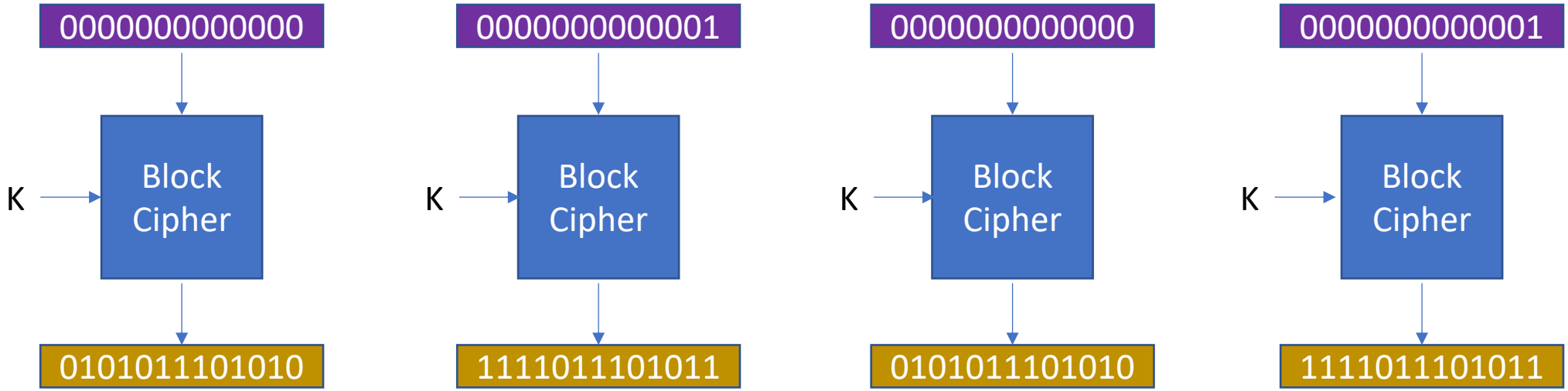
It starts with CS370{ but I do not want to let you know the rest.

Block Cipher Modes

- A way of using block cipher
- A block cipher can only encrypt/decrypt a block
- How can we chain this and use it to encrypt
  - Arbitrary length of message?
- ECB / CBC / CTR

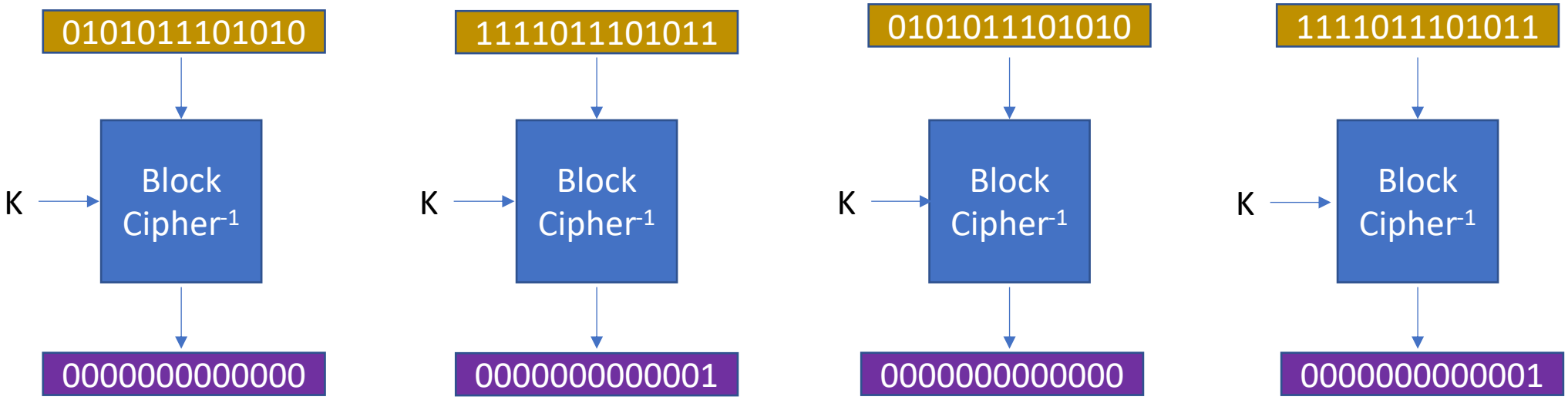
Electronic Code Book (ECB)

- Use Block Cipher Encryption as Encryption



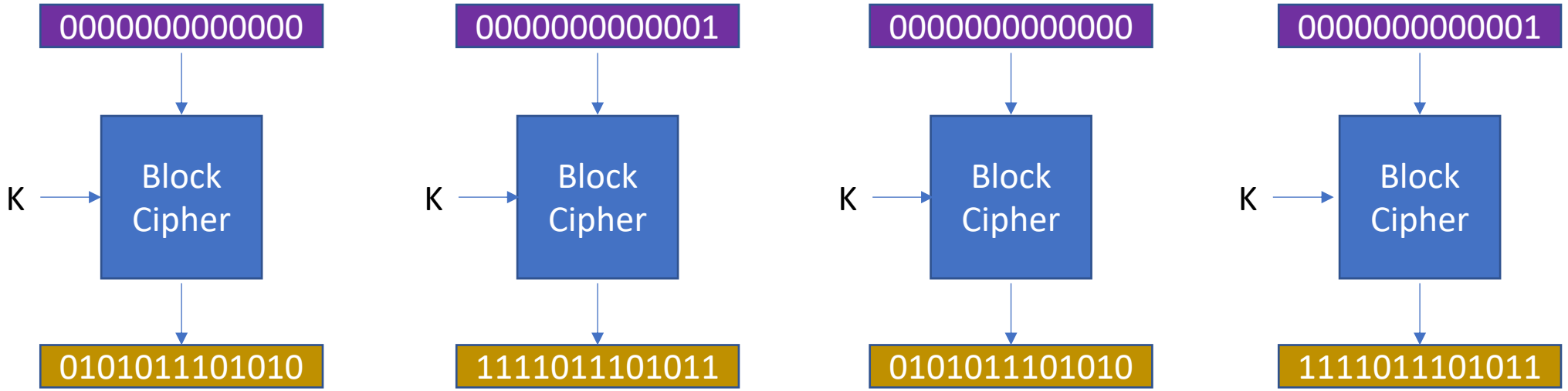
Electronic Code Book (ECB)

- Use Block Cipher Decryption as Decryption



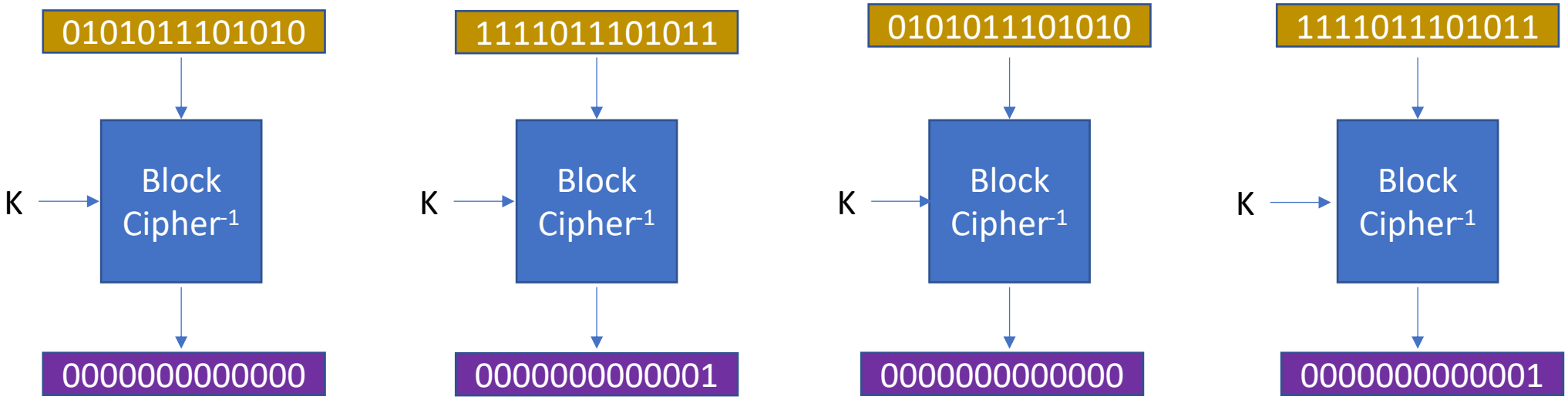
ECB

- We can run encryption in parallel



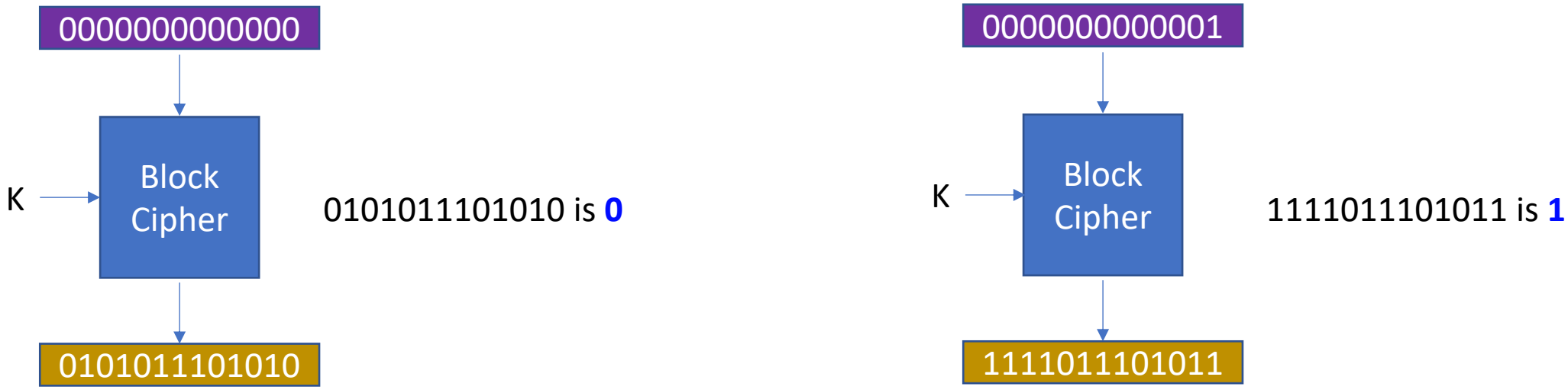
ECB

- We can run decryption in parallel

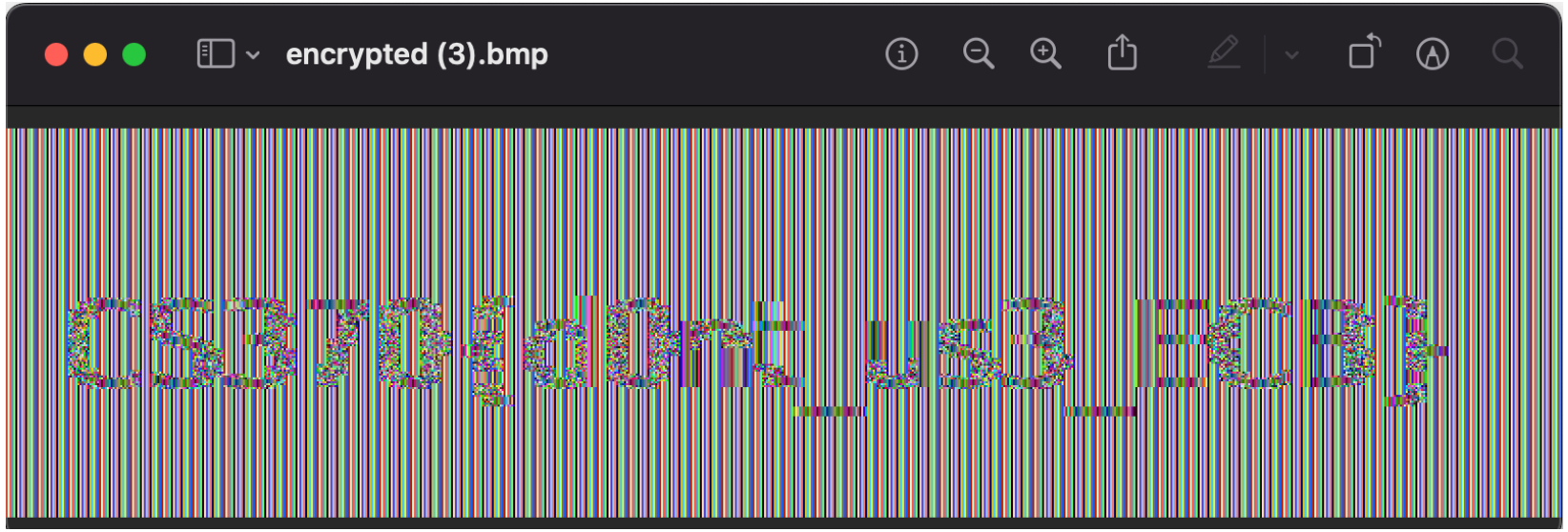


ECB Weakness

- A plaintext block will be encrypted to a specific ciphertext block



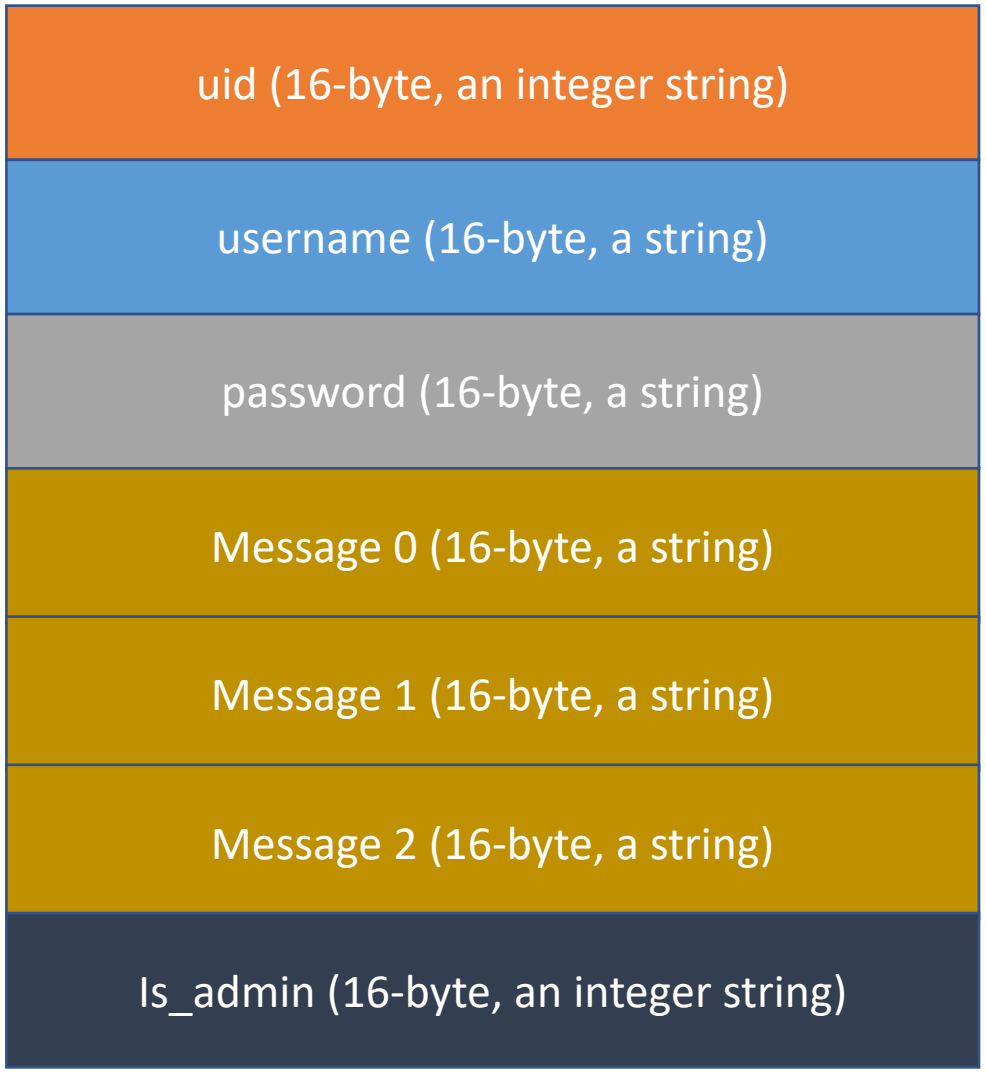
ECB-is-insecure



ecb-attack

- You need to exploit the weakness of ECB mode to get 3 flags

```
blue9057@blue9057-vm-ctf1 ~/crypto/ecb-attack <ruby-head>
$ ./launcher
Running Command: [/home/labs/crypto/ecb-attack/ecb-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: encrypted.user
Decrypted user information:
       uid      : 1
  username      : 'notadministrator'
  password      : 'passwordpassword'
   message      : 'The sky is blue, cs370 crypto is a boring class.'
  is_admin      : 0
```



ecb-attack

- How to start?
  - fetch crypto
- Run ./launcher
  - Type ‘encrypted.user’

```
┌─blue9057@blue9057-vm-ctf1 ~/crypto/ecb-attack <ruby-head>
└─$ ./launcher
Running Command: [/home/labs/crypto/ecb-attack/ecb-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: encrypted.user
Decrypted user information:
       uid      : 1
  username      : 'notadministrator'
  password      : 'passwordpassword'
   message      : 'The sky is blue, cs370 crypto is a boring class.'
  is_admin      : 0
```

ecb-attack

- Encrypted.user
  - The example encrypted user

```
blue9057@blue9057-vm-ctf1 ~/crypto/ecb-attack <ruby-head>
$ ./launcher
Running Command: [/home/labs/crypto/ecb-attack/ecb-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: encrypted.user
Decrypted user information:
      uid      : 1
 username      : 'notadministrator'
 password      : 'passwordpassword'
  message      : 'The sky is blue, cs370 crypto is a boring class.'
 is_admin      : 0
```

ecb-attack

- Create valid encrypted data…
  - uid == 0
  - is_admin == 1
  - password to something else

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

ecb-attack

- Making uid == 0

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

```
blue9057@blue9057-vm-ctf1 ~/crypto/ecb-attack <ruby-head>
$ ./launcher
Running Command: [/home/labs/crypto/ecb-attack/ecb-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: encrypted.user
Decrypted user information:
      uid      : 1
 username      : 'notadministrator'
 password      : 'passwordpassword'
  message      : 'The sky is blue, cs370 crypto is a boring class.'
 is_admin      : 0
```

ecb-attack

- Making uid == 0

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

is_admin = 0

ecb-attack

- Making uid == 0

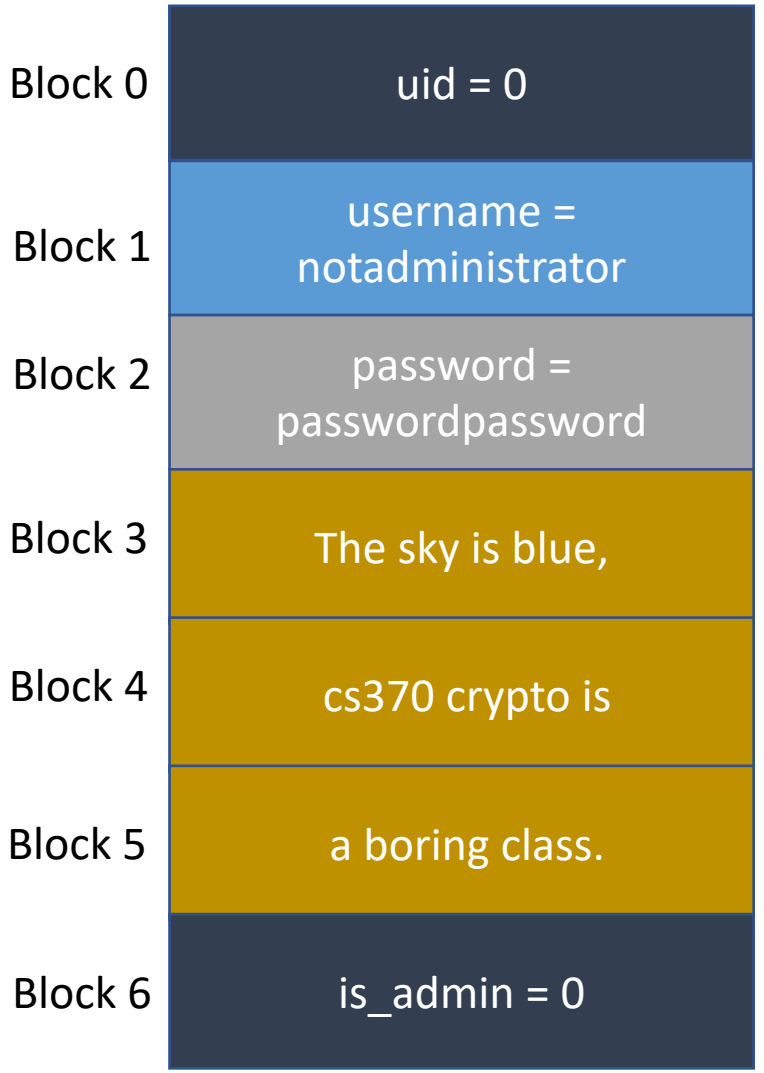
```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

ecb-attack

- Making uid == 0

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

block[0] = block[6]



ecb-attack

In template.py. Copy this as a solution.py to start with!
The result will be stored in flag1.user

- Making uid == 0

```
def create_file_for_flag_1():
    bytestring = ''

    # use whatever copied array here
    copied_blocks_bytes = copy.deepcopy(blocks_bytes)
    copied_blocks_hex = copy.deepcopy(blocks_hex)
    copied_blocks_int = copy.deepcopy(blocks_int)

    # XXX: Your code here; transform the blocks here
    copied_blocks_int[0] = copied_blocks_int[6]

    # in case you used blocks_int
    bytestring = convert_int_blocks_to_bytestring(copied_blocks_int)

    # write as flag1.user
    with open("flag1.user", "wb") as f:
        f.write(bytestring)
```

ecb-attack

- Run ./launcher
  - Supply flag1.user as the file

```
└─$ ./launcher
Running Command: [/home/labs/crypto/ecb-attack/ecb-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: flag1.user
Decrypted user information:
        uid        : 0
  username         : 'notadministrator'
  password         : 'passwordpassword'
   message         : 'The sky is blue, cs370 cry
  is_admin         : 0
```

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4): 1
For flag 1
User's uid == 0
You made its uid == 0! Reward is flag 1!
```

  - uid == 0

ecb-attack

- is_admin == 1

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

ecb-attack

- is_admin == 1

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

Uid = 1

ecb-attack

- is_admin == 1

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

ecb-attack

- is_admin == 1

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

block[6] = block[0]

ecb-attack

In template.py. Copy this as a solution.py to start with!
The result will be stored in flag2.user

- is_admin == 1

```
def create_file_for_flag_2():
    bytestring = ''

    # use whatever copied array here
    copied_blocks_bytes = copy.deepcopy(blocks_bytes)
    copied_blocks_hex = copy.deepcopy(blocks_hex)
    copied_blocks_int = copy.deepcopy(blocks_int)

    # XXX: Your code here; transform the blocks here
    copied_blocks_int[6] = copied_blocks_int[0]

    # in case you used blocks_int
    bytestring = convert_int_blocks_to_bytestring(copied_blocks_int)

    # write as flag2.user
    with open("flag2.user", "wb") as f:
        f.write(bytestring)
```

ecb-attack

- Run ./launcher
  - Supply flag2.user as the file

```
└─$ ./launcher
Running Command: [/home/labs/crypto/ecb-attack/ecb-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: flag2.user
Decrypted user information:
       uid      : 1
  username      : 'notadministrator'
  password      : 'passwordpassword'
   message      : 'The sky is blue, cs370 crypto is a
  is_admin      : 1
```

  - is_admin == 1

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4): 2
For flag 2
User's is_admin == 1
You made it as an admin! Reward is flag 2!
cs370{M0y3 0nE t0 iS aDmIn}
```

ecb-attack

- change password to a different string

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

ecb-attack

- change password to a different string

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

username = notadministrator

ecb-attack

- change password to a different string

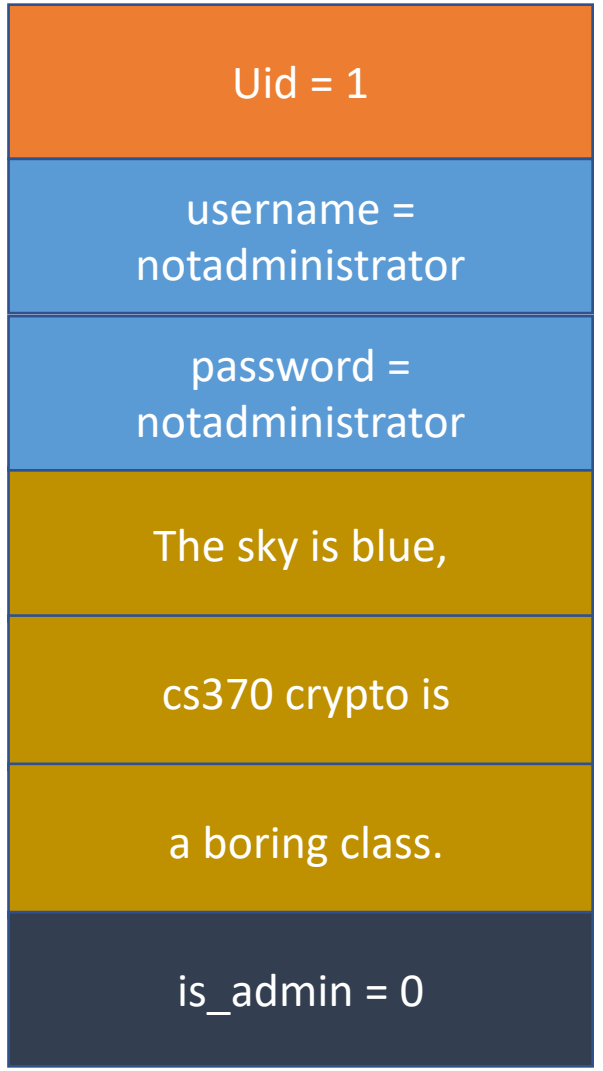
```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin == 1
3. I changed the password to something else
4. quit
Your choice (1-4):
```

ecb-attack

In template.py. Copy this as a solution.py to start with!
The result will be stored in flag3.user

- change password to a different string

```
def create_file_for_flag_3():
    bytestring = ''

    # use whatever copied array here
    copied_blocks_bytes = copy.deepcopy(blocks_bytes)
    copied_blocks_hex = copy.deepcopy(blocks_hex)
    copied_blocks_int = copy.deepcopy(blocks_int)

    # XXX: Your code here; transform the blocks here
    copied_blocks_int[2] = copied_blocks_int[1]

    # in case you used blocks_int
    bytestring = convert_int_blocks_to_bytestring(copied_blocks_int)

    # write as flag3.user
    with open("flag3.user", "wb") as f:
        f.write(bytestring)
```



ecb-attack

- change password to a different string
- ./launcher with flag3.user

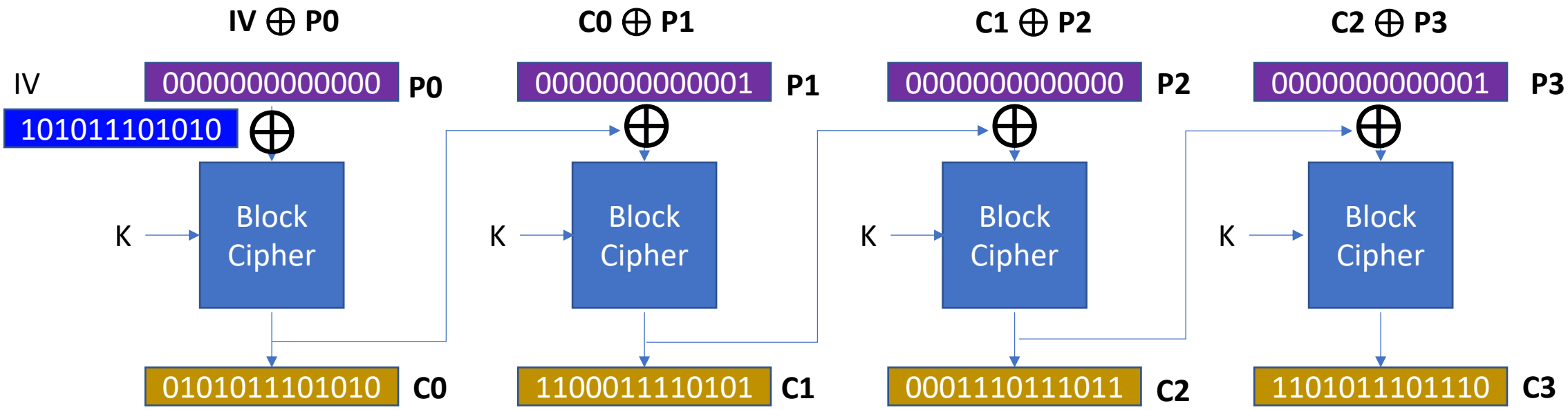
- Type notadministrator as password!

ECB Weakness

- A fixed plaintext <-> ciphertext mapping
- How can we avoid this?
- CBC Mode!

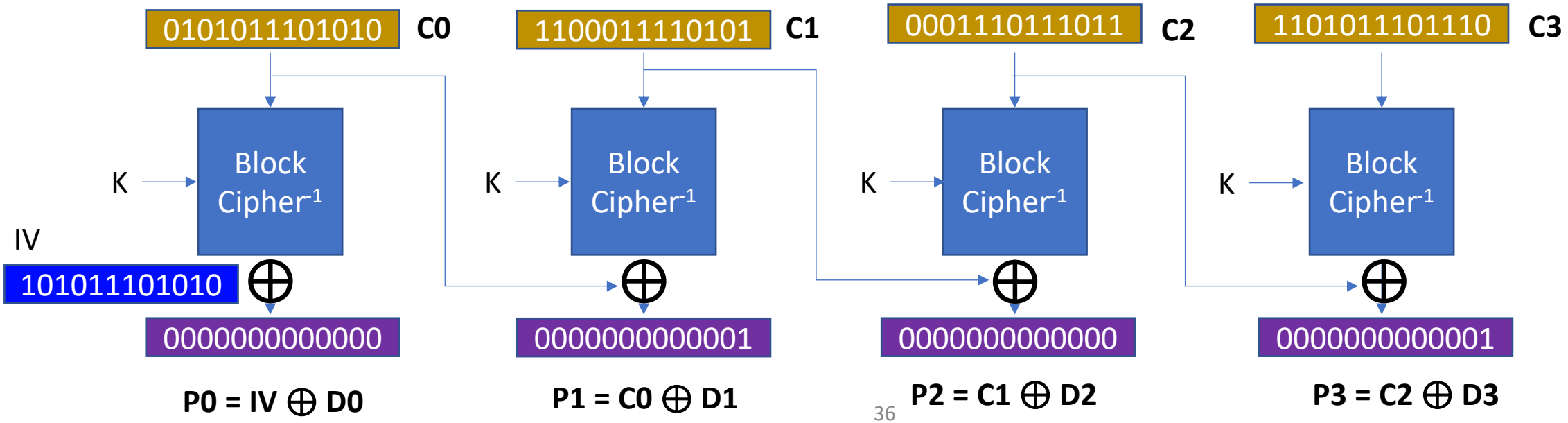
Cipher Block Chain (CBC)

- Apply XOR between the IV (Initialization Vector) and the plaintext
- Chain the previous ciphertext block to the plaintext with XOR
- Run Encryption on Xor'ed data



Cipher Block Chain (CBC)

- Decrypt first, and then apply XOR
- Starting with IV, XOR the previous ciphertext to the decrypted data

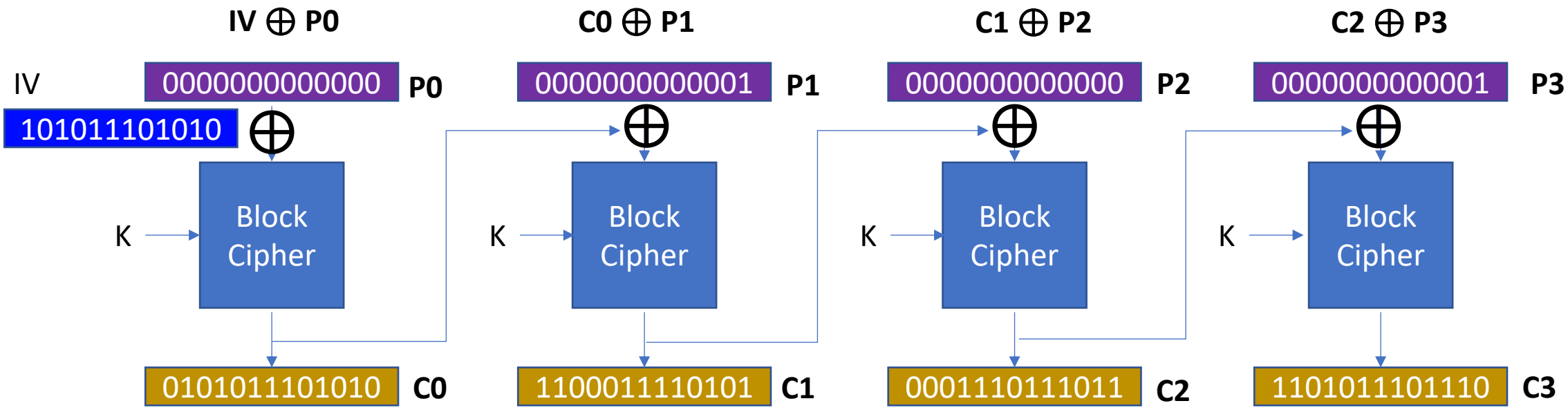


CBC Benefits

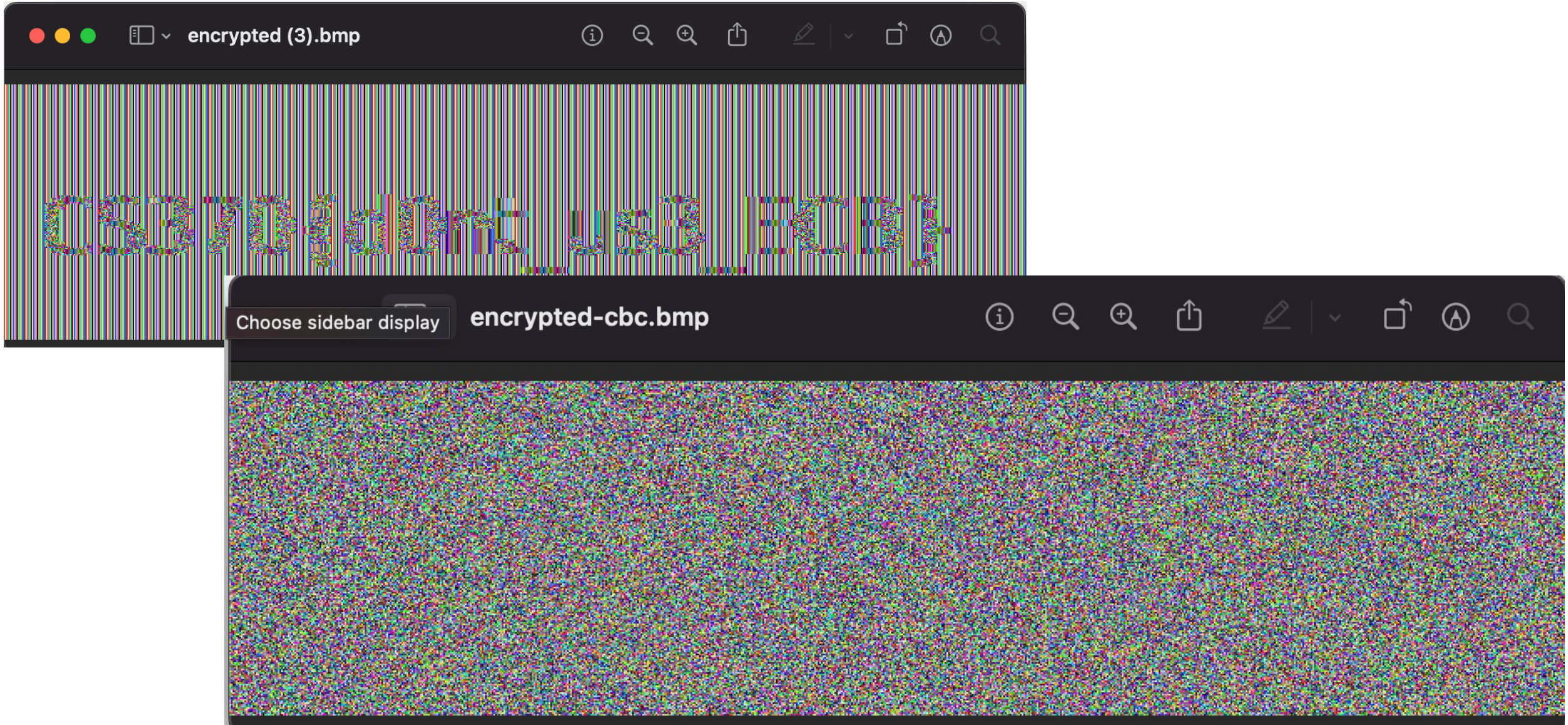
- ECB Weakness: 0 always encrypted in a fixed value… we can switch ciphertext to launch an attack
- Let's make each input to the block cipher looks like random
- Solution:
  - 1. Use a random IV, xor that to the plaintext; input will be random
  - 2. If the Block Cipher is PRP, the ciphertext looks like random
  - 3. Chain that random (ciphertext) to the next plaintext block…

CBC Benefits

- Solution:
  - 1. Use a random IV, xor that to the plaintext; input will be random
  - 2. If the Block Cipher is PRP, the ciphertext looks like random
  - 3. Chain that random (ciphertext) to the next plaintext block…

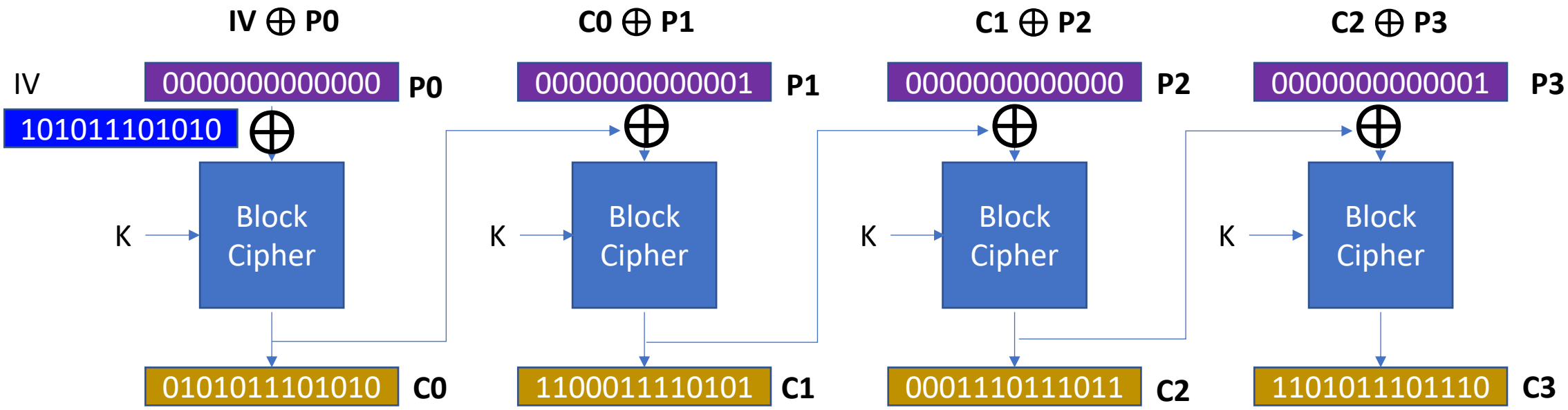


CBC-is-secure



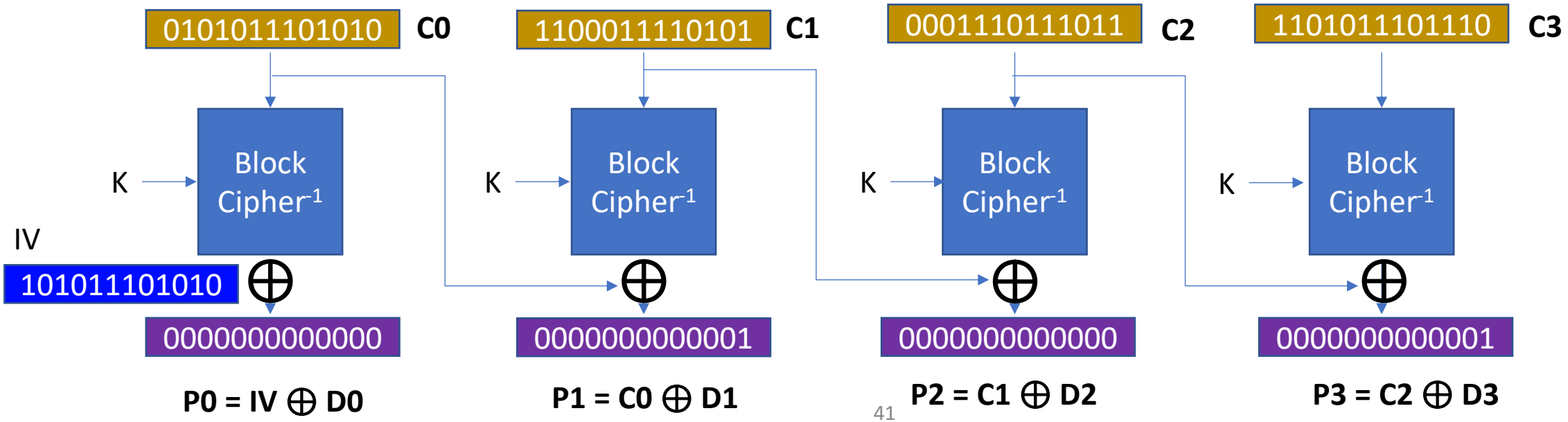
CBC Drawback

- We cannot parallelize encryption
- We need to have a previous ciphertext block to encrypt the next block



CBC Drawback

- We can run decryption in parallel
- We have all ciphertext at the start!



cbc-attack

- Encrypted.user
  - The example encrypted user

```
blue9057@blue9057-vm-ctf1 ~/crypto/cbc-attack <ruby-head>
$ ./launcher
Running Command: [/home/labs/crypto/cbc-attack/cbc-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: encrypted.user
Decrypted user information:
       uid      : 1
  username      : 'notadministrator'
  password      : 'passwordpassword'
   message      : 'The sky is blue, cs370 crypto is a boring class.'
  is_admin      : 0
```

cbc-attack

- Encrypted.user
  - The example encrypted user

- Need to accomplish:

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin != 0
3. I changed the message from:
a boring class.
to:
a superb class.
4. quit
Your choice (1-4):
```

cbc-attack

- uid == 0

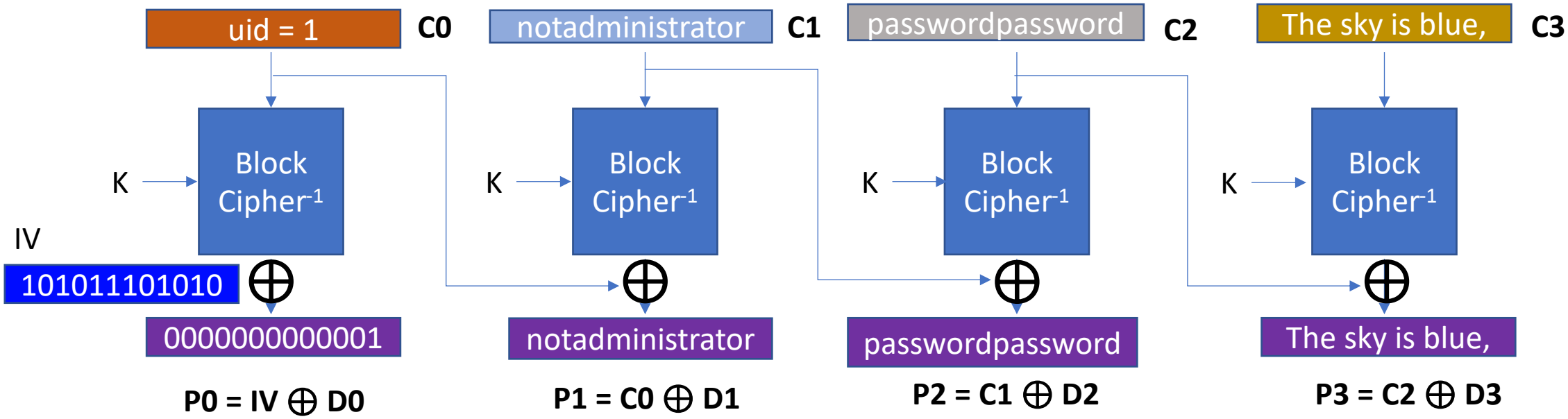
```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin != 0
3. I changed the message from:
a boring class.
to:
a superb class.
4. quit
Your choice (1-4):
```

is_admin = 0

We cannot re-use this

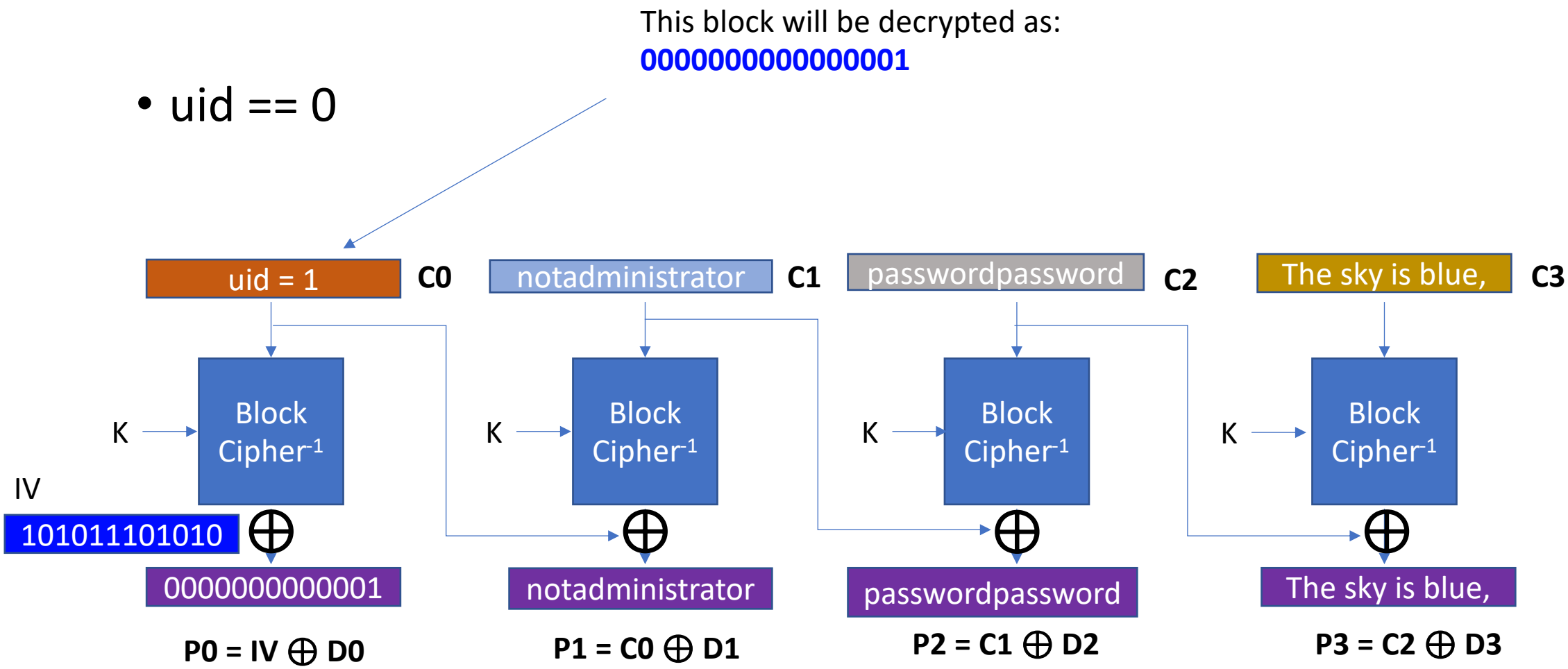
cbc-attack

- uid == 0



cbc-attack

- uid == 0

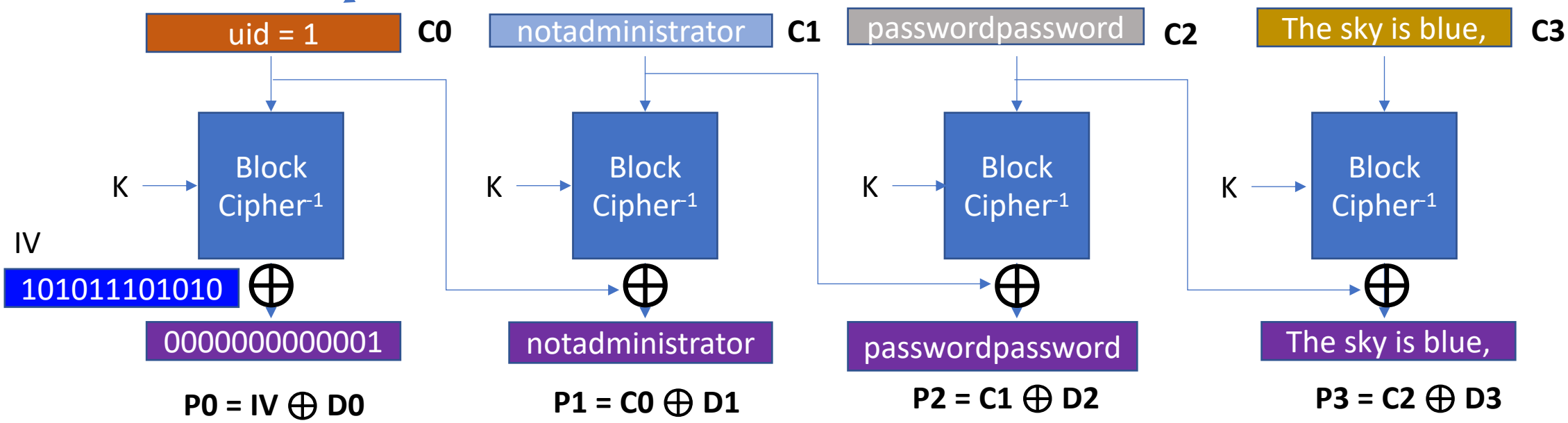


cbc-attack

- uid == 0

This block will be decrypted as:
0000000000000001

We don't manipulate the ciphertext

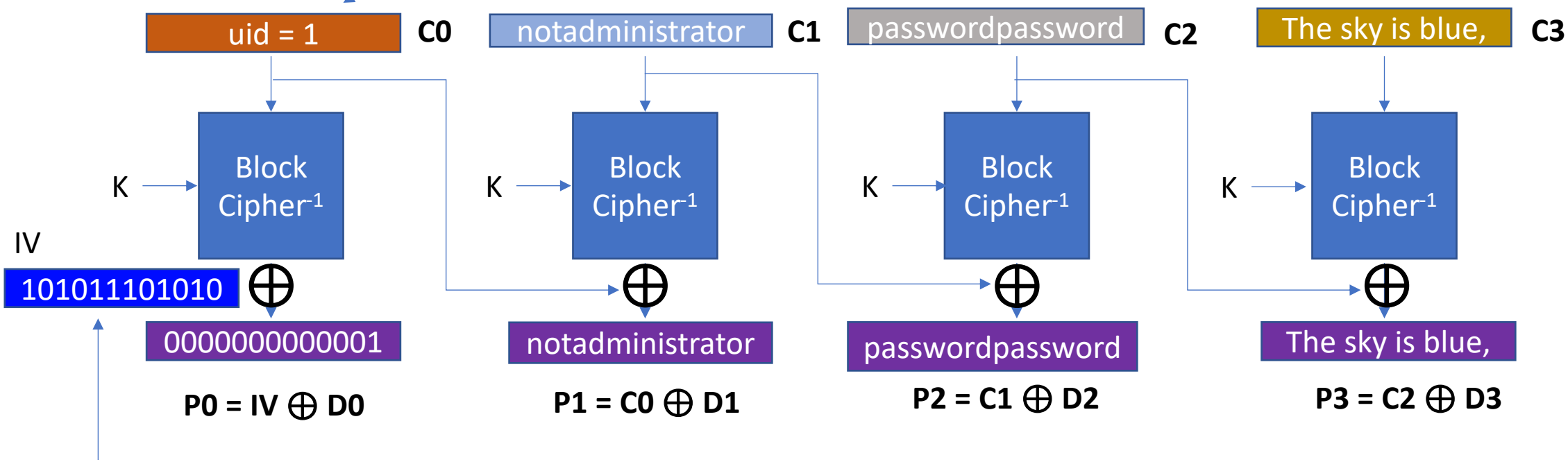


cbc-attack

- uid == 0

This block will be decrypted as:
0000000000000001

We don't manipulate the ciphertext



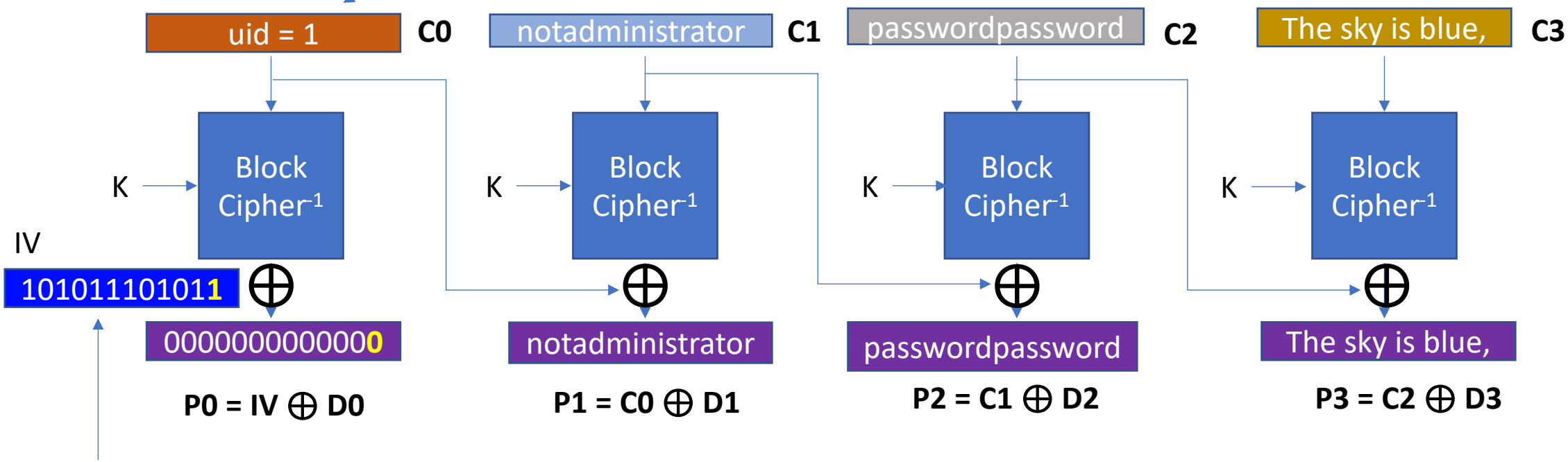
We manipulate IV; change the last bit from 0 to 1

cbc-attack

- uid == 0

This block will be decrypted as:
0000000000000001

We don't manipulate the ciphertext



We manipulate IV; change the last bit from 0 to 1

cbc-attack

```
def create_file_for_flag_1():
    bytestring = ''

    # use whatever copied array here
    copied_blocks_bytes = copy.deepcopy(blocks_bytes)
    copied_blocks_hex = copy.deepcopy(blocks_hex)
    copied_blocks_int = copy.deepcopy(blocks_int)

    # XXX: Your code here; transform the blocks here
    # block 0 is IV, -1 is the last integer. ^= 1 means flipping the last bit
    # i.e., change 1 to 0, 0 to 1..
    copied_blocks_int[0][-1] ^= 1

    # in case you used blocks_int
    bytestring = convert_int_blocks_to_bytestring(copied_blocks_int)

    # write as flag1.user
    with open("flag1.user", "wb") as f:
        f.write(bytestring)
```

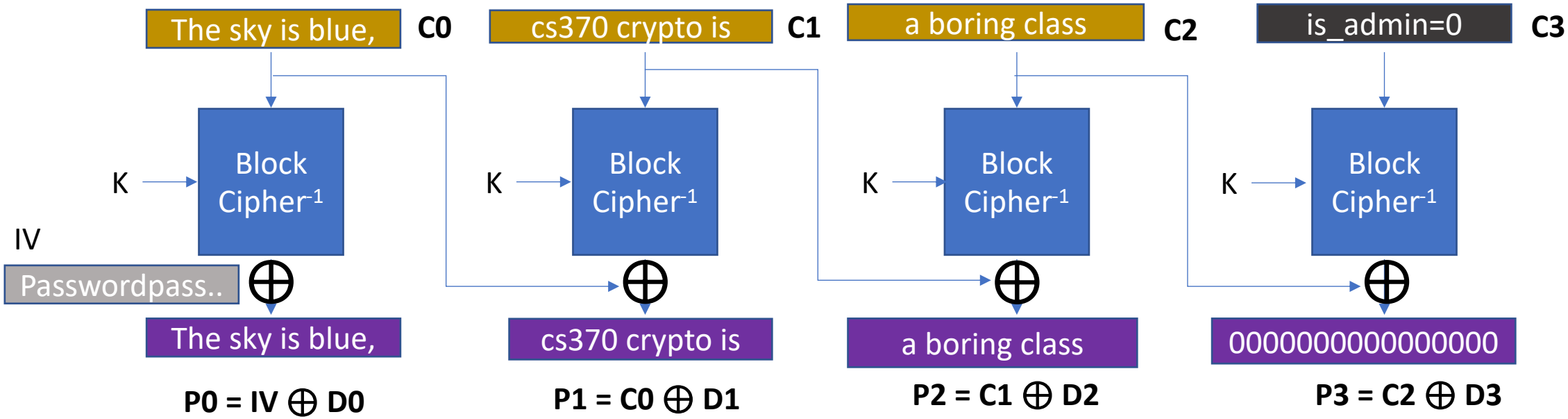
cbc-attack

```
└─$ ./launcher
Running Command: [/home/labs/crypto/cbc-attack/cbc-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: flag1.user
Decrypted user information:
      uid       : 0
  username      : 'notadministrator'
  password      : 'passwordpassword'
   message      : 'The sky is blue, cs370 crypto is a boring class.'
  is_admin      : 0
```

```python
def create_file_for_flag_1():
    bytestring = ''

    # use whatever copied array here
    copied_blocks_bytes = copy.deepcopy(blocks_bytes)
    copied_blocks_hex = copy.deepcopy(blocks_hex)
    copied_blocks_int = copy.deepcopy(blocks_int)

    # XXX: Your code here; transform the blocks here
    # block 0 is IV, -1 is the last integer. ^= 1 means flipping the last bit
    # i.e., change 1 to 0, 0 to 1..
    copied_blocks_int[0][-1] ^= 1

    # in case you used blocks_int
    bytestring = convert_int_blocks_to_bytestring(copied_blocks_int)

    # write as flag1.user
    with open("flag1.user", "wb") as f:
        f.write(bytestring)
```

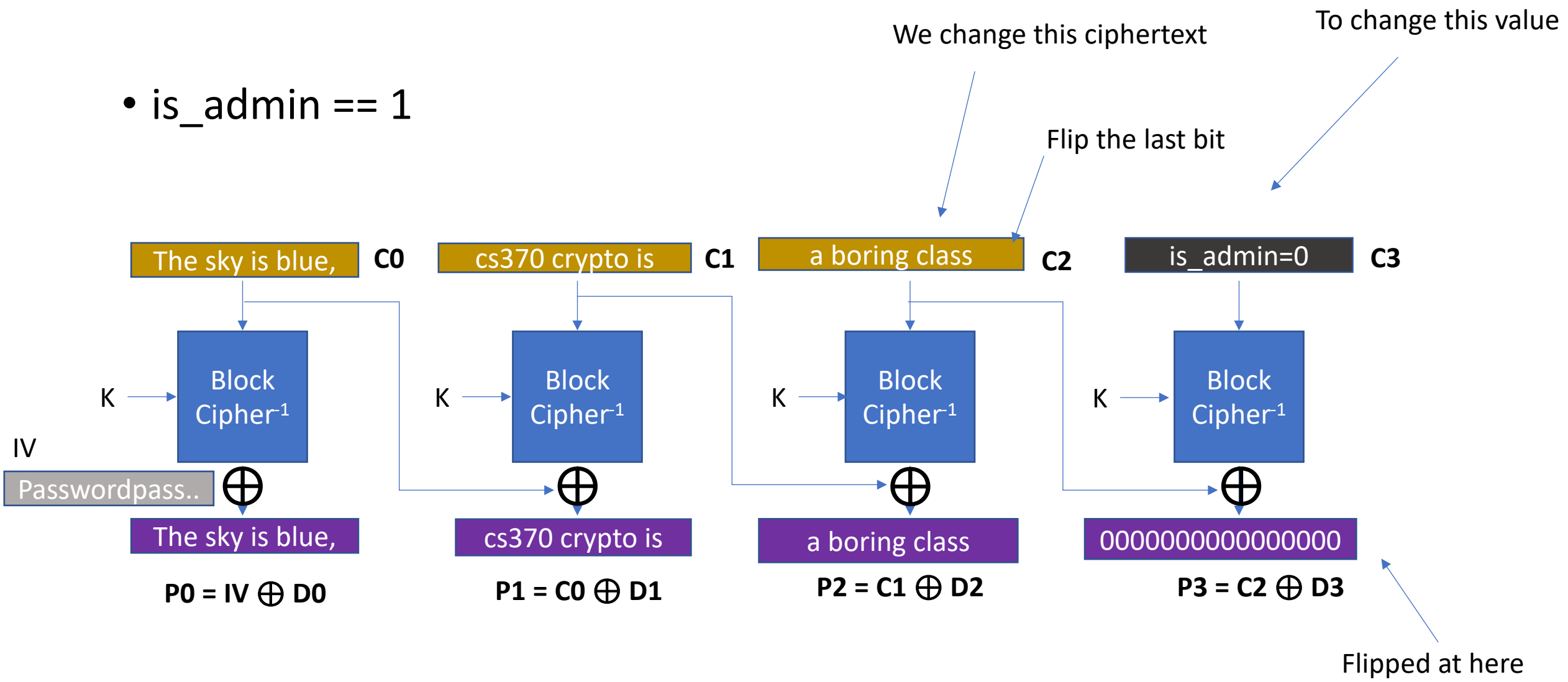
cbc-attack

- is_admin == 1



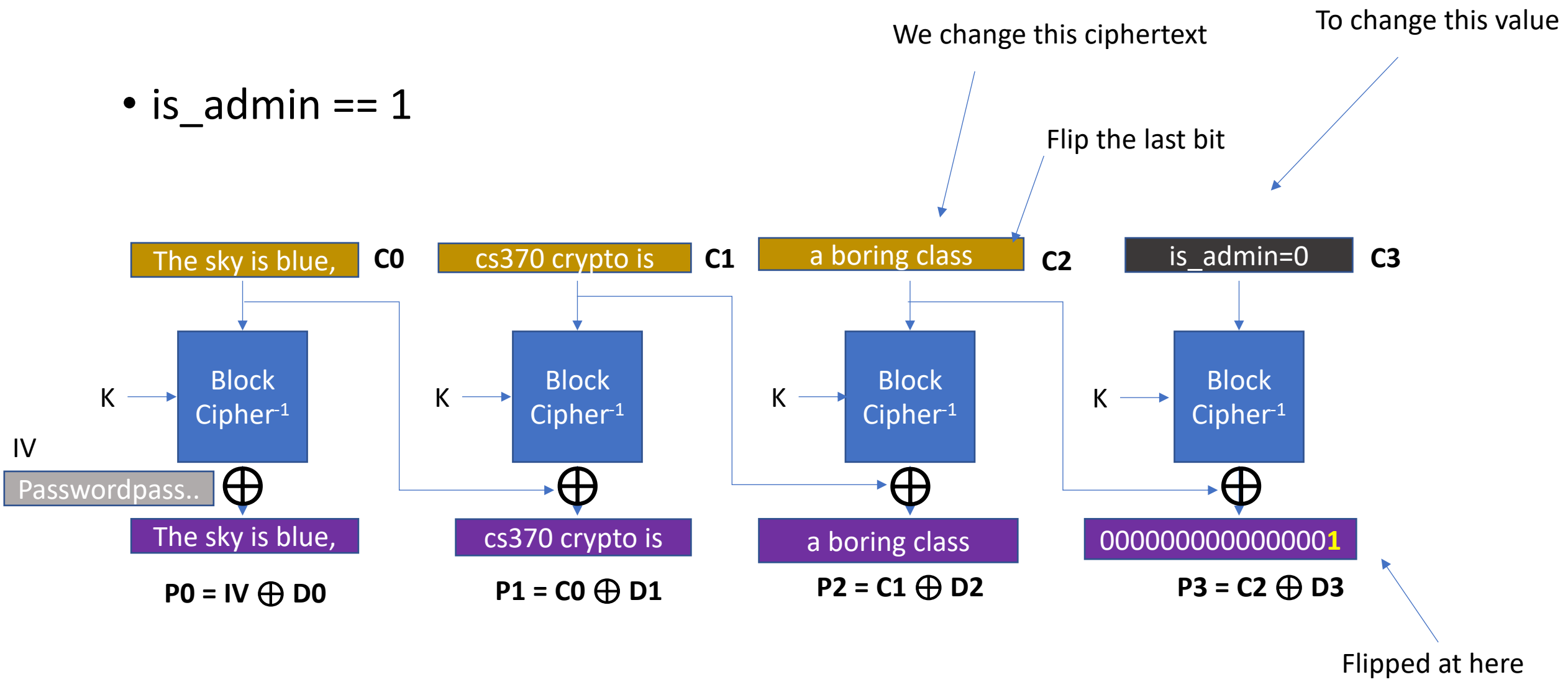
cbc-attack

- is_admin == 1



cbc-attack

- is_admin == 1



cbc-attack

- is_admin == 1

```
└─$ ./launcher
Running Command: [/home/labs/crypto/cbc-attack/cbc-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: flag2.user
Decrypted user information:
       uid      : 1
  username      : 'notadministrator'
  password      : 'passwordpassword'
   message      : 'The sky is blue, cs370 crypto is9\xe8l\xdd\x1a\x0c+4\x16\x11\
xc6\x91\xfbq\xff0'
  is_admin      : 1

Raw data: '0000000000000001notadministratorpasswordpasswordThe sky is blue, cs37
0 crypto is9\xe8l\xdd\x1a\x0c+4\x16\x11\xc6\x91\xfbq\xff0000000000000001'


Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin != 0
3. I changed the message from:
a boring class.
to:
a superb class.
4. quit
Your choice (1-4): 2
For flag 2
User's is_admin == 1
You made it as an admin! Reward is flag 2!
```

We broke the plaintext of block 6
But who cares?

cbc-attack

- is_admin == 1

```
└─$ ./launcher
Running Command: [/home/labs/crypto/cbc-attack/cbc-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: flag2.user
Decrypted user information:
      uid      : 1
  username     : 'notadministrator'
  password     : 'passwordpassword'
   message     : 'The sky is blue, cs370 crypto is9\xe8l\xdd\x1a\x0c+4\x16\x11\
xc6\x91\xfbq\xff0'
  is_admin     : 1

Raw data: '0000000000000001notadministratorpasswordpasswordThe sky is blue, cs37
0 crypto is9\xe8l\xdd\x1a\x0c+4\x16\x11\xc6\x91\xfbq\xf0000000000000001'


Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin != 0
3. I changed the message from:
a boring class.
to:
a superb class.
4. quit
Your choice (1-4): 2
For flag 2
User's is_admin == 1
You made it as an admin! Reward is flag 2!
```

We can make is_admin == 1

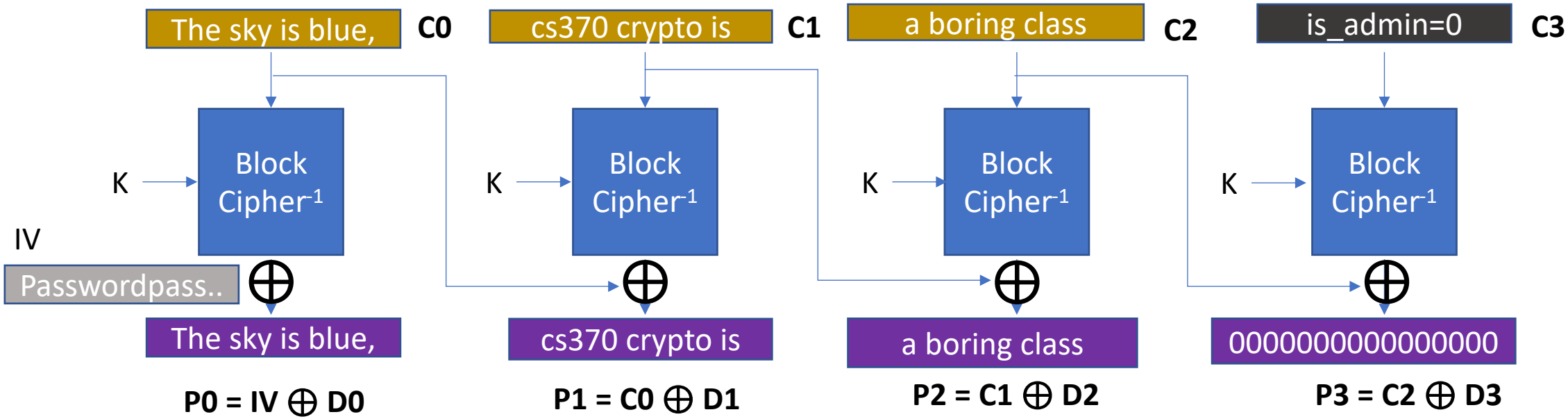
cbc-attack

- boring to superb?
- Need to accomplish:

```
Choose which flag do you want to get:
1. I made uid == 0 (super user)
2. I made is_admin != 0
3. I changed the message from:
a boring class.
to:
a superb class.
4. quit
Your choice (1-4):
```

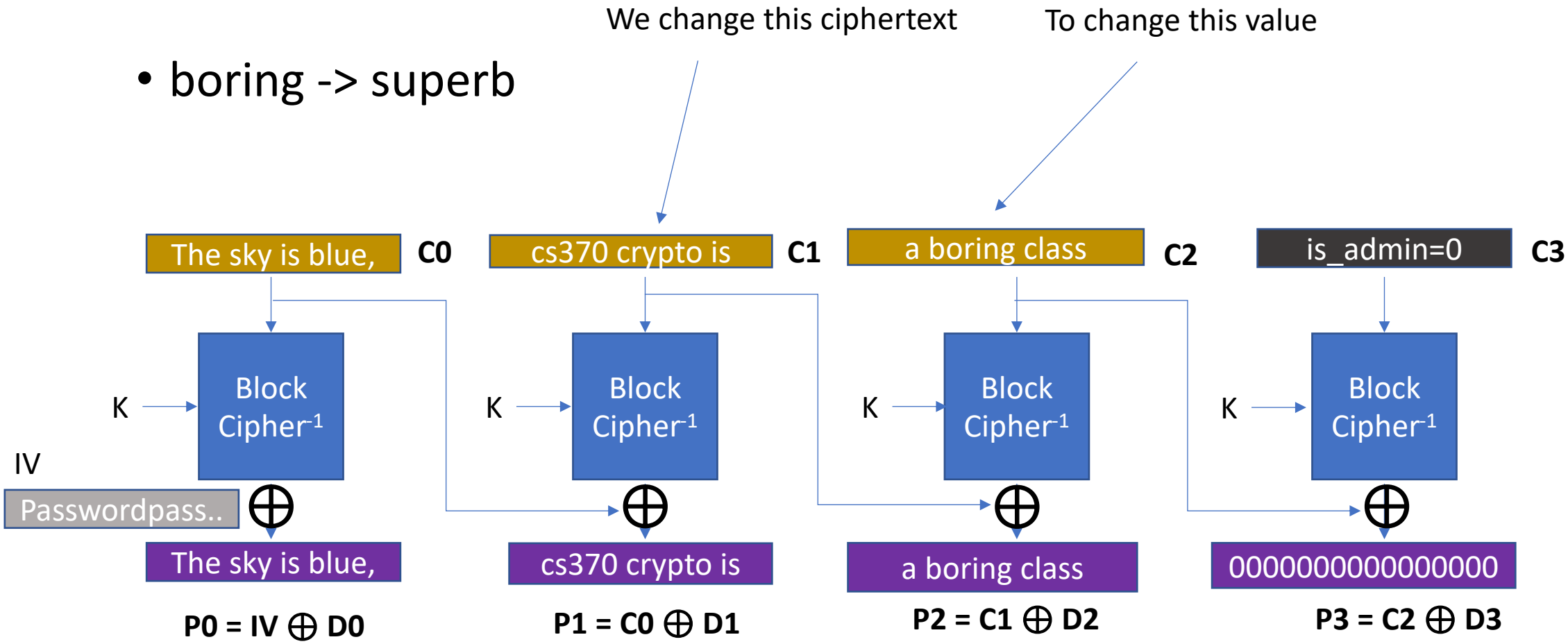
cbc-attack

- boring -> superb



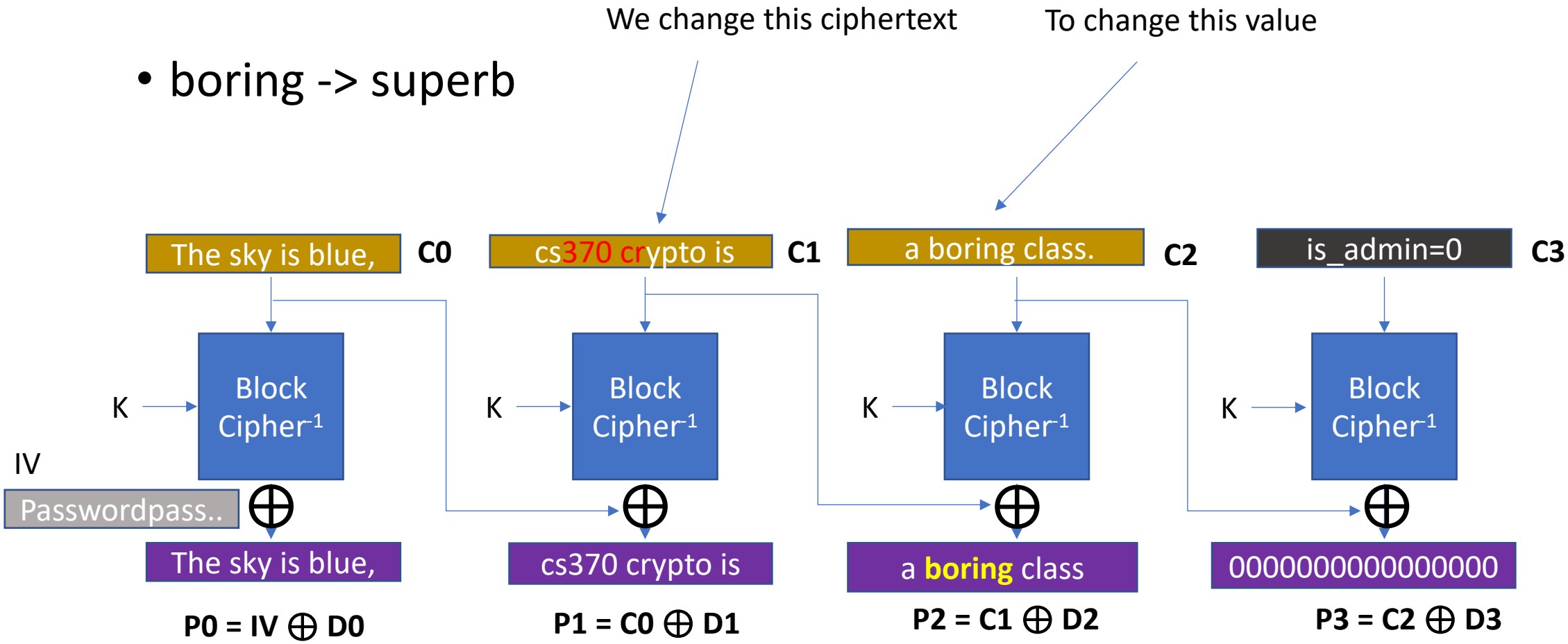
cbc-attack

- boring -> superb



cbc-attack

- boring -> superb



cbc-attack

- boring -> superb

```
# FYI: 16-byte aligned string
"""
The sky is blue,
 cs370 crypto is
 a boring class.                # No, not at all.....
"""
def create_file_for_flag_3():
    bytestring = ''

    # use whatever copied array here
    copied_blocks_bytes = copy.deepcopy(blocks_bytes)
    copied_blocks_hex = copy.deepcopy(blocks_hex)
    copied_blocks_int = copy.deepcopy(blocks_int)

    # XXX: Your code here; transform the blocks here
    b = 'boring'
    s = 'superb'
    k = []
    # build an XOR key between boring and superb
    for i in range(len(b)):
        k.append(ord(b[i]) ^ ord(s[i]))

    for i in range(len(k)):
        copied_blocks_int[5][3+i] ^= k[i]

    # in case you used blocks_int
    bytestring = convert_int_blocks_to_bytestring(copied_blocks_int)

    # write as flag3.user
    with open("flag3.user", "wb") as f:
        f.write(bytestring)
```

Build the key

Apply XOR

cbc-attack

- boring -> superb

```
└─$ ./launcher
Running Command: [/home/labs/crypto/cbc-attack/cbc-decryptor.py]
Key was loaded successfully!
Give me the filename of your encrypted object: flag3.user
Decrypted user information:
       uid      : 1
  username      : 'notadministrator'
  password      : 'passwordpassword'
   message      : 'The sky is blue,l\xd7\xdd\x98\xbfsD\xb6\xa0{\x89\x8c@\x81]\x99 a superb class.'
  is_admin      : 0
```

- It smashs the block before ‘superb’ but the validator does not care..

ECB/CBC Drawbacks

- ECB Pros: can encrypt/decrypt in parallel
- ECB Cons: Attackers can easily stitch ciphertext to build plaintext

- CBC Pros: can decrypt in parallel, ciphertext pattern is not fixed!
- CBC Cons: Cannot encrypt in parallel

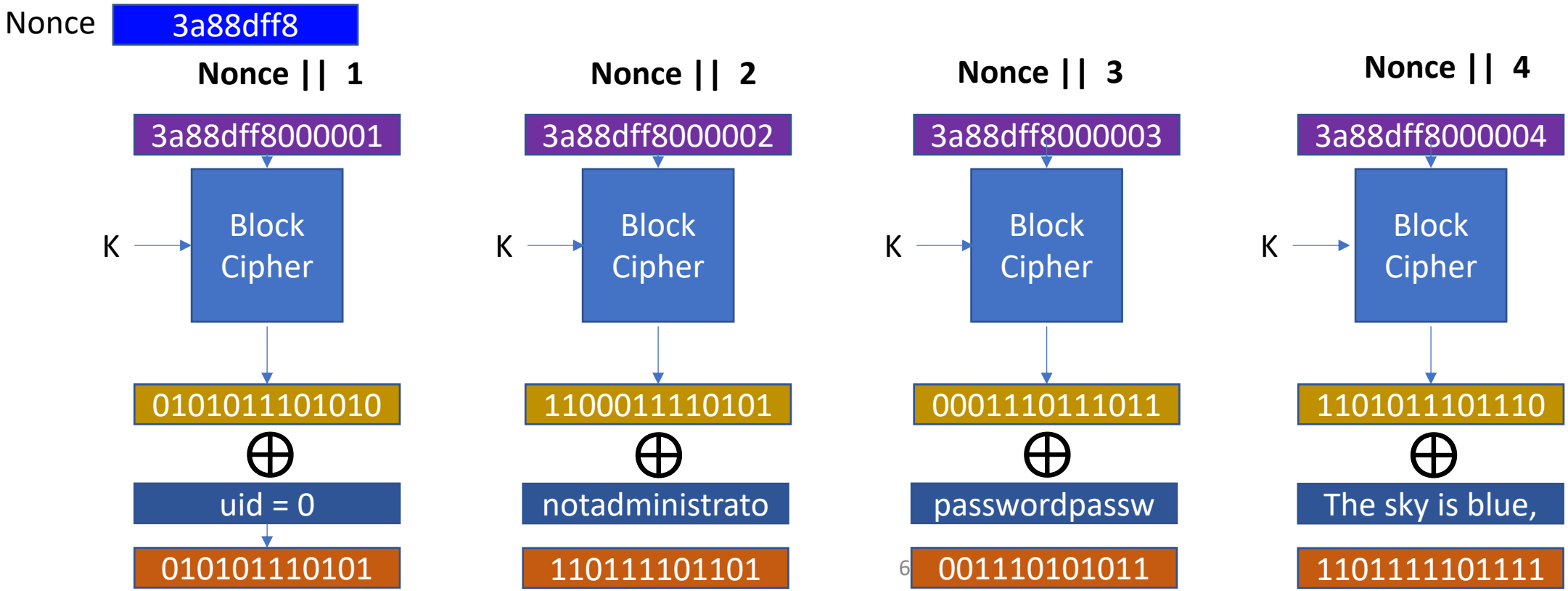
- Can we have a mode that
  - Can en/decrypt in parallel and
  - Ciphertext pattern is not fixed?

Counter Mode

- CTR (Counter mode)
- Start with a random nonce || counter
- It uses the block cipher as random number generator
- V = Enc(nonce || counter)
- Then, XOR this V to the plaintext
  - C = P $\oplus$ V

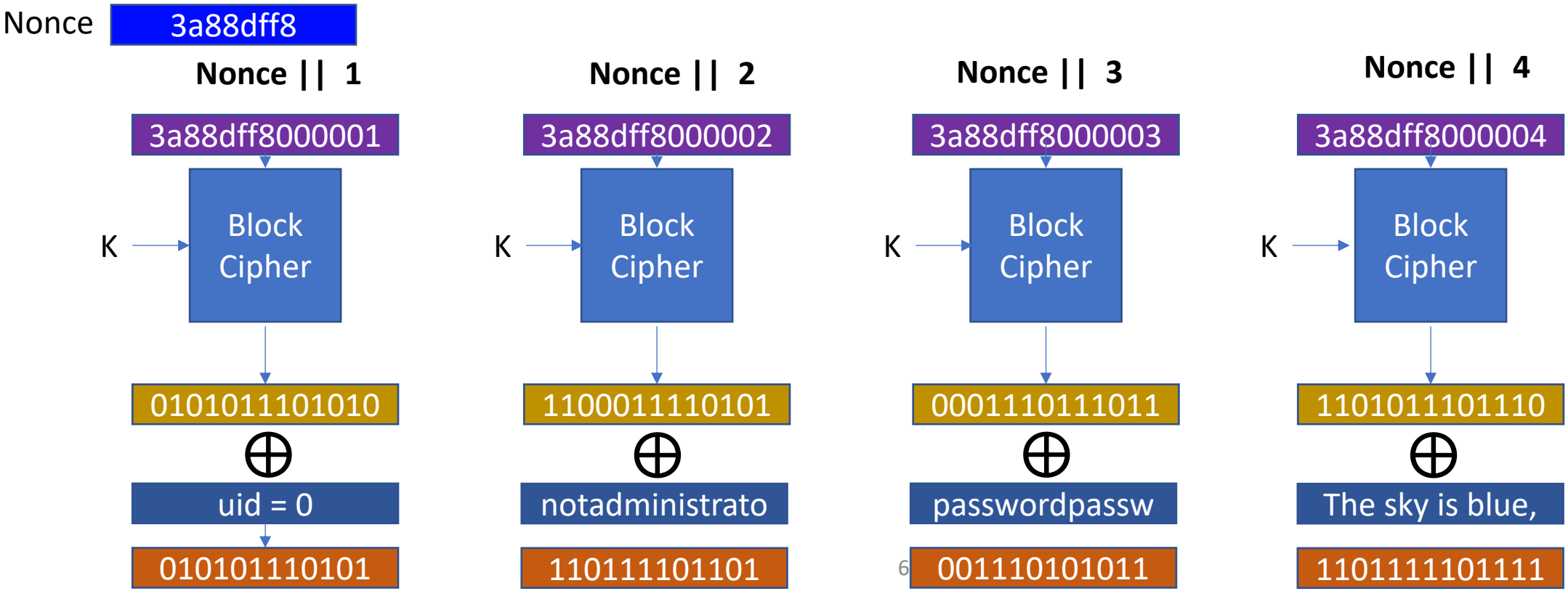
Counter Mode

- CTR (Counter mode)
- Start with a random nonce || counter



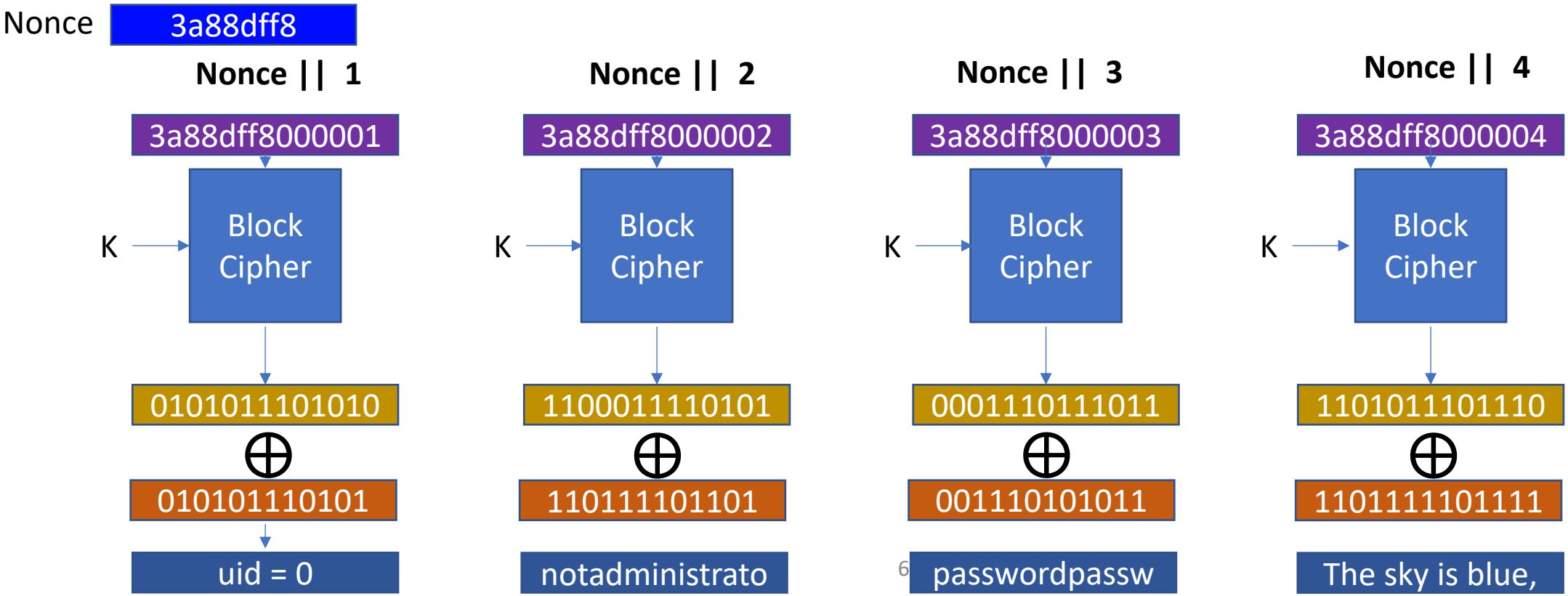
Counter Mode Encryption

- CTR (Counter mode)
- Start with a random nonce || counter



Counter Mode Decryption

- CTR (Counter mode)
- Start with a random nonce || counter



CTR Benefits

- Can encrypt in parallel
- Can decrypt in parallel
  - Basically, encryption and decryption are both encrypting counters
  - We apply XOR to the output with (plaintext/ciphertext)

CTR Weaknesses

- Any bitflip in the ciphertext will be direct bitflip in the plaintext..

Challenges

- ECB-attack
  - Hint: Moving the ciphertext to a different row!
- CBC-attack
  - Hint: apply bitflip to the IV or previous ciphertext. That will show directly in the plaintext
- CTR-attack
  - Hint: apply bitflip to any ciphertext. That will show directly in the plaintext
- CBC-is-secure
  - Can you try all password to decrypt the CBC-encrypted data?
  - Use passwords from password-list.txt

Summary

- ECB (Electronic Code Book)
  - Can run Encryption/Decryption in parallel
  - Leaks plaintext block patterns in ciphertext blocks
- CBC (Cipher Block Chain)
  - Always use random number to be XOR'ed to the plaintext
    - No ciphertext pattern leaking
  - Can run decryption in parallel
  - Apply XOR to the-block-before will affect to the plaintext block
- CTR
  - Always use the random nonce || counter as a block cipher input
  - Apply XOR to the ciphertext block will affect to the plaintext block