# Block Ciphers

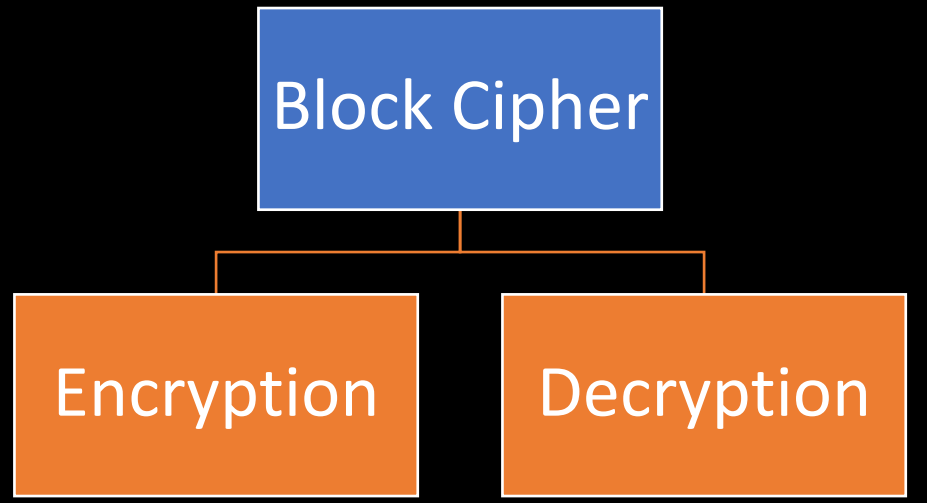
# Content

| Content |
| --- |
| Introduction |
| Constructing Block Ciphers |
| Modes of Operation |
| Advanced Encryption Standard |

# Introduction

What is a block cipher?

- A cipher combined with a mode of operation to process data in blocks.



# Introduction

- Secure block cipher = pseudorandom permutation (PRP).
  - PRP is a function to shuffle the input.

- Security objectives:
  - Cannot produce any ciphertext without a key
  - Cannot discover any pattern in plaintext/ciphertext
  - Indistinguishable from random permutation
  - Impossible to recover the secret key
  - Cannot recover plaintext from ciphertext without the key.

# Introduction

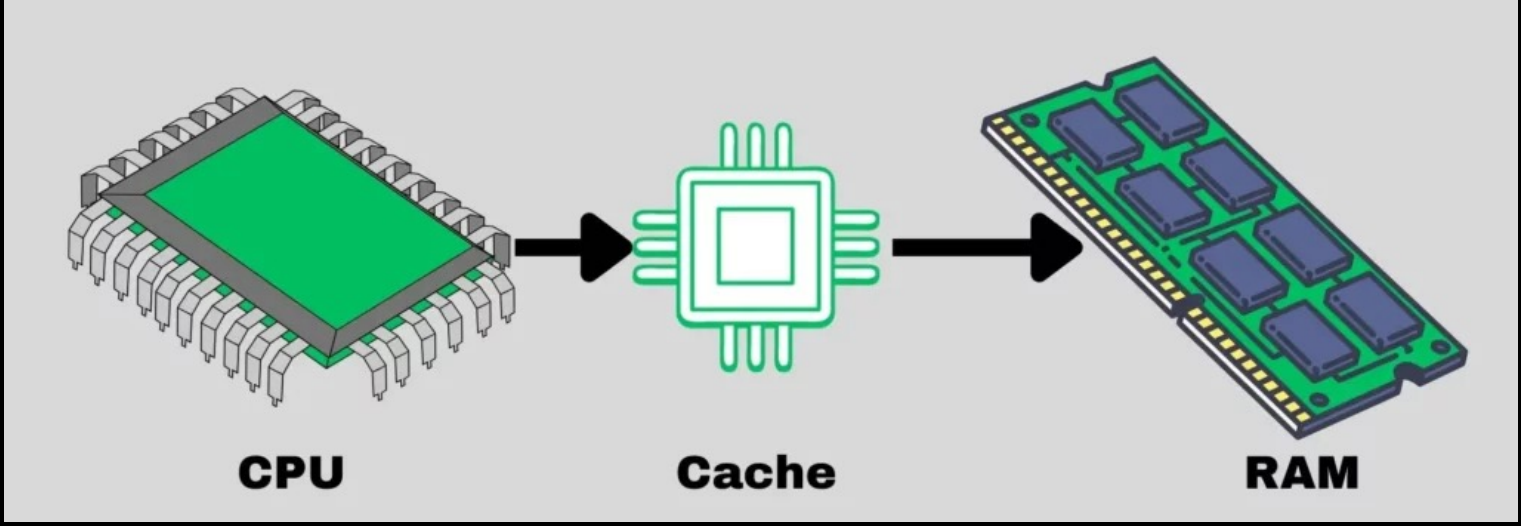
- A block cipher depends on two value:
  - **Block size:** the length of single unit processed by the PRP of the cipher.
  - **Key size:** the length of the key used in encryption and decryption.

- Most block ciphers have either 64-bit or 128-bit blocks
  - DES's blocks have 64 ($2^6$) bits
  - AES's blocks have 128 ($2^7$) bits.

- What is the ideal block size?

# Introduction

- A block size should not be too large to minimize memory footprint.
- Blocks of 64, 128, 256 bits are short enough.
  - Such size can fit into the registers of most CPUs.
  - Allow efficient implementations.



# Introduction

- While blocks shouldn’t be too large, they also shouldn’t be too small.
- Short block sizes make the cipher susceptible to codebook attacks.
  - An attack on a block cipher, where you generate every possible plaintext and consequently every possible ciphertext for a fixed key.
  - KPA model.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | E0 | F7 | 6 | 1A | 23 | 3E | 47 | 5C | 6E | 79 | 83 | 9B | A2 | B2 | CE | D8 |
| 1 | F6 | 5 | 19 | 22 | 3D | 46 | 5B | 6D | 78 | 82 | 9A | A1 | B1 | CD | D7 | EF |
| 2 | 4 | 18 | 21 | 3C | 45 | 5A | 6C | 77 | 81 | 99 | A0 | B0 | CC | D6 | EE | F5 |
| 3 | 17 | 20 | 3B | 44 | 59 | 6B | 76 | 80 | 98 | AF | BF | CB | D5 | ED | F4 | 3 |
| 4 | 2F | 3A | 43 | 58 | 6A | 75 | 8F | 97 | AE | BE | CA | D4 | EC | F3 | 2 | 16 |
| 5 | 39 | 42 | 57 | 69 | 74 | 8E | 96 | AD | BD | C9 | D3 | EB | F2 | 1 | 15 | 2E |
| 6 | 41 | 56 | 68 | 73 | 8D | 95 | AC | BC | C8 | D2 | EA | F1 | 0 | 14 | 2D | 38 |
| 7 | 55 | 67 | 72 | 8C | 94 | AB | BB | C7 | D1 | E9 | F0 | F | 13 | 2C | 37 | 40 |
| 8 | 66 | 71 | 8B | 93 | AA | BA | C6 | D0 | E8 | FF | E | 12 | 2B | 36 | 4F | 54 |
| 9 | 70 | 8A | 92 | A9 | B9 | C5 | DF | E7 | FE | D | 11 | 2A | 35 | 4E | 53 | 65 |
| A | 89 | 91 | A8 | B8 | C4 | DE | E6 | FD | C | 10 | 29 | 34 | 4D | 52 | 64 | 7F |
| B | 90 | A7 | B7 | C3 | DD | E5 | FC | B | 1F | 28 | 33 | 4C | 51 | 63 | 7E | 88 |
| C | A6 | B6 | C2 | DC | E4 | FB | A | 1E | 27 | 32 | 4B | 50 | 62 | 7D | 87 | 9F |
| D | B5 | C1 | DB | E3 | FA | 9 | 1D | 26 | 31 | 4A | 5F | 61 | 7C | 86 | 9E | A5 |
| E | C0 | DA | E2 | F9 | 8 | 1C | 25 | 30 | 49 | 5E | 60 | 7B | 85 | 9D | A4 | B4 |
| F | D9 | E1 | F8 | 7 | 1B | 24 | 3F | 48 | 5D | 6F | 7A | 84 | 9C | A3 | B3 | CF |

# Content

| Content |
| --- |
| Introduction |
| Constructing Block Ciphers |
| Modes of Operation |
| Advanced Encryption Standard |
| Simon and Speck Ciphers |

# Constructing Block Ciphers

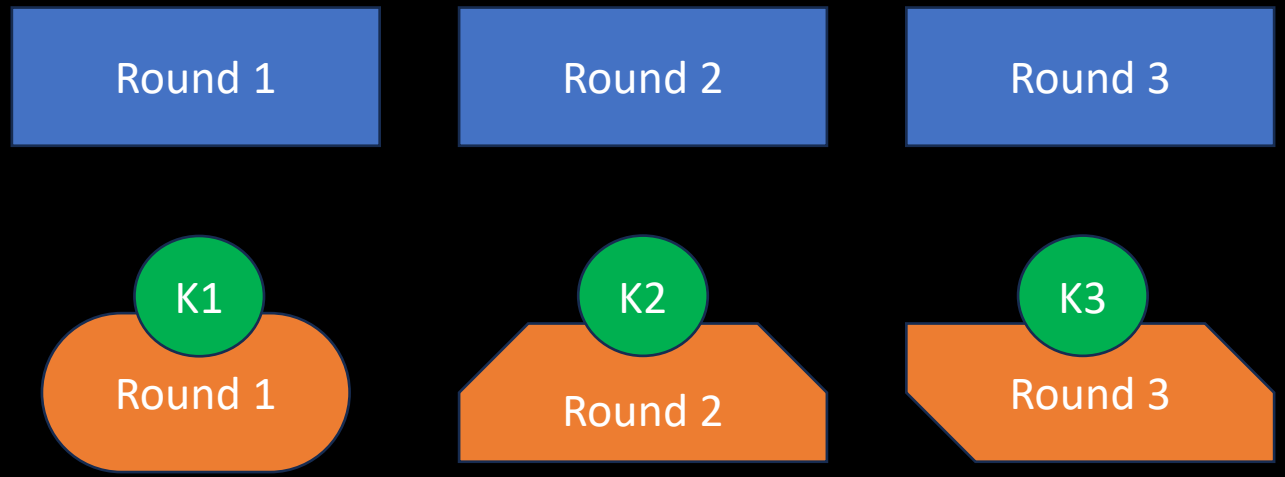
- Encryption: performing a sequence of rounds.
  - Each round performs weak operations on its own.
  - More rounds → strong cipher.

- A block cipher with three rounds: $\boldsymbol{R_3}\left(\boldsymbol{R_2}\left(\boldsymbol{R_1}(P)\right)\right)$

- To decrypt, each round should have an inverse: $\boldsymbol{iR_1}\left(\boldsymbol{iR_2}\left(\boldsymbol{iR_3}(C)\right)\right)$

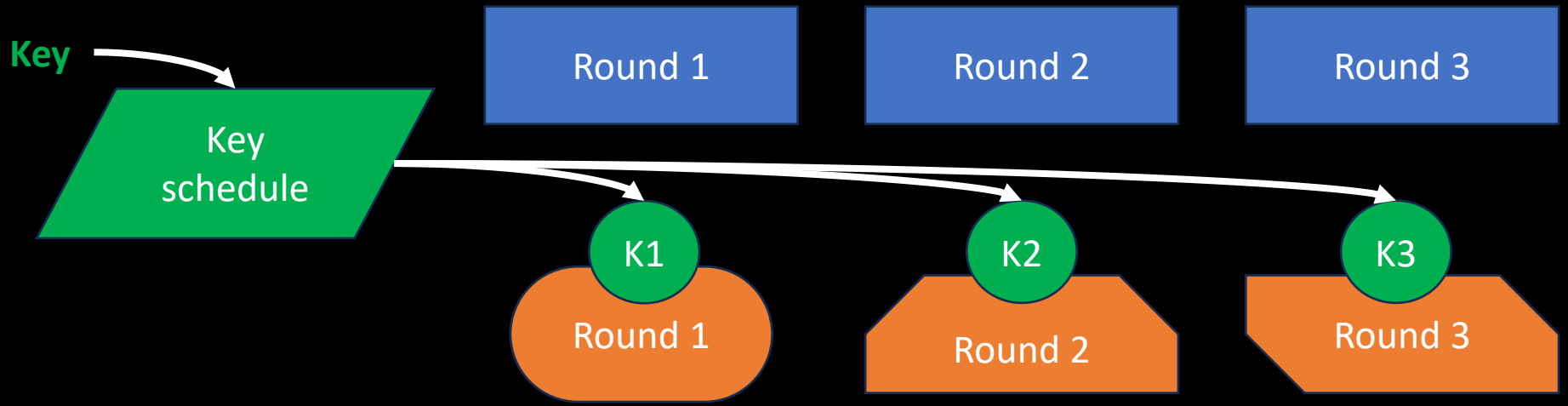
# Constructing Block Ciphers

- The round functions are identical, but they are parameterized by a value called the *round key*.
  - Different keys → different rounds.



# Constructing Block Ciphers

- The round functions are identical, but they are parameterized by a value called the *round key*.
  - Different keys → different rounds.
- Round keys are derived from the main key, K, using a key schedule algorithm.



# Constructing Block Ciphers

- Identical rounds → *slide attack*.
- Slide attacks look for two plaintext/ciphertext pairs (P1, C1) and (P2, C2), where P2 = **R**(P1) if **R** is the cipher's round.
  - Knowing the input/output of a single round helps recovering the key.

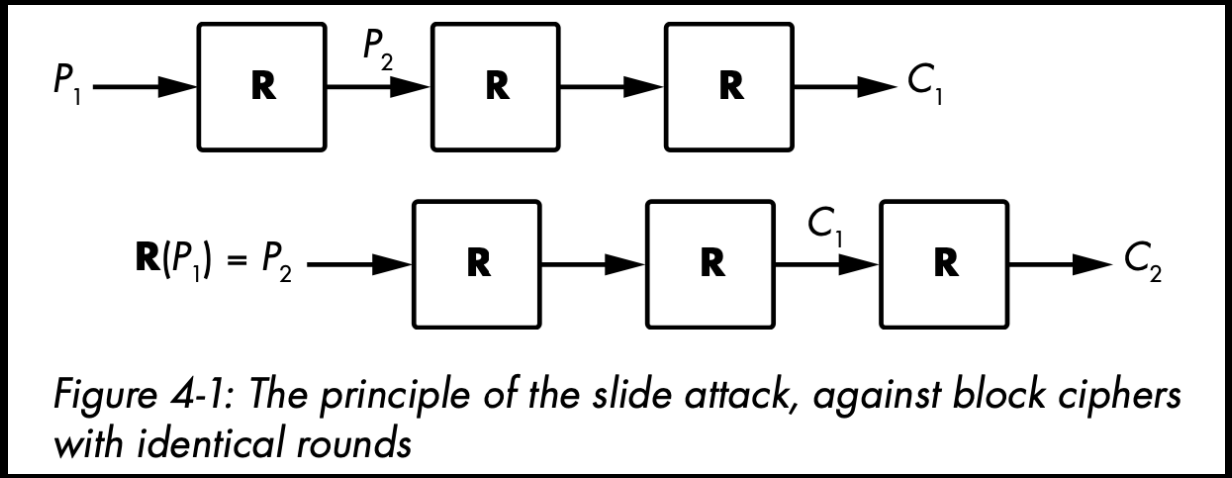
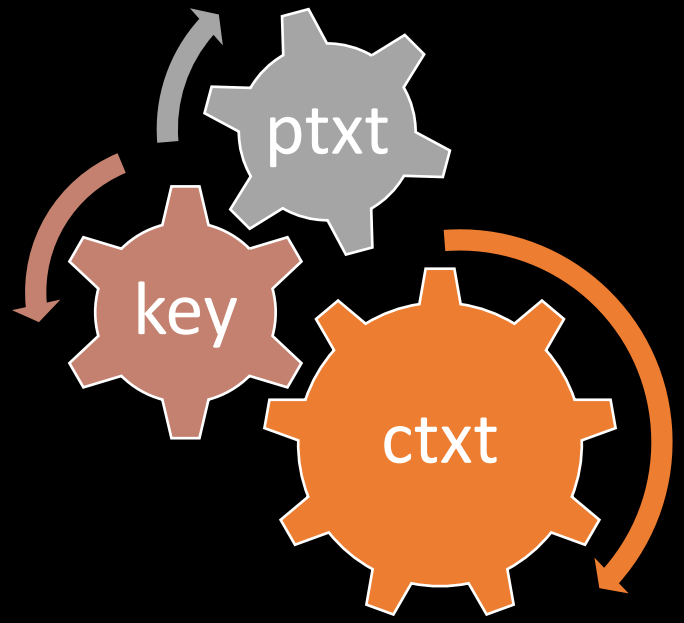



*Figure 4-1: The principle of the slide attack, against block ciphers with identical rounds*

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=62e452197f49c29e499342d978e562999c09cbbf

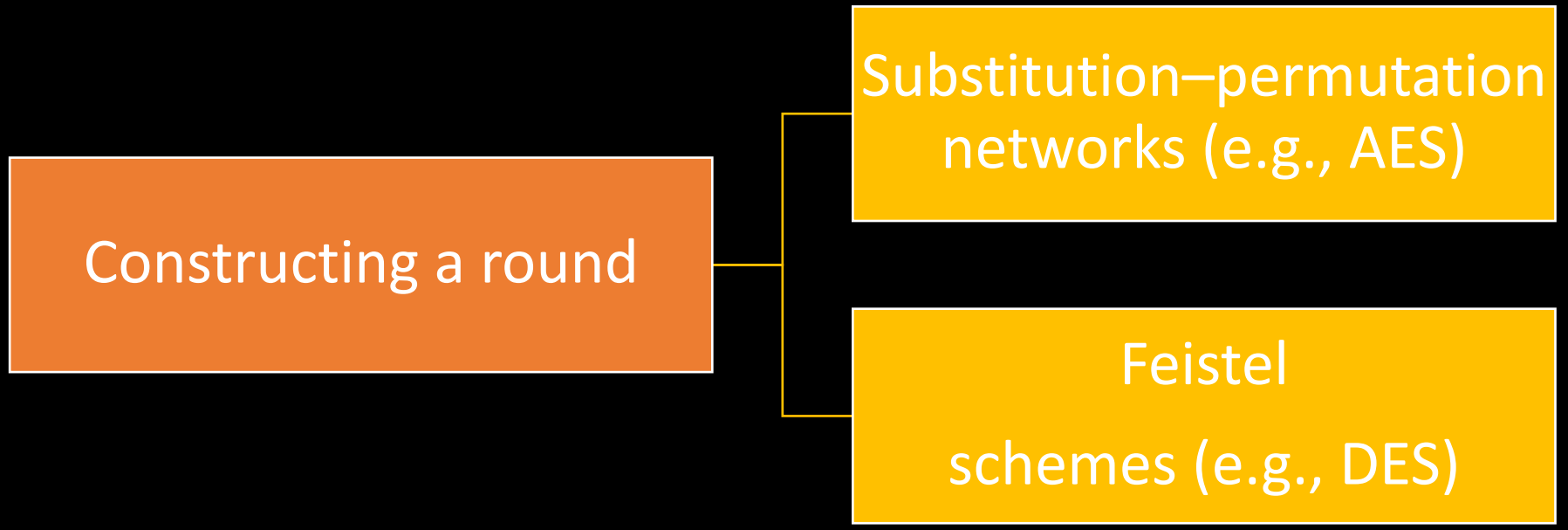
# Constructing Block Ciphers

- Properties of block ciphers:
  - **Confusion**: the input (plaintext and key) undergoes complex transformations.
  - **Diffusion**: the transformations depend equally on all bits of the input.
    - Changing 1 bit plaintext changes half of the ciphertext



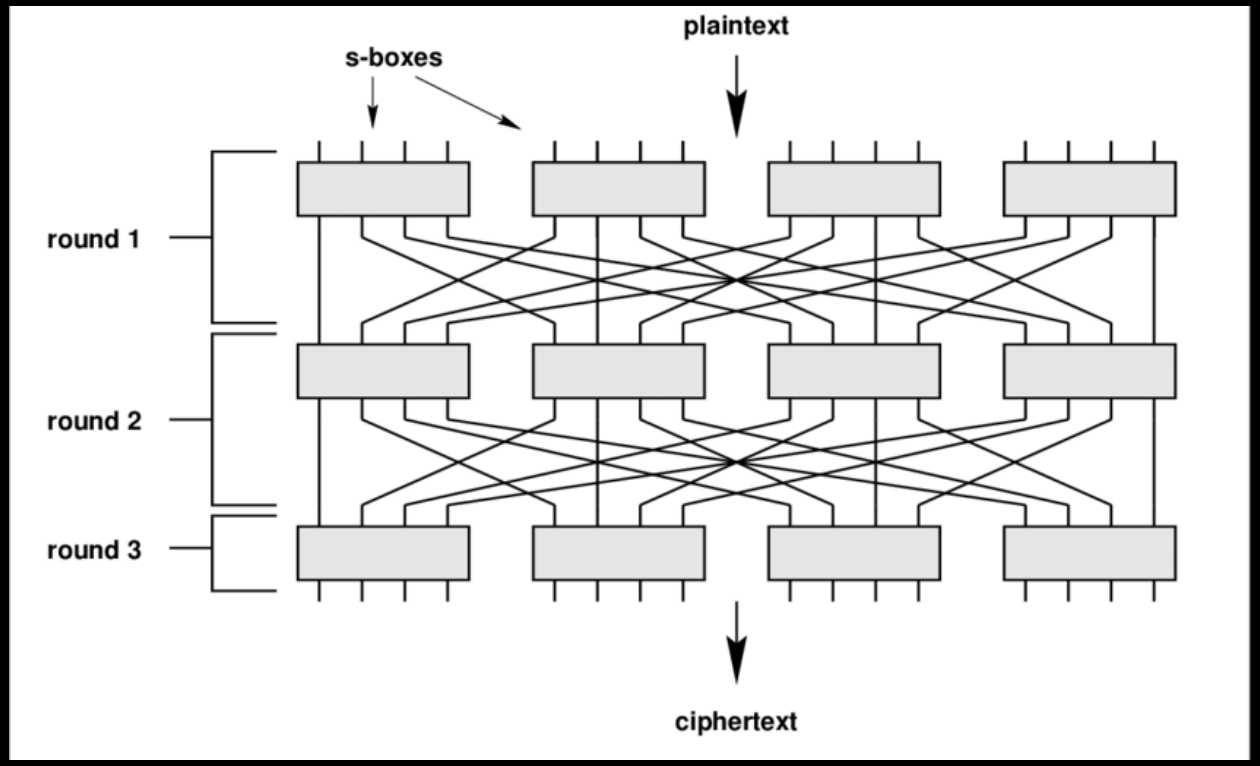
# Constructing Block Ciphers

- Two techniques to construct rounds



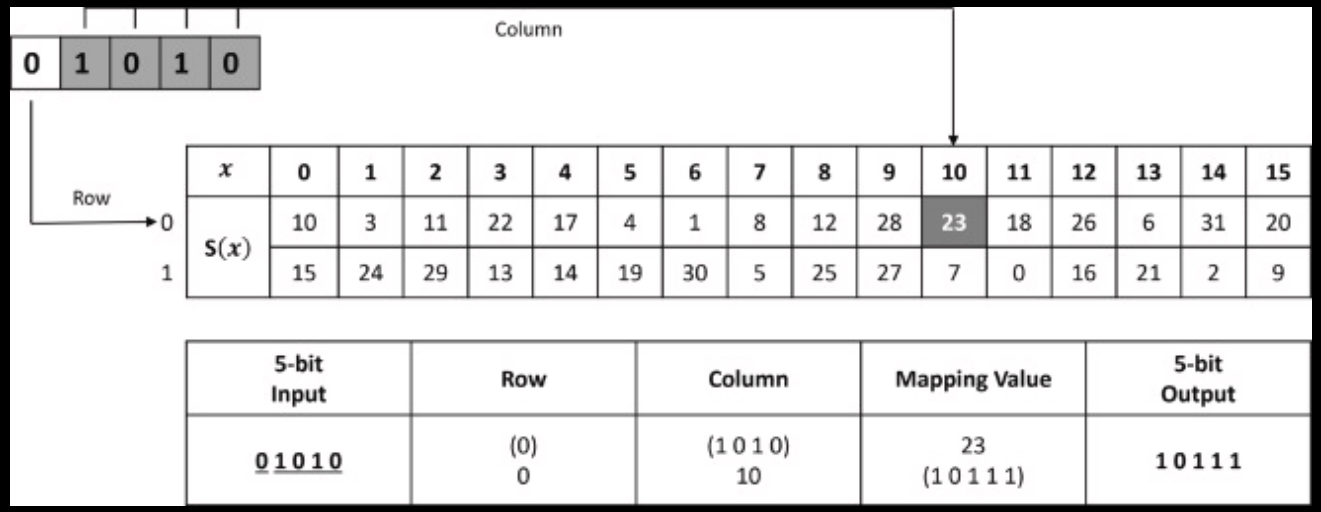
# Constructing Block Ciphers

- A Substitution-Permutation Network (SPN):
  - takes in blocks of plaintext and keys,
  - applies substitution layers (S-boxes) and permutation layers.



# Constructing Block Ciphers

- Substitution boxes are lookup tables that transform chunks of 4 or 8 bits.
  - Example, DES S-Box:



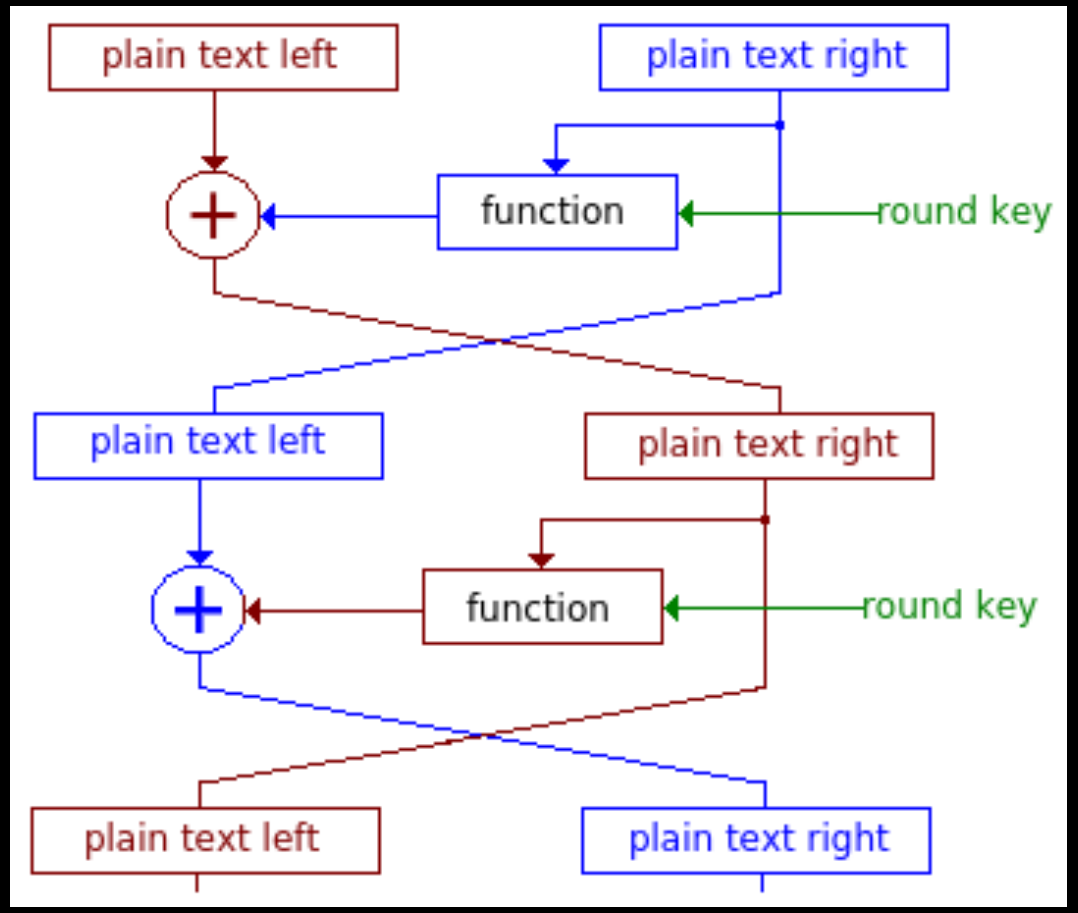

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S(x)$ | 10 | 3 | 11 | 22 | 17 | 4 | 1 | 8 | 12 | 28 | 23 | 18 | 26 | 6 | 31 | 20 |
| | 15 | 24 | 29 | 13 | 14 | 19 | 30 | 5 | 25 | 27 | 7 | 0 | 16 | 21 | 2 | 9 |

| 5-bit Input | Row | Column | Mapping Value | 5-bit Output |
|---|---|---|---|---|
| **0 1 0 1 0** | (0)<br>0 | (1 0 1 0)<br>10 | 23<br>(1 0 1 1 1) | **1 0 1 1 1** |

- S-boxes must be designed carefully:
  - Cryptographically secure
  - No statistical bias

# Constructing Block Ciphers

- Feistel Schemes works as follows:
  1. Split the ciphertext block into two halves, L and R.
  2. Set L to L $\oplus$ F(R), where F is a round function.
  3. Swap the values of L and R.
  4. Go to step 2 and repeat a certain number of times.
  5. Merge L and R into the one output block.



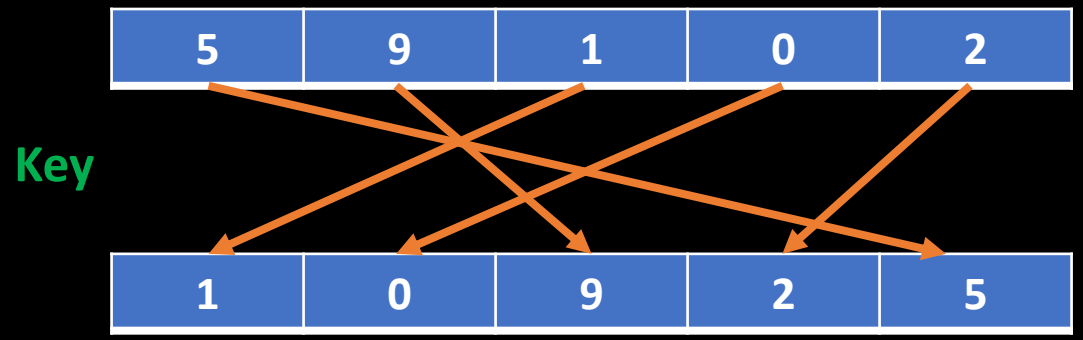
# Constructing Block Ciphers

- In a Feistel scheme, the F function can be:
  - Pseudorandom permutation (PRP)
  - Pseudorandom function (PRF)

| Property | Pseudorandom Permutation (PRP) | Pseudorandom Function (PRF) |
|---|---|---|
| Output | Pseudorandom output | |
| Mapping | Shuffles the data and creates a unique mapping. Must be unique. E.g., PRP(0xAB) ≠ PRP(0xCD) | Transforms the input based on random function. Doesn't have to be unique. E.g., PRF(0xAB) = PRF(0xCD) |
| Invertibility | Always invertible with respect to the key. | Not necessarily invertible. |
| Key | Both use a secret key to transform the plaintext | |

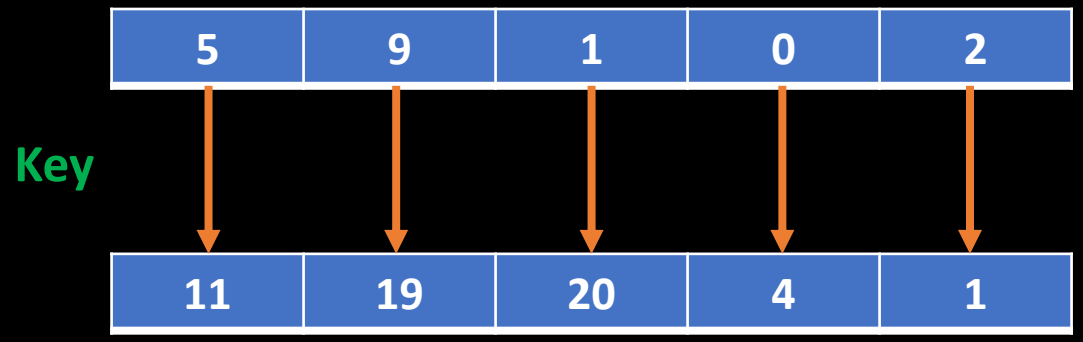
# Constructing Block Ciphers

- In a Feistel scheme, the F function can be:
  - Pseudorandom permutation (PRP) – bijective.



# Constructing Block Ciphers

- In a Feistel scheme, the F function can be:
  - Pseudorandom function (PRF)

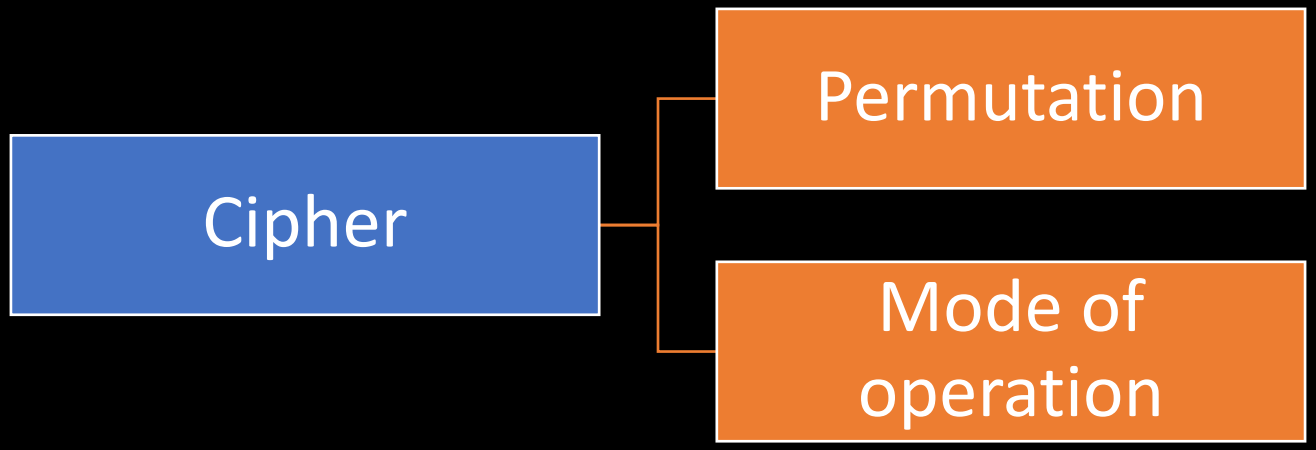


# Content

| Content |
| --- |
| Introduction |
| Constructing Block Ciphers |
| Modes of Operation |
| Advanced Encryption Standard |

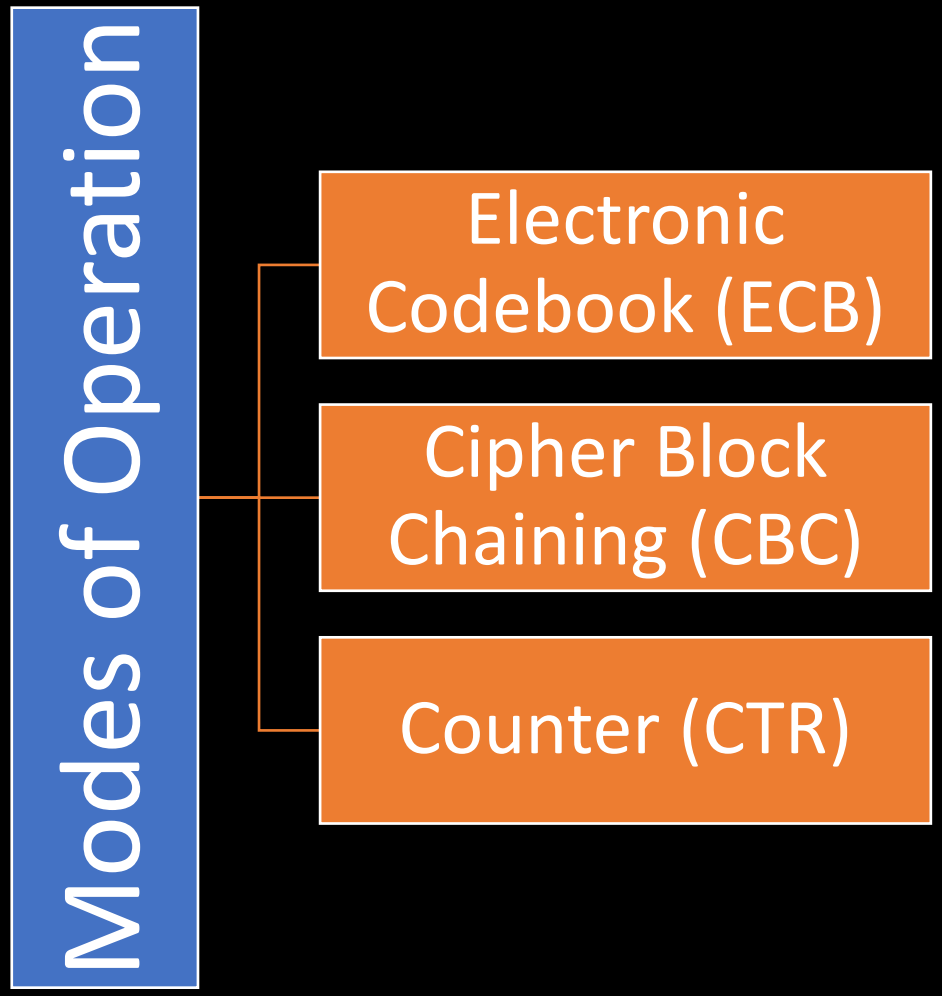
# Modes of Operation

- Encryption schemes combine a permutation with a mode of operation to handle messages of any length.



# Modes of Operation

- Basic modes of operation



# Modes of Operation – ECB

- ECB is the simplest, the dumpiest.

1. Takes plaintext blocks $P_1, P_2, \ldots, P_N$
2. Processes each independently by computing $C_1 = E(K, P_1), C_2 = E(K, P_2)$

- Insecure mode.

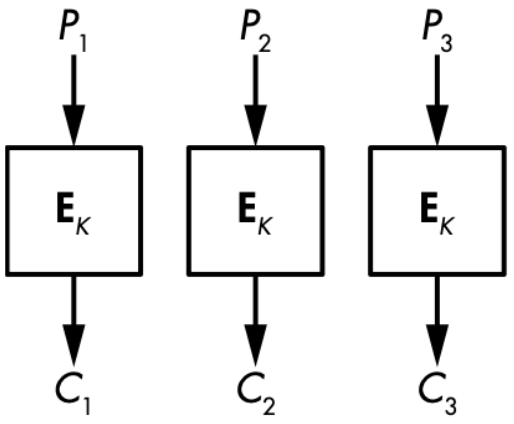



Figure 4-6: The ECB mode

# Modes of Operation – ECB

- ECB is not semantically secure
  - **Identical plaintext** blocks results in **identical ciphertext** blocks.

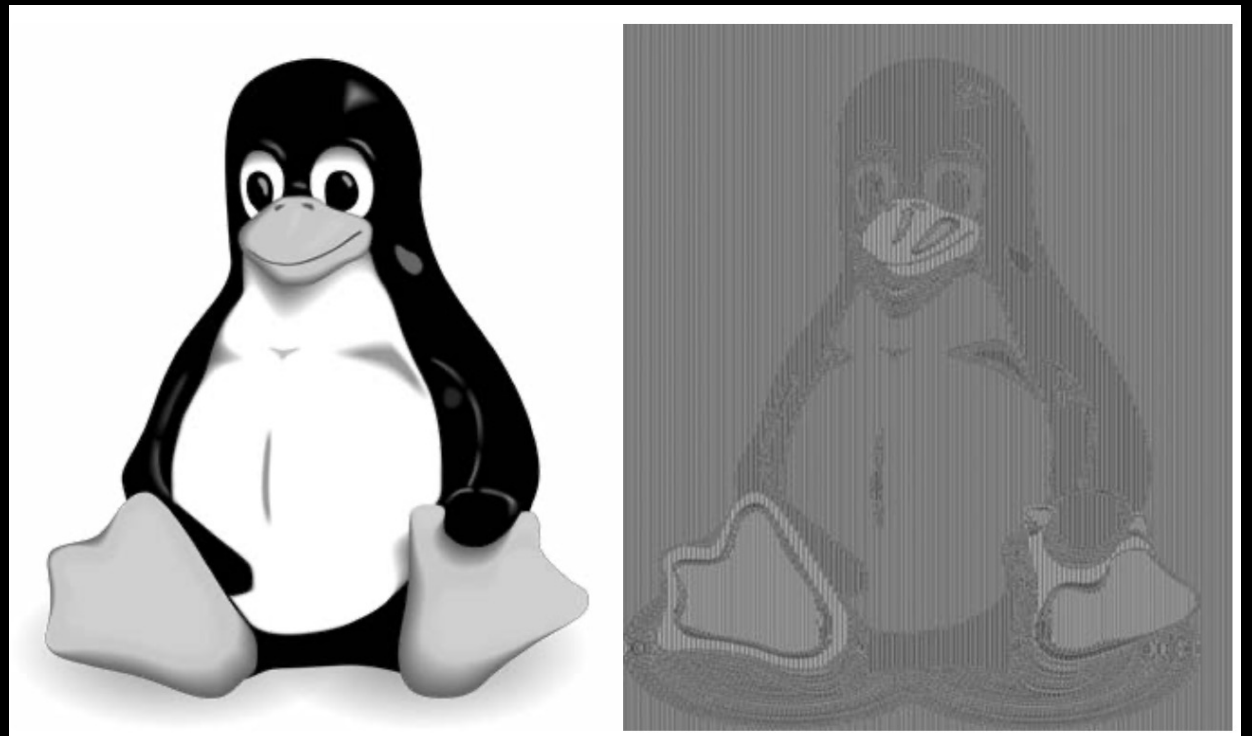


*Figure 4-7: The original image (left) and the ECB-encrypted image (right)*

# Modes of Operation – CBC

- CBC makes each ciphertext block dependent on all the previous blocks.
  - Ensures that identical plaintext blocks won't be identical ciphertext blocks.

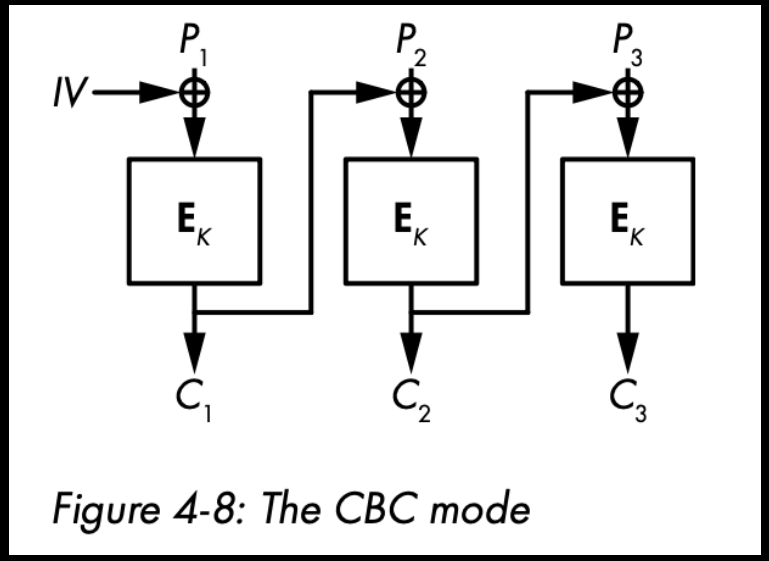
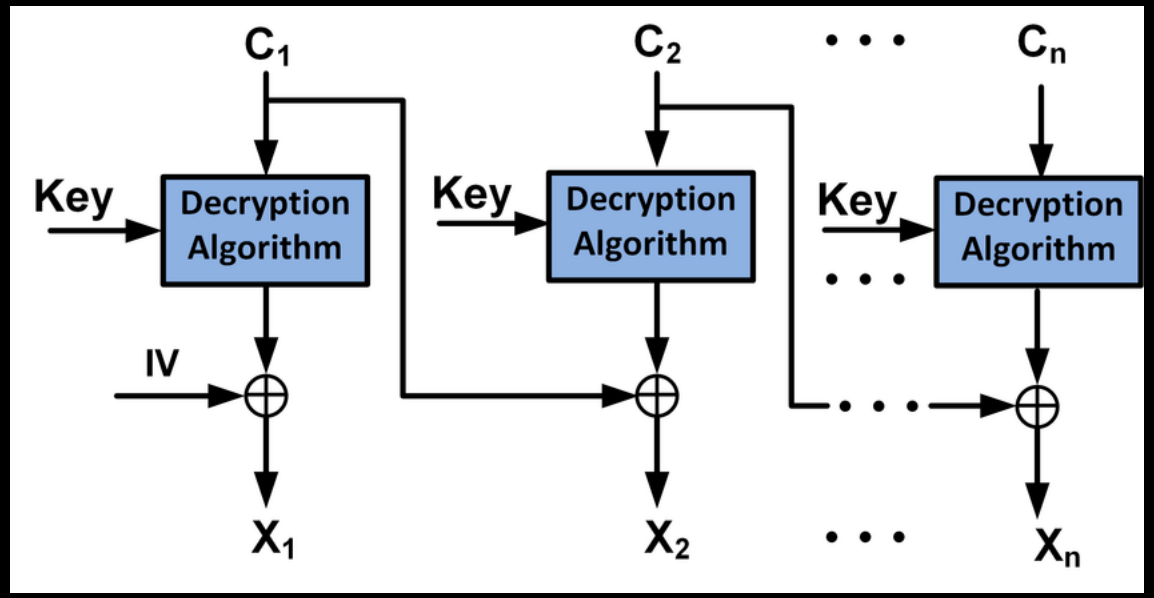


Figure 4-8: The CBC mode

- $C_i = E(K, P_i \oplus C_{i-1})$
- When encrypting the first block, $P_1$, CBC takes a random initial value (IV)

# Modes of Operation – CBC

- Decryption needs to know the IV used to encrypt, so it's sent along with the ciphertext, in the clear.
- Decryption can be parallelized.



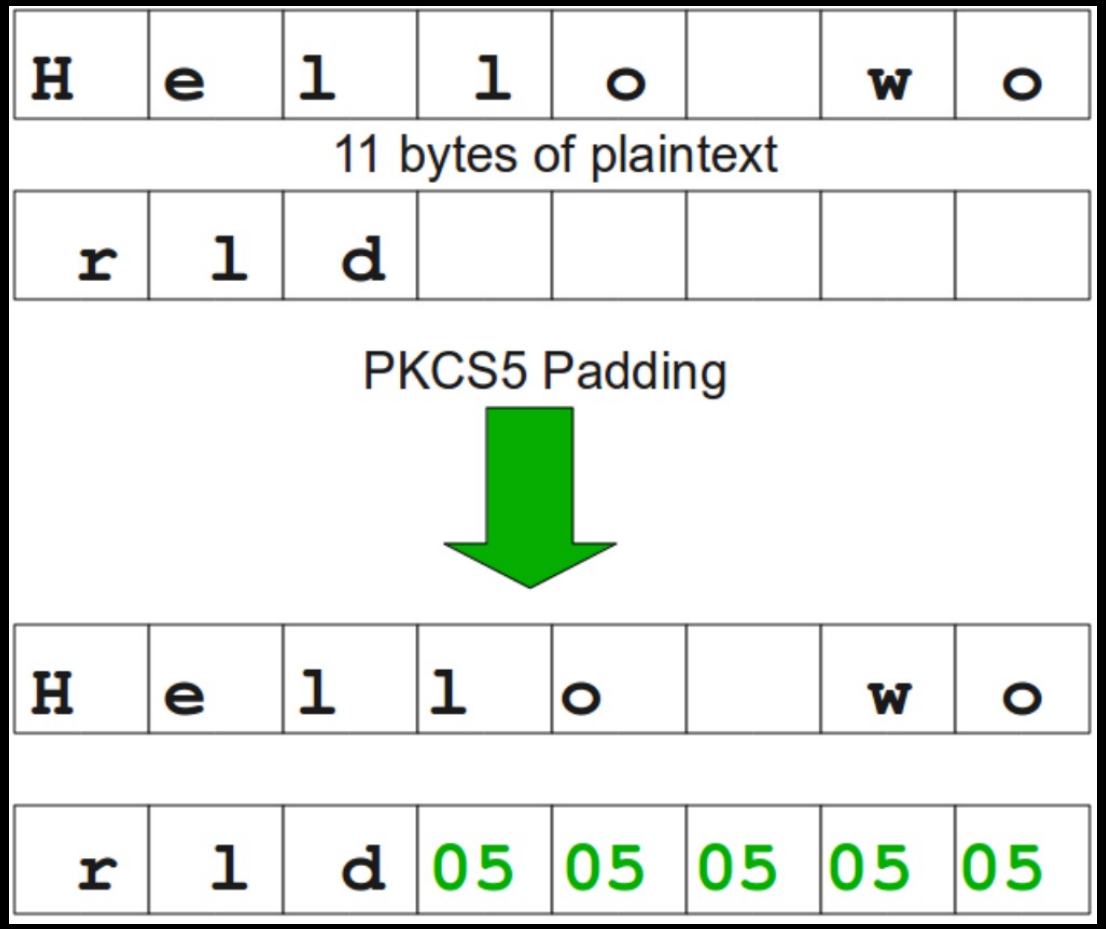
# Modes of Operation – CBC

- Two ways to align the plaintext to the block size in CBC mode.

| Padding | Ciphertext stealing |
| --- | --- |
| • Add extra bytes to fill the last block<br>• $01, 02\ 02, 03\ 03\ 03, \ldots$, fifteen $0f$<br>• If ptxt aligns to blocks, pad with sixteen 10<br>• Vulnerable to padding oracle attack<br>• Increase the ciphertext size | • Extend the last block with bits from the previous ciphertext block, and then encrypts the resulting block.<br>• No increase in ctxt size<br>• Not vulnerable to padding oracle attack<br>• NIST has three implementation variants of the CBC-CS mode |

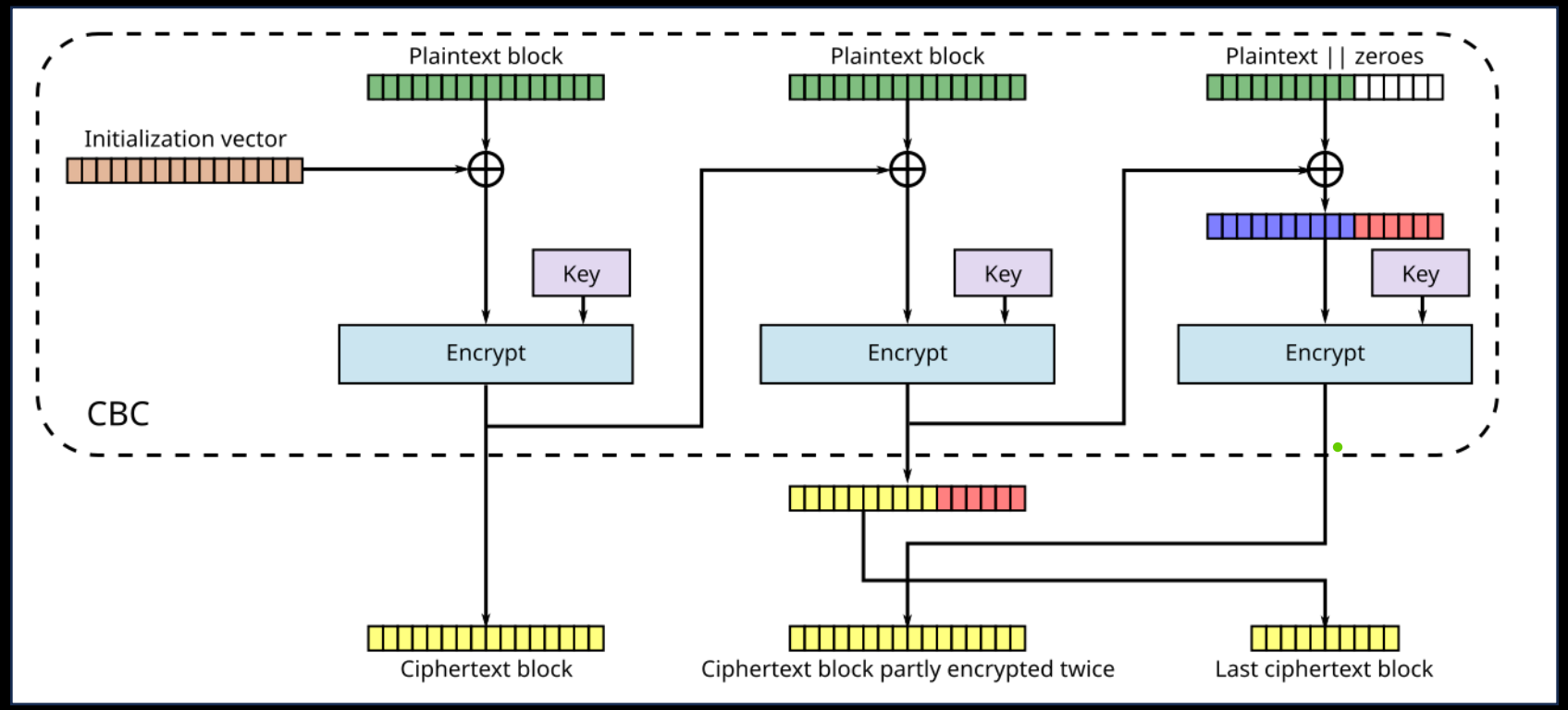
# Modes of Operation – CBC

- Padding



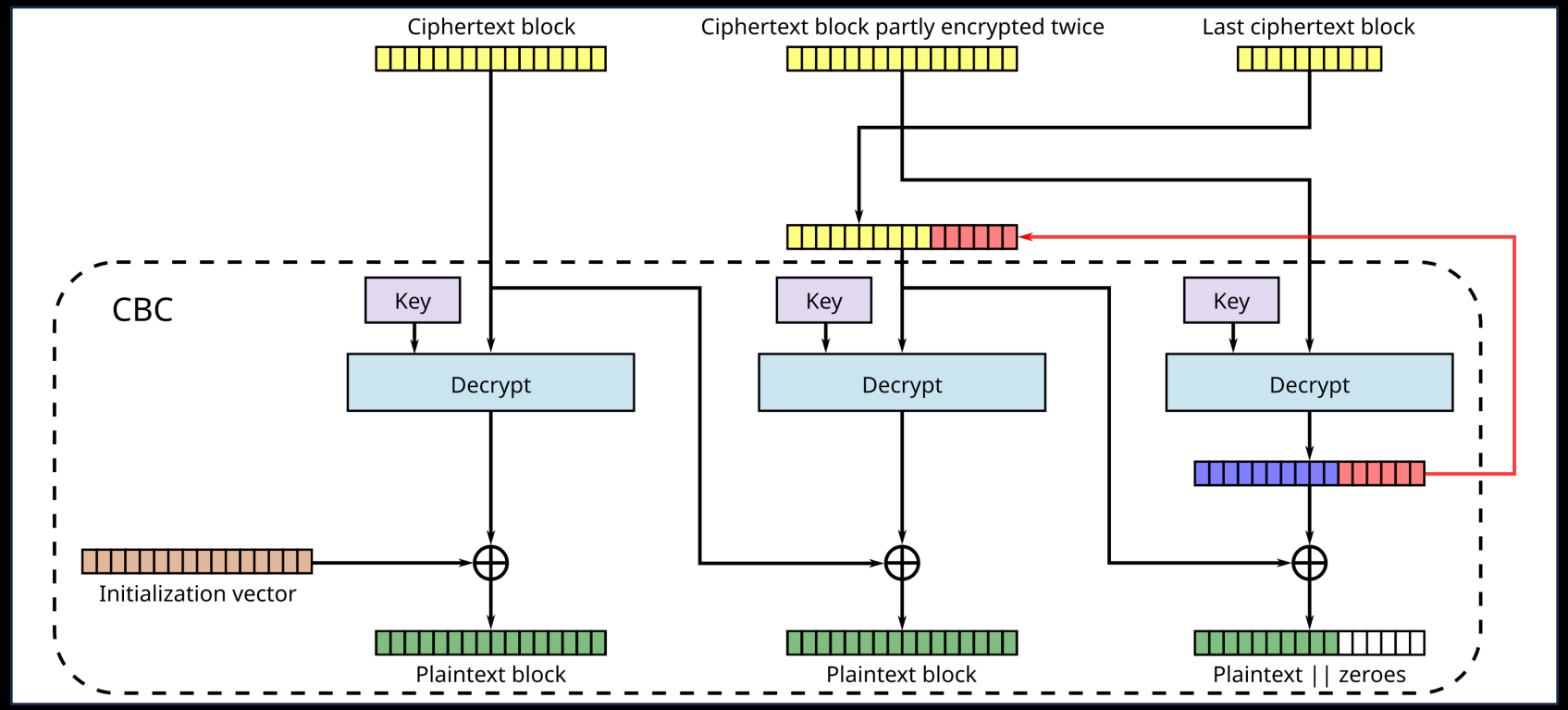
# Modes of Operation – CBC

- Encryption in CBC-CS



# Modes of Operation – CBC

- Decryption in CBC-CS



# Modes of Operation – CBC

- Example: assume we are encrypting ptxt="ABCD", the block size is 3 bytes, IV=123, and the key is $K$.

# Modes of Operation – CBC

ABC

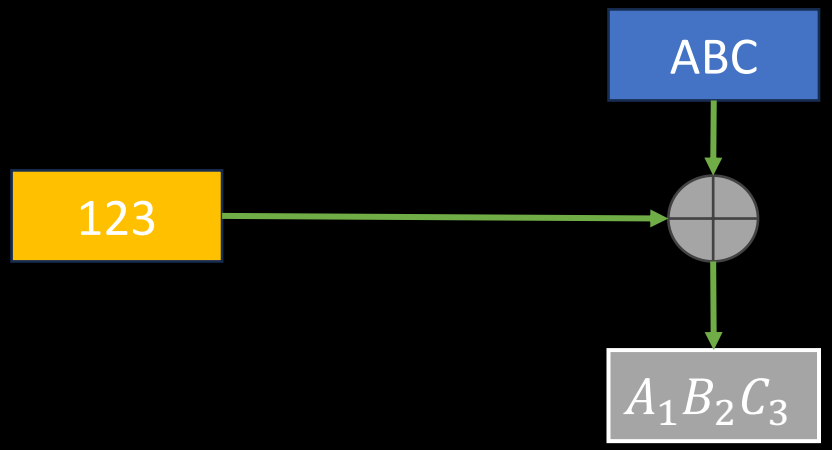
D

# Modes of Operation – CBC

ABC

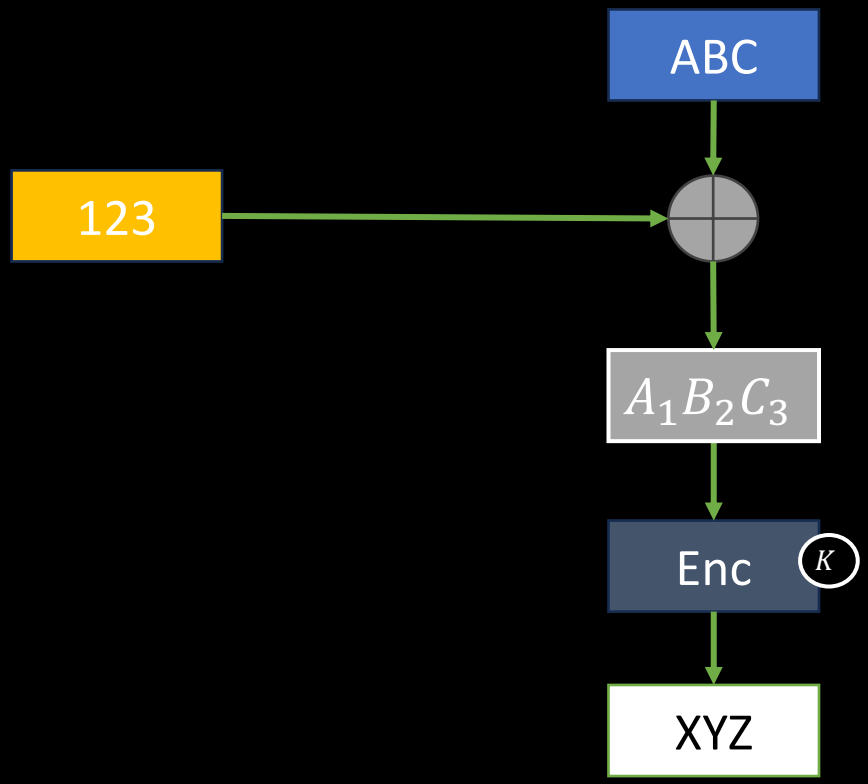
D00

# Modes of Operation – CBC



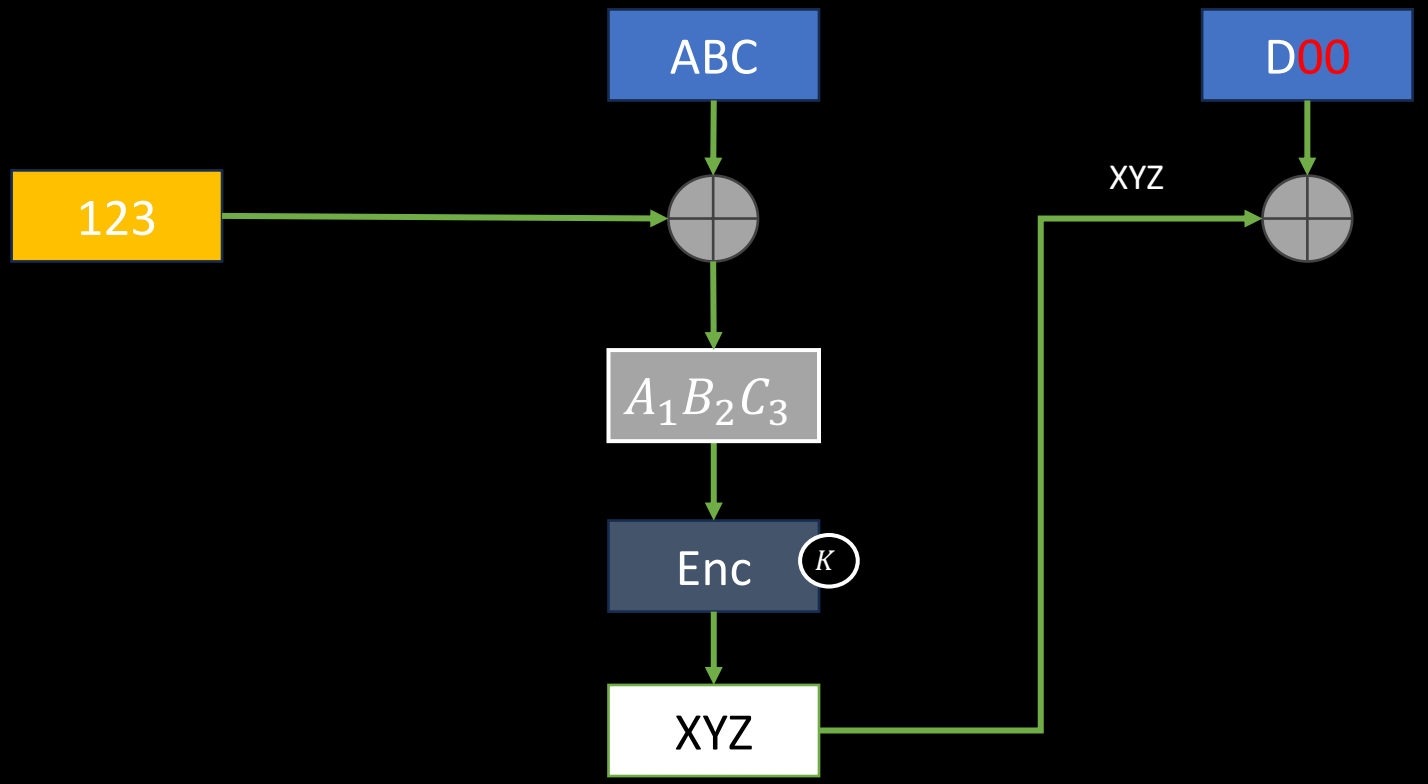

D00

# Modes of Operation – CBC

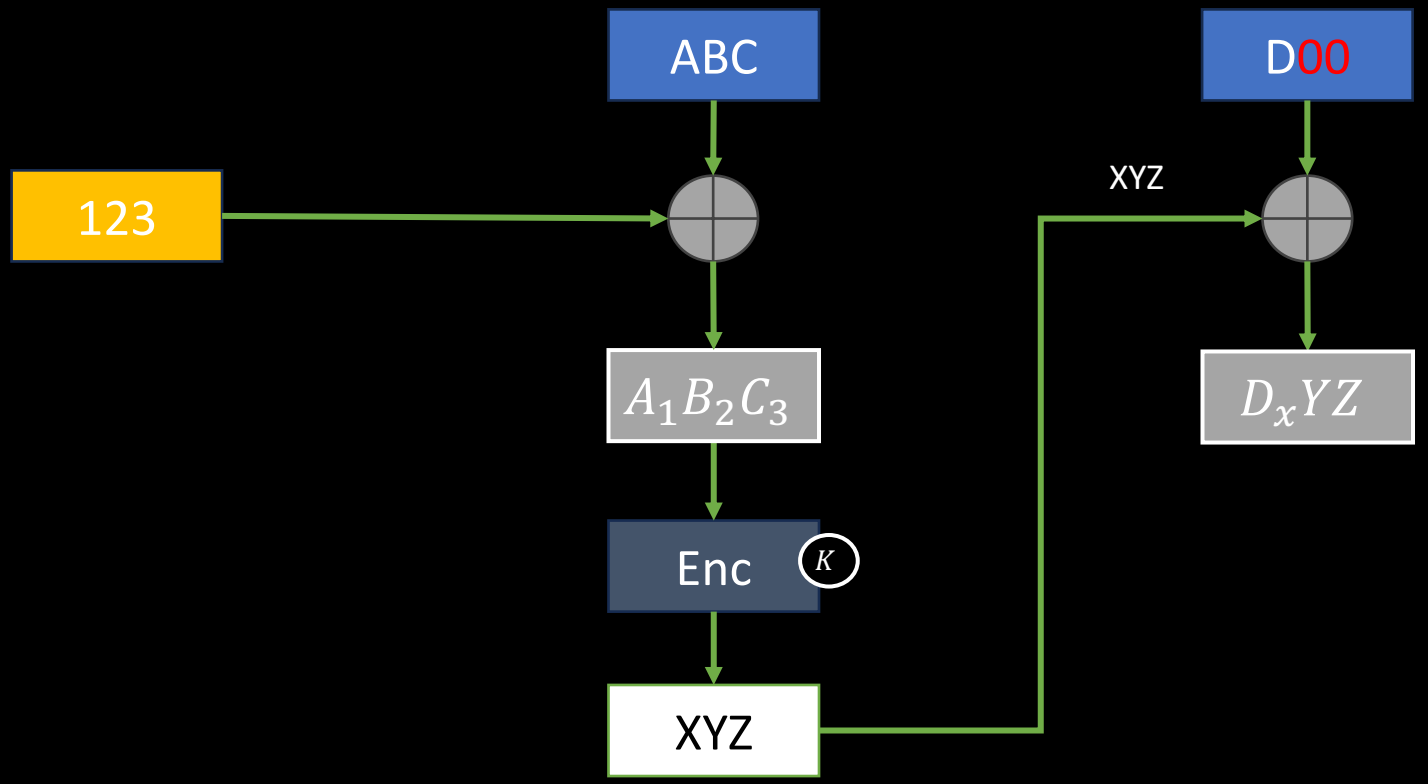



D00

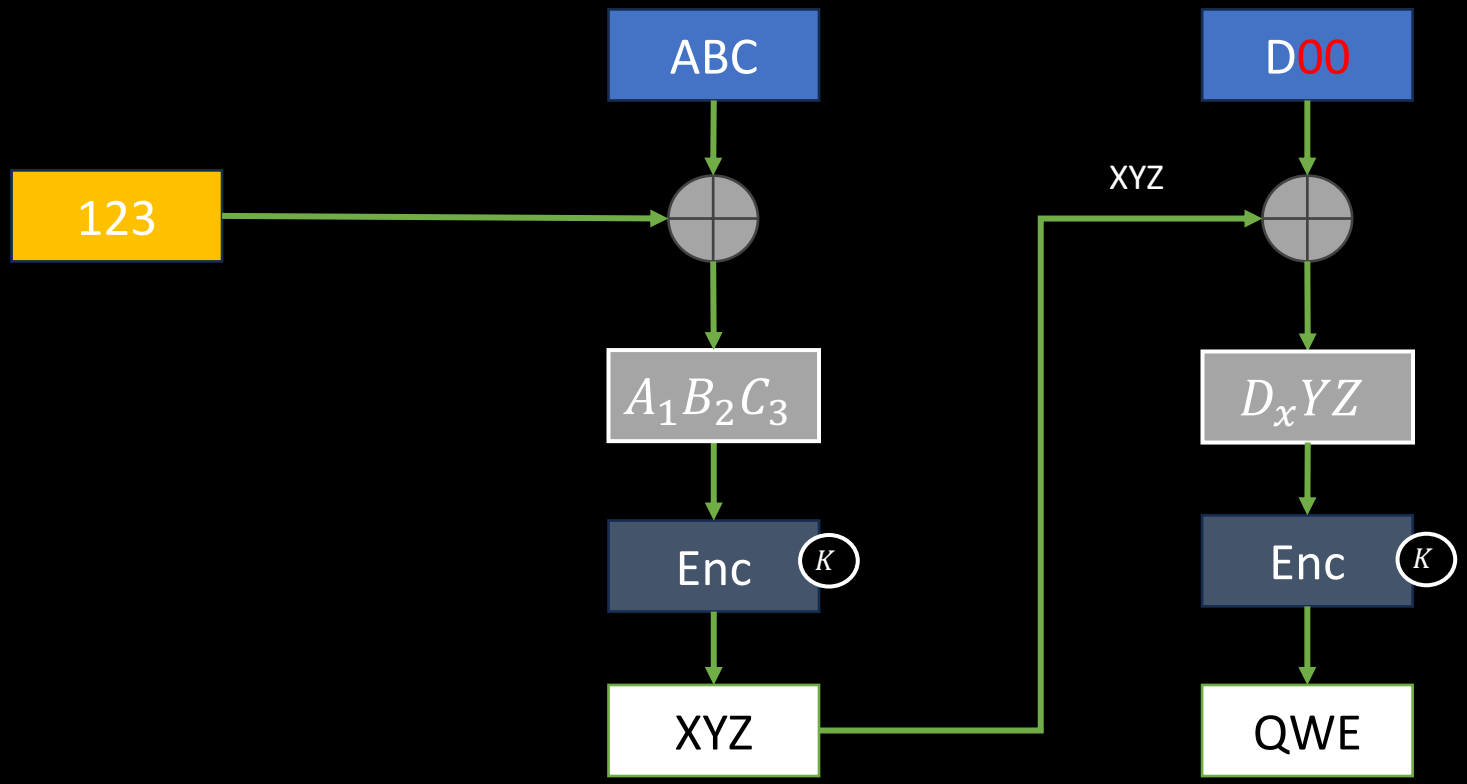
# Modes of Operation – CBC



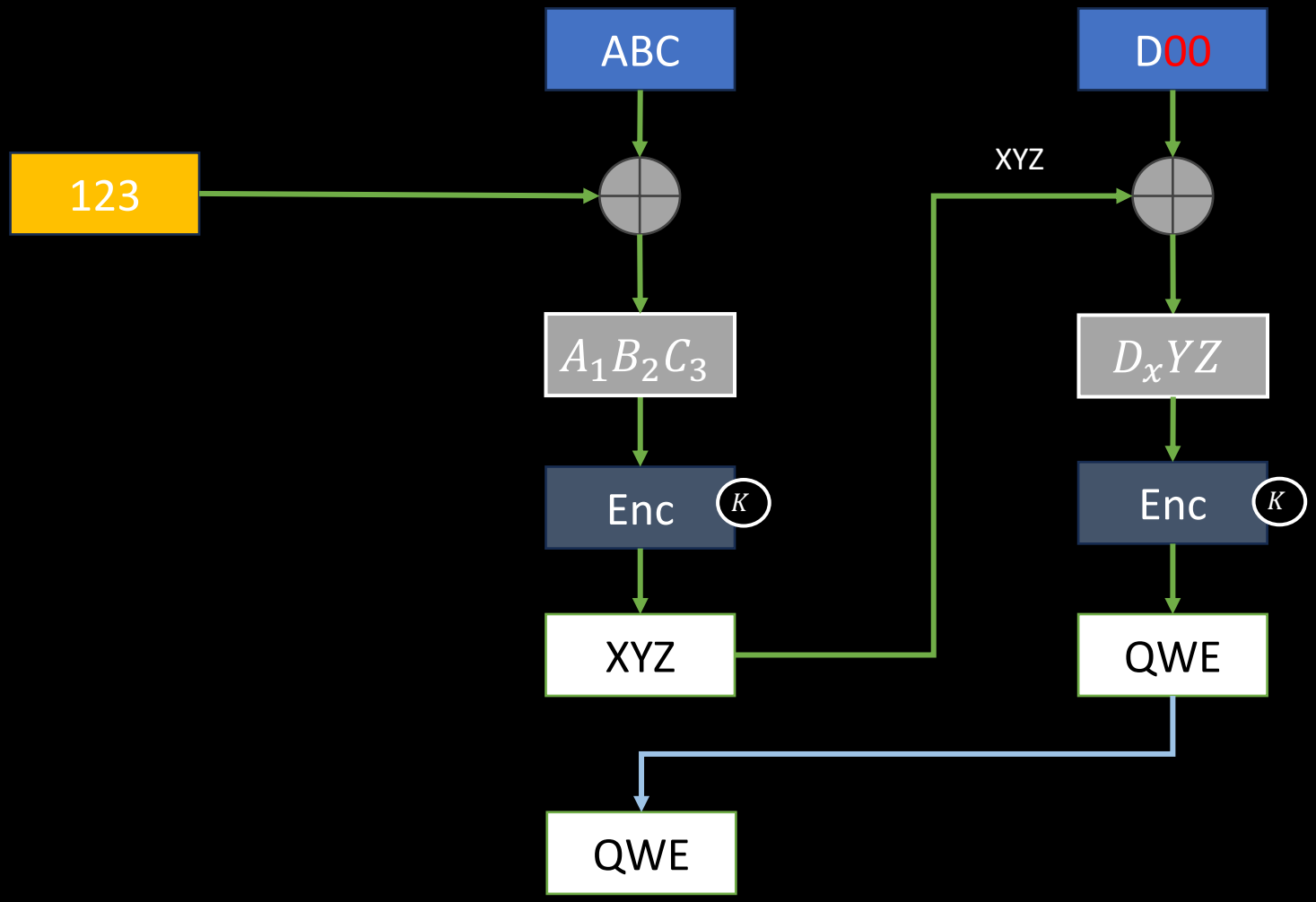
# Modes of Operation – CBC



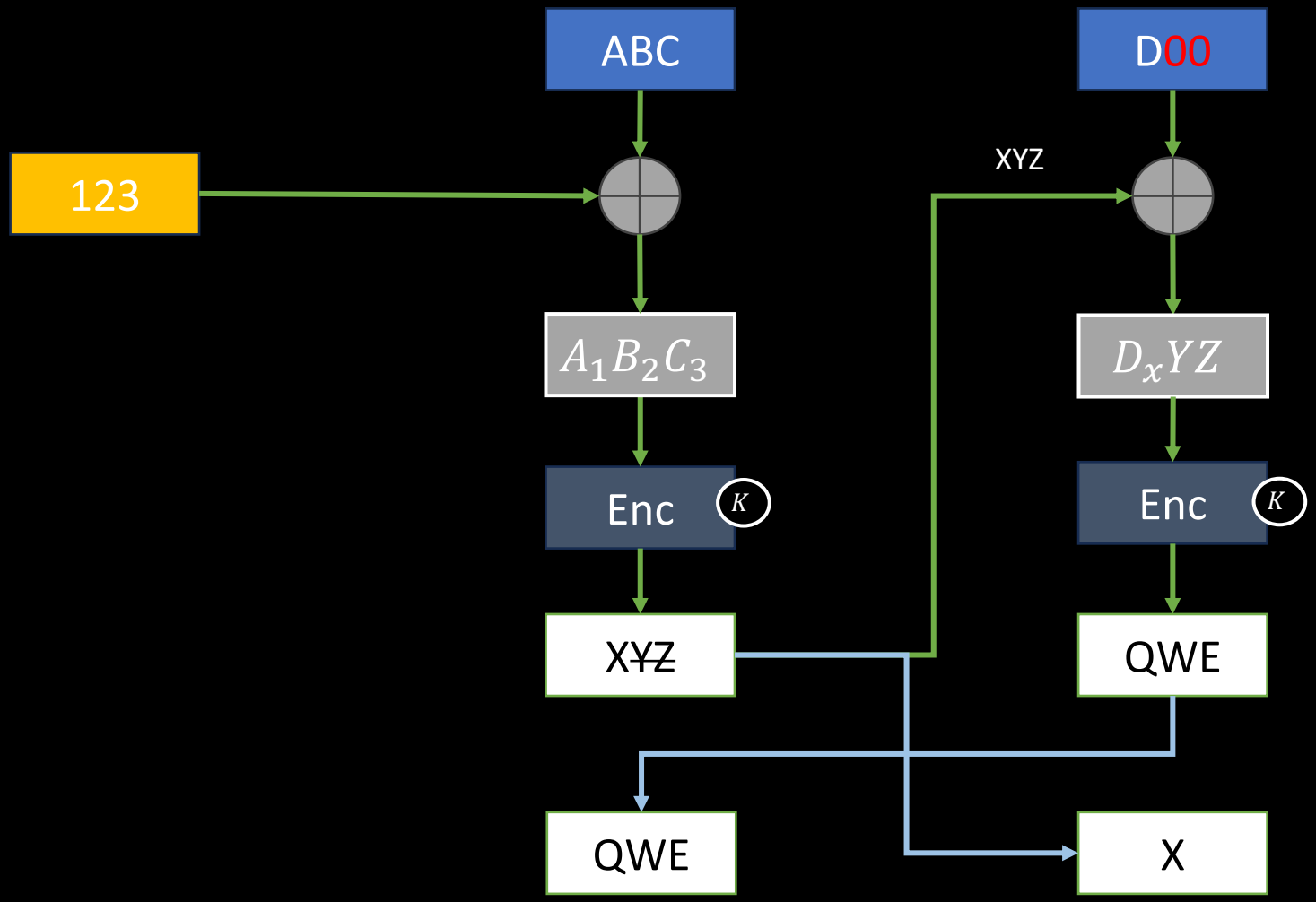
# Modes of Operation – CBC



# Modes of Operation – CBC



# Modes of Operation – CBC

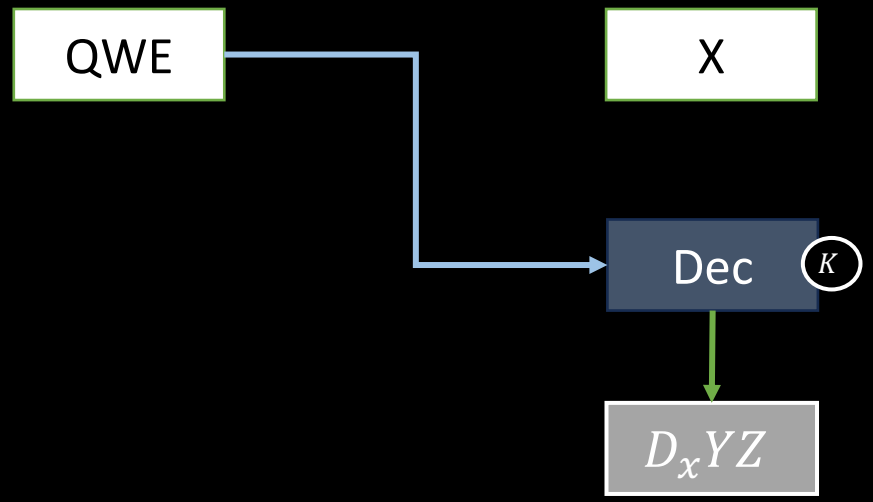


# Modes of Operation – CBC

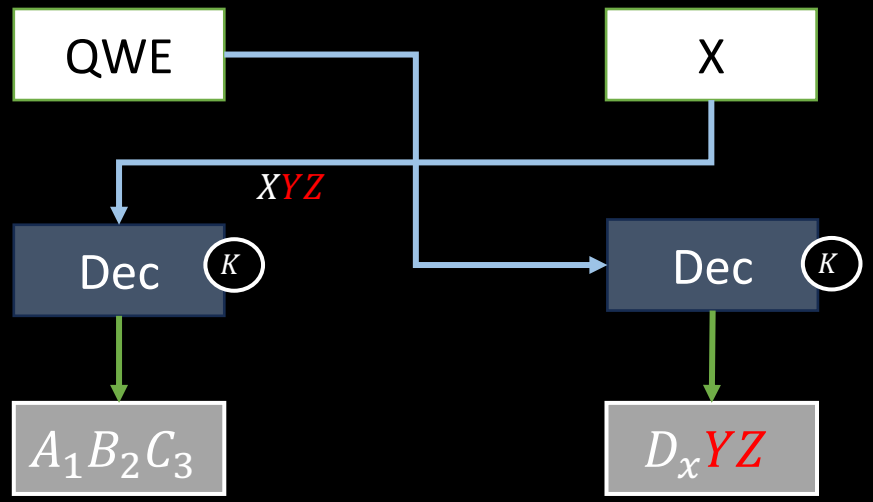
QWE

X

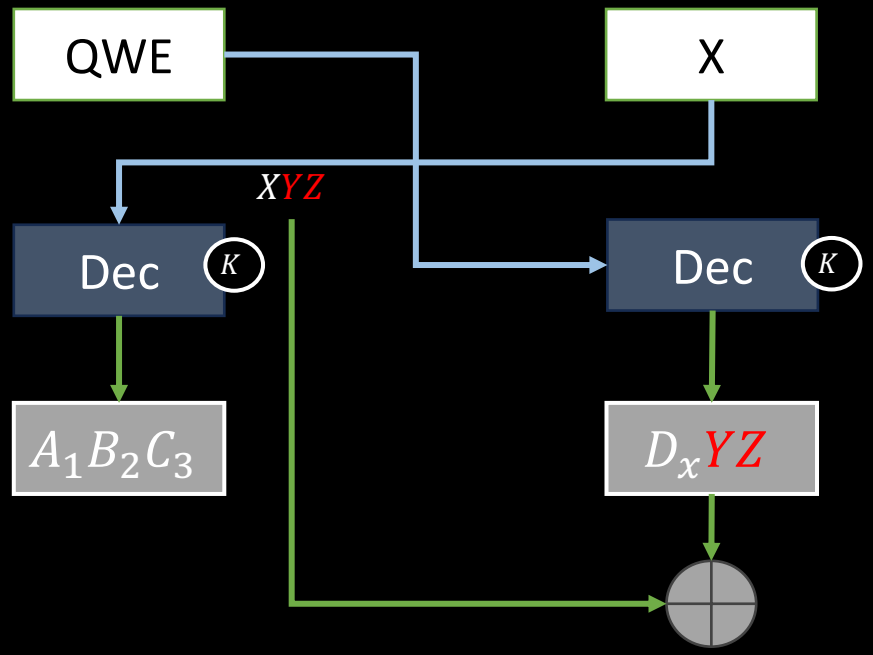
# Modes of Operation – CBC



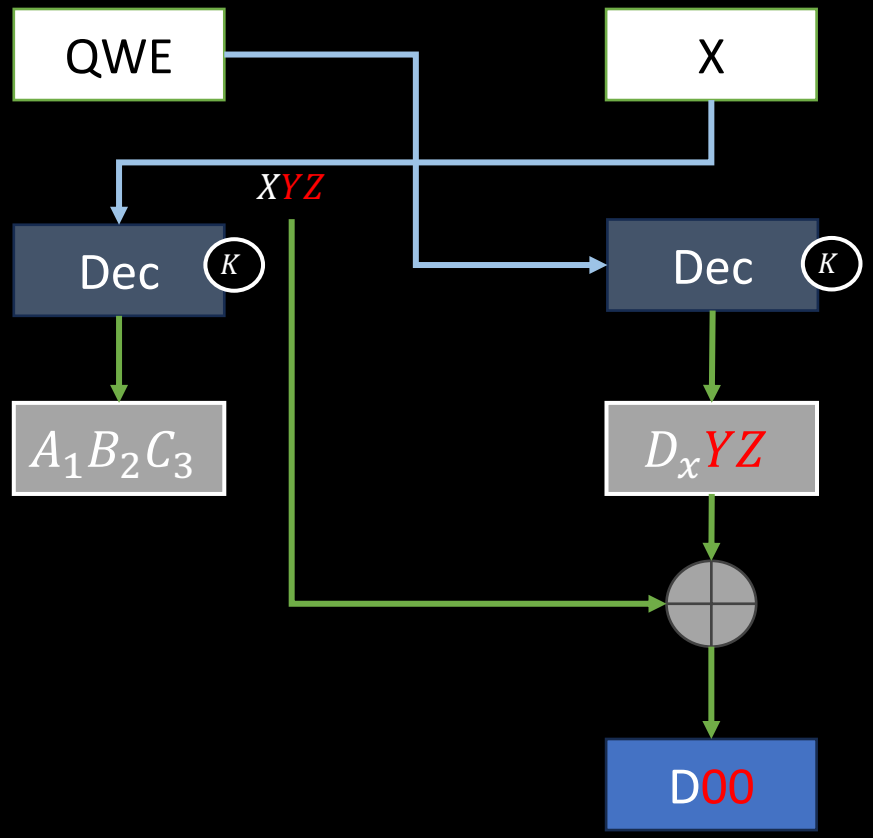
# Modes of Operation – CBC



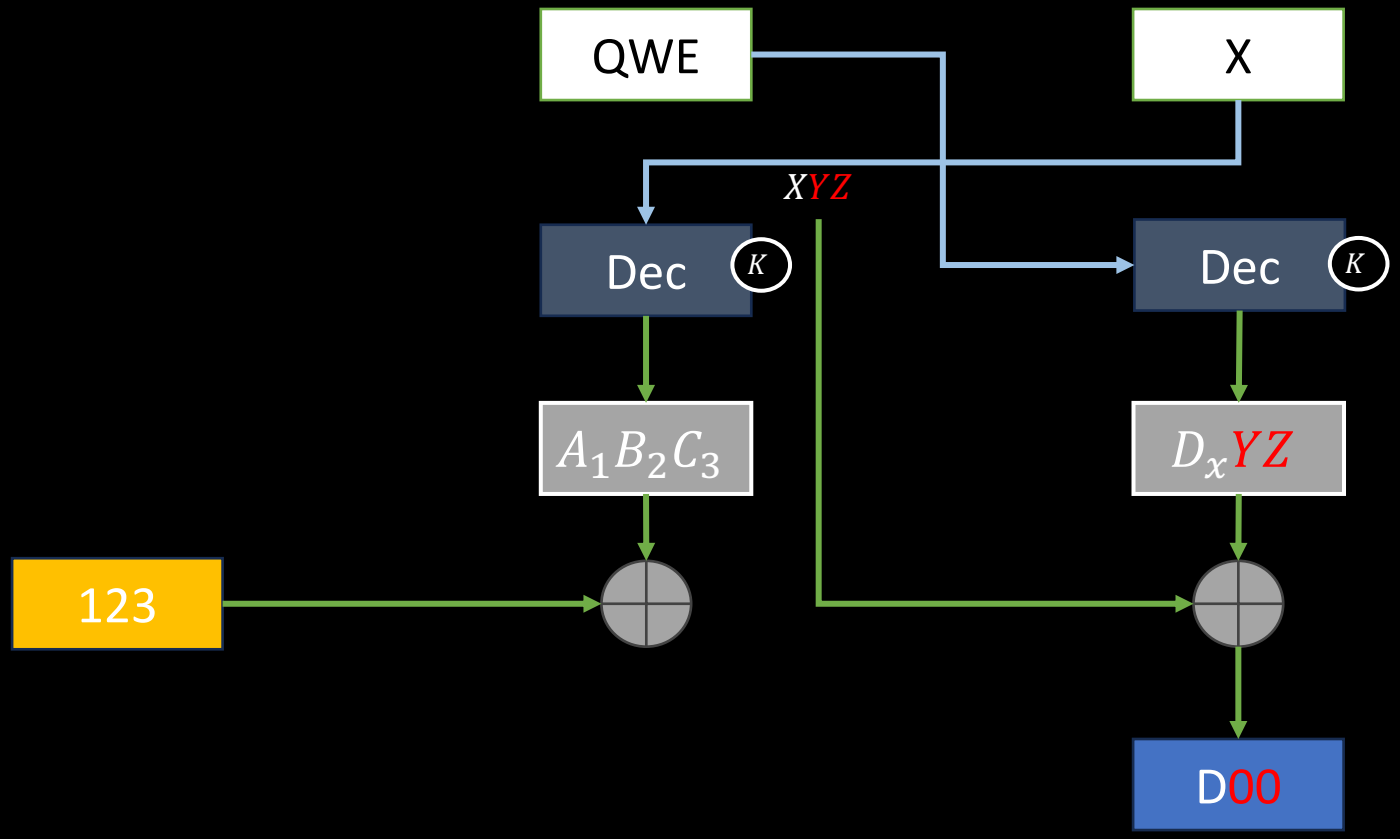
# Modes of Operation – CBC



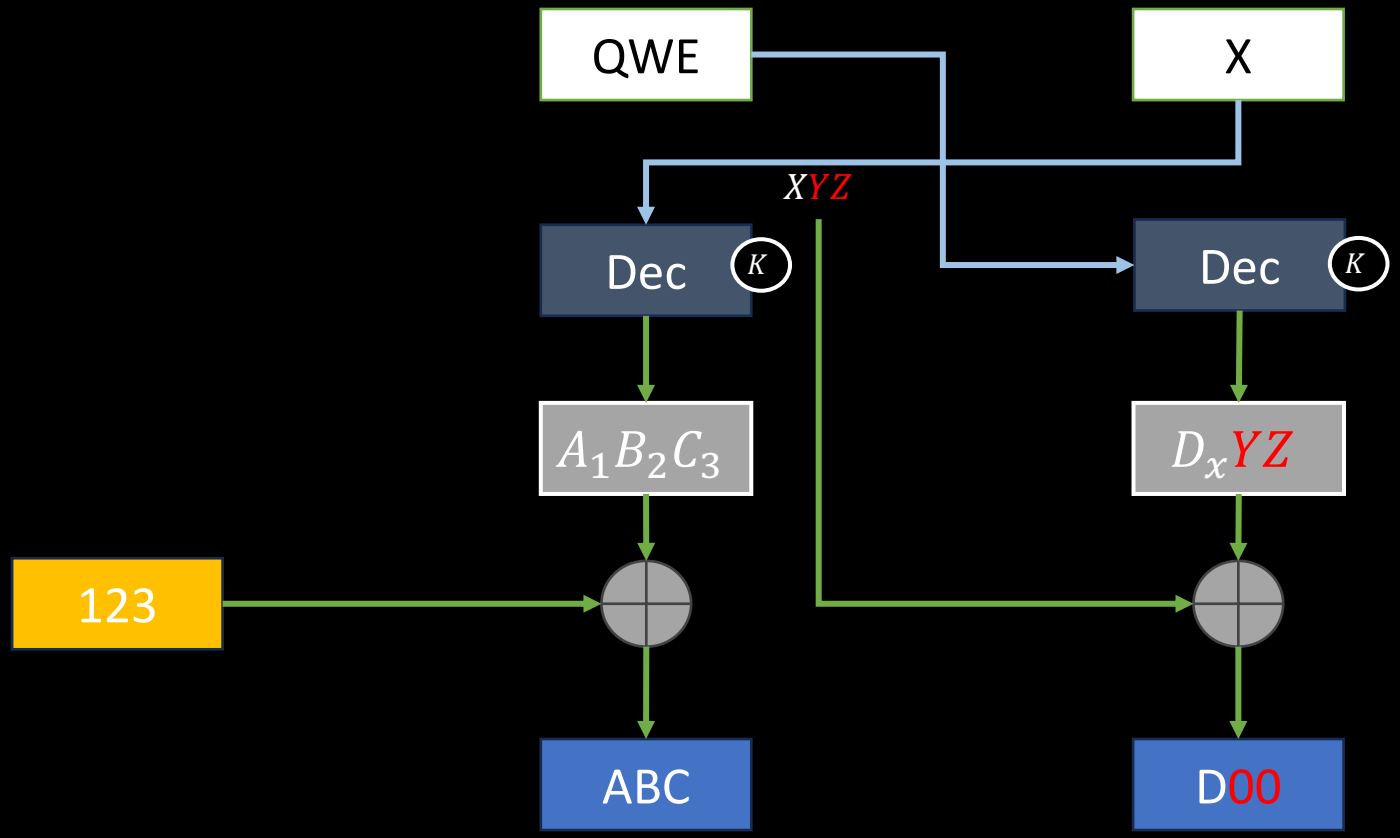
# Modes of Operation – CBC



# Modes of Operation – CBC



# Modes of Operation – CBC



# Modes of Operation – CTR

- In CTR mode, a cipher encrypts blocks composed of a **counter** and a **nonce**.
  - **Counter**: an integer incremented for each block.
    - Blocks in a message use different counters.
    - Different messages can use the same counter sequence (1, 2, 3, …).
  - **Nonce**: a number used only once.
    - The same for all blocks in a single message.
    - No two messages should use the same nonce.
    - not secret.
- The encryption of $Nonce||CTR$ is XORed with the plaintext.

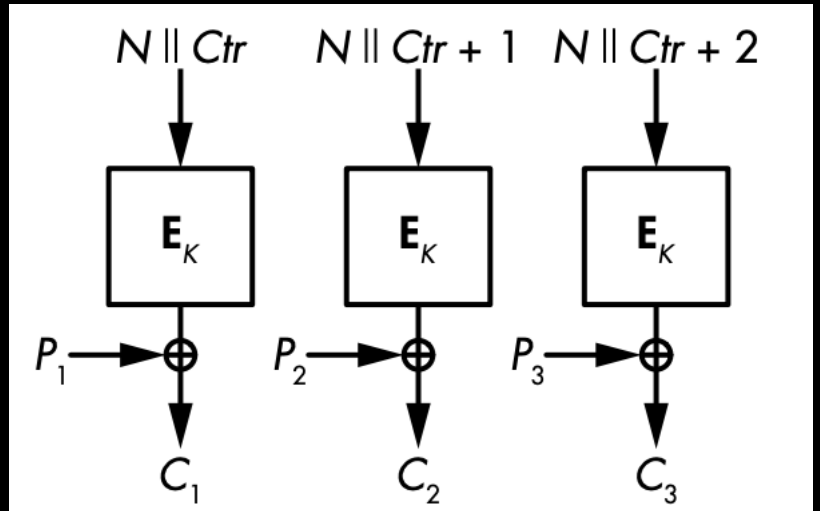


Figure 4-10: The CTR mode

# Modes of Operation – CTR

- Nonces must be unique, but not necessarily random.

- To ensure uniqueness:
  - Some techniques include a timestamp within a random nonce.
  - Update the counter for every plaintext you encrypt.

- Advantage: faster than any mode.
  - More than parallelizable.
  - Start encrypting before knowing the message by picking a nonce and computing the stream that you'll later XOR with the plaintext.

# Content

| Content |
| --- |
| Introduction |
| Constructing Block Ciphers |
| Modes of Operation |
| Advanced Encryption Standard |

# Advanced Encryption Standard

- AES is a symmetric block cipher.
  - Fixed block size, 128 bits.
  - Key size: 128 (most common), 192, 256 bits.
- It processes a 16-byte block as a 2D array.
- Transforms the bytes, rows, and columns to produce the final ciphertext

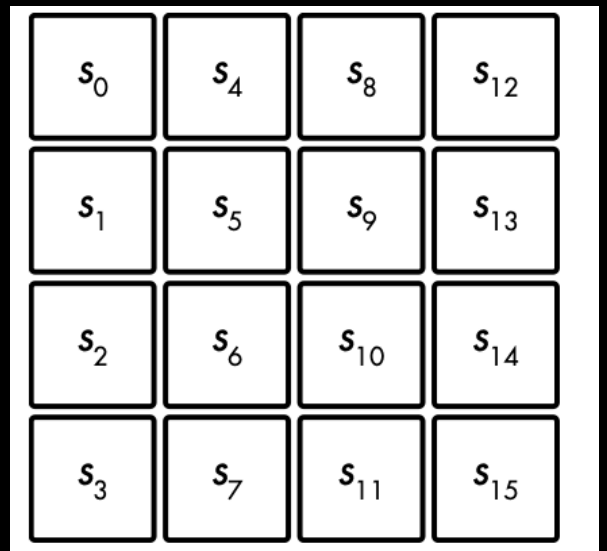
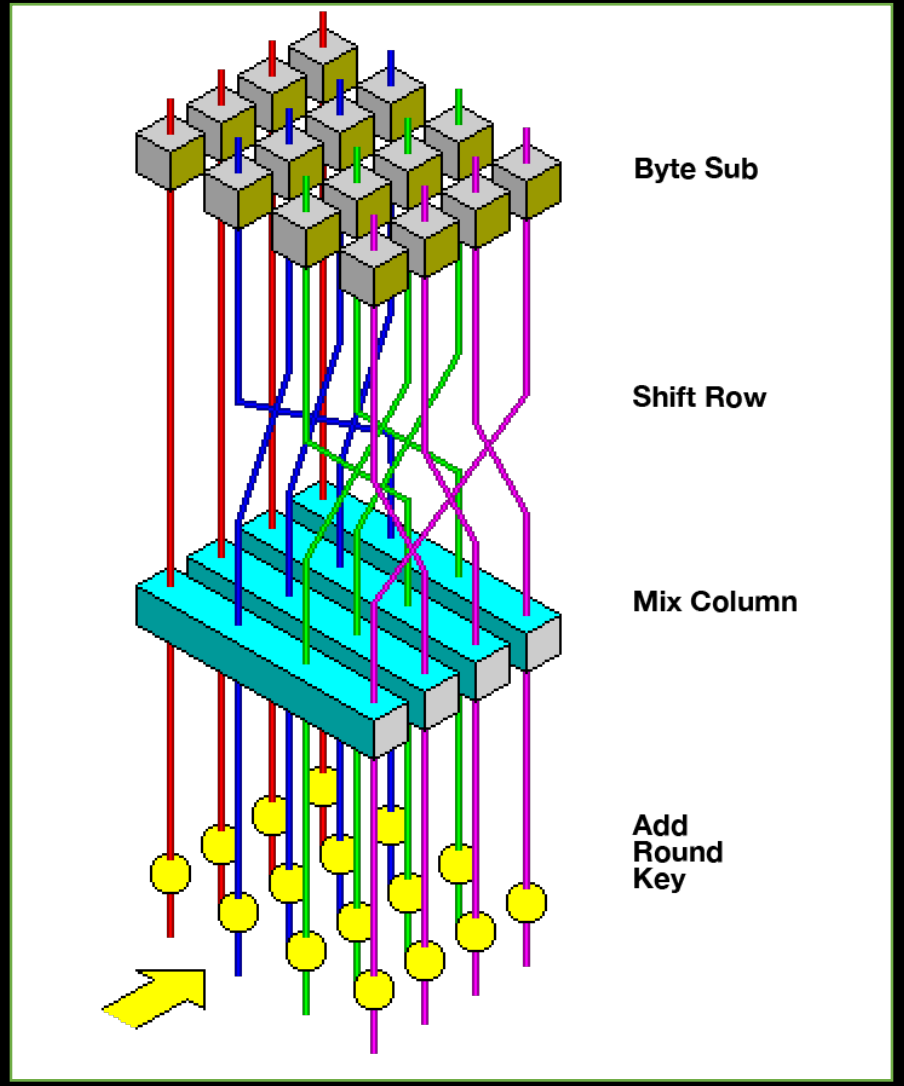



*Figure 4-3: The internal state of AES viewed as a 4 × 4 array of 16 bytes*

# Advanced Encryption Standard

- AES uses an SPN structure.
  - 128-bit key → 10 rounds.
  - 192-bit key → 12 rounds.
  - 256-bit key → 14 rounds.
- Each round consists of:
  - **SubBytes**: Replaces each byte (s0, s1, . . . , s15) with another byte according to an S-box.
  - **ShiftRows**: shift the last three row cyclically a certain number of steps.
  - **MixCols**: a linear mixing operations on the columns of the state.
  - **AddRoundKey**: XORs a round key to the internal state.



# Advanced Encryption Standard

- In a full-AES implementation:
  - We start by **AddRoundKey** with the ptxt
  - The last round does not include **MixColumns**.
- Substitution layers → **SubBytes** operations.
- Permutation layers → **ShiftRows** + **MixColumns** operations.

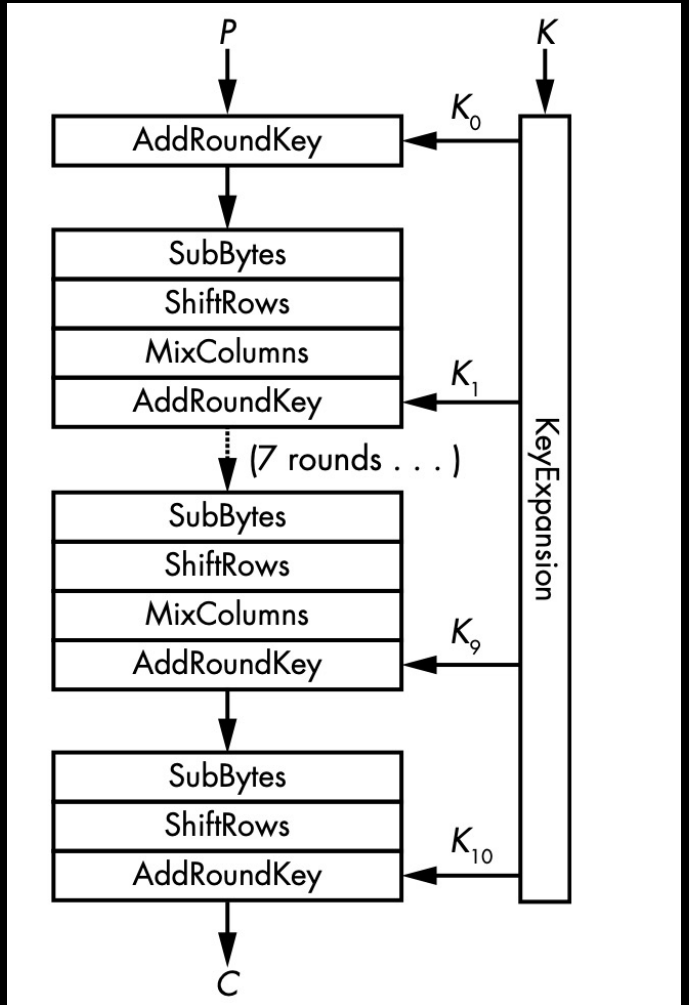



*Figure 4-4: The internal operations of AES*

# Advanced Encryption Standard

**KeyExpansion**: the key schedule algorithm.

- Takes the master 16-bytes key
- Creates the round keys.
  - Each subkey is 16 bytes.
- **Drawback**: given one of the round keys, an attacker can recover other round keys and the master key.
  - A single round key can be recovered through a side-channel attack.

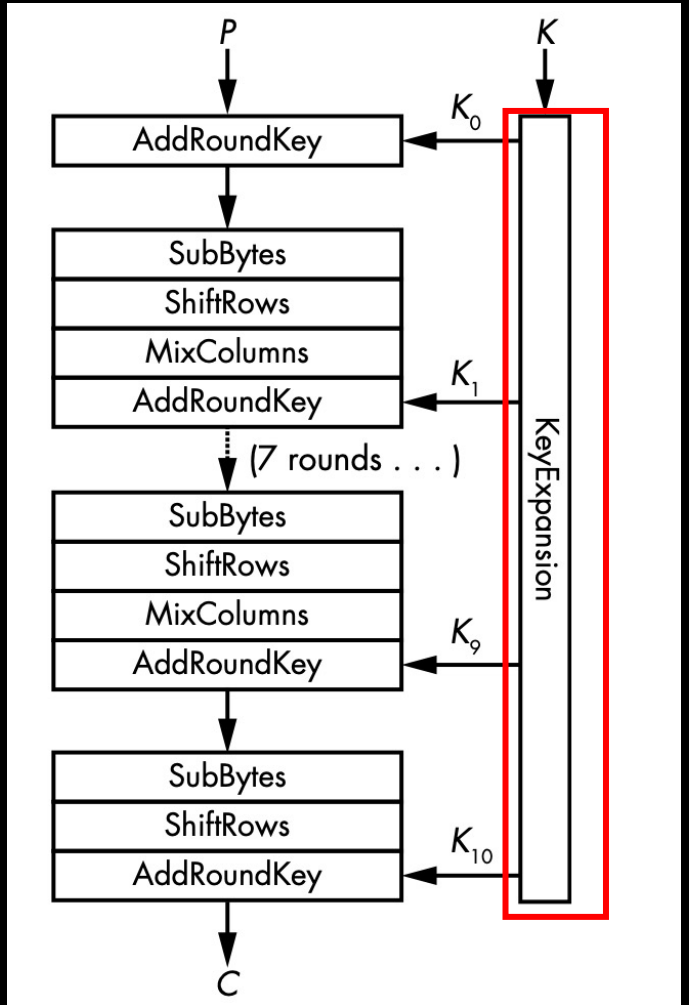
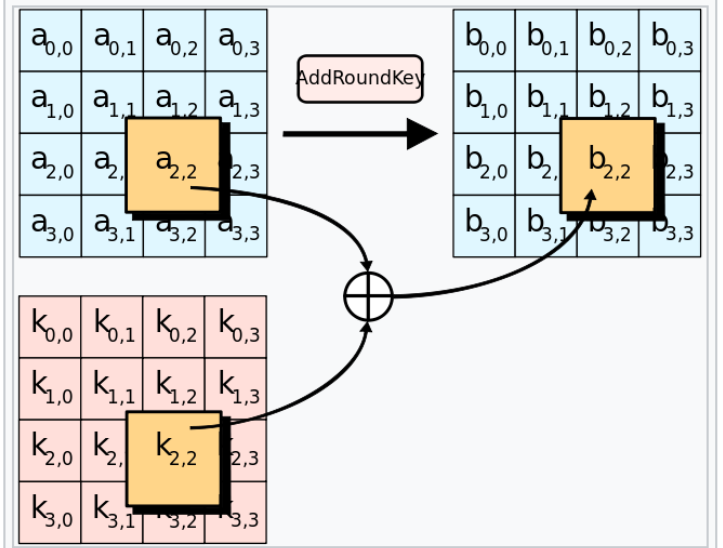



*Figure 4-4: The internal operations of AES*

# Advanced Encryption Standard

**AddRoundKey**

- Combine the key with the internal state.
- XORs each byte from the key with a byte from the internal state.




In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XOR operation (⊕).

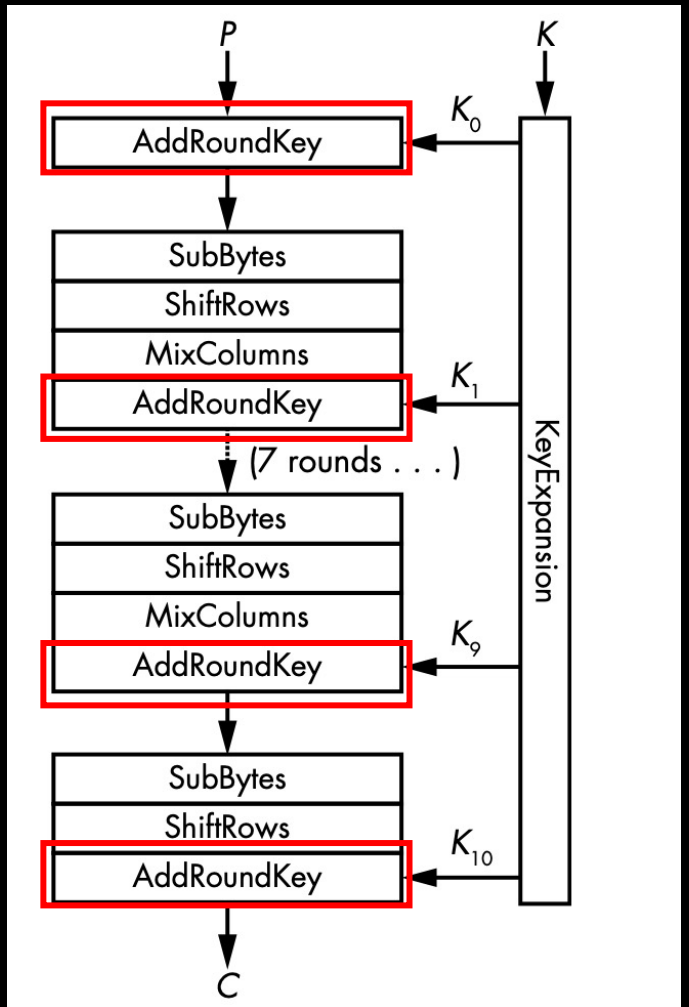
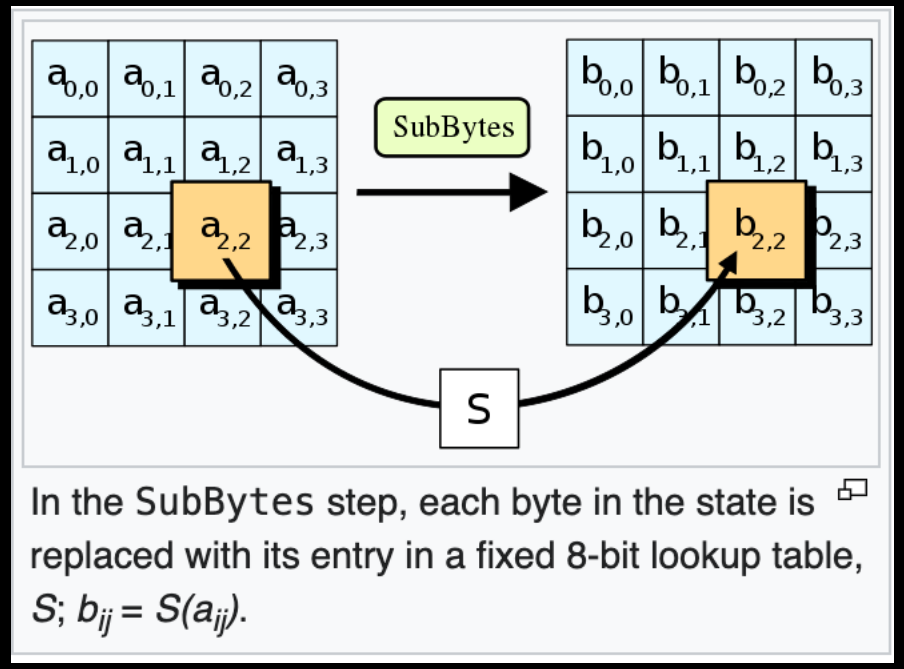



*Figure 4-4: The internal operations of AES*

# Advanced Encryption Standard

**SubBytes**

- Each byte in the state array is replaced with another byte using an 8-bit substitution box.




In the SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, $S$; $b_{ij} = S(a_{ij})$.

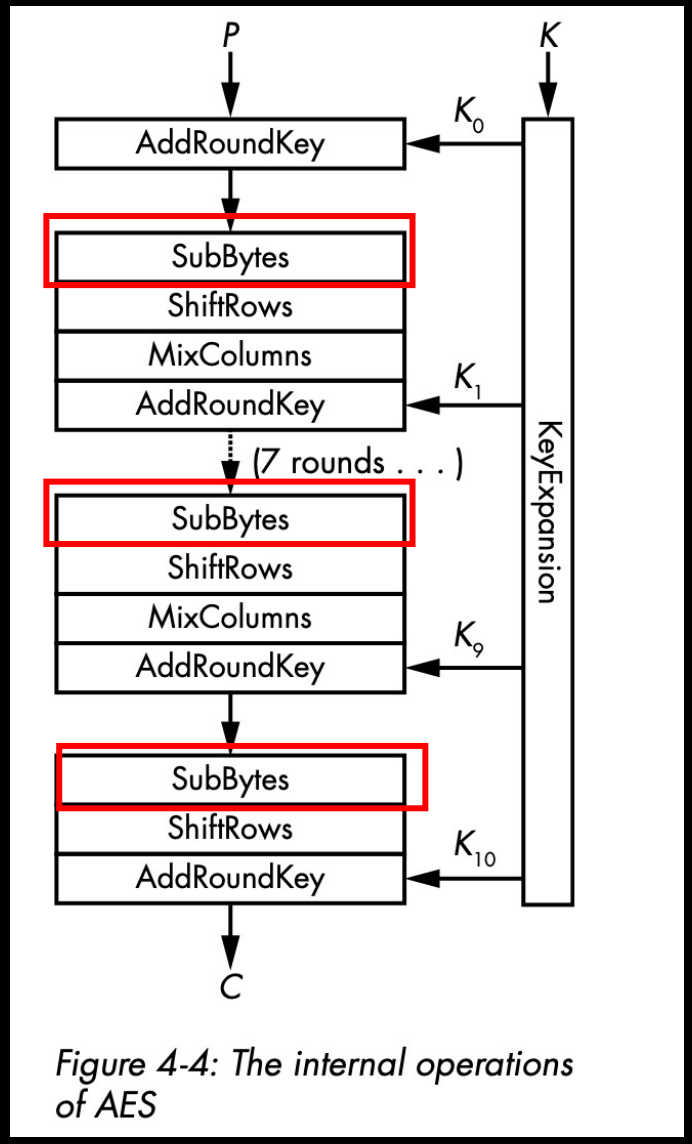
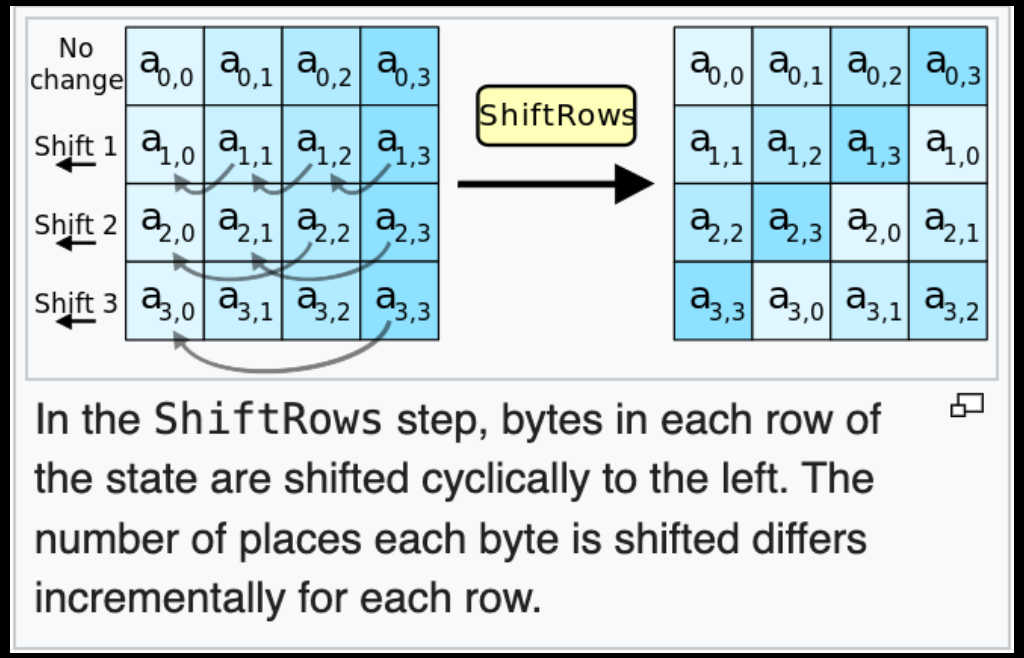



*Figure 4-4: The internal operations of AES*

# Advanced Encryption Standard

**ShiftRows**

- Cyclically shift the bytes in each row by a certain offset.



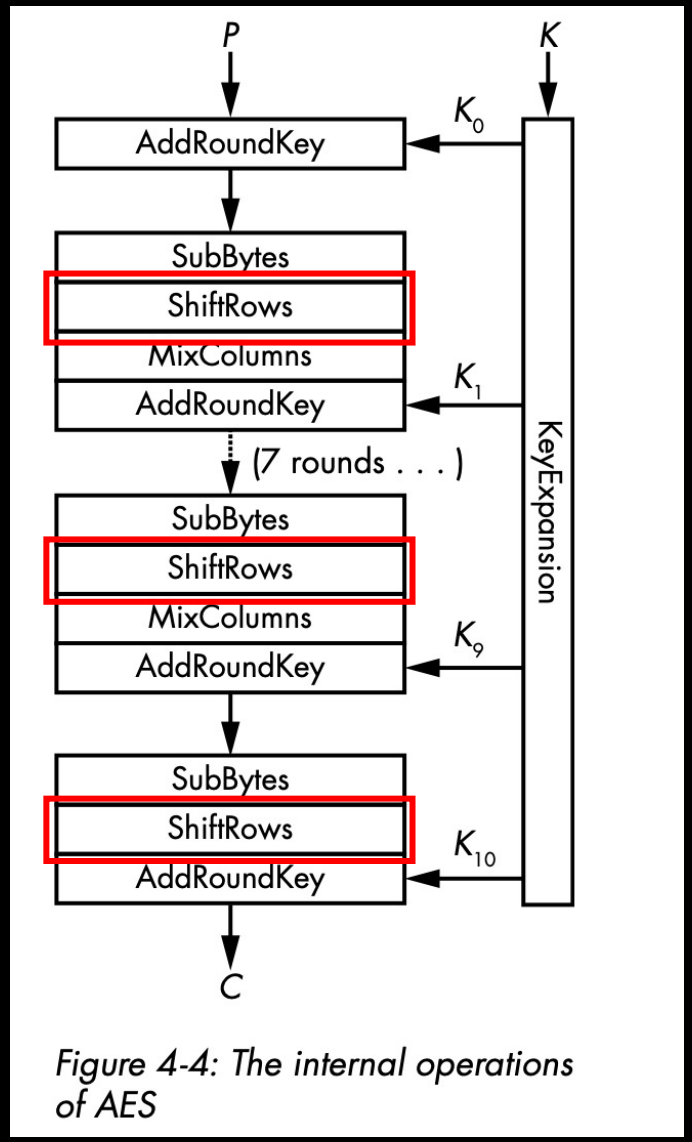




*Figure 4-4: The internal operations of AES*

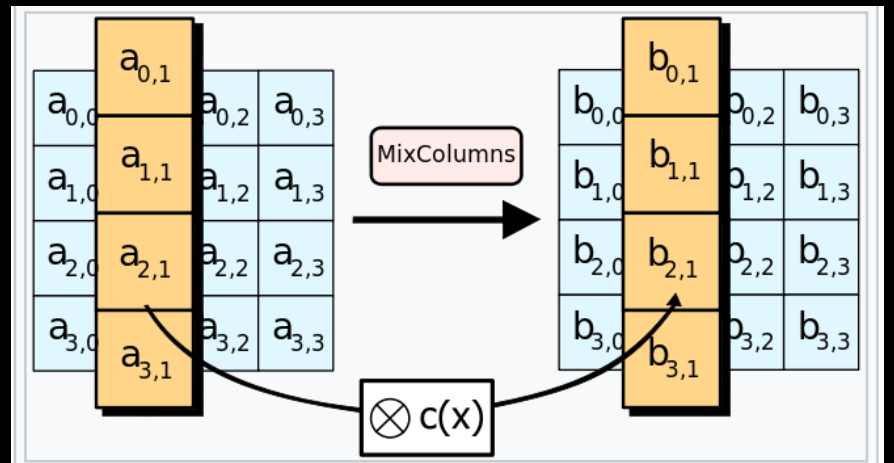
# Advanced Encryption Standard

**MixColumns**

- The four bytes of each column of the state are combined using an invertible linear transformation.

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

$$0 \leq j \leq 3$$




In the `MixColumns` step, each column of the state is multiplied with a fixed polynomial $c(x)$.

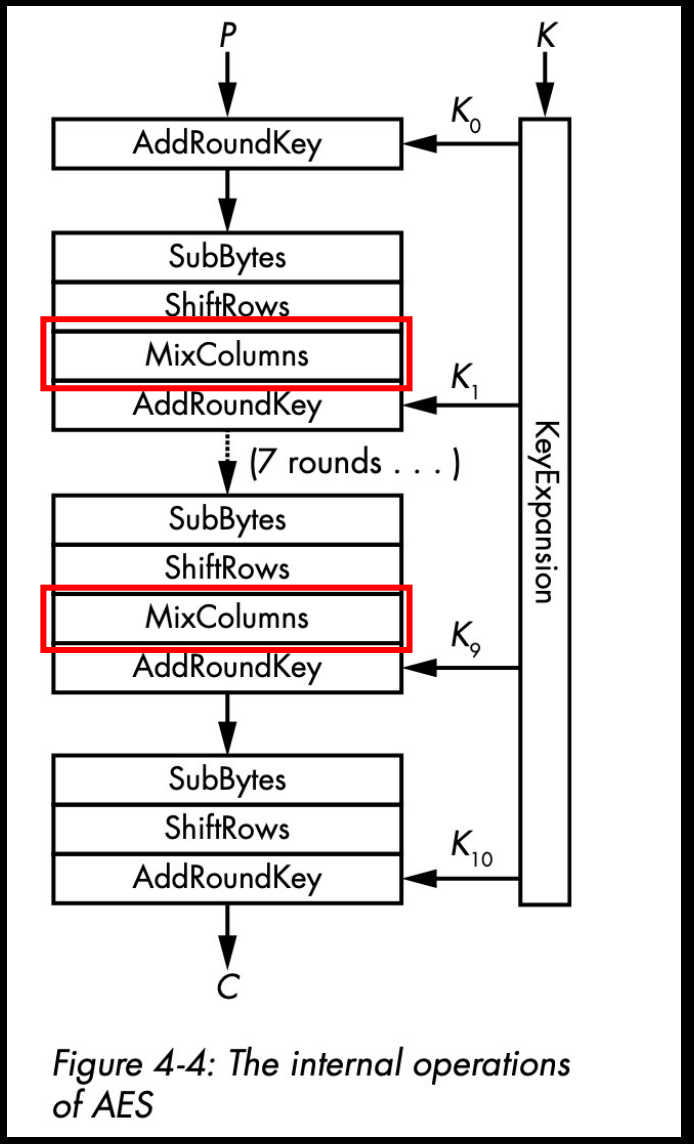
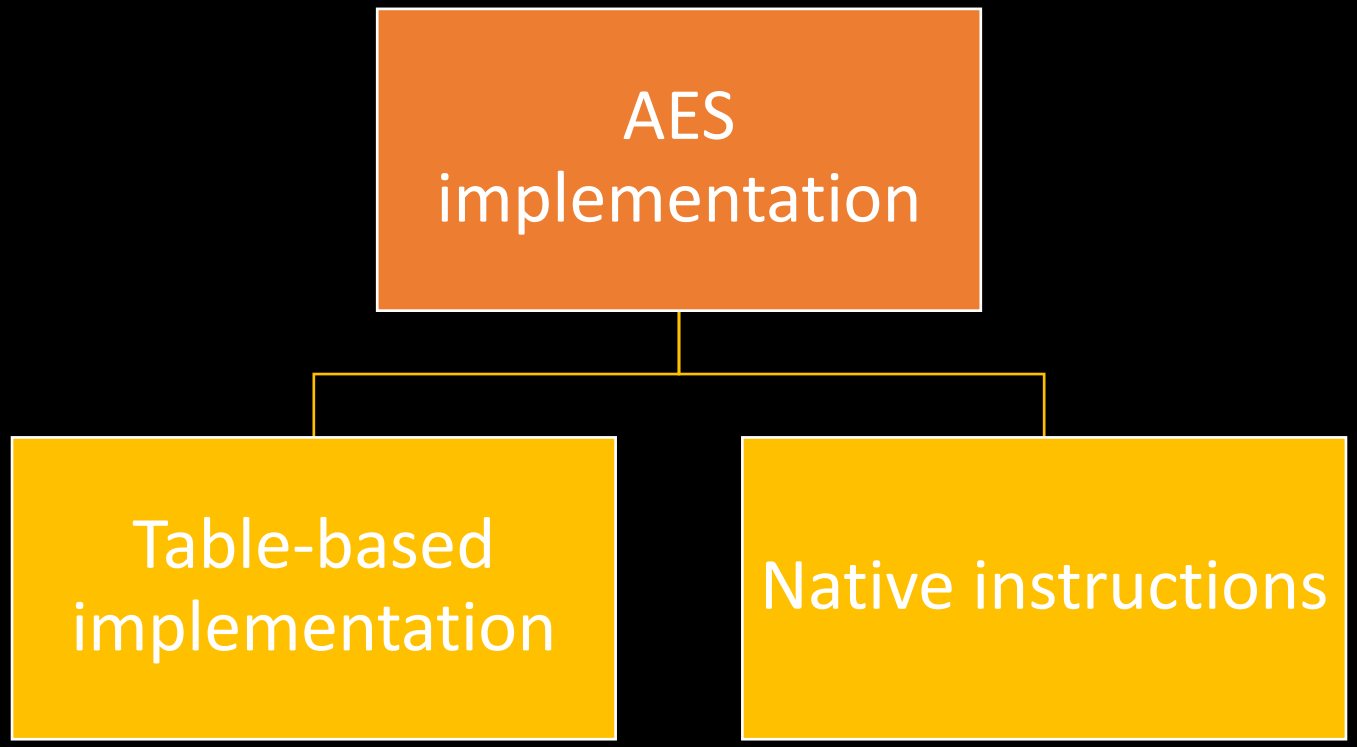



*Figure 4-4: The internal operations of AES*

# Advanced Encryption Standard

- Real-world implementation of AES is different from the algorithm.
- Fast AES software uses special techniques:



# Advanced Encryption Standard

**Table- based implementations**:

- Instead of SubBytes-ShiftRows-MixColumns, use a hardcoded table.
  - The table is hardcoded into the program.
  - Loaded in memory during runtime.
  - Perform lookups to the table to transform a given byte.
- Vulnerable to cache-timing attacks.
  - Exploits timing variations when a program access elements in cache memory.
  - Timings leak information about which element was accessed, which in turn leaks information on the secrets involved.
  - Difficult to avoid.

# Advanced Encryption Standard

**Table- based implementations**:

```
    /* round 1: */
    t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[ 4];
    t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[ 5];
    t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[ 6];
    t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[ 7];
    /* round 2: */
    s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[ 8];
    s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[ 9];
    s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[10];
    s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[11];
--snip--
```

*Listing 4-2: The table-based C implementation of AES in OpenSSL*

# Advanced Encryption Standard

**Native implementation (AES-NI)**:

- Solve the problem of cache-timing attacks on AES software implementations.
- Example: $AESENC$ instruction will execute a round on a given block.
  - 10X faster than software.

- Assuming $xmm5$ to $xmm15$ registers hold the subkeys, and $xmm0$ has ptxt block.

```
PXOR       %xmm5,  %xmm0
AESENC     %xmm6,  %xmm0
AESENC     %xmm7,  %xmm0
AESENC     %xmm8,  %xmm0
AESENC     %xmm9,  %xmm0
AESENC     %xmm10, %xmm0
AESENC     %xmm11, %xmm0
AESENC     %xmm12, %xmm0
AESENC     %xmm13, %xmm0
AESENC     %xmm14, %xmm0
AESENCLAST %xmm15, %xmm0
```

*Listing 4-3: AES native instructions*

# Advanced Encryption Standard

- Implement AES using $pycryptodome$

# TASK

**Challenge:** You've been tasked with decrypting a flag encrypted with AES-128 in CBC mode. But, there's more to it than just AES. This isn't your typical AES encryption challenge. The secret flag has been encrypted using a weak AES key. The key is 16 bytes long, but the catch is that each byte of the key is identical, and it's derived from a random value.

- The ciphertext is shown below
- Hint: check the class's code for how a key was generated.

b'\xb5W\xae"k(u\x18*"e\xf1\x98\x17\xa0\xe6\xc2\xd1\xf6\x98i\xa5[>\x02\xf1\xb6\xd5\xdb;\xc4ZJ\xc0\xf1\xa0\x0b*\xa5\xfej%\xb8\x1e\x07\xb3\x02\x16'