北京大學
PEKING UNIVERSITY

PaddlePi 实验介绍

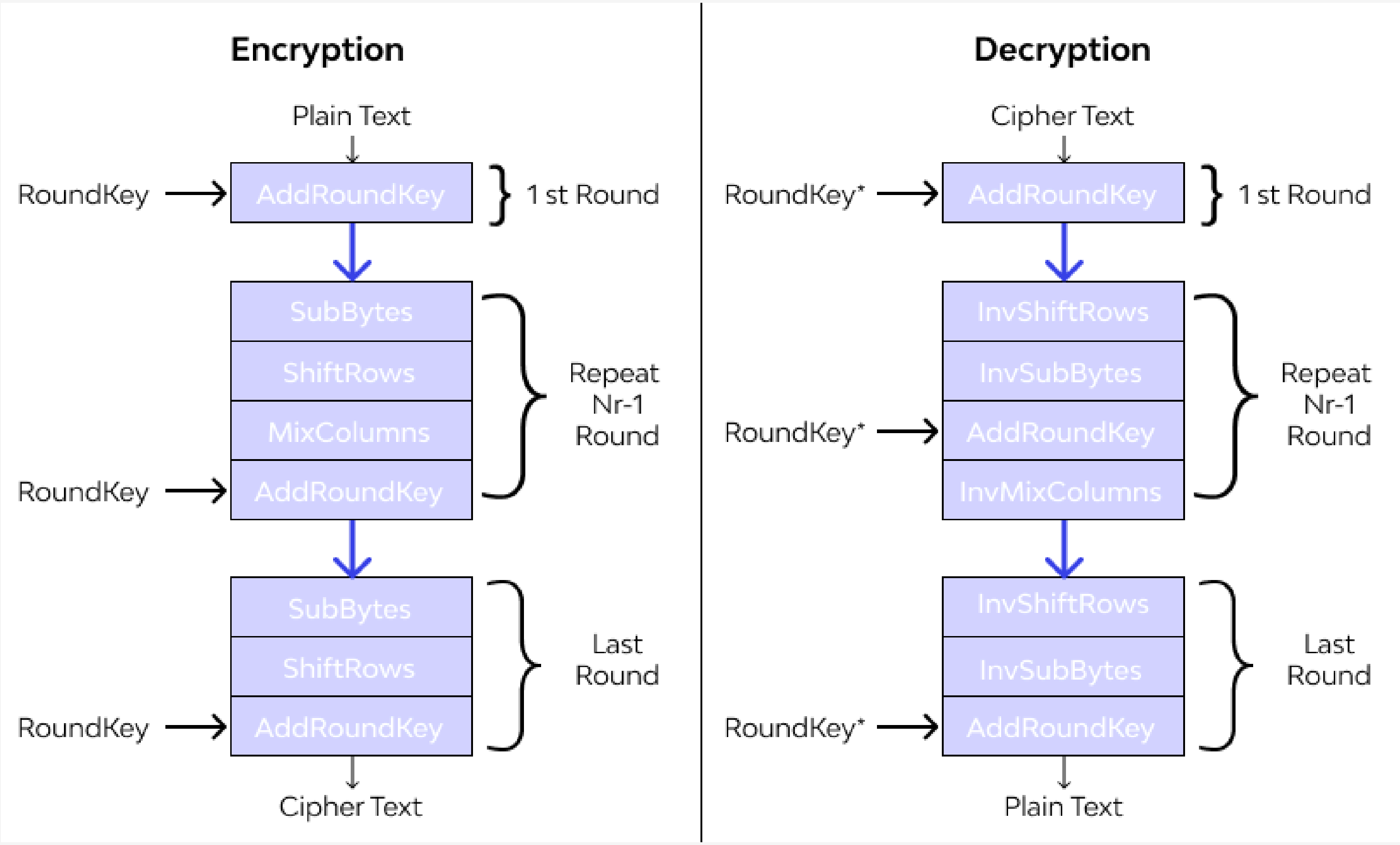
(3.2) AES 256

彭伟桀

2024.11.6

AES: Advanced Encryption Standard

- 加密单位： 128bits → 128bits
- 加密配置：
  - 128-bit key ↔ 10 rounds
  - 192-bit key ↔ 12 rounds
  - 256-bit key ↔ 14 rounds
- 具体步骤
  - 密钥拓展：Key Expansion
  - 加密：
    - SubBytes
    - ShiftRows
    - MixColumns
    - AddRoundKey



背景知识

- AES 把每个 8-bit byte 视为 $\mathrm{GF}(2^8)$ 的一个元素
  - $\mathrm{GF}(2^8) = \mathrm{GF}(2)[x]/(x^8 + x^4 + x^3 + x + 1)$
  - e.g. $0\mathrm{x}64 = 0\mathrm{b}01100100 = x^6 + x^5 + x^2$
  - 作为有限域，定义了加法和乘法，而且存在逆元
- AES 每次对 128 bits (16 bytes) 进行操作，具体来说，它把 16 bytes 视为 4×4 列优先矩阵，矩阵中元素视为 $\mathrm{GF}(2^8)$ 中元素，每个并在此基础上进行操作加解密。

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix} \xRightarrow{\text{AES}} \begin{bmatrix} c_0 & c_4 & c_8 & c_{12} \\ c_1 & c_5 & c_9 & c_{13} \\ c_2 & c_6 & c_{10} & c_{14} \\ c_3 & c_7 & c_{11} & c_{15} \end{bmatrix}$$

SBox: substitute box

$\text{Sbox} : 8bit \to 8bit$

1. 将输入映射为其 $\mathrm{GF}(2^8)$ 中乘法逆元 $(0 \mapsto 0)$
2. 应用如下仿射变换

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

AES S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Key Expansion

从输入的 128/192/256-bit 密钥扩展出每轮加密需要的密钥，这里以 128bit 为例

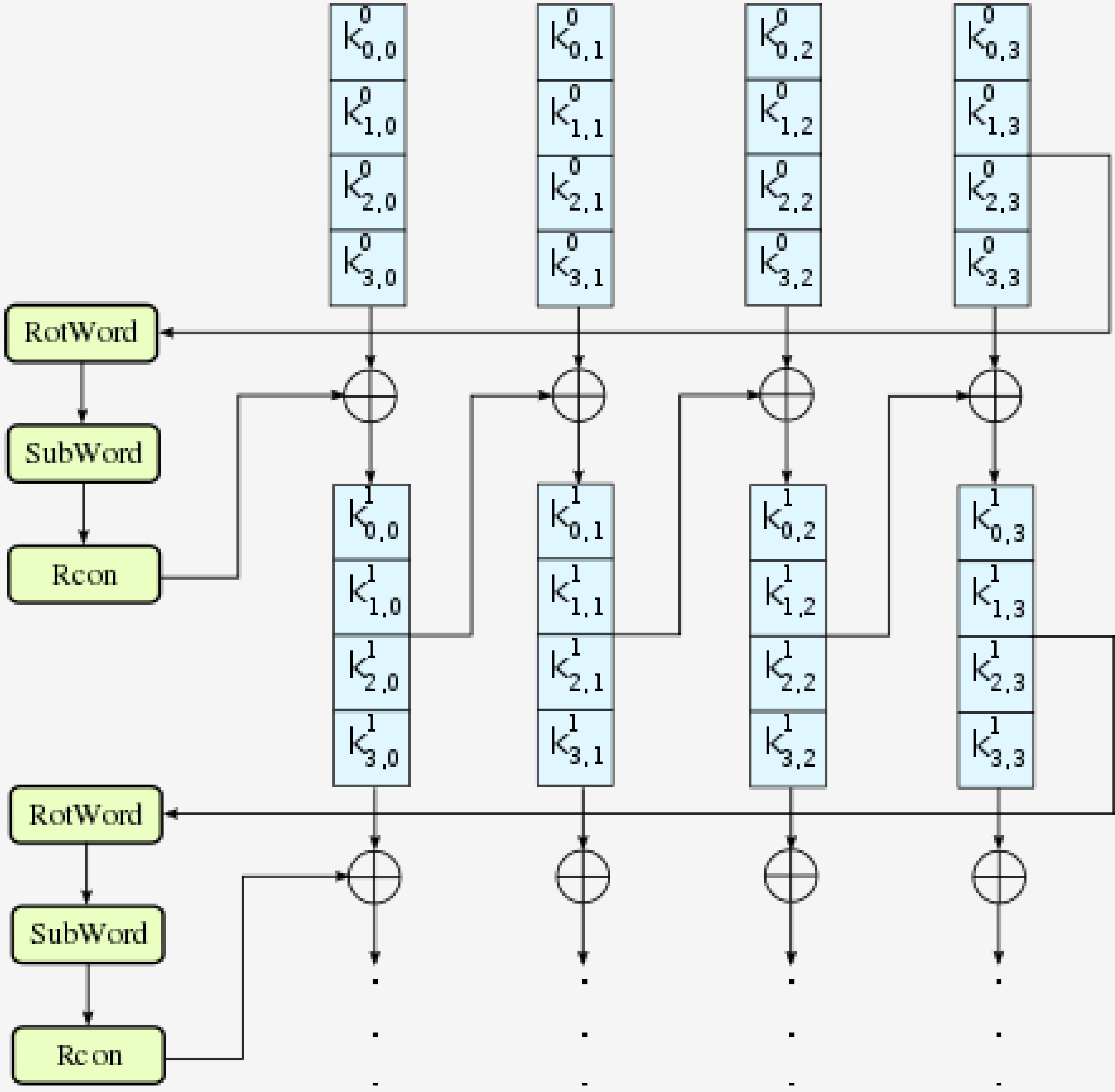
$$\begin{aligned}\mathrm{RotWord}([b_0 \quad b_1 \quad b_2 \quad b_3]) \\ = [b_1 \quad b_2 \quad b_3 \quad b_0]\end{aligned}$$

$$\begin{aligned}\mathrm{SubWord}([b_0 \quad b_1 \quad b_2 \quad b_3]) \\ = [S(b_0) \quad S(b_1) \quad S(b_2) \quad S(b_3)]\end{aligned}$$

Rcon： 常量，对应每一轮

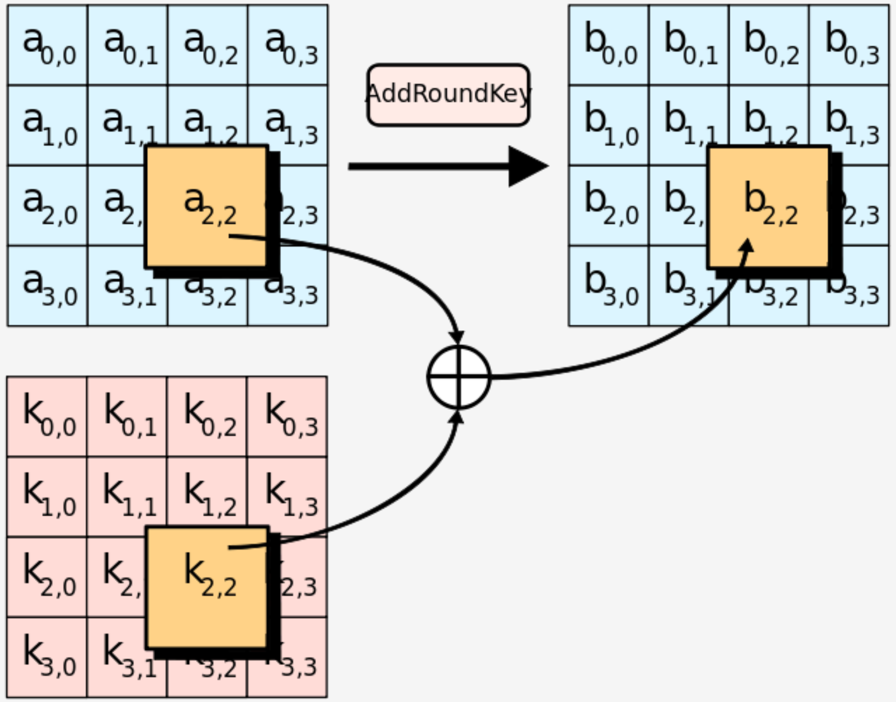
$$rcon_i = [rc_i \quad 00_{16} \quad 00_{16} \quad 00_{16}]$$

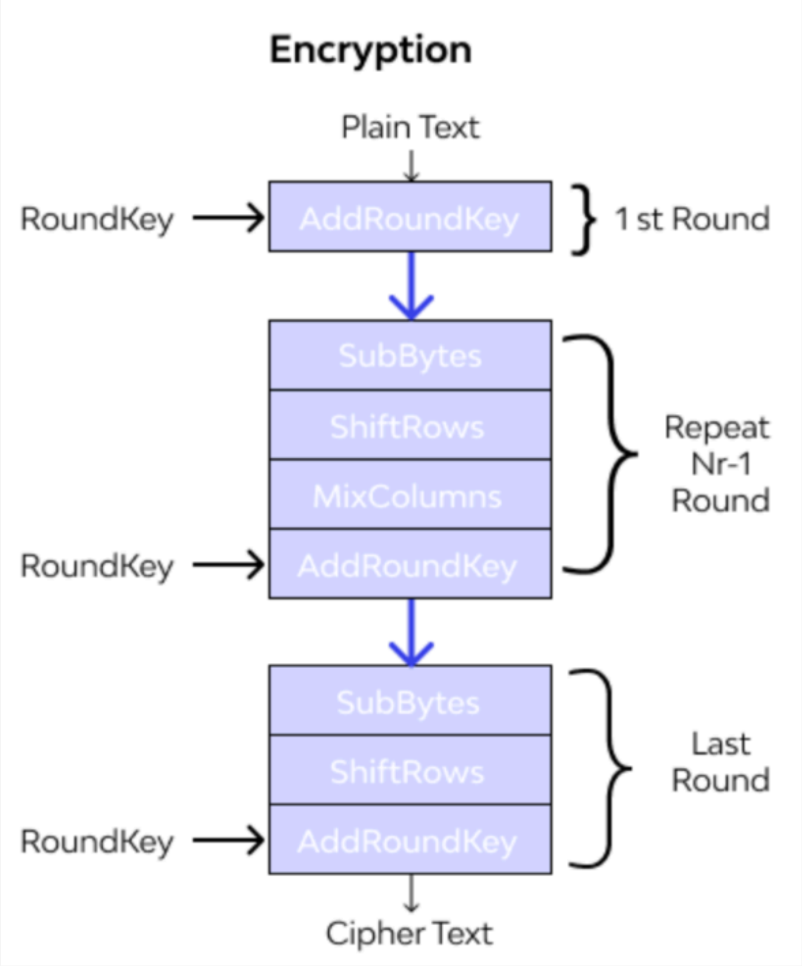
$$rc_i = x^{i-1} \quad \mathrm{mod} \;\; x^8 + x^4 + x^3 + x + 1$$



加密步骤

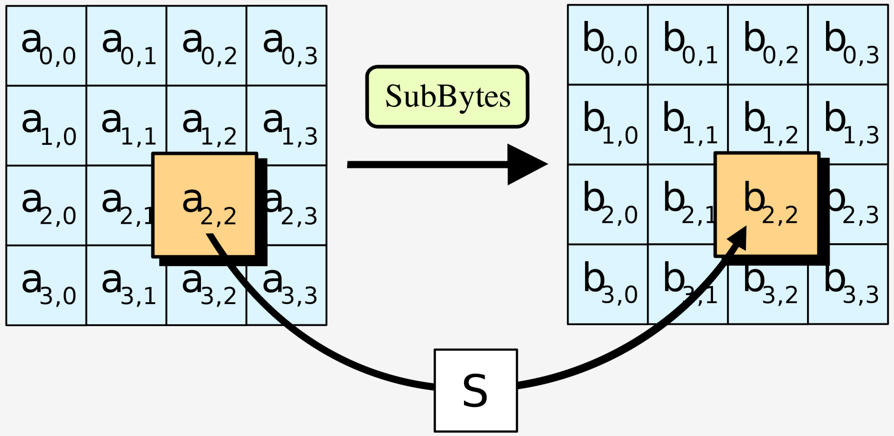
1. **AddRoundKey**

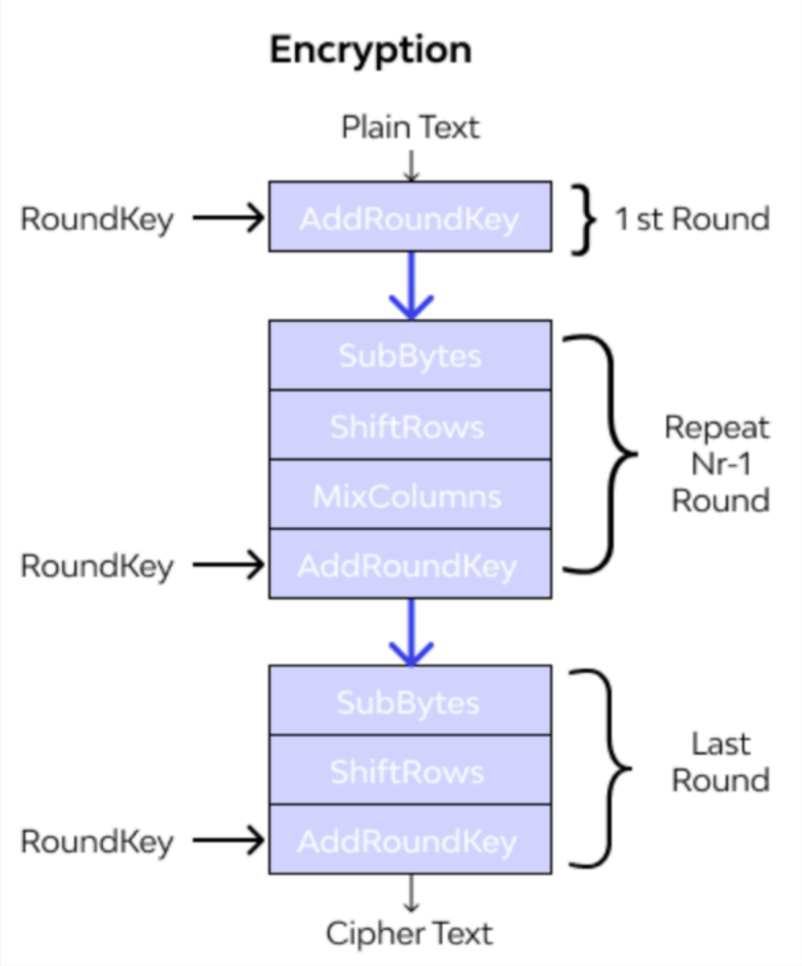




加密步骤

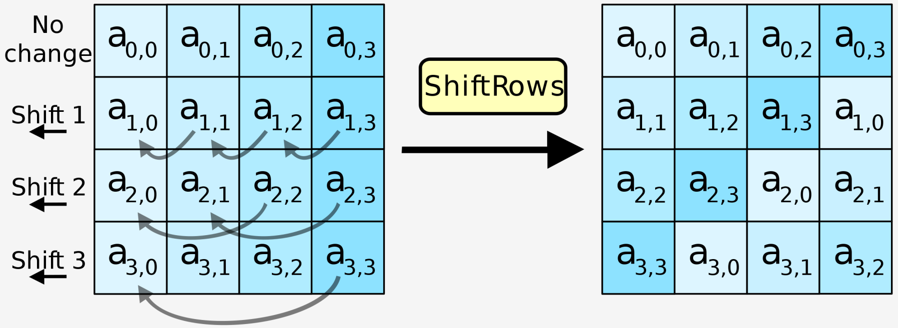
1. AddRoundKey
2. **SubBytes**

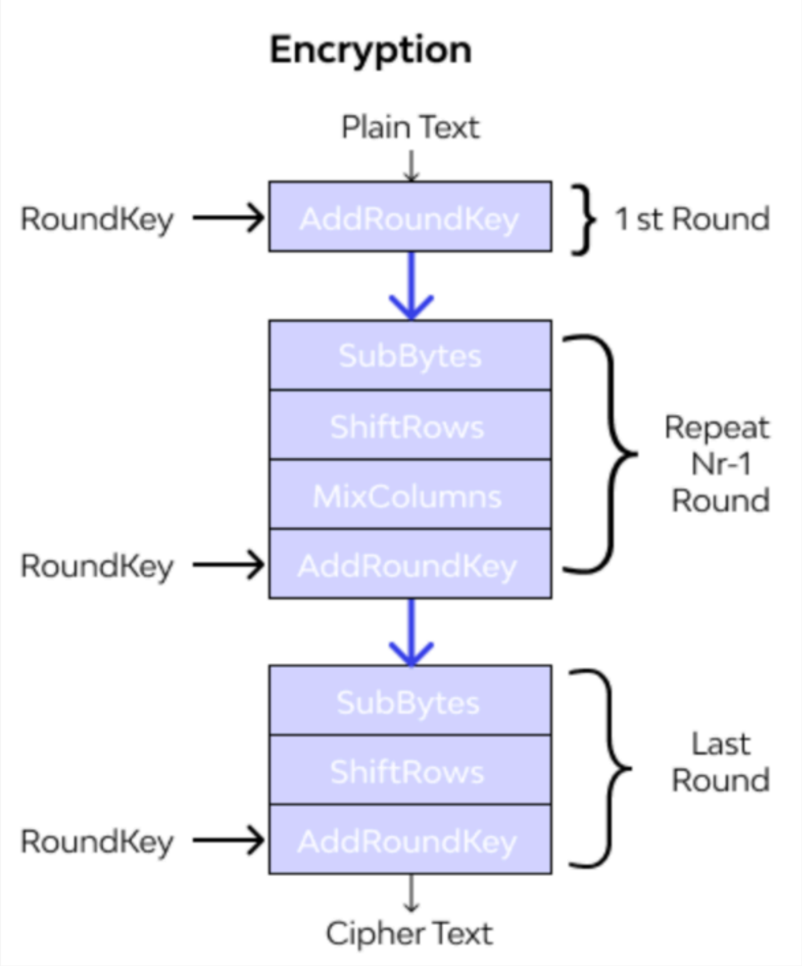




加密步骤

1. AddRoundKey
2. SubBytes
3. **ShiftRows**

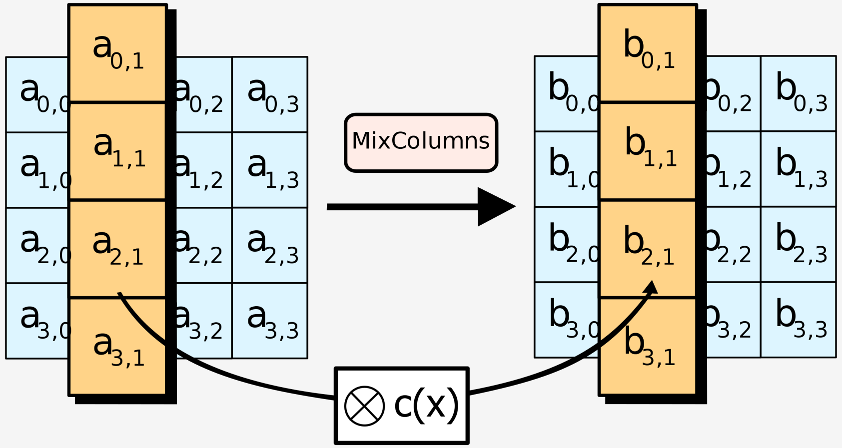


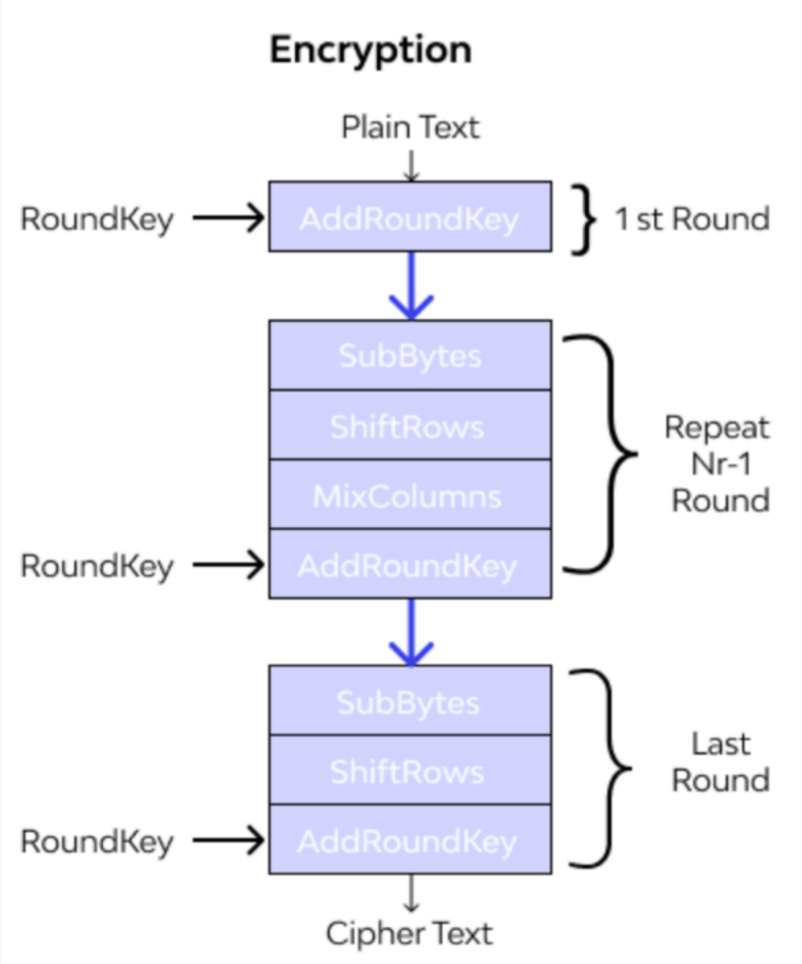


加密步骤

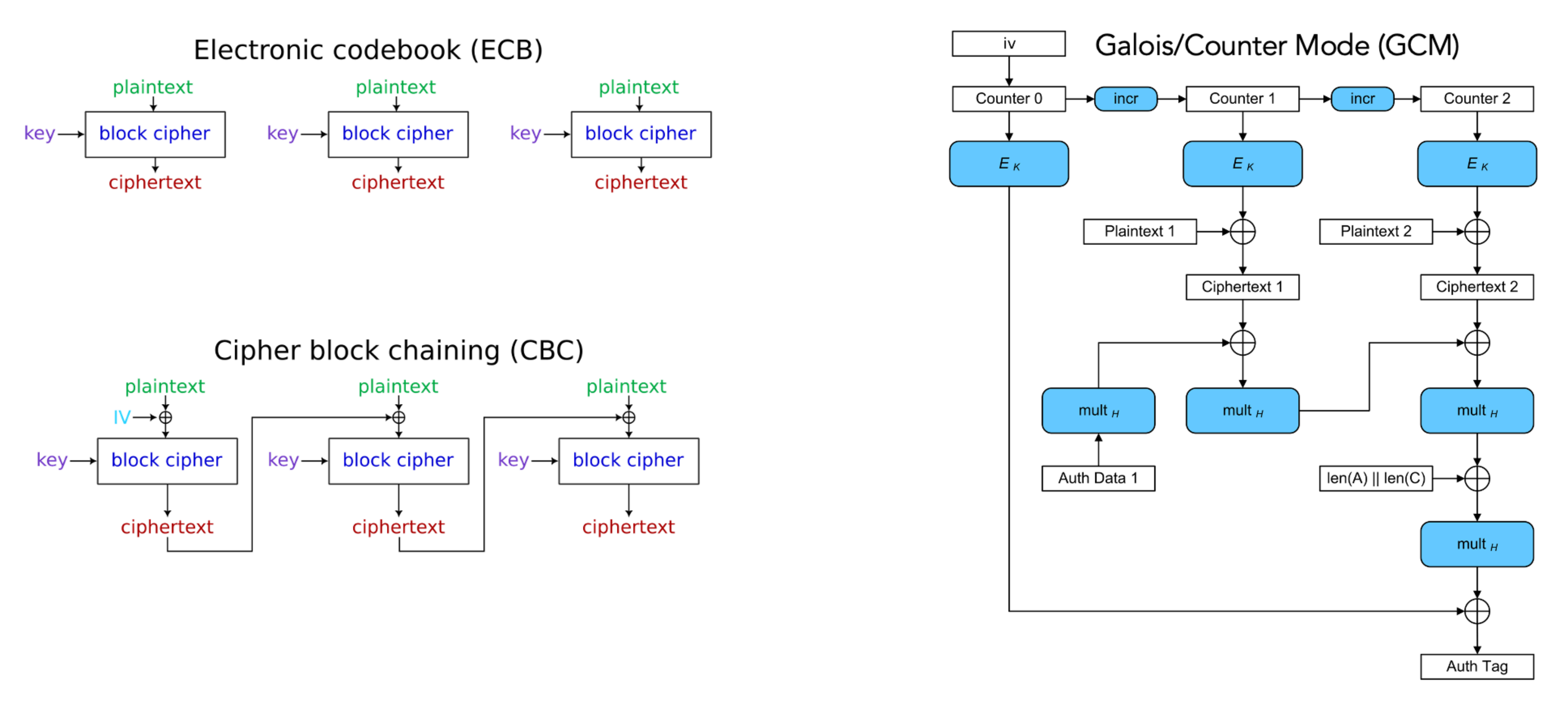
1. AddRoundKey
2. SubBytes
3. ShiftRows
4. **MixColumns**

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

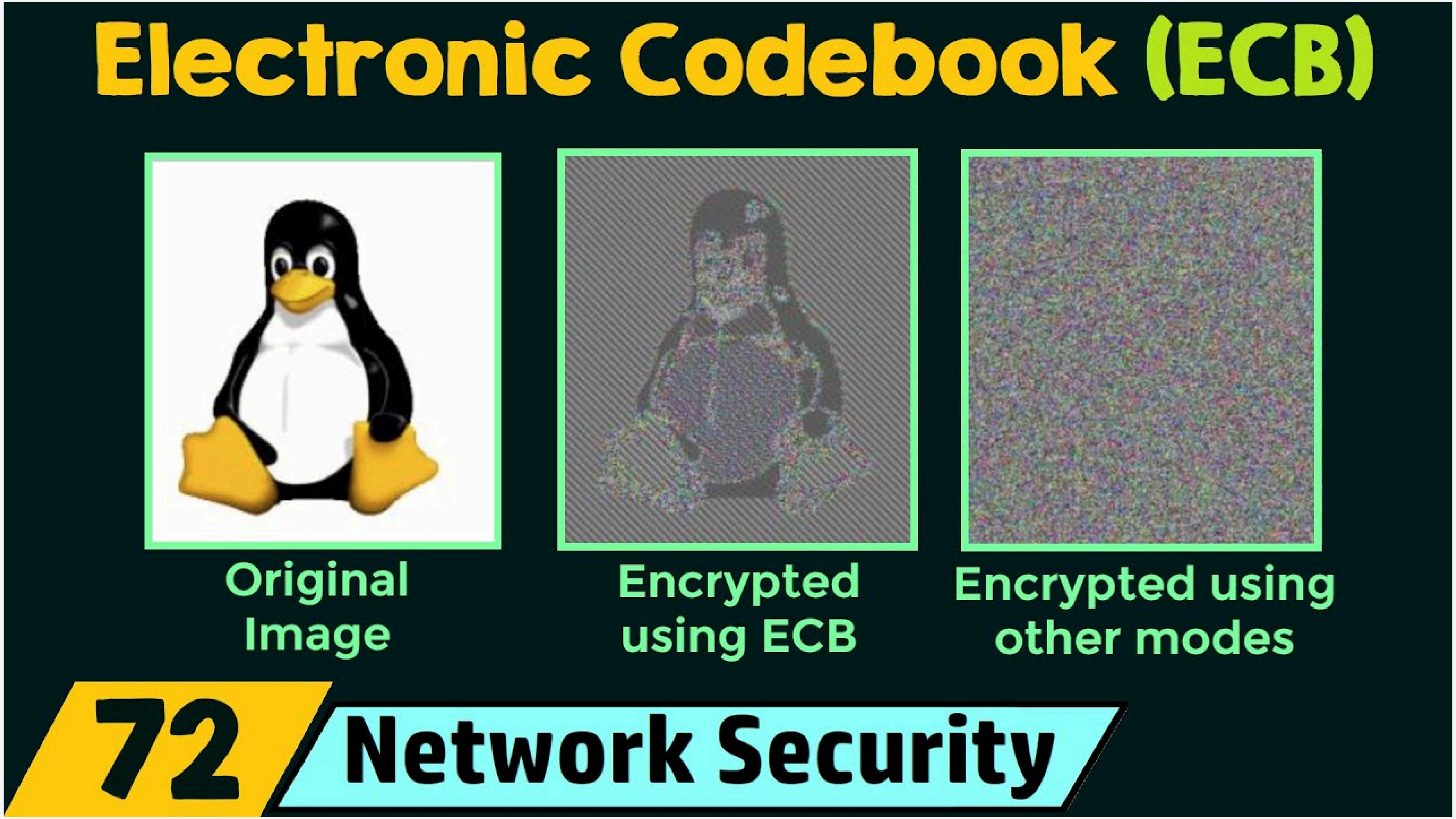




加密模式 ECB/CBC/GCM



ECB 并不安全


Electronic Codebook (ECB)
Original Image
Encrypted using ECB
Encrypted using other modes
72
Network Security

K210 AES API

K210 提供了如下的加密 API 排列组合：

mode	key length	operation	transfer
ecb	128	encrypt	none (cpu)
cbc	192	decrypt	dma
gcm	256		

参数要求：

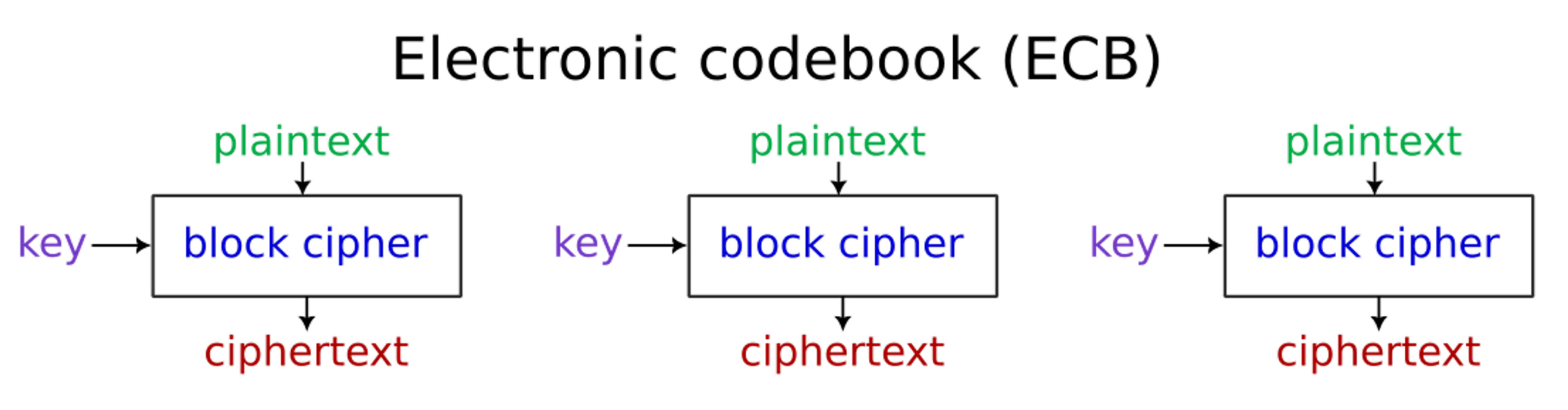
- 对于 ECB/CBC，原理上需要 padding 到 16B，输出 buffer 需要是 16B 的倍数
- 对于 GCM，原理上无需 padding，但是 DMA 要求 4B 对齐，Tag 的大小要求是 16B

```c
// 命名：aes_{mode}{len}_hard_{operation}{transfer_suffix}
void aes_ecb256_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data);
void aes_cbc256_hard_decrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data);
void aes_gcm256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
                                 gcm_context_t *context,
                                 uint8_t *input_data, size_t input_len,
                                 uint8_t *output_data, uint8_t *gcm_tag);
// 底层 API，无需直接调用，实际会使用 MMIO 与 AES 加速器通信
void aes_init(uint8_t *input_key, size_t input_key_len, uint8_t *iv, size_t iv_len, uint8_t *gcm_aad,
              aes_cipher_mode_t cipher_mode, aes_encrypt_sel_t encrypt_sel, size_t gcm_aad_len, size_t input_data_len);
void aes_process(uint8_t *input_data, uint8_t *output_data, size_t input_data_len, aes_cipher_mode_t cipher_mode);
void gcm_get_tag(uint8_t *gcm_tag);
```

ECB API

```
void aes_ecb256_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data);
void aes_ecb256_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data);
```

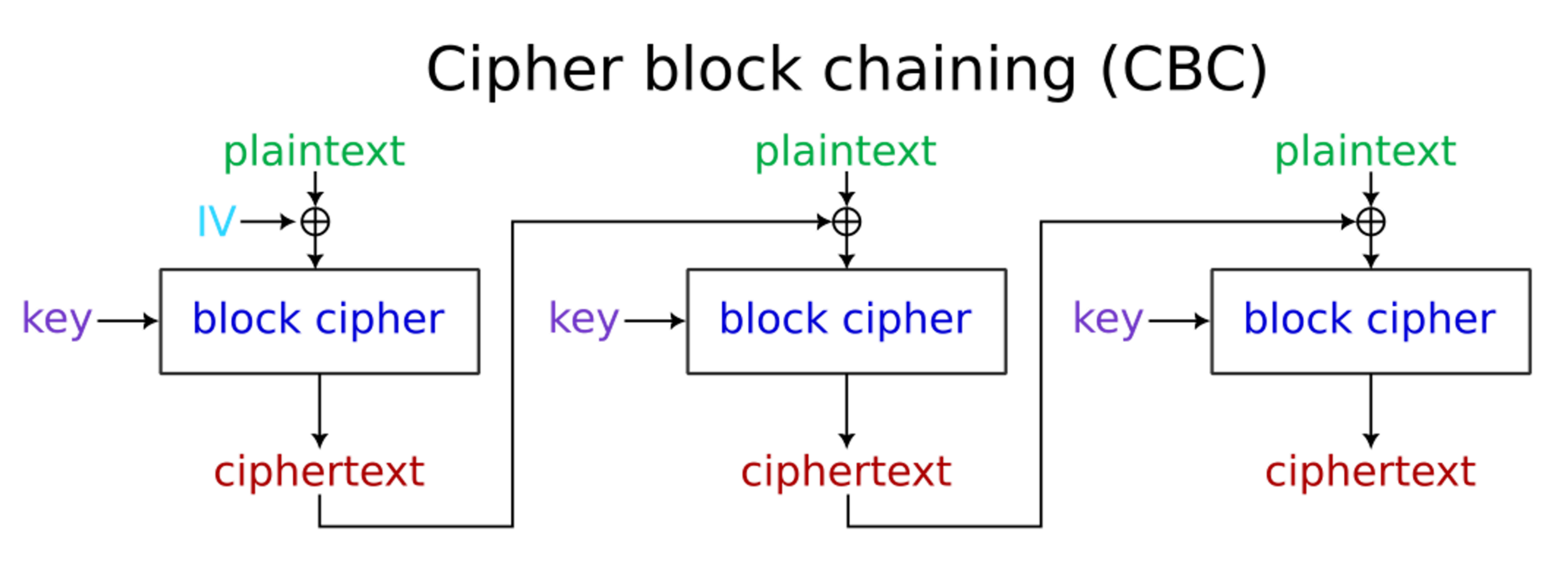
最简单，直接提供密钥、输入数据及长度即可加解密



CBC API

```
void aes_cbc256_hard_decrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data);
void aes_cbc256_hard_encrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data);
typedef struct {
  uint8_t *input_key; /* The buffer holding the encryption or decryption key. */
  uint8_t *iv; /* The initialization vector. must be 128 bit */
} cbc_context_t;
```

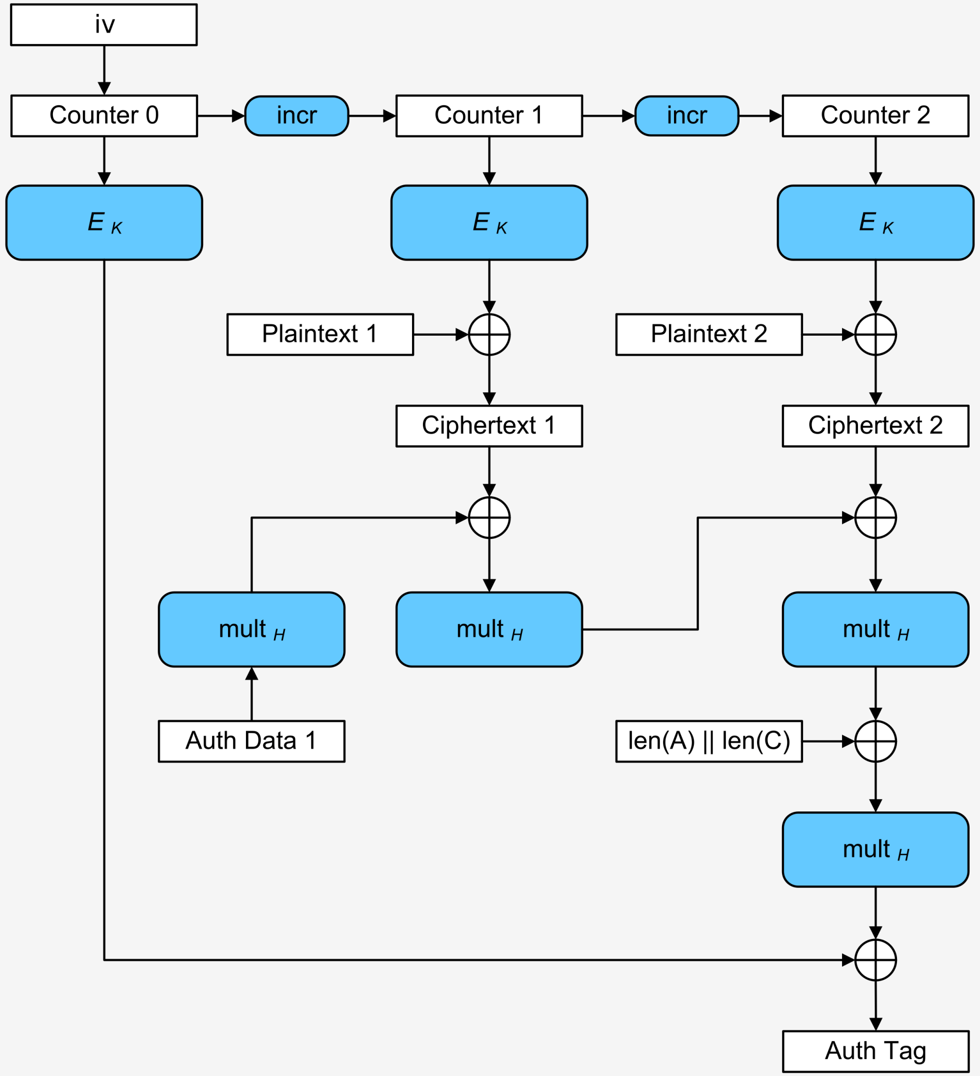
除了提供密钥和输入数据之外，还需要提供 IV (Initial Vector)



GCM API

```c
void aes_gcm256_hard_decrypt(
        gcm_context_t *context, uint8_t *input_data,
        size_t input_len, uint8_t *output_data, uint8_t *gcm_tag);
void aes_gcm256_hard_encrypt(
        gcm_context_t *context, uint8_t *input_data,
        size_t input_len, uint8_t *output_data, uint8_t *gcm_tag);
typedef struct {
  /* The buffer holding the encryption or decryption key. */
  uint8_t *input_key;
  /* The initialization vector. must be 96 bit */
  uint8_t *iv;
  /* The buffer holding the Additional authenticated data. or NULL */
  uint8_t *gcm_aad;
  /* The length of the Additional authenticated data. or 0L */
  size_t gcm_aad_len;
} gcm_context_t;
```

除了提供密钥、输入数据、IV 之外，可选附加 AuthData，输出会额外有一个 AuthTag，可以用来校验数据完整性。



DMA API

```c
void aes_ecb256_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num,
        uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data);
void aes_ecb256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
        uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data);

void aes_cbc256_hard_decrypt_dma(
        dmac_channel_number_t dma_receive_channel_num, cbc_context_t *context,
        uint8_t *input_data, size_t input_len, uint8_t *output_data);
void aes_cbc256_hard_encrypt_dma(
        dmac_channel_number_t dma_receive_channel_num, cbc_context_t *context,
        uint8_t *input_data, size_t input_len, uint8_t *output_data);

void aes_gcm256_hard_decrypt_dma(
        dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data,
        size_t input_len, uint8_t *output_data, uint8_t *gcm_tag);
void aes_gcm256_hard_encrypt_dma(
        dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data,
        size_t input_len, uint8_t *output_data, uint8_t *gcm_tag);
```

需额外指定输出用的 DMA 通道，其他与非 DMA API 完全一致。

示例代码讲解

文件结构

```
aes_256_test
├── aes2.c                 // mbedtls 的 AES 实现 {{
├── aes2.h
├── aes_cbc.c
├── aes_cbc.h
├── cipher.c
├── cipher.h
├── cipher_internal.h
├── cipher_wrap.c
├── config.h
├── gcm.c
├── gcm.h                  // }}
├── main.c                 // 主程序
└── README.md
```

该示例使用 mbedtls 作为软件版本的 AES 实现，并且与使用硬件 AES 加速器 (w/,w/o DMA)进行对比

示例代码讲解

加密 API 的使用

```
// aes_dma_get_data () {
cbc_context_t cbc_context;
gcm_context_t gcm_context;
cbc_context.input_key = input_key;
cbc_context.iv = iv;
gcm_context.gcm_aad = aes_aad;
gcm_context.gcm_aad_len = aad_size;
gcm_context.input_key = input_key;
gcm_context.iv = iv;
```

初始化 context

```
if (cipher_mod == AES_CBC) {
        aes_cbc256_hard_encrypt_dma(
                DMAC_CHANNEL1, &cbc_context,
                aes_data, data_size, aes_hard_out_data);
} else if (cipher_mod == AES_ECB) {
        aes_ecb256_hard_encrypt_dma(
                DMAC_CHANNEL1, input_key,
                aes_data, data_size, aes_hard_out_data);
} else {
        aes_gcm256_hard_encrypt_dma(
                DMAC_CHANNEL1, &gcm_context,
                aes_data, data_size, aes_hard_out_data, gcm_hard_tag);
}
```

根据不同的加密模式调用不同的函数

`aes_cpu_get_data()` 跟 `aes_dma_get_data()` 类似，只是用了非 DMA 版本 API。
函数中还测量了调用 API 所用 #cycles，这里省略了具体的测量代码。

示例代码讲解

```c
check_result_t aes_check (uint8_t *input_key,
    size_t key_len,
    uint8_t *iv,
    size_t iv_len,
    uint8_t *aes_aad,
    size_t aad_size,
    aes_cipher_mode_t cipher_mod,
    uint8_t *aes_data,
    size_t data_size)
{
    check_result_t ret = AES_CHECK_PASS;

    memset(aes_soft_in_data, 0, AES_TEST_PADDING_LEN);
    memset(aes_soft_out_data, 0, AES_TEST_PADDING_LEN);
    memset(aes_hard_out_data, 0, AES_TEST_PADDING_LEN);

    aes_dma_get_data(input_key, key_len, iv, iv_len, aes_aad, aad_size, cipher_mod, aes_data, data_size);
    aes_soft_get_data(input_key, key_len, iv, iv_len, aes_aad, aad_size, cipher_mod, aes_data, data_size);
    ret |= aes_encrypt_compare_hard_soft(cipher_mod, data_size);

    memset(aes_hard_out_data, 0, AES_TEST_PADDING_LEN);
    aes_cpu_get_data(input_key, key_len, iv, iv_len, aes_aad, aad_size, cipher_mod, aes_data, data_size);
    ret |= aes_encrypt_compare_hard_soft(cipher_mod, data_size);
    memset(aes_soft_out_data, 0, AES_TEST_PADDING_LEN);
    ret |= aes_check_decrypt(input_key, key_len, iv, iv_len, aes_aad, aad_size, cipher_mod, aes_data, data_size);
    return ret;
}
```

分别调用不同软件实现，硬件加速器，不用 DMA 的硬件加速器实现，并检查正确性。

示例代码讲解

```c
// main() {
for (cipher = AES_ECB; cipher < AES_CIPHER_MAX; cipher++)
{
        printf("[%s] test all byte ... \n", cipher_name[cipher]);
        if (AES_CHECK_FAIL == aes_check_all_byte(cipher))  // 输入数据长度 0..256
        {
                printf("aes %s check_all_byte fail\n", cipher_name[cipher]);
                return -1;
        }
        printf("[%s] test all key ... \n", cipher_name[cipher]);
        if (AES_CHECK_FAIL == aes_check_all_key(cipher))   // 测试 256/key_len 组密钥
        {
                printf("aes %s check_all_key fail\n", cipher_name[cipher]);
                return -1;
        }
        printf("[%s] test all iv ... \n", cipher_name[cipher]);
        if (AES_CHECK_FAIL == aes_check_all_iv(cipher))    // 测试 256/iv_len 组 iv
        {
                printf("aes %s check_all_iv fail\n", cipher_name[cipher]);
                return -1;
        }
        // ...
}
```

示例功能演示

- 所用编译下载命令如下：

```
# in `build` directory
cmake .. -DPROJ=aes_256_test -DTOOLCHAIN=/opt/kendryte-toolchain/bin
make -j
kflash -b 3000000 -t aes_256_test.bin
```

- 编译程序并下载运行
- 可以看到如右图所示的输出
- 可以看到使用硬件加速器的实现远快于软件实现
- 但输出使用 CPU 拷贝会快于 DMA
  - 可能是因为数据长度只有 1029B 比较短
  - 另外输入总是使用 CPU 拷贝，可能是瓶颈

```
--- Miniterm on /dev/ttyUSB0  115200,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
begin test 0
[aes-ecb-256] test all byte ...
[aes-ecb-256] test all key ...
[aes-ecb-256] test all iv ...
[aes-ecb-256] [1029 bytes] cpu time = 85 us, dma time = 94 us, soft time = 2066 us
[aes-cbc-256] test all byte ...
[aes-cbc-256] test all key ...
[aes-cbc-256] test all iv ...
[aes-cbc-256] [1029 bytes] cpu time = 85 us, dma time = 93 us, soft time = 1826 us
[aes-gcm-256] test all byte ...
[aes-gcm-256] test all key ...
[aes-gcm-256] test all iv ...
[aes-gcm-256] [1029 bytes] cpu time = 86 us, dma time = 103 us, soft time = 488 us
aes-256 test pass

--- exit ---
```

北京大學
PEKING UNIVERSITY

请大家批评指正