中国矿业大学计算机学院

2019 级本科生课程报告

课程名称 密 码 学

研究主题 SM4 分组加密算法研究

小组成员 凌子健、李春阳、罗敏行、

姚炜柏、罗贤俊、陶梅悦、于清

任课教师 汪 楚 娇

报告时间 2021.12.22

目录

1 SM4算法介绍

2012年3月，国家密码管理局正式公布了包含SM4分组密码算法在内的《祖冲之序列密码算法》等6项密码行业标准。与DES和AES算法类似，SM4算法是一种分组密码算法。其分组长度为128bit，密钥长度也为128bit。加密算法与密钥扩展算法均采用32轮非线性迭代结构，以字（32位）为单位进行加密运算，每一次迭代运算均为一轮变换函数F。SM4算法加/解密算法的结构相同，只是使用轮密钥相反，其中解密轮密钥是加密轮密钥的逆序。

SMS4分组加密算法是中国无线标准中使用的分组加密算法，在2012年已经被国家商用密码管理局确定为国家密码行业标准，标准编号GM/T 0002-2012并且改名为SM4算法，与SM2椭圆曲线公钥密码算法，SM3密码杂凑算法共同作为国家密码的行业标准，在我国密码行业中有着极其重要的位置。

SM4有很高的灵活性，所采用的S盒可以灵活地被替换，以应对突发性的安全威胁。算法的32轮迭代采用串行处理，这与AES中每轮使用代换和混淆并行地处理整个分组有很大不同。

S盒是一种利用非线性变换构造的分组密码的一个组件，主要是为了实现分组密码过程中的混淆的特性和设计的。SMS4算法中的S盒在设计之初完全按照欧美分组密码的设计标准进行，它采用的方法是能够很好抵抗差值攻击的仿射函数逆映射复合法。

参考官方网站：密码行业标准化：http://www.gmbz.org.cn/main/bzlb.html

	性算法			algorithm						
GM/T 0002-2012	SM4分组密码算法	现行	推荐性GM/T	SM4 block cipher algorithm	中国科学院数据与通信保护研究教育中心	中国科学院数据与通信保护研究教育中心、国家密码管理局商用密码检测中心	20120321	20120321	已上升为国标	文件查看
				Public key						

2 SM4算法原理

本算法是一个分组算法。该算法的分组长度为128比特，密钥长度为128比特。加密算法与密钥扩展算法都采用32轮非线性迭代结构。解密算法与加密算法的结构相同，只是轮密钥的使用顺序相反，解密轮密钥是加密轮密钥的逆序。

2.1 术语说明

2.1.1 字与字节

用Z_2^e表示e-比特的向量集，Z_2^{32}中的元素称为字，Z_2^8中的元素称为字节。

2.1.2 S盒

S盒为固定的8比特输入8比特输出的置换，记为$Sbox(X)$

2.1.3 基本运算

在本算法中采用了以下基本运算：

- $\oplus$ 32比特异或
- $<<< i$ 32比特循环左移i位

2.1.4 密钥及密钥参量

加密密钥长度为 128 比特, 表示为$\mathrm{MK}=(\mathrm{MK}_0,\mathrm{MK}_1,\mathrm{MK}_2,\mathrm{MK}_3)$, 其中 $\mathrm{MK}_i(\mathrm{i}=0,1,2,3)$为 字。

轮密钥表示为 $(\mathrm{rk}_0,\mathrm{rk}_1,\ldots,\mathrm{rk}_{31})$, 其中 $\mathrm{rk}_{\mathrm{i}}(\mathrm{i}=0,\ldots,31)$ 为字。轮密钥由加密密钥生成。

$\mathrm{FK}=(\mathrm{FK}_0,\mathrm{FK}_1,\mathrm{FK}_2,\mathrm{FK}_3)$为系统参数,

$\mathrm{CK}=(\mathrm{CK}_0,\mathrm{CK}_1,\ldots,\mathrm{CK}_{31})$为固定参数, 用于密钥扩展算法,

其中 $\mathrm{FK}_{\mathrm{i}}(i=0,\ldots,3)$、$\mathrm{CK}_{\mathrm{i}}(i=0,\ldots,31)$ 为字。

2.2 轮函数F

本算法采用非线性迭代结构, 以字为单位进行加密运算, 称一次迭代运算为一轮变换。

设输入为$(X_0,X_1,X_2,X_3)\in(Z_2^{32})^4$,轮密钥为$rk\in Z_2^{32}$,则轮函数F为:

$$F(X_0,X_1,X_2,X_3,rk)=X_0\oplus T(X_1\oplus X_2\oplus X_3\oplus rk)$$

2.2.1 合成置换T

$T:Z_2^{32}\rightarrow Z_2^{32}$，是一个可逆变换，由非线性变换$\tau$和线性变换$L$复合而成,即$T(x)=L(\tau(x))$

2.2.1.1 非线性变换τ

τ 由 4 个并行的S盒构成。
设输入为 $A=(a_0,a_1,a_2,a_3)\in\left(Z_2^8\right)^4$, 输出为 $B=(b_0,b_1,b_2,b_3)\in\left(Z_2^8\right)^4$, 则

$$(b_0,b_1,b_2,b_3)=\tau(A)=(\mathrm{Sbox}\,(a_0),\mathrm{Sbox}\,(a_1),\mathrm{Sbox}\,(a_2),\mathrm{Sbox}\,(a_3))$$

2.2.1.2 线性变换L

非线性变换 τ 的输出是线性变换 L的输入。设输入为 $B\in Z_2^{32}$, 输出为 $C\in Z_2^{32}$, 则

$$C=L(B)=B\oplus(B<<<2)\oplus(B<<<10)\oplus(B<<<18)\oplus(B<<<24)$$

2.2.2 S盒

S 盒中数据均采用 16 进制表示。

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

输入‘ef’，则经S盒后的值为表中第e行和第f列的值，Sbox(‘ef’)= ‘84’。

2.3 加/解密算法

定义反序变换 R为:

$$R(A_0, A_1, A_2, A_3) = (A_3, A_2, A_1, A_0), A_i \in Z_2^{32}, i = 0, 1, 2, 3$$

设明文输入为 $(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$, 密文输出为$(Y_0, Y_1, Y_2, Y_3) \in (Z_2^{32})^4$, 轮密钥为 $rk_i \in Z_2^{32}, i = 0, 1, 2, \ldots, 31$ 。则本算法的加密变换为:

$$\begin{gathered} X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i) \\ = X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i), i = 0, 1, \ldots, 31. \\ (Y_0, Y_1, Y_2, Y_3) = R(X_{32}, X_{33}, X_{34}, X_{35}) = (X_{35}, X_{34}, X_{33}, X_{32}) \end{gathered}$$

本算法的解密变换与加密变换结构相同, 不同的仅是轮密钥的使用顺序。

密时轮密钥的使用顺序为: $(rk_0, rk_1, \ldots, rk_{31})$

解密时轮密钥的使用顺序为:$(rk_{31}, rk_{30}, \ldots, rk_0)$

2.4 密钥扩展算法

本算法中加密算法的轮密钥由加密密钥通过密钥扩展算法生成。

加密密钥

$MK = (MK_0, MK_1, MK_2, MK_3), MK_i \in Z_2^{32}, i = 0, 1, 2, 3;$

令 $K_i \in Z_2^{32}, i = 0, 1, \ldots, 35$, 轮密钥为 $rk_i \in Z_2^{32}, i = 0, 1, \ldots, 31$, 则轮密钥生成方法为:

首先,

$$(K_0, K_1, K_2, K_3) = (\mathrm{MK}_0 \oplus \mathrm{FK}_0, \mathrm{MK}_1 \oplus \mathrm{FK}_1, \mathrm{MK}_2 \oplus \mathrm{FK}_2, \mathrm{MK}_3 \oplus \mathrm{FK}_3)$$

然后, 对 $i = 0, 1, 2, \ldots, 31$ ：

$$rk_i = K_{i+4} = K_i \oplus T'\left(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i\right)$$

说明:

(1) T' 变换与加密算法轮函数中的 T 基本相同, 只将其中的线性变换 L 修改为以下L':

$$L'(\mathrm{B}) = \mathrm{B} \oplus (\mathrm{B} <<< 13) \oplus (\mathrm{B} <<< 23);$$

(2) 系统参数 FK的取值, 采用 16 进制表示为:

$$\mathrm{FK}_0 = (\mathrm{A3\ B1BAC}), \mathrm{FK}_1 = (56\mathrm{AA}3350),$$
$$\mathrm{FK}_2 = (677\mathrm{D}9197), \mathrm{FK}_3 = (\mathrm{B}27022\mathrm{DC})$$

(3) 古定参数CK的取值方法为:

设 $ck_{i,j}$为 CK_i 的第 j 字节 $(i = 0, 1, \ldots, 31; j = 0, 1, 2, 3)$, 即 $CK_i = (ck_{i,0}, ck_{i,1}, ck_{i,2}, ck_{i,3}) \in \left(Z_2^8\right)^4$,则 $ck_{i,j} = (4i + j) \times 7 (\mathrm{mod}\ 256)$ 。 32 个固定参数 CK_i , 其 16 进制表示为:

00070e15	1c232a31	383f464d	545b6269
70777e85	8c939aa1	a8afb6bd	c4cbd2d9
e0e7eef5	fc030a11	181f262d	343b4249
50575e65	6c737a81	888f969d	a4abb2b9
c0c7ced5	dce3eaf1	f8ff060d	141b2229
30373e45	4c535a61	686f767d	848b9299
a0a7aeb5	bcc3cad1	d8dfe6ed	f4fb0209
10171e25	2c333a41	484f565d	646b7279

2.5 加密解密可逆分析

这里设加密密钥

$MK = (MK_0, MK_1, MK_2, MK_3), MK_i \in Z_2^{32}, i = 0, 1, 2, 3;$

令 $K_i \in Z_2^{32}, i = 0, 1, \ldots, 35$, 则可以生成轮密钥为 $rk_i \in Z_2^{32}, i = 0, 1, \ldots, 31,$

在加密中密钥$i : 0 \to 31$, 在解密中$i : 31 \to 0$, 即：每一轮的密钥一致。

设明文输入为 $(X_0, X_1, X_2, X_3) \in \left(Z_2^{32}\right)^4$

设 密文输出为$(Y_0, Y_1, Y_2, Y_3) \in \left(Z_2^{32}\right)^4$,

根据加密原则:

$$(Y_0, Y_1, Y_2, Y_3) = R\left(X_{32}, X_{33}, X_{34}, X_{35}\right) = (X_{35}, X_{34}, X_{33}, X_{32})$$

而加密流程

$$X_{i+4} = X_i \oplus T\left(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i\right), i = 0, 1, \ldots, 31.$$

设$C = T\left(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i\right)$, 则$X_i := X_{i+4} \oplus X_i$

故可解出X_i,以此类推得到：(X_0, X_1, X_2, X_3)

3 SM4算法流程

3.1 完整流程如下：

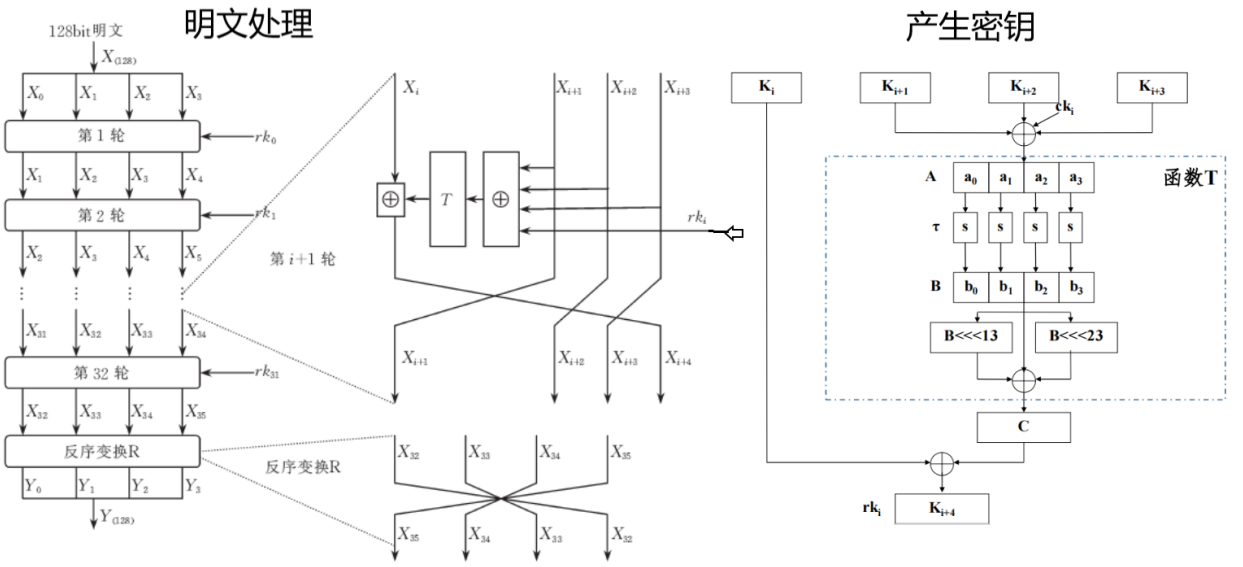


图 1 SMS4 加密算法整体结构

3.2 密钥扩展算法

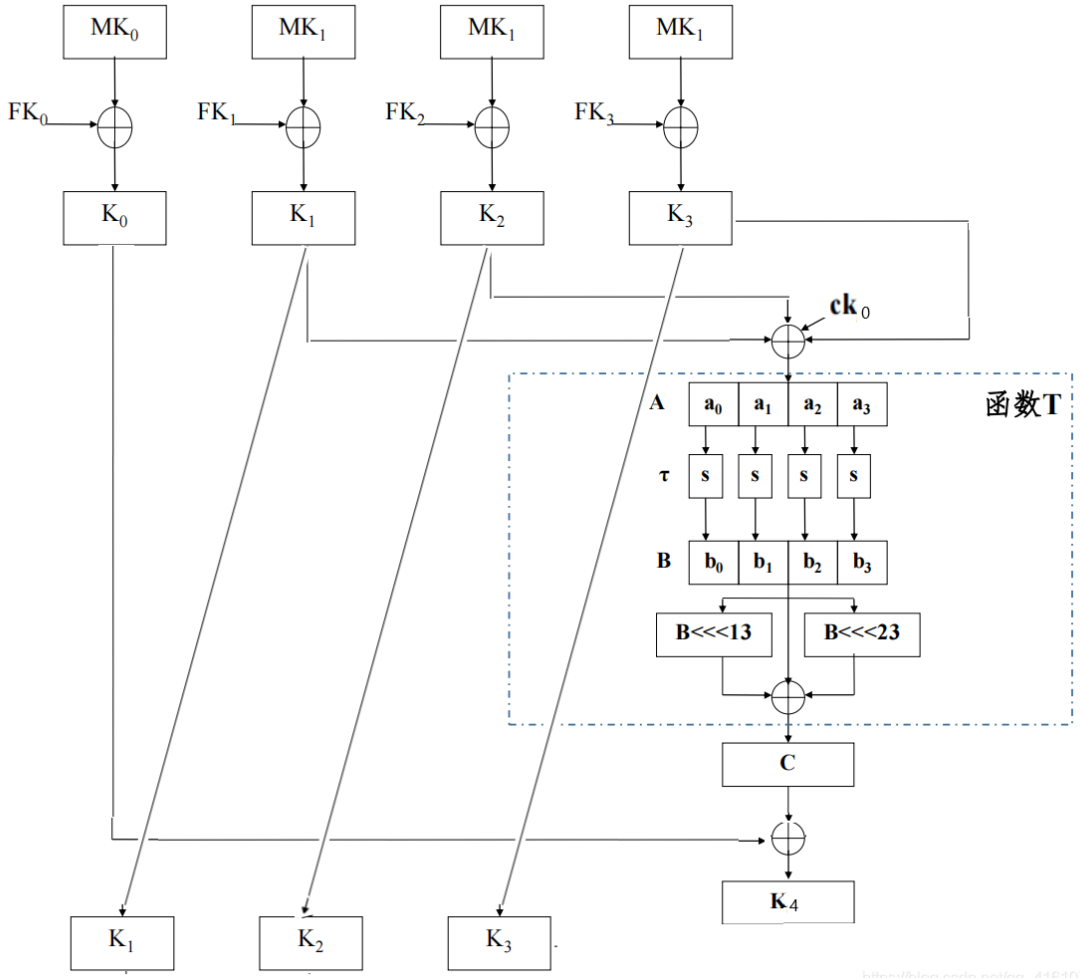
第一步：密钥与系统参数的异或:

$$(K_0, K_1, K_2, K_3) = (\mathrm{MK}_0 \oplus \mathrm{FK}_0, \mathrm{MK}_1 \oplus \mathrm{FK}_1, \mathrm{MK}_2 \oplus \mathrm{FK}_2, \mathrm{MK}_3 \oplus \mathrm{FK}_3)$$

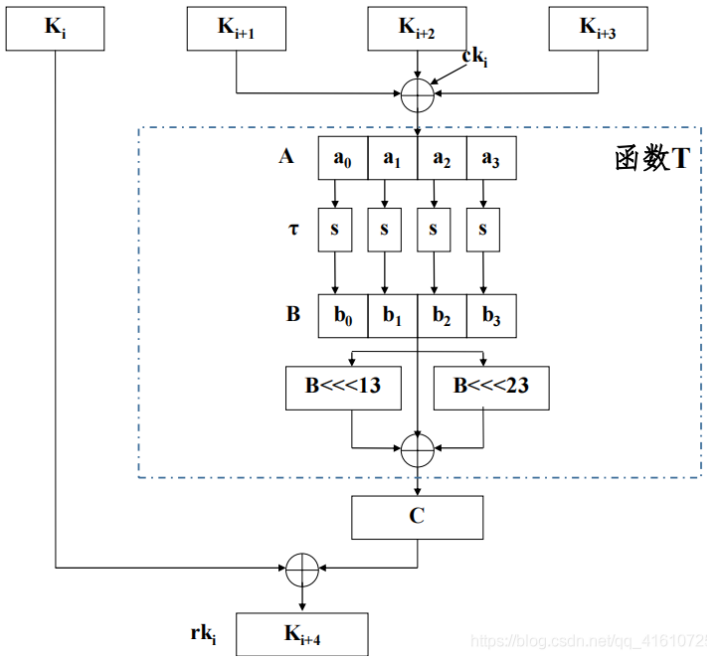
第二步：获取子密钥:

$$rk_i = K_{i+4} = K_i \oplus T'\left(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i\right)$$

流程如下：



函数T:



3.3 明文加密

明文处理大致分解为3步：

1）、将128bit的明文分成4个32bit的字(X_1, X_2, X_3, X_4)。

2）、将上述得到的字进行32轮的轮操作。

3）、最后将进行过32轮操作的4个字进行反序变换后组成128bit的密文。

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)$$
$$= X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i), i = 0, 1, \ldots, 31.$$
$$(Y_0, Y_1, Y_2, Y_3) = R(X_{32}, X_{33}, X_{34}, X_{35}) = (X_{35}, X_{34}, X_{33}, X_{32})$$

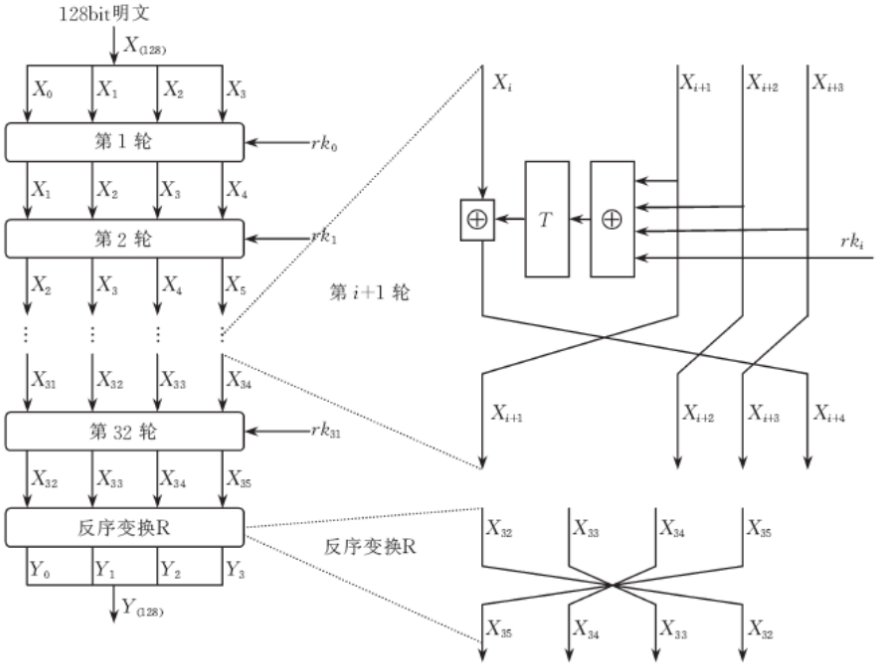
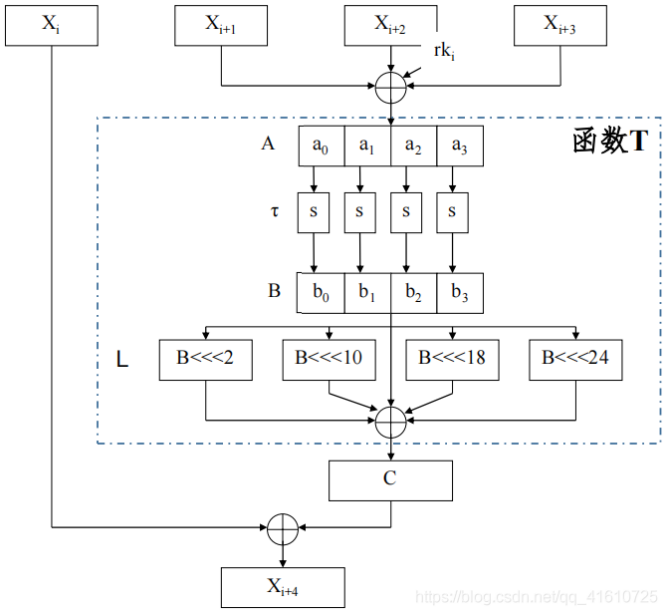


图 1　SMS4 加密算法整体结构



4 SM4代码介绍

4.1 定义S盒和FK,CK

如下：

```
# S盒
S_BOX = [0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2, 0x28, 0xFB, 0x2C, 0x05, 0x2B,
        0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06, 0x99, 0x9C, 0x42,
        0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43, 0xED, 0xCF, 0xAC, 0x62, 0xE4, 0xB3, 0x1C,
        0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA, 0x75, 0x8F, 0x3F, 0xA6, 0x47, 0x07, 0xA7, 0xFC,
        0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19, 0xE6, 0x85, 0x4F, 0xA8, 0x68, 0x6B, 0x81, 0xB2, 0x71,
        0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B, 0x70, 0x56, 0x9D, 0x35, 0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58,
        0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B, 0x01, 0x21, 0x78, 0x87, 0xD4, 0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27,
        0x52, 0x4C, 0x36, 0x02, 0xE7, 0xA0, 0xC4, 0xC8, 0x9E, 0xEA, 0xBF, 0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5,
        0xA3, 0xF7, 0xF2, 0xCE, 0xF9, 0x61, 0x15, 0xA1, 0xE0, 0xAE, 0x5D, 0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD,
        0x93, 0x32, 0x30, 0xF5, 0x8C, 0xB1, 0xE3, 0x1D, 0xF6, 0xE2, 0x2E, 0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29,
        0x23, 0xAB, 0x0D, 0x53, 0x4E, 0x6F, 0xD5, 0xDB, 0x37, 0x45, 0xDE, 0xFD, 0x8E, 0x2F, 0x03, 0xFF, 0x6A,
        0x72, 0x6D, 0x6C, 0x5B, 0x51, 0x8D, 0x1B, 0xAF, 0x92, 0xBB, 0xDD, 0xBC, 0x7F, 0x11, 0xD9, 0x5C, 0x41,
        0x1F, 0x10, 0x5A, 0xD8, 0x0A, 0xC1, 0x31, 0x88, 0xA5, 0xCD, 0x7B, 0xBD, 0x2D, 0x74, 0xD0, 0x12, 0xB8,
        0xE5, 0xB4, 0xB0, 0x89, 0x69, 0x97, 0x4A, 0x0C, 0x96, 0x77, 0x7E, 0x65, 0xB9, 0xF1, 0x09, 0xC5, 0x6E,
        0xC6, 0x84, 0x18, 0xF0, 0x7D, 0xEC, 0x3A, 0xDC, 0x4D, 0x20, 0x79, 0xEE, 0x5F, 0x3E, 0xD7, 0xCB, 0x39,
        0x48]

# 系统参数FK
FK = [0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc]

# 固定参数CK
CK = [0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269, 0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
      0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249, 0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
      0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229, 0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
```

```
        0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209, 0x10171e25,
0x2c333a41, 0x484f565d, 0x646b7279]
```

4.2 定义T函数

$T(x) = L(\tau(x))$

这里分别定义$\tau(x)$和$L(x)$

$\tau(x)$

```
    @staticmethod
    def _s_box(n: int):
        result = bytearray()
        # 将 32bit 拆分成 4x8bit, 依次进行S盒变换
        for item in list(n.to_bytes(4, 'big')):
            result.append(S_BOX[item])
        return int.from_bytes(result, 'big')
```

$L(x)$

```
    @staticmethod
    def _rot_left(n, m):
        """循环左移"""
        return ((n << m) | (n >> (32 - m))) & 0xFFFFFFFF
```

4.3 密钥扩展

```
    def _generate_key(self, key: bytes):
        """密钥生成"""
        key_r, key_temp = [0 for _ in range(32)], [0 for _ in range(4)]
        # 将 128bit 拆分成 4x32bit
        for i in range(4):
            temp = int.from_bytes(key[4 * i:4 * i + 4], 'big')
            key_temp[i] = temp ^ FK[i]
        # 循环生成轮密钥
        for i in range(32):
            box_in = key_temp[1] ^ key_temp[2] ^ key_temp[3] ^ CK[i]
            box_out = self._s_box(box_in)
            key_r[i] = key_temp[0] ^ box_out ^ self._rot_left(box_out, 13) ^
self._rot_left(box_out, 23)
            key_temp = key_temp[1:] + [key_r[i]]
        return key_r

```

4.4 主体循环（加密解密一样）

```
def _do(self, text: bytes, key_r: list):
        text_ = [0 for _ in range(4)]
        # 将 128bit 转化成 4x32bit
        for i in range(4):
            text_[i] = int.from_bytes(text[4 * i:4 * i + 4], 'big')
        for i in range(32):
            box_in = text_[1] ^ text_[2] ^ text_[3] ^ key_r[i]
            box_out = self._s_box(box_in)
            temp = text_[0] ^ box_out ^ self._rot_left(box_out, 2) ^
self._rot_left(box_out, 10)
            temp = temp ^ self._rot_left(box_out, 18) ^
self._rot_left(box_out, 24)
            text_ = text_[1:] + [temp]
        text_ = text_[::-1]  # 结果逆序
        # 将 4x32bit 合并成 128bit
        result = bytearray()
        for i in range(4):
            result.extend(text_[i].to_bytes(4, 'big'))
        return bytes(result)
```

4.5 加密解密接口

```
    def encrypt(self, plaintext: bytes):
        return self._do(plaintext, self._key_r)

    def decrypt(self, ciphertext: bytes):
        return self._do(ciphertext, self._key_r[::-1])
```

4.6 运行验证

这里只关注算法本身的加密，未实现分组密码的分组的加密模式和填充，故只支持标准的128bit输入和输出。

明文：`00112233445566778899aabbccddeeff`

密钥：`0123456789ABCDEFFEDCBA9876543210`

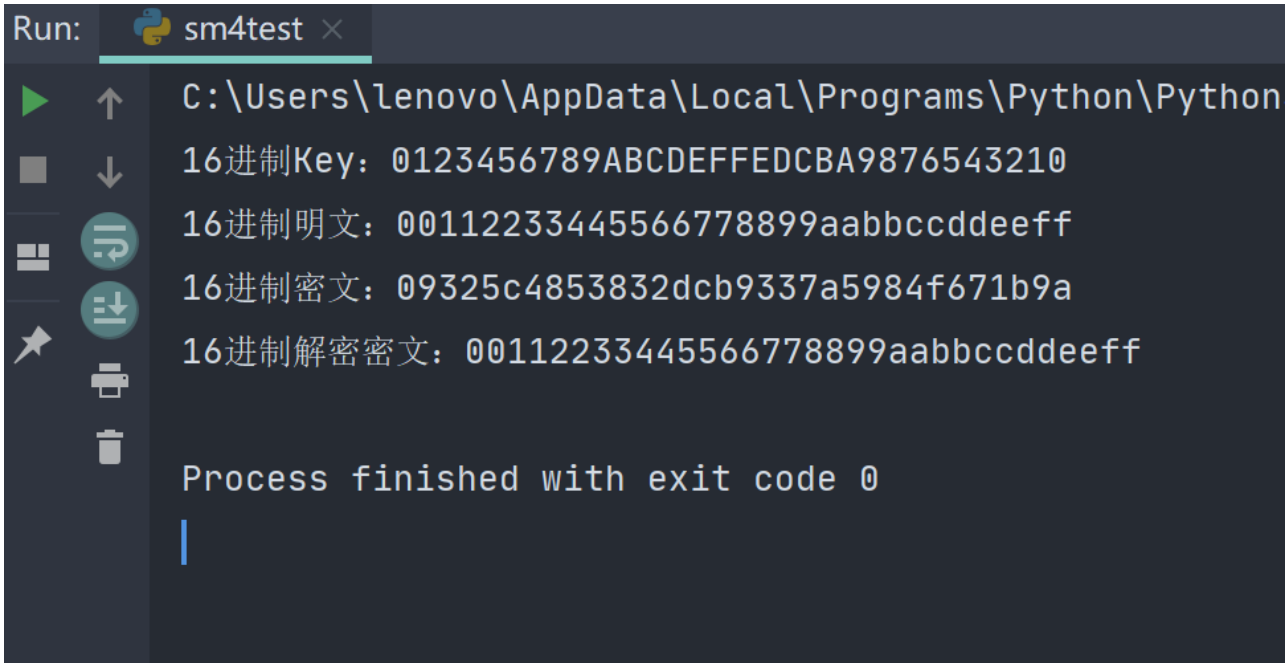
密文：`09325c4853832dcb9337a5984f671b9a`

运行代码：

```
if __name__ == "__main__":
    print("16进制Key：0123456789ABCDEFFEDCBA9876543210")
    # 128bit密钥
    key = bytes.fromhex("0123456789ABCDEFFEDCBA9876543210")
    print("16进制明文：00112233445566778899aabbccddeeff")
    # 128bit明文
```

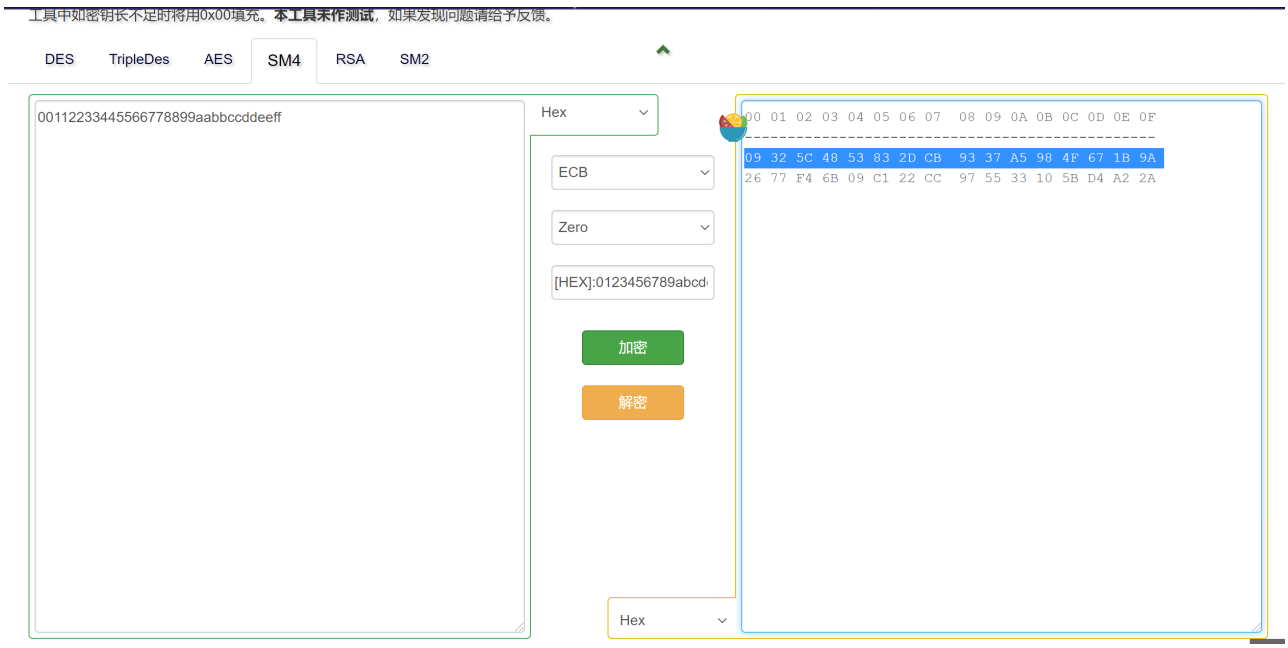
```
    plaintext = bytes.fromhex("00112233445566778899aabbccddeeff")
    sm4 = SM4Cipher(key)
    print("16进制密文："+ sm4.encrypt(plaintext).hex())
    # 09325c4853832dcb9337a5984f671b9a
    encryption = sm4.encrypt(plaintext).hex()
    encryp_txt = bytes.fromhex(str(encryption))

    print("16进制解密密文："+sm4.decrypt(encryp_txt).hex())
```

结果：



网站验证：

网站：SM4加密网站：https://the-x.cn/cryptography/Sm4.aspx



5 SM4安全分析

1977年，DES成为美国政府的商用加密标准，并授权在非密级政府通信中使用，随后该算法在国际上得到广泛使用。但该算法的56 bit密钥太短，已不适合用于当今分布式开放网络对数据加密安全性的要求。AES算法在此阶段应运而生，并最终成为取代DES的新一代数据加密标准。在DES向AES过渡的过程中，NIST将3DES指定为过渡的加密标准，3DES是DES的一个安全变形，通过执行3次DES达到增加密钥长度和安全性的目的。2012年3月21日，我国国家密码管理局发布了SM4算法（原SMS4分组密码算法）。SM4、AES与3DES算法整体特性如图所示。

算法	分组长度/bit	密钥长度/bit	迭代轮数	算法结构	算法特性
SM4	128	128	32	非平衡 Feistel 网络	基本轮函数加迭代，含非线性变换
AES	128	128/182/256	10/12/14	SP（代换－置换）网络	排列、置换加迭代，含非线性变换
3DES	64	112/168	16×3=48	平衡 Feistel 网络	使用标准的算术和逻辑运算，先替代后置换，不含非线性变换

SM4的安全强度和计算效率介于AES与3DES之间。考虑3DES算法安全性较低，且现有的应用系统正逐步用AES替代3DES。

5.1 分组算法计算量与安全性

在计算量方面，SM4与AES-128算法的计算量差别较大，因此不做计算量的统计与直接对比。在安全性方面，SM4的安全强度等同于AES-128，但是近年来一些密码分析表明，SM4的安全性略弱于AES-128[15]。由于SM4的密钥长度固定为128 bit，没有提供更长的可选密钥长度，在安全等级要求越来越高的情况下，SM4可能面临应用范围受限的问题。

对SM4和AES算法进行性能测试，测试方法为对每轮数据循环迭代加密，循环次数N=10000，测试算法的加/解密运行速度。

5.1.1 算法性能对比

对SM4和AES算法进行ECB、CBC工作模式下的性能测试，以AES为基准，ECB模式SM4与AES算法运行速度比如表8所示，CBC模式SM4与AES算法运行速度比如表9所示。由表8可知，ECB模式下AES加密性能高于OpenS SL SM4，解密性能OpenS SL SM4略高于AES。通过对比可以发现，A厂商优化后的SM4算法加/解密速度均优于标准AES软实现，约为标准AES软实现的3.5倍，为OpenS SL SM4的4倍，但与硬件AES-NI仍差距较大。

表 8 ECB 模式 SM4 与 AES 算法运行速度比

加 / 解密	输入长度	A 厂商 SM4/ AES	OpenSSL SM4/ AES	AESEVP/ AES
加密	32 B	0.99	0.81	10.69
	128 B	2.9	0.83	25.08
	512 B	3.34	0.83	31.96
	1 KB	3.35	0.84	33.54
	4 KB	3.43	0.81	34
	16 KB	3.54	0.82	35.1
	64 KB	3.5	0.82	35.19
	256 KB	3.53	0.82	35.4
	1 MB	3.52	0.82	34.48
解密	32 B	1.31	1.07	11.13
	128 B	3.92	1.12	29.96
	512 B	4.47	1.12	40.97
	1 KB	4.47	1.12	43.08
	4 KB	4.57	1.10	43.54
	16 KB	4.71	1.11	45.04
	64 KB	4.73	1.08	45.44
	256 KB	4.74	1.13	45.7
	1 MB	4.71	1.12	45.48

由表9可知，CBC模式下AES加密性能高于OpenS SL SM4，OpenS SL SM4解密性能略高于AES，整体效果无较大差异。通过对比可以发现，A厂商优化后的SM4算法加/解密速度均优于标准AES软实现，其中加密运算约为标准AES软实现的1.5倍，为OpenS SL SM4的1.3倍；解密运算约为标准AES软实现的4.6倍，为OpenS SL SM4的4.2倍，但加/解密均与硬件AES-NI的差距较大。

5.1.2 硬件加速对算法性能的影响

OpenS SL提供了EVP接口，此接口封装了多种密码算法，在实际使用时，通常会调用AES算法的EVP接口来实现加/解密运算。EVP接口支持调用AES-NI硬件加速技术，使算法性能大幅度提升。由上述分析可知，SM4加/解密性能整体上与AES相差不大，在加密运算上性能稍差。经过A厂商优化后的SM4性能有显著提升，且明显优于AES软实现，但与硬件AES-NI仍存在较大差距。

5.2 SM4的攻击

5.2.1 概述

SM4密码算法是中国发布的商用分组密码算法，也是目前中国密码行业标准以及国家信息安全技术标准密码算法之一。针对SM4密码算法的侧信道攻击研究，主要是针对SM4密码算法的侧信道故障分析攻击，以及针对 S M 4 密码算法的侧信道相关性能量分析攻击等研究内容。

5.2.2 模板攻击

模板攻击通过采集可控的密码芯片的能量曲线来构建模板最启迸行模板匹配? 完成对密码芯片的攻击，模板攻击通常由两个阶段构成：第一阶段：是模板构建，第二阶段是模板匹配。

详细参考论文：针对SM4密码算法的模板攻击_匡晓云:https://chn.oversea.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFD&dbname=CJFDLAST2021&filename=CDQX202105004&uniplatform=OVERSEAS_CHS&v=2Y_ZEuZQTVR98KIsy47vaNQLwVYf-gw0LrERs4WAK8gkuC_QF0YlOpSbShEtOYp9

5.3 SM4改进

5.3.1 基于动态思想的 SM4 算法改进方法

M4 算法内部的加密要素包括：系统参数 FK、固定参数 CK 以及S 盒等。由于这些加密要素都是固定公开的，这对攻击者利用统计学方法破解密文提供了便利，使得 SM4 加密算法存在安全隐患。

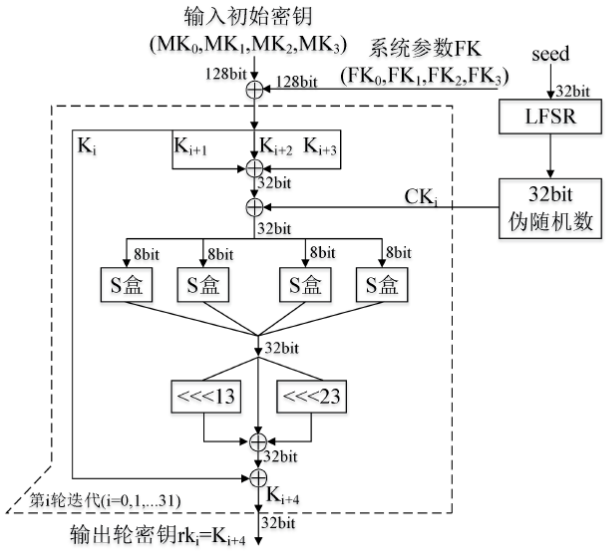
动态思想是在不改变密码算法结构的前提下，通过随机变换密码中的一些加密要素（参数、表、函数等），以实现动态加密机制的设计思想。基于动态思想，在保证算法正确合理的基础上，将 SM4 算法中的加密要素由原本“固定公开”变为随机函数“随机秘密”生成，并在其余加密组件的配合下，就能实现“一次一变”的加密机制。随机函数的结果是生成动态加密机制中加密要素的依据，只要将这个结果与密钥一起发送给接收方，接收方就能正确解密出明文。

目前针对加密要素 CK 参数、FK 参数的改进方法大多基于预存储方法。这类方法将原始的参数进行数量上的扩展，扩展为一个由 n 个参数构成的参数查找表。SM4密钥扩展算法进行密钥扩展时，会随机选择参数查找表中的一个参数参与计算。预存储方法下参数的随机性与预先存储的参数个数有关，为了具有更好的随机性，往往需要预存储很多参数，这会显著增加算法的资源消耗。有研究将 SM4 算法中使用的单一 S 盒替换为两个，一个是 SM4 算法中的原 S 盒，另一个是赵尔凡等人设计的基于“双混沌映射”的 S 盒。该方法利用算法迭代轮数的奇偶性来选择 S 盒，迭代轮数为奇数时选用原 S 盒，为偶数时选用引入的新 S 盒。这种由算法轮数的奇偶来选用 S 盒的方式随机性不足，若在攻击者已知新 S 盒的情况下，该方法与原 SM4 算法差别不大。

5.3.2 基于 LFSR 动态生成 CK 参数的密钥扩展算法

线性反馈移位寄存器 LFSR 是一种常见的产生伪随机数的方法。LFSR 由 n 个 D 触发器和若干个异或门组成，其输出是 n 位二进制数。g0~gn 是反馈系数，它们的取值为 0 或 1（gn 必须为 1）。给定 n 个 D 触发器的初始值（常被称为随机种子 seed），下一次时钟有效沿到来时，n 个 D 触发器的输出值（Q1~Qn）会更新，且更新的值与初始值和反馈系数有关。

为了提高 SM4 算法的安全性，本文将 SM4 密钥扩展算法中参与每一轮迭代计算的固定参数 CKi（i=0~31）由 LFSR 生成的伪随机数代替。基于 LFSR 动态生成 CK 参数的 SM4 密钥扩展算法如图所示。



密钥扩展时，图 中 LFSR（n=32）生成的 32bit 伪随机数代替原本固定公开的CK 参数参与每一轮迭代计算。LFSR（n=32）生成的伪随机数最多有 $2^{32}-1$ 种状态，故基于 LFSR 生成的 CK 参数具有一定的随机特性。

随机种子 seed 提供 LFSR 内部 32 个 D 触发器的初始值，反馈系数决定了伪随机数的状态转移图，算法使用者可以不对外公开自己设定的随机种子 seed 与反馈系数的值，故基于 LFSR 生成的 CK 参数具有一定秘密性。由于其他人难以得知每一轮随机生成的CK 参数的具体值，所以提高了算法被破译的难度。

只要加、解密双方采用相同结构的 LFSR，并且加密方将随机种子 seed 与密钥一起传递给解密方，解密方的密钥扩展算法就能正确地生成加密时使用的轮密钥 rk_i，进而正确地解密密文。

基于 LFSR 动态选择 S 盒的密钥扩展及加密算法

6 附录

6.1 完整代码

```
"""
@Time    : 2021/12/5
@Author  : LowlyLi
@Version : 1.0
@File    : SM4.py
@Introduce: SM4 国密4
"""

# S盒
S_BOX = [0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2, 0x28, 0xFB, 0x2C, 0x05, 0x2B,
         0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06, 0x99, 0x9C, 0x42,
         0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43, 0xED, 0xCF, 0xAC, 0x62, 0xE4, 0xB3, 0x1C,
         0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA, 0x75, 0x8F, 0x3F, 0xA6, 0x47, 0x07, 0xA7, 0xFC,
         0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19, 0xE6, 0x85, 0x4F, 0xA8, 0x68, 0x6B, 0x81, 0xB2, 0x71,
```

```
        0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B, 0x70, 0x56, 0x9D, 0x35,
0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58,
        0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B, 0x01, 0x21, 0x78, 0x87, 0xD4,
0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27,
        0x52, 0x4C, 0x36, 0x02, 0xE7, 0xA0, 0xC4, 0xC8, 0x9E, 0xEA, 0xBF,
0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5,
        0xA3, 0xF7, 0xF2, 0xCE, 0xF9, 0x61, 0x15, 0xA1, 0xE0, 0xAE, 0x5D,
0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD,
        0x93, 0x32, 0x30, 0xF5, 0x8C, 0xB1, 0xE3, 0x1D, 0xF6, 0xE2, 0x2E,
0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29,
        0x23, 0xAB, 0x0D, 0x53, 0x4E, 0x6F, 0xD5, 0xDB, 0x37, 0x45, 0xDE,
0xFD, 0x8E, 0x2F, 0x03, 0xFF, 0x6A,
        0x72, 0x6D, 0x6C, 0x5B, 0x51, 0x8D, 0x1B, 0xAF, 0x92, 0xBB, 0xDD,
0xBC, 0x7F, 0x11, 0xD9, 0x5C, 0x41,
        0x1F, 0x10, 0x5A, 0xD8, 0x0A, 0xC1, 0x31, 0x88, 0xA5, 0xCD, 0x7B,
0xBD, 0x2D, 0x74, 0xD0, 0x12, 0xB8,
        0xE5, 0xB4, 0xB0, 0x89, 0x69, 0x97, 0x4A, 0x0C, 0x96, 0x77, 0x7E,
0x65, 0xB9, 0xF1, 0x09, 0xC5, 0x6E,
        0xC6, 0x84, 0x18, 0xF0, 0x7D, 0xEC, 0x3A, 0xDC, 0x4D, 0x20, 0x79,
0xEE, 0x5F, 0x3E, 0xD7, 0xCB, 0x39,
        0x48]

# 系统参数FK
FK = [0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc]

# 固定参数CK
CK = [0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269, 0x70777e85,
0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
      0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249, 0x50575e65,
0x6c737a81, 0x888f969d, 0xa4abb2b9,
      0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229, 0x30373e45,
0x4c535a61, 0x686f767d, 0x848b9299,
      0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209, 0x10171e25,
0x2c333a41, 0x484f565d, 0x646b7279]


class SM4Cipher:
    def __init__(self, key: bytes):
        if not len(key) == 16:
            raise ValueError("SM4 key must be length of 16. ")
        self._key_r = self._generate_key(key)
        self.block_size = 16

    def encrypt(self, plaintext: bytes):
        return self._do(plaintext, self._key_r)

    def decrypt(self, ciphertext: bytes):
        return self._do(ciphertext, self._key_r[::-1])
```

```
49
50      def _do(self, text: bytes, key_r: list):
51          text_ = [0 for _ in range(4)]
52          # 将 128bit 转化成 4x32bit
53          for i in range(4):
54              text_[i] = int.from_bytes(text[4 * i:4 * i + 4], 'big')
55          for i in range(32):
56              box_in = text_[1] ^ text_[2] ^ text_[3] ^ key_r[i]
57              box_out = self._s_box(box_in)
58              temp = text_[0] ^ box_out ^ self._rot_left(box_out, 2) ^
    self._rot_left(box_out, 10)
59              temp = temp ^ self._rot_left(box_out, 18) ^
    self._rot_left(box_out, 24)
60              text_ = text_[1:] + [temp]
61          text_ = text_[::-1]  # 结果逆序
62          # 将 4x32bit 合并成 128bit
63          result = bytearray()
64          for i in range(4):
65              result.extend(text_[i].to_bytes(4, 'big'))
66          return bytes(result)
67
68      def _generate_key(self, key: bytes):
69          """密钥生成"""
70          key_r, key_temp = [0 for _ in range(32)], [0 for _ in range(4)]
71          # 将 128bit 拆分成 4x32bit
72          for i in range(4):
73              temp = int.from_bytes(key[4 * i:4 * i + 4], 'big')
74              key_temp[i] = temp ^ FK[i]
75          # 循环生成轮密钥
76          for i in range(32):
77              box_in = key_temp[1] ^ key_temp[2] ^ key_temp[3] ^ CK[i]
78              box_out = self._s_box(box_in)
79              key_r[i] = key_temp[0] ^ box_out ^ self._rot_left(box_out, 13)
    ^ self._rot_left(box_out, 23)
80              key_temp = key_temp[1:] + [key_r[i]]
81          return key_r
82
83      @staticmethod
84      def _s_box(n: int):
85          result = bytearray()
86          # 将 32bit 拆分成 4x8bit，依次进行S盒变换
87          for item in list(n.to_bytes(4, 'big')):
88              result.append(S_BOX[item])
89          return int.from_bytes(result, 'big')
90
91      @staticmethod
92      def _rot_left(n, m):
93          """循环左移"""
```

```
        return ((n << m) | (n >> (32 - m))) & 0xFFFFFFFF

if __name__ == "__main__":
     # 128bit密钥
    key = bytes.fromhex("0123456789ABCDEFFEDCBA9876543210")
    # 128bit明文
    plaintext = bytes.fromhex("00112233445566778899aabbccddeeff")
    sm4 = SM4Cipher(key)
    print(sm4.encrypt(plaintext).hex())
    # 09325c4853832dcb9337a5984f671b9a
    encryption = sm4.encrypt(plaintext).hex()
    encryp_txt = bytes.fromhex(str(encryption))
    print(sm4.decrypt(encryp_txt).hex())
    # 00112233445566778899aabbccddeeff

"""
# 样例一
key = "0123456789ABCDEFFEDCBA9876543210"  # 16进制字符串
plaintext = "0123456789ABCDEFFEDCBA9876543210"  # 16进制字符串
ciphertext = "681edf34d206965e86b3e94f536e4246"  # 16进制字符串

# 样例二
key = "0123456789ABCDEFFEDCBA9876543210"  # 16进制字符串
plaintext = "00112233445566778899aabbccddeeff"  # 16进制字符串
ciphertext = "09325c4853832dcb9337a5984f671b9a"  # 16进制字符串

# 样例三
key = "456789ABCDEFFEDCBA98765432100123"  # 16进制字符串
plaintext = "2233445566778899aabbccddeeff0011"  # 16进制字符串
ciphertext = "58ab414d84fb3008b0bee987f97021e6"  # 16进制字符串

# 样例四
key = "89ABCDEFFEDCBA9876543210012345 67"  # 16进制字符串
plaintext = "445566778899aabbccddeeff00112233"  # 16进制字符串
ciphertext = "5937a929a2d9137216c72a28cd9cf619"  # 16进制字符串

"""
```