Acronym

AES	Advanced Encryption Standard
SM4	A block cipher used in the Chinese National Standard
WLAN	Wireless Local Area Network
DPA	Differential Power Analysis
LSB	Least Significant Bit
MSB	Most Significant Bit
EM	Electromagnetic
$\oplus$	XOR
$<< i$	Shift i bit to the left
MI	Multiplicative Inverse

Project : Model the mask implementation of SM4 in C

The project aims to implement the masked SM4 algorithm in C. SM4 is a block cipher officially published by Chinese government [2], and already widely applied on WLAN products. Since it has a similar structure as AES, it is possible to build AES and SM4 in the same hardware. By doing so, it's promising to increase the diversity of Tiempo's product.

Since AES is vulnerable to DPA, the implementation of AES in hardware is mostly masked. For SM4, a study [3] also reveals that SM4 cannot resist DPA attack, and several papers have proposed masking techniques to reduce information leakage. The key step of masking lies in the only nonlinear operation called Sub-Bytes (s-box). Therefore, in this part, we firstly develop SM4 and AES s-box algorithm in the similar coding structure. Then, we try to build SM4 s-box in the function of AES s-box. Since the masked AES has already implemented in Tiempo's product, by doing this, it's promising that these two kinds of block cypher can be integrated based on the existing hardware.

Apart from combining SM4 and AES s-box, we also refer to published studies and simulate masked s-box for SM4 in C. On the other hand, we implement s-box through composite field. In many studies, building s-box through composite field helps to save cost in hardware implementation.

Brief

In the second project, firstly we take a look at the encryption process of AES and SM4. Then, we focus on s-box (sub bytes) algorithm. This part is a nonlinear function and the fundamental computing unit of AES and SM4 that occupies most of area and power consumption in circuits. Since this is the critical part for the hardware implementation, and also the vulnerable part that suffers from side-channel attack[Note], it is important to analyze the s-box and then optimize it. Therefore, in this report we begin with software simulation to analyze the s-box. To further realize s-box algorithm in C, the first step is to understand its algebraic structure. After having that concept, we introduce our software implementation of both AES and SM4 s-box.

In Tiempo's product, the masked AES s-box has already been implemented in hardware. To base on this and develop its own masked SM4 s-box, we firstly try to simulate SM4 s-box in function of AES s-box in the software. Then, we develop another masked SM4 s-box which is independent of AES s-box. Furthermore, for future implementation on hardware with a more efficient use and optimized chip area, we study how to generate s-box in composite field and then realize it in software.

[Note] side-channel attack is a kind of attack in which an attacker tries to get secret data manipulated by a device by observing signals emanating from it. There are several types of emanation that can be used in practice such as electromagnetic (EM fields leak information when the chip's processors run their functions and algorithms), power monitoring (attacker studies the power consumption of a cryptographic hardware device), timing attack (attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms).

Development

1. Introduction

1.1 AES

Advanced Encryption Standards (AES) is a subset of Rijndael block cipher that is applied to encrypt electronic data. It has been adopted by the U.S government and now used worldwide. *Figure 9* shows the AES encryption process. The input takes a plaintext with size of 128 bits (16 bytes) and a key which length can be 128, 192 or 256 bits (16, 24, 32 bytes). With different key sizes, the cipher executes different number of rounds : 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key. *Figure 8* compares the plaintext, key size and cipher rounds of AES and SM4.

	plaintext (bits)	key size (bits)	cipher rounds
AES	128	128	10
		192	12
		256	14
SM4	128	128	32

Figure 8. AES and SM4 comparison

The encryption process consists of four steps. There are substitute bytes (usually called “**sub bytes**”), **shift rows**, **mix columns** and **add round key**. In “sub bytes”, an s-box is used to perform a byte by byte substitution of the block. Then, do a simple permutation in “shift rows”. And then, in “mix columns” process, each byte of a column is mapped by a given matrix into a new value. For “add round key”, the plaintext block are bitwise XORed with the round key.

(Download the code : https://github.com/YWsGithub/s-box/tree/master/AES_complete)

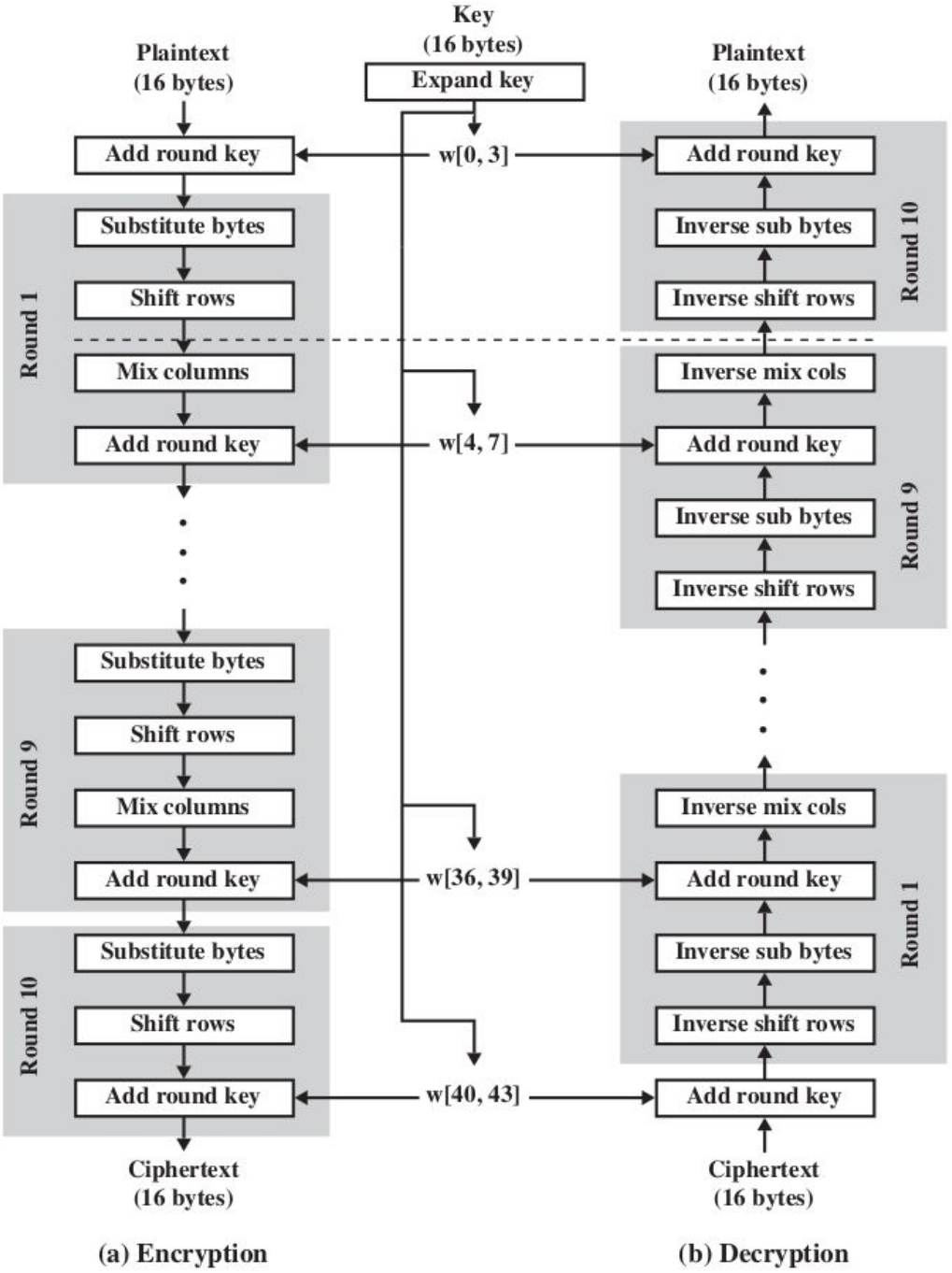

Figure 9. AES Encryption Process [5]

1.2 SM4

SM4 is the first published comercial block cipher standard of China and is widely used in WLAN products. Followed the notations in official standard [2], Z_2^e is the set of e-bit vector, so Z_2^{32} are called 32-bit words, and the elements of Z_2^8 are called 8-bit characters or bytes. Similar to AES, the input takes a plaintext with size of 128 bits, expressed as $(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$, but the key size is limited to only 128 bits which is denoted as $(MK_0,$

$MK_1, MK_2, MK_3) \in (Z_2^{32})^4$. *Figure 10* shows SM4 encryption process : The plaintext executes 32 times of rund functions with nonlinear substitutions and finally do the reverse substitution. Here,

$$R(X_{32}, X_{33}, X_{34}, X_{35}) = (X_{35}, X_{34}, X_{33}, X_{32}) = (Y_0, Y_1, Y_2, Y_3) \in (Z_2^{32})^4$$

The round function in *Figure 10* can be written as a function :

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

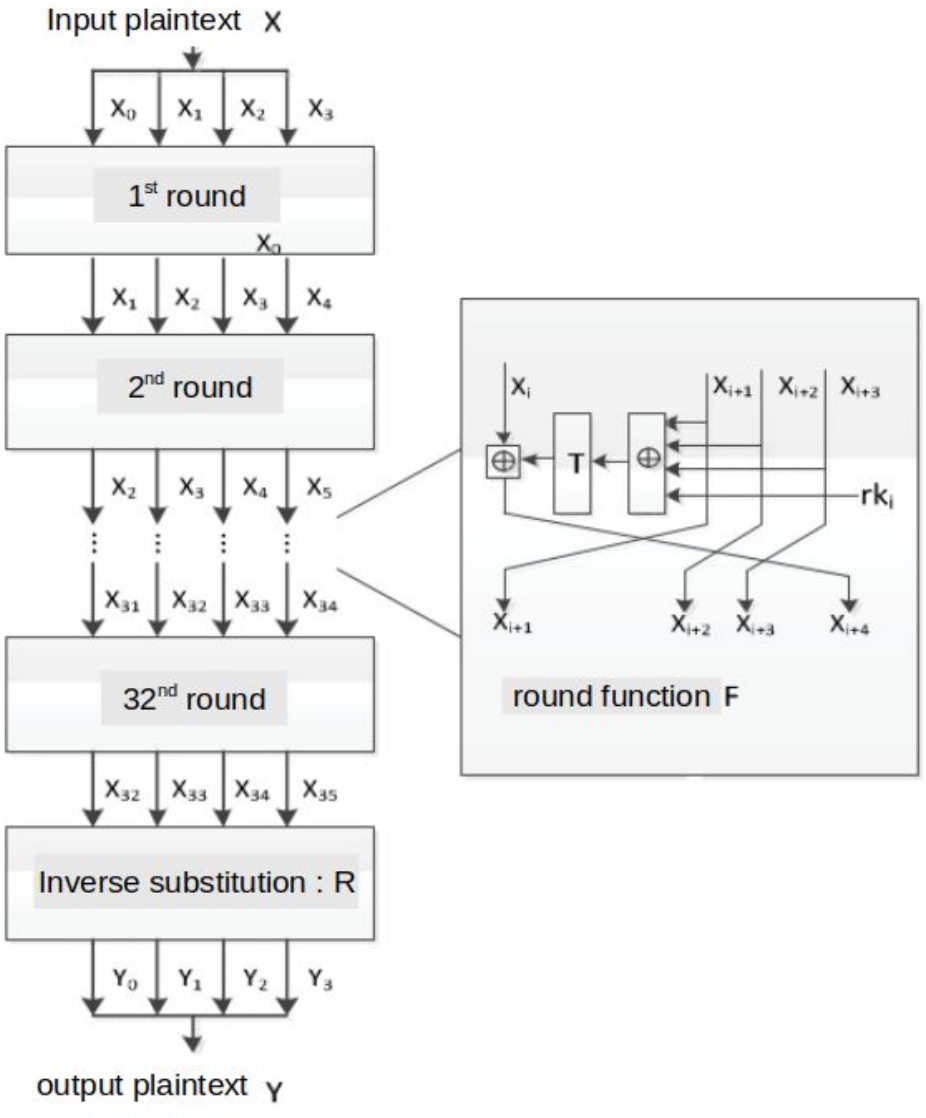

Figure 10. SM4 Encryption Process [6]

Figure 11. shows the SM4 round structure denoted as the round function *F*, and *T* is a substitution that generates 32 bits output from 32 bits input : $T(.) = L(\tau(.))$ where τ is used for a non-linear substitution with s-box. *L* is a linear substitution :

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

In the round function, there is a round key $rk_i \in Z_2^{32}$, $i = 0, 1, \ldots, 31$:

$$rk_i = K_{i+4} = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i) , K_i \in Z_2^{32}$$
$$(K_0, K_1, K_2, K_3) = (MK_0 \oplus FK_0, MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3)$$
$$T'(.) = L'(\tau(.)), L' = x \oplus (x<<13) \oplus (x<<23)$$

$FK = (FK_0 , FK_1 , FK_2 , FK_3)$ and $CK = (CK_0 , CK_1 , \ldots, CK_{31})$ are two kinds of 32-bits constant parameters referred in [2].

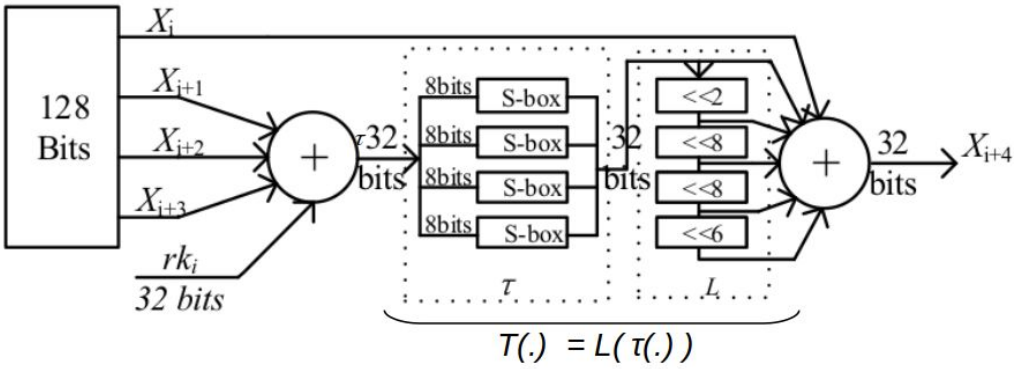

Figure 11. A round structure of SM4 [7]

For this project, we provide complete SM4 algorithm in C. Our code uses the test vector which is exactly the same as the one in official document [2]. As shown in *Figure 12*, in our main function, we firstly print out the test vector (plaintext), then input the key and do the encryption for one time to see the result meets the official document or not. After first confirmation, we decrypt the result to check whether the code can reverse back the original plaintext. In addition, we encrypt the plaintext for one million times and print the result out to check it is as expected or not. Then, we do one million decryption to this result for making sure the whole code functions well.

(Download code : https://github.com/YWsGithub/s-box/tree/master/SM4_complete)

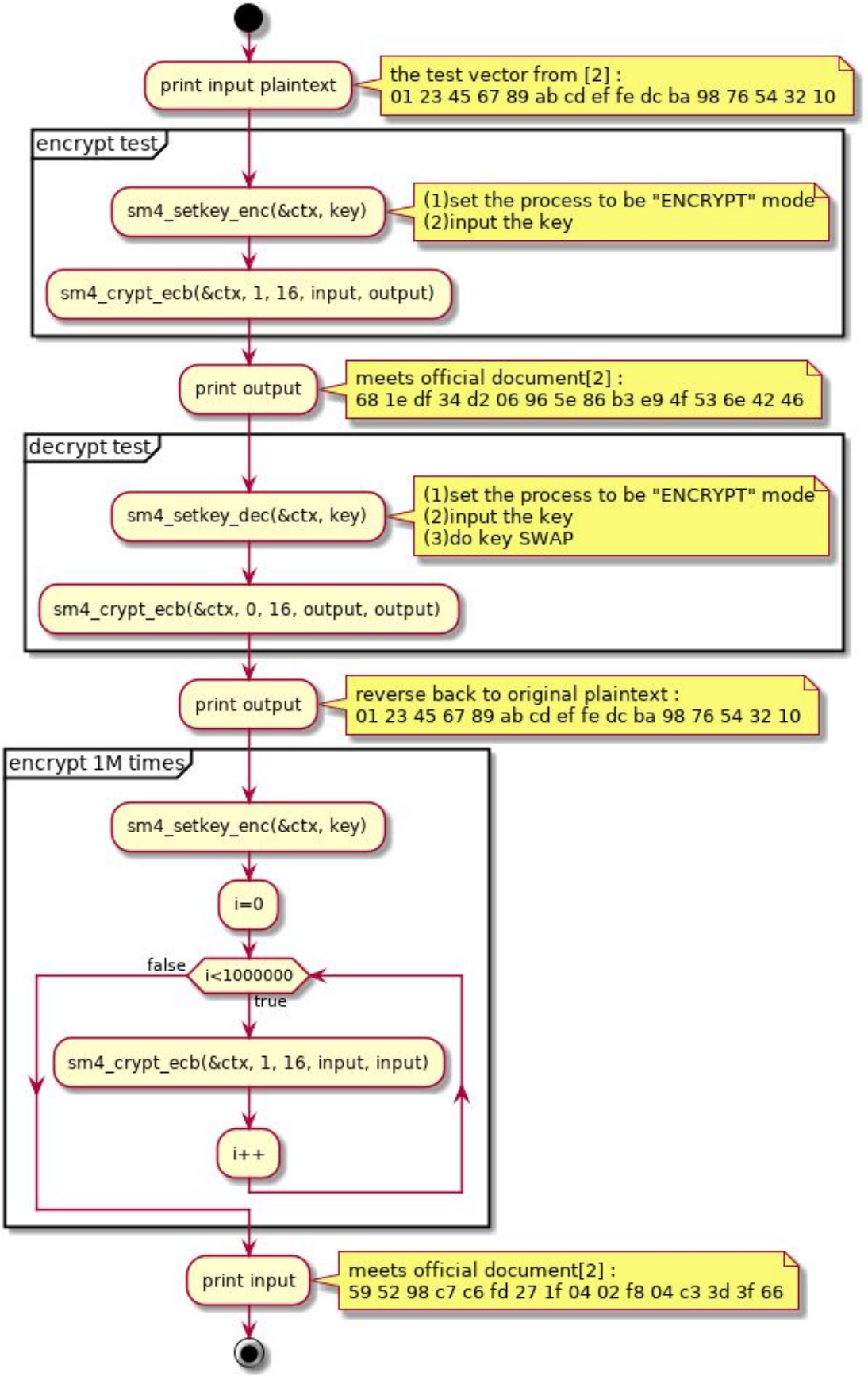

Figure 12. main function of SM4 encryption testing

2. S-box

The S-box for AES and SM4 is a look-up table with 8-bit input and 8-bit output. In part of the encryption, S-box offers good non-linear properties. During this process, each byte in plaintext expressed as two hexadecimal digits (8-bit) is taken as input value, and this input value will be substituted by corresponding output value from s-box map. For example, the input value is $\{83\}_{16}$, then we find the value at 8^{th} row / 3^{rd} column of S-box. As the *Figure 13 shows*, in AES S-box, we can get the output of $\{EC\}_{16}$. As to SM4 S-box, we obtain output $\{D2\}_{16}$.

(a) 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

(b) SM4 s-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	D6	90	E9	FE	CC	E1	3D	B7	16	B6	14	C2	28	FB	2C	05
10	2B	67	9A	76	2A	BE	04	C3	AA	44	13	26	49	86	06	99
20	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	ED	CF	AC	62
30	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	FA	75	8F	3F	A6
40	47	07	A7	FC	F3	73	17	BA	83	59	3C	19	E6	85	4F	A8
50	68	6B	81	B2	71	64	DA	8B	F8	EB	0F	4B	70	56	9D	35
60	1E	24	0E	5E	63	58	D1	A2	25	22	7C	3B	01	21	78	87
70	D4	00	46	57	9F	D3	27	52	4C	36	02	E7	A0	C4	C8	9E
80	EA	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	A1
90	E0	AE	5D	A4	9B	34	1A	55	AD	93	32	30	F5	8C	B1	E3
a0	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0D	53	4E	6F
b0	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6D	6C	5B	51
c0	8D	1B	AF	92	BB	DD	BC	7F	11	D9	5C	41	1F	10	5A	D8
d0	0A	C1	31	88	A5	CD	7B	BD	2D	74	D0	12	B8	E5	B4	B0
e0	89	69	97	4A	0C	96	77	7E	65	B9	F1	09	C5	6E	C6	84
f0	18	F0	7D	EC	3A	DC	4D	20	79	EE	5F	3E	D7	CB	39	48

Figure 13. S-box of (a) AES (b) SM4

Because AES and SM4 s-box have similar structure, they can be built in the same algebraic basis. Here we are going to generate these two s-boxes with shared functions in C. However, in order to achieve that, it is necessary to understand certain algebraic structures of s-box before.

For both AES and SM4, the S-box construction are based on two processes : **multiplicative inverse (MI)** and **affine transformations**. For the 8-bits input and output, each byte is interpreted as polynomials over Galois field (GF) of 2^8 called **Rijndael's finite field**. For instance, the $\{83\}_{16} = \{10000011\}_2$ can be expressed as a polynomial x^7+x+1. When it comes to calculate multiplicative inverse of x^7+x+1, in Rijndael's finite field it employs $x^8+x^4+x^3+x+1$ as its irreducible polynomial, which can be written as $\{100011011\}_2$ and $\{11b\}_{16}$. (In our code, we use hexadecimal expression)

3. S-box Algebraic Basis

3.1 Multiplicative Inverse (MI)

In numerical calculation, multiplicative inverse is simply the reciprocal of a number *x*, which is *1/x*. However, when calculating multiplicative inverse of polynomials in Rijndael's finite field, it is more complex than that. To have a clearer view, please see the calculation below :

$$\{83\}_{16} \equiv x^7+x+1$$
$$\{80\}_{16} \equiv x^7$$

the multiplication of $\{83\}_{16}$ and $\{80\}_{16}$ is :

$$(x^7+x+1)(x^7) = x^{14}+x^8+x^7$$

then, divide the result above by irreducible polynomial $x^8+x^4+x^3+x+1$, then we get the residue = 1 :

$$x^{14}+x^8+x^7 = (x^8+x^4+x^3+x+1)(x^6+x^2+x+1)+1$$

This can also be expressed as :

$$(100000110000000 \bmod 1000111011)_2 = \{4180 \bmod 11b\}_{16} = \{01\} = 1$$

In this case, we can denote $\{83\}_{16} \bullet \{80\}_{16} = \{01\}_{16}$ and the multiplicative inverse of $\{83\}_{16}$ is $\{80\}_{16}$.

In the demonstration above, we take a known result $\{80\}_{16}$ to explain. However, in real cases, we only have inputs such as $\{83\}_{16}$ and require for its multiplicative inverse. To get the multiplicative inverse, the extended Euclidean algorithm is widely used in cryptography.

Figure 14 shows how extended Euclidean algorithm operates. Take the example above, *a(x)* is the irreducible polynomial $x^8+x^4+x^3+x+1$ and *b(x)* is the input $\{83\}_{16} \equiv x^7+x+1$. Started with the division of *a(x)* and *b(x)*, and then take the remainder *r* and

quotient *q* to calculate *w(x)* and *u(x)*. After that, just keep calculating the division of divisor and remainder. Until the final remainder is zero, the iteration stops, and the w(x) in previous loop is taken as MI, which is $\{80\}_{16} \equiv x^7+x+1$ in our example.

Extended Euclidean Algorithm for Polynomials			
Calculate	**Which satisfies**	**Calculate**	**Which satisfies**
$r_{-1}(x) = a(x)$		$v_{-1}(x) = 1; w_{-1}(x) = 0$	$a(x) = a(x)v_{-1}(x) + bw_{-1}(x)$
$r_0(x) = b(x)$		$v_0(x) = 0; w_0(x) = 1$	$b(x) = a(x)v_0(x) + b(x)w_0(x)$
$r_1(x) = a(x) \bmod b(x)$ $q_1(x) =$ quotient of $a(x)/b(x)$	$a(x) = q_1(x)b(x) + r_1(x)$	$v_1(x) = v_{-1}(x) - q_1(x)v_0(x) = 1$ $w_1(x) = w_{-1}(x) - q_1(x)w_0(x) = -q_1(x)$	$r_1(x) = a(x)v_1(x) + b(x)w_1(x)$
$r_2(x) = b(x) \bmod r_1(x)$ $q_2(x) =$ quotient of $b(x)/r_1(x)$	$b(x) = q_2(x)r_1(x) + r_2(x)$	$v_2(x) = v_0(x) - q_2(x)v_1(x)$ $w_2(x) = w_0(x) - q_2(x)w_1(x)$	$r_2(x) = a(x)v_2(x) + b(x)w_2(x)$
$r_3(x) = r_1(x) \bmod r_2(x)$ $q_3(x) =$ quotient of $r_1(x)/r_2(x)$	$r_1(x) = q_3(x)r_2(x) + r_3(x)$	$v_3(x) = v_1(x) - q_3(x)v_2(x)$ $w_3(x) = w_1(x) - q_3(x)w_2(x)$	$r_3(x) = a(x)v_3(x) + b(x)w_3(x)$
• • •	• • •	• • •	• • •
$r_n(x) = r_{n-2}(x) \bmod r_{n-1}(x)$ $q_n(x) =$ quotient of $r_{n-2}(x)/r_{n-3}(x)$	$r_{n-2}(x) = q_n(x)r_{n-1}(x) + r_n(x)$	$v_n(x) = v_{n-2}(x) - q_n(x)v_{n-1}(x)$ $w_n(x) = w_{n-2}(x) - q_n(x)w_{n-1}(x)$	$r_n(x) = a(x)v_n(x) + b(x)w_n(x)$
$r_{n+1}(x) = r_{n-1}(x) \bmod r_n(x) = 0$ $q_{n+1}(x) =$ quotient of $r_{n-1}(x)/r_{n-2}(x)$	$r_{n-1}(x) = q_{n+1}(x)r_n(x) + 0$		$d(x) = \gcd(a(x), b(x)) = r_n(x)$ $v(x) = v_n(x); w(x) = w_n(x)$

Figure 14. extended Euclidean algorithm *[5]*

3.2 Affine Transformation

Affine transformation includes a 8x8 matrix *M* and a vector *V* with 8 elements. *Figure 15* is the affine transformation to generate AES s-box. Here, $b = (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$ is the 8 bit input, b_0 is LSB, b_7 is MSB and $s = (s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7)$ is the 8 bit output. According to different order of bits that written in C-code, the affine matrix *M* and vector *V* are also needed to be reversed. The differences of two kinds of input with different *M* and *V* are shown in *Figure 15. (a) & (b).*

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

(a)

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

(b)

Figure 15. Affine Transformation of AES s-box

4. AES s-box vs. SM4 s-box

For both AES (S_{AES}) and SM4 (S_{SM4}) sbox, the algebraic expression can be simplified as :

$$S_{AES}(x) = I(x) \cdot M + V$$
$$S_{SM4}(x) = I(x \cdot A + C) \cdot A + C$$

where I is multiplicative inverse over GF(2^8) and x is the 8-bits input. For SM4, affine transformation consists of matrix A and vector C but with different values from AES. *Figure 16* shows the value of A and C :

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

Figure 16. Affine Transformation of SM4 s-box

Also for MI, according to [8], the irreducible polynomial is :

$$x^8+x^7+x^6+x^5+x^4+x^2+1 \equiv \{111110101\}_2 \equiv \{1f5\}_{16}$$

which is different from AES. Although this does not show in the simplified expression, in our code, the function for MI, named inverse, is marked clearly :

```
/* a is the irreducible polynomial, b is the input
 * SM4 : a = 0x1f5, AES : a = 0x11b             */
int inverse(int a,int b){
```

With these information, we can build the same functions with only different values for AES and SM4 s-box (Appendix B)

5. Masked s-box

One countermeasure against side-channel attacks is masking the data during calculation, through adding or multiplying by random values. Here we follow [9] to use random masking for SM4 s-box :

$$S_{SM4}(x) = I(x \cdot A + C) \cdot A + V$$
$$S_{masked\_SM4}(x) = I((x+m) \cdot A + C) \cdot A + V$$
$$= S_{SM4}(x) + m \cdot A \cdot A$$

where m is the 8-bit random mask. In [9], they derive a result that after making S_{masked_SM4} is equal to $S_{SM4}(x) + m \cdot A \cdot A$. We follow this result to build the code, however, only few elements in s-box meet this result. We suspected that why we cannot succeed is because the derivation in [9] is through composite field but we calculate in Rijndael field.
(The trial code : https://github.com/YWsGithub/s-box/tree/master/SM4_masked_s-box)

6. Generate s-box through Composite Field

Many designs have been studied to optimize s-box and using composite field (also called subfield) is one of the popular means. In composite Galois field $GF((2^4)^2)$, the 8-bits input is divided into two 4-bits inputs : $X_h, X_l \in GF(2^4)$. For example :

$$\{83\}_{16} \equiv \{10000011\}_2 \Rightarrow X_h = \{1000\}_4, X_l = \{0011\}_4$$

and in $GF(((2^2)^2)^2)$, an 8-bits input is divided into four 2-bits inputs in the similar way. In many researches, it has been demonstrated that generating s-box through composite field helps reduce the number of gates in the s-box thus decrease the area and cost of the chip.

Here, the first reference [6] we employ is through $GF((2^4)^2)$. *Figure 17 shows* the entire s-box computing process for AES (the upper one with output *Z*) and SM4 (second one with output *S*). For AES, the input first goes to mapping function δ, then does the MI over $GF((2^4)^2)$. After that, the data does the affine transformation with matrix M combined with inverse mapping function δ^{-1}. For SM4, the input do the affine transformation first, then

goes to mapping function *T and* MI over GF((2⁴)²). After that, the data does the affine transformation again but with the matrix *A* combined with inverse mapping function T^{-1}.

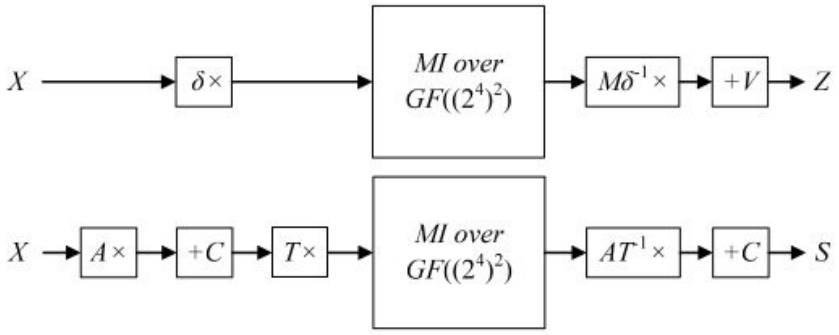

Figure 17. S-box computing process [6]

The different mapping functions used by [6] for AES (*δ*) and SM4 (*T*) are shown in *Figure 18*.

$$\delta = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \delta^{-1} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, T^{-1} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Figure 18. Mapping matrix for AES (δ) and SM4 (T) [6]

Figure 19 demonstrates the multiplicative inversion over GF((2⁴)²). At the beginning, the input is divided into two 4-bit elements : A_h and A_l. In our coding, we separate the

process to seven steps. The 1st one is to XOR A_h and A_l. In 2nd step, A_l multiplies with the result of 1st step. The 3rd step, A_h does $()^2 \times \nu_l$, which means to multiply by square over GF(2^4) and ν_l is the coefficient of the GF(2^2) irreducible polynomial, ν_l = $\{0010\}_2$. At 4th step, XOR the results of 2nd and 3rd step, then do the multiplicative inverse over GF(2^4) in 5th step. For the 6th and 7th, take A_h and A_l to multiply the result of 5th respectively. Finally, merge the two 4-bits result of 6th and 7th as MSB and LSB to be a 8-bits output.

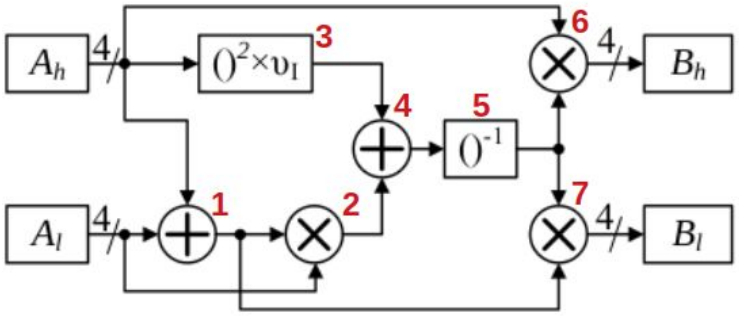

Figure 19. MI over GF((2^4)^2) [6]

Unfortunately, even by following this reference, we didn't succeed to generate the right s-box for AES and SM4. We also look into an citation [11] of this reference but all the calculating details are all the same. Therefore, by now there is no suspected cause.
(The trial code : https://github.com/YWsGithub/s-box/tree/master/Sbox_CFA_DPR)

From our second reference [10], we find out the complete C code for building s-box through GF((((2^2)^2)^2). But when we apply the same inverse function and mapping to SM4, we still cannot get the correct s-box. We suspected the mapping function because the matrix is derived from irreversible polynomials and it should be different for SM4 and AES. Besides, it is strange that the mapping matrix of AES (*Figure 20*) is different from the one in our first reference.

$$\delta = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}, \delta^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Figure 20. Mapping matrix for AES (δ) [10]

(The trial code : https://github.com/YWsGithub/s-box/tree/master/Sbox_CFA_Canright)

Conclusion

In this project, we briefly introduce the encryption process of AES and SM4. Then we focus on how to build s-box. At first, we introduce s-box algebraic structure and base on this to implement AES and SM4 s-box. After that, we compare the difference of these two s-boxes in order to realize SM4 s-box function of AES s-box. Unfortunately, before the end of this internship, we cannot find any reliable reference to build this. On the other hand, we follow some references to build masked SM4 s-box and try to realize the s-box through composite field. For masked SM4 s-box, we don't get the expected result. As for the s-box calculation through composite field, we have a complete code for AES but there's no correct result for SM4. Though the result of project 2 does not meet our expectation, we now have a deeper understanding about s-box implementation and optimization.

Future Work

Since Tiempo has its own masked AES implementation on hardware, for the follow-up, we can try to realize this mask in software and then make the SM4 version with the known difference written in this report. Moreover, there are many researches that already published some ways to mask s-box, they could also be applied to our s-box in C.

For the composite field part, if we could have a deeper understanding of the theory, we can try to confirm our first reference or even modify the AES s-box code from our second reference. After achieving to realize the masked s-box in composite field, the s-box can be implemented into the complete SM4 encryption code which we provided in this report.

Bibliography

1. *TESIC_RMA_Test_Specification,* Tiempo S.A.S
2. *GM/T0002-2012 SM4 block cipher algorithm*, State Cryptography Administration of the People's Republic of China
3. *Differential Power Analysis Attack on SMS4 Block Cipher*, 2008 4th IEEE International Conference on Circuits and Systems for Communications
4. *ARM-USB-OCD-H, ARM-USB-OCD OLIMEX OPENOCD ARM JTAG DEBUGGERS user's manual*
5. *Cryptography and Network security Principles and Practice 5th edition*, William Stallings
6. *DPR Based AES/SM4 Encryption Highly Efficient Implementation*, Qiangjia Bi et. al
7. *Improvements of SM4 Algorithm and Application in Ethernet Encryption System Based on FPGA*, Hai Cheng et. al
8. *Analysis of the SMS4 block cipher*, Lei Hu et. al
9. *Design of a Masked S-box for SM4 Based on Composite Field*, Hao Liang et. al
10. *A Very Compact Rijndael S-box*, D. Canright
11. *A new compact hardware architecture of S-Box for block ciphers AES and SM4*, Yaoping Liu