Research on Modern Higher Education 4, 01001 (2017)
DOI: 10.24104/rmhe/2017.04.01001

An efficient hardware implementation of SM4

Huafeng Chen* & Yanbing Jiang
Zhejiang University of Media and Communications, Hangzhou, Zhejiang, China

ABSTRACT: SM4 is widely used in WLAN WAPI resource restricted devices. This paper proposes an 8-bit iteration structure of SM4 at ultra-low cost, which manages key expansion and encryption working alternately with only one S-Box. Furthermore to minimize the area, on-the-fly key expansion mechanism is utilized to reserve the memory for round keys and constants keys are generated by equation dynamically instead of reading from large look-up table. The area results of its ASIC hardware implementation show that the gate count of our design is only 2.51 K, gaining almost 18.0% area reduction compared to the latest works.

Keywords: SM4; on-the-fly key expansion; constant key; iteration structure

1 INTRODUCTION

The SM4 algorithm was issued in January, 2006 by the Office of Security Commercial Code Administration (OSCCA) and announced to be the symmetric cryptographic algorithm in trusted computing platform specification in China in December, 2007[1-2]. Nowadays it is widely used in Chinese National Standard for Wireless LAN WAPI.

SM4 is a block cipher that generates a 128-bit output from a 128-bit input and a 128-bit key after 32 non-linear rounds. It has a concise structure and each round has a few XOR operations, a non-linear substitution and a linear substitution. Encryption and decryption have the same structure except that the order of round keys for decryption is reversed. Key expansion has a similar structure to encryption and uses the same non-linear substitution. The conciseness and similarity make SM4 easy to implement.

There are many implementations of SM4 algorithm now, optimized for various objects. One direction is to improve the throughput using pipeline structures. A fast implementation on smart cards unrolls the iteration loops partly from 32 rounds to 8 rounds [3]. Most works implement key expansion and encryption separately to improve the parallelism and unroll 32 rounds fully [4-7]. They can process 32 data blocks simultaneously and return a result every cycle. However, the high throughput is achieved by duplicating S-Box and other logics, resulting in a heavy consumption of hardware resources.

The other direction is to reduce the hardware cost using iteration structures [5-8]. Only one data block is in process at any time and the output is returned after 32 clock cycles. Using small data width, reducing the number of S-Box and utilizing lightweight key expansion mechanism were proposed to make AES compact [9-11]. AES is a typical block cipher and the method for it can be applied to SM4. Key expansion and encryption share the iteration structure, reducing the number of S-Box by half [7]. An 8-bit iteration structure was proposed to make SM4 ultra-compact, where only one S-Box is utilized by resource multiplex technique and the constant keys are rescheduled [8]. We call the ultra-compact SM4 design "UCSM4" for short. There are two key expansion mechanisms: one is to pre-compute and store all round keys, and the other is to provide round keys on-the-fly, called on-the-fly key expansion [9]. The former one is applied in most previous works including UCSM4, due to its high performance and low energy consumption when input keys remain the same. The latter one has to perform key expansion for every data block, but it does not need to store the round keys, saving hardware resource a lot. In embedded systems, low cost consideration is extremely important for designs on resource-restricted devices. However, previous works reduce the hardware consumption at a limited level, since low cost is not their primary consideration. Our goal in this work is to minimize the area.

*Corresponding author: walfen@126.com

This paper proposes a new hardware implementation of SM4 at ultra-low cost (ULSM4). Like UCSM4, the iteration structure of ULSM4 takes 8-bit data width as process unit and supports the computation of key expansion and encryption. The main contribution is that we first utilized on-the-fly key expansion and generated constant keys dynamically by equation on 8-bit iteration structure of SM4, which minimizes the area. We compared ULSM4 with the latest work UCSM4 on ASIC platform and the logic synthesis results show that ULSM4 occupies only 2.51 K gates at SMIC18 technology, 18.0% less than UCSM4. So the methods to minimize the area are proved to be effective and ULSM4 is more suitable for resource restricted devices.

2 SM4 ALGORITHM

2.1 *Terminology*

The standard SM4 algorithm is described in 32 bits. In following parts, the variable in upper case is a 32-bit vector and that in lower case is an 8-bit vector. Some terminologies are defined in Table 1.

Table1. Terminology of SM4

Symbol	Description
$S()$	Substitution box with 8-bit data width
$\oplus$	Bitwise XOR
$<<< i$	Circular left of a 32-bit vector with i bits shifted left

2.2 *Functions*

τ is the non-linear substitution which consists of 4 S-Boxes in parallel. The equation is

$$B = \tau(A) = (S(a_0), S(a_1), S(a_2), S(a_3)) \quad (1)$$

where $A=(a_0,a_1,a_2,a_3)$ is the 32-bit input; and B is the 32-bit output.

L and L' are the linear substitutions which take the output of τ as input. L is for encryption and L' is for key expansion. The equations are

$$\begin{aligned} C = L(B) = B \oplus (B <<< 2) \oplus (B <<< 10) \\ \oplus (B <<< 18) \oplus (B <<< 24) \end{aligned} \quad (2)$$

$$C = L'(B) = B \oplus (B <<< 13) \oplus (B <<< 23) \quad (3)$$

where B is the 32-bit input; and C is the 32-bit output.

F and F' are the round functions which update 32 bits at one time. F is for encryption and F' is for key expansion. The round functions consist of a non-linear substitution and a linear substitution. The equation of F is

$$F(X_0, X_1, X_2, X_3, RK) = X_0 \oplus (L \bullet \tau)(X_1 \oplus X_2 \oplus X_3 \oplus RK) \quad (4)$$

the equation of F' is

$$F'(K_0, K_1, K_2, K_3, CK) = K_0 \oplus (L' \bullet \tau)(K_1 \oplus K_2 \oplus K_3 \oplus CK) \quad (5)$$

where (X_0, X_1, X_2, X_3) is the 128-bit input for encryption; RK is the 32-bit round key; (K_0, K_1, K_2, K_3) is the 128-bit input for key expansion; and CK is the 32-bit constant key.

2.3 *Key expansion*

Key expansion generates the round keys used in encryption. MK is the 128-bit input key, which is initialized by the 128-bit system parameter FK. The equation of initialization is

$$\begin{aligned} K_0 = MK_0 \oplus FK_0, K_0 = MK_1 \oplus FK_1 \\ K_2 = MK_2 \oplus FK_2, K_3 = MK_3 \oplus FK_3 \end{aligned} \quad (6)$$

where MK = (MK_0, MK_1, MK_2, MK_3); and FK= (FK_0, FK_1, FK_2, FK_3).

Key expansion takes 32 iterations of round function F' to generate round keys, and each round needs a 32-bit constant key. The process of key expansion is

$$RK_i = K_{i+4} = F'(K_0, K_1, K_2, K_3, CK_i)(i = 0, ..., 31) \quad (7)$$

The constant key can be calculated by equation and the equation is

$$ck_{i,j} = ((4i + j) \times 7) (\mathrm{mod}\, 256) \quad (8)$$

where $CK_i = ck_{i,1}, ck_{i,2}, ck_{i,3}$; i =0,…,31; and j = 0,1,2,3.

2.4 *Encryption*

Encryption and decryption have the same structure which takes 32 rounds of non-linear substitution. The difference between them is the order of round keys, that is, (rk_0, rk_1,..., rk_{31}) is for encryption and (rk_{31}, rk_{30}, ..., rk_0) is for decryption. Encryption proceeds as below:

$$\begin{aligned} X_{i+4} &= F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, RK_i) \\ &\quad (i = 0, ..., 31) \\ (Y_0, Y_1, Y_2, Y_3) &= R(X_{32}, X_{33}, X_{34}, X_{35}) \\ &= (X_{35}, X_{34}, X_{33}, X_{32}) \end{aligned} \quad (9)$$

where (X_0, X_1, X_2, X_3) is the 128-bit input plaintext; (Y_0, Y_1, Y_2, Y_3) is the 128-bit output cipher text; and R is the reverse substitution.

3 ULTRA-LOW COST IMPLEMENTATION OF SM4

Previous works of SM4 for low-cost have shown that a small width for process unit and resource reutilization technique can make SM4 implementations compact. So our design ULSM4 has an 8-bit process unit

and only one S-Box and the iteration structure supports the computation of key expansion and encryption. To make the hardware cost lower, ULSM4 employs on-the-fly key expansion to reserve the large memory for generated round keys and generates 8-bit constant keys by equation instead of LUT. So ULSM4 is implemented at ultra-low cost.

3.1 *Overview*

The proposed design supports encryption and decryption in Electric Code Block (ECB) mode. The block diagram is shown in Figure 1. X3~X0 and K3~K0 are the input interface for data and key separately, and Y3~Y0 is the output interface for data. The bus width of both input and output interface is 8 bits. Signal start enables the computation and edflag determines to perform encryption or decryption. When signal done is high, the results are returned by the output interface. The ULSM4 consists of a control module and a data path module. The control module schedules the data path module to perform key expansion or encryption and maintains the update of registers by signals ksel and dsel. The data path module updates 8 bits at a time and needs four cycles to perform one round of functions *F* and *F'*. Since ULSM4 employs on-the-fly key expansion and the iteration structure is shared by key expansion and encryption, the data path module has two 128-bit register sets to store the intermediate data, *XR* = {*XR*0, *XR*1, *XR*2, *XR*3} for encryption and *KR* = {*KR*0, *KR*1, *KR*2, *KR*3} for key expansion.

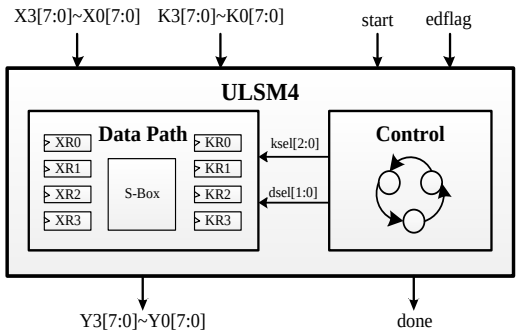

Figure 1. Block diagram of ULSM4

3.2 *On-the-fly key expansion*

ULSM4 applies on-the-fly key expansion mechanism, which produces the round keys on-the-fly. Since key expansion can't be avoided even when the input keys are unchanged, we combine key expansion and encryption/decryption into one process. For encryption, the process has 32 iterations including one round for key expansion and one round for encryption. To get the round key ready for the encryption round in the same iteration, the round for key expansion is operated first. KR and XR registers store the immediate results and are updated when the corresponding round is completed. The process for encryption is shown in Algorithm 1.

Algorithm 1: On-the-fly Key Expansion for Encryption

Input: Plain text(X_0, X_1, X_2, X_3) and key(K_0, K_1, K_2, K_3)
Output: Cipher text(Y_0, Y_1, Y_2, Y_3)
(KR0, KR1, KR2, KR3) = (K_0, K_1, K_2, K_3);
(XR0,XR1,XR2,XR3) = (X_0, X_1, X_2, X_3);
for $i = 0$ *to* 31 **do**
  $RK_i = F'$(KR0, KR1, KR2, KR3, CK_i);
  (KR0, KR1, KR2, KR3) = (KR1, KR2, KR3, RK_i);
  $RX_i = F$(XR0, XR1, XR2, XR3, KR3);
  (XR0, XR1, XR2, XR3) = (XR1, XR2, XR3, RX_i);
(Y_0, Y_1, Y_2, Y_3) = (XR3, XR2, XR1, XR0); `// Reverse`
return (Y_0, Y_1, Y_2, Y_3);

For decryption, the order of round keys is reversed. On-the-fly key expansion for decryption is performed in three steps. The first step is to get the final round key by performing key expansion alone. The second step is to reverse the KR register to make the key order right. The final step is to combine key expansion and decryption into one process. The process also has 32 iterations and the round count is decreased from 31 to 0. Unlike encryption, the round for decryption is performed first and the round for key expansion generates the round key used in the next iteration. At the last four iterations, round keys are ready in KR registers and left shift operation is enough to return the round keys, therefore the round for key expansion is not performed. The process for decryption is shown in Algorithm 2.

Algorithm 2: On-the-fly Key Expansion for Decryption

Input: Cipher text(Y_0, Y_1, Y_2, Y_3) and key(K_0, K_1, K_2, K_3)
Output: Plain text(X_0, X_1, X_2, X_3)
(KR0, KR1, KR2, KR3) = (K_0, K_1, K_2, K_3);
(XR0,XR1,XR2,XR3) = (Y_0, Y_1, Y_2, Y_3);
for $i = 0$ *to* 31 **do**
  $RK_i = F'$(KR0, KR1, KR2, KR3, CK_i);
  (KR0, KR1, KR2, KR3) = (KR1, KR2, KR3, RK_i);
(KR0, KR1, KR2, KR3) = (KR3, KR2, KR1, KR0); `// Reverse`
for $i = 31$ *to* 0 **do**
  $RX_i = F$(XR0, XR1, XR2, XR3, KR0);
  (XR0, XR1, XR2, XR3) = (XR1, XR2, XR3, RX_i);
  if $i \geq 4$ **then**
    $RK_{i-4} = F'$(KR0, KR1, KR2, KR3, CK_i);
    (KR0, KR1, KR2, KR3) = (KR1, KR2, KR3, RK_{i-4});
  else
    (KR0, KR1, KR2, KR3) = (KR1, KR2, KR3, 0);
(X_0, X_1, X_2, X_3) = (XR3, XR2, XR1, XR0);
return (X_0, X_1, X_2, X_3);

So on-the-fly key expansion mechanism only uses 128-bit registers to generate the round keys, rather than storing all the round keys in a 32x32 bit memory, saving hardware resource a lot.

3.3 *Iteration Structure*

The iteration architecture of ULSM4 is shown in Figure 2. The architecture is 8-bit except the linear substitution, which contains 32-bit circular left shift and can't be divided into byte operation. XR and KR reg-

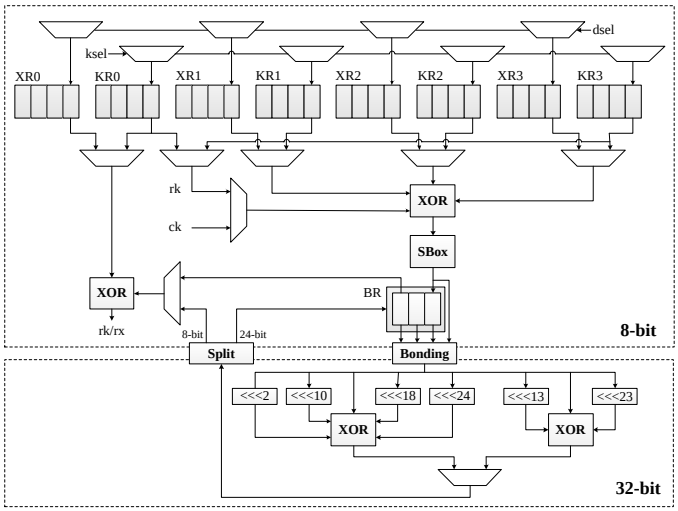

Figure 2. Iteration structure of ULSM4

isters are implemented as shift registers, shifting 8 bits to the left in one clock cycle. Only the rightmost byte updates through 8-bit multiplexer controlled by dsel or ksel and is selected to perform 8-bit XOR operations. Register BR acts as left shift register to store the output of S-Box in the first three cycles of a round and is reused to store the lower 24 bits of linear substitution's output at the last cycle. The output interface is 8-bit, and the results need four cycles to return. For encryption, the round key *rk* comes from register KR3, while for decryption, it comes from register KR0.

When the iteration structure is scheduled to perform key expansion, constant keys are needed. The constant keys are implemented by LUT in previous works and the size of LUT is 32x32 bits, too costly. We choose to generate the constant keys dynamically by equation, and only one equation is implemented due to the 8-bit process unit. The equation in hardware implementation is

$$\begin{aligned} ck_{i,j} &= (4i+j)\times 7 \bmod 256 \\ &= ((4i+j) << 3 - (4i+j)) \bmod 256 \end{aligned} \tag{10}$$

where $i = 0,\ldots,31$ is the round count; and $j = 0,1,2,3$ is the cycle count in a round.

The addition and left shift in the equation can be avoided by simply wiring, so only one subtraction is needed.

4 EXPERIMENTAL RESULTS

We implemented ULSM4 on ASIC platform and carry out the logic synthesis of typical case at SMIC18 technology by using Synopsys Design Compiler. The frequency in synthesis script is set to 185 MHz since the latest work UCSM4 [8] is chosen as the baseline. UCSM4 has an 8-bit iteration structure with only one S-Box shared by key expansion and encryption. The round keys are pre-computed and stored in the memory, and the constant keys are implemented by LUT. Based on UCSM4, a design applying on-the-fly key expansion is also implemented and called OTFSM4. OTFSM4 implements constant keys by LUT, but it does not reschedule them, so the main difference between UCSM4 and OTFSM4 is the key expansion mechanism. The synthesis results are shown in Table 2.

Table 2. Logic synthesis results @ SMIC18 and 185 MHz

Item	Area/um^2			Gate Count/*K*
	Combinational	Non-combinational	Total	
UCSM4[8]	19772	10797	30569	3.06
OTFSM4	11794	13513	25308	2.53
ULSM4	11557	13496	25053	2.51

For OTFSM4, the total area is 25308 um^2, where the area of combinational logics is 11794 um^2 and that of non-combinational logics is 13513 um^2. Key expansion and encryption are serialized in UCSM4 while they work in turn in OTFSM4. So the number of registers for immediate results is 128 bits for the baseline and 256 bits for OTFSM4. This is the reason that the area of non-combinational logics in OTFSM4 is larger than UCSM4. But on-the-fly key expansion saves the memory for round keys, making the area of combinational logics in OTFSM4 smaller. The hardware consumption of OTFSM4 falls to 82.8% of UCSM4.

The only difference between OTFSM4 and ULSM4 is that ULSM4 generates the constant keys by equation instead. The generation of constant key does not use registers for both LUT and equation, so the area of ULSM4's non-combinational logic is almost unchanged compared to OTFSM4. But the area of ULSM4's combinational logic drops from 11794 um^2

Table 3. Comparison of throughput

Item	Mode	Key	Cycles	Frequency/MHz	Throughput/Mbps
UCSM4[8]	Encryption	Changed	256	185	92.5
	Decryption	Changed	256	185	92.5
	Encryption	Unchanged	128	185	185
	Decryption	Unchanged	128	185	185
ULSM4	Encryption	Not care	256	435	217.5
	Decryption	Not care	372	435	149.7

to 11557 um^2. The equation to generate an 8-bit constant key only needs 8-bit subtraction, while LUT has a size of 32x32 bits and needs many multiplexers to select the 8-bit constant key. So generating 8-bit constant key by equation can save hardware resource a lot by. The total area of ULSM4 is 25053 um^2, reducing area by 1.0% compared to OTFSM4 and by 18.0% compared to UCSM4. The results show that our design ULSM4 has a lower consumption of hardware resource.

We also compare ULSM4's throughput with UCSM4 and the results are shown in Table 3. The throughput is determined by maximum frequency and cycles to finish SM4 algorithm. The memory to store all round keys is on the critical path, so the maximum frequency of UCSM4 is only 185 MHz. UCSM4 needs 256 cycles to finish an encryption or decryption when input keys are changed and 128 cycles when input keys remain the same. So the highest throughput of UCSM4 is 185MHz. However, ULSM4 has a maximum frequency of 435 MHz and does not care about the state of input keys. It needs 256 cycles to finish an encryption and 372 cycles to finish a decryption. The throughput of encryption is 217.5 Mbps and that of decryption is 149.7 Mbps. So ULSM4 has a higher maximum throughput than UCSM4.

5 CONCLUSION

In this paper, we proposed ULSM4, an iteration structure of SM4 aiming at low cost. To take full advantage of hardware resource and reduce the logic complexity, ULSM4 takes 8-bit data width as process unit and has only one S-Box shared by key expansion and encryption. To minimize the area, on-the-fly key expansion is applied to reserve the memory for generated round keys, and 8-bit constant keys are generated dynamically by equation. The synthesis result shows that ULSM4 reduces area by 18.0% compared to the latest work UCSM4 (by 17.2% for on-the-fly key expansion and by 0.8% for generating constant keys by equation). So ULSM4 is more efficient in resource utilization and more suitable to provide data protection for resource restricted applications.

REFERENCES

[1] GM/T 0002-2012. The SMS4 block cipher [S]. China: OSCCA, 2006.

[2] GM/T 0011-2012. Functionality and interface specification of cryptographic support platform for trusted computing [S]. China: OSCCA, 2007.

[3] Zhang D W, Ding W R, Ding D. 2008. Fast implementation of SMS4 cryptographic algorithms on smart card. *International Conference on Intelligent Information Hiding and Multimedia Signal Processing. Harbin, China: IEEE*, pp: 287-290.

[4] Zhao M, Shou G C, Hu Y H, *et al*. 2011. High-speed architecture design and implementation for SMS4-GCM. *Third International Conference on Communications and Mobile Computing. Qingdao, China: IEEE*, pp: 15-18.

[5] Jin Y E, Shen H B, You R Q. 2006. Implementation of SMS4 block cipher on FPGA. *First International Conference on Communications and Networking in China. Beijing, China: IEEE*, pp: 1-4.

[6] Gao X W, Lu E H, Xian L Q, et al. 2008. FPGA implementation of the SMS4 block cipher in Chinese WAPI standard. *International Conference on Embedded Software and Systems Symposia. Sichuan, China: IEEE*, pp: 104-106.

[7] Wang H S, Li S G. 2011. High performance FPGA implementation for SMS4. *Communications in Computer and Information Science*, 163: 469-475.

[8] Shang M, Zhang Q L, Liu Z B, *et al*. 2014. An ultra-compact hardware implementation of SMS4. *IIAI Third International Conference on Advanced Applied Informatics. Kitakyushu: IEEE*, pp: 86-90.

[9] Haghighizadeh F, Attarzadeh H, Sharifkhani M. 2010. A compact 8-bit AES crypto-processor. *Second International Conference on Computer and Network Technology. Bangkok: IEEE*, pp: 71-75.

[10] Benhadjyoussef N, Wajih E, Machhout M, *et al*. 2012. A compact 32-bit AES design for embedded system. *7th International Conference on Design & Technology of Integrated Systems in Nanoscale Era. Gammarth: IEEE*, pp: 1-4.

[11] Tay J J, Wong M M, Hijazin I. 2014. Compact and low power AES block cipher using lightweight key expansion mechanism and optimal numbers of S-boxes. *Internal Symposium on Intelligent Signal Processing and Communication Systems. Kuching: IEEE*, pp: 108-114.