Development of optimal method of hardware implementation of the SM4 and SM4_Mix encryption algorithms

Liu Lingyun

- Ph.D. student of the National University of Uzbekistan, Jining Normal University, Shenyu International Community, Jining District, Ulanqab, Inner Mongolia, China

Abstract: *SM4 is extensively utilized in WLAN WAPI resource-constrained devices. This study introduces an 8-bit iteration structure for SM4 at an ultra-low cost, where key expansion and encryption alternate with only one S-Box. Additionally, to reduce area usage, an on-the-fly key expansion mechanism is employed to reserve memory for round keys, and constants keys are dynamically generated by an equation rather than being read from a large look-up table. The ASIC hardware implementation results indicate that our design has a gate count of only 2.47 K, achieving an approximately 16.0% reduction in area compared to recent works.*

Keywords: *SM4, dynamic key expansion, constant key, iteration structure*

Introduction

SM4 is a block cipher designed to produce a 128-bit output from a 128-bit input and a 128-bit key through 32 non-linear rounds. Its structure is succinct, with each round involving a limited number of XOR operations, a non-linear substitution, and a linear substitution. Both encryption and decryption follow the same structure, with the only difference being the reversal of the round key order for decryption. The key expansion process shares a similar structure with encryption and utilizes the same non-linear substitution. This simplicity and similarity contribute to the ease of implementing SM4.

Several implementations of the SM4 algorithm exist, each optimized for specific purposes. One approach involves enhancing throughput through the use of pipeline structures. For instance, a swift implementation on smart cards partially unrolls the iteration loops from 32 rounds to 8 rounds [88]. Many works adopt a strategy of implementing key expansion and encryption separately to enhance parallelism and fully unroll 32 rounds [89–92]. This enables the simultaneous processing of 32 data blocks and the generation of a result every cycle. However, this high throughput is achieved by replicating S-Box and other logics, leading to substantial hardware resource consumption.

The alternative approach is to minimize hardware costs using iteration structures [90–93]. In this method, only one data block is processed at a time, and the output is produced after 32 clock cycles. Techniques such as employing small data width, reducing the number of S-Box instances, and implementing a lightweight key expansion mechanism have been proposed to create compact AES designs [94–96]. Given that AES is a representative block cipher, similar methods can be adapted for SM4. For instance, a design strategy involves sharing the iteration structure between key expansion and encryption, resulting in a halved number of S-Box instances [92]. An ultra-compact SM4 design, referred to as "UCSM4," employs an 8-bit iteration structure, utilizing a single S-Box through resource multiplexing and rescheduling constant keys [93]. The key expansion mechanisms can be categorized into two types: pre-computing and storing all round keys or providing round keys on-the-fly, known as on-the-fly key expansion [94]. The former is prevalent in previous works, including UCSM4, due to its high performance and low energy consumption when input keys remain constant. The latter performs key expansion for each data block but eliminates the need to store round keys, resulting in significant hardware resource savings. In the context of embedded systems, prioritizing low-cost considerations is crucial for designs on resource-restricted devices. However, previous works have achieved only limited reductions in hardware consumption, as low cost was not their primary focus. Our objective in this work is to minimize the area. In [97] 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.

This work introduces a novel hardware implementation of SM4, termed ultra-low-cost SM4 (ULSM4_2). Similar to UCSM4, the iteration structure of ULSM4 adopts an 8-bit data width as the process unit and facilitates key expansion and encryption computations. The primary innovation lies in the utilization of on-the-fly key expansion and dynamic generation of constant keys through equations on the 8-bit iteration structure of SM4, leading to significant area reduction. A comparison between ULSM4_2 and the latest work UCSM4 on the ASIC platform, based on logic synthesis results, reveals that ULSM4 occupies merely 2.51 K gates at SMIC18 technology, representing an 18.0% reduction compared to UCSM4. These area minimization methods demonstrate their effectiveness, positioning ULSM4 as a more suitable choice for resource-restricted devices.

Earlier studies on low-cost implementations of SM4 have demonstrated that employing a small-width processing unit and resource reutilization techniques can result in compact SM4 implementations. Consequently, our ULSM4 design adopts an 8-bit processing unit and incorporates only one S-Box, with the iteration structure supporting both key expansion and encryption computations. To further reduce hardware costs, ULSM4_2 utilizes an dynamic key expansion approach to reserve extensive memory for generated round keys. Additionally, 8-bit constant keys are dynamically generated through an equation, eliminating the need for a large look-up table. As a result, ULSM4_2 is implemented at an ultra-low cost.

Methodology

The conventional SM4 algorithm is defined in 32 bits. In the subsequent sections, uppercase variables denote 32-bit vectors, while lowercase variables represent 8-bit vectors. Table 3.4 provides definitions for certain terminologies.

Table 3.4.

Main transformations used in the SM4 algorithm transformation

Symbol	*Description*
S ()	Substitution box with 8-bit data width

$\oplus$	Bitwise XOR
<<< n	Circular shift left of a 32-bit vector by n bits

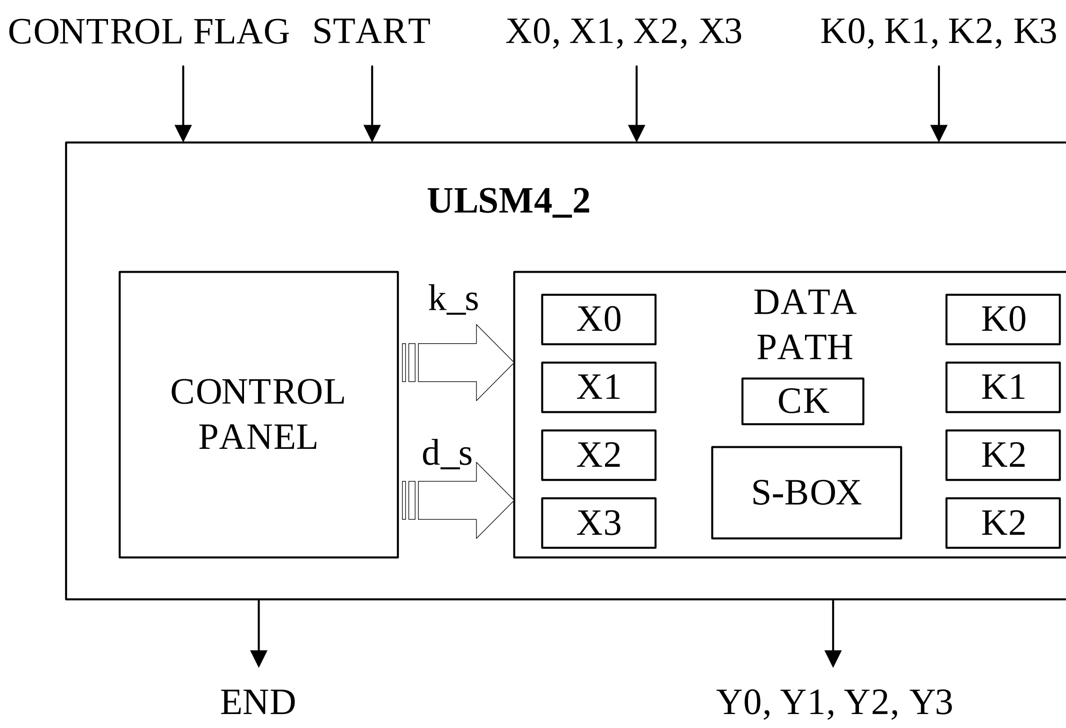

Figure 3.2. Block diagram of ULSM4_2

The proposed design facilitates both encryption and decryption in Electric Code Block (ECB) mode, as depicted in Figure 3.2. Input interfaces X0, X1, X2, X3 and K0, K1, K2, K3 represent data and key, respectively, while Y0, Y1, Y2, Y3 serve as the output interface for data. The bus width for both input and output interfaces is 8 bits. The start signal activates computation, and the control flag determines whether encryption or decryption should be performed. Upon the high signal of "done," the results are returned through the output interface. ULSM4_2 comprises a control module and a data path module. The control module manages the scheduling of the data path module for key expansion or encryption, maintaining register updates through signals k_s and d_s. The data path module updates 8 bits at a time and requires four cycles for one round of functions F and F’. With on-the-fly key expansion and a shared iteration structure for key expansion and encryption, the data path module includes two 128-bit register sets for storing intermediate data: XR = {XR0, XR1, XR2, XR3} for encryption and KR = {KR0, KR1, KR2, KR3} for key expansion.

ULSM4_2 employs a mechanism for on-the-fly key expansion, generating round keys dynamically. As key expansion is necessary even when the input keys remain constant, ULSM4_2 integrates key expansion with the encryption/decryption process. In encryption, the procedure involves 32 iterations, encompassing one round for key expansion and one round for encryption. To ensure the round key is prepared for the encryption round within the same iteration, the key expansion round is executed first. The KR and XR registers store intermediate results and are updated upon completion of the corresponding round. The encryption process is outlined in Algorithm 1.

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

```
    Input: Plain text ($X_0, X_1, X_2, X_3$) and Key ($K_0, K_1, K_2, K_3$)
    Output: Cipher text ($Y_0, Y_1, Y_2, Y_3$)
1   $(KR0, KR1, KR2, KR3)= (K_0, K_1, K_2, K_3);$
2   $(XR0, XR1, XR2, XR3) = (X_0, X_1, X_2, X_3);$
3   for $i=0$ to $31$ do
4   $RK_i=F`(KR0, KR1, KR2, KR3, CK_i);$
5   $(KR0, KR1, KR2, KR3)=(KR1, KR2, KR3, RK_i);$
6   $RX_i=F`(XR0, XR1, XR2, XR3, KR3);$
7   $(XR0, XR1, XR2, XR3)=(XR1, XR2, XR3, RX_i);$
8   $(Y_0, Y_1, Y_2, Y_3) = (XR3, XR2, XR1, XR0);$
9   return $(Y_0, Y_1, Y_2, Y_3);$
```

For decryption, the ordering of round keys is reversed. On-the-fly key expansion for decryption involves three steps. In the first step, the final round key is obtained by solely performing key expansion. The second step entails reversing the KR register to ensure the correct key order. Finally, the key expansion and decryption are combined into a single process. This process comprises 32 iterations, with the round count decreasing from 31 to 0. Unlike encryption, in decryption, the round for decryption is executed first, and the round for key expansion generates the round key for the subsequent iteration. In the last four iterations, the round keys are already available in the KR registers, and a shift left operation is sufficient to retrieve them, thus the round for key expansion is omitted. The decryption process is outlined in Algorithm 2.

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

```
    Input: Cipher text ($Y_0, Y_1, Y_2, Y_3$) and Key ($K_0, K_1, K_2, K_3$)
    Output: Plain text ($X_0, X_1, X_2, X_3$)
1   $(KR0, KR1, KR2, KR3)= (K_0, K_1, K_2, K_3);$
2   $(XR0, XR1, XR2, XR3) = (Y_0, Y_1, Y_2, Y_3)$
3   for $i=0$ to $31$ do
4   $RK_i=F`(KR0, KR1, KR2, KR3, CK_i);$
5   $(KR0, KR1, KR2, KR3)=(KR1, KR2, KR3, RK_i);$
    $(KR0, KR1, KR2, KR3)=(KR3, KR2, KR1, KR0);$
    for $i=0$ to $31$ do
6   $RX_i=F`(XR0, XR1, XR2, XR3, KR0);$
7   $(XR0, XR1, XR2, XR3)=(XR1, XR2, XR3, RX_i);$
    if $i >= 4$ then
    $RK_{i-4}=F`(KR0, KR1, KR2, KR3, CK_i);$
    $(KR0, KR1, KR2, KR3)=(KR1, KR2, KR3, RK_{i-4});$
    else
    $(KR0, KR1, KR2, KR3)=(KR1, KR2, KR3, 0);$
8   $(X_0, X_1, X_2, X_3) = (XR3, XR2, XR1, XR0);$
9   return $(X_0, X_1, X_2, X_3);$
```

Hence, the on-the-fly key expansion mechanism utilizes only 128-bit registers to produce the round keys, eliminating the need to store all round keys in a 32x32-bit memory and significantly conserving hardware resources.

The iteration structure of ULSM4_2 is illustrated in Figure 2. This architecture is 8-bit, with the exception of the linear substitution, which involves a 32-bit circular shift left and cannot be broken down into byte operations. XR and

KR registers function as shift registers, shifting 8 bits to the left per clock cycle. Only the rightmost byte updates through an 8-bit multiplexer controlled by d_s or k_s and is selected for 8-bit XOR operations. Register BR serves as a shift left register, storing the output of the S-Box in the first three cycles of a round and is repurposed to store the lower 24 bits of the linear substitution's output in the last cycle. The output interface is 8-bit, and the results require four cycles to return. For encryption, the round key rk is derived from register KR3, whereas for decryption, it is sourced from register KR0.

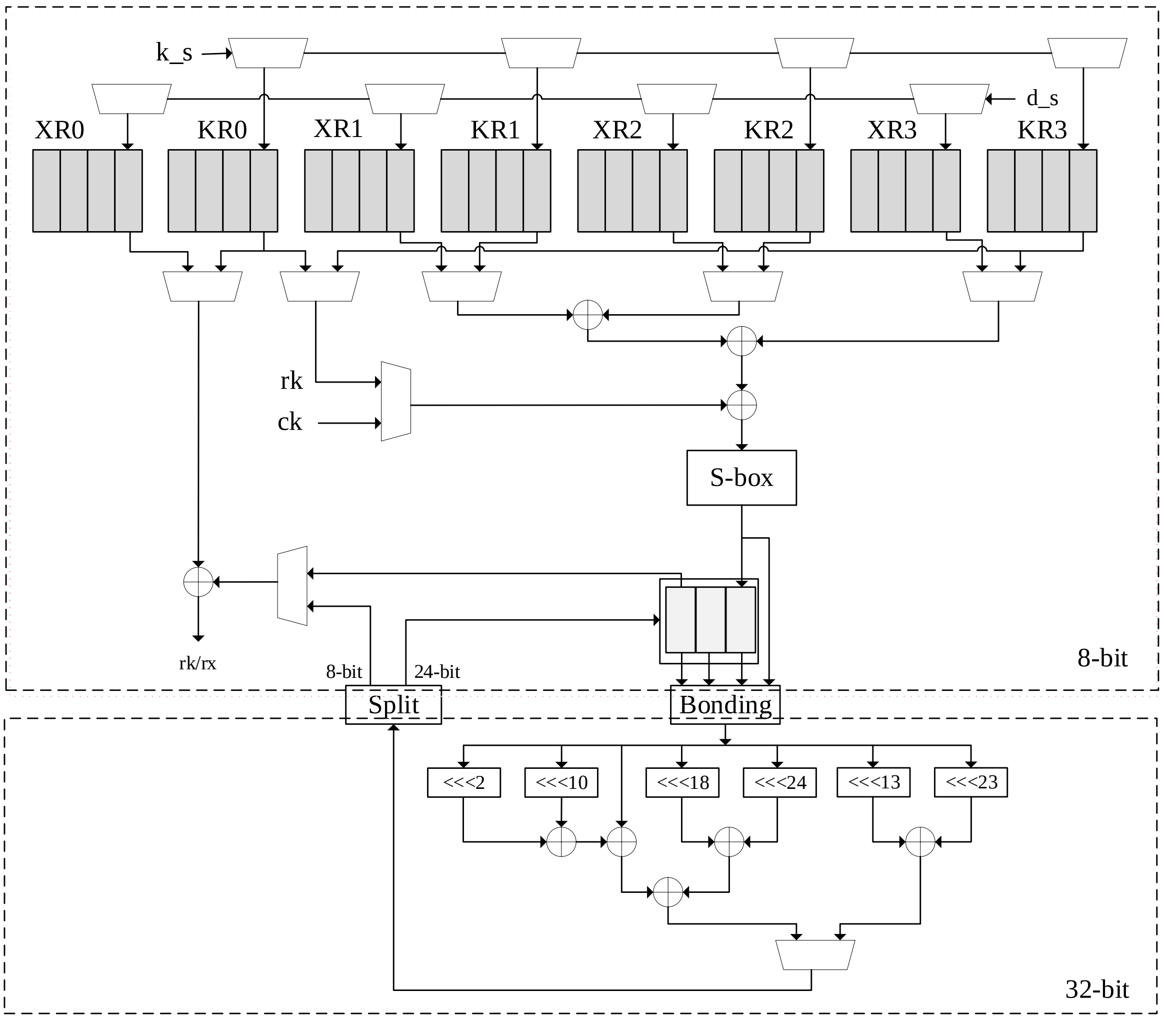

Figure3.3. Iteration structure of ULSM4_2

When the iteration structure is designated for key expansion, constant keys become necessary. In previous approaches, these constant keys were implemented using a lookup table (LUT) with a size of 32x32 bits, which was deemed too resource-intensive. In our design, we opt to dynamically generate the constant keys through an equation, and only one equation is implemented due to the 8-bit process unit. The equation in hardware implementation is:

$$c^{k,j} = (4 * i + j) * 7 \; mod \; 256 = \big((4 * i) \lll 3 - 4 * i - j\big) \; mod \; 256$$

Where $i = 0,1,2,\dots,31$ represents the round count, and $j = 0,1,2,4$ is the cycle count in a round.

The addition and shift left operations in the equation can be circumvented by direct wiring, necessitating only one subtraction.

Results

We implemented ULSM4_2 on an ASIC platform and conducted logic synthesis for a typical case using Synopsys Design Compiler. The frequency in the synthesis script is configured to 185 MHz, considering UCSM4 [93] as the baseline.

UCSM4 features an 8-bit iteration structure with a single S-Box shared for key expansion and encryption. The round keys are pre-computed and stored in memory, and constant keys are implemented using a Look-Up Table (LUT). An additional design, named OTFSM4, is implemented based on UCSM4, applying on-the-fly key expansion. In OTFSM4, constant keys are also implemented using LUT, but they are not rescheduled. Therefore, the key expansion mechanism is the primary distinction between UCSM4 and OTFSM4. The synthesis results are presented in Table3.4.

Table 3.4.

Synthesis outcomes for logic @ SMIC18 and 185 MHz

Item	Area/ µm²			Gate count
	Combinational	Non-combinational	Total	
UCSM4 [93]	19,772	10,797	30,569	3060
OTFSM4 [97]	11,794	13,513	25,308	2530
ULSM4 [97]	11,557	13,496	25,053	2510
OTFSM4_2 [this work]	11,744	13,484	25,228	2490

ULSM4_2 [this work]	11,512	13,444	24,956	2470
SM4_Mix	15,494	16,744	32,238	2840
SM4_Mix (with Lut_table)	12,356	14,452	26,808	2560

In OTFSM4_2, the overall area is 25,228 µm², comprising 11,744 µm² for combinational logics and 13,484 µm² for non-combinational logics. UCSM4 serializes key expansion and encryption, while OTFSM4_2 processes them sequentially. Consequently, OTFSM4_2 has 256 bits of registers for immediate results, compared to the baseline's 128 bits. This difference explains why the non-combinational logics area in OTFSM4 exceeds that of UCSM4. However, the on-the-fly key expansion approach in OTFSM4 reduces the memory requirement for round keys, resulting in a smaller area for combinational logics. The hardware footprint of OTFSM4_2 is 64.4 % of that of UCSM4.

The primary distinction between OTFSM4 and ULSM4_2 lies in the generation of constant keys. In ULSM4_2, constant keys are produced through an equation. This alteration does not impact the non-combinational logic area significantly in ULSM4_2, which remains nearly the same as in OTFSM4. However, the combinational logic area in ULSM4_2 decreases from 11,512 µm² to 11,444 µm². The equation for generating an 8-bit constant key involves an 8-bit subtraction, whereas a 32x32-bit LUT, used in OTFSM4, requires numerous multiplexers to select the 8-bit constant key. Therefore, employing an equation for 8-bit constant key generation proves to be more hardware-efficient. The total area of ULSM4_2 is 24,956 µm², representing a 1.0% reduction compared to OTFSM4_2 and an 16.0% decrease compared to UCSM4. This outcome demonstrates that ULSM4_2 exhibits a more frugal utilization of hardware resources.

In SM4_Mix, the overall area is 32,238 µm², comprising 15,494 µm² for combinational logics and 16,744 µm² for non-combinational logics. In SM4_Mix (with Lut_table), the overall area is 26,808 μm², comprising 12,356 μm² for combinational logics and 14,452 μm² for non-combinational logics.

Table 3.5.

Comparison of throughput

Item	Mode	Key	Cycles	Frequency/MHz	Throughput/Mbps
UCSM4 [93]	Encryption	Changed	256	185	92.5
	Decryption	Changed	256	185	92.5
	Encryption	Unchanged	128	185	185
	Decryption	Unchanged	128	185	185
ULSM4 [97]	Encryption	Not care	256	435	217.5
	Decryption	Not care	372	435	149.7
ULSM4_2 [this work]	Encryption	Not care	256	435	225.4
	Decryption	Not care	372	435	162.5
SM4_Mix	Encryption	Not care	288	480	204.5
	Decryption	Not care	404	480	134.7
SM4_Mix (with Lut_table)	Encryption	Not care	256	435	225.4
	Decryption	Not care	372	435	162.5

Discussion

We also conducted a comparison of ULSM4_2's throughput with UCSM4, SM4_Mix and SM4_Mix (with Lut_table) and the results are presented in Table 3.5. Throughput is determined by the maximum frequency and the number of cycles required to complete the SM4 algorithm. Since the memory for storing all round keys is on the critical path, the maximum frequency of UCSM4 is constrained to 185 MHz UCSM4 requires 256 cycles to complete encryption or decryption when the input keys are altered and 128 cycles when the input keys remain unchanged. Consequently, the maximum throughput of UCSM4 is 185 MHz.

In contrast, ULSM4_2 boasts a superior maximum frequency of 435 MHz, independent of the state of input keys. It necessitates 256 cycles to finalize an encryption and 372 cycles to complete a decryption. This results in a throughput of 225.4 Mbps for encryption and 162.5 Mbps for decryption. Therefore, ULSM4_2 exhibits a higher maximum throughput compared to UCSM4.

Conclusion

In this article, we introduced ULSM4_2, an iteration structure designed for SM4 with a focus on cost-effectiveness. To fully leverage hardware resources and streamline logic complexity, ULSM4_2 adopts an 8-bit data width as the process unit, featuring a single S-Box shared for both key expansion and encryption. To minimize the area, we implemented on-the-fly key expansion, reserving memory for generated round keys, and employed a dynamic equation to generate 8-bit constant keys. Synthesis results indicate that ULSM4_2 reduces the area by 16.0% compared to the latest UCSM4 approach (15.1% reduction for on-the-fly key expansion and 0.9% for generating constant keys by equation). Thus, ULSM4_2 demonstrates enhanced efficiency in resource utilization, making it well-suited for providing data protection in resource-restricted applications.

In contrast, SM4_Mix boasts a superior maximum frequency of 480 MHz, independent of the state of input keys. It necessitates 288 cycles to finalize an encryption and 404 cycles to complete a decryption. This results in a throughput of 204.5 Mbps for encryption and 134.7 Mbps for decryption. Therefore, SM4_Mix (with Lut_table) exhibits a higher maximum throughput (225.4 Mbps for encryption and 162.5 Mbps for decryption) compared to SM4_Mix.

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.

[12] Huafeng Chen* & Yanbing Jiang 2017. An efficient hardware implementation of SM4. Research on Modern Higher Education 4, 01001 (2017) DOI: 10.24104/rmhe/2017.04.01001