Fast implementations of ARX-based lightweight block ciphers (SPARX, CHAM) on 32-bit processor

Byoungjin Seok and Changhoon Lee

Abstract
Recently, many lightweight block ciphers are proposed, such as PRESENT, SIMON, SPECK, Simeck, SPARX, GIFT, and CHAM. Most of these ciphers are designed with Addition–Rotation–Xor (ARX)-based structure for the resource-constrained environment because ARX operations can be implemented efficiently, especially in software. However, if the word size of a block cipher is smaller than the register size of the target device, it may process inefficiently in the aspect of memory usage. In this article, we present a fast implementation method for ARX-based block ciphers, named two-way operation. Moreover, also we applied SPARX-64/128 and CHAM-64/128 and estimated the performance in terms of execution time (cycles per byte) on a 32-bit Advanced RISC Machines processor. As a result, we achieved a large amount of improvement in execution time. The cycles of round function and key schedule are reduced by 53.31% and 31.51% for SPARX-64/128 and 41.22% and 19.40% for CHAM-64/128.

Keywords
SPARX, CHAM, ARX, lightweight block cipher, resource-constrained, ARM

Introduction
In these days, Internet of things (IoT) technologies were rapidly developed, and they are already involved in our lives. As IoT devices are more deeply embedded around us, the security of IoT became more important because they deal with our sensitive data. However, IoT devices consist of small memory and low power compared to generally used PC or laptop.1 These computing environments are called resource-constrained. For the security of resource-constrained devices, security technologies have to be composed more lightly, so we need lightweight block ciphers, which can be the basis of lightweight security technologies. In reality, a lot of lightweight block ciphers are being proposed internationally such as PRESENT,2 SIMON/SPECK,3 Simeck,4 SPARX,5 GIFT,6 and CHAM7. Most of these block ciphers are designed with Addition–Rotation–Xor (ARX)-based structure. ARX-based block ciphers can be implemented faster and more compact than substitution–permutation network (SPN)-based block ciphers. However, this does not mean that ARX-based block ciphers can perfectly perform without any influence that decreases the performance of IoT devices. Moreover, because of the lack of computing resources, it has to be implemented considering their operating environment.

Many of ARX-based block ciphers are recently proposed to have various instances that can support...
different block and key sizes. For example, in SIMON/SPECK, they provide five cases of block size (32, 48, 64, 96, and 128) and seven cases of key size (64, 72, 96, 128, 144, 192, and 256). So, SIMON/SPECK has 10 instances according to a combination of block size and key size, and it has a different number of rounds between each other. The word size can be adjusted according to block size and key size. For example, with smaller key size and block size, they perform with 16 bits of word size. And with a bigger key size and block size, they perform with 32 bits of word size. However, in reality, IoT network consists of heterogeneous devices based on various processors including AVR, MSP430, and ARM. These IoT devices are interconnected with each other. When we apply the block cipher for confidentiality in this situation, we have to implement various instances of block ciphers for communicating with each other, although the block ciphers were designed for a specific target processor. If the block cipher’s word size is smaller than the word size of microcontroller (MCU), it can be inefficient because it creates empty space in the general-purpose register of MCU.

In this article, we introduce a fast implementation method of ARX-based block ciphers, including SPARX-64/128 and CHAM-64/128, on 32-bit processors. To improve block cipher’s speed, we propose a parallel implementation of modular addition and rotation, named two-way operation method to this technique. Also, we modify the round function of SPARX-64/128 and CHAM-64/128 to apply the two-way operation method. This article is organized as follows. In section “Preliminaries,” we describe the SPARX-64/128 and CHAM-64/128, which are our target cipher. And in section “Related work,” we introduce related work about the implementation of lightweight block ciphers. In section “Proposed implementation technique,” we introduce our proposal implementation technique, named two-way operation. And we also propose a fast implementation method of SPARX-64/128 and CHAM-64/128 using two-way operation technique. In section “Experiment,” we describe our experimental results in terms of performance. In section “Conclusion,” we summarize our work and conclude this article.

Preliminaries

Description of SPARX-64/128

SPARX was presented by Dinu et al. in ASIACRYPT 2006. For provable security of ARX-based block cipher, it uses large s-box (ARX-based) and simple linear permutation. This approach is named “Long trail design strategy (LTS)” and it can prove the security against differential and linear cryptanalysis. In SPARX, the large s-box is denoted as $A$, and it is a SPECKEY which is the keyless version of SPECK round function. SPECKEY can be implemented in software because SPECK consists of ARX operations. The linear permutation is linear mixing function $w$ words based on Feistel structure, denoted as $\lambda_w$. The structure of the SPARX round function is a combination, which is called step function, and a combination consists of $r_a$ times of $A$ and one time of $\lambda_w$. The SPARX round function is shown in Figure 1. The key schedule is simply defined by using $A$ and operation, which rotates the word, and the overview of SPARX key schedule is shown in Figure 2.

![Figure 1. SPARX round function.](image-url)
SPARX is a family of ciphers and has three instances according to block size $n$ and key size $k$, denoted as SPARX-$n/k$. In details, each of instances is SPARX-64/128, SPARX-128/128, and SPARX-128/256, and their parameters are shown in Table 1. Depending on block size and key size, step structure, and the key schedule $K^k_n$ are different. In this article, we focus on SPARX-64/128. The step structure of SPARX-64/128 is shown in Figure 3. And the key schedule in SPARX-64/128 is defined as shown in Figure 4, denoted as $K^64_{64}$. All internal operations of SPARX are implemented using 16-bit variables.

**Description of CHAM-64/128**

CHAM was presented by Koo et al. in ICISC 2017. For efficiency of the cipher, CHAM was designed considering the resource-constrained instruction set (8-bit AVR, 16-bit MSP). Specifically, AVR and MSP have no instructions which can rotate an arbitrary number. So, authors designed with the rotation which is close to 0 or 8 as possible in the round function, for example, 1-bit rotation and 8-bit rotation. Also, they design a very simple key schedule which can be calculated without updating the key. These features can be more efficient in terms of software. Likewise the SPARX in section “Description of SPARX-64/128,” CHAM is also a family of block ciphers and has three instances, denoted as CHAM-64/128, CHAM-128/128, and CHAM-128/256. Parameters of CHAM are shown in Table 2.

In contrast to SPARX, there is no change in the structure of the round function and the key schedule in

| Parameter       | SPARX-64/128 | SPARX-128/128 | SPARX-128/256 |
|------------------|--------------|---------------|---------------|
| State words $w$  | 2            | 4             | 4             |
| Key words $v$    | 4            | 4             | 8             |
| Rounds/step $r_a$| 3            | 4             | 4             |
| Steps $n_s$      | 8            | 8             | 10            |

**Table 1. Parameters of SPARX.**

![Figure 2. SPARX key schedule.](image)

![Figure 3. Step structure of SPARX-64/128.](image)

![Figure 4. Key schedule of SPARX-64/128; $K^64_{64}$.](image)
Table 2. Parameters of CHAM.

| Parameter     | CHAM-64/128 | CHAM-128/128 | CHAM-128/256 |
|---------------|-------------|--------------|--------------|
| Block size $n$| 64          | 128          | 128          |
| Key size $k$  | 4           | 4            | 8            |
| Word size $w$ | 16          | 32           | 32           |
| Number of rounds $r$ | 4 | 4 | 8 |

CHAM according to block size $n$ and key size $k$. CHAM round function is illustrated in Figure 5 in every instance, and it just changed word size and bit size of internal operations. And Figure 6 shows the key schedule of CHAM-64/128. As you can see in the word size of instances, CHAM-64/128 is the only one using 16-bit word.

Related work

In this section, we introduce some recently conducted research results that are related to software implementations of ARX-based lightweight block cipher. Most of the existing results were focused on a specific target processor, especially on AVR, MSP, and ARM, and researchers consider the architecture of the target processor or available instruction set. In LightSec ’14, Beaulieu et al. introduced optimized implementation result of SIMON/SPECK on 8-bit AVR processor. They considered the feature of the instruction set of AVR and optimized memory access. Also, they present implementing techniques according to three resource-constrained scenarios (lack of RAM, lack of ROM, and required fast runtime).

In ICUFN ’17, Park et al. introduced implementation technique of Simeck on 16-bit MSP430 processor. Loop unrolling method is generally used for better performance. However, if the ciphertext has to be calculated through many rounds, the code size becomes huge. So, the authors implement the round function of Simeck using loop rolling for code size minimization. Then, they store the index of the loop in general-purpose register to use the register-mode-addressing technique which can reuse the index. This approach can optimize not only code size but also cycles. And in WISA ’17, Park et al. also introduced parallel implementation technique of SIMON/SPECK on 32-bit ARM-NEON processor. They use the NEON instruction set, which is a Single Instruction Multiple Data (SIMD) instructions of ARM-NEON processor. Also, they apply the Single Instruction Multiple Thread (SIMT) method. SIMT program can be parallel processing in the core of the processor. In PlatCon-18, Park et al. improve the performance of Simeck on Intel processor using AVX2 instruction set.

Seo et al. introduced optimized implementation technique of LEA, HIGHT on 8-bit AVR, 16-bit MSP, 32-bit ARM, and 32-bit ARM-NEON in TECS ’18. They consider the number of registers and available instruction set of each processor and then revise the internal operations of target algorithms to minimize the cycles. In ICUFN ’18, Park et al. introduced implementation result of CHAM written in Javascript for use in the web browser. They revised the round function of CHAM to reduce the number of rounds and then conducted an experiment on a PC web browser such as Chrome and Firefox.

Proposed implementation technique

Two-way operation technique

Many of ARX-based block ciphers define internal operations using 16-bit variables, such as a modular addition in $2^{16}$, rotation in 16 bits. However, if these operations work on a 32-bit processor, they can be inefficient because they create empty space in the register. For example, if we calculate a modular addition using 16-bit variables on a 32-bit processor, the status of the register during the operation is shown in Figure 7. The
main idea is that the internal operations of block cipher run parallel using the empty space.

In this article, we named this approach the two-way operation. Two-way operation technique is simple, but it can run two operations at once and get rid of the empty space in registers. It seems similar to SIMD instructions in the perspective of the parallel implementation, which processes multiple data at a time. However, the two-way operation does not use a single instruction. This technique is an alternative method for parallel processing of multiple data on a single variable if the SIMD instruction does not exist in the target processor. The two-way operation is only for wordwise operations. We do not need to consider bitwise operations such as exclusive-or because it works without effect between each bit and also there is no any relation to variable size. It just expands the variable size and put two data to the variable. Otherwise, in the case of wordwise operations, there are relations between data size and variable size.

To calculate two modular additions defined with $2^{16}$ using a 32-bit variable, we add a condition that checks the existence of a carry in the addition process. You can see our proposed two-way modular addition process in Algorithm 1. Two-way modular addition compares which one is bigger between the addition result and original data of operand. These data are the 16 bits from the least significant bit (LSB). If the addition result was smaller than original data, it means that a carry occurs during the addition process. In this case, we subtract integer 1 to the rest of the addition results that means 16 bit from the most significant bit (MSB). It can maintain the non-linearity of modular addition defined with modulo $2^{16}$ in the 32-bit variable separately.

To calculate two rotations defined with $2^{16}$ in parallel, we use the bit masking to bring the arbitrary number. When we rotate arbitrary number $n$ to the left side, we first calculate $n$ bits left shift and $(16 - n)$ bits right shift. This approach is generally used to implement rotation. However, after shift operation, we mask with specific bits, which mean an arbitrary number to both shift result, and then, we calculate exclusive-or to each other. The whole process of two-way rotation is shown in Algorithm 2.

Two-way operation technique can be seen requiring the additional operations like checking a carry or bit masking to calculate two operations separately. However, in terms of performance, it does not require more cost. If we implement the two operations separately, it has to load data to the register twice, and it requires more cycles than other simple mathematical operations. That is, calculating twice to general operations spend more times than two-way modular addition, two-way rotation. Furthermore, when applying the two-way operation technique to block cipher, we
can achieve the additional improvement of performance with reducing some procedure of cipher such as block rotation if the structure of block cipher is revised to apply our proposed technique optimally.

Implementation of SPARX-64/128

Now, we describe how to revise SPARX-64/128 to implement using two-way operation technique. As you can see in section “Description of SPARX-64/128,” SPARX round function uses $A$ function, which substitutes input bits to output bits as following ARX-box in the step function. We suppose that the ARX-box is just some function-based ARX operations. Also, each $A$ function handles the divided blocks, including MSB-side of plaintext and LSB-side of plaintext. In this procedure, there is no relation between two blocks until $L$ function. It means that it can be reconfigured dealing with both of the two blocks. The revised SPARX-64/128 round function is shown in Figure 8. If we use this round function, it can reduce the number of times a half to load data in $A$ function. It does not be doubled the performance because it also gets some cost when we were applying the two-way operation technique. However, as we mentioned before, the operation for loading data requires more cycles than other operations and so we can get a substantially improved performance. You can see the details of the performance of revised SPARX-64/128 in section “Experiment.”

For using the revised round function, we have to align the round keys to combined form that there are two consecutive round keys. It means that we calculate the two round keys, sort the key words, and then make the round keys by combining the sorted key words one by one. Thus, this procedure can cause a decrease in performance. So, we first prepare words which are the results of word rotation to the left side and then we combine the results with original words. Also, we revised the key schedule which operates with the revised round function. The revised key schedule uses $A^*$ function, which revised $A$ functions in round function by applying two-way operations. The revised key schedule of SPARX-64/128 is shown in Figure 9. It can calculate the two consecutive round keys, and it operates using 32-bit variables instead of 16 bits. Therefore, it can be more efficient than the original key schedule on a 32-bit processor.

**Figure 8.** The revised round function of SPARX-64/128.

**Implementation of CHAM-64/128**

We also revised round function and key schedule of CHAM-64/128. The round function operates to change only one word in one round and then rotates the whole words without any changes in other words. It means that we can calculate the results of each round if we know two words used for encrypting one word. Using this feature, we can separate the results of even-round and odd-round. The revised round function of CHAM-64/128 works like as shown in Figure 10. In this revised round function, there is no word rotation because we separate each result and we can get the same results after operating four rounds of original round function. Therefore, this revised key schedule could be more efficient than the original key schedule.

To the key schedule, we expand the variable size from 16 bits to 32 bits and then combine two key words. When combining the key words, we configure a gathering of even words or odd words for getting the aligned round key. The round key of the revised key schedule can be applied immediately to the revised round function. Also, it can calculate four round keys at once. The revised key schedule of CHAM-64/128 is shown in Figure 11.

**Experiment**

**Environment for experiment**

For testing our implementations, including SPARX-64/128 and CHAM-64/128, we chose a target board which
is based on 32-bit ARM Cortex-M3 processor. The target board is LAUNCHXL-CC2640R2 made by Texas Instrument Inc. for evaluation of IoT software. The details of the target board are shown in Figure 12. Also, we measured the performance on the Code Composer Studio (CCS), which runs a cross-compile. This target board includes a debugger, so we can get the number of cycles that the processor consumes for processing the instructions. Our implementations are written in C languages. You can check and download the implementation results in our repository.\(^{16}\)

**Performance results**

To check the improvement of our implementations, we prepared two versions of SPARX-64/128 and CHAM-
One was programmed following own algorithm without any revision, and the another was programmed following the revised algorithm, which is based on two-way operations technique. Also, we compared the performances of the key schedule and the encryption. Our experiment results are shown in Table 3. Two-way operations-based implementations performed better than original implementations. In the case of SPARX-64/128, the consuming of cycle reduced about 31.31% in key schedule and 53.31% in encryption. Also, in the case of CHAM-64/128, the consuming of cycle reduced about 19.40% in key schedule and 41.22% in encryption. The performance of round functions improved almost twice, and it is the result of the number of rounds reduced half of the original algorithm by revising based on two-way operations. However, the performance improvement of key schedule was smaller than the case of the round function. Because the round keys have to be aligned in each round to apply the revised round function, the revised round function also requires the alignment of the input, and this alignment is carried out just at the point of beginning and end of encryption.

**Conclusion**

In this article, we presented fast implementation techniques for ARX-based lightweight block ciphers, including SPARX-64/128 and CHAM-64/128 on 32-bit processors. We proposed a method which can calculate two modular additions or two rotations defined with \(2^{16}\) on 32-bit variable in parallel, named two-way operations technique. Also, we revised the round function and the key schedule of our target ciphers using two-way operation technique. In our experiment, the revised block ciphers performed better than the original block ciphers. The performance improvements of SPARX-64/128 are about 31.31% in key schedule and 53.31% in encryption. Also, the performance improvements of CHAM-64/128 are about 19.40% in key schedule and 41.22% in encryption. The proposed technique is not limited to the above block ciphers. If we run the block cipher that is defined with smaller word size than the target processor, it can be applied to all ARX-based block ciphers by revising the key schedule and round function following two-way operations technique.

In this article, the results show the improvement of SPARX and CHAM implementation performance using the proposed technique. Our experiment was conducted with C language. However, in a practical environment, many techniques are using the assembly instruction set. In future works, we will research about optimization techniques for various block ciphers and apply other optimization techniques such as SIMD assembly instructions or memory-access optimization.

**Declaration of conflicting interests**

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.
Funding
The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported as part of Military Crypto Research Center (UD170109ED) funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD).

ORCID iD
Byoungjin Seok https://orcid.org/0000-0002-8130-5011

References
1. Hossain MM, Fotouhi M and Hasan R. Towards an analysis of security issues, challenges, and open problems in the Internet of things. In: Proceedings of the IEEE world congress on services (SERVICES), New York, 27 June–2 July 2015, pp.21–28. New York: IEEE.
2. Bogdanov A, Knudsen LR, Leander G, et al. Present: an ultra-lightweight block cipher. In: Proceedings of the international workshop on cryptographic hardware and embedded systems, Vienna, 10–13 September 2007, pp.450–466. Berlin: Springer.
3. Beaulieu R, Treatman-Clark S, Shors D, et al. The Simon and speck lightweight block ciphers. In: Proceedings of the 52nd ACM/EDAC/IEEE design automation conference (DAC), San Francisco, CA, 8–12 June 2015, pp.1–6. New York: IEEE.
4. Yang G, Zhu B, Suder V, et al. The Simeck family of lightweight block ciphers. In: Proceedings of the 52nd ACM/EDAC/IEEE design automation conference (DAC), San Francisco, CA, 8–12 June 2015, pp.307–329. Berlin: Springer.
5. Dinu D, Perrin L, Udovenko A, et al. Sparx: a family of ARX-based lightweight block ciphers provably secure against linear and differential attacks. In: Proceedings of NIST workshop on Lightweight Crypto CRYPTO’16, 18 October 2016, p.121. Gaithersburg, MD: NIST.
6. Banik S, Pandey SK, Peyrin T, et al. Gift: a small present. In: Proceedings of the 19th international conference on cryptographic hardware and embedded systems, Taipei, Taiwan, 25–28 September 2017, pp.321–345. Berlin: Springer.
7. Koo B, Roh D, Kim H, et al. Cham: a family of lightweight block ciphers for resource-constrained devices. In: Proceedings of the international conference on information security and cryptology, Seoul, South Korea, 29 November–1 December 2017, pp.3–25. Berlin: Springer.
8. Hahm O, Baccelli E, Petersen H, et al. Operating systems for low-end devices in the Internet of things: a survey. IEEE Internet Things 2015; 3(5): 720–734.
9. Beaulieu R, Shors D, Smith J, et al. The Simon and speck block ciphers on AVR 8-bit microcontrollers. In: Proceedings of the international workshop on lightweight cryptography for security and privacy, Istanbul, 1–2 September 2014, pp.3–20. Berlin: Springer.
10. Park T, Seo H, Lee G, et al. Efficient implementation of simeck family block cipher on 16-bit msp430. In: Proceedings of the ninth international conference on ubiquitous and future networks (ICUFN), Milan, 4–7 July 2014, pp.983–988. New York: IEEE.
11. Park T, Seo H, Lee G, et al. Parallel implementations of Simon and speck, revisited. In: Proceedings of the international workshop on information security applications, Jeju, South Korea, 24-26 August 2017, pp.283–294. Berlin: Springer.
12. Park T, Seo H and Kim H. Fast implementation of simeck family block ciphers using avx2. In: Proceedings of the international conference on platform technology and service (PlatCon), Jeju, South Korea, 29–31 January 2018, pp.1–6. New York: IEEE.
13. Seo H, Jeong I, Lee J, et al. Compact implementations of ARX-based block ciphers on IoT processors. ACM T Embed Comput S 2018; 17(3): 60.
14. Park C, Park T, Seo H, et al. Optimization of CHAM encryption algorithm based on Javascript. In: Proceedings of the tenth international conference on ubiquitous and future networks (ICUFN), Prague, 3–6 July 2018, pp.774–778. New York: IEEE.
15. Texas Instruments Inc. SimpleLink™ Bluetooth® low energy CC2640R2F wireless MCU LaunchPad™ development kit: LAUNCHXL-CC2640R2. Dallas, TX: Texas Instruments Inc., 2017.
16. Seok B. The implementation results of SPARX-64/128, CHAM-64/128, 2019, https://github.com/ByoungJin-Seok/Two-way