Parallel Implementations of CHAM

Hwajeong Seo[1], Kyuhwang An[1], Hyeokdong Kwon[1],
Taehwan Park[2], Zhi Hu[3], and Howon Kim[2⋆]

[1] Hansung University, Republic of Korea
hwajeong84@gmail.com, tigerk9212@gmail.com, hdgwon@naver.com
[2] Pusan National University, Republic of Korea
{pth5804,howonkim}@gmail.com
[3] Central South University, China
huzhi_math@csu.edu.cn

Abstract. In this paper, we presented novel parallel implementations of CHAM-64/128 block cipher on modern ARM-NEON processors. In order to accelerate the performance of the implementation of CHAM-64/128 block cipher, the full specifications of ARM-NEON processors are utilized in terms of instruction set and multiple cores. First, the SIMD feature of ARM processor is fully utilized. The modern ARM processor provides 2×16-bit vectorized instruction. By using the instruction sets and full register files, total 4 CHAM-64/128 encryptions are performed at once in data parallel way. Second, the dedicated SIMD instruction sets, namely NEON engine, is fully exploited. The NEON engine supports 8×16-bit vectorized instruction over 128-bit `Q` registers. The 24 CHAM-64/128 encryptions are performed at once in data parallel way. Third, both ARM and NEON instruction sets are well re-ordered in interleaved way. This mixed approach hides the pipeline stalls between each instruction set. Fourth, the multiple cores are exploited to maximize the performance in thread level. Finally, we achieved the 4.2 cycles/byte for implementation of CHAM-64/128 on ARM-NEON processors. This result is competitive to the parallel implementation of LEA-128/128 and HIGHT-64/128 on same processor.

Keywords: CHAM, ARM-NEON Processor, Parallel Computation, Software Implementation

1 Introduction

Modern processors support Single Instruction Multiple Data (SIMD) instruction sets, including SSE/AVX for INTEL and NEON for ARM, respectively. Since the SIMD instruction sets can issue multiple tasks in single instruction, traditional Single Instruction Single Data (SISD) based implementations can be accelerated by using SIMD instruction sets. Recently, many block cipher implementations, including LEA, HIGHT, SIMON, and SPECK, were re-programmed in SIMD instruction sets and achieved much higher performance than previous

⋆ Corresponding Author

classical works [16, 14, 13]. Furthermore, the modern processors support multiple cores/threads and each core/thread performs the task in parallel way. The most well-known multiple core framework is OpenMP and many works also evaluated the maximum performance of block cipher implementations by using OpenMP library, including LEA and HIGHT [21, 15]. In this paper, we exploit a parallel computing power into the novel lightweight block cipher, CHAM, which was released at ICISC'17 by NSR [10]. The CHAM block cipher can be efficiently implemented in both platforms ranging from low-end embedded microprocessors to high-end personal computers. Contrary to previous implementations of CHAM block cipher, which mainly focused on serial computations on embedded processors, this paper introduces the feasibility of parallel implementation of CHAM on modern ARM-NEON processors. To improve the performance, we fully exploit the SIMD architectures with novel techniques. It is also worth to note that all the proposed methods are not limited to CHAM implementation but this can be applied to the other cryptography implementations with simple modifications

The remainder of this paper is organized as follows. In Section 2, the basic specifications of CHAM block cipher and target ARM-NEON platform are described. In Section 3, the compact parallel implementations of CHAM block cipher on ARM-NEON platform are described. In Section 4, the performance of proposed methods in terms of execution timing is evaluated. Finally, Section 5 concludes the paper.

2 Related Works

2.1 CHAM Block Cipher

Lightweight cryptography is a fundamental technology to reduce the hardware chip size and accelerate the execution time for the Internet of Things (IoT) devices. Recently, a number of block ciphers have been designed for being lightweight features. The approaches are largely divided into two approaches, including Substitute Permutation Network (SPN) and Addition-Rotation-XOR (ARX). For the SPN architecture, PRESENT and GIFT block ciphers received the attention. The SPN block ciphers utilize the simple 4-bit S-BOX and permutation to achieve the high security as well [1, 5]. For the ARX architecture, HIGHT, LEA, SPECK, SIMON, and CHAM block ciphers have been proposed. The ARX block cipher exploit the simple ARX operations [9, 2, 8, 10].

In ICISC 2017, a family of lightweight block ciphers CHAM was announced by the Attached Institute of ETRI [10]. The family consists of three ciphers, including CHAM-64/128, CHAM-128/128, and CHAM-128/256. The CHAM block ciphers are of the generalized 4-branch Feistel structure based on ARX operations. The ARX operations ensure high performance on both software and hardware platforms and security against the timing attack. Particularly, the CHAM block cipher does not require updating a key state by using of $stateless-on-the-fly$ key schedule. With above nice features, the implementation of CHAM block ciphers achieved outstanding figures. In this paper, we targeted

Table 1. Instruction set summary for ARM

Mnemonics	Operands	Description	Cycles
ADD	Rd, Rr	Add without Carry	1
EOR	Rd, Rr	Exclusive OR	1
ROL	Rd, Rr,#imm	Rotate Left Through Carry	1
ROR	Rd, Rr,#imm	Rotate Right Through Carry	1

Table 2. Instruction set summary for NEON

Mnemonics	Operands	Description	Cycles
VADD	Qd,Qn,Qm	Vector Addition	1
VEOR	Qd,Qn,Qm	Vector Exclusive-or	1
VSHL	Qd,Qm,#imm	Vector Left Shift	1
VSRI	Qd,Qm,#imm	Vector Right Shift with Insert	2

the most lightweight variant of CHAM, namely CHAM-64/128, for lightweight implementations. However, this work is not limited to only CHAM-64/128, but the technique is also easily applied to other variants such as CHAM-128/128 and CHAM-128/256.

2.2 ARM-NEON Processor

Advanced RISC Machine (ARM) is an instruction set architecture (ISA) design by ARM for high-performance 32-bit embedded applications. Although ARM cores are usually larger and more complex than 8-bit AVR and 16-bit MSP processors, most ARM designs also have competitive features, in terms of low power consumptions, high-speed computations, and high code density. The ARM family has developed from the traditional ARM1 to advanced Cortex architectures in these days. They provide large number of pipeline stages and various caches, SIMD extensions and simple load/store architecture. Most instructions of the ARM are computed in a single cycle except memory access and some arithmetic instructions. In particular, the inline-barrel shifter instruction allows the shifted/rotated second operand without loss of clock cycles before the main operations. The load and store operations are performed in multiple data. Their 32-bit wise instructions mainly used for ARX architectures are described in Table 1.

NEON is a 128-bit SIMD architecture from the modern ARM Cortex-A series. One of the biggest difference between traditional ARM processors and new ARM-NEON processors is SIMD features. The NEON engine offers 128-bit wise registers and instructions. Each register is considered as a vector of elements of the same data type and this data type can be signed/unsigned 8-bit, 16-bit, 32-bit, or 64-bit. The detailed instructions for ARX operations are described in Table 2. This feature provides a more precise operation in various word sizes, which allows us to perform multiple data in single instruction, and the NEON

engine can accelerate data processing by at least 3X that provided by ARMv5 and at least 2X that provided by ARMv6 SIMD instructions. This nice structure have accelerated the implementations by converting single instruction single data model to SIMD in previous works. In CHES 2012, NEON-based cryptography implementations including Salsa20, Poly1305, Curve25519, and Ed25519 were presented [4]. In order to enhance the performance, the authors provided novel bit-rotation, integration of multiplication, and reduction operations exploiting NEON instructions. In CT–RSA'13, ARM–NEON implementation of `Grøstl` shows that 40 % performance enhancements than the previous fastest ARM implementation [7]. In HPEC 2013, a multiplicand reduction method for ARM-NEON was introduced for the NIST curves [6]. In CHES 2014, the Curve41417 implementation adopts 2-level Karatsuba multiplication in the redundant representation [3]. In ICISC 2014, Seo et al. introduced a novel 2-way Cascade Operand Scanning (COS) multiplication for RSA implementation [17, 18]. In ICISC 2015, Seo et al. introduced a efficient modular multiplication and squaring operations for NIST P-521 curve [19]. Recently, Ring-LWE implementation is also accelerated by taking advantages of NEON instructions [11].

For the case of ARM implementations for ARX block ciphers, LEA block cipher implementations have been actively studied. First LEA implementation utilized the basic 32-bit ARM instruction set for high performance [8]. In [20], the parallel implementation of LEA was suggested, which utilized the SIMD instruction sets for high performance. In particular, the interdependency between each instruction set is optimized and multiple plaintexts and round keys are used in parallel way. In [21, 15], the LEA block cipher is efficiently implemented in ARM and NEON instruction sets. Afterward, the compact ARM and NEON implementations are performed in interleaved way. This approach hides the latency of ARM computations into NEON computations and enhances the performance significantly. In [15], the auxiliary function is implemented with 4-bit wise LUT operation supported in NEON architecture (`vtbl`), which translates the auxiliary function into two 4-bit wise LUT operations. In [14, 13], lightweight block ciphers, including SIMON and SPECK, are efficiently implemented over ARM-NEON processor. For the case of new lightweight CHAM block cipher, there are no ARM–NEON implementations. In this paper, we introduce a new implementation technique and new ARM–NEON results for CHAM block cipher.

3 Proposed Parallel Implementations of CHAM-64/128

In this section, we investigate the compact implementations of CHAM-64/128 on ARM–NEON processors. Particularly, we targeted the most lightweight variant of CHAM (i.e. CHAM-64/128) for lightweight implementations. However, this work is not limited to only CHAM-64/128, but the technique is also easily applied to other variants such as CHAM-128/128 and CHAM-128/256.

Table 3. 32-bit instructions over 32-bit ARM, where `R1` and `R0` represent destination and source registers

Addition	Exclusive-or	Right Rotation by 31
`ADD R1, R1, R0`	`EOR R1, R1, R0`	`ROR R1, #31`

3.1 Parallel Implementation in ARM

CHAM-64/128 block cipher can be efficiently implemented on the 16-bit processor, because the platform provides 16-bit wise addition, bit-wise exclusive-or, and rotation operations, which are main 16-bit ARX operations for CHAM-64/128 block cipher. However, our target processor is 32-bit ARM processor and it is inefficient to implement the 16-bit wise operations in 32-bit ARM instructions described in Table 3, since the half of word is not utilized.

In order to optimize the 16-bit operations in 32-bit instruction sets, legacy ARM SIMD instruction set (`UADD16 R1, R1, R0`) is utilized. The instruction set divides the 32-bit registers into lower 16-bit and higher 16-bit and performs 2 16-bit addition in parallel way (like 2-way SIMD). Since the bit-wise exclusive-or does not generate the carry bits, the ARM instruction set is directly used without modification. The CHAM-64/128 block cipher requires 16-bit wise left rotation by 1-bit and 8-bit offsets. In order to establish the 2-way 16-bit wise rotation, the new rotation techniques are exploited. In Algorithm 1, the descriptions of left rotation by 1-bit for 2 16-bit variables are given. Unlike 32-bit wise rotation, ARM processor does not support 16-bit wise computations. For this reason, two mask variables (`0x0101` and `0xFEFE`) are set. Both mask variables are used to separate the 32-bit rotated input data into overflow bits and original bits. Afterward, both bits are integrated into one, which generates the rotated outputs. In Algorithm 2, the descriptions of left rotation by 8-bit for 2 16-bit variables are given. Similarly, one mask variable (`0x0F0F`) is used to handle the bits.

Algorithm 1: Left rotation by 1-bit for 2 16-bit variables

Input: input data (`R1`), mask values (`0x0101: R2`, `0xFEFE: R3`), temporal variable (`R4`)
Output: output data (`R0`)

```
1: ROR R0, R1, #31
2: AND R4, R0, R2
3: AND R0, R0, R3
4: ADD R0, R0, R4, ROR#16
```

For the data load, only 32-bit wise access is available in 32-bit ARM processor. However, the word size of CHAM-64/128 is 16-bit and SIMD instruction requires 16-bit wise alignments. For this reason, all input data should be finely

Algorithm 2: Left rotation by 8-bit for 2 16-bit variables

Input: input data (`R1`), mask values (`0x0F0F: R2`), temporal variable (`R3`)
Output: output data (`R0`)

```
1: AND R3, R1, R2
2: ROR R1, R1, #8
3: AND R1, R1, R2
4: ADD R1, R1, R3, ROR#24
```

aligned before performing the parallel computations. The detailed descriptions are given in Algorithm 3. In Algorithm 3, 4 plaintexts are transposed in parallel way. In Step 1 and 3, lower 16-bit plaintexts are extracted[4]. In Step 2 and 4, the lower 16-bit part of other plaintexts are added to the extracted plaintext. In Step 5–10, the higher 16-bit plaintexts are extracted and combined. Similarly in Step 11–20, the remaining plaintexts are transposed. After the encryption, transposed data sets are re-ordered to the original format.

For the register level optimization, we assigned 8 registers for plaintext, 3 registers for mask variables, 2 register for temporal storage, and 1 register for memory pointer in encryption. The round keys are stored in the stack and the variables are directly accessed through the stack pointer. This is available since the round key size is very short for CHAM block ciphers.

3.2 Parallel Implementation in NEON

The basic 16-bit wise ARX operations of CHAM-64/128 block cipher are established with NEON instructions (See Table 4) and 24 encryptions are performed at once. This implementation assumes that all encryption shares same round key pair. For this reason, only one round key is loaded and duplicated from ARM register to the NEON registers, which reduces the number of memory access to load the round key pairs. Similar to the ARM SIMD approach, input data should be re-ordered to meet the SIMD friendly data format. The transpose operation is performed with 12 NEON transpose instruction sets. The detailed descriptions are given in Algorithm 4. In each 128-bit `Q` register, 2 64-bit plaintexts are stored. In order to align the data into 16-bit wise, two different transpose operations are required. First the operands are transposed by 16-bit wise. Afterward, the data is transposed again by 32-bit wise and this outputs the finely 16-bit aligned results. For the register level optimization, we assigned 12 registers for plaintext and 4 registers for temporal storage. This ensures 24 CHAM-64/128 encryptions at once.

3.3 Interleaved Implementation

ARM processor and NEON engine are independent processing units, which means that they can issue the computations independently. This parallel fea-

[4] `UXTH` instruction only extracts lower 16-bit from 32-bit register

Algorithm 3: Transpose operation for 4 plaintext in parallel way

Input: input data (`R5--R12`)
Output: output data (`R2--R9`)

```
 1: UXTH R2, R5
 2: ADD R2, R2, R7, LSL#16

 3: UXTH R3, R9
 4: ADD R3, R3, R11, LSL#16

 5: LSR R4, R5, #16
 6: ROR R7, R7, #16
 7: ADD R4, R4, R7, LSL#16

 8: LSR R5, R9, #16
 9: ROR R11, R11, #16
10: ADD R5, R5, R11, LSL#16

11: UXTH R7, R6
12: LSR R9, R6, #16
13: ADD R6, R7, R8, LSL#16

14: UXTH R7, R10
15: LSR R11, R10, #16
16: ADD R7, R7, R12, LSL#16

17: ROR R8, R8, #16
18: ADD R8, R9, R8, LSL#16

19: ROR R12, R12, #16
20: ADD R9, R11, R12, LSL#16
```

Table 4. 16-bit instructions over 128-bit ARM-NEON, where `Q1` and `Q0` represent destination and source registers

Addition	Exclusive-or	Right Rotation by 9
`VADD.I16 Q1, Q1, Q0`	`VEOR Q1, Q1, Q0`	`VSHL.I16 Q1, Q0, #9`
		`VSRI.16 Q1, Q0, #7`

Algorithm 4: Transpose operation for 24 plaintext in parallel way

Input: input data (`Q0--Q11`)
Output: output data (`Q0--Q11`)

```
VTRN.16 Q0, Q1
VTRN.16 Q2, Q3
VTRN.16 Q4, Q5
VTRN.16 Q6, Q7
VTRN.16 Q8, Q9
VTRN.16 Q10, Q11

VTRN.32 Q0, Q2
VTRN.32 Q1, Q3
VTRN.32 Q4, Q6
VTRN.32 Q5, Q7
VTRN.32 Q8, Q10
VTRN.32 Q9, Q11
```

ture can be utilized to improve the performance by hiding the latency of ARM operations into that of NEON [15]. In sequential order, the processing time is exactly the sum of ARM and NEON execution timing while parallel order hides the ARM timing into NEON timing. The interleaved CHAM-64/128 encryption is performed as follows.

$$\texttt{Load} \rightarrow \texttt{Transpose} \rightarrow \texttt{Encryption} \rightarrow \texttt{Transpose} \rightarrow \texttt{Store}$$

Throughout the execution, `Transpose` and `Encryption` routines are performed in interleaved way. Only, `LOAD` and `STORE` routines are sequentially performed since it has inter-dependency between ARM and NEON instruction sets. In the interleaved CHAM implementation, 24 CHAM encryptions in NEON engine and 4 CHAM encryption are performed within optimized execution timing. This approach significantly reduces the computation timing for ARM processor and enhances the performance.

3.4 Multiple Thread Utilization

Recent ARM-NEON processors have multiple cores and each core can perform independent work loads in parallel way. In order to exploit the full specifications of multiple processing, SIMT programming library, namely OpenMP, is a good candidate. The target device has four physical cores, which can lead to theoretically 4 times of performance improvements. The basic implementations are parallelized using OpenMP library. The detailed program codes are described in Table 5. The `id` variables are critical data and defined as `private`. The `PLAINTEXT` is defined as `shared` variables. The `PLAINTEXT` is indexed by `id` variables, which ensures that each thread accesses to different data. Since we

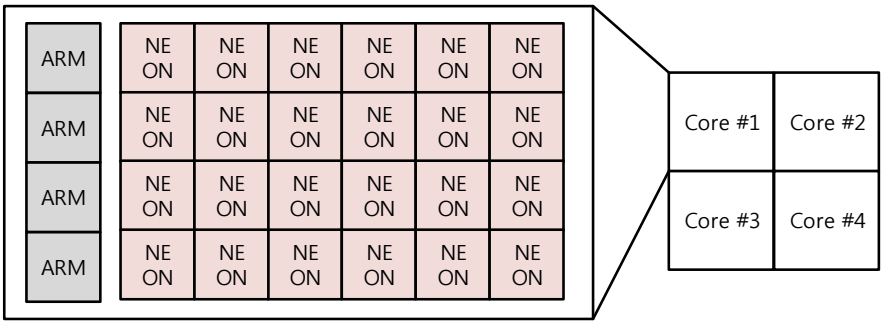

Fig. 1. Detailed architectures of parallel CHAM-64/128 encryptions in multiple cores

Table 5. Parallel Implementation of CHAM-64/128 in OpenMP

```
#pragma omp parallel private(id) shared(PLAINTEXT)
    #pragma omp for
      for(id=0;id<TOTAL;id++)
          CHAM-64/128_Encryption(ROUNDKEY,PLAINTEXT[id]);
```

assigned each encryption operation into single thread, the four different CHAM-64/128 encryption routines are performed in four different cores. As we can see in Figure 1, 4 encryption (ARM) and 24 encryptions (NEON) in each core are performed, simultaneously. Total 112 (28×4) parallel encryptions are performed at once in 4 different cores.

4 Evaluation

To evaluate the performance of the CHAM-64/128 block cipher on the ARM–NEON processor, we used a Raspberry Pi 3 Model B (Cortex–A53 quad-core processor). For the ARM-NEON processor, we measured the execution time in both single-core and multiple-core cases. The performance is measured in system time function.

The comparison results are drawn in Table 6. The previous CHAM-64/128 implementation on ARM instruction sets achieved the 134 cycle/byte [10]. Since the work does not utilize the NEON instruction set and target processor is the low-end processor, the low performance is achieved. Unlike previous works, we utilized the ARM SIMD instruction sets and tested over high-end processors. The ARM SIMD based implementation utilizes four CHAM-64/128 encryptions at once and it enhances the performance by 65% for ARM platform. To the best of my knowledge, this is the first parallel implementation of CHAM-64/128 block cipher. For this reason, we only report our results in Table 6. The NEON requires 13.9 clock cycles per byte for CHAM-64/128 implementations. This is

3.26x faster than ARM SIMD based implementation. The both ARM and NEON implementations are finely integrated together, which hides the latency for ARM instruction sets. The results show 4.3% performance enhancements.

Table 6. Comparison of CHAM-64/128 encryptions on ARM/NEON architectures, c/b: cycle/byte, 1 : one way communication over Cortex-M3 processor.

Method	Speed(c/b)	Instruction
Koo et al. [10]	134[1]	ARM
Proposed Method ver 1	45.6	ARM
Proposed Method ver 2	13.9	NEON
Proposed Method ver 3	13.3	ARM–NEON

The performance is accelerated again by exploiting the full multiple cores in the platforms. Since the target platform is quad-core architecture, it improves the performance by increasing the number of threads. Interestingly, the performance decreases when the number of threads is larger than the number of cores, because many number of threads requires a number of context-switching procedures and this causes performance bottleneck (See Figure 2). Finally, fully parallelized CHAM-64/128 implementations achieved the 4.2 cycle/byte with 4 threads. This is not exactly four times faster than single core but this is the highest CHAM-64/128 performance reported ever. Compared with other lightweight block ciphers such as LEA-128/128 and HIGHT-64/128, the proposed CHAM-64/128 implementations show very competitive results.

5 Conclusion

In this paper, we presented new parallel implementation methods for CHAM-64/128 block cipher on representative SIMD platforms, namely ARM-NEON. We firstly optimized the both ARM and NEON implementations after then both methods are performed in interleaved way. The implementation results are further accelerated by taking advantages of multiple cores. Finally, we achieved performance enhancement up-to 13.3 cycle/byte with single core implementation, which is 90% faster than previous implementations on ARM processors. By enabling the features of multiple cores, we achieved 4.2 cycle/byte which improved again the our single core results by 68.4%.

The proposed methods improved the CHAM-64/128 block cipher on ARM-NEON platforms. For this reason, there are many future works remained. First, we can directly improve the similar ARX block ciphers such as SPECK and SIMON. Recent works by [14, 13] do not consider any techniques covered in this paper. For this reason, we expect high performance enhancements by using the proposed methods. Second, we only explore the ARM-NEON platform. However, INTEL and AMD processors also provide SIMD instructions such as AVX and

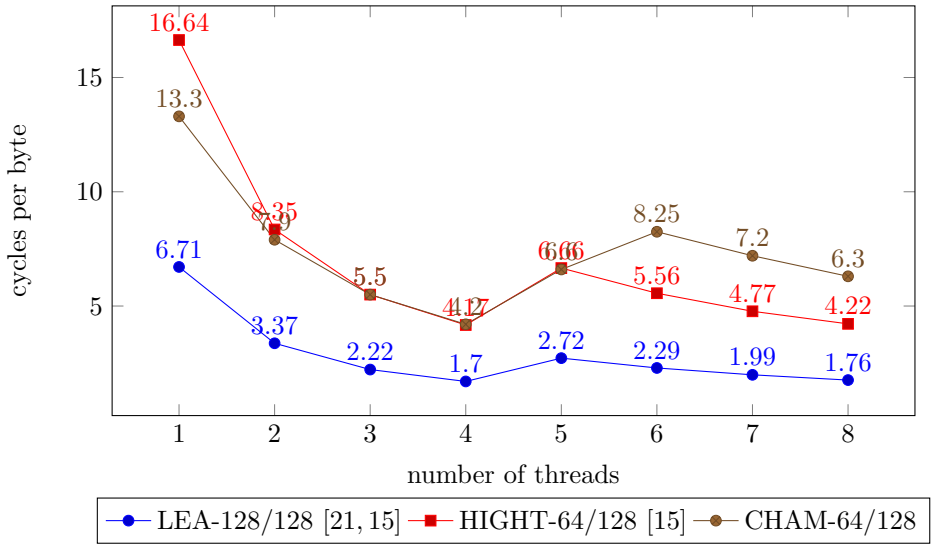

Fig. 2. Comparison of LEA-128/128, HIGHT-64/128, and CHAM-64/128 block ciphers results on ARM-NEON processors (Raspberry Pi 3) in terms of execution time (cycles per byte) depending on the number of threads

SSE. Since the SIMD instructions are very similar to NEON instructions, this can be directly applied to the other SIMD instruction sets. Furthermore, these platforms support multiple cores and multiple threads. We can further explore the strong features of OpenMP on these platforms.

6 Acknowledgement

This work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1C1B5075742) and the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(2014-1-00743) supervised by the IITP(Institute for Information & communications Technology Promotion). Zhi Hu was partially supported by the Natural Science Foundation of China (Grant No. 61602526).

References

1. S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo. GIFT: a small PRESENT. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 321–345. Springer, 2017.
2. R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers. The SIMON and SPECK lightweight block ciphers. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

3. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and P. Schwabe. Kummer strikes back: new DH speed records. In *Advances in Cryptology–ASIACRYPT 2014*, pages 317–337. Springer, 2014.
4. D. J. Bernstein and P. Schwabe. NEON crypto. In *Cryptographic Hardware and Embedded Systems–CHES 2012*, pages 320–339. Springer, 2012.
5. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 450–466. Springer, 2007.
6. A. Faz-Hernández, P. Longa, and A. H. Sánchez. Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV–GLS curves. In *Topics in Cryptology–CT-RSA 2014*, pages 1–27. Springer, 2014.
7. S. Holzer-Graf, T. Krinninger, M. Pernull, M. Schläffer, P. Schwabe, D. Seywald, and W. Wieser. Efficient vector implementations of AES-based designs: a case study and new implemenations for Grøstl. In *Topics in Cryptology–CT-RSA 2013*, pages 145–161. Springer, 2013.
8. D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D.-G. Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In *Information Security Applications–WISA 2013*, pages 3–27. Springer, 2013.
9. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, et al. HIGHT: A new block cipher suitable for low-resource device. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 46–59. Springer, 2006.
10. B. Koo, D. Roh, H. Kim, Y. Jung, D.-G. Lee, and D. Kwon. CHAM: A family of lightweight block ciphers for resource-constrained devices. In *International Conference on Information Security and Cryptology (ICISC'17)*, 2017.
11. Z. Liu, R. Azarderakhsh, H. Kim, and H. Seo. Efficient software implementation of Ring-LWE encryption on IoT processors. *IEEE Transactions on Computers*, 2017.
12. D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright. Fast software AES encryption. In *Fast Software Encryption–FSE 2010*, pages 75–93. Springer, 2010.
13. T. Park, H. Seo, M. Garam Lee, A.-A. Khandaker, Y. Nogami, and H. Kim. Parallel implementations of SIMON and SPECK, revisited.
14. T. Park, H. Seo, and H. Kim. Parallel implementations of SIMON and SPECK. In *Platform Technology and Service (PlatCon), 2016 International Conference on*, pages 1–6. IEEE, 2016.
15. H. Seo, I. Jeong, J. Lee, and W.-H. Kim. Compact implementations of ARX-based block ciphers on IoT processors. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(3):60, 2018.
16. H. Seo and H. Kim. Low-power encryption algorithm block cipher in JavaScript. *J. Inform. and Commun. Convergence Engineering*, 12(4):252–256, 2014.
17. H. Seo, Z. Liu, J. Großschädl, J. Choi, and H. Kim. Montgomery modular multiplication on ARM-NEON revisited. In *Information Security and Cryptology–ICISC 2014*, pages 328–342. Springer, 2014.
18. H. Seo, Z. Liu, J. Großschädl, and H. Kim. Efficient arithmetic on ARM-NEON and its application for high-speed RSA implementation. *Security and Communication Networks*, 9(18):5401–5411, 2016.
19. H. Seo, Z. Liu, Y. Nogami, T. Park, J. Choi, L. Zhou, and H. Kim. Faster ECC over $\mathbb{F}_{2^{521}-1}$ (feat. NEON). In *International Conference on Information Security and Cryptology*, pages 169–181. Springer, 2015.

20. H. Seo, Z. Liu, T. Park, H. Kim, Y. Lee, J. Choi, and H. Kim. Parallel implementations of LEA. In *Information Security and Cryptology–ICISC 2013*, pages 256–274. Springer, 2013.
21. H. Seo, T. Park, S. Heo, G. Seo, B. Bae, Z. Hu, L. Zhou, Y. Nogami, Y. Zhu, and H. Kim. Parallel implementations of LEA, revisited. In *International Workshop on Information Security Applications*, pages 318–330. Springer, 2016.