Parallel Implementations of   
SEED on ARM Processors

Minho Song, Siwoo Eum, Sangwon Kim, Hwajeong Seo\*

Hansung University, Seoul (02876), South Korea

smino0906@gmail.com, shuraatum@gmail.com, kim3875@gmail.com, hwajeong84@gmail.com\*

*Abstract*— The SEED block cipher is a 128-bit symmetric key encryption algorithm developed by the Korea Internet Security Agency (KISA) and it is predominantly used in South Korea. This paper introduces a parallel implementation of the SEED block cipher on ARMv8 processors. The ARMv8 processor represents the latest 64-bit ARM architecture and  
includes ASIMD (Advanced Single Instruction Multiple Data) supporting the parallel processing. By making good use of specific instructions in ASIMD, the efficiency of parallel implementation can be improved. Utilizing this nice capability, we implemented 8 parallel encryption blocks to enhance the substitution layer of the SEED block cipher using two different S-boxes. Our implementation achieved a 5.66× performance improvement for 8-plaintext scenarios compared to previous works.

Keywords—SEED block cipher, parallel implementations, ARM64

# Introduction

As data sizes grow and internet speeds increase, the need for faster data encryption also rises. This trend is driven by advancements in hardware that enable quick operations and offer a range of functionalities. The ARMv8 architecture, the latest 64-bit ARM design, supports Advanced Single Instruction Multiple Data (ASIMD), also known as the NEON engine, which allows for parallel arithmetic operations on multiple data points, enhancing computational efficiency. Various ciphers are implemented using this ASIMD[1-3].

In this paper, we introduces the SEED block cipher implementation using ASIMD, which is implemented in various environments[4-5]. The SEED block cipher, established as a Telecommunications Technology Association (TTA) in September 1999 and later recognized as an ISO/IEC International Block Cipher Algorithm IETF standard in 2005, features a Feistel structure[6]. It includes a substitution layer using two distinct S-boxes, a permutation layer, and an AddRoundKey layer. To optimize the substitution layer, we examined an 8-PT configuration, allowing eight 128-bit blocks to be encrypted simultaneously, significantly enhancing performance

# ARMv8 Architecture

The ARMv8 architecture, announced in October 201, is ARM's first 64-bit embedded architecture. It supports both the 32-bit mode (AArch32) and the new 64-bit mode (AArch64) with an A64 instruction set, improving performance and scalability[7]. It can be used with the existing 32-bit instruction set (A32/T32), and provides a larger address space and improved computational performance. There are 31 general-purpose registers, where x0-x30 are used for 64-bit operations and w0-w30 are used for 32-bit operations. ARMv8 also includes 32 128-bit vector registers (v0-v31). This register structure greatly improves data processing efficiency and allows complex operations to be performed more quickly.

In addition, the ARMv8 architecture strengthens memory safety by adding the Pointer Authentication (PA) function. This function improves system security by increasing resistance to memory vulnerabilities. For these various reasons, this architecture has had a significant impact on the smartphone industry and has also been widely adopted in various laptops and other devices.

Table 1 provides the instruction set of the ARM processor used for the parallel implementation of the SEED block cipher. In Table1, Xn represents the source scalar register, Vd represents the destination vector register, Vt represents the transferred vector register, and Vn and Vm represent the source vector registers[8].

1. Instruction set for optimized parallel implementation SEED block cipher

|  |  |  |  |
| --- | --- | --- | --- |
| **asm** | **Operands** | **Description** | **Operation** |
| EOR | Vd, Vn, Vm | Bitwise exclusive OR | Vd ← Vn ⊕ Vm |
| SUB | Vd, Vn, Vm | Subtract | Vd ← Vn - Vm |
| LD4 | Vt, (Xn) | Load multiple single-element structures | Vt ← (Xn) |
| MOVI | Vt, #imm | Move immediate | Vt ← #imm |
| TBL | Vd, Vn, Vm | Table vector lookup | Vd ← Vn[Vm] |
| TBX | Vd, Vn, Vm | Table vector lookup extension | Vd ← Vn[Vm] |
| TRN1 | Vd, Vn, Vm | Transpose vectors (primary) | Vd ← Vn[Vm] |
| TRN2 | Vd, Vn, Vm | Transpose vectors (secondary) | Vd ← Vn[Vm] |

# SEED Algorithm

The SEED block cipher, developed by the Korea Internet Security Agency(KISA) in February 1999, is a block encryption algorithm developed to protect information that may have a sensitive impact when disclosed in the private sector, such as the Internet, e-commerce, and wireless communications, and to protect personal privacy.

The overall structure of the SEED algorithm consists of a Feistel structure, and uses a 128-bit plaintext block and a 128-bit key as input, and outputs a 128-bit ciphertext block after a total of 16 rounds. The SEED block cipher uses the KEY foe each round to perform an F function operation. Afterwards, an XOR operation is performed, and the 64-bit on the left and right are swapped due to the Feistel structure. The following Fig. 1. Schematizes the overall structure of the SEED algorithm.

1. The overall structure of the SEED algorithm

## F Function

Block cipher algorithms with a Feistel structure can be classified according to the characteristics of the F function.  
SEED’s F function consists of a modified 64-bit Feistel form. The F function receives two 32-bit blocks (C, D) as input and outputs two 32-bit blocks (C’, D’). That is, during the encryption process, the 64-bit block (C, D) and the  
64-bit round key = (; ) are treated as inputs to the F function. Thus, 64-bit blocks (C’, D’) are output. (: number of round)

Fig. 2. Schematizes the overall structure diagram of the F function.

## G Function

The G function is an operation inside the F function and is described as follows:

도표, 스케치, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명

In this G function, an operation is performed through two different S-boxes. Afterwards, the result value is obtained by permutation and XOR of the operation values. Fig. 3. Schematizes the overall structure diagram of the G function.

도표, 라인, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명

1. The overall structure diagram of the F function

도표, 스케치, 그림, 라인이(가) 표시된 사진

자동 생성된 설명

1. The overall structure diagram of the G function

## S-Box

In the SEED block cipher, non-linear S-boxes are used, and the nonlinear S-boxes S1 and S2 used inside the G function are generated using the following formulas.(=247, =251,=159,=56)

# Implementation of SEED on ARM

Instructions used for optimized implementation are described as follows:

* TRN and UZP instruction: To facilitate parallel operations, the indices of each block need to be aligned using the TRN and UZP instructions. The TRN instruction selects elements alternately from two source vector registers, combining even-indexed elements from both sources into one result vector and odd-indexed elements into another, effectively ”transposing” the vectors. TRN1 generates the first result vector, while TRN2 generates the second. The UZP instruction also selects elements alternately from two source vector registers. UZP1 combines even-indexed elements from the first source and odd-indexed elements from the second source into one result vector, while UZP2 generates the second result vector.
* TBL instruction: The TBL instruction creates a new vector by selecting elements from a table according to specified indices. The TBL instruction selects  
  an element from a table using the index of another vector. This command is very useful for shuffling or reordering vector data. Substitution and permutation can be implemented efficiently by using the TBL instruction.
* LD4 instruction: ARMv8 supports various load instructions, among which the LD4 instruction is used for the parallel implementation. This is mainly due to data processing efficiency and memory access optimization. The LD1 instruction loads one data element, while the LD4 instruction loads four data elements at a time. However, for parallel implementation, the input values ​​must be aligned in registers.

In order to proceed with parallel implementation using the TBL instruction, the indexes required for the operation must exist in the same register. In the case of the SEED block cipher, two different S-boxes S1 and S2 are used. Therefore, only the indexes using S1 are aligned to one register, and the indexes using S2 are aligned to another register. 128-bit can be used in one register, but since the operation is performed 16-bit at a time, the parallel operation is performed in 8 blocks.

In the 8-PT parallel implementation, the input values ​​are loaded using the LD4 instruction, and eight plaintext blocks are loaded into the registers. The loaded values ​​appear as in Fig. 4(a). When first loaded, indices using different S-boxes exist in the same register. In order to perform S-box operation using the TBL instruction, indices using the same S-box must be aligned to the same register. Therefore, TRN1, TRN2, UZP1, and UZP2 are used to align them as in Fig. 4(b). The algorithm for sorting is as shown in Algorithm1. Each register holds two indices per block using the same S-box. This arrangement optimizes the parallel processing capability by enabling parallel computation using a different S-box for each register.

The SEED block cipher uses a Feistel structure to perform 64-bit calculations. And the internal F function performs 32-bit calculations. Accordingly, the S-box calculations inside the G function originally perform 32-bit calculations. However, this paper performs 8-block parallel calculations, so it performs 256-bit calculations. It performs calculations using two adjacent registers that use different S-boxes.

텍스트, 스크린샷, 사각형, 직사각형이(가) 표시된 사진

자동 생성된 설명

1. Two types of state for 8-PT parallel implemenataion. (a) 8-PT state Type 1. (b) 8-PT state Type 2.

|  |
| --- |
| **Algorithm1** : 8-PT alignment from Type1 to Type2 |
| // v0 – v7 : PT  // v28 – v31 : temp   1. trn1.16b v28, v0, v2 2. trn1.16b v29, v1, v3 3. trn2.16b v30, v0, v2 4. trn2.16b v31, v1, v3 5. uzp1.8h v0, v28, v28 6. uzp1.8h v1, v29, v29 7. uzp1.8h v2, v30, v30 8. uzp1.8h v3, v31, v31 9. uzp2.8h v4, v28, v28 10. uzp2.8h v5, v29, v29 11. uzp2.8h v6, v30, v30 12. uzp2.8h v7, v31, v31 |

S-box operations can be easily and efficiently performed using TBL and TBX instructions. The TBL and TBX instructions perform table vector lookup. TBL instruction first read the vector value stored in the Vn register and use that value as the index of the Vm register. After that, the value stored in the corresponding index of Vm is stored in Vd. Since the S-box is stored in Vm, efficient operations are possible if the corresponding instructions are used.

Since the S-box index of the SEED cipher is 256, it is stored in the v16-v31 register. After that, the S-box operation is performed through the TBL instruction starting from v16. The operation is as shown in Algorithm2.

|  |
| --- |
| **Algorithm2** : S-box operation using TBL instruction |
| // v16 – v31 : S-box  // v15 : 0x40  // v12 – v14 : temp  .macro sbox reg, address   1. sub.16b v14, \reg, v15 2. tbl.16b \reg, {v16-v19}, \reg 3. sub.16b v13, v14, v15 4. tbx.16b \reg, {v20-v23}, v14 5. sub.16b v12, v13, v15 6. tbx.16b \reg, {v24-v27}, v13 7. tbx.16b \reg, {v28-v31}, v12   .endm |

After performing the S-box operation, the Permutation operation is performed. First, , , , and must be used, but since continuously loading and using them in the register causes performance degradation, they are stored in four registers and used. Since plaintext register alignment was performed for parallel operation, the registers to be used for - must also be aligned. To match the plaintext register, v8 should contain , values, v9 should contain , values, v10 should contain , values, and v11 should contain , values. Put - in each empty register and align them using the TRN instruction to facilitate subsequent calculations. The operation is as shown in Algorithm3.

|  |
| --- |
| **Algorithm3** : align - |
| // v8 – v11 : -   1. trn1.16b v12, v8, v10 2. mov v8.16b, v12.16b 3. trn1.16b v12, v9, v11 4. mov v9.16b, v12.16b 5. trn1.16b v12, v10, v8 6. mov v10.16b, v12.16b 7. trn1.16b v12, v11, v9 8. mov v11.16b, v12.16b |

# Results and Evaluation

This section presents a performance evaluation of the SEED block cipher on the ARMv8 architecture. The evaluation focused solely on encryption, excluding the  
key generation process. Tests were conducted on a 2022 MacBook Air equipped with the Apple M2 chip, an ARMv8 processor. The optimization option was set to -O3. A comparative analysis was performed against the reference implementation, and the performance of the reference implementation was measured in the same environment. Measured against clock cycles/10000 after running the  
encryption function 10,000 times. The performance results are shown in Table2.

1. Performance comparison of implementation of the SEED block cipher

|  |  |  |  |
| --- | --- | --- | --- |
| **Imple.** | **Target** | **Parallel** | **CPB** |
| Reference C | 64-bit ARMv8 Apple M2 | 1-PT | 1096 |
| This work | 64-bit ARMv8 Apple M2 | 8-PT | 1546 |

The result of Reference C was 1096 cycles, and the result of this study was 1546 cycles. If you only look at the cycles, the results of the study seem insufficient, but 8-PT calculations are performed due to parallel implementation.  
This shows a performance improvement of approximately 5.66x compared to Reference C.

# Conclusion

In this paper, we present the parallel implementation of the SEED block cipher on ARMv8 architectures. By utilizing the TBL and LD4 instructions, we achieve parallel encryption of 8 plaintext blocks simultaneously. This method can also  
be applied to other block ciphers, facilitating their parallel implementation. We anticipate that this approach will enhance the performance of other block ciphers when adopted.

# ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services, 50%) and this work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2022-0-00627, Development of Lightweight BIoT technology for Highly Constrained Devices, 50%).

##### References

1. Song, Jingyo, and Seog Chung Seo. "Secure and fast implementation of ARX-Based block ciphers using ASIMD instructions in ARMv8 platforms." *IEEE Access* 8 (2020): 193138-193153.
2. Fujii, Hayato, Félix Carvalho Rodrigues, and Julio López. "Fast AES implementation using ARMv8 ASIMD without cryptography extension." *Information Security and Cryptology–ICISC 2019: 22nd International Conference, Seoul, South Korea, December 4–6, 2019, Revised Selected Papers 22*. Springer International Publishing, 2020.
3. Eum, Siwoo, et al. "Parallel Implementations of ARIA on ARM Processors and Graphics Processing Unit." *Applied Sciences* 12.23 (2022): 12246.
4. Yi, Jaeyoung, et al. "Fully pipelined hardware implementation of 128-bit SEED block cipher algorithm." *Reconfigurable Computing: Architectures, Tools and Applications: 5th International Workshop, ARC 2009, Karlsruhe, Germany, March 16-18, 2009. Proceedings 5*. Springer Berlin Heidelberg, 2009.
5. Seo, Young-Ho, Jong-Hyeon Kim, and Dong-Wook Kim. "Hardware implementation of 128-bit symmetric cipher SEED." *Proceedings of Second IEEE Asia Pacific Conference on ASICs. AP-ASIC 2000 (Cat. No. 00EX434)*. IEEE, 2000.
6. Lee, H. J., et al. *The SEED encryption algorithm*. No. rfc4269. 2005..
7. Wang, K. C. "ARMv8 Architecture and Programming." *Embedded and Real-Time Operating Systems*. Cham: Springer International Publishing, 2023. 505-792.
8. ARMv8-A Instruction Set Architecture. Available online: https://documentation-service.arm.com/static/613a2c38674a052ae3 6ca307 (accessed on 26 June 2019