Parallel Implementation of Simeck Family Block Cipher by using ARM NEON

Taehwan Park
School of Electrical and Computer Engineering
Pusan National University
Busan, Republic of Korea
pth5804@pusan.ac.kr

Hwajeong Seo
IT Computer System
Hansung University
Seoul, Republic of Korea
hwajeong@hansung.ac.kr

Chanhui Park, Howon Kim*
School of Electrical and Computer Engineering
Pusan National University
Busan, Republic of Korea
{chan70921, howonkim}@pusan.ac.kr

***Abstract*—In these days, it takes a long time, a lot of operations and memory to implement the existing block ciphers based on SPN architecture such as AES, and etc. on the resource-constrained IoT device environment. By this reason, there are a lot of light-weight block ciphers such as SIMON, SPECK, and Simeck support various block/key sizes. In this paper, authors propose efficient implementation methods and results of the Simeck family block cipher on the 32-bit ARM by using ARM NEON SIMD and SIMT. Proposed methods on Simeck32/64 and Simeck64/128 are 8.9 cycles/byte and 12.0 cycles/byte. Proposed methods are 41.4% and 46.9% faster than previous related work respectively.**

I. Introduction

In these days, there are a lot of the Internet of Things (IoT) devices according to development of IoT technologies. These IoT devices have resource-constrained environments such as low Computing ability (working frequency), RAM, and ROM size. For this reason, implementing the existing block ciphers based on SPN (Substitution and Permutation Network) such as AES, PRESENT, and etc on IoT device needs a lot of memory and operations. For solving this problem, many cryptographists and mathematician researched and proposed the light-weight cryptography such as SIMON, SPECK [1], Simeck [2] and so on. Light-weight block ciphers have ARX (Addition, Rotation, and eXclusive-OR) operations for encryption, decryption, and key expansion. These support various block and key sizes such as 32-bit, 64-bit size data block, and key data so, it can have light-weight characteristic for efficient implementation on IoT device. The Simeck family block cipher [2] was proposed in CHES 2015 and it is suitable for RFID systems succeeded the architecture of SIMON and SPECK [1]. Previous works on Simeck focused on cryptanalysis and implementing on 8-bit AVR, 16-bit MSP430 IoT device environment and ARM Cortex-A series support ARM NEON engine for multimedia data SIMD (Single Instruction Multiple Data) operations and all most of the smartphone and some IoT devices or IoT gateway have ARM Cortex series CPU. For these reasons, we proposed the parallel implementation methods and performance results of the Simeck family block cipher on 32-bit ARM IoT device (Cortex-A53, ARMv8) by using ARM NEON SIMD and OpenMP SIMT. Simeck family block cipher implementations with proposed methods can be used at various IoT applications for providing data encryption/decryption and security.

The remainder of this paper is organized as follows: the Simeck family block cipher, related works on implementing cryptographic algorithms on 32-bit ARM environment by using ARM NEON SIMD, and OpenMP SIMT are described in section 2, the proposed methods of efficient implementing Simeck family block ciphers on 32-bit ARM by using ARM NEON and OpenMP SIMT at section 3, experiment and evaluation are described in section 4 and the final conclusion of this paper is described in section 5.

II. Background and Related works

In this subsection, we describe the Simeck family block cipher algorithm and previous research works related to the Simeck block cipher implementation, implementation of cryptographic algorithms on the 32-bit ARM by using ARM NEON SIMD and OpenMP SIMT.

A. Simeck Family Block Cipher

The Simeck family block cipher was proposed in CHES 2015 [2] and it is suitable for hardware environment and RFID systems and succeeded the architecture of SIMON and SPECK [1]. The Simeck family block cipher have ARX(Addition/AND ($\odot$), Rotation (Rotation Left, $\lll$), eXclusive-OR ($\oplus$)) operations. There are 3 types of Simeck block cipher according to supporting block/key size such as Simeck32/64, SImeck48/96, and Simeck64/128.

Fig. 1 shows the encryption round function of Simeck block cipher at *i*-th round. In Fig. 1, l_i denotes left word, r_i denotes right word, and k_i denotes i-th round key. The i-th round function of Simeck block cipher can be represented as the following equation.

R_{k_i} (l_i,r_i)=($r_i \oplus$f(l_i)$\oplus k_i$,f(x) = (x$\odot$ *ROL1(x)*) $\oplus$ *ROL5(x)*.

In the equation for Simeck encryption round function, the *ROLr()* function means *r*-bit left rotation operation.

Fig. 2 shows key schedule of Simeck block cipher. Key schedule function of Simeck block cipher generates round keys by using master key K consists of the initial state(t_2, t_1, t_0, k_0) as four words. There are 2 types of initial state such as (1, 1, 1, 1) and (1, 1, 1, 1, 1, 1). Simeck32/64 and Simeck48/96 use the first initial state. In the case of Simeck64/128 uses the

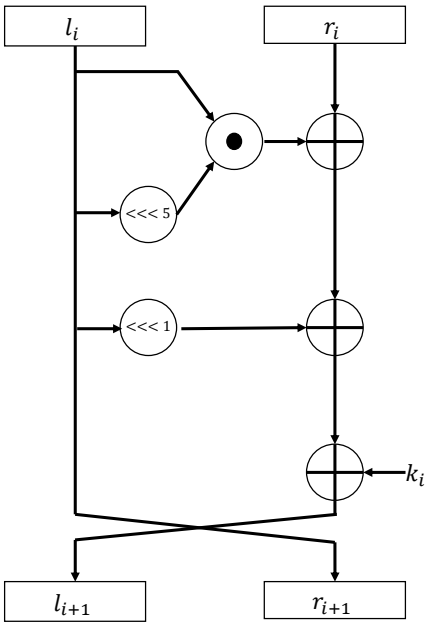

Fig. 1. Simeck Family Block Cipher Encryption Round Function

second initial state. The constant value (C) in key schedule can be written as 2^n - 4 = 0xFF ... FC. The (Z_{j_i} in Fig. 2 means the *i*-bit of sequence Z. There are 2 types of sequences such as Z_0 and Z_1. The Z_0 has 31 periods, it can be generated by using the primitive polynomial x^5+x^2+1, and the Z_1 has 63 periods, it can be generated by using the primitive polynomial x^6+x+1. Simeck32/64, and Simeck48/96 use the Z_0 sequence and Simeck64/128 use the Z_1 sequence for key schedule. The $R_{C\oplus(x_j)_i}$ in Fig. 2 means Round key at each round.

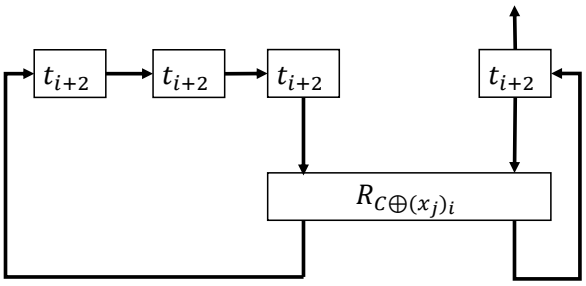

Fig. 2. Simeck Family Block Cipher Key Expansion

Table. I describes Simeck family block cipher specification such as block/key size and the number of rounds. From Simeck32/64 to Simeck64/128 has 32, 36, and 44 rounds respectively.

There are 2 types of research results on the Simeck family block ciphers. First one is cryptanalysis on Simeck. Stefan method [3] gave a brief comparison of SIMON and Simeck block cipher according to changing architecture from SIMON to Simeck. Nasour method [4] proposed linear cryptanalysis method and results on round-reduced Simeck family block cipher. Zhang et al. [5] proposed zero correlation linear cryptanalysis result on Simeck and security evaluation. Qiao et al. [6] proposed the differential security evaluation by using dynamic key-guessing techniques on the Simeck. The last one is efficient implementations of Simeck family block ciphers on IoT embedded board such as 8-bit AVR, and 16-bit MSP430. Park et al. [7] proposed efficient implementation results of the Simeck on 8-bit ATmega128 environment Efficient implementation of Simeck on the 16-bit MSP430 environment in ICUFN 2017 [8].

TABLE I
SIMECK FAMILY BLOCK CIPHER

Block cipher	Block size(bit)	Key size(bit)	Round(T)
Simeck32/64	32	64	32
Simeck48/96	48	96	36
Simeck64/128	64	128	44

B. ARM NEON SIMD

ARM Cortex-A series supports Advanced SIMD (aka *NEON*) NEON supports 64-bit, and 128-bit SIMD instruction sets and registers such as 32 D registers (64-bit size, D0-D15) and 16 Q registers (128-bit size, Q0-Q31). 128-bit size Q register can be written as 8-bit×16, 16-bit×8, 32-bit×4 data. NEON also supports intrinsic function like Intel AVX2 SIMD, so the user can easily implement their own source code. One of the characteristics of NEON is that it can operate independently so, it does not receive any operation control from ARM CPU during NEON operation.

C. OpenMP SIMT

OpenMP (Open Multi-Processing) is API for supporting multi-platform shared memory multiprocessing in C, C++, and Fortran. GCC compiler supports OpenMP. One of the strong points of OpenMP is simple and portable, so it can be used on anywhere easily. If the user uses OpenMP then it can accelerate performance.

D. Cryptographic Algorithm implementation on ARM using ARM NEON

ARM NEON SIMD techniques are used on the parallel implementation of cryptographic algorithms and there are many implementation results by using ARM NEON. Park et al. [17] proposed the parallel implementation of SIMON and SPECK by using ARM NEON intrinsics and SIMT. Seo et al. [9] proposed the parallel implementation of LEA block cipher by using ARM assembly and ARM NEON instruction. Seo et al. [10] proposed compact LEA block cipher GCM mode implementation methods and optimized binary field multiplication for GCM by using ARM NEON on the ARM Cortex-A9 platform. Seo et al. [11] proposed high-performance implementation of SGCM(Sophie German Counter Mode) of ARM environment by using ARM NEON. In AEAD(Authenticated Encryption and Authentication Decryption), Dos santos et al. [12] proposed pipeline oriented implementation of NORX AE

scheme on ARM processor by using ARM NEON. In Post-Quantum Cryptography area, Zhe Liu et al. [13] proposed the efficient implementation of Ring-LWE encryption on ARM using ARM NEON and Streit Silvan method [14] proposed efficient New Hope lattice-based key exchange scheme on ARMv8-A environment using NEON. Jalali Amir et al. [15] proposed efficient supersingular isogeny Diffie-Hellman key exchange protocol on 64-bit ARM using NEON.

III. Proposed Methods

In this section, we proposed the efficient parallel implementation methods of the Simeck family block cipher on the ARM Cortex-A series environment by using ARM NEON SIMD and SIMT. At the first, we proposed ARM NEON optimization by using ARM NEON intrinsic functions. Second, we proposed speed optimization by considering NEON pipeline and using OpenMP SIMT.

A. ARM NEON Optimization

ARM NEON supports intrinsic functions for operation. There are 2 types of operations for implementing cryptographic algorithms by using ARM NEON. First one is about data store, load, set. Table. II describes ARM NEON intrinsic functions for data load, store, and set. For efficient implementation, we used *uint16x8_t* data type for Simeck32/64 and *uint32x4_t* data type for Simeck64/128. In the case of NEON set intrinsic function, we used it for setting Q register values as Simeck encryption round key at each round. We assume that data blocks for encryption by using NEON are already aligned.

TABLE II
ARM NEON Intrinsic functions for Data Store, Load, and Set

Operation	NEON Intrinsic Function
Load	*uint16x8_t vld1q_u16 (__transfersize(8) uint16_t const * ptr)*
	*uint32x4_t vld1q_u32 (__transfersize(4) uint32_t const * ptr)*
Store	*void vst1q_lane_u16 (__transfersize(1) uint16_t * ptr, uint16x8_t val, __constrange(0,7) int lane)*
	*void vst1q_lane_u32 (__transfersize(1) uint32_t * ptr, uint32x4_t val, __constrange(0,3) int lane)*
Set	uint16x8_t vdupq_n_u16 (uint16_t value)
	uint32x4_t,vdupq_n_u32(uint32_t value)

The second one is Data bitwise AND, bitwise XOR (eXclusive-OR), shift, and rotation operations for implementing Simeck family block ciphers. Table. III describes ARM NEON intrinsic functions for bitwise AND, bitwise XOR (eXclusive-OR), shift, and rotation operations. In the case of rotation operation, we used shift left intrinsic functions and shift right and insert intrinsic functions such as *vsriq_n_u16* and *vsriq_n_u32* which support insert data after right shifting operation. It can save code size because if there is no shift right and insert intrinsic function and instruction, we have to use the bitwise OR intrinsic function or instruction after left shifting for rotation operation.

TABLE III
ARM NEON Intrinsic functions for ARX Operation

Operation	NEON Intrinsic Function
bitwise AND	*uint16x8_t vandq_u16 (uint16x8_t a,uint16x8_t b)*
	uint32x4_t vandq_u32 (uint32x4_t a, uint32x4_t b)
bitwise XOR	*uint16x8_t veorq_u16 (uint16x8_t a,uint16x8_t b)*
	uint32x4_t veorq_u32 (uint32x4_t a, uint32x4_t b)
Shift Left (SL)	*uint16x8_t vshlq_n_u16 (uint16x8_t a,uint16x8_t b)*
	uint32x4_t vshlq_n_u32 (uint32x4_t a, uint32x4_t b)
Rotation Left r-bit	*vsriq_n_u16(SL(X,r), X, (16-r))*
	vsriq_n_u32(SL(X,r), X, (32-r))

B. Speed Optimization

In this subsection, we describe speed optimization by considering NEON pipeline and using OpenMP SIMT.

1) NEON Pipeline Optimization: If we use NEON instructions or intrinsic functions as SISD (Single Instruction Single Data) such as using result data right before as operand data at next operation, it causes data hazard (stall). If data hazard occurs, it can degrade performance. For avoiding data hazard, implementing do not use result data right before as operand at next operation is important. For this reason, we re-designed order of operation at each NEON register for efficiency and high-performance.

2) OpenMP SIMT Optimization: For speed optimization by using OpenMP SIMT, we used *omp_set_num_threads*(♯) OpenMP function for setting the number of threads. In that function, ♯ means the number of threads. For OpenMP SIMT(Single Instruction Multiple Thread) optimized implementation, we used the macro(♯*pragma omp parallel for*).

IV. Experiment and Evaluation

In this section, we describe experimental environment including 32-bit ARM target board and evaluation of performance by comparing Park et al. [17] and the proposed methods. Especially, Simeck block cipher has similar architecture with SIMON block cipher so, we compared performance between proposed methods and SIMON in Park et al. [17].

A. Experimental Setup

For the experiment, we used development environment such as Table IV. experiment environment is same as Park et al. [17]. We use GCC compiler (GCC v4.9.2) for developing proposed method and compile with some options(-*mcpu=cortex-a53 -mfloat-abi=hard -mfpu=neon-fp-armv8 -mneon-for-64bits -O3 -funroll-loops -fopen*) for efficient implementation. Compile options like *-mcpu=cortex-a53 -mfloat-abi=hard -mfpu=neon-fp-armv8 -mneon-for-64bits* are for using ARM NEON intrinsics or assembly, *-mcpu* compile option defines the CPU type and architecture. Our experiment target board has ARM Cortex-A53 so, we set as *cortex-a53*.*-mfloat-abi, -mfpu* compile option defines the floating-point application

binary interface type and floating-point unit type. we used the floating-point application as hard (generating floating-point instructions with FPU-specific calling conventions) and floating-point type as neon-fp-armv8 because ARM Cortex-A53 is the ARMv8 architecture. *-mneon-for-64bits* compile option lets NEON handle scalar 64-bits operations. *-fopen* compile option is for compiling with OpenMP library. *-O3* compile option optimized more for code size and execution time. *-funroll-loops* compile option determines and unroll loops whose number of iterations at compile time.

TABLE IV
DEVELOPMENT ENVIRONMENT

Operating System(OS)	Raspbian GNU/LINUX 8
CPU	Quad Core Broadcom BCM2837 (Cortex A-53, ARMv8)
RAM	1GB
Development tool	GCC v4.9.2

In this experiment, we use target device as RaspberryPi3 Model B development board. It has 32-bit ARM Cortex-A53 (ARMv8) CPU, 1.2GHz frequency, and 1.0GB RAM such as Table V.

TABLE V
RASPBERRYPI3 MODEL B SPECIFICATION

CPU	Quad Core Broadcom BCM2837 (Cortex A-53, ARMv8)
Frequency	1.2GHz
RAM	1.0GB

B. Evaluation

In this section, we describe the evaluation of performance by comparing the proposed methods and Park et al. [17] on the 32-bit ARM Cortex-A53 target board. Park et al. [17] used ARM NEON assembly language at the same target board environment. We evaluated performances as average cycles/byte on 100,000 times encryption. Table VI and Fig. 3 describe performance comparison results.

Proposed Simeck32/64 has 8.9 cycles/byte and it is 41.4% faster than SIMON32/64 of Park et al. [17]. In the case of proposed Simeck64/128, it has 12.0 cycles/byte and is 41.4% faster than SIMON64/128 of Park et al. [17].

Fig. 5-A describes proposed Simeck32/64 performance by using OpenMP SIMT and Fig. 5-B describes proposed Simeck64/128 performance by using OpenMP SIMT. We calculated performance by increasing the number of threads. if the number of threads is 4 or 5, at that time, it has good

TABLE VI
PERFORMANCE EVALUATION

Methods	**Cycles/Byte**
SIMON32/64 [17]	15.2
Proposed Simeck32/64	**8.9**
SIMON64/128 [17]	22.6
Proposed Simeck64/128	**12.0**

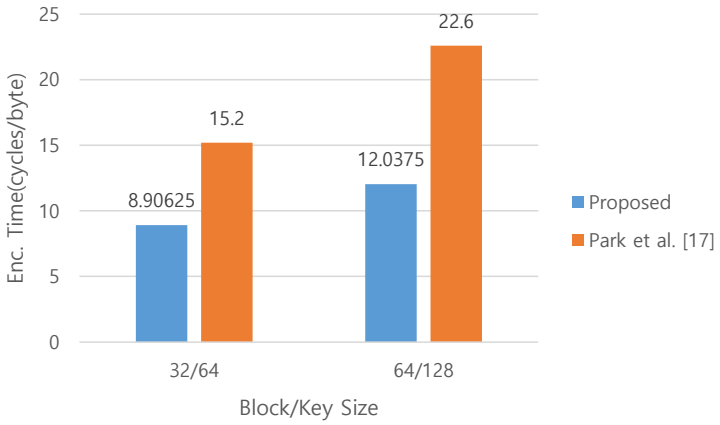

Fig. 3. Performance Comparion Results

performance because target board has the Quad-core CPU. We compiled proposed method with some optimized option for high-performance so, it affects the performance of proposed OpenMP SIMT optimized implementation.

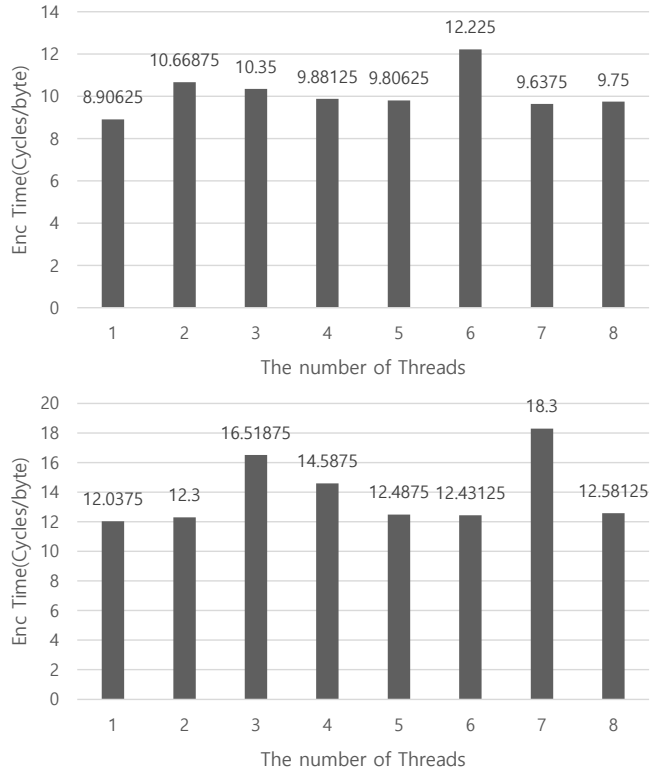

Fig. 4. Proposed Method(SIMT) Performance(upper: Proposed Simeck32/64, bottom: Proposed Simeck64/128, measurement unit: cycles/byte)

V. CONCLUSION

In this paper, we proposed the efficient implementation methods and performance results of Simeck family block cipher on ARM environment by using ARM NEON and

OpenMP SIMT. For optimization, we used ARM NEON intrinsic functions for SIMD, optimized NEON pipeline and used OpenMP SIMT techniques. Proposed Simeck32/64 has 8.9 cycles/byte and it is 41.4% faster than SIMON32/64 of Park et al. [17]. In the case of proposed Simeck64/128, it has 12.0 cycles/byte and is 41.4% faster than SIMON64/128 of Park et al. [17]. In the case of OpenMP SIMT optimization, if the number of threads is 4 or 5, at that time, it has better performance according to the characteristic of the quad-core environment and it is affected by GCC optimized compile options for high-performance. Our proposed method implementation source code can be found at the github[1]. In the future, we research on efficient implementation of Simeck on GPGPU environment by using CUDA.

ACKNOWLEDGMENT

This material of Taehwan Park, Chanhui Park, and Howon Kim is based upon work supported by the Ministry of Trade, Industry & Energy(MOTIE, Korea) under Industrial Technology Innovation Program(No.10073236).

This work of hwajeong seo was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1C1B5075742).

REFERENCES

[1] 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*. IEEE, 2015, pp. 1–6.

[2] G. Yang, B. Zhu, V. Suder, M. D. Aagaard, and G. Gong, "The simeck family of lightweight block ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 307–329.

[3] S. Kölbl and A. Roy, "A brief comparison of simon and simeck." *IACR Cryptology ePrint Archive*, vol. 2015, p. 706, 2015.

[4] N. Bagheri, "Linear cryptanalysis of reduced-round simeck variants," in *International Conference in Cryptology in India*. Springer, 2015, pp. 140–152.

[5] K. Zhang, J. Guan, B. Hu, and D. Lin, "Security evaluation on simeck against zero correlation linear cryptanalysis." *IACR Cryptology ePrint Archive*, vol. 2015, p. 911, 2015.

[6] K. Qiao, L. Hu, and S. Sun, "Differential security evaluation of simeck with dynamic key-guessing techniques." *IACR Cryptology ePrint Archive*, vol. 2015, p. 902, 2015.

[7] T. Park, H. Seo, B. Bae, and H. Kim, "Efficient implementation of simeck family block cipher on 8-bit processor," *Journal of information and communication convergence engineering*, vol. 14, no. 3, pp. 177–183, 2016.

[8] T. Park, H. Seo, G. Lee, and H. Kim, "Efficient implementation of simeck family block cipher on 16-bit msp430," in *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*. IEEE, 2017, pp. 983–988.

[9] 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*. Springer, 2016, pp. 318–330.

[10] H. Seo, G. Lee, T. Park, and H. Kim, "Compact gcm implementations on 32-bit armv7-a processors," in *Information and Communication Technology Convergence (ICTC), 2017 International Conference on*. IEEE, 2017, pp. 704–707.

[11] H. Seo, "High performance implementation of sgcm on high-end iot devices," *Journal of information and communication convergence engineering*, vol. 15, no. 4, pp. 212–216, 2017.

[1]https://github.com/pth5804/Simeck_ARM_NEON

[12] L. C. dos Santos, J. López, and C. U. Z. Vaz, "Pipeline oriented implementation of norx for arm processors," 2017.

[13] Z. Liu, R. Azarderakhsh, H. Kim, and H. Seo, "Efficient software implementation of ring-lwe encryption on iot processors," *IEEE Transactions on Computers*, 2017.

[14] S. Streit and F. De Santis, "Post-quantum key exchange on armv8-a: A new hope for neon made simple," *IEEE Transactions on Computers*, 2017.

[15] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Supersingular isogeny diffie-hellman key exchange on 64-bit arm," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[16] (2015) The implementations(in c and python) of the simeck family of block ciphers. [Online]. Available: https://github.com/bozhu/Simeck

[17] T. Park, H. Seo, G. Lee, K. Md. Al-Amin, Y. Nogami, and H. Kim, "Parallel implementations of simon and speck, revisited," in *International Workshop on Information Security Applications*. Springer, 2017, pp. 1–12.