JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 22, No. 8: 1133~1138, Aug. 2018

ARM Cortex-M3 프로세서 상에서의 LEA 암호화 고속 구현

서화정*

High Speed Implementation of LEA on ARM Cortex-M3 processor

Hwa-jeong Seo*

Department of IT Engineering, Hansung University, Seoul 02876, Korea

요 약

경량 블록암호화 (LEA: Lightweight Encryption Algorithm)는 암호화 연산의 효율성과 높은 보안성으로 인해 국내에서 가장 활발히 사용되고 있는 블록암호화 알고리듬이다. 지금까지 많은 LEA 구현 연구가 진행 되었지만 다양한 플랫폼과의 보안 통신이 필요한 사물인터넷 환경에 활용 가능한 일체형 구현 기법은 제시되고 있지 않다. 본 논문에서는 다양한 플랫폼과 효율적으로 보안 통신이 가능하도록 하는 일체형 구현 기법을 이용하여 LEA를 ARM Cortex-M3 프로세서 상에서 구현한다. 이를 위해 키생성과 암호화 과정에 필요한 인자들을 가용 가능한 레지스터를 이용하여 저장하였으며 바렐쉬프터 (Barrel-shifter)를 활용하여 회전 연산을 최적화하였다. 해당 기법은 라운드키를 저장하지 않기 때문에 저사양 프로세서 상에서 RAM의 사용량을 최소화한다. 구현 결과물은 ARM Cortex-M3 프로세서 상에서 평가되었으며 34 cycles/byte 안에 수행가능함을 확인할 수 있었다.

ABSTRACT

Lightweight Encryption Algorithm (LEA) is one of the most promising lightweight block cipher algorithm due to its high efficiency and security level. There are many works on the efficient LEA implementation. However, many works missed the secure application services where the IoT platforms perform secure communications between heterogeneous IoT platforms. In order to establish the secure communication channel between them, the encryption should be performed in the on-the-fly method. In this paper, we present the LEA implementation performing the on-the-fly method over the ARM Cortex-M3 processors. The general purpose registers are fully utilized to retain the required variables for the key scheduling and encryption operations and the rotation operation is optimized away by using the barrel-shifter technique. Since the on-the-fly method does not store the round keys, the RAM requirements are minimized. The implementation is evaluated over the ARM Cortex-M3 processor and it only requires 34 cycles/byte.

키워드 : ARM Cortex-M3, LEA, Barrel-shifter, 일체형

Keywords : ARM Cortex-M3, LEA, Barrel-shifter, on-the-fly

Received 8 May 2018, **Revised** 14 May 2018, **Accepted** 30 May 2018

*** Corresponding Author** Hwa-jeong Seo(E-mail:hwajeong@hansung.ac.kr, Tel:+82-2-760-8033)
Department of IT Engineering, Hansung University, Seoul 02876, Korea

Open Access http://doi.org/10.6109/jkiice.2018.22.8.1133 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

경량 암호화 알고리듬 (Lightweight Encryption Algorithm, LEA)은 국가보안기술연구소에서 개발되어 WISA'13에서 발표되었다 [1]. LEA 알고리듬은 덧셈, 회전 그리고 XOR 연산과 같이 간단한 연산을 기본으로 하는 Addition-Rotation-XOR (ARX) 구조를 따른다. 해당 구조는 기본적인 연산들을 활용하여 구현가능하기 때문에 소프트웨어와 하드웨어 구현 시 매우 효율적이다. 특히 3개의 연산 중 회전 연산의 경우 소프트웨어 상에서는 바렐쉬프터 (Barrel shifter) 그리고 하드웨어 상에서는 wiring을 통해 간단하게 구현 가능한 특징을 가진다. 지금까지 많은 LEA 최적화 구현 논문이 발표되었다. ICISC'13에서는 ARMv7-A 프로세서 상에서 LEA 암호화를 구현한 결과가 소개되었다 [2]. 특히 Single Instruction Multiple Data (SIMD) 연산자인 NEON을 이용하여 LEA 연산을 task-parallelism관점에서 구현하는 방안이 제안되었다. 2014년도에는 웹 프로그래밍 언어인 자바스크립트 상에서 LEA를 구현한 결과가 소개되었다 [3]. 해당 논문에서는 웹브라우저 환경 상에서도 LEA를 통해 효율적으로 암호화 통신이 가능함을 제시하였다. WISA'15에서는 저전력으로 동작하는 경량 사물인터넷 디바이스 상에서의 최적화 LEA 구현결과가 소개되었다 [4]. WISA'16에서는 ARMv7-A 프로세서 상에서 ARM과 NEON 명령어 셋을 혼용하여 연산하는 방안이 제시되었다 [5]. 이를 기반으로 하여 동시에 다수의 암호화 연산 수행이 가능하도록 하였다. 가장 최근에는 64 비트 ARMv8-A 프로세서 상에서 LEA 구현한 결과가 제시되었다 [6].

하지만 기존의 연구는 하나의 키에 대한 암호화 연산 최적화 구현 결과만을 제시하였다. 이는 기존의 네트워크 환경과 같이 하나의 컴퓨터와 다량의 정보를 교환하는 경우에는 효과적이다. 그러나 사물인터넷 환경과 같이 다수의 센서들로부터 소량의 정보가 빠르게 유통되는 경우에는 비효율적인 특징을 가진다. 그 이유는 키생성을 통해 라운드 키를 생성하는 경우에는 라운드 키를 저장하기 위한 RAM의 소모가 늘어나게 되며 키생성 연산도 따로 구현하게 될 경우 부하가 추가적으로 발생하게 된다. 본 논문에서는 일체형 (on-the-fly) 구현 기법을 통해 키 생성과 암호화 연산을 동시에 구현하도록 하며 이를 통해 RAM 사용량과 다수의 센서와의 통신 시간을 최적화하였다.

본 논문의 구성은 다음과 같다. 2장에서는 경량 암호화 알고리듬 LEA와 타겟 프로세서인 ARM Cortex-M3에 대해 알아본다. 3장에서는 최적화 기법에 대해 확인해 본다. 4장에서는 본 논문에서 제안한 기법을 통해 얻어낸 성능 향상에 대해 확인해 본다. 5장에서는 본 논문을 마무리한다.

Ⅱ. 관련 연구

2.1. LEA 대칭키 암호화 알고리듬

LEA 대칭키 암호화 알고리듬은 국가보안기술 연구소에 의해 2013년에 국제학회 WISA에서 발표되었다 [1]. 기존의 AES 대칭키 암호화 알고리듬에서 Substitution Permutation Network (SPN) 구조를 활용하여 설계되었다. 반면에 LEA 대칭키 암호화는 Addition-Rotation-XOR(ARX) 구조를 사용하여 복잡한 S-box 연산 없이 암호화를 수행한다. ARX 구조 상에서는 암호화 연산이 고정된 시간 안에 수행이 가능하여 이전 SPN 구조 상에서 취약했던 Timing 공격에도 강인한 장점을 가진다 [7].

2.2. 타겟 프로세서: ARM Cortex-M3

아두이노 DUE의 메인 프로세서로 활용되고 있는 ARM Cortex-M3 프로세서는 사물인터넷 서비스 구현에 매우 효과적인 특징을 가지고 있다. ARM Cortex-M3 프로세서의 장점으로는 32-비트로 동작하는 연산자, 저전력 프로세싱 그리고 저렴한 가격으로 볼 수 있다. 프로세서는 ARMv7-M 구조를 따르며 3단계 파이프 라인을 지원한다. 프로그램 코드를 최적화하기 위해서 16-비트 단위로 연산자를 구성하는 Thumb-1 명령어 셋을 지원하며 이를 이용하면 동일한 32-비트 ARM 연산자로 동작하는 프로그램 코드를 절반에 가까운 코드로 구현하는 것이 가능하다. 이와 더불어 Thumb-2 명령어 셋도 지원한다. 기존 Thumb-1 명령어 셋의 경우 16-비트의 한계로 인해 작성이 불가능한 코드가 발생하게 된다. 이러한 경우에는 32-비트 연산자인 ARM 명령어를 활용하여 성능과 코드 크기를 조절한 것으로 살펴볼 수 있다. 본 논문에서 사용하는 ARM Cortex-M3 프로세서 상의 명령어 셋인 덧셈, eXclusive-or 그리고 비트-시프트는 표 1과 같다.

Table. 1 Cortex-M3 instruction set summary

Assembler	Description	Cycles
ADD	Addition	1
EOR	XOR	1
ROR	Logical shift right	1

Ⅲ. 제안하는 LEA 구현 기법

LEA 암호화 알고리듬의 기본적인 연산은 32-비트 기반의 덧셈, XOR 그리고 비트-시프트이다. 덧셈과 XOR의 경우 표 1에서와 같이 ADD 그리고 EOR 연산자를 이용하면 1 클록 안에 구현이 가능하다. 회전연산의 경우에도 ROL 연산자와 barrel shifter를 활용하면 매우 효율적으로 구현 가능하다. 따라서 32-비트 프로세서 상에서의 ARX 연산 구현은 다른 프로세서에 비해 효과적임을 확인할 수 있다.

지금까지 많은 LEA 암호 알고리듬 구현 결과물이 제안되었다. 대부분의 결과물은 키스캐줄링 보다는 암호화 연산에 집중되어 연구되었다. 하지만 최근들어 부상하고 있는 사물인터넷 환경에서는 암호화 연산과 더불어 키를 생성하는 연산에 대한 최적화가 함께 고려되어야 한다. 그 이유는 이기종의 사물인터넷 플랫폼과 실시간으로 통신해야 하는 사물인터넷 환경에서는 하나의 고정된 키가 아니라 다수의 키로 각각의 사물에 맞추어 보안 통신해야하는 경우가 늘어나고 있기 때문이다. 만약 다수의 사물과의 통신을 위해 라운드 키를 모두 저장하게 될 경우 저전력 사물인터넷 프로세서의 RAM 저장 공간이 매우 부족해지는 문제가 발생하게 된다. 따라서 실시간으로 새로운 키로 암호화를 수행할 수 있는 방안에 대한 연구가 필요하다. 고정된 키로 계속적인 연산을 수행하는 구현 기법은 키 스캐쥴링과 암호화 부분을 분리하여 연산을 수행하는 것이다. 해당 구현은 한번 키 스캐쥴링을 수행하고 이후에는 키 스캐쥴링된 라운드 키를 활용하여 암호화를 지속적으로 수행하는 방법으로 구현되게 된다. 이와 달리 키 스캐쥴링과 암호화 연산을 한 번에 구현하는 경우에는 라운드 키를 해당 라운드에 생성하게 되고 그 값을 바로 암호화 연산에 활용하는 구현 기법이다. 해당 구현 기법은 마스터 키에 따른 라운드 키를 따로 저장하지 않아도 되는 장점으로 인해 다수의 마스터 키로 보안 통신을 수행하는 경우에는 매우 효율적이라고 할 수 있다. 예를 들어 라운드 키를 저장하는 구현 기법을 활용할 경우 LEA-128 구현은 384 바이트의 라운드 키를 하나의 사물과의 통신을 위해 유지해야 한다. 그림 1에서는 두 구현 기법의 차이가 나타나 있다. 먼저 분리형 구현은 마스터키를 통해 생성된 라운드키가 평문과 함께 암호화되어 암호문을 도출하게 된다. 일체형 구현의 경우에는 마스터 키와 평문을 넣어서 연산하게 될 경우 암호문이 바로 출력되도록 하고 있다.

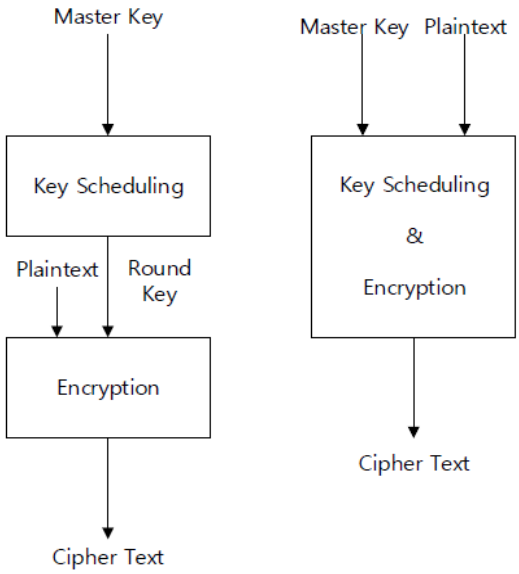

Fig. 1 Separated (left) and interleaved (right) implementations

일체형 구현을 위해 먼저 고려해야 하는 부분은 연산 성능을 높이기 위해 메모리에 대한 접근을 최소화해야 한다는 것이다. 일체형 구현 시 레지스터에는 평문 (4), 델타 (4), 마스터 키 (4), 임시 저장 공간 (1), 그리고 메모리 포인터 (3)가 할당되어야 하며 총 16개이다. 하지만 현재 ARMv6-M 프로세서에서 개발자가 활용 가능한 레지스터는 총 14개이다 (R0-R12, R14). 따라서 필요한 레지스터 공간을 확보하기 위해서 불필요한 레지스터 사용을 줄이도록 하였다. 일체형 구현의 경우 델타와 마스터 키에 대한 메모리 포인터는 델타와 마스터 키의 값이 모두 레지스터에 올라와 있는 경우에는 이를 유지할 필요가 없다. 따라서 두 개의 메모리 포인터 레지스터는 델타와 마스터 키를 메모리에서 레지스터로 불러온 이

후에 다른 용도로 사용되도록 하였다. 이를 통해 필요한 레지스터의 개수인 14개를 모두 확보할 수 있었다.

키 스캐쥴링 연산은 델타 값에 회전연산을 취한 결과값과 키를 더해주고 해당 결과값을 특정 비트 만큼 회전연산을 취해 주는 방식을 취하고 있다. 여기서 두 번의 회전 연산 중 하나의 회전 연산의 경우 barrel shifter를 활용하여 최적화가 가능하다. 두 개의 회전 연산 중 하나의 회전 연산에만 barrel shifter가 적용 가능한 이유는 계산되어 나오게 되는 라운드 키가 암호화 연산에서 회전되어 들어가게 되는데 암호화 연산 평문에 대해서도 barrel shifter가 적용이 되어야 하기 때문이다. barrel shifter의 특성 상 하나의 인자에 대해서만 연산이 가능하기 때문에 두 개의 연산자를 동시에 회전을 적용할 수 없었다.

표 2에는 키 스캐쥴링을 첫 번째 라운드에서 수행되는 연산을 ARM 명령어 셋으로 나타낸 것이다. 회전 연산의 경우 델타에 적용되는 부분에 대해서는 덧셈과 동시에 수행될 수 있도록 barrel shifter를 적용하여 구현하였다. 이를 통해 한 라운드 당 총 4번의 회전 연산을 최적화하여 구현하는 것이 가능하였다. 그 이후에는 계산된 라운드 키에 대하여 각각 회전 연산을 수행하는 과정을 거치고 있다. 앞에서 살펴본 바와 같이 해당 연산은 barrel shifter로 최적화가 불가능한 부분이다. 총 4번에 걸쳐서 2번의 회전연산과 1번의 덧셈이 수행되게 되면 해당 라운드의 라운드 키가 도출되게 된다.

Table. 2 first round of key scheduling in ARM instruction, where R11, R12, R14, R1 registers are key and R7 register is delta variable

```
ADD R11, R11, R7, ROR#0
ROR R11, #31
ADD R12, R12, R7, ROR#31
ROR R12, #29
ADD R14, R14, R7, ROR#30
ROR R14, #26
ADD R1, R1, R7, ROR#29
ROR R1, #21
```

한 라운드 암호화 연산은 덧셈, eXclusive-or 그리고 회전연산으로 구성되게 된다. 여기서 특히 3 번의 회전연산의 경우 barrel-shifter를 활용하게 되면 추가적인 연산 구현없이 연산 가능한 장점을 가진다.

Table. 3 first and second round of encryption in ARM instruction, where R3, R4, R5, R6 are plaintext, R11, R12, R14, R1 registers are key, R7 register is delta variable, and R0 register is temporal register

```
//Round #1
ADD R11, R11, R7, ROR#0
ROR R11, #31
ADD R12, R12, R7, ROR#31
ROR R12, #29
ADD R14, R14, R7, ROR#30
ROR R14, #26
ADD R1, R1, R7, ROR#29
ROR R1, #21
EOR R6, R6, R12
EOR R0, R5, R1
ADD R6, R6, R0
EOR R5, R5, R12
EOR R0, R4, R14
ADD R5, R5, R0
EOR R4, R4, R12
EOR R0, R3, R11
ADD R4, R4, R0

//Round #2
ADD R11, R11, R8, ROR#31
ROR R11, #31
ADD R12, R12, R8, ROR#30
ROR R12, #29
ADD R14, R14, R8, ROR#29
ROR R14, #26
ADD R1, R1, R8, ROR#28
ROR R1, #21
EOR R3, R12, R3
EOR R0, R1, R6, ROR#3
ADD R3, R3, R0
EOR R6, R12, R6, ROR#3
EOR R0, R14, R5, ROR#5
ADD R6, R6, R0
EOR R5, R12, R5, ROR#5
EOR R0, R11, R4, ROR#23
ADD R5, R5, R0
```

표 3에는 첫 번째 라운드와 두 번째 라운드의 구현 코드를 표기하고 있다. 키 생성의 경우 표 2에서와 같이 첫 번째 라운드와 두 번째 라운드가 동일한 방식으로 구현되어 있음을 확인할 수 있다. 하지만 암호화 연산의 경우에는 첫 번째 라운드와 두 번째 라운드에서 서로 상이한 구현 기법을 따르게 된다. 해당 구현 기법은 회전 연

산을 생략하는 구현 기법이다. 따라서 두 번째 라운드부터는 첫 번째 라운드에서 생략한 회전 연산을 적용하면서 EOR 연산을 수행하고 있음을 확인할 수 있다. 해당 barrel shifter 기법은 매 라운드 마다 3번의 회전 연산을 최적화하여 구현하는 장점을 가지고 있다. 라운드 2와 같은 방식으로 나머지 라운드에 대해서도 연산이 반복되게 되며 이를 통해 전체 키스캐쥴링과 암호화 연산을 동시에 수행하게 된다.

이와 더불어 메모리 접근을 보다 효율적으로 하기 위해 LDM 연산자를 활용하여 메모리에 접근하였다. 해당 연산자는 접근해야 하는 메모리 주소를 자동으로 계산해 줌으로써 메모리 접근 속도를 향상 시킨다.

Ⅳ. 성능 평가

본 장에서는 기존에 제안된 기법과 본 논문에서 제안하는 기법의 성능을 비교 분석한다. 성능 분석을 위해 ARM Cortex-M3 프로세서를 탑재하고 있는 Arduino-DUE 개발 보드 상에서 소프트웨어를 Arduino IDE로 개발하였다. 프로그램은 어셈블리언어 처리를 위해 inline assembly 기법을 적용하여 Arduino IDE가 인식 가능한 방향으로 제작하였다. 연산속도를 도출하기 위해 프로세서의 SysTick 사용하였다. 특히 Arduino-DUE 개발 보드는 84MHz로 동작을 시켰으며 정확한 결과 도출을 위해 각각의 연산은 1000번씩 수행하여 결과를 도출하였다.

Table. 4 Comparison of LEA implementations on ARMv6 Cortex-M3 @84MHz (c/b: cycles/byte, MB/s: mega bytes/second)

Method	Speed		RAM (byte)	ROM (byte)
	(c/b)	(MB/s)		
Seo et al. [8]	80	1.05	464	296
This work	34	2.47	228	1,532

표 4에는 구현된 결과와 최신 연구 결과의 성능을 비교하여 나타내고 있다. 연산 속도의 경우 분리형 구현이 80 cycle/byte가 도출되었지만 일체형의 경우 34 cycle/byte가 도출되었다. 이는 기존 연구에 비해 약 57.5%의 성능이 개선된 결과로써 매우 효율적인 구현이 되었음을 확인할 수 있다. 이와 더불어 RAM 사용량은 분리형의 경우 라운드 키를 모두 저장해야 하는 부하가 있기 때문에 464 바이트가 최소한 필요하다. 하지만 일체형 구현의 경우에는 라운드 키를 저장할 필요가 없기 때문에 228 바이트만을 가지고도 구현이 가능하다. 이는 기존 기법에 비해 RAM 사용량이 50.8% 감소하였음을 확인할 수 있다. ROM 사용량의 경우 약 5배가 증가하였음을 확인할 수 있다. 하지만 Arduino-DUE의 경우 512 KB의 ROM이 제공됨을 확인할 수 있다. 1.5KB는 약 0.3%의 ROM을 차지하기 때문에 ROM 사용량의 증가는 큰 부하를 발생시키지 않는다. 따라서 본 결과를 통해 제안된 기법은 사물인터넷 환경 상에서의 이기종 간 암호화에 매우 효율적인 접근 방안임을 확인할 수 있다.

Ⅴ. 결 론

본 논문에서는 경량 블록 암호화 알고리듬 LEA를 최신 저전력 ARM 프로세서인 ARMv6 Cortex-M3 구조 상에서 최적화 구현하고 그 성능을 확인해 보았다. 특히 이전 구현 결과물과는 달리 사물인터넷 환경 상에서의 이기종 간의 안전한 보안 통신을 위해 일체형 암호화 구현 기법을 적용하였으며 이를 최적화 구현하였다. 해당 구현은 최근 구현 결과인 [8]에 비해 약 57.5%의 연산 속도 향상을 기록하였으며 RAM 사용량도 50.8% 감소시켰다. 해당 구현 기법은 유사한 ARX 암호화인 SPECK과 SIMON에 바로 적용가능한 기술로써 그 효용성이 높다 [9]. 본 연구 결과는 초고속 데이터 암호화에도 활용이 가능하다 [10].

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 was partly supported by 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)

References

[1] 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*, Jeju, pp. 3-27, 2013.

[2] 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*, Seoul, pp. 256-274, 2013.

[3] H. Seo and H. Kim, “Low-power encryption algorithm block cipher in JavaScript,” *Journal of information and communication convergence engineering*, vol. 12, no. 4, pp. 252-256, Dec. 2014.

[4] H. Seo, Z. Liu, J. Choi, T. Park, and H. Kim, “Compact implementations of LEA block cipher for low-end microprocessors,” *In Information Security Applications WISA 2015*, Jeju, pp. 28-40, 2015.

[5] H. Seo, T. Park, S. Heo, G. Seo, B. Bae, Z. Hu, L. Zhou, Y. Nogami, Y. Zhu, H. Kim, “Parallel Implementations of LEA, Revisted,” *In Information Security Applications, WISA 2016*, Jeju, pp. 318-330, 2016.

[6] H. Seo, “High Speed Implementation of LEA on ARMv8,” *Journal of the Korea Institute of Information and Communication Engineering*, vol. 21, no. 10, pp. 1929-1934, Oct. 2017.

[7] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, “Fast software AES encryption,” *In Fast Software Encryption FSE 2010*, Seoul, pp. 75-93, 2010.

[8] H. Seo, I. Jeong, J. Lee, W. Kim, “Compact Implementations of ARX-Based Block Ciphers on IoT Processors,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 3, pp. 1-16, Jun. 2018.

[9] T. Park, H. Seo, H, Kim., “Parallel Implementations of SIMON and SPECK,” *IEEE International Conference on Platform Technology and Service*, Jeju, pp. 1-6, 2016.

[10] V. Sujatha, "Auditing of Storage Security on Encryption," *Asia-pacific Journal of Convergent Research Interchange*,vol.3, no. 2, pp. 1-9, Jun. 2017,

서화정(Hwa-jeong Seo)

2010년 2월: 부산대학교 컴퓨터공학과 학사 졸업
2012년 2월: 부산대학교 컴퓨터공학과 석사 졸업
2012년 3월 ~ 2016년 1월: 부산대학교 컴퓨터공학과 박사 졸업
2016년 1월~2017년 3월: 싱가포르 과학기술청
2017년 4월~현재: 한성대학교 IT 융합공학부 조교수
※관심분야 : 정보보호, 암호화 구현, IoT