Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.3, Jun. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.3.623

저사양 마이크로 컨트롤러에서 ARX 경량 암호를 위한 효율적인 Rotation 구현 방법 연구*

김 민 우,† 권 태 경‡
연세대학교 정보보호연구실

A Study of Implementing Efficient Rotation for ARX Lightweight Block Cipher on Low-level Microcontrollers*

Minwoo Kim,† Taekyoung Kwon‡
Information Security Lab, Graduate School of Information, Yonsei University

요 약

이기종 기기가 상호 연결되어 통신하는 IoT 환경에서는 모든 기기가 일정한 보안 수준을 갖추어야 한다. 그러나 통신·계산 기능이 제약된 기기에서는 상대적으로 암호 알고리즘의 성능이 저하되어 최적화 또는 효율적인 구현 방법이 필요하다. 본 논문에서는 ARX 경량 블록 암호를 대상으로 레지스터를 고려한 효율적인 Rotation 구현 방법을 연구한다. 실제 기기를 이용한 성능 측정을 통해 수정된 Rotation의 효율성을 실증적으로 검증한다. 실험 결과, 수정된 Rotation이 이전보다 개선된 성능을 보여주었으며, 특히, 16비트 MSP 환경에서 실제 기기와 시뮬레이션 성능 측정 결과 사이에 유의한 차이가 있음을 발견하였다.

ABSTRACT

Heterogeneous IoT devices must satisfy a certain level of security for mutual connections and communications. However, a performance degradation of cryptographic algorithms in resource constrained devices is inevitable and so an optimization or efficient implementation method is necessary. In this paper, we study an efficient implementation method for rotation operations regarding registers for running ARX lightweight block ciphers. In a practical sense, we investigate the performance of modified rotation operations through experiments using real experiment devices. We show the improved performance of modified rotation operations and discover the significant difference in measured performance between simulations and real experiments, particularly for 16-bit MSP microcontrollers.

Keywords: ARX, Lightweight Block cipher, Microcontroller, AVR, MSP, ARM, Rotation, Implementation

I. 서 론

다양한 성능의 기기가 상호 연결되어 통신하는 IoT 환경에서는 안전한 통신 환경을 위해 모든 기기가 일정한 보안 수준을 갖추어야 한다. 스마트 홈의 경우, 8MHz의 8비트 CPU부터 1GHz의 32비트 CPU까지 다양한 성능의 기기가 사용되고 있으며 안전성을 보장하기 위해 대칭키, 비대칭키 암호화 등 다양한 암호 기술을 사용하고 있다. 그러나 기존 암호 기술을 통신·계산 기능이 제약된 기기에 그대로 사용하기에는 에너지 소비, 연산 속도 등의 문제가 발생한다[1]. 또한, 통신 프로토콜의 경우 하위 호환을 지원하지만 약한 암호 알고리즘을 사용하는 저사

Received(04. 22. 2016), Accepted(05. 26. 2016)
* 본 연구는 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2015R1A2A2A01004792)
† 주저자, mu.kim@yonsei.ac.kr
‡ 교신저자, taekyoung@yonsei.ac.kr(Corresponding author)

양 기기와 통신하게 된다면 고사양의 기기도 약한 암호 알고리즘을 사용해야 한다. 이는 공격자에게 손쉽게 악용될 수 있으므로 보완이 필요하다.

이를 해결하기 위해 다양한 경량 암호 알고리즘이 등장하였으며 알고리즘 효율성 분석을 위한 다양한 성능 측정 프로젝트가 진행되고 있다[2,3,4,5,6,7]. 프로젝트별 자세한 설명은 5장 관련 연구에서 다룬다. 하지만 성능 측정 프로젝트별로 알고리즘 구현 방법에 차이가 존재하여 통일된 구현 방법이 필요하다. 또한, 프로젝트 중 가장 최근 진행된 FELICS는 8비트, 16비트, 32비트 환경을 대상으로 시뮬레이션을 이용하여 성능 측정을 수행하였으나, 시뮬레이션을 이용한 성능 측정 결과는 실제 기기를 이용한 성능 측정 결과와 차이가 있을 수 있다. 따라서, 본 연구에서는 실제 기기를 이용하여 성능 측정을 진행하며, 시뮬레이션 결과와 비교를 수행한다.

최근 등장하는 경량 블록 암호 알고리즘은 ARX (Addition, Rotation, XOR) 연산을 기반으로 암·복호화를 수행한다. ARX 기반 알고리즘은 치환 테이블 기반 알고리즘에 비해 최적화 효율성이 높지만, 안전성 확보를 위해 다수의 라운드 수행이 필요하다. 이로 인해 ARX 기반 암호 알고리즘은 구현 방법에 따라 성능 차이가 발생한다. 따라서, 본 논문에서는 ARX 기반 암호 알고리즘의 Rotation 연산에 초점을 맞추어 효율적인 구현 방법에 대해 연구한다. 특히, 연산 효율성이 급격하게 저하되는 8비트 AVR, 16비트 MSP 환경을 대상으로 구현 방법을 설명한다. 또한, 실제 기기를 이용한 실험을 통해 구현 방법에 따른 성능 차이를 실증적으로 검증한다.

본 연구의 주요 사항은 다음과 같다.

- 효율적인 ARX 기반 암호 알고리즘 구현을 위한 Rotation 구현 방법 검증
- 실제 기기와 시뮬레이션을 이용한 성능 측정 결과 비교 분석

II. ARX 경량 블록 암호 알고리즘

본 논문에서는 대표적인 경량 블록 암호 알고리즘인 SIMON[8], SPECK[8], LEA[9]를 대상으로 분석을 진행한다. 세 알고리즘 모두 ARX 기반의 연산을 수행하며, 성능 측정 프레임워크를 통해 효율성이 검증되었기 때문에 본 연구의 결과를 검증하기에 적합하다.

ARX 연산은 프로세서가 제공하는 기본 연산을 그대로 사용할 수 있기 때문에 연산 속도가 빠르며, S-Box 저장이 불필요하여 코드 크기나 구현 면적 대비 최적화 효율성이 높다. 그러나, S-Box 기반 연산에 비해 안전성을 확보하기 어려워 높은 라운드 수가 요구되며 안전성 분석이 어렵다.

SIMON, SPECK, LEA에서는 안전성을 위해 다수의 라운드와 Rotation 연산을 수행한다. 각 알고리즘에서 수행하는 라운드 수와 Rotation 연산은 다음 Table 1. 과 같다.

라운드별로 다수의 Rotation 수행하기 때문에 각 알고리즘의 성능은 Rotation 연산 속도에 의존한다. SIMON, SPECK은 블록 크기에 따라 32비트, 48비트, 64비트 Rotation 연산을 수행하며 LEA는 32비트 Rotation을 수행한다. 32비트 기기에서는 Rotation 연산 속도에 문제가 없으나 8비트, 16비트 기기에서는 연산 속도가 저하되므로 최적화된 구현 방법이 필요하다. 따라서, 본 논문에서는 8비트 AVR, 16비트 MSP 기기에서 효율적인 Rotation 구현 방법에 대해 알아본다.

Table 1. Specification of SIMON, SPECK, LEA (ROR #: Rotation Right #-bit, ROL #: Rotation Left #-bit)

Block cipher	Block size	Key size	Rounds	Required Rotation for Encryption	Required Rotation for Decrpytion
SIMON	64	96	42	ROL 1 ROL 8 ROL 2	ROL 1 ROL 8 ROL 2
		128	44		
	96	96	52		
		144	54		
	128	128	68		
		192	69		
		256	72		
SPECK	64	96	26	ROR 8 ROL 3	ROR 3 ROL 8
		128	27		
	96	96	28		
		144	29		
	128	128	32		
		192	33		
		256	34		
LEA	128	128	24	ROR 3 ROR 5 ROL 9	ROR 9 ROL 5 ROL 3
		256	28		
		512	32		

III. 효율적인 Rotation 구현 방법

일반적으로 Rotation 연산을 구현할 때 시프트 연산(〉〉, 〈〈)을 이용한다. 다음 식 1은 32비트 변수 x가 i 번 Rotation 연산되는 코드를 작성한 예이다.

```
ROR(x, i) (x >> i) | (x << (32 - i))    (1)
ROL(x, i) (x << i) | (x >> (32 - i))
```

위 코드는 32비트 레지스터를 사용하는 기기에서는 하나 또는 두 개의 레지스터를 이용하여 효율적인 연산이 가능하다. 그러나 8비트, 16비트 레지스터를 사용하는 기기에서는 연산 수행에 다수의 레지스터가 필요하다. 8비트 레지스터의 경우, 32비트 연산에 4개의 레지스터가 사용되며 상황에 따라 추가 레지스터가 더 필요하다. 다음 식 2의 어셈블리 코드는 8비트 AVR 환경에서 32비트 왼쪽 Rotation 연산 수행 과정을 보여준다. r8~r11은 32비트 변수가 저장된 레지스터이며, r12는 캐리비트가 저장된 레지스터이다.

```
lsl r8
rol r9
rol r10                    (2)
rol r11
adc r8, r12
```

최하위 레지스터 r8에서는 최상위 비트를 상위 레지스터에서 사용할 캐리비트로 내보내기 위해 lsl 연산을 사용하여 r9~r11의 레지스터는 캐리비트를 최하위 비트로 설정하기 위해 rol 연산을 수행한다.

8비트 환경에서는 32비트 연산을 위해 다수의 레지스터를 사용하여 연산을 수행하기 때문에 연산 효율성이 떨어지게 된다. 이를 개선하기 위한 방법을 찾던 중 8비트 AVR 환경에서는 1번의 왼쪽 Rotation과 8번의 Rotation 연산에 대해 특별한 연산을 수행한다는 점을 파악할 수 있었다.

1번의 왼쪽 Rotation의 경우 자기 자신과의 덧셈 연산만으로 Rotation이 가능하다. 다음 식 3의 어셈블리 코드는 AVR-GCC 컴파일러를 이용하여 생성한 1번의 왼쪽 Rotation이다. r24~27은 32비트 변수가 저장된 레지스터이며 r1은 캐리비트가 저장된 레지스터이다.

Table 2. Modified rotation using 1 and 8 bit rotations only

Original	Modified
ROL 1	ROL 1
ROL 2	ROL 1, ROL 1
ROL 3	ROL 1, ROL 1, ROL 1
ROL 4	ROL 1, ROL 1, ROL 1, ROL 1
ROL 5	ROL 8, ROR 1, ROR 1, ROR 1
ROL 6	ROL 8, ROR 1, ROR 1
ROL 7	ROL 8, ROR 1
ROL 8	ROL 8
ROL 9	ROL 8, ROL 1
ROL 10	ROL 8, ROL 1, ROL 1

```
add r24, r24
adc r25, r25
adc r26, r26               (3)
adc r27, r27
adc r24, r1
```

최하위 레지스터 r24는 add 연산을 수행하며 캐리비트를 내보내며 r25~27은 adc 연산을 수행하여 하위 레지스터에서 발생한 캐리비트를 포함한 덧셈 연산을 수행한다.

8번의 Rotation은 방향과 상관없이 레지스터 간 교환(Swap)으로 Rotation이 가능하다. 다음 식 4의 어셈블리 코드는 AVR-GCC 컴파일러를 이용하여 생성한 8번의 왼쪽 Rotation이다. r24~27은 32비트 변수가 저장된 레지스터이며 r18은 임시 저장 레지스터이다.

```
mov r18, r24
mov r24, r27
mov r27, r26               (4)
mov r26, r25
mov r25, r18
```

최하위 레지스터 r24의 값을 임시 레지스터 r18에 저장한 후에 각 레지스터별로 스왑 연산을 수행한다.

위와 같은 특별한 연산 방법으로 인해 1과 8 번의 Rotation 연산은 다른 횟수의 Rotation에 비해 연산 속도가 빠르다. 이점에 착안하여 모든 Rotation 연산을 1과 8 번의 Rotation 연산으로 구현하면 연산 속도 향상이 가능하다. 예를 들어, 2번의 왼쪽 Rotation의 경우 1번의 왼쪽 Rotation을 2번 수

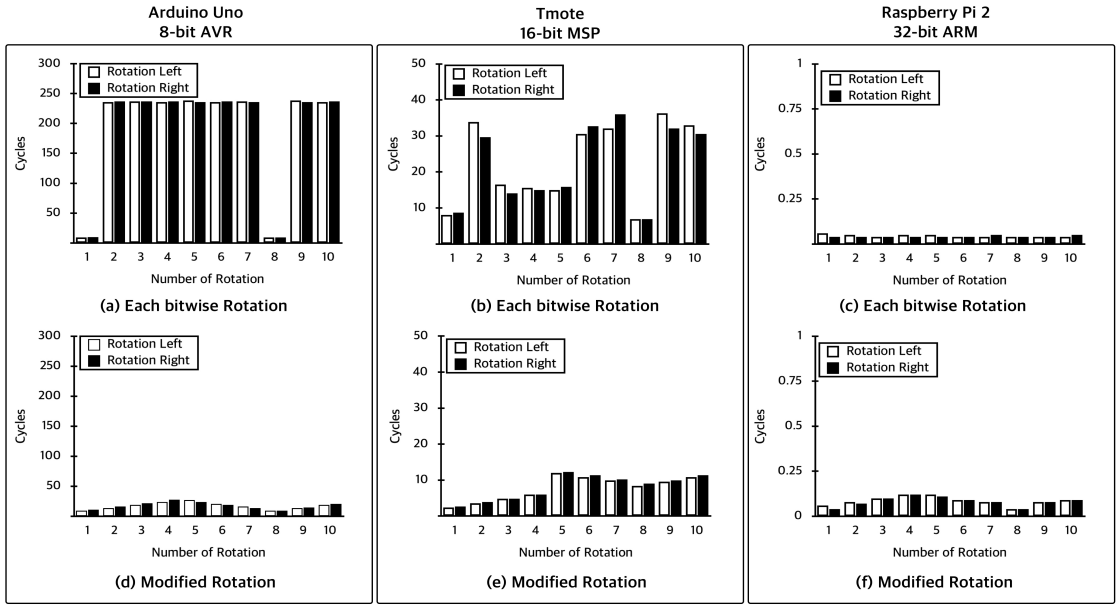


Fig. 1. Performance comparison of bitwise rotations for 32-bit blocks Arduino Uno (a, d), Tmote (b, e), Raspberry Pi 2 (c, f) (a), (b), (c): Each bitwise rotations, (d), (e), (f): Modified rotation using 1 and 8 bit rotations only

행하도록 코드를 작성하며, 3번의 경우는 1번의 Rotation을 3번을 수행하도록 한다. 5회 이상의 Rotation일 경우 8번의 Rotation을 수행한 후 원하는 Rotation 횟수에 맞춰 역방향 혹은 정방향으로 1번의 Rotation을 추가적으로 수행한다. 다음 Table 2.는 1~10번의 왼쪽 Rotation을 1과 8번의 Rotation 만으로 수행되도록 변경한 예이다.

변경된 Rotation 구현 방법은 8비트 AVR 환경뿐만 아니라 16비트 MSP 환경에도 동일하게 적용가능하다. 그러나, 32비트 ARM 환경에서는 기존 연산 방법의 효율성이 뛰어나 변경된 Rotation 구현 방법은 적합하지 않다.

변경된 Rotation의 성능 개선 정도를 실증적으로 알아보기 위해 실제 기기를 이용하여 성능 측정을 수행하였다.

IV. 성능 측정

실험에는 8비트 AVR 기반 마이크로 컨트롤러 아두이노 우노, 16비트 MSP 기반 티모트, 32비트 ARM 기반 라즈베리 파이 2를 사용하였다. 각 기기의 상세 성능은 다음 Table 3.에 정리되어 있다. 실험은 32비트 Rotation을 기준으로 2가지로 분류하여 성능 측정을 수행하였다. 첫 번째로, 구현 방법에 따른 Rotation 별 성능 차이를 측정하였으며, 두 번째로는 구현 방법에 따른 블록 암호의 성능 차이를 측정하였다. 또한, FELICS 성능 측정 프레임워크

Table 3. Specification of IoT device microcontrollers

	Arduino Uno	Tmote	Raspberry Pi 2
MCU	AVR ATmega328	MSP MSP4301611	ARM Cortex-A7
CPU	8-bit RISC	16-bit RISC	32-bit RISC
Frequncy	16MHz	8Mhz	900Mhz
I/O	14 pins	16 pins	HDMI, USB
GPU	-	-	Videocore IV
MEMORY	32KB Flash 2KB SRAM	48KB Flash 10KB SRAM	1GB DRAM (LPDDR2)
EEPROM	4	-	-
Storage	External SD	1024KB	Micro SD

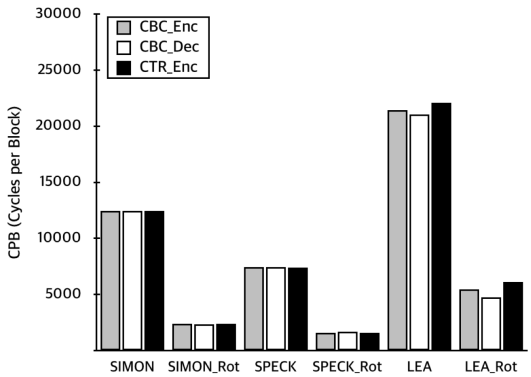


Fig. 2. Performance of Lightweight block ciphers (SIMON, SPECK, LEA) on Arduino Uno (#_Rot: using modified rotation)

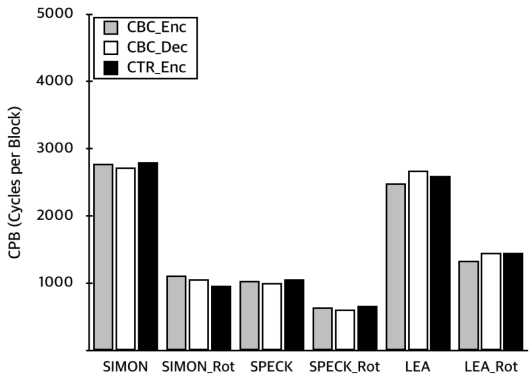


Fig. 3. Performance of Lightweight block ciphers (SIMON, SPECK, LEA) on Tmote (#_Rot: using modified rotation)

[6]의 시뮬레이션 성능 측정 결과와 실제 기기를 통한 성능 측정 결과와의 차이를 비교하였다. 실험에 사용된 블록 암호 알고리즘의 코드는 FELICS 성능 측정 프레임워크[7]에 구현된 코드를 이용하였으며 실제 기기에서 동작 가능하도록 수정 후 실험을 진행하였다.

4.1 로테이션별 성능 측정 결과

다음 Fig. 1-(a).는 아두이노 우노에서 측정한 32비트 Rotation의 횟수별 사이클 수를 보여준다. 기존 코드의 경우 1번과 8번 외의 Rotation 수행 시 성능이 급격하게 저하되는 모습을 보여준다. 그러나, 개선된 구현 방식을 통해 Rotation의 성능을 크게 향상시킬 수 있었다. Fig. 1-(d).는 1과 8번의 Rotation만을 이용하도록 수정된 코드의 성능 측정 결과이다. 이전보다 약 10배 이상 향상된 성능을 보여준다.

Fig. 1-(b), (e).는 티모트에서 측정한 32비트 Rotation과 변형된 Rotation의 횟수별 사이클 수를 보여준다. 8비트 AVR 환경의 아두이노 우노의 결과와 동일하게 1번과 8번의 Rotation 성능이 가장 우수하게 나타났다. 또한, 변형된 Rotation 구현 방법을 통해 약 3배의 성능 향상이 가능하였다.

다음 Fig. 1-(c), (f).는 라즈베리 파이 2에서 측정한 32비트 Rotation과 변형된 Rotation의 횟수별 사이클 수를 보여준다. 타 기기와는 다르게 모든 Rotation의 성능이 균일한 모습을 보여준다. 이로 인해 변형된 Rotation 적용 시 성능 저하가 발생하므로 32비트 ARM 환경에서는 기존 Rotation 구현 방법이 가장 우수한 성능을 보여준다.

4.2 블록 암호 성능 측정 결과

실험에는 대표적인 ARX 경량 블록 암호 알고리즘인 LEA, SIMON, SPECK을 사용하였다. 32비트 Rotation을 수행하도록 하기 위해 LEA는 128비트 블록 크기, SIMON과 SPECK은 64비트 블록 크기로 설정하였으며, 비밀키는 128비트로 동일하게 설정하였다. 성능 측정은 변경된 Rotation을 통해 성능 개선이 가능한 아두이노 우노와 티모트에서만 수행하였다. 다음 Fig. 2.는 아두이노 우노에서 측정한 블록 암호별 성능 측정 결과를 보여준다. 성능 측정 결과 개선된 구현 방식을 통해 알고리즘의 수행 속도가 약 78% 향상되었다.

티모트의 경우도 개선된 구현 방식을 통해 알고리즘의 수행 속도가 약 48% 향상되었다. 티모트의 성능 측정 결과는 Fig. 3.에 정리되어 있다.

4.3 블록 암호 성능 측정 결과 비교

추가적으로 본 논문에서는 실제 기기를 이용한 성능 측정 결과와 시뮬레이션을 통한 성능 측정 결과의 차이를 비교하였다. 실제 기기는 이전과 동일하게 아두이노 우노, 티모트 라즈베리 파이 2를 이용하였으며, 시뮬레이션은 FELICS 성능 측정 프레임워크를 이용하였다. 32비트 ARM 환경의 측정 결과는 모두 실제 기기를 이용한 측정 결과이다.

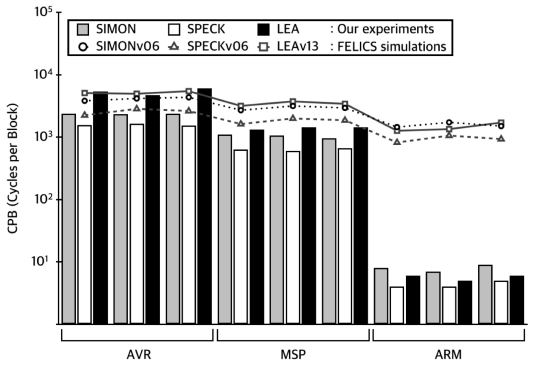


Fig. 4. Performance comparison of our experiments and FELICS simulations (log scale, #_v#: Algorithm version in FELICS)

다음 Fig. 4.는 성능 측정 결과를 비교한 그래프이다. 막대그래프는 실제 기기 성능 측정 결과이며 선 그래프는 FELICS 시뮬레이션 측정 결과이다.

측정 결과 8비트 AVR과 16비트 MSP 환경에서 유사한 성능 측정 결과가 나타났다. 두 결과 값 차이가 유의한 의미가 있는지 파악하기 위해 8비트, 16비트 환경의 성능 측정 결과를 실제 기기와 시뮬레이션으로 분류하여 t-test를 수행하였다. 그 결과, 8비트 AVR 환경은 두 집단 간의 유의한 차이가 없었으며, 16비트 MSP 환경에서는 유의한 차이($t=-6.249$, $p < 0.001$)가 나타났다. 또한, 32비트 ARM 환경에서도 실험 기기의 성능 차이로 인해 결과 값만으로도 많은 차이를 보인다. FELICS 성능 측정 프레임워크에서 사용된 32비트 기기는 아두이노 두에로 84MHz의 클럭 속도로 본 논문에서 사용한 32비트 기기인 라즈베리 파이 2와 약 10배정도의 성능 차이가 나기 때문이다.

분석 결과, 16비트 환경에서 실제 기기와 시뮬레이션 성능 측정 결과에 유의한 차이가 있었다. 특정 환경에서만 유의한 차이가 나타났지만, 모든 환경에 대해 측정 결과의 신뢰성이 보장되지 않음을 확인하였다. 따라서 보다 신뢰성 있는 측정 결과를 위해 성능 측정 연구에서는 실제 기기를 이용하여 실험 결과를 도출해야 한다.

V. 관련 연구

다양한 경량 암호 알고리즘이 등장함에 따라 통합된 환경에서 각 알고리즘의 성능을 비교하기 위해 eBACS[2], ECRYPT Ⅱ[3], BLOC[4], XBX[5], ATHENa[6], FELICS[7] 등 다양한 성능 측정 프로젝트가 등장하고 있다.

Bernstein과 Lange가 수행한 eBACS 프로젝트는 개인 컴퓨터와 서버를 이용하여 다양한 암호 알고리즘의 성능을 측정하였다[2]. 인라인 어셈블리를 포함한 C, Bash, Python을 기반으로 다양한 알고리즘이 구현하여 벤치마킹 프레임워크인 SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives)을 개발하였다. SUPERCOP은 C, C++, 어셈블리로 구현된 알고리즘을 다양한 컴파일러와 컴파일 옵션을 자동으로 변경해가며 수행 시간을 측정한다.

Balasch 등이 수행한 ECRYPT Ⅱ 프로젝트는 Atmel AVR Attiny45 8비트 마이크로 컨트롤러를 이용하여 암호 알고리즘의 성능을 측정하였다[3]. 기존 ECRYPT 프로젝트는 FPGA, ASIC로 알고리즘을 구현하여 수행 시간, 메모리 사용량, 코드 크기를 측정하였다. 이를 확장한 ECRYPT Ⅱ는 알고리즘을 어셈블리로 구현하여 수행 시간, 메모리 사용량, 코드 크기를 측정한다.

Cazorla 등이 수행한 BLOC 프로젝트는 16bit MSP430F1611 마이크로 컨트롤러를 이용하여 경량 블록 암호 알고리즘의 성능을 측정하였다[4]. C 언어를 기반으로 알고리즘을 구현하여 암복호화 수행 시간, 메모리 사용량, 코드 크기를 측정하였다.

Cazorla 등이 수행한 XBX 프로젝트는 SUPERCOP의 확장으로 8비트, 16비트, 32비트의 마이크로 컨트롤러를 이용하여 암호 알고리즘의 성능을 측정하였다[5]. C, Perl, Bash를 기반으로 알고리즘을 구현하여 수행 시간, 메모리 사용량, 코드 크기를 측정하였다.

George Mason 대학에서 수행한 ATHENa 프로젝트는 이전 프로젝트들과 달리 암호 알고리즘을 하드웨어 구현하여 성능을 측정하였다[6]. VHDL, Verilog를 기반으로 알고리즘을 구현하여 수행 시간, 메모리 사용량, 코드 크기를 측정한다.

Dinu 등이 수행한 FELICS (Fair Evaluation of Lightweight Cryptographic Systems) 프로젝트는 8비트 AVR, 16비트 MSP, 32비트 ARM 마이크로 컨트롤러를 이용하여 경량 암호 알고리즘의 성능을 측정하였다[7]. 인라인 어셈블리를 포함한 C, Bash, Python을 기반으로 FELICS 시스템을 개발하였으며, C, 어셈블리로 구현된 알고

리즘의 수행 시간, 메모리 사용량, 코드 크기를 측정한다.

성능 측정 프로젝트 결과를 보면 모든 알고리즘이 8비트, 16비트 환경에서 성능이 저하되는 문제점이 있다. 따라서, 이를 해결하기 위해 SIMON, SPECK, LEA를 대상으로 8비트 환경에 최적화된 구현 방법을 제안하는 논문이 등장하였다.

Beaulieu 등은 8비트 AVR 마이크로 컨트롤러에서 SIMON, SPECK의 최적화된 구현 방법을 제시하였다[10]. 램 사용량 최소화, 처리량 최대화, 플래시 메모리 사용량 최소화 3가지의 상황에 따라 최적화된 구현 방법을 제시한다. 사전 연산을 통한 메모리 사용량 최적화, 반복문 제거를 통해 성능 최적화를 수행하였다.

Seo 등은 8비트 AVR 마이크로 컨트롤러에서 LEA의 최적화된 구현 방법을 제시하였다[11]. 8비트 레지스터를 사용하는 AVR의 특징에 맞추어 연산 방법을 개선하여 최적화된 성능을 보여준다.

본 연구에서는 선행 연구에서 제안한 8비트 AVR 기기 최적화 방법을 일반화하여 소개한다. 레지스터를 고려한 Rotation 연산으로 Rotation을 사용하는 모든 알고리즘에 적용 가능한 최적화된 Rotation 연산 구현 방법에 대해 논한다.

VI. 결 론

본 논문에서는 ARX 경량 블록 암호 알고리즘의 효율적인 Rotation 구현 방법에 대해 소개하였다. 8비트 AVR, 16비트 MSP 환경을 대상으로 레지스터를 고려하여 효율적인 연산이 가능한 구현 방법을 연구하였다. 수정된 구현 방법을 통해 8비트 AVR 환경에서는 기존보다 약 10배, 16비트 MSP 환경에서는 약 3배 빠른 Rotation 수행 속도를 보여주었다. ARX 경량 블록 암호에 적용한 결과, 8비트 AVR 환경에서는 약 78%, 16비트 MSP 환경에서는 약 48% 개선된 수행 속도를 보여주었다. 그러나, 32비트 환경에서는 수정 전 Rotation의 연산 속도가 더 빠르게 나타났다.

실제 기기와 시뮬레이션 사이의 성능 측정 결과 차이를 검증한 결과 유의한 차이가 존재하였다. 16비트 환경에서만 유의한 차이가 나타났지만, 시뮬레이션을 이용한 성능 측정 연구 결과가 모든 환경에서 신뢰성이 보장되지 않음을 확인할 수 있었다. 연구 편의성을 위해 시뮬레이션을 이용한 성능 측정 연구도 가능하지만 이는 지양해야 하며 보다 신뢰성 있는 연구 결과를 위해 실제 기기를 이용하여 연구를 수행하여야 한다.

References

[1] C. Kolias, A. Stavrou, and J. Voas, "Securely Making "Things" Right," Computer, vol.48, no. 9, pp. 84-88, Sept. 2015.

[2] D.J. Bernstein and T. Lange (editors), "eBACS: ECRYPT Benchmarking of Cryptographic Systems," http://bench.cr.yp.to, accessed 7 Mar. 2015.

[3] J. Balasch, B. Ege, T. Eisenbarth, B. Gerard, Z. Gong, T. Gijneyse, S. Heyse, S. Kerckhof, F. Koeune, T. Plos, T. Poppelmann, F. Regazzoni, F. Standaert, G.V. Assche, R.V. Keer, L.O. Oldeneel, and I. Maurich, "Compact implementation and performance evaluation of hash functions in ATtiny devices," CARDIS 2012, LNCS 7771, pp. 158-172, 2013.

[4] M. Cazorla, K. Marquet, and M. Minier, "Survey and benchmark of lightweight block ciphers for wireless sensor networks," Security and Cryptography (SECRYPT), 2013 International Conference on. IEEE, pp.1-6, Jul. 2015.

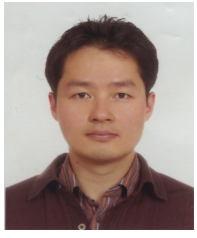
[5] C. Wenzel-Benner, and J. Gräf, "XBX: eXternal Benchmarking eXtension for the SUPERCOPcrypto benchmarking framework," Cryptographic Hardware and Embedded Systems, CHES 2010, LNCS 6225, pp. 294-305, Aug. 2010.

[6] K. Gaj, J.P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamoi, and B.Y. Brewster, "Athena-automated tool for hardware evaluatioN: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs," Field Programmable Logic and Applications (FPL), 2010 International Conference on.

IEEE, pp. 414-421, Sept. 2010.
〔7〕 D. Dinu, Y.L. Corre, D. Khovratovich, J. Großschädl, L. Perrin, and A. Biryukov, "Triathlon of Lightweight Block Ciphers for the Internet of Things," IACR Cryptology ePrint Archive 2015-209, Jul. 2015.
〔8〕 R. Beaulieu, D. Shors, J. Smith, S. T. Lark, B. Weeks, and L. Wingers, "The SIMON and SPECK Families of light-weight Block Ciphers." IACR Cryptology ePrint Archive 2013-404, Jun. 2013.
〔9〕 D. Hong, J. Lee, D. Kim, D. Kwon, K. Ryu, and D. Lee, "LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors," WISA 2013, LNCS 8267, pp. 3-27, Aug. 2014.
〔10〕 R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK block ciphers on AVR 8-bit microcontrollers." LightSec 2014, LNCS 8898, pp. 3-20, Sept. 2014.
〔11〕 H. Seo, Z. Liu, J. Choi, T. Park, and H. Kim, "Compact Implementations of LEA Block Cipher for Low-End Microprocessors," IACR Cryptology ePrint Archive 2015-732, Jul. 2015.

〈저자소개〉

김 민 우 (Minwoo Kim) 학생회원
2015년 2월: 세종대학교 컴퓨터공학과 학사
2015년 3월~현재: 연세대학교 정보대학원 석사과정
〈관심분야〉 암호프로토콜, 네트워크 보안, 스마트폰 보안 등



권 태 경 (Taekyoung Kwon) 종신회원
1992년 2월: 연세대학교 컴퓨터과학과 학사
1995년 2월: 연세대학교 컴퓨터과학과 석사
1999년 8월: 연세대학교 컴퓨터과학과 공학박사
1999년~2000년: U.C. Berkely Post-Doc.
2001년~2013년 8월: 세종대학교 컴퓨터공학과 교수
2007년~2008년: Univ. Maryland at College Park 교환교수
2013년 9월~현재: 연세대학교 정보대학원 교수
〈관심분야〉 암호프로토콜, 네트워크 프로토콜, 센서네트워크 보안, HCI 보안 등