

# IoT 서비스 가용성 강화를 위한 경량 암호 **CHAM S/W** 최적화 구현

2018 국가암호공모전 II-B분야

# 목차

- 연구 배경
- 관련 연구 및 연구 필요성
  - CHAM 경량 암호
  - CHAM 경량 암호 S/W 최적화 구현 연구
  - 연구의 필요성
- 제안 기법
  - CHAM 경량 암호 최적화 구현 구조
  - 8비트 프로세서 환경 상에서의 CHAM 경량암호 최적화 구현 기법
  - 16비트 프로세서 환경 상에서의 CHAM 경량암호 최적화 구현 기법
  - Intel 64비트 프로세서 환경 상에서의 CHAM 경량암호 최적화 구현 기법
- 성능 평가
  - 8비트 프로세서 환경 상에서의 최적화 구현 결과물 성능 평가
  - 16비트 프로세서 환경 상에서의 최적화 구현 결과물 성능 평가
  - Intel 64비트 프로세서 환경 상에서의 최적화 구현 결과물 성능 평가
- 결론 및 향후 계획

# 연구 배경

- **사물인터넷 기술 발전에 따른 경량 암호 수요 증가**
  - 사물인터넷 기술 발전에 따라 자원 제한적 디바이스 사용의 급증과 환경 특성으로 인해, 기존 AES [1], PRESENT [2]와 같은 블록암호보다 경량화된 블록암호 수요 증가
  - **LEA [3], SIMON, SPECK [4]**과 같은 ARX(Addition, Rotation, XOR) 기반 경량 암호 **연구 활발**
  - **CHAM [5]** : 국가보안기술연구소에서 연구 개발한 신규 경량 암호(ICISC'17)
- **국산 경량 암호 활용 증진**
  - LEA [3] : 국내 TTA 표준, ISO 국제 표준 예정, KCMVP 검증대상 암호 알고리즘
  - 신규 경량 암호 **CHAM [5]**에 대한 활용 방안 연구를 통한 국산 경량 암호 활용 증진 필요
- **사물인터넷 서비스 가용성 강화를 위한 CHAM 경량 암호 S/W 최적화 구현 연구**
  - 서버 환경 상에서의 대용량 암호화 데이터에 대한 **CHAM** 경량 암호 고속 병렬화 연산 필요 → **Load-Balancing** 강화를 통한 **IoT 서비스 가용성 강화**
  - 8비트 마이크로프로세서 환경 상에서의 **CHAM** 경량 암호 고속화  
→ **End 디바이스 환경 상에서의 고속 암호화를 통한 IoT 서비스 가용성 강화**



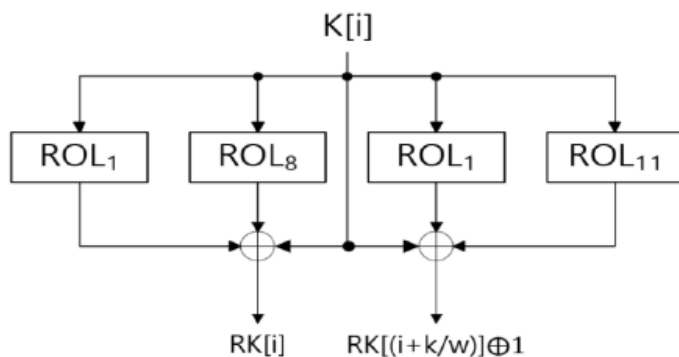
# 관련 연구(1/2)

## • CHAM 경량 암호 [5]

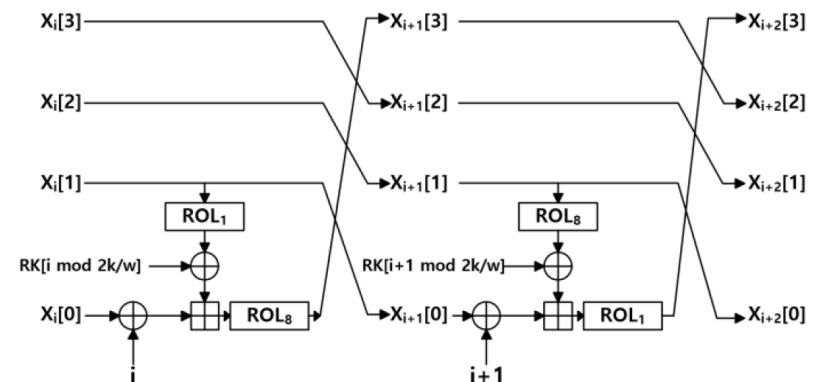
- 2017년 ICISC에서 국가보안기술연구소에서 발표
- CHAM-64/128, CHAM-128/128, CHAM-128/256
- ARX(Addition, Rotation, XOR)연산 기반의 4-branch Feistel 구조

cipher	n	k	r	w	k/w
CHAM-64/128	64	128	80	16	8
CHAM-128/128	128	128	80	32	4
CHAM-128/256	128	256	96	32	8

<CHAM 경량 암호 파라미터>



<CHAM 경량 암호 키 생성과정>



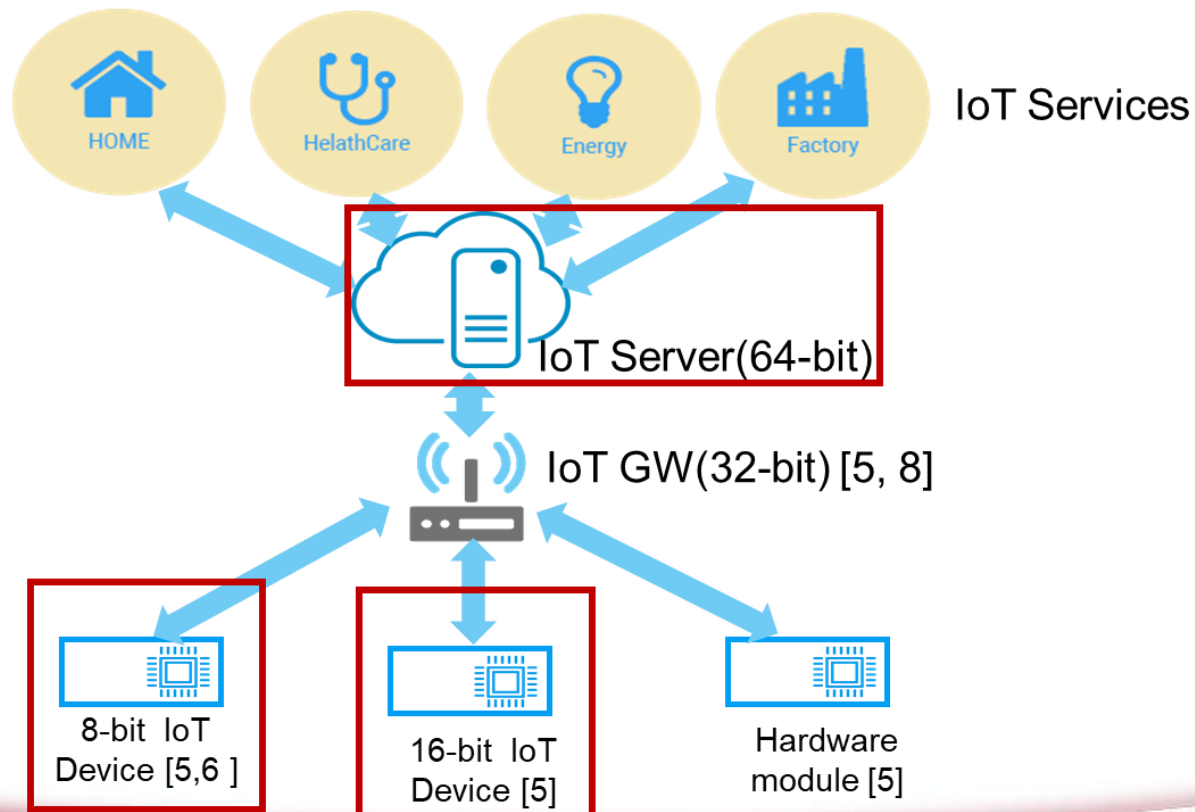
<CHAM 경량 암호 암호화 과정>

# 관련 연구(2/2)

- CHAM 경량 암호 SW 최적화 구현 연구
  - Koo, Bonwook, et al. [5]에서는 **CHAM** 경량 암호 제안 및 **8/16/32-bit** 환경 상에서의 암호 **SW** 최적화 구현 성능을 제시함
    - ASM사용
    - 4라운드 unrolled(ARM환경 상에서의 CHAM128/128제외))
  - Seo, Hwajeong method [6]에서는 **8비트 AVR** 환경 상에서의 CHAM 경량 암호 메모리 최적화 구현 연구 결과 제시
    - 1, 2번 라운드 ASM 구현 및 반복 수행
    - 메모리 공간 정렬을 통한 키 접근 효율성 강화
  - Park, Chanhui, et al. [7]에서는 CHAM 경량 암호에 대한 **JavaScript** 기반 최적화 구현 결과 제시
    - 4라운드 통합 구조 제시
  - Seo, Hwajeong, et al. [8]에서는 CHAM-64/128에 대해 ARM-NEON 프로세서 환경 상에서의 **ARM+NEON** 기반의 최적화 구현 결과 제시
    - NEON 기반 SIMD 최적화 구현
    - ARM+NEON Mix 모델 기반 고속 병렬 최적화 구현
    - OpenMP 기반의 멀티 쓰레딩 고속화 구현

# 연구의 필요성

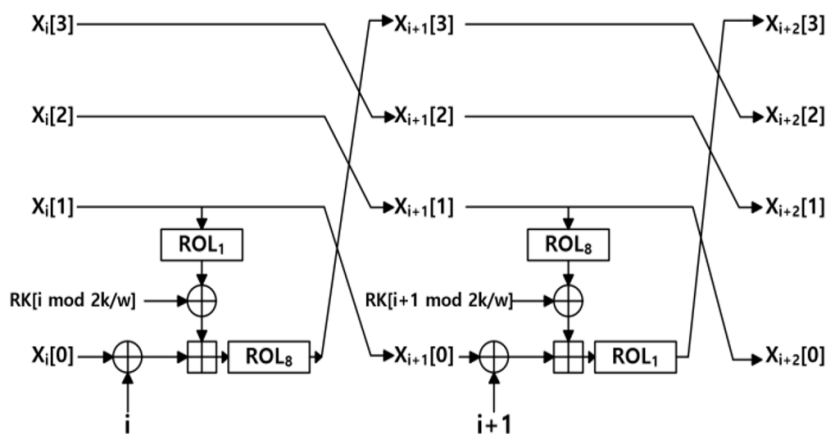
- CHAM 경량암호 SW **최적화 구현** 현황 및 최적화 구현 **필요** 분야
  - 8, 16-bit IoT 디바이스 환경 상 최적화 구현 [5, 6]  
32-bit IoT GW 환경 상 최적화 구현 [5, 8]
  - 하드웨어 모듈 환경 상에서의 최적화 구현 [5]
  - **IoT 서비스 가용성 강화**를 위한 **IoT Server** 환경, **8/16-bit IoT 디바이스 환경** 등에서의 추가적인 최적화 구현 연구 필요



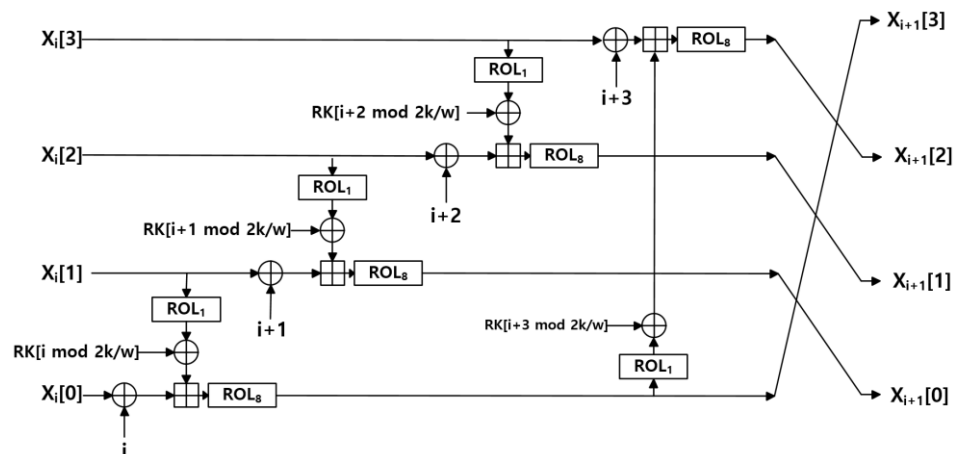
# 제안 기법

## (CHAM 경량 암호 암호화 과정 2라운드 통합 구현 구조)

- CHAM 경량 암호 암호화 과정 **2라운드 통합 구현 구조** [5, 6]
  - CHAM 경량 암호의 **홀수, 짝수 라운드 통합** 수행 구조 기반 구현
- CHAM 경량 암호 암호화 과정 **4라운드 통합 최적화 구현 구조** [7]
  - CHAM 경량 암호의 홀수, 짝수 라운드 구조 및 **4라운드 수행 후, 암호문의 위치가 원 위치로 돌아오는 특성을** 기반하여 4라운드 통합 최적화 구현 구조 설계



<CHAM 경량 암호 암호화 과정 2라운드 통합 구조>

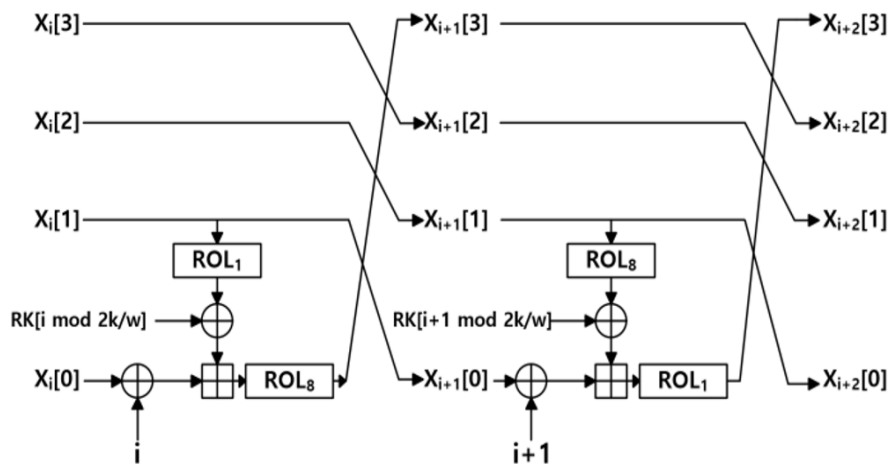


<CHAM 경량 암호 암호화 과정 4라운드 통합 구조>

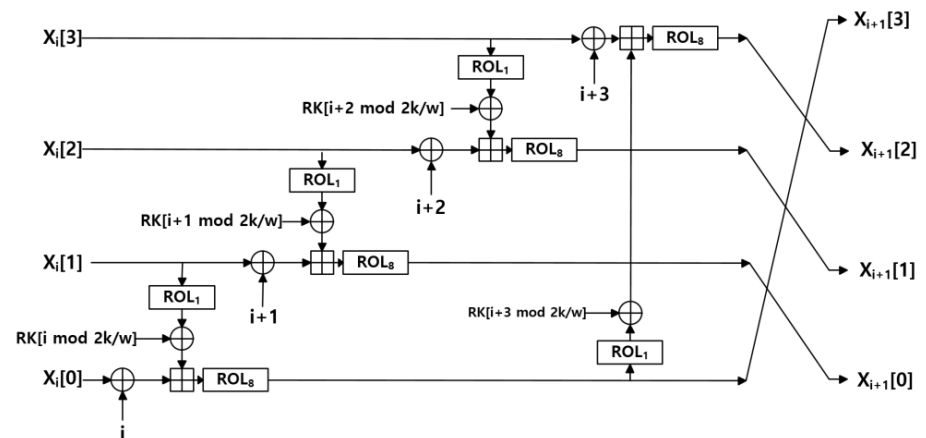
## 제안 기법

### (CHAM 경량암호 8비트 프로세서 상에서의 최적화 구현) (1/4)

- CHAM 경량 암호 암호화 라운드 함수 최적화 구현
  - 2라운드 통합 구조 기반 최적화 구현 [5, 6]
  - 4라운드 통합 구조 기반 최적화 구현 [6, 7]



2Round 통합 CHAM 구현 [6]



4Round 통합 CHAM 구현 [7]



## 제안 기법

### (CHAM 경량암호 8비트 프로세서 상에서의 최적화 구현) (2/4)

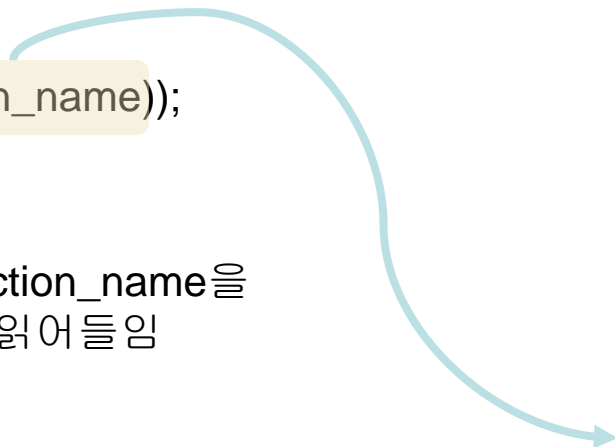
※ 어셈블리 최적화

※ 라운드 키 (메모리) 접근 최적화

```
__attribute__((section (section_name)));
```

pgm\_read\_byte\_near()

→ 해당 메소드를 사용하여 section\_name을  
16bit 단위로 차례로 메모리로 읽어들이м



0x50
0x40
0x30
0x20
0x10
0x00

<Enc 수행부분 : RK 호출 방식>

LPM C1, Z+

EOR X2, C1



R31

R30

※ Z 포인터 사용에서 R31 사용 X, R30만 사용

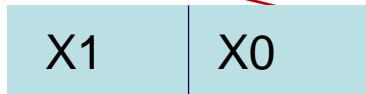
## 제안 기법

### (CHAM 경량암호 8비트 프로세서 상에서의 최적화 구현) (3/4)

※ 8비트 회전 연산 생략 (연산 시 회전된 형태로 연산 지속)

※ 카운터 최적화 (카운터의 최대값은 80이므로, 1 바이트 카운터 활용)

**1Round**  
ADD 결과



X1, X0에 저장

MOV X4, X1  
MOV X5, X0  
로 간단하게 회전  
(2Cycle Clock 소요)

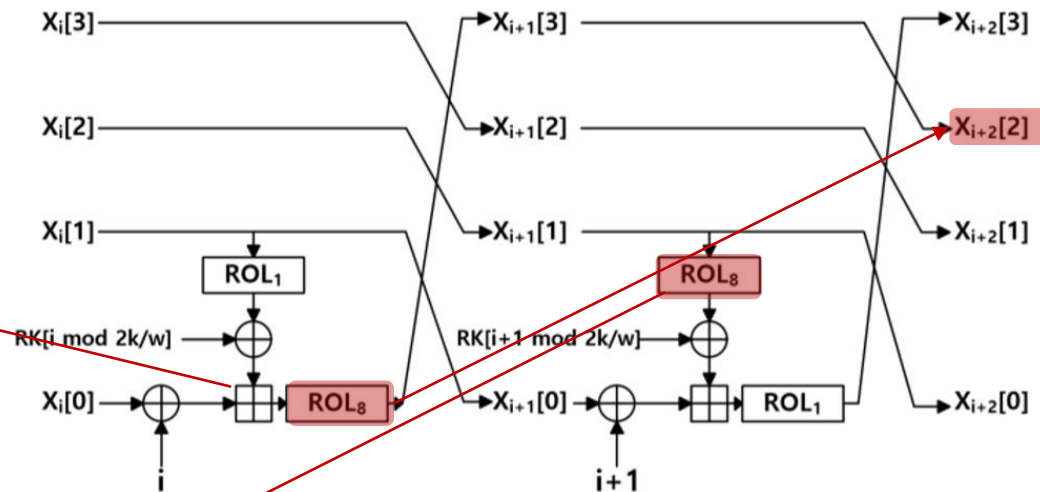
**2Round**

회전을 생략하고 순서를 반대로 하여 연산 지속

LPM C1, Z+  
EOR X5, C1

LPM C1, Z+  
EOR X4, C1

순서 반대 → (0 Cycle Clock 소요)

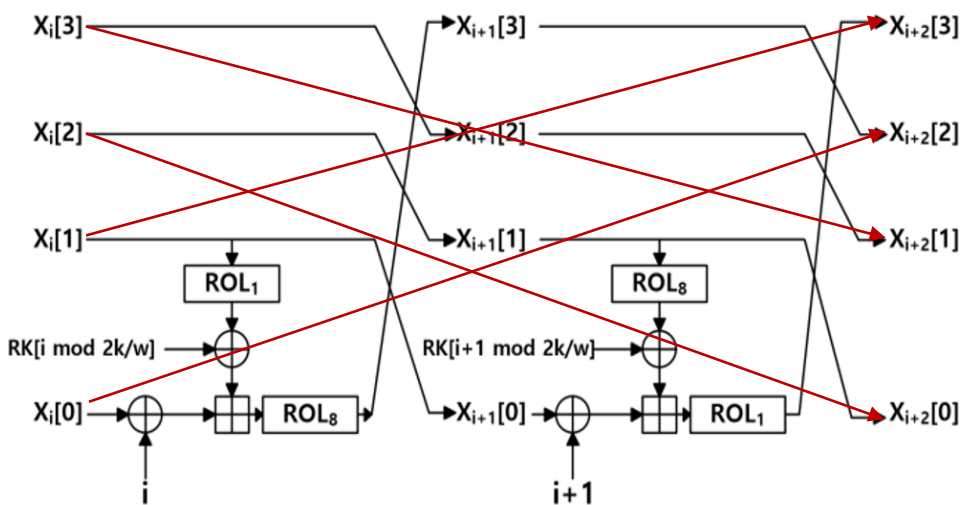


## 제안 기법

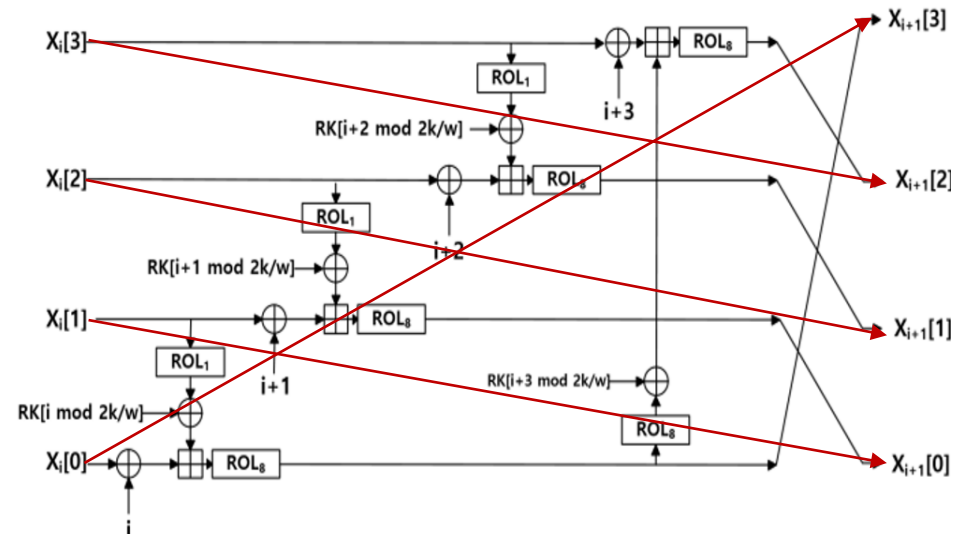
### (CHAM 경량암호 8비트 프로세서 상에서의 최적화 구현) (4/4)

※ 2라운드 방식의 구현에서는 2라운드 종료 후, 레지스터 정렬 적용

※ 4라운드 방식의 구현에서는 4라운드 종료 후, 레지스터 정렬 적용



2Round 통합 CHAM 구현 [2]

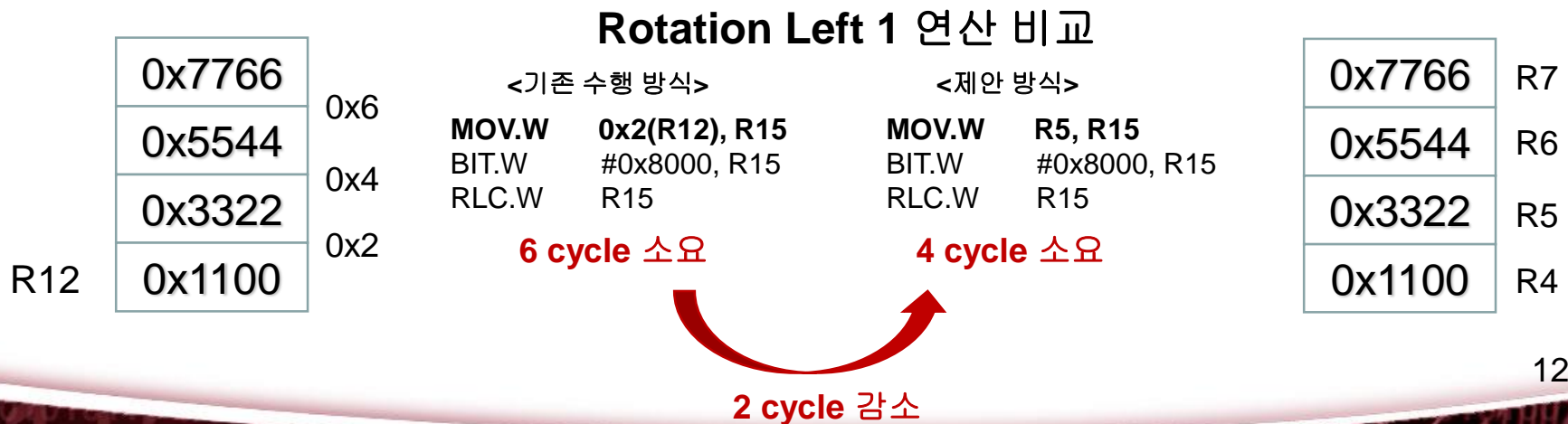


4Round 통합 CHAM 구현 [3]

# 제안 기법

## (CHAM 경량암호 16비트 프로세서 상에서의 최적화 구현)

- 암호화 라운드 처리 방식
  - 16bit MSP430 Instruction set을 활용한 Fully Unrolling 방식
- CHAM-64/128에 대한 고속 최적화 구현
- 최적화 구현 방식
  - MSP430 어셈블리 Instruction set 기반 구현
  - Park et al.[7]에서 제안한 4 라운드 통합 구조의 Fully unrolling 기반 최적화 구현
  - Indexed mode를 최소한으로 사용하여 cycle 수를 최소화





## 제안 기법

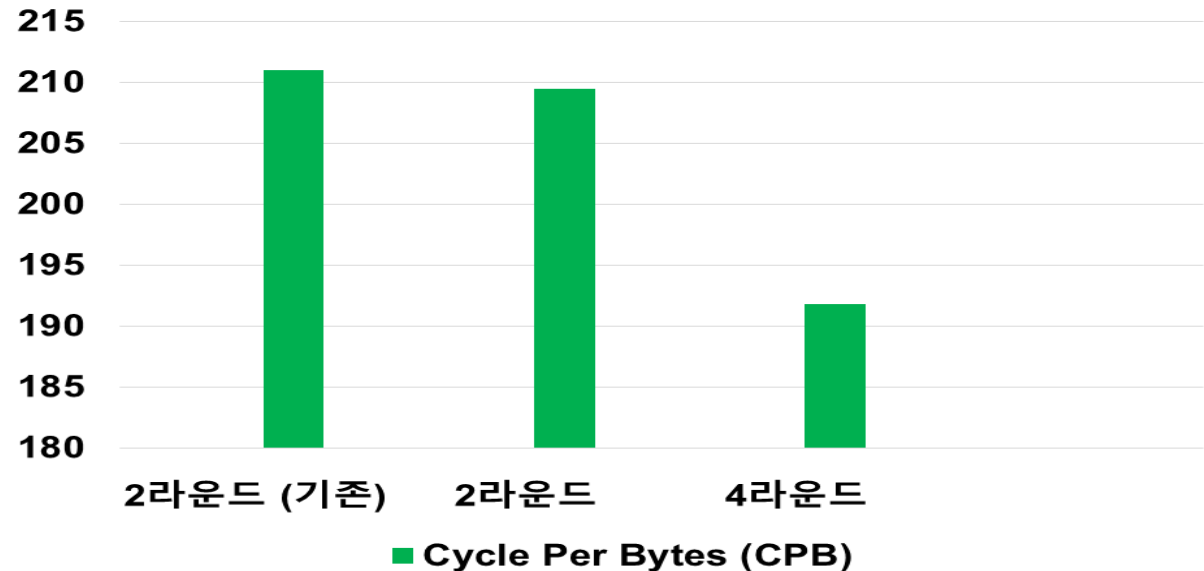
### (CHAM 경량암호 Intel 64비트 프로세서 상에서의 최적화 구현)

- 암호화 라운드 처리 방식
  - 2 라운드 통합 방식
  - 4 라운드 통합 방식
- CHAM 경량 암호 별 다중 데이터 블록 개수 지원
  - CHAM-64/128: 16 blocks/32 blocks
  - CHAM-128/128, CHAM-128/256: 8 blocks/16 blocks
- 최적화 구현 방식
  - Intel AVX2 Intrinsics 기반 구현
  - 다중 데이터 블록 별 AVX2 레지스터 fully usage
  - AVX2 병렬 고속화 파이프라이닝 구현  
(CHAM-64/128: 32 blocks, CHAM-128/128, CHAM-128/256: 16 blocks)

# 성능 평가 (8비트 AVR프로세서 환경)

- 성능 평가 환경
  - Atmel Studio 6.2 IDE
  - AVR-GCC
  - ATMEGA128 (8bit microcontroller)
  - 4KBytes EEPROM
  - 128KBytes Flash Rom
  - 4KBytes SRAM

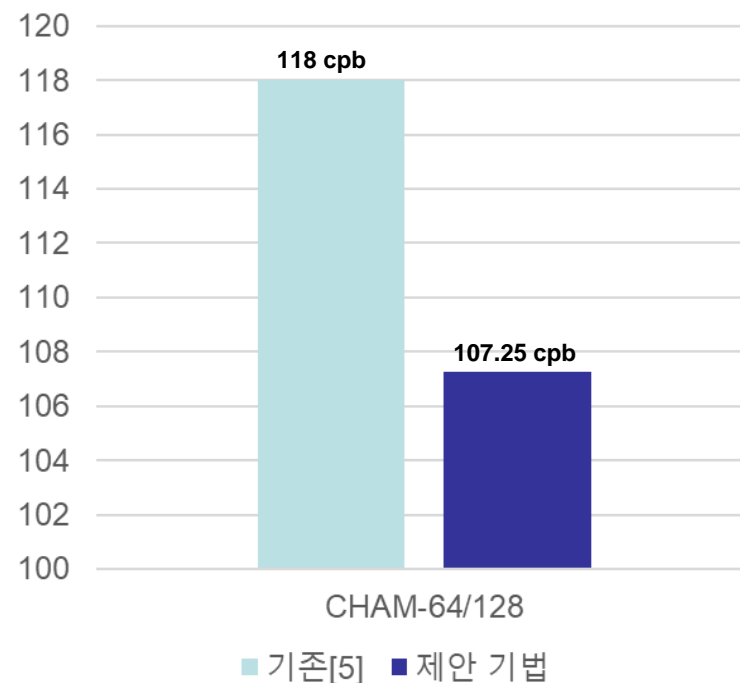
8 Bit AVR 성능평가



- 제안 기법
  - 명령어 최적화
  - 레지스터 최적화
  - 4round 통합 구조의 Fully unrolling
  - 기존 2라운드 기반 [6] 대비 **9.1% 속도 향상**

# 성능 평가(16비트 MSP430 프로세서 환경)

- 성능 평가 환경
  - TI MSP430F1611(16bit RISC architecture)
  - 16개의 레지스터(51 Instruction set)
  - Flash memory : 48 KByte
  - RAM : 10 KByte
- Koo et al. [5] 기법
  - Fixed-key 시나리오
  - 4 round unroll
  - 118 cpb
- 제안 기법
  - Fixed-key 시나리오
  - 4round 통합 구조의 Fully unrolling
  - **107.75 cpb**
  - 기존 기법 대비 **약 8.68%** 성능 향상



기법	cpb
CHAM-64/128[5]	118
<b>CHAM-64/128 (Fully unrolling)</b>	<b>107.25</b>

# 성능 평가(Intel 64비트 프로세서 환경)

- 성능 평가 환경
  - GCC compiler version 5.4.0
  - 컴파일 옵션: -O3 -fomit-frame-pointer -mavx2 -march = native -std = c99 -mtune = native -fwrapv -funroll-loops
  - 운영 체제: Ubuntu 16.04.3 LTS 64-bit
  - CPU/RAM: Intel® Core™ i7-6700 / 32GB RAM
  - 성능 평가: 10,000,000회 수행 평균 cycles/byte 측정
- 성능 평가(1/2)
  - CHAM-64/128: 4라운드 통합(16 blocks/32blocks) 각각 **1.25%, 10.08%** 성능향상
  - CHAM-128/128: 4라운드 통합(8 blocks/16blocks) 각각 **11.81%, 15.59%** 성능향상
  - CHAM-128/256: 4라운드 통합(8 blocks/16blocks) 각각 **2.02%, 12.52%** 성능향상

CHAM64/128	2라운드 통합 (cpb)		4라운드 통합 (cpb)	
	16 blocks	32 blocks	16 blocks	32 blocks
	2.53	2.18	2.50	<b>1.98</b>
CHAM128/128	2라운드 통합 (cpb)		4라운드 통합 (cpb)	
	8 blocks	16 blocks	8 blocks	16 blocks
	2.52	2.17	2.25	<b>1.88</b>
CHAM128/256	2라운드 통합 (cpb)		4라운드 통합 (cpb)	
	8 blocks	16 blocks	8 blocks	16 blocks
	2.76	2.49	2.70	<b>2.21</b>



# 성능 평가(Intel 64비트 프로세서 환경)

## • 성능 평가(2/2)

- Park et al. [9] : **Simeck** 경량 암호에 대한 **AVX2 최적화 구현** 및 대용량 데이터에 대한 최적화 구현 방식 기반의 Adaptive Encryption 방식 제안
- Park et al. [9] 대비 **42.24% 암호 연산 고속화 수행 가능**
- Park et al. [9] 대비 **고속의 Adaptive Encryption 수행 가능** → 서비스 가용성 강화

기법	단위: cpb
<b>CHAM64/128</b> 제안기법 (4라운드 통합, 32 blocks)	<b>1.98</b>
Simeck64/128 for 16 blocks [9]	4.6875

<AVX2 최적화 구현 성능 비교>

기법	예상 성능 (단위: cpb)
<b>Adaptive Encryption</b> (CHAM64/128, 4라운드 통합 기준)	$(2.50 \times n1) + (1.98 \times n2)$
Adaptive Encryption(Simeck64/128) [9]	$(4.6875 \times n1) + (4.6146 \times n2) + (4.6875 \times n3) + (6.8750 \times n4)$

<AVX2 최적화 구현기반 Adaptive Encryption 성능 비교>

# 결론 및 향후 계획

- 결론

- 8bit AVR 프로세서 환경에서 2라운드, 4라운드 통합 구현 개발
- 16bit MSP430 프로세서 환경에서 CHAM-64/128에 대한 4라운드 통합 최적화 구현 개발
- Intel 64bit 프로세서 환경 상에서의 2라운드, 4라운드 통합 구조 AVX2 기반의 최적화 구현 개발
- **IoT End 디바이스 환경과 서버 환경 상에서의 CHAM 경량암호 최적화 구현**  
→ IoT 서비스 가용성 강화

- 향후 계획

- 8-bit AVR 프로세서 최적화 구현 연구
  - 8bit AVR 프로세서 환경 상에서의 4라운드 기반 최적화 추가 구현 연구
  - CHAM 암호에 대한 **WHITE BOX 암호화 구현 방안 연구**
- 16-bit MSP430 프로세서 최적화 구현 연구
  - 16bit MSP430 프로세서 환경 상에서의 4라운드 통합 구조 기반 **CHAM-128/128, 128/256 최적화 추가 구현 연구**
- 64-bit Intel 프로세서 최적화 구현 연구
  - OpenMP기반 SIMT(Single Instruction Multiple Thread) 구현 기법 적용 연구
  - OpenSSL,mbedTLS 등 (D)TLS 보안 통신에 CHAM 블록암호 적용 연구

# 참고 문헌

- [1] Rijmen, Vincent, and Joan Daemen. "Advanced encryption standard." Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology (2001): 19-22.
- [2] Bogdanov, Andrey, et al. "PRESENT: An ultra-lightweight block cipher." International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2007.
- [3] Hong, Deukjo, et al. "LEA: A 128-bit block cipher for fast encryption on common processors." International Workshop on Information Security Applications. Springer, Cham, 2013.
- [4] Beaulieu, Ray, et al. "The SIMON and SPECK lightweight block ciphers." Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE. IEEE, 2015.
- [5] Koo, Bonwook, et al. "CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices." International Conference on Information Security and Cryptology. Springer, Cham, 2017.
- [6] Seo, Hwajeong method, " Memory-Efficient Implementation of Ultra-Lightweight Block Cipher Algorithm CHAM on Low-End 8-Bit AVR Processors." JKII SC 28.3 (2018): 545-550.
- [7] Park, Chanhui, et al. "Optimization of CHAM Encryption Algorithm based on Javascript." Ubiquitous and Future Networks (ICUFN), 2018 10th International Conference on. IEEE, 2018.
- [8] Seo, Hwajeong, et al. "Parallel Implementations of CHAM." International Workshop on Information Security Applications. Springer, Cham, 2018.
- [9] Park, Taehwan, et al. "Secure Data Encryption for Cloud-Based Human Care Services." Journal of Sensors 2018 (2018).

감사합니다