# 8-bit AVR 프로세서 상의 Revised CHAM 어셈블리 최적 구현

권혁동*, 김현지**, 박재훈**, 심민주**, 서화정**†
*한성대학교 정보컴퓨터공학과
**한성대학교 IT융합공학부
korlethean@gmail.com, khj1594012@gmail.com, p9595jh@gmail.com,
minjoos9797@gmail.com, hwajeong84@gmail.com

# The Optimal Assembly Implementation of Revised CHAM on 8-bit AVR Processor

Hyeok-Dong Kwon*, Hyun-Ji Kim**, Jae-Hoon Park**,
Min-Joo Sim**, Hwa-Jeong Seo**†
*Dept. of Information Computer Engineering, Hansung University
**Dept. of IT Convergence Engineering, Hansung University

요 약

경량 암호는 컴퓨팅 파워가 부족한 저사양 프로세서를 위해 개발되었다. CHAM은 국산 경량 암호 중 하나로, 세 가지의 규격을 제공하며 ARX 구조를 사용한 암호이다. CHAM 발표 이후, 라운드 수를 조절하여 성능을 향상시킨 Revised CHAM이 제안되었다. 기존 CHAM은 8-bit AVR 프로세서 상에서 최적 구현이 이루어졌지만, 최신 기술인 Revised CHAM은 해당 구현물이 존재하지 않는다. 따라서 8-bit AVR 프로세서를 대상으로 Revised CHAM-64/128을 최적 구현하여 최상의 성능으로 연산이 진행되도록 한다. 본 논문에서는 최적 구현에 사용한 기법들을 소개하며, 기존에 제안된 기법과 성능 비교를 통해 본 기법의 우수함을 서술한다.

## 1. 서론

CHAM은 저사양 프로세서를 대상으로 개발된 국산 경량 암호로 ARX 연산을 사용한다. 이후 CHAM의 라운드 수를 변경한 Revised CHAM이 제안되었다. 본 논문에서는 Revised CHAM의 규격 중 하나인 64/128의 최적 구현을 시도한다. 대상 프로세서는 8-bit AVR 프로세서인 Atmega128 프로세서이다. 논문의 구성은 다음과 같다. 2장에서 CHAM에 관한 내용과 기존 최적화 구현물에 대해 확인하며 3장에서 제안 기법에 적용된 최적화 기법을 확인한다. 4장에서 기존 기법과의 성능비교를 진행하고 5장에서 결론을 맺는다.

## 2. 국산 경량 암호 CHAM

### 2.1 Revised CHAM[1][2]

Revised CHAM의 원형인 CHAM은 ICISC'17에서 발표된 경량 암호이다. CHAM은 Addition, Rotation, eXclusive-or의 세 가지 연산을 사용한다. CHAM에서 사용하는 파라미터는 표 1로 명시한다.

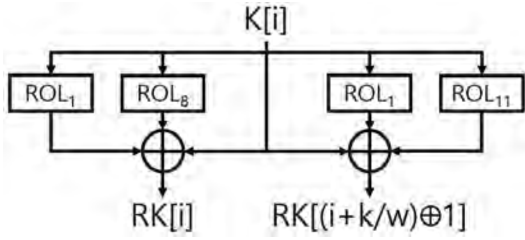
<표 1> CHAM의 파라미터

| | n | k | w | k/w | r |
|---|---|---|---|---|---|
| CHAM-64/128 | 64 | 128 | 16 | 8 | 80(88) |
| CHAM-128/128 | 128 | 128 | 32 | 4 | 80(112) |
| CHAM-128/256 | 128 | 256 | 32 | 8 | 96(120) |

이후 ICISC'19에서 CHAM의 개량형인 Revised CHAM이 발표되었다. Revised CHAM은 CHAM과 동일한 구조 및 규격을 지니지만, 라운드 수만 다르다. 표 1의 괄호 부분이 Revised CHAM의 파라미터이다.

Revised CHAM은 키 상태를 저장하지 않는 스테이트리스 기법을 사용하며, 그림 1과 같은 키 스케줄링을 통해 라운드 키를 생성한다. Pseudo 코드 형태로 표현하면 다음과 같다.

for $(i=0;\ i<k/w;\ i++)\{$
$RK[i] \leftarrow K[i] \oplus (K[i] \lll 1) \oplus (K[i] \lll 8)$
$RK[(i+k/w) \oplus 1] \leftarrow K[i] \oplus (K[i] \lll 1) \oplus (K[i] \lll 11)$
$\}$



(그림 1) Revised CHAM의 키 스케줄링

이후 평문을 4개의 블록으로 나눈 후, 라운드 함수를 반복하여 암호화를 진행한다. 라운드 함수의 구조는 전체적으로 동일하지만 짝수, 홀수 라운드 별로 회전 연산의 횟수만 다르다. 라운드 함수는 그림 2와 같으며 Pseudo 코드로 표현하면 다음과 같다.

$$
\begin{aligned}
&\mathrm{for}\,(i=0;\ i<r;\ i{+}{+})\{\\
&\quad (x_{i+1},\ y_{i+1},\ z_{i+1},\ w_{i+1}) \leftarrow (y_i,\ z_i,\ w_i,\backslash\\
&\quad (((x_i \oplus i)+((y_i \lll \alpha_i)\oplus RK_{i \bmod 2k/w}))\bmod 2^w \lll \beta_i))\\
&\}
\end{aligned}
$$

코드에서 α, β는 i값에 따라 달라진다. 만약 i가 짝수라면 α = 1, β = 8이며 반대로 홀수라면 α = 8, β = 1이 대입된다.

### 2.2 기존 제안 기법

[3]은 CHAM을 8-bit AVR 프로세서 상에서 최적 구현하였다. [3]에 적용된 기법은 다음과 같다.

첫째로 라운드 키 접근이다. CHAM-64/128의 라운드 키는 32바이트이다. 즉, 메모리에 대한 최대 범위는 32이므로, 레지스터 하나로 모든 메모리 접근이 가능하다. 이를 구현하기 위해서는 상위 주소를 고정한 채, 하위 주소에 0x00값을 설정한다. 따라서 1바이트 오프셋 연산으로 메모리 접근이 가능하다.

둘째는 카운터 최적화이다. CHAM은 라운드 카운터와 XOR하는 부분이 있다. 64-128 규격은 80라운드를 거치는데, 레지스터 하나의 표현 범위가 256이므로 레지스터 하나로 카운터 관리가 가능하다.

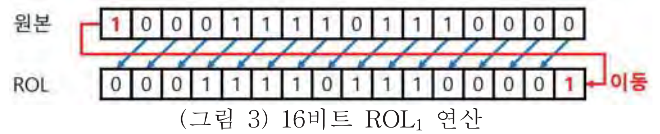
마지막은 메모리 접근 최적화이다. 후 증가 명령어를 활용하여 메모리 접근 이후, 다음에 연산할 메모리 주소로 자동으로 이동하도록 한다.

## 3. 제안 기법

[3]은 CHAM에 대한 최적 구현이므로 Revised CHAM-64/128에 대한 최적 구현을 한다. 본 장에서는 [3]의 기법을 포함한 채로, 추가적으로 적용한 기법을 서술한다.

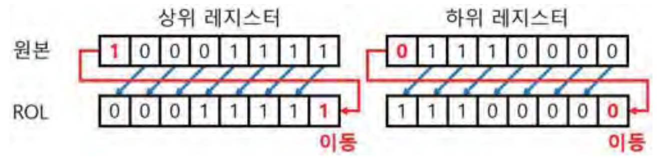
### 3.1 회전 연산 최적화

Revised CHAM-64/128의 블록 크기는 16비트이며, 회전 연산을 취한 결과는 그림 3과 같다.



(그림 3) 16비트 $ROL_1$ 연산

하지만 대상 프로세서는 8비트 레지스터를 사용하므로 블록 하나를 두 개의 레지스터에 나누어 저장한다. 때문에 단순하게 회전 연산을 적용하면 그림 4와 같이 의도하지 않은 결과가 도출된다.

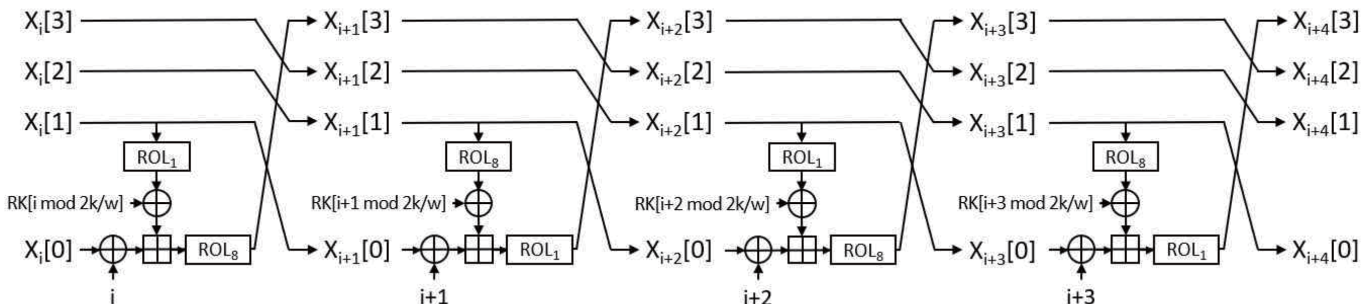


(그림 4) 8비트 레지스터 상의 16비트 $ROL_1$ 연산

정상적인 회전 연산을 위해서는 ROL 외에 추가적인 명령어가 필요하다. 최소한으로 구현하기 위해서 표 2 좌측의 명령어 모음을 사용한다.

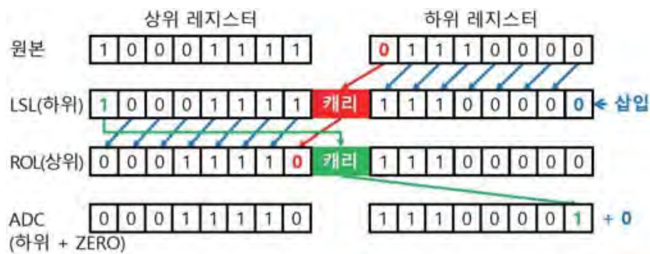
<표 2> 회전 연산 최적화 코드

| $ROL_1$ | | $ROL_8$ | |
|---|---|---|---|
| LSL | LO | MOV | TEMP, LO |
| ROL | HI | MOV | LO, HI |
| ADC | LO, ZERO | MOV | HI, TEMP |



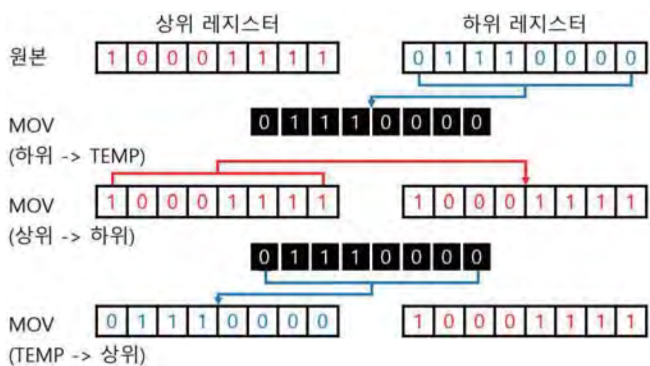
(그림 2) CHAM의 라운드 함수

가장 먼저, 하위에 LSL 명령어를 사용하여 모든 비트를 좌측으로 1비트씩 옮긴다. 이때 좌측 끝 비트는 캐리에 저장된다. ROL 명령어는 상위에 사용하며, 캐리에 있는 하위의 좌측 끝 비트를 상위의 우측 끝 비트에 저장한다. ROL 명령어로 인해 상위의 좌측 끝 비트는 캐리에 저장된 채로 종료된다. 마지막으로 ADC 명령어로 하위와 제로 레지스터를 더해준다. ADC 명령어는 캐리를 최하위 비트에 더해줄 수 있기 때문에 캐리에 저장된 상위의 좌측 끝 값을 하위의 우측 끝에 저장하여 회전 연산을 3사이클로 완성한다. 이 과정은 그림 5와 같다.




(그림 5) $ROL_1$ 최적 구현 코드 동작 과정

Revised CHAM에는 8회전 연산도 존재한다. 블록 하나는 16비트이며 이를 8회전 한다면, 중앙을 기준으로 앞뒤의 값이 반전된다. 대상 레지스터는 8비트 레지스터를 사용하므로 본 특징을 활용 가능하다. 구현에는 표 2 우측의 코드를 사용한다. 레지스터 값을 교차하여 8회전 연산을 구현한다면 3사이클이 소요되며 동작은 그림 6과 같다. 추가로 본 연산을 두 번 연속 연산할 경우, 초깃값으로 복구된다.



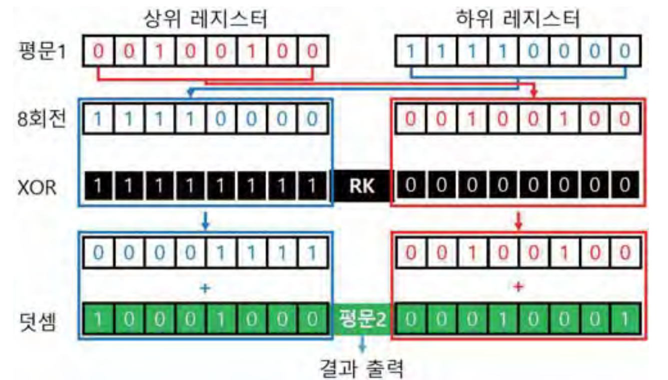

(그림 6) $ROL_8$ 최적 구현 코드 동작 과정

3.2 회전 연산 횟수 최적화

그림 2에서 $X_i[0]$의 1라운드, 그리고 $X_i[2]$의 2라운드에서 8회전 연산을 제거할 수 있다. 우선 $X_i[0]$의 경우 1라운드의 마지막에서 8회전 연산을 거친다. 하지만 4라운드에서 $X_i[0]$가 입력 값으로 사용될 때, 다시 8회전 연산을 거친다. 3.1절에서 서술하였듯, 8비트 레지스터 상에서 8회전 연산을 두 번 거치면 값이 원상으로 돌아온다.

따라서 1라운드의 8회전 연산을 생략한다면 4라운드에서도 생략 가능하다[4]. 다만 $X_i[0]$에는 8회전된 값이 있어야 이후 연산 결과가 정상적이므로 4라운드 종료 직전에 $X_i[0]$에 8회전 연산을 취해준다.

$X_i[2]$의 일반적인 연산 과정은 그림 7과 같다.




(그림 7) 기존 2라운드의 연산 과정

하지만 $X_i[2]$는 XOR와 덧셈 연산 시에, 표 3의 코드와 같이 레지스터를 교차해서 연산하는 것으로 자체적으로 8회전 연산을 포함시킬 수 있다. 동작을 묘사한다면 그림 8과 같이 표현 가능하다. 그림 8의 동일 색상의 음영끼리 연산을 수행한다.

<표 3> 레지스터 교차 연산 코드

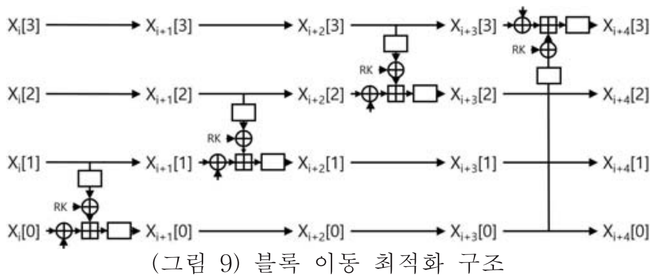
| Register reverse method | |
|---|---|
| LPM | RK, Z+ // RK 하위 8비트 |
| EOR | HI2, RK |
| LPM | RK, Z+ // RK 상위 8비트 |
| EOR | LO2, RK |
| ADD | LO1, HI2 |
| ADC | HI1, LO2 |

(그림 8) 8회전 연산을 생략한 2라운드 연산 과정

이것으로 Revised CHAM-64/128의 전체 88라운드 중 44회의 8회전 연산이 생략되며, 132사이클을 줄일 수 있다.

3.3 블록 이동 최적화[5]

Revised CHAM-64/128은 평문을 4개의 블록으로 나누고 라운드의 종료 직전 모든 블록은 좌측방향으로 이동한다. 이때 각 블록은 4라운드 이후 제자리로 돌아오는 것을 알 수 있다. Revised CHAM-64/128은 4의 배수인 88라운드로 동작한다. 때문에 1라운드 시작 시의 블록 위치와 88라운드 종료 시의 블록 위치는 동일하다. 즉, 블록 위치를 이동하지 않더라도 최종 출력은 동일하다. 이것으로 라운드 함수 구조를 그림 9의 형태로 변형할 수 있다. 블록 이동을 생략하면 라운드별 5개의 MOVW 명령어가 생략되며, 동작 사이클 관점에서는 880사이클이 감소된다.




(그림 9) 블록 이동 최적화 구조

## 4. 성능 평가

본 장에서는 구현물의 성능 비교를 한다. 구현에는 Atmega128 프로세서를 사용한다.

8-bit AVR 프로세서에 대한 CHAM 최적 구현은 [3]이 존재하나, 해당 구현물은 Revised CHAM에는 대응하지 않는다. 그러므로 본 논문의 구현물과 비교하기 위해 해당 구현물을 Revised CHAM 버전으로 이식하여 비교한다. 결과는 표 4와 같다.

<표 4> 성능 평가 결과

| 구현물 | ROM | RAM | cpb |
|---|---|---|---|
| [3] + Revised | 152 | 3 | 232 |
| This Work | 198 | 3 | 209 |

제안기법이 [3]에 비해 다소 큰 코드 사이즈를 가진다. 하지만 속도 면에서 [3]은 232cpb로 동작하는 반면, 제안하는 구현물은 209cpb로 동작한다. 이는 9.9%의 속도 향상에 해당된다.

## 5. 결론

본 논문에서는 Revised CHAM-64/128을 8-bit AVR 프로세서를 대상으로 최적 구현하였고, 기존 구현에 비해 9.9%의 속도 향상을 달성했다.

향후 과제로는 Revised CHAM의 남은 규격 두 가지에 대해서 최적 구현을 진행할 예정이다. 또한 본 구현물의 성능을 보다 더 개선할 수 있는 사항에 대해 확인해본다.

## 참고문헌


[1] Bonwook Koo, Dongyoung Roh, Hyeonjin Kim, Younghoon Jung, Dong-Geon Lee, and Daesung Kwon, "CHAM: A Family of Lightweight Block Ciphersfor Resource-Constrained Devices," *International Conference on Information Security and Cryptology*, Seoul, Korea, pp 3-25, 2017.

[2] Dongyoung Roh, Bonwook Koo, Younghoon Jung, IlWoong Jeong, Dong-Geon Lee, Daesung Kwon, and Woo-HwanKim, "Revised Version of Block Cipher CHAM," *International Conference on Information Security and Cryptology*, Seoul, Korea, pp 1-19, 2019.

[3] Hwajeong Seo, "Memory-Efficient Implementation of Ultra-Lightweight Block Cipher Algorithm CHAM on Low-End 8-Bit AVR Processors," *Journal of the Korea Institute of Information Security & Cryptology*, Vol. 28, No. 3, pp 545-550, 2018.

[4] Sujin Lee, Junyoung Kang, Dowon Hong, and Changho Seo, "Research for Speed Improvement Method of Lightweight Block Cipher CHAM using NEON SIMD," *Journal of KIISE*, Vol. 46, No. 5, pp 485-491, 2019.

[5] Taeung Kim, and Deukjo Hong, "Software Implementation of Lightweight Block Cipher CHAM for Fast Encryption," *Journal of the Korea Society of Computer and Information*, Vol. 23, No. 10, pp 111-117, 2018.

[6] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark,Bryan Weeks, and Louis Wingers, "The SIMON and SPECK Block Ciphers on AVR 8-bit Microcontrollers," *International Workshop on Lightweight Cryptography for Security and Privacy*, Istanbul, Turkey, pp 3-20, 2014.