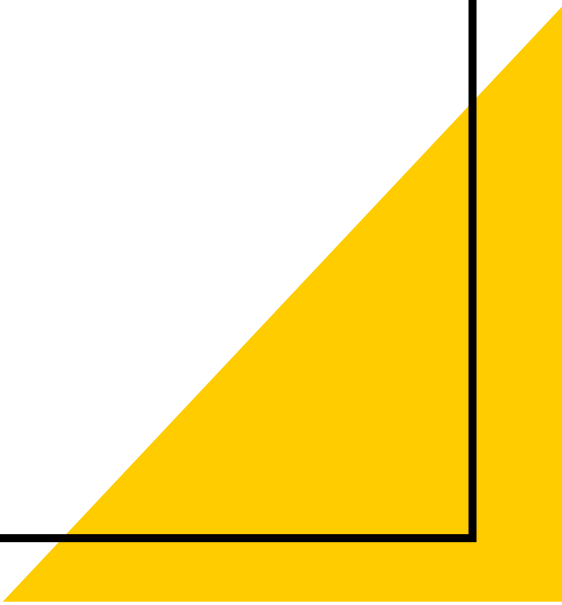


8-BIT AVR 프로세서 상에서의 블록암호 알고리즘 CHAM 메모리 최적화 구현 논문 리뷰

IT정보공학과 신명수

목차

1. 초경량 블록암호 알고리즘 CHAM 소개
 2. 구현 환경 및 지표
 3. CHAM의 구조와 이전 CHAM 구현 결과
 4. 메모리 최적화 구현
 5. 성능 평가
- 
- A yellow decorative triangle is located in the bottom right corner of the slide, pointing towards the top right.

1. 초경량 블록암호 알고리즘 CHAM

- ICISC'17 에서 발표된 암호 알고리즘으로 저성능 사물인터넷 플랫폼(8-bit AVR, MSP430, 32-bit ARM) 상에서 최적화 구현이 가능함.
- ARX 기반 암호 알고리즘
- Stateless 라운드 키 기법을 사용함.
- 64/128 , 128/128, 128/256 3가지 암호화 모드를 제공한다.

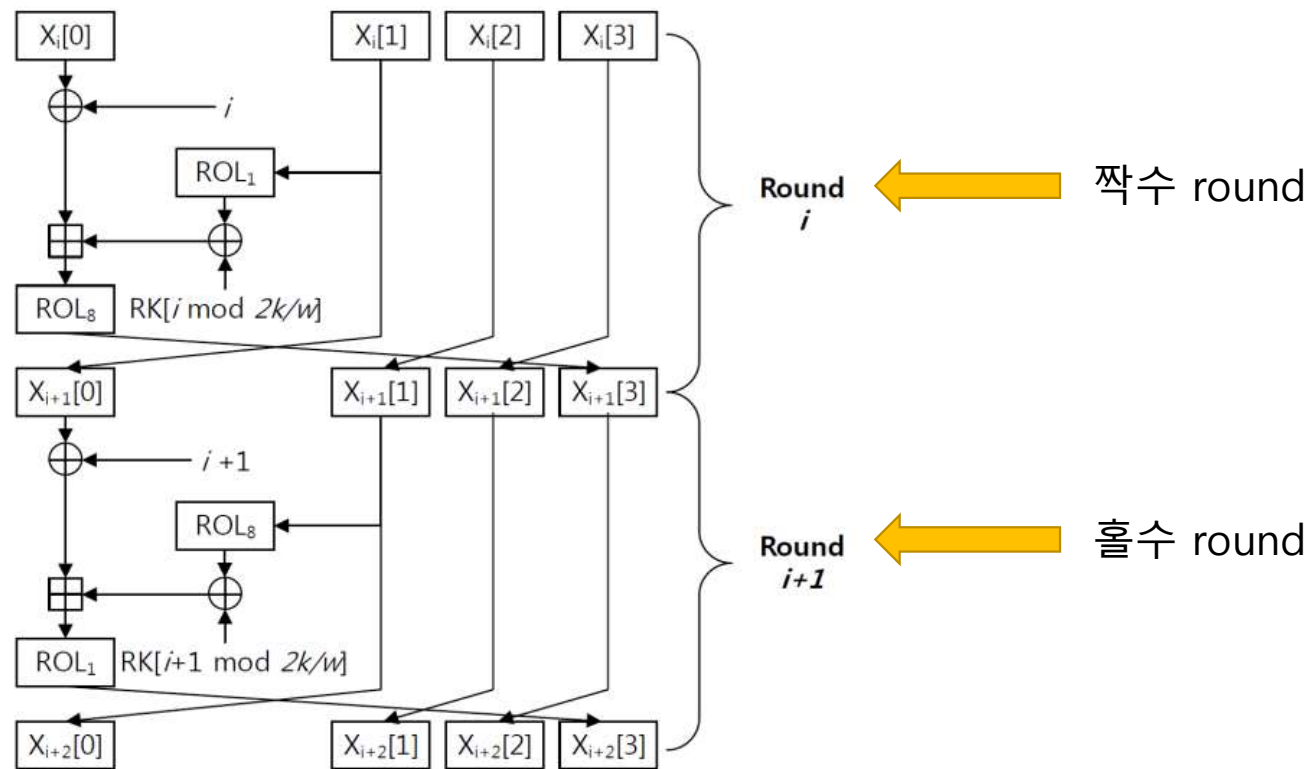
1. 초경량 블록암호 알고리즘 CHAM

- ARX 기반 암호 알고리즘
 - S-Box와 P-Box가 있는 SPN구조와 달리 Addition, Rotation, Exclusive-OR 연산으로 구성됨.
 - 대표적인 알고리즘으로는 HIGHT, LEA, SIMON, SPECK 등이 있다.
- Stateless 라운드 키 기법을 사용함.
 - 키 저장 공간을 줄일 수 있다.
- 64/128 , 128/128, 128/256 3가지 암호화 모드를 제공한다.
 - 64/128 -> 64-bit 평문 / 128-bit 마스터키 / 80 round / 16 word
 - 128/128 -> 128-bit 평문 / 128-bit 마스터키 / 80 round / 32 word
 - 128/256 -> 128-bit 평문 / 256-bit 마스터키 / 96 round / 32 word

2. 구현 환경 및 지표

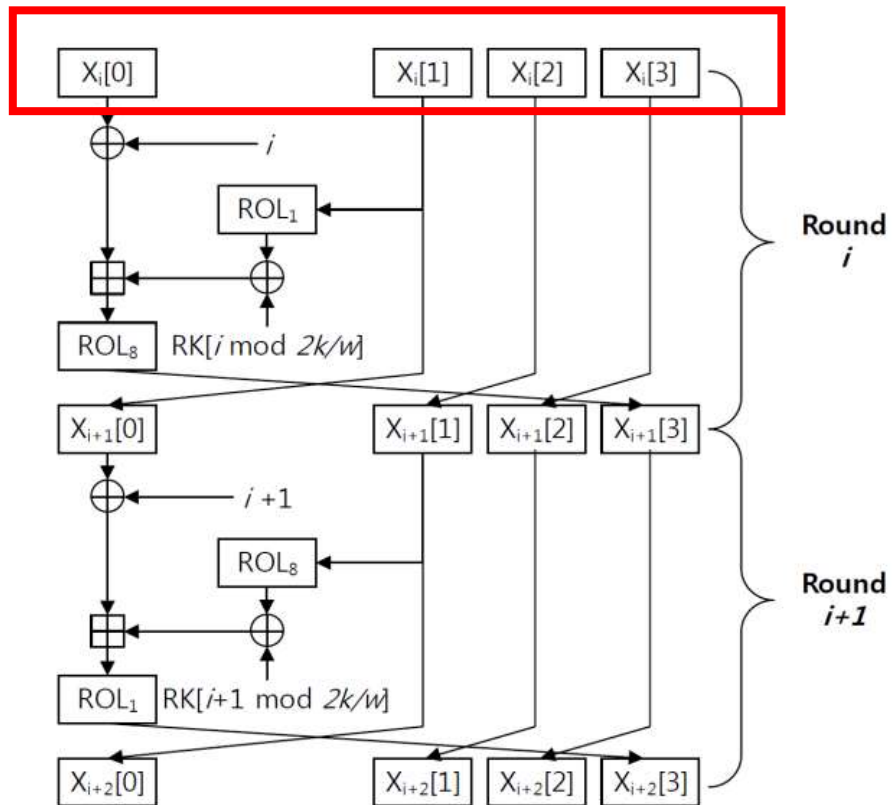
- Atmel AVR ATmega128 프로세서 환경에서 구현이 진행되었음.
 - 아두이노 우노에 적용되어 활발히 활용되고 있음.
 - Harvard 구조를 따르며 동작 주파수는 16MHz 이다.
 - 32개의 8비트 범용 레지스터를 가지고 있음.
- 암호화 운영방식 (가변키 / 고정키 암호화)
- 성능 평가 지표 RANK
 - NSA에서 도입한 지표로 $\frac{(\frac{10^6}{CPB})}{FLASH+2 \times RAM}$ 로 구성된다.
 - CPB – Cycles Per Byte, FLASH – 코드 저장 공간

3. CHAM 구조

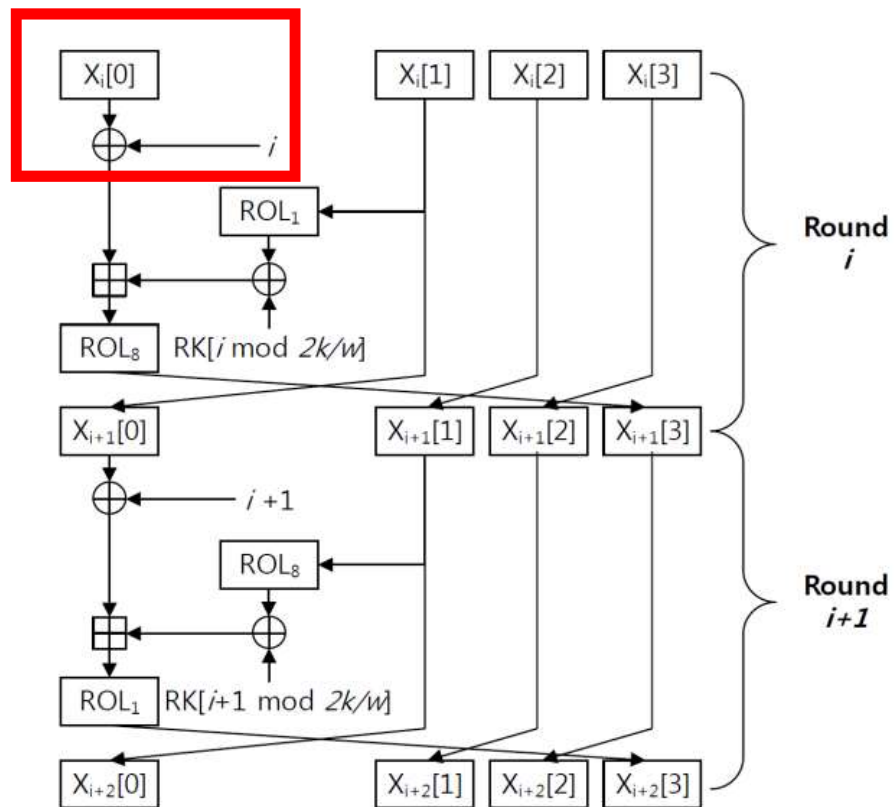


3. CHAM 구조

평문을 4등분해서 $X[0] \sim X[3]$ 까지 들어가게 됨.

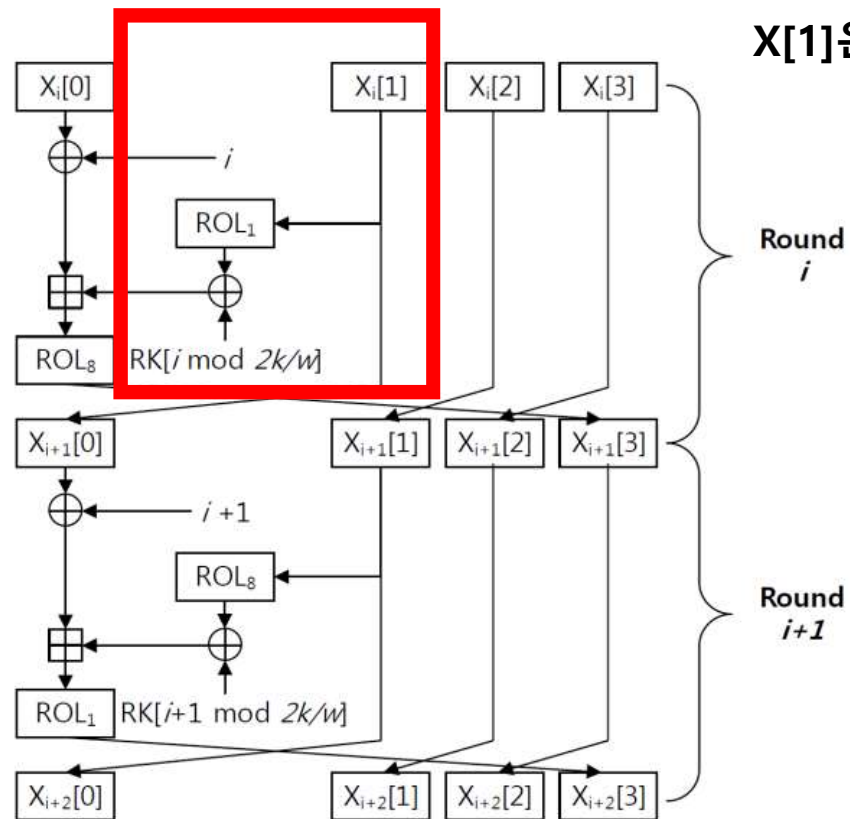


3. CHAM 구조



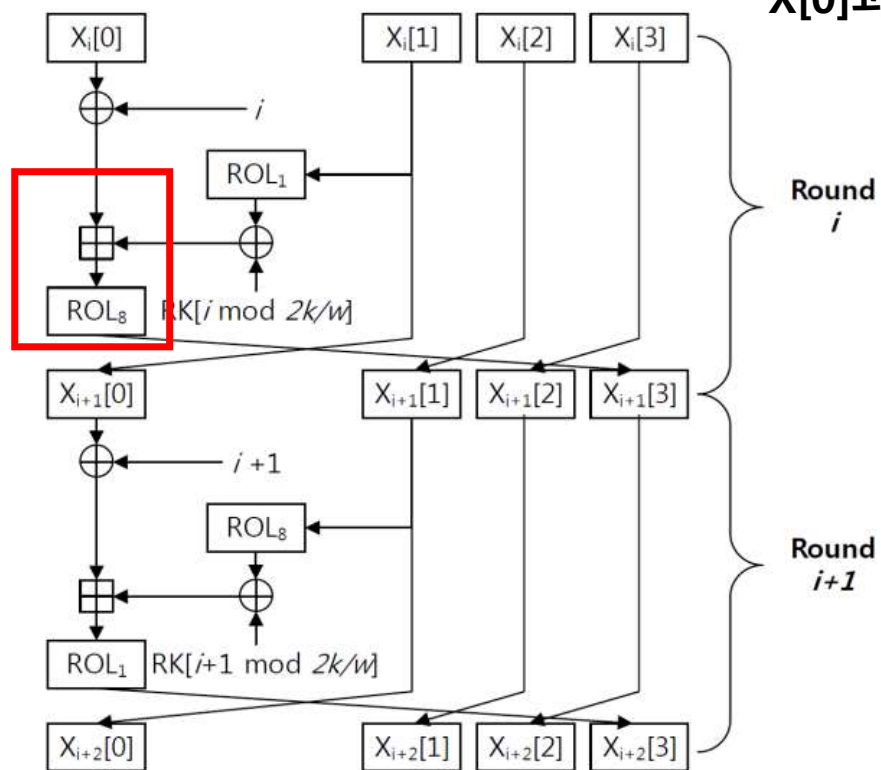
$X[0]$ 는 라운드 카운터와 XOR 연산을 진행함.

3. CHAM 구조



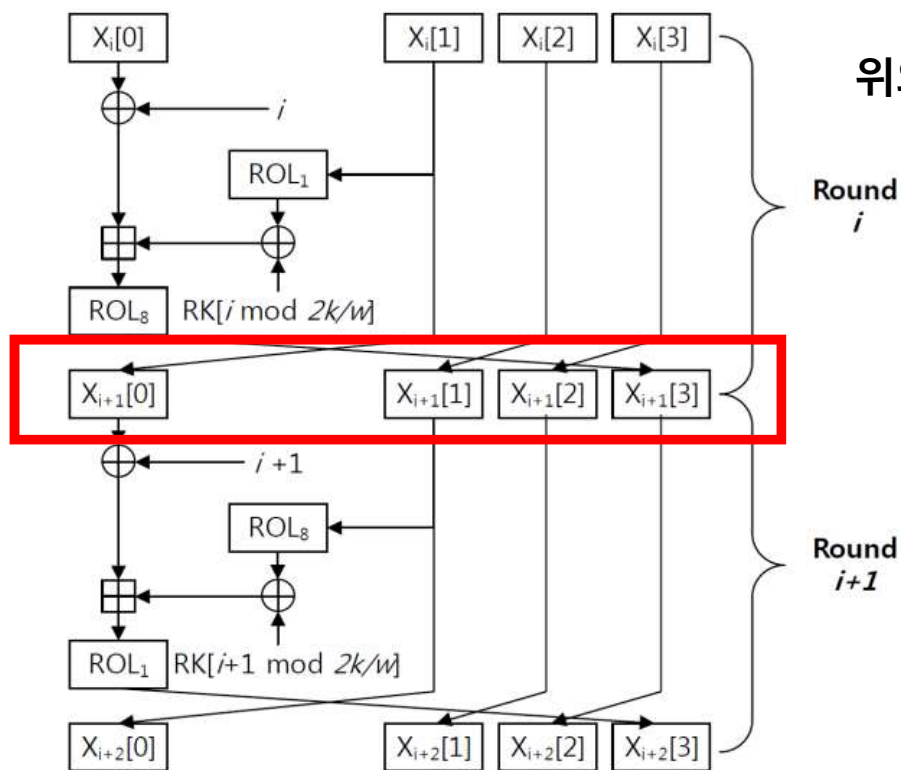
$X[1]$ 은 rotation left 1회 후, round key와 XOR연산한다.

3. CHAM 구조



$X[0]$ 과 $X[1]$ 을 더한 다음 rotation left 연산을 8회 수행한다.

3. CHAM 구조



$$X_i[0] \rightarrow X_{i+1}[3]$$

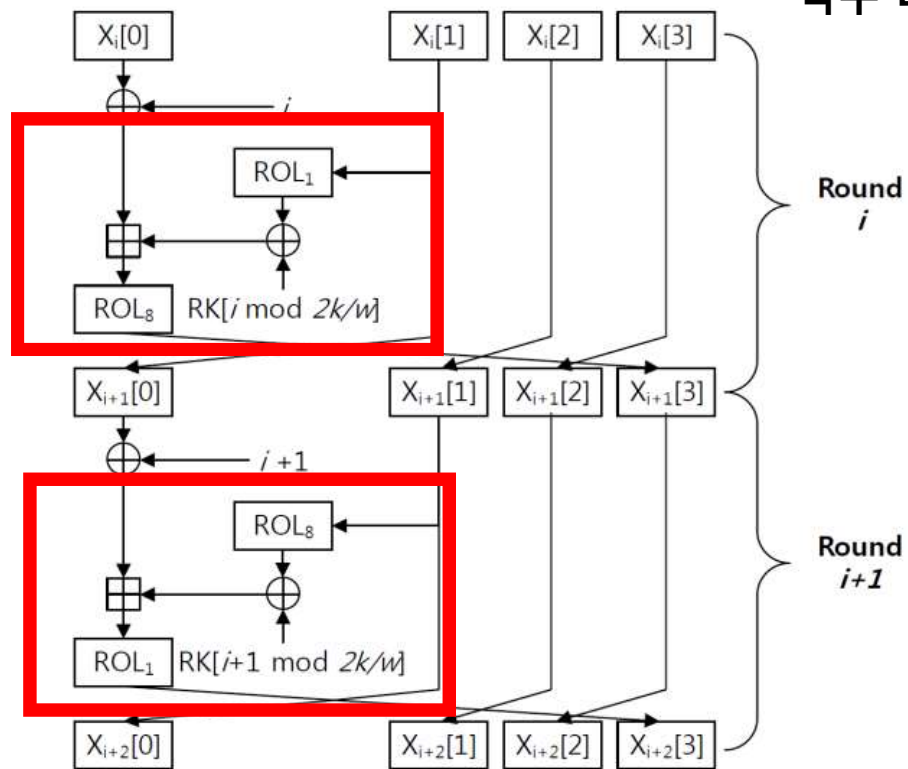
$$X_i[1] \rightarrow X_{i+1}[0]$$

$$X_i[2] \rightarrow X_{i+1}[1]$$

$$X_i[3] \rightarrow X_{i+1}[2]$$

위와 같이 한 워드씩 이동하여 저장된다.

3. CHAM 구조



짝수 라운드와 홀수 라운드는 rotation 연산 횟수만 다름.

3. 이전 CHAM 구현 결과

- ICISC'17에 발표된 논문에서는 'ADD', 'EOR', 'LSL', 'ROL'로 구성되어 있으며 4라운드만을 구현하여 이를 반복적으로 수행하는 구조로 구현되어 있다.
- 64/128, 128/128 -> 80라운드로 구성되어 있어 코드 크기가 약 1/20로 줄어든다.
- 128/256 -> 96라운드로 구성되어 있어 코드 크기가 약 1/24로 줄어든다.

4. 메모리 최적화 구현

- 라운드 키 접근 최적화
- 카운터 최적화
- 메모리 접근 최적화

4. 메모리 최적화 구현 – 라운드 키 접근 최적화

- 8-bit AVR 프로세서의 주소는 16비트를 가지며 2개의 8비트 레지스터를 합쳐서 접근하게 된다.
- 라운드 키를 반복적으로 접근하는데, 128비트 키의 경우 총 32바이트 256비트 키의 경우 64바이트가 사용된다.
- 키를 메모리 주소에서 하위주소가 0x00 번지가 오도록 정렬하면 키에 대한 접근을 1바이트 오프셋 연산으로 해결이 가능하다.

4. 메모리 최적화 구현

- 카운터 최적화
 - 64/128, 128/128 모드의 카운터 최대값은 80이고 128/256 모드의 카운터 최대값은 96.
 - 따라서 1바이트 레지스터만 사용하여 카운터를 관리한다.
- 메모리 접근 최적화
 - 메모리 접근 이전, 이후 연산을 활용 (X포인트 레지스터, Z포인트 레지스터 등)하여 수행하였음.
 - X-, X+, Z-, Z+

4. 메모리 최적화 구현 – Ver. 1

- 하나의 라운드만을 구현한 후 이를 반복적으로 수행하는 방법.
 - ex. 64/128의 경우 80 라운드를 진행하는데 1 라운드만을 구현하여 코드를 $\frac{1}{4}$ 만큼 줄일 수 있음.
- 홀수 라운드인지, 짝수 라운드인지 판단하는 부분이 있어야 한다.

Program	Odd round (R0 is odd)	Even round (R0 is even)
ANDI R0, 1	0x01	0x00
DEC R0	0x00	0xFF
COM R0	0xFF	0x00
ANDI R0, 7	0x07	0x00

Program	Odd round (R0 is odd)	Even round (R0 is even)
ANDI R0, 1	0x01	0x00
DEC R0	0x00	0xFF
ANDI R0, 7	0x00	0x07

4. 메모리 최적화 구현 – Ver. 1

- 홀수 라운드인지, 짝수 라운드인지 판단

Program	Odd round (R0 is odd)	Even round (R0 is even)
ANDI R0, 1	0x01	0x00
DEC R0	0x00	0xFF
COM R0	0xFF	0x00
ANDI R0, 7	0x07	0x00

Program	Odd round (R0 is odd)	Even round (R0 is even)
ANDI R0, 1	0x01	0x00
DEC R0	0x00	0xFF
ANDI R0, 7	0x00	0x07

Table 4. rotation by offset

STEP:

```
LSL X0  
ROL X1  
ADC X0, ZERO  
DEC COUNT  
CPI COUNT, 0  
BRGE STEP
```

4. 메모리 최적화 구현 – Ver. 1

- COUNT 수에 따라 반복되는 구조
- 8-bit 기반 rotation 연산을 생략할 수 없음.

Table 4. rotation by offset

STEP:
LSL X0
ROL X1
ADC X0, ZERO
DEC COUNT
CPI COUNT, 0
BRGE STEP

4. 메모리 최적화 구현 – Ver. 2

- 두개의 라운드(홀수, 짝수)를 구현한 후, 이를 반복적으로 수행하는 방법.
- 8비트 rotation 연산을 생략할 수 있다.
 - 하나의 레지스터가 8비트를 가지므로 8-bit left rotation을 진행하는 것과 두 레지스터를 swap하는 것이 같은 결과를 만든다.
- Rotation 연산을 효율적으로 수행하기 위해 2라운드가 끝난 후에 2 word 단위로 회전을 한 번에 수행할 수 있다.

Table 5. register alignment

MOV X4, X1
MOV X5, X0
MOVW X0, X2
MOVW X2, X6
MOVW X6, X8

5. 성능 평가

- 다음은 고정키 기반 암호화를 저사양 8-bit AVR프로세서 상에서 구현했을 때 얻을 수 있는 RANK 파라미터 비교이다.
- 이전 구현 결과는 27.9 RANK를 기록하였음.
- Ver. 2 의 경우 29.9 RANK를 달성함.

Table 6. Performance comparison using the rank metric on 8-bit AVR processors, under the fixed-key scenario.

Alg.	ROM (byte)	RAM (byte)	Time (cpb)	RANK
64-bit plaintext / 128-bit secret key				
Proposed CHAM ver. 2	152	3	211	29.9
SPECK [3]	218	0	154	29.8
CHAM [4]	202	3	172	27.9
SIMON [3]	290	0	253	13.6
HIGHT [3]	336	0	311	9.6
Proposed CHAM ver. 1	166	2	966	6.1
128-bit plaintext / 128-bit secret key				
Proposed CHAM ver. 2	270	13	187	18.0
CHAM [4]	362	16	148	17.1
SPECK [3]	460	0	171	12.7
LEA [4]	754	17	203	6.3
Proposed CHAM ver. 1	270	12	657	5.2
128-bit plaintext / 256-bit secret key				
Proposed CHAM ver. 2	302	13	223	13.6
CHAM [4]	396	16	177	13.2
SPECK [6]	476	0	181	11.6
Proposed CHAM ver. 1	302	12	786	3.9

5. 성능 평가

- Ver. 1의 경우 1라운드만을 구현하여 코드 크기를 줄일 수 있을 것으로 기대되었다.
- 하지만 CHAM알고리즘이 2 라운드마다 연산이 반복되는 매우 간단한 구조라 범용적으로 동작시키는 코드는 연산 속도와 코드 크기에서 모두 좋지 않은 성능을 도출함.
- Ver. 1에서 제시한 오프셋 프로그램 루틴은 다른 시큐어 프로그램에 활용이 가능하다.