8-bit AVR 환경에서 CHAM 최적 구현

IT정보공학과 신명수

제5회 2022

부채널정보분석 경진대회

한국정보보호학회 부채널분석연구회에서는
암호 구현 · 분석 기술의 발전 및 전문인력
양성을 위해 **부채널정보분석 경진대회**를
다음과 같이 개최합니다.

주 최 | 한국전자통신연구원, 한국인터넷진흥원
국가보안기술연구소, 군사안보지원사령부

주 관 | 한국정보보호학회 부채널분석연구회

대회 내용 및 참여 방법 안내

구 분		내용 및 참여방법	일정 및 제출처
학부 (개인)	최적화 구현	**블록암호 CHAM, PIPO 고속화 구현** - ATmega128 상에서 구현속도로 평가 - 안내사항 및 프로젝트 템플릿 다운로드 주소 https://scacontest.com/notifications	• 제출 마감일 : 22.08.31 • 제출처 : 국민대 서석충 교수 (scseo@kookmin.ac.kr)
	부채널 분석	**부채널분석 CTF (Capture The Flag)** - 문제 별 암호키 해독 및 점수획득 - 참여 안내 및 대회 홈페이지 https://scacontest.com	• 대회 일정 : 22.08.01~22.08.31 • 문제 공지 : 8월 1일 (월) 12:00

목차

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.

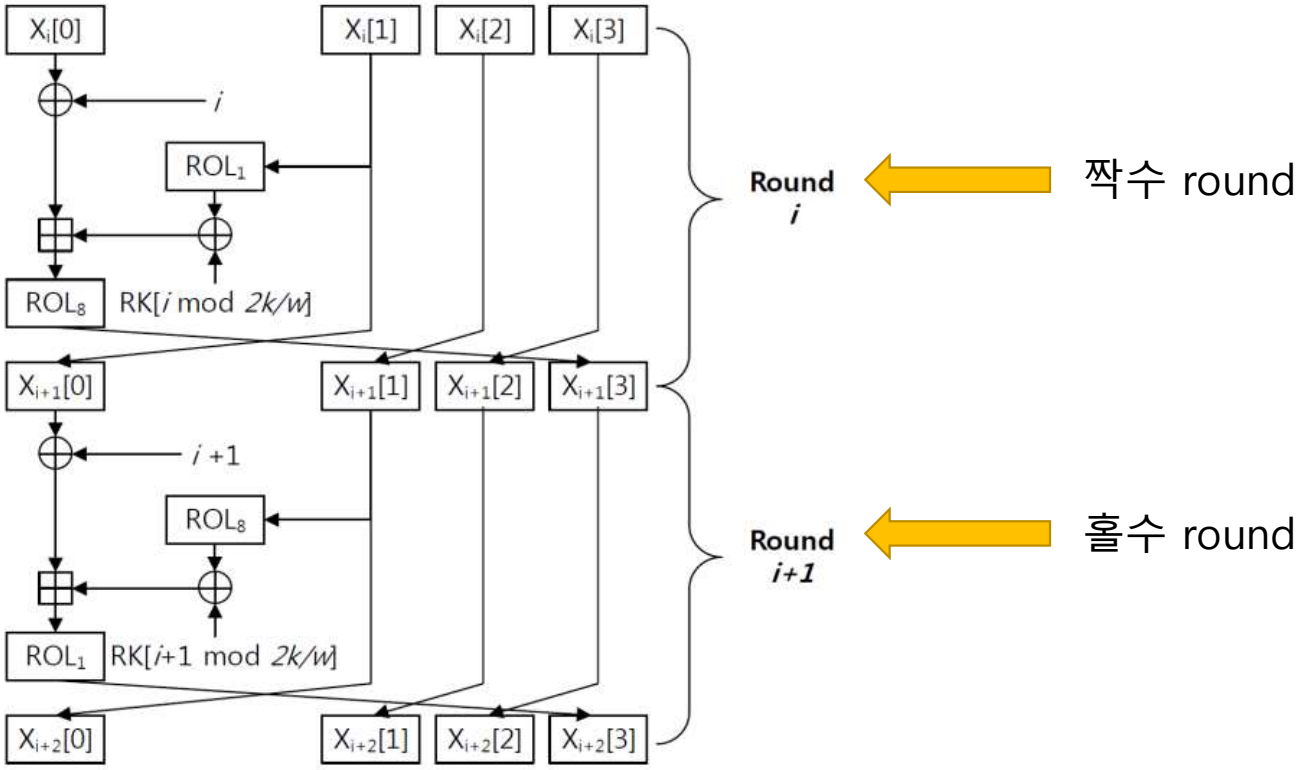
- Stateless 라운드 키 기법을 사용함.
  - 키 저장 공간을 줄일 수 있다.

- 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 : clock cycle

3. CHAM 구조

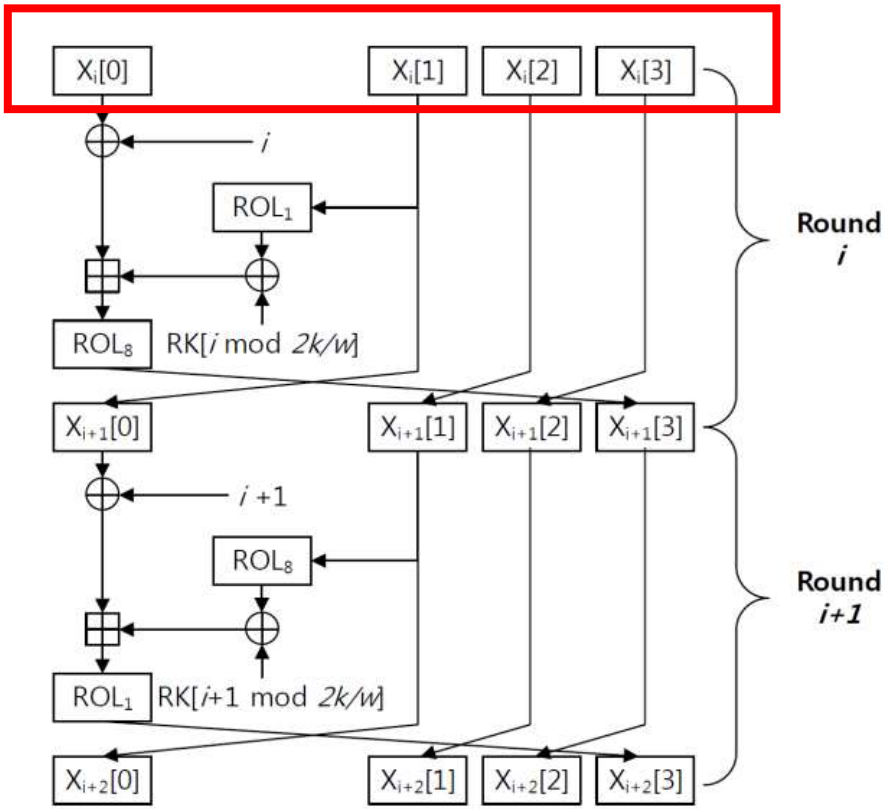


짝수 round

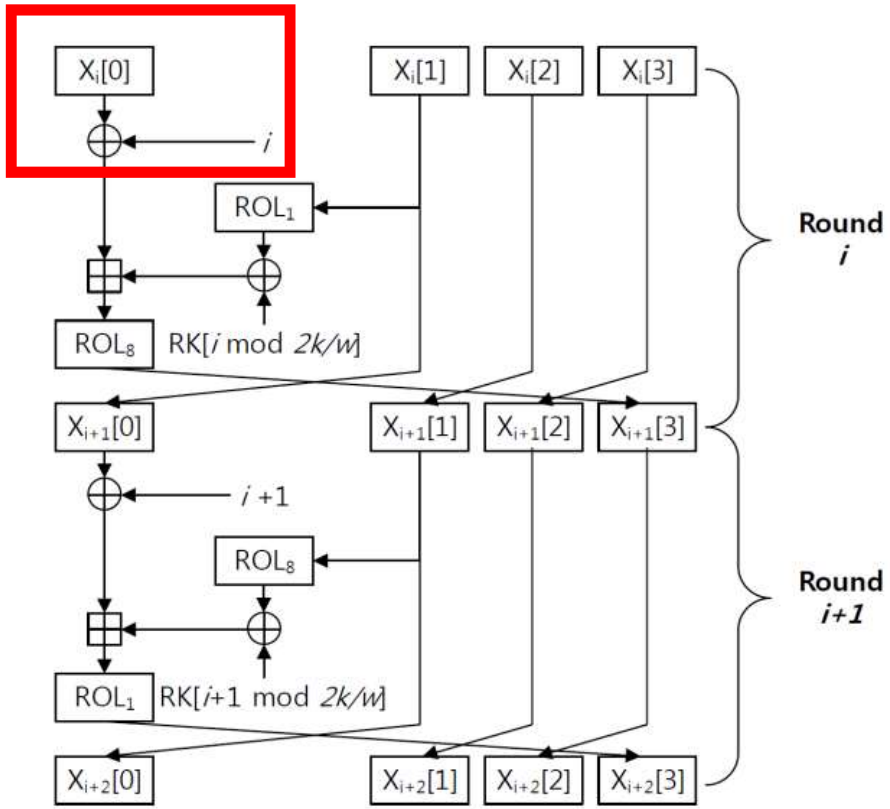
홀수 round

3. CHAM 구조

평문을 4등분해서 X[0]~X[3]까지 들어가게 됨.

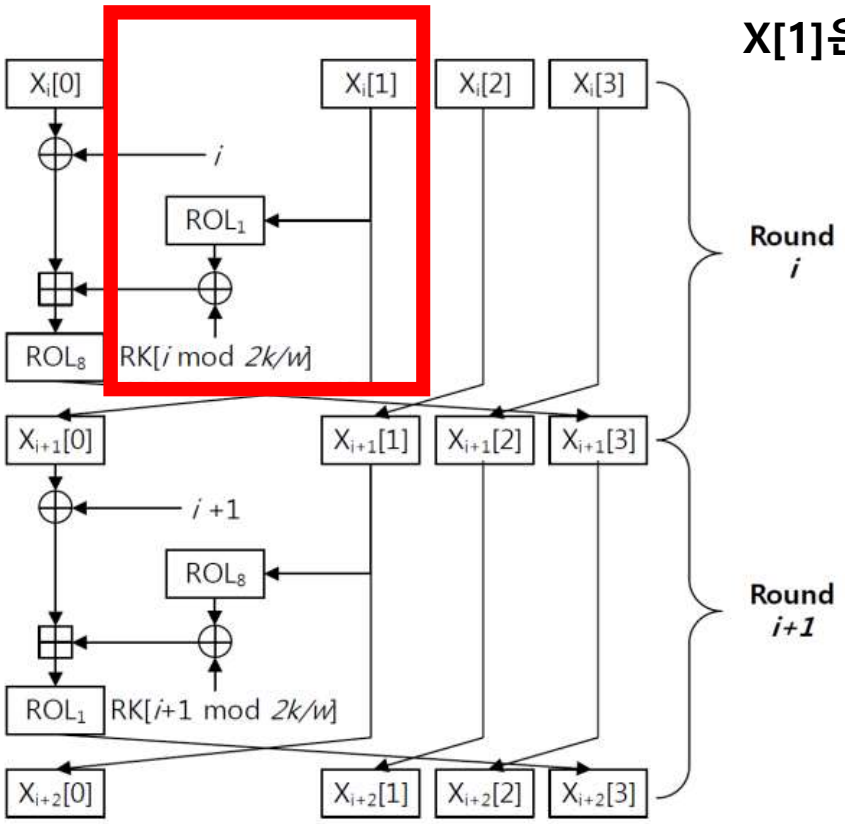


3. CHAM 구조



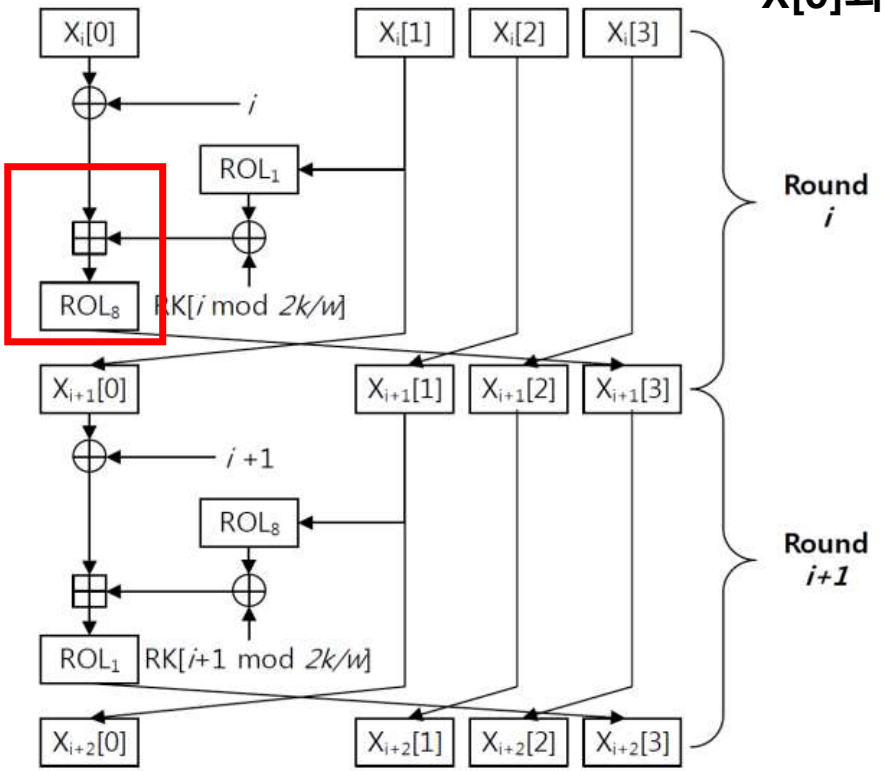
X[0]는 라운드 카운터와 XOR 연산을 진행함.

3. CHAM 구조



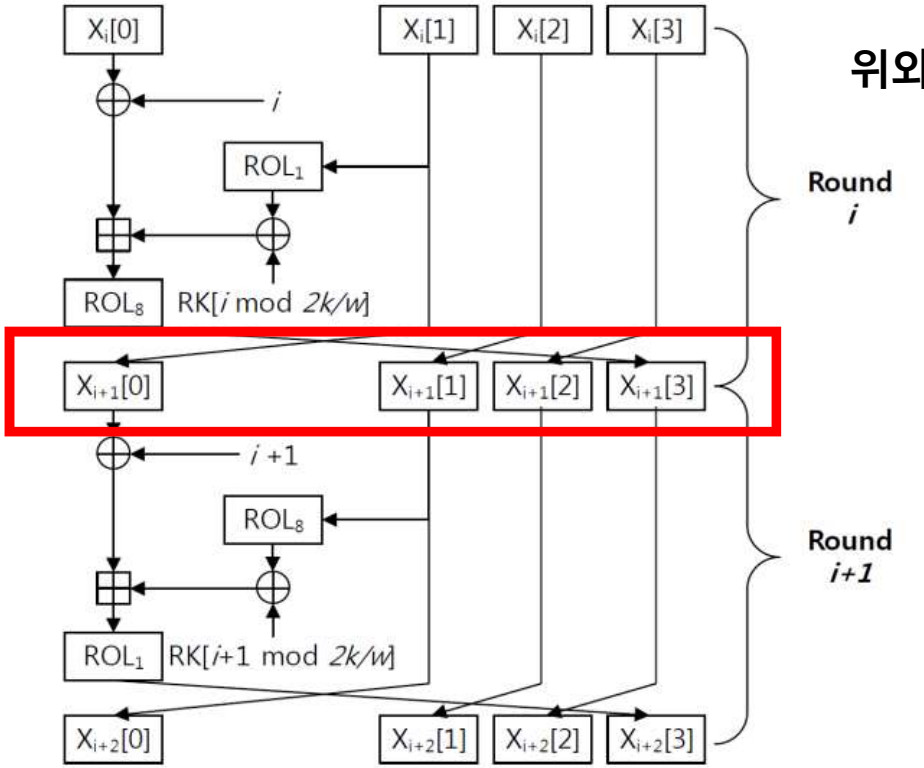
X[1]은 rotation left 1회 후, round key와 XOR연산한다.

3. CHAM 구조



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

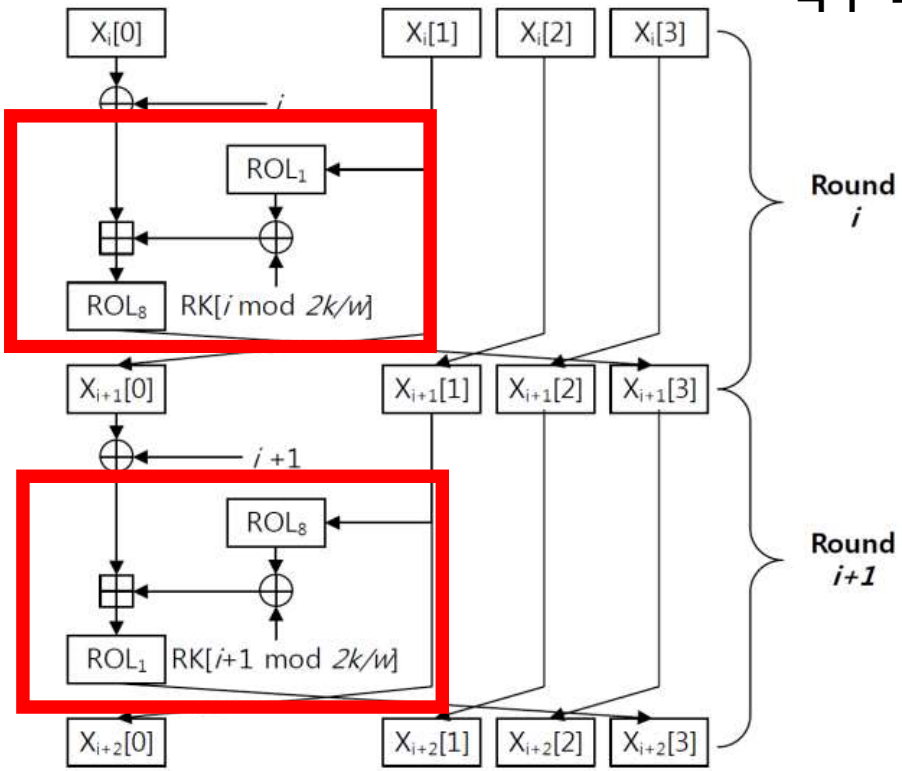
3. CHAM 구조



$$X_i[0] \rightarrow X_{i+1}[3]$$
$$X_i[1] \rightarrow X_{i+1}[0]$$
$$X_i[2] \rightarrow X_{i+1}[1]$$
$$X_i[3] \rightarrow X_{i+1}[2]$$

위와 같이 한 워드씩 이동하여 저장된다.

3. CHAM 구조



짝수 라운드와 홀수 라운드는 rotation 연산 횟수만 다름.

4. 구현 및 평가 – 기본 구현

- 홀수 라운드와 짝수 라운드를 구현하여 44회 동작하도록 만드는 방법.

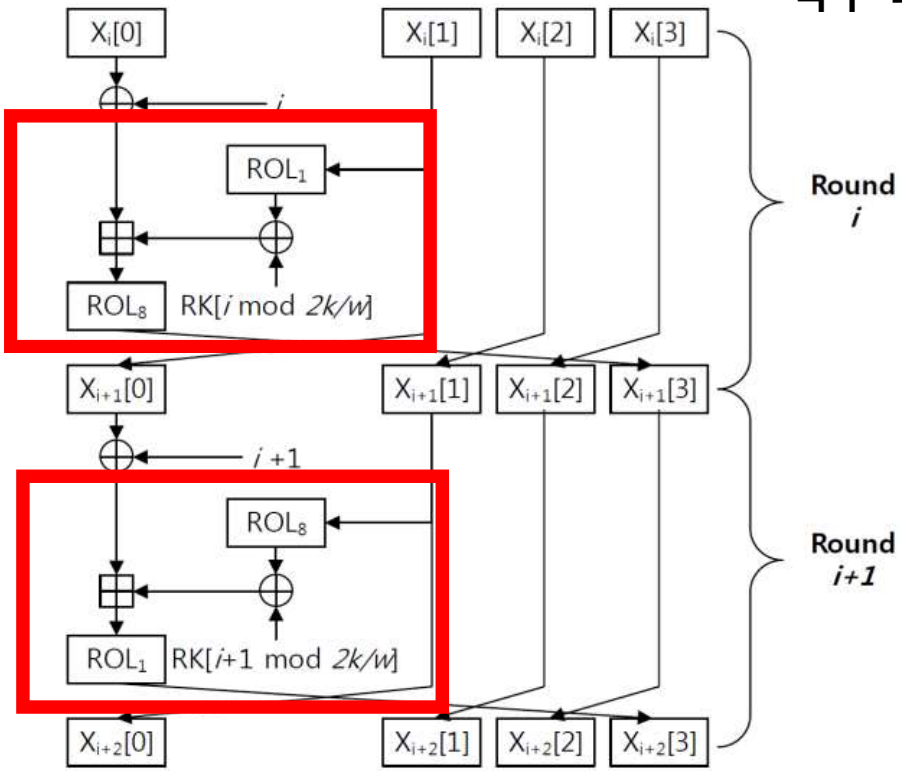
```
// 13 operations
.macro ODD_ROUND
    MOVW TM0, PT10
    ROL16 TM0, TM1

    // XOR TMP, RK
    LD RK, Z+
    EOR TM0, RK
    LD RK, Z+
    EOR TM1, RK

    // XOR PT[0], RC
    EOR PT00, RC

    // PT[0] ^ TMP
    ADD PT00, TM0
    ADC PT01, TM1

    // ROL16(PT[0], 8)
    SWAP16 PT00, PT01, TM0

    INC RC
.endm
```

```
// TMP = PT[1]
// ROL16(TMP, 1)
```

```
// 13 operations
.macro EVEN_ROUND
    MOV TM1, PT20
    MOV TM0, PT21

    LD RK, Z+
    EOR TM0, RK
    LD RK, Z+
    EOR TM1, RK

    EOR PT10, RC

    ADD PT10, TM0
    ADC PT11, TM1

    ROL16 PT10, PT11

    INC RC
.endm
```

```
LOOP:
ANDI R30, 31
ODD_ROUND
EVEN_ROUND
// ROTATE
SWAP64 PT00, PT10, PT20, PT30, TM0

CPSE RC, CNT
RJMP LOOP
```

4. 구현 및 평가 – Rotation 줄이기



짝수 라운드와 홀수 라운드는 rotation 연산 횟수만 다름.

4. 구현 및 평가 – rotation 줄이기

- ROL

```
.macro STORED_ROUND1
    INC RC                  // RC += 1
    MOVW TM0, PT10          // TMP = PT[1]

    ROL16 TM0, TM1          // ROL16(TMP, 1)

    EOR PT00, RC            // PT[0] ^= RC

    EOR TM0, RK10           // TMP ^= RK
    EOR TM1, RK11

    ADD PT00, TM0           // PT[0] + TMP
    ADC PT01, TM1

.endm
```

```
.macro STORED_ROUND5
    INC RC                  // RC += 1
    MOVW TM0, PT10          // TMP = PT[1]

    ROL16 TM0, TM1          // ROL16(TMP, 1)

    EOR PT01, RC            // PT[0](SWAPED) ^= RC

    EOR TM0, RK50           // TMP ^= RK
    EOR TM1, RK51

    ADD PT01, TM0           // PT[0](SWAPED) += TMP
    ADC PT00, TM1

.endm
```

4. 구현 및 평가 – 8 round 구현

- 1~8 라운드를 모두 구현한 후 8개의 라운드를 11번 반복하는 방법

```
.macro EIGHT_ROUNDS
    ROUND1
    ROUND2
    ROUND3
    ROUND4
    ROUND5
    ROUND6
    ROUND7
    ROUND8
    SUBI R30, 16
.endm
```

```
.macro ROUND1
    MOVW TM0, PT10          // TMP = PT[1]

    ROL16 TM0, TM1          // ROL16(TMP, 1)

    EOR PT00, RC            // PT[0] ^= RC

    EOR_RK TM0, TM1         // TMP ^= RK

    ADD PT00, TM0           // PT[0] + TMP
    ADC PT01, TM1

    INC RC                  // RC += 1
.endm
```

4. 구현 및 평가 – 8 round 구현 + RK저장

- 16개의 레지스터에 라운드키의 절반을 저장하는 방법

```
#define RK10 R2
#define RK11 R3
#define RK20 R4
#define RK21 R5
#define RK30 R6
#define RK31 R7
#define RK40 R8
#define RK41 R9
#define RK50 R10
#define RK51 R11
#define RK60 R12
#define RK61 R13
#define RK70 R14
#define RK71 R15
#define RK80 R16
#define RK81 R17
```

```
.macro STORED_ROUND1
    MOVW TM0, PT10              // TMP = PT[1]

    ROL16 TM0, TM1              // ROL16(TMP, 1)

    EOR PT00, RC                // PT[0] ^= RC

    EOR TM0, RK10               // TMP ^= RK
    EOR TM1, RK11

    ADD PT00, TM0               // PT[0] + TMP
    ADC PT01, TM1

    INC RC                      // RC += 1
.endm
```

```
.macro STORED_EIGHT_ROUNDS
    STORED_ROUND1
    STORED_ROUND2
    STORED_ROUND3
    STORED_ROUND4
    STORED_ROUND5
    STORED_ROUND6
    STORED_ROUND7
    STORED_ROUND8
.endm
```

```
EIGHT_ROUNDS
STORED_EIGHT_ROUNDS
EIGHT_ROUNDS
STORED_EIGHT_ROUNDS
EIGHT_ROUNDS
STORED_EIGHT_ROUNDS
EIGHT_ROUNDS
STORED_EIGHT_ROUNDS
EIGHT_ROUNDS
STORED_EIGHT_ROUNDS
```

4. 구현 및 평가 – rotation 줄이기

```
void cham64_encrypt_8RNR(uint8_t* dst, const uint8_t* src, const uint8_t* rks);          // CLOCK CYCLE 1340
void cham64_encrypt_8R(uint8_t* dst, const uint8_t* src, const uint8_t* rks);            // CLOCK CYCLE 1516
void cham64_encrypt_4R(uint8_t* dst, const uint8_t* src, const uint8_t* rks);            // CLOCK CYCLE 1560
void cham64_encrypt_2R(uint8_t* dst, const uint8_t* src, const uint8_t* rks);            // CLOCK CYCLE 1912
void cham64_encrypt2(uint8_t* dst, const uint8_t* src, const uint8_t* rks);              // CLOCK CYCLE 1479
void cham64_encrypt_8RNR2(uint8_t* dst, const uint8_t* src, const uint8_t* rks);         // CLOCK CYCLE 1304
void cham64_encrypt_8R_storedRK(uint8_t* dst, const uint8_t* src, const uint8_t* rks);   // CLOCK CYCLE 1201
void cham64_encrypt_storedRK(uint8_t* dst, const uint8_t* src, const uint8_t* rks);      // CLOCK CYCLE 1197
void cham64_encrypt_storedRK2(uint8_t* dst, const uint8_t* src, const uint8_t* rks);     // CLOCK CYCLE 1197
void cham64_encrypt_ns(uint8_t* dst, const uint8_t* src, const uint8_t* rks);            // CLOCK CYCLE 1193
```