Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.24, NO.4, Aug. 2014 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2014.24.4.585

UICC 16bit 상에서의 LEA 구현 적합성 연구*

김 현 일,† 박 철 희, 홍 도 원,‡ 서 창 호
공주대학교

A LEA Implementation study on UICC-16bit*

Hyun-il Kim,† Cheolhee Park, Dowon Hong,‡ Changho Seo
Kongju National University

요 약

본 논문에서는 2013년 12월 18일에 국내표준(TTA-KO-12.0223)으로 제정된 블록암호 알고리즘 LEA(Lightweight Encryption Algorithm)[1]에 대한 UICC-16bit 상에서의 최적 구현에 대해 연구한다. LEA의 전체적인 구조를 설명하며, 키 스케줄 과정에서 고정된 상수를 통해 미리계산이 가능하여 효율성을 높일 수 있다는 점을 제시하며 이러한 개선된 키 스케줄링을 통하여 얻게되는 상수 table을 이용하여 UICC-16비트 상에 구현하였다. 또한 UICC-16bit 상에서 국내 표준 블록암호 ARIA와의 성능 비교를 통해 LEA블록암호의 우수성을 평가하였다.

ABSTRACT

In this paper, we study the LEA[1] block cipher system in UICC-16bit only. Also, we explain a key-schedule function and encryption/decryption structures, propose an advanced modified key-scheduling, and perform LEA in UICC-16bit that we proposed advanced modified key-scheduling. Also, we compare LEA with ARIA that proposed domestic standard block cipher, and we evaluate the efficiency on the LEA algorithm.

Keywords: UICC-16bit, LEA(Lightweight Encryption Algorithm), Key scheduling

I. 서 론

현대사회는 컴퓨터와 인터넷 사용의 막대한 증가로 인하여 정보보안의 중요성이 갈수록 커지고 있다. 보안의 3대 요소인 기밀성(Confidentiality), 무결성(Integrity), 가용성(Accessibility) 등의 서비스를 제공하는 것을 정보보호라 하며, 암호화 및 인증 등을 수행하는 암호 시스템 사용은 정보보호 서비스의 가장 일반적인 방법이라 할 수 있다. 암호 시스템에 사용되는 암호는 크게 대칭키와 비대칭키로 나뉜다. 대칭키 암호는 두 사용자 A와 B가 서로 같은 키를 이용해 암호화 및 복호화 과정을 수행하는 방식이며, 현재에도 데이터 암호화나 메시지 무결성 검사 등에 많이 사용되는 방식이다.[2] 대칭키 암호는 스트림 암호(Stream cipher)와 블록 암호(Block cipher)로 나뉘며 현재 대표적인 블록암호로는 3-DES(Triple Data Encryption Standard)[3], AES(Advanced Encryption Standard)[4]등이 있고 우리나라는 자체적으로 개발한 SEED[5]와 ARIA[6]등이 있다. 하지만 국내에서는 고속, 경량, 저전력을 요구하는 암호 시스템을 필요로 하게 되었고 이에 따라 32비트 플랫폼에서 효율적으로 구동되는

접수일(2014년 5월 9일), 수정일(2014년 7월 14일), 게재확정일(2014년 7월 15일)
* 본 연구는 교육부와 한국연구재단의 지역혁신 창의인력 양성사업(No.2013H1138A2032077)과 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대 정보·컴퓨팅기술 개발사업(No.2011-0029926)의 지원을 받아 수행된 연구임.
† 주저자, hyunil89@kongju.ac.kr
‡ 교신저자, dwhong@kongju.ac.kr(Corresponding author)

LEA(Lightweight Encryption Algorithm)알고리즘을 제안하였다.[1]

LEA는 128비트 블록 단위로 암복호화를 수행하고 128, 192, 256비트의 비밀키를 사용할 수 있으며 라운드함수는 Addtion, Rotation, XOR의 연산만으로 구성되어있어 연산 속도가 다른 블록암호에 비해 빠르고 키스케줄 과정이 간단하며 S-box를 사용하지 않으므로 경량 구현이 가능하다.[1] 또한 COSIC은 현존하는 블록암호에 대한 공격들에 대하여 안전성을 가지고 있다고 분석하였다.[7] 따라서 이러한 장점을 가진 LEA는 여러 임베디드 시스템에서 유용하게 쓰일 가능성이 매우 높다. 현대 고사양 임베디드 시스템들은 마이크로프로세서 고성능화로 인하여 32비트 프로세서를 사용하고 일부 제품은 64비트 프로세서를 사용하는 경우도 많다.[8] 하지만, 저사양 임베디드 시스템에서는 효율성을 위하여 여전히 8비트나 16비트 프로세서를 많이 사용하고 있다.[9] 이에 따라 32비트에서 뿐만 아니라 16비트에서의 암호모듈 구현은 필요할 수 밖에 없으며 기본적인 데이터를 암호화하는 블록암호는 암호모듈의 핵심이라 할 수 있다. 따라서 본 논문은 안전하고, 저용량에 고효율을 만족하는 LEA를 UICC-16bit 상에서의 구현을 목표로 하였다.

본 논문의 2장에서는 LEA의 암/복호화 과정 및 키 스케줄링 과정을 설명하고, 3장에서는 개선된 키 스케줄링과 함께 UICC-16bit에서의 구현 과정을 설명하며 ARIA와의 효율성 비교 후 4장에서 결론을 내린다.

II. LEA 알고리즘 [1, 9]

LEA의 규격은 다음 Table 1. 과 같다.

Table 1. Specification of LEA

LEA- k	Size of block	Key length	Number of rounds
LEA-128	128	128	24
LEA-192	128	192	28
LEA-256	128	256	32

LEA는 128비트 블록 단위로 암/복호화를 하며, 128비트, 192비트, 256비트의 가변 키 길이를 가지며 라운드 수는 각각 차례대로 24라운드, 28라운드, 32라운드이다. 먼저 암/복호화시 사용되는 기호에 대하여 정의한다.

2.1 기호 정의

P : 평문(Plaintext). 32비트 4워드로 이루어져 있다. (P = P[0]||P[1]||P[2]||P[3])

C : 암호문(Ciphertext). 32비트 4워드로 이루어져 있다. (C = C[0]||C[1]||C[2]||C[3])

E : 암호화 함수. D : 복호화 함수.

K : 메인키. T : 라운드 키 임시 변수.

$RK_i[j]$: 라운드 키. 각 라운드 당 32비트 6워드로 이루어져있다.

$\delta[i]$: 키 스케줄시 사용되는 라운드 상수.

X_i : i번째 라운드의 값. 암/복호화 과정시 중간값이다. ($X_i = X_i[0] \| X_i[1] \| X_i[2] \| X_i[3]$)

$x \oplus y$: 비트단위 XOR.

$x \boxplus y$: 모듈로 2^{32}에서의 덧셈.

$x \boxminus y$: 모듈로 2^{32}에서의 뺄셈.

$ROTL_i(x)$: x를 i비트 왼쪽으로 순환이동.

$ROTR_i(x)$: x를 i비트 오른쪽으로 순환이동.

2.2 암호화 및 복호화 과정

LEA의 암호화 과정은 키 스케줄링을 통하여 얻은 192비트 라운드 키를 이용해 암호화한다. 128비트 블록을 32비트 4워드로 나누어 덧셈, 순환이동, XOR 연산 후 다음 라운드로 진행한다. 암호화 과정을 그림으로 나타내면 Fig.1.과 같고, 식으로 나타내면 Fig.2.와 같다.

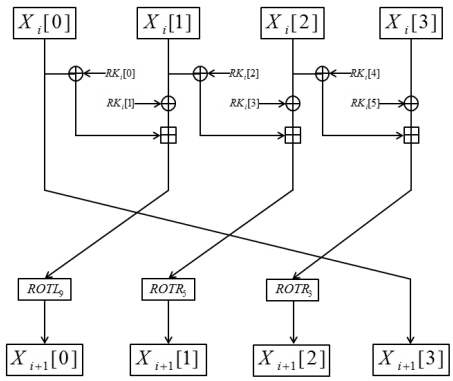


Fig. 1. An encryption round function on i-round

for i = 0 to 23;
{
$X_{i+1}[0] \leftarrow ROTL_9(X_i[0] \oplus RK_i[0]) \boxplus (X_i[1] \oplus RK_i[1]))$;
$X_{i+1}[1] \leftarrow ROTR_5(X_i[1] \oplus RK_i[2]) \boxplus (X_i[2] \oplus RK_i[3]))$;
$X_{i+1}[2] \leftarrow ROTR_3(X_i[2] \oplus RK_i[4]) \boxplus (X_i[3] \oplus RK_i[5]))$;
$X_{i+1}[3] \leftarrow X_i[0]$;
}

Fig. 2. An encryption schedule

LEA의 복호화 과정은 키 스케줄링을 통하여 얻은 192비트 라운드 키를 암호화와 반대 방향으로 이용하여 복호화한다. 128비트 블록을 32비트 4워드로 나누어 뺄셈, 순환이동, XOR연산 후 다음 라운드로 진행한다. 복호화 과정을 그림으로 나타내면 Fig.3.과 같고, 식으로 나타내면 Fig.4.와 같다.

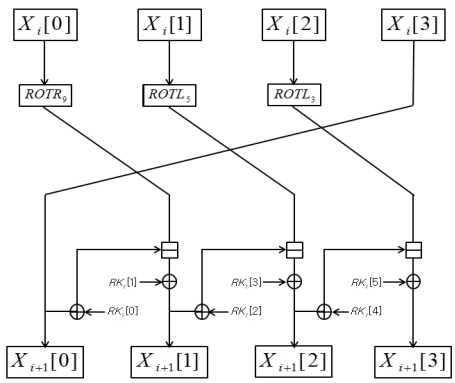


Fig. 3. A decryption round function on i-round

for i = 0 to 23;
{
$X_{i+1}[0] \leftarrow X_i[3]$;
$X_{i+1}[1] \leftarrow (ROTR_9(X_i[0]) \boxminus (X_{i+1}[0] \oplus RK_{23-i}[0])) \oplus RK_{23-i}[1]))$;
$X_{i+1}[2] \leftarrow (ROTL_5(X_i[1]) \boxminus (X_{i+1}[1] \oplus RK_{23-i}[2])) \oplus RK_{23-i}[3]))$;
$X_{i+1}[3] \leftarrow (ROTL_3(X_i[2]) \boxminus (X_{i+1}[2] \oplus RK_{23-i}[4])) \oplus RK_{23-i}[5]))$;
}

Fig. 4. A decryption schedule

2.3 라운드 키 생성과정

라운드 키 생성과정은 8개의 32비트 상수를 사용하며 키 스케줄링 함수를 거쳐 192비트 라운드 키를 생성한다. 이 때 사용되는 상수는,

$\delta[0] = 0xc3efe9db$, $\delta[1] = 0x44626b02$,
$\delta[2] = 0x79e27c8a$, $\delta[3] = 0x78df30ec$,
$\delta[4] = 0x715ea49e$, $\delta[5] = 0xc785da0a$,
$\delta[6] = 0xe04ef22a$, $\delta[7] = 0xe5c40957$

이며, LEA-128은 $\delta[0] \sim \delta[3]$, LEA-192는 $\delta[0] \sim \delta[5]$, LEA-256은 $\delta[0] \sim \delta[7]$ 의 상수를 사용하며 모듈로 연산과 순환이동을 거친 후 나누어진 메인 키에 덧셈연산을 취한 후 다시 순환이동을 거쳐 최종 라운드 키가 된다. 각각 식으로 나타내면,

〈LEA-128〉
T〔4〕 ← K
for i = 0 to 23;
{
$T[0] \leftarrow ROTL_1(T[0] \boxplus ROTL_i(\delta[i \bmod 4]))$;
$T[1] \leftarrow ROTL_3(T[1] \boxplus ROTL_{i+1}(\delta[i \bmod 4]))$;
$T[2] \leftarrow ROTL_6(T[2] \boxplus ROTL_{i+2}(\delta[i \bmod 4]))$;
$T[3] \leftarrow ROTL_{11}(T[3] \boxplus ROTL_{i+3}(\delta[i \bmod 4]))$;
$RK_i \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$;
}

〈LEA-192〉
T〔6〕 ← K
for i = 0 to 27;
{
$T[0] \leftarrow ROTL_1(T[0] \boxplus ROTL_i(\delta[i \bmod 6]))$;
$T[1] \leftarrow ROTL_3(T[1] \boxplus ROTL_{i+1}(\delta[i \bmod 6]))$;
$T[2] \leftarrow ROTL_6(T[2] \boxplus ROTL_{i+2}(\delta[i \bmod 6]))$;
$T[3] \leftarrow ROTL_{11}(T[3] \boxplus ROTL_{i+3}(\delta[i \bmod 6]))$;
$T[4] \leftarrow ROTL_{13}(T[4] \boxplus ROTL_{i+4}(\delta[i \bmod 6]))$;
$T[5] \leftarrow ROTL_{17}(T[5] \boxplus ROTL_{i+5}(\delta[i \bmod 6]))$;
$RK_i \leftarrow (T[0], T[1], T[2], T[3], T[4], T[5])$;
}

〈LEA-256〉
T〔6〕 ← K
for i = 0 to 31;
{
$T[6i \bmod 8] \leftarrow$
$ROTL_1(T[6i \bmod 8] \boxplus ROTL_i(\delta[i \bmod 6]))$;
$T[6i+1 \bmod 8] \leftarrow$
$ROTL_3(T[6i+1 \bmod 8] \boxplus ROTL_{i+1}(\delta[i \bmod 6]))$;
$T[6i+2 \bmod 8] \leftarrow$
$ROTL_6(T[6i+2 \bmod 8] \boxplus ROTL_{i+2}(\delta[i \bmod 6]))$;
$T[6i+3 \bmod 8] \leftarrow$
$ROTL_{11}(T[6i+3 \bmod 8] \boxplus ROTL_{i+3}(\delta[i \bmod 6]))$;
$T[6i+4 \bmod 8] \leftarrow$
$ROTL_{13}(T[6i+4 \bmod 8] \boxplus ROTL_{i+4}(\delta[i \bmod 6]))$;
$T[6i+5 \bmod 8] \leftarrow$

$ROTL_{17}(T[6i+5 \bmod 8] \boxplus ROTL_{i+5}(\delta[i \bmod 6]));$
$RK_i \leftarrow$
$(T[6i \bmod 8], T[6i+1 \bmod 8], T[6i+2 \bmod 8],$
$T[6i+3 \bmod 8], T[6i+4 \bmod 8], T[6i+5 \bmod 8]);$
}
이다.

III. UICC-16bit 상에서의 구현

3.1 효율적인 키 스케줄링 과정

LEA 키 스케줄링 과정에서 LEA-k는 k비트 비밀키를 이용해 192비트 라운드 키를 추출한다. 이 때 추출되는 라운드 키 192비트는 32비트 6워드로 이루어져 있으며 각 워드는 $ROTL_n(T[j] \boxplus ROTL_{i+j}(\delta[i \bmod k]))$라는 식으로 나타낼 수 있다. 이 식에서 덧셈 연산 전의 $T[j]$는 비밀키에 따라 변경되는 가변값이 되고, $ROTL_i(\delta[i \bmod k])$은 고정된 상수 δ를 이용하기 때문에 비밀키 값이 변경되어도 변하지 않는 값이므로 미리 계산하여 상수화가 가능하다. LEA-128을 기준으로,

δ_i에서 i를 순환이동의 횟수라 하면,
$i=0$: $\delta_0[0], \delta_1[0], \delta_2[0], \delta_3[0]$
$i=1$: $\delta_1[1], \delta_2[1], \delta_3[1], \delta_4[1]$
$i=2$: $\delta_2[2], \delta_3[2], \delta_4[2], \delta_5[2]$
$i=3$: $\delta_3[3], \delta_4[3], \delta_5[3], \delta_6[3]$
$i=4$: $\delta_4[0], \delta_5[0], \delta_6[0], \delta_7[0]$
$i=5$: $\delta_5[1], \delta_6[1], \delta_7[1], \delta_8[1]$
$\circ$
$\circ$
$\circ$
$i=k$: $\delta_k[i \bmod 4], \delta_{k+1}[i \bmod 4],$
$\delta_{k+2}[i \bmod 4], \delta_{k+3}[i \bmod 4]$

즉, $i=[i(\bmod 4)]$ 일 때의 값들을 k번 왼쪽으로 순환이동 하면 된다. LEA-192는 모듈로 6에서 이루어지므로 6라운드씩 반복되며, LEA-256은 모듈로 8에서 이루어지므로 8라운드씩 반복된다. 키 길이당 미리 계산된 상수는 Table 3.과 같다.

16비트 환경에서는 16×24×8(LEA-128 기준) 만큼의 상수가 들어가게 된다. 각 LEA당 table의 크기는 Table 2.와 같다.

Table 2. Code size of Pre-computed constants

LEA-k	size of constant table
LEA-128	32×24×4=3072 bit=0.375 KB (16bit - 16×24×8)
LEA-192	32×28×6=5376 bit=0.656 KB (16bit - 16×28×12)
LEA-256	32×32×6=6144 bit=0.75 KB (16bit - 16×32×12)

3.2 UICC 상의 구현 및 ARIA와의 효율성 비교

UICC는 애플리케이션, 파일 시스템, 보안 메커니즘, 암호 알고리즘 등을 포함하고 있는 일종의 스마트 카드이다.[10] 본 논문에서는 저사양 환경에서 많이 사용되는 UICC-16bit에서의 구현을 목적으로 두었는데 16비트 환경이므로 암호 알고리즘은 저전력, 경량화, 효율성의 특징을 지녀야 함은 당연하며 이러한 환경에 알맞게 설계된 LEA를 UICC-16bit 상에 최적으로 구현하였다. 일반 32비트 LEA는 128비트의 블록을 32비트 4워드로 분할하여 암/복호화를 수행하고 192비트의 라운드 키는 각각 32비트 6워드로 이루어져 있다. 16비트 전용 LEA는 블록을 16비트 8워드로 분할하여 수행하고 16비트 12워드로 이루어져있다. 단, LEA는 덧셈과 뺄셈이 모듈로 2^{32}에서 이루어지기 때문에 16비트 환경에서는 자릿수 올림과 자릿수 내림에 대한 매크로가 필요하며 또한 순환이동도 n자리를 수행한다면 n자리만큼 넘겨줘야 할 매크로가 필요하다. 이러한 과정을 거쳐 코딩 한 후 UICC-

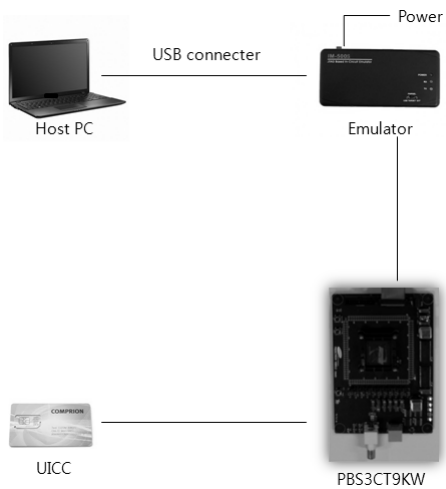


Fig. 5. Development structure of UICC

Table 3. Key-scheduling and Constants

<table>
<tr><th colspan="3">Advanced modified Key-Scheduling algorithm and constants (32bits only)</th></tr>
<tr><td>LEA-128</td><td>$T[4] \leftarrow K$
for i = 0 to 23;
{
$T[0] \leftarrow ROTL_1(T[0] \boxplus CS[i][0])$;
$T[1] \leftarrow ROTL_3(T[1] \boxplus CS[i][1])$;
$T[2] \leftarrow ROTL_6(T[2] \boxplus CS[i][2])$;
$T[3] \leftarrow ROTL_{11}(T[3] \boxplus CS[i][3])$;
$RK_i \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$;
}</td><td>//for LEA-128
static unsigned int CS[24][4] = {
0xc3efe9db, 0x87dfd3b7, 0x0fbfa76f, 0x1f7f4ede,
0x88c4d604, 0x1189ac09, 0x23135812, 0x4626b024,
0xe789f229, 0xcf13e453, 0x9e27c8a7, 0x3c4f914f,
0xc6f98763, 0x8df30ec7, 0x1be61d8f, 0x37cc3b1e,
0x3efe9dbc, 0x7dfd3b78, 0xfbfa76f0, 0xf7f4ede1,
0x8c4d6048, 0x189ac091, 0x31358122, 0x626b0244,
0x789f229e, 0xf13e453c, 0xe27c8a79, 0xc4f914f3,
0x6f98763c, 0xdf30ec78, 0xbe61d8f1, 0x7cc3b1e3,
0xefe9dbc3, 0xdfd3b787, 0xbfa76f0f, 0x7f4ede1f,
0xc4d60488, 0x89ac0911, 0x13581223, 0x26b02446,
0x89f229e7, 0x13e453cf, 0x27c8a79e, 0x4f914f3c,
0xf98763c6, 0xf30ec78d, 0xe61d8f1b, 0xcc3b1e37,
0xfe9dbc3e, 0xfd3b787d, 0xfa76f0fb, 0xf4ede1f7,
0x4d60488c, 0x9ac09118, 0x35812231, 0x6b024462,
0x9f229e78, 0x3e453cf1, 0x7c8a79e2, 0xf914f3c4,
0x98763c6f, 0x30ec78df, 0x61d8f1be, 0xc3b1e37c,
0xe9dbc3ef, 0xd3b787df, 0xa76f0fbf, 0x4ede1f7f,
0xd60488c4, 0xac091189, 0x58122313, 0xb0244626,
0xf229e789, 0xe453cf13, 0xc8a79e27, 0x914f3c4f,
0x8763c6f9, 0x0ec78df3, 0x1d8f1be6, 0x3b1e37cc,
0x9dbc3efe, 0x3b787dfd, 0x76f0fbfa, 0xede1f7f4,
0x60488c4d, 0xc091189a, 0x81223135, 0x0244626b,
0x229e789f, 0x453cf13e, 0x8a79e27c, 0x14f3c4f9,
0x763c6f98, 0xec78df30, 0xd8f1be61, 0xb1e37cc3
};</td></tr>
<tr><td>LEA-192</td><td>$T[6] \leftarrow K$
for i = 0 to 27;
{
$T[0] \leftarrow ROTL_1(T[0] \boxplus ROTL_i(\delta[i \bmod 6]))$;
$T[1] \leftarrow ROTL_3(T[1] \boxplus ROTL_{i+1}(\delta[i \bmod 6]))$;
$T[2] \leftarrow ROTL_6(T[2] \boxplus ROTL_{i+2}(\delta[i \bmod 6]))$;
$T[3] \leftarrow ROTL_{11}(T[3] \boxplus ROTL_{i+3}(\delta[i \bmod 6]))$;
$T[4] \leftarrow ROTL_{13}(T[4] \boxplus ROTL_{i+4}(\delta[i \bmod 6]))$;
$T[5] \leftarrow ROTL_{17}(T[5] \boxplus ROTL_{i+5}(\delta[i \bmod 6]))$;
$RK_i \leftarrow (T[0], T[1], T[2], T[3], T[4], T[5])$;
}</td><td>//for LEA-192
static unsigned int CS[28][6] = {
0xc3efe9db, 0x87dfd3b7, 0x0fbfa76f, 0x1f7f4ede, 0x3efe9dbc, 0x7dfd3b78,
0x88c4d604, 0x1189ac09, 0x23135812, 0x4626b024, 0x8c4d6048, 0x189ac091,
0xe789f229, 0xcf13e453, 0x9e27c8a7, 0x3c4f914f, 0x789f229e, 0xf13e453c,
0xc6f98763, 0x8df30ec7, 0x1be61d8f, 0x37cc3b1e, 0x6f98763c, 0xdf30ec78,
0x15ea49e7, 0x2bd493ce, 0x57a9279c, 0xaf524f38, 0x5ea49e71, 0xbd493ce2,
0xf0bb4158, 0xe17682b1, 0xc2ed0563, 0x85da0ac7, 0x0bb4158f, 0x17682b1e,
0xfbfa76f0, 0xf7f4ede1, 0xefe9dbc3, 0xdfd3b787, 0xbfa76f0f, 0x7f4ede1f,
0x31358122, 0x626b0244, 0xc4d60488, 0x89ac0911, 0x13581223, 0x26b02446,
0xe27c8a79, 0xc4f914f3, 0x89f229e7, 0x13e453cf, 0x27c8a79e, 0x4f914f3c,
0xbe61d8f1, 0x7cc3b1e3, 0xf98763c6, 0xf30ec78d, 0xe61d8f1b, 0xcc3b1e37,
0x7a9279c5, 0xf524f38a, 0xea49e715, 0xd493ce2b, 0xa9279c57, 0x524f38af,
0x2ed0563c, 0x5da0ac78, 0xbb4158f0, 0x7682b1e1, 0xed0563c2, 0xda0ac785,
0xfe9dbc3e, 0xfd3b787d, 0xfa76f0fb, 0xf4ede1f7, 0xe9dbc3ef, 0xd3b787df,
0x4d60488c, 0x9ac09118, 0x35812231, 0x6b024462, 0xd60488c4, 0xac091189,
0x9f229e78, 0x3e453cf1, 0x7c8a79e2, 0xf914f3c4, 0xf229e789, 0xe453cf13,
0x98763c6f, 0x30ec78df, 0x61d8f1be, 0xc3b1e37c, 0x8763c6f9, 0x0ec78df3,
0xa49e715e, 0x493ce2bd, 0x9279c57a, 0x24f38af5, 0x49e715ea, 0x93ce2bd4,
0xb4158f0b, 0x682b1e17, 0xd0563c2e, 0xa0ac785d, 0x4158f0bb, 0x82b1e176,
0xa76f0fbf, 0x4ede1f7f, 0x9dbc3efe, 0x3b787dfd, 0x76f0fbfa, 0xede1f7f4,
0x58122313, 0xb0244626, 0x60488c4d, 0xc091189a, 0x81223135, 0x0244626b,
0xc8a79e27, 0x914f3c4f, 0x229e789f, 0x453cf13e, 0x8a79e27c, 0x14f3c4f9,
0x1d8f1be6, 0x3b1e37cc, 0x763c6f98, 0xec78df30, 0xd8f1be61, 0xb1e37cc3,
0x279c57a9, 0x4f38af52, 0x9e715ea4, 0x3ce2bd49, 0x79c57a92, 0xf38af524,
0x0563c2ed, 0x0ac785da, 0x158f0bb4, 0x2b1e1768, 0x563c2ed0, 0xac785da0,
0xdbc3efe9, 0xb787dfd3, 0x6f0fbfa7, 0xde1f7f4e, 0xbc3efe9d, 0x787dfd3b,
0x0488c4d6, 0x091189ac, 0x12231358, 0x244626b0, 0x488c4d60, 0x91189ac0,
0x29e789f2, 0x53cf13e4, 0xa79e27c8, 0x4f3c4f91, 0x9e789f22, 0x3cf13e45,
0x63c6f987, 0xc78df30e, 0x8f1be61d, 0x1e37cc3b, 0x3c6f9876, 0x78df30ec
};</td></tr>
<tr><td>LEA-256</td><td>$T[6] \leftarrow K$
for i = 0 to 31;
{
$T[6i \bmod 8] \leftarrow ROTL_1(T[6i \bmod 8] \boxplus CS[i][0])$;
$T[6i+1 \bmod 8] \leftarrow ROTL_3(T[6i+1 \bmod 8] \boxplus CS[i][1])$;
$T[6i+2 \bmod 8] \leftarrow ROTL_6(T[6i+2 \bmod 8] \boxplus CS[i][2])$;
$T[6i+3 \bmod 8] \leftarrow ROTL_{11}(T[6i+3 \bmod 8] \boxplus CS[i][3])$;
$T[6i+4 \bmod 8] \leftarrow ROTL_{13}(T[6i+4 \bmod 8] \boxplus CS[i][4])$;
$T[6i+5 \bmod 8] \leftarrow ROTL_{17}(T[6i+5 \bmod 8] \boxplus CS[i][5])$;
$RK_i \leftarrow$
$(T[6i \bmod 8], T[6i+1 \bmod 8], T[6i+2 \bmod 8],$
$T[6i+3 \bmod 8], T[6i+4 \bmod 8], T[6i+5 \bmod 8])$;
}</td><td>//for LEA-256
static unsigned int CS[32][6] = {
0xc3efe9db, 0x87dfd3b7, 0x0fbfa76f, 0x1f7f4ede, 0x3efe9dbc, 0x7dfd3b78,
0x88c4d604, 0x1189ac09, 0x23135812, 0x4626b024, 0x8c4d6048, 0x189ac091,
0xe789f229, 0xcf13e453, 0x9e27c8a7, 0x3c4f914f, 0x789f229e, 0xf13e453c,
0xc6f98763, 0x8df30ec7, 0x1be61d8f, 0x37cc3b1e, 0x6f98763c, 0xdf30ec78,
0x15ea49e7, 0x2bd493ce, 0x57a9279c, 0xaf524f38, 0x5ea49e71, 0xbd493ce2,
0xf0bb4158, 0xe17682b1, 0xc2ed0563, 0x85da0ac7, 0x0bb4158f, 0x17682b1e,
0x13bc8ab8, 0x27791570, 0x4ef22ae0, 0x9de455c0, 0x3bc8ab81, 0x77915702,
0xe204abf2, 0xc40957e5, 0x8812afcb, 0x10255f97, 0x204abf2e, 0x40957e5c,
0xefe9dbc3, 0xdfd3b787, 0xbfa76f0f, 0x7f4ede1f, 0xfe9dbc3e, 0xfd3b787d,
0xc4d60488, 0x89ac0911, 0x13581223, 0x26b02446, 0x4d60488c, 0x9ac09118,
0x89f229e7, 0x13e453cf, 0x27c8a79e, 0x4f914f3c, 0x9f229e78, 0x3e453cf1,
0xf98763c6, 0xf30ec78d, 0xe61d8f1b, 0xcc3b1e37, 0x98763c6f, 0x30ec78df,
0xea49e715, 0xd493ce2b, 0xa9279c57, 0x524f38af, 0xa49e715e, 0x493ce2bd,
0xbb4158f0, 0x7682b1e1, 0xed0563c2, 0xda0ac785, 0xb4158f0b, 0x682b1e17,
0xbc8ab813, 0x79157027, 0xf22ae04e, 0xe455c09d, 0xc8ab813b, 0x91570277,
0x04abf2e2, 0x0957e5c4, 0x12afcb88, 0x255f9710, 0x4abf2e20, 0x957e5c40,
0xe9dbc3ef, 0xd3b787df, 0xa76f0fbf, 0x4ede1f7f, 0x9dbc3efe, 0x3b787dfd,
0xd60488c4, 0xac091189, 0x58122313, 0xb0244626, 0x60488c4d, 0xc091189a,
0xf229e789, 0xe453cf13, 0xc8a79e27, 0x914f3c4f, 0x229e789f, 0x453cf13e,
0x8763c6f9, 0x0ec78df3, 0x1d8f1be6, 0x3b1e37cc, 0x763c6f98, 0xec78df30,
0x49e715ea, 0x93ce2bd4, 0x279c57a9, 0x4f38af52, 0x9e715ea4, 0x3ce2bd49,
0x4158f0bb, 0x82b1e176, 0x0563c2ed, 0x0ac785da, 0x158f0bb4, 0x2b1e1768,
0x8ab813bc, 0x15702779, 0x2ae04ef2, 0x55c09de4, 0xab813bc8, 0x57027791,
0xabf2e204, 0x57e5c409, 0xafcb8812, 0x5f971025, 0xbf2e204a, 0x7e5c4095,
0xdbc3efe9, 0xb787dfd3, 0x6f0fbfa7, 0xde1f7f4e, 0xbc3efe9d, 0x787dfd3b,
0x0488c4d6, 0x091189ac, 0x12231358, 0x244626b0, 0x488c4d60, 0x91189ac0,
0x29e789f2, 0x53cf13e4, 0xa79e27c8, 0x4f3c4f91, 0x9e789f22, 0x3cf13e45,
0x63c6f987, 0xc78df30e, 0x8f1be61d, 0x1e37cc3b, 0x3c6f9876, 0x78df30ec,
0xe715ea49, 0xce2bd493, 0x9c57a927, 0x38af524f, 0x715ea49e, 0xe2bd493c,
0x58f0bb41, 0xb1e17682, 0x63c2ed05, 0xc785da0a, 0x8f0bb415, 0x1e17682b,
0xb813bc8a, 0x70277915, 0xe04ef22a, 0xc09de455, 0x813bc8ab, 0x02779157,
0xf2e204ab, 0xe5c40957, 0xcb8812af, 0x9710255f, 0x2e204abf, 0x5c40957e
};</td></tr>
</table>

16bit에서 구현하였으며 구현에 사용된 호스트 PC는 Intel Core(TM) i5-3230 cpu@2.60GHz 이고 타겟 시스템은 16-bit microcontroller unit인 PBS3CT9KW이며 에뮬레이터는 IM-500S를 사용하였고, Compiler는 CalmSHINE16Plus이다. Fig.6.은 UICC 작업환경을 나타낸다.수행 결과 PBS3CT9KW는 빅엔디안(Big Endian)이므로 구현시 EndianChange 과정을 수행하였고 ARIA와의 효율성 비교는 호스트pc에서 진행하였으며 ECB, CTR 운영모드를 사용하여 비교하였다. Fig.7.은 UICC-16bit 상에서 CTR모드를 사용하는 LEA 블록암호의 구현 결과 화면이다.

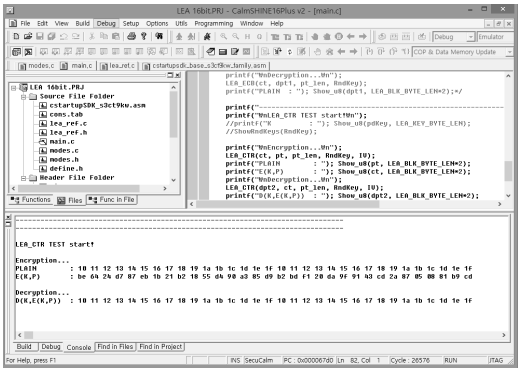


Fig. 6. LEA_CTR mode on UICC-16bit

3.3 ARIA 및 기존 LEA와의 효율성 비교

ARIA 알고리즘은[6] UICC-16bit 상에 알맞게 구현되어있는 코드를, 기존 LEA 알고리즘은 reference 코드를 이용하였으며 효율성 비교는 호스트 PC(Intel(R) Core(TM) i5-3230M cpu@2.60 (GHz)에서 benchmark코드를 이용하여[11] 비교하였다. 기존 LEA는 개선된 키 스케줄링을 사용하지 않는다면 수많은 회전이동연산을 거치게 되는데 LEA 알고리즘은 32비트 단위로 연산을 수행하기 때문에 16비트 환경에 알맞게 구현할 시 순환이동연산을 적용할 때마다 다른 워드로 넘겨주는 매크로가 계속 실행되야 하므로 최적화된 코드에 비해 5배 이상의 매우 비효율적인 속도가 측정되었다. 또한 ARIA와의 비교 결과는 LEA-128이 ARIA-128보다 약 1.57배(키스케줄 1.66배, 암호화 1.49배, 복호화 1.58배), CTR 모드를 기준으로 약 1.52배(키스케줄 1.66배, 암호화 1.45배, 복호화 1.45배) 빠른 것으로 나타났다. (Table 4. 참조)

IV. 결론

임베디드 시스템(Embedded System)은 유비쿼터스(Ubiquitous) 컴퓨팅 시스템의 핵심으로 최근 많이 사용되고 있는 전자기기들은 대부분 임베디드 시스템을 탑재하고 있으며[12] 근래 32비트 혹은 64비트 사양의 임베디드 시스템을 많이 사용하고 있지만 아직도 여전히 저사양 환경에서는 16비트 사양을 많이 사용하고 있다. 본 논문에서는 고속, 경량, 저전력을 만족하는 LEA 블록암호를 UICC-16bit상에 구현하였으며 효율적인 키 스케줄링을 이용하여 저사양 임베디드 시스템이 요구하는 효율적인 측면에 한층 더 가깝게 LEA를 구현하였고 국내 표준 블록암호 ARIA와의 효율성을 UICC-16bit상에서 비교하였다. 그 결과 ARIA-128보다 약 1.5배 이상의 효율을 보였다. 따라서 효율적인 16비트 기반의 LEA 블록암호를 구현하였으며 구현 효율성 비교의 결과로 LEA의 우수성을 입증하였다. 또한 경량암호 LEA의 저전력적인 특성을 확인하기 위해 추후 타 알고리즘과의 전력소모 비교실험을 진행 할 예정이다.

Table 4. Compare code speed (16bits only)

	Mode of operation	Key round (cycles/byte)	Encryption (cycles/byte)	Decryption (cycles/byte)
LEA-128 (32bits only reference code) [11]	ECB	145.88	80.03	79.16
	CTR	145.88	90.38	90.31
ARIA-128	ECB	165.06	171.69	172.06
	CTR	165.06	186.00	186.13
LEA-128 (Advanced code)	ECB	99.25	110.81	105.00
	CTR	99.25	119.50	119.69

References

[1] TTAK.KO-12.0223, "128-Bit Block Cipher LEA," Telecommunications Technology Association, pp.1-24, December 18, 2013.

[2] Christof Paar and Jan Pelzl, "Understanding Cryptography, A text-book for students and practitioners," Springer, pp.3-5, 2010.

[3] W.C Barker and Elaine Barker, SP 800-67, Revision 1, "Recommendation for the Triple Data Encryption Algorithm(TDEA) Block Cipher," NIST, pp.1-12, January, 2012.

[4] National Institute of Standards and Technology, "Advanced Encryption Standards," NIST FIPS PUB 197,pp.1-26, November 26, 2001.

[5] TTAS.KO-12.0004/R1, "128-bit Block Cipher SEED," Telecommunications Technology Association, pp.4-9, December 21, 2005.

[6] KS X 1213:2004, "128 bit Block Encryption Algorithm ARIA," Korean Agency for Technology and Standards (KATS), December 30, 2004,

[7] Andrey Bogdanov, Nicky Mouha, Elmar Tischhauser, Deniz Toz, Kerem Varici, Vesselin Velichkov, Meiqin Wang, Qingju Wang, and Vincent Rijmen, "Security Evaluation of the Block Cipher LEA Final Report," COSIC, Belgium, pp.3-30, July 7, 2011.

[8] Hwa-jeong Seo, Howon Kim, "Recent Trends in Implementing Cryptography with Embedded Microprocessors," JKIISC, pp.815-824, October, 2013.

[9] Hokyoon Lee, S.W Kim, and Y.S. Han, "Pair Register Allocation Algorithm for 16-bit Instruction Set Architecture (ISA) Processor," Korea Institute Processing Society, pp.265-270, December, 2011.

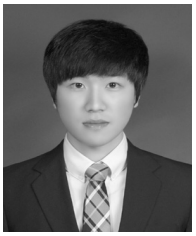
[10] Hakdoo Kim and Sungik Jeon, "UICC File System Design and Implementation for java Card," CICS"03 , pp.584-587, November 21, 2003.

[11] http://www.kcryptoforum.or.kr/, Block cipher LEA reference code in notice board, Korea Cryptography Forum, May 14, 2014.

[12] Chi-Seong Park, "A Study on the Implementation of Cryptographic Modules and its Applications to Security Protocols on Embedded Systems," Master's thesis, Kookmin Univ, Korea, December, 2012.

〈저자소개〉

김 현 일 (Hyunil Kim) 학생회원
2014년 2월: 공주대학교 응용수학과 학사 졸업
2014년 3월~현재: 공주대학교 융합과학과 석사 재학
〈관심분야〉 암호모듈 구현, 데이터 보호 기술



박 철 희 (Cheolhee Park) 정회원
2014년 2월: 공주대학교 응용수학과 학사 졸업
2014년 9월~ : 공주대학교 수학과 석사 재학
〈관심분야〉 암호모듈 구현, 데이터 보호 기술

홍 도 원 (Dowon Hong) 종신회원
1994년 2월: 고려대학교 수학과 학사
2000년 2월: 고려대학교 수학과 박사
2000년 4월~2012년 2월: 한국전자통신연구원 팀장, 책임연구원
2012년 3월~현재: 공주대학교 응용수학과 부교수
〈관심분야〉 암호기술, 프라이버시 보호기술

서 창 호 (Changho Seo) 종신회원
1990년: 고려대학교 수학과 학사
1992년: 고려대학교 수학과 이학석사
1996년: 고려대학교 수학과 이학박사
1996년~1996년: 국방과학연구소 선임연구원
1996년~2000년: 한국전자통신연구원 선임연구원, 팀장
2000년~현재: 공주대학교 응용수학과 교수
〈관심분야〉 암호알고리즘, PKI, 무선인터넷 보안 등