ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 19-03-33

j.inst.Korean.electr.electron.eng.Vol.23,No.3,971~977,September 2019
http://dx.doi.org/10.7471/ikeee.2019.23.3.971
221

아두이노를 이용한 AES와 LEA의 암복호화 속도 측정

Measurements of Encryption and Decryption Times of AES and LEA Algorithms on an Arduino MCU

권 영 준*, 신 형 식*★

Yeongjun Kwon*, Hyungsik Shin*★

Abstract

This paper presents an experimental result showing the encryption and decryption times of the AES and LEA algorithms. AES and LEA algorithms are international and Korean standards for block ciphers, respectively. Through experiments, this paper investigates the applicability of the LEA algorithm for light weight IoT devices. In order to measure the encryption and decryption times, 256-bit and 128-bit secret keys were randomly generated for AES and LEA, respectively. Under our test environment using an Arduino microcontroller, the AES algorithm takes about 45㎳ for encryption and decryption processes, whereas the LEA algorithm takes about 4㎳. Even though processing times of each algorithm may vary much under different implementation and test environments, this experimental result shows that the LEA algorithm can be applied to many light weight IoT devices for security goals.

요 약

본 논문에서는 국제 표준 블록 암호 AES와 국산 표준 경량 블록 암호인 LEA의 암복호화 속도를 비교 실험한 결과를 소개하고, LEA의 사물인터넷 기기 통신에의 활용 가능성을 확인한다. 두 암호 알고리즘의 속도 측정을 위하여, AES의 경우는 256비트의 무작위 생성 비밀키를, LEA의 경우는 128비트의 무작위 생성 비밀키를 이용하여 암복호화를 수행하였다. 아두이노를 이용한 실험을 진행한 결과, 256비트 비밀키 AES 알고리즘의 경우 암복호화에 약 45㎳가 소모되었고, LEA의 경우 128비트 비밀키에 대하여 암복호화에 약 4㎳가 소모되었다. 알고리즘 구현 방식과 실험 환경에 따라 속도 차이는 매우 다양할 수 있으나, 본 실험 결과를 통하여 LEA 알고리즘은 경량 에너지 환경을 요구하는 사물인터넷 기기의 보안 알고리즘으로서 충분히 고려해볼 만하다는 것을 확인하였다.

Key words : block cipher, encryption, decryption, AES, LEA

Ⅰ. 서론

정보통신 기술이 발전하면서 사물인터넷(Internet of Things, IoT) 기술이 4차 산업혁명의 중요한 분야로 주목받고 있다. 사물인터넷은 다양한 사물 기기들을 인터넷에 연결하여 사물 기기들에서 수집

* School of Electronic and Electrical Engineering, Hongik University
★ Corresponding author
E-mail : hyungsik.shin@hongik.ac.kr, Tel : +82-2-320-1661
※ Acknowledgment
This work was supported by Korea Electric Power Corporation (Grant Number : R18XA02).
This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science, ICT & Future Planning (MSIP)) (no. 2017R1C1B5015901).
Manuscript received Sep. 5, 2019; revised Sep. 24, 2019; accepted Sep. 26, 2019.

되는 많은 정보를 공유함으로써, 인류 생활에 여러 가지 공익을 창출하고자 하는 총체적 기술을 의미한다.

사물인터넷 기술은 그 본질적 특성상 다양한 정보를 통신망을 통하여 송수신하는데, 만약 악의적 사용자가 이러한 정보를 송수신 과정에서 탈취하거나 변조한다면, 커다란 금전적 또는 물리적 피해를 초래할 수 있다. 이에 대한 대비책으로 많은 종류의 보안 기술들이 끊임없이 개발되고 적용됐으며, 그 중심에는 암호 알고리즘 기술이 있다.

다양한 블록 암호 알고리즘 기술은 여러 가지 보안 기술의 핵심 요소 기술 중 하나이다. 블록 암호 알고리즘 기술로서는, 미국 NIST(National Institute of Standards and Technology)가 표준으로 지정한 블록 암호기술인 AES(Advanced Encryption Standard) 알고리즘이 많이 사용되고 있다[1]. AES는 강력한 보안성능과 운영 모드의 다양성과 같은 장점 덕분에 무선랜, 블루투스 등의 여러 가지 유무선 통신 시스템의 보안 알고리즘으로 사용되어왔다.

AES 알고리즘은 많은 장점에도 불구하고, 데이터를 고속으로 암·복호화하고 그 과정에서의 전력 소모를 최소화해야 하는 사물인터넷 기술의 특성상, 저전력 소모를 요구하는 사물인터넷 기기에서 사용하기에는 적합하지 않을 수 있다. 이에 따라, 경량 블록 암호 알고리즘 기술에 대한 수요가 증가하고 있다. 따라서 이 논문에서는 국내에서 개발한 경량 블록 암호 알고리즘 LEA[2],[3]와 앞서 언급한 AES 알고리즘의 암·복호화 속도를 아두이노 하드웨어를 이용하여 비교한다. 암·복호화에 사용하는 LEA 알고리즘은 한국인터넷진흥원(KISA)에서 제공하는 아두이노용 소스 코드[4]를 사용했으며, AES는 소스 코드 공유 플랫폼인 Github에 공개된 AES 알고리즘 소스 코드[5]를 사용하였다.

II. 본론

1. 암호화 알고리즘

AES는 128비트(bit)의 입력을 128/192/256비트 비밀키를 이용하여 암·복호화를 진행하고, LEA는 128비트의 입력을 128/192/256비트 비밀키를 이용하여 암·복호화를 진행한다. 본 논문에서는 AES의 경우 256비트 비밀키 알고리즘(AES256)을, LEA의 경우 128비트 비밀키 알고리즘(LEA128)을 사용하였다. 아래에서 AES256과 LEA128의 암·복호화 과정을 순서대로 간략하게 소개한다.

가. AES 알고리즘

AES256의 경우 14라운드로 구성되며 그림 1은 암·복호화 과정을 나타낸 개략도이다. 암호화 과정을 살펴보면, 우선 128비트 입력을 256비트 비밀키의 첫 128비트와 XOR 연산을 하고, 연산이 끝난 128비트의 결과물을 8비트씩 4×4 행렬로 정렬하여 SubBytes, ShiftRows, MixColumns, AddRoundKey의 네 가지 과정을 14번 반복하여 암호화를 수행한다. 단 마지막 라운드에서는 MixColumns 과정을 생략한다.

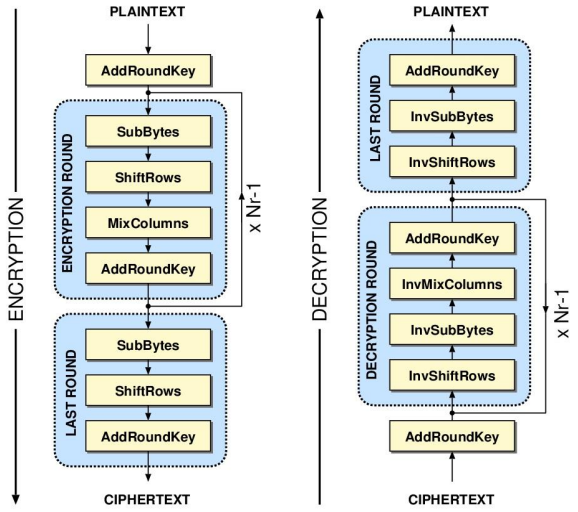

Fig. 1. Overview of AES algorithm[6].
그림 1. AES 알고리즘의 동작 개략도[6]

(1) SubBytes(SB)

SubBytes 과정은 표 1과 같이 8비트씩 16진수 4×4 행렬로 정렬된 128비트의 입력을 그림 2의 S-box를 이용해 변환하는 과정이다. 표 1에서 행렬 S의 각 원소를 행(i)과 열(j)값에 따라 $S_{i,j}$로 표기하였다.

Table 1. 4×4 matrix formed by an input block.
표 1. 입력 블록으로부터 생성된 4×4 행렬

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Fig. 2. AES S-box[1].
그림 2. AES 알고리즘에서 사용되는 S-box[1]

(2) ShiftRows(SR)

ShiftRows 과정은 S-box를 통해 얻은 새로운 4×4 행렬을 표 2에 설명한 방법처럼 순환 이동시켜 새로운 행렬을 얻는 과정이다.

Table 2. ShiftRows process.
표 2. ShiftRows 과정

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	→	$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$		$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$		$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$		$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

(3) MixColumns(MC)

MixColumns 과정은 입력으로 주어진 4×4 행렬을 아래의 수식 1과 같이 미리 정해진 특정한 하나의 4×4 행렬과 곱 연산을 수행하는 과정이다. 단 마지막 라운드(14라운드)의 경우는 이 과정을 생략한다.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02\,03\,01\,01 \\ 01\,02\,03\,01 \\ 01\,01\,02\,03 \\ 03\,01\,01\,02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad (1)$$

위의 수식 1을 좀 더 자세하게 기술하면 아래 네 개의 식으로 표현할 수 있으며, ⊕는 XOR 연산을 의미한다.

$$S'_{0,c} = (02 \bullet S_{0,c}) \oplus (03 \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$$

$$S'_{1,c} = S_{0,c} \oplus (02 \bullet S_{1,c}) \oplus (03 \bullet S_{2,c}) \oplus S_{3,c}$$

$$S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (02 \bullet S_{2,c}) \oplus (03 \bullet S_{3,c})$$

$$S'_{3,c} = (03 \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (02 \bullet S_{3,c})$$

(4) AddRoundKey(ARK)

AddRoundKey 과정은 위의 과정까지 수행한 결과인 4×4 행렬과 라운드 키를 XOR 연산 수행하는 과정이다. 라운드 키는 다음에 설명하는 키 확장 과정을 통해서 얻는다.

(5) AES 키 확장(Key Schedule)

비밀키를 확장하여 각 라운드에 사용하는 라운드 키를 얻는 방법은 다음과 같다. 먼저, 256비트 비밀키를 표 3에 나타낸 것처럼 4×8 크기의 행렬로 배열한다. 이 행렬의 i번째 행, j번째 열을 $W_{i,j}$라고 표기한다. 첫 여덟 개의 열($W_{i,0} \sim W_{i,7}$)중 처음 네 개는 0번째($W_{i,0} \sim W_{i,3}$), 나머지 네 개는 1번째($W_{i,4} \sim W_{i,7}$) 라운드 키가 된다.

Table 3. 4×8 matrix from secret key
표 3. 비밀키로부터 얻어진 4×8 행렬

$W_{0,0}$	$W_{0,1}$	$W_{0,2}$	$W_{0,3}$	$W_{0,4}$	$W_{0,5}$	$W_{0,6}$	$W_{0,7}$
$W_{1,0}$	$W_{1,1}$	$W_{1,2}$	$W_{1,3}$	$W_{1,4}$	$W_{1,5}$	$W_{1,6}$	$W_{1,7}$
$W_{2,0}$	$W_{2,1}$	$W_{2,2}$	$W_{2,3}$	$W_{2,4}$	$W_{2,5}$	$W_{2,6}$	$W_{2,7}$
$W_{3,0}$	$W_{3,1}$	$W_{3,2}$	$W_{3,3}$	$W_{3,4}$	$W_{3,5}$	$W_{3,6}$	$W_{3,7}$

이후의 라운드 키를 구하는 과정은 다음과 같다. 우선 2번째 라운드 키의 첫 번째 열인 $W_{i,8}$은 표 4와 같이 $W_{i,7}$을 순환 이동시킨 다음 그림 1의 S-box를 이용해 변환하고, 표 5에 주어진 Rcon 행렬의 $(j/8)$열과 XOR 연산을 취하여 얻는다. 이후의 $W_{i,j}$값은 $W_{i,j-8}$과 XOR 연산을 수행한 값이 된다. 단, j가 4의 배수였던 경우 W_{j-8}과의 XOR 연산 전에 $W_{i,j}$를 S-box를 이용해 변환한다. 이렇게 얻은 $W_{i,j}$를 4개씩 묶어 라운드키로 사용한다.

Table 4. Key Expansion example of $W_{i,8}$
표 4. $W_{i,8}$ 키 확장 예시

$W_{0,7}$	Rotation	$W_{1,7}$	SB	XOR with Rcon	01	XOR with $W_{i,0}$	$W_{0,8}$
$W_{1,7}$		$W_{2,7}$			00		$W_{1,8}$
$W_{2,7}$		$W_{3,7}$			00		$W_{2,8}$
$W_{3,7}$		$W_{0,7}$			00		$W_{3,8}$

Table 5. Rcon Table.
표 5. Rcon 표

j/8	1	2	3	4	5	6	7	8	9	10
RCON	01	02	04	08	10	20	40	80	1B	36
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00

(6) AES 복호화

복호화의 경우 암호화 방법과 같은 방식으로 진행되지만, 바이트 치환 변환(InvSubBytes), InvShiftRows, InvMixColumns에서 사용하는 연산이 암호화 연산과 약간 다르다[1].

나. LEA 알고리즘

LEA는 ARX(Addition, Rotation, XOR)의 세 가지 연산을 주로 사용하는 알고리즘으로, LEA128 알고리즘은 128비트 입력을 받고 128비트 비밀키를 사용한다. 또한, 암·복호화를 진행할 때 총 24라운드를 반복적으로 수행한다. 각 라운드에는 192비트의 라운드 키를 사용하며 그 과정은 그림 3과 같다.

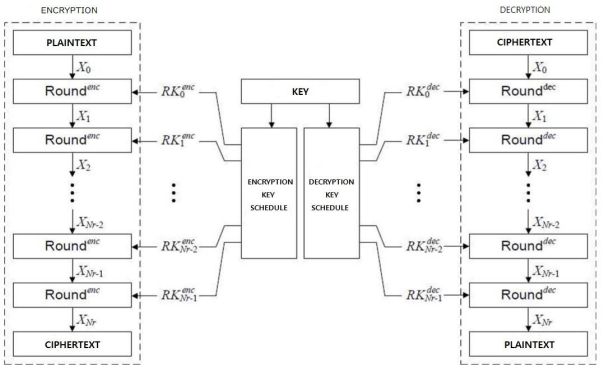

Fig. 3. LEA algorithm process schematic[3].
그림 3. LEA 알고리즘 동작 개략도[3]

(1) LEA 암호화 알고리즘

LEA 알고리즘의 암호화 방식을 살펴보면, 라운드마다 128비트의 입력을 32비트씩 4개의 블록(X_i[0], X_i[1], X_i[2], X_i[3])으로 분할한 후, i라운드에서 i번째 라운드키 RKi^{enc}와 i번째 입력 비트 X_i를 입력으로 하여 Round^{enc} 연산을 수행하여 X_{i+1}을 얻음을 알 수 있다.

i번째 라운드의 Round^{enc} 함수는 다음의 과정을 통해서 수행된다.

1. $X_{i+1}[0] = \mathrm{ROL}_9((X_i[0] \oplus RKi^{enc}[0]) \boxplus (X_i[1] \oplus RKi^{enc}[1]))$
2. $X_{i+1}[1] = \mathrm{ROR}_5((X_i[1] \oplus RKi^{enc}[2]) \boxplus (X_i[2] \oplus RKi^{enc}[3]))$
3. $X_{i+1}[2] = \mathrm{ROR}_3((X_i[2] \oplus RKi^{enc}[4]) \boxplus (X_i[3] \oplus RKi^{enc}[5]))$
4. $X_{i+1}[3] = X_i[0]$

ROL과 ROR은 각각 좌, 우측으로 순환 이동, $\boxplus$는 32비트 덧셈을 의미한다.

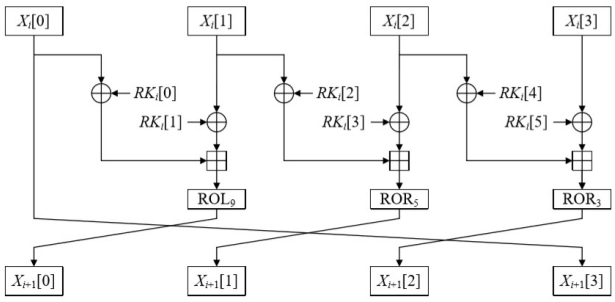

Fig. 4. LEA encryption process[2].
그림 4. LEA 암호화 과정[2]

(2) LEA 키 확장

128비트 비밀키를 32비트씩 4개 블록 K[0]~K[3]으로 나눈 뒤 C0[0]~C0[3]와 각각 mod 2^{32} 덧셈 연산(32비트 덧셈)을 수행한 다음 C0[0]부터 C0[3]까지 순서대로 1, 3, 6, 11 좌측 순환한 것을 T[0]~T[3]에 저장한다. 그 후 T함수를 T[0], T[1], T[2], T[1], T[3], T[1] 순서로 $RKi^{enc}[i]$, $(i = 0 \sim 5)$에 저장하면 1라운드 키가 된다. 2라운드는 위와 같은 방법으로 수행하되 128비트 비밀키 대신 전 라운드의 T값을 C1[0]~C1[3]와 덧셈 연산을 한다. 같은

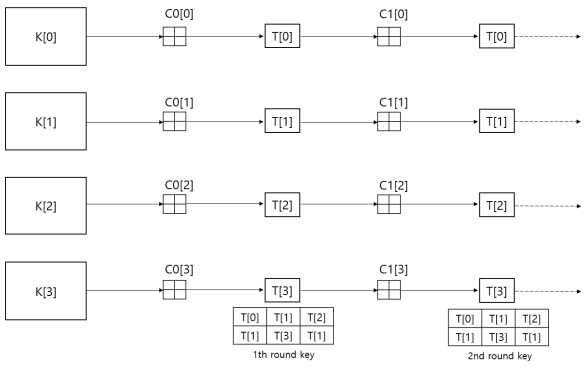

Fig. 5. LEA key schedule.
그림 5. LEA 키 확장 과정

방식을 반복해서 C23[0]～C23[3]를 사용해 24라운드 키까지 구하면 키 확장 과정이 완료된다.

여기서 각 라운드에서 사용되는 Ci[0]～Ci[3]를 구하는 알고리즘은 다음과 같다.

```
for i = 0 to 23 do
    for j = 0 to 3 do
        Ci[j] ← ROL(i+j)(δ[i mod 4])
    end for
end for
```

여기서 사용되는 32비트 상수들 δ[i] ($0 \le i \le 7$)는 δ[0] = c3EFE9DB, δ[1] = 44626B02, δ[2] = 79E27C8A, δ[3] = 78DF30EC, δ[4] = 715EA49E, δ[5] = C785dA0A, δ[6] = E04EF22A, δ[7] = E5C40957로 정의되며 이 값들은 $\sqrt{766965}$의 HEX값으로 76, 69, 65는 L, E, A의 아스키코드 값에서 가져왔다. 따라서 128비트 키 확장 과정을 정리하면 다음과 같다.

```
T ← K
for i = 0 to 23 do
    T[0] ← ROL1(T[0]⊞Ci[0])
    T[1] ← ROL3(T[1]⊞Ci[1])
    T[2] ← ROL6(T[2]⊞Ci[2])
    T[3] ← ROL11(T[3]⊞Ci[3])
    RKi^enc ← (T[0], T[1], T[2], T[1], T[3], T[1])
end for
```

(3) LEA 복호화

복호화도 각 라운드마다 128비트의 입력 비트를 32비트씩 4개의 블록(X_i[0], X_i[1], X_i[2], X_i[3])으로 분할한 후, i라운드에서 i번째 라운드키 RKi^{dec}와 i번째 입력 비트 X_i를 입력으로 하는 Round^{dec}연산을 수행한다.

```
for i = 0 to 23 do
    Xi+1[0] ← Xi[3]
    Xi+1[1] ← (ROR9(Xi[0])⊟(Xi+1[0]⊕RKi^dec[0]))
              ⊕RKi^dec[1]
    Xi+1[2] ← (ROL5(Xi[1])⊟(Xi+1[1]⊕RKi^dec[2]))
              ⊕RKi^dec[3]
    Xi+1[3] ← (ROL3(Xi[2])⊟(Xi+1[2]⊕RKi^dec[4]))
              ⊕RKi^dec[5]
end for
```

ROL은 좌측으로 순환 이동, ⊟는 mod 2^{32} 뺄셈 연산을 의미한다.

복호화키 RKi^{dec}의 경우 RKi^{enc}를 구하는 과정과 동일하되, RKi^{enc}의 역순으로 저장된 키다.

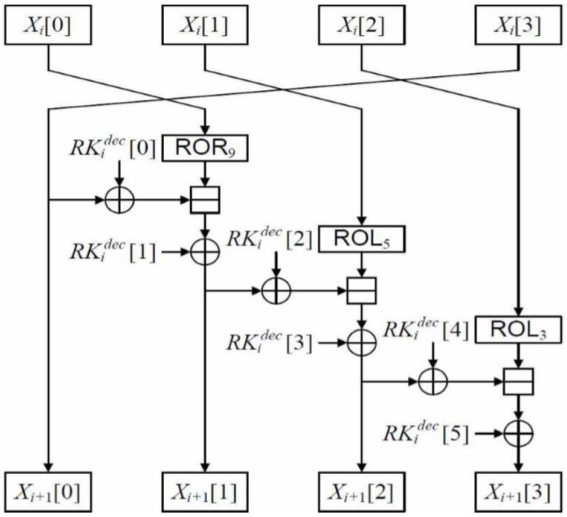

Fig. 6. LEA decryption process[3].
그림 6. LEA 복호화 과정[3]

2. 아두이노

아두이노(Arduino)는 공개 소스 기반의 단일 보드 마이크로콘트롤러(microcontroller)로서, 처음에는 학생들이 쉽게 작품을 설계할 수 있게 하려고 제작되었다. 현재에는 아두이노를 비즈니스에 활용하는 기업들도 늘어나는 추세에 있으며, 특히 사물인터넷과 연계하여 많이 사용하는 추세이다. 따라서, 본 논문에서는 블록 암호 알고리즘의 암・복호화 속도 비교를 위한 하드웨어로 아두이노를 채택하여 실험을 진행하였다. 아두이노는 다양한 제품군을 출시하고 있는데, 본 실험에서 사용한 제품은 아두이노 우노(Uno)로서, 그림 7과 같은 구조로 이루어져 있으며 상세한 제원은 표 6과 같다.

Fig. 7. Arduino Uno R3 Front[7].
그림 7. 아두이노 우노 R3 전면[7]

Table 6. ADUINO UNO SPEC[7].
표 6. 아두이노 UNO 제원[7]

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7 - 12V
Input Voltage (limit)	6 - 20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Clock Speed	16 MHz

3. LEA와 AES의 암・복호화 속도 비교

두 알고리즘의 암・복호화 속도를 비교하기 위하여, 아두이노에서 “asdfasdfasdfasdf”의 평문을 128비트 크기의 아스키(ASCII) 코드로 변환하여 입력한 뒤, 무작위로 비밀키를 생성해 라운드키로 확장하여 암・복호화 과정을 수행하였다. 이러한 수행을 50번 반복하여 라운드 키 생성, 암호화 시간, 복호화 시간의 평균과 최빈값을 각각 측정하였다.

수행 시간 측정을 위하여 아두이노 시간 측정 함수인 micro()를 사용하였는데, micro() 함수 특성상 4㎲ 단위로 측정되기 때문에 측정된 시간에는 약간의 오차가 존재할 수 있다. 측정된 시간은 그림 8과 그림 9와 같이 비밀키 생성, 키 확장 시간, 평문, 암호문, 암・복호화 시간, 복호화된 평문을 시리얼 통신으로 전달받아 확인하였다.

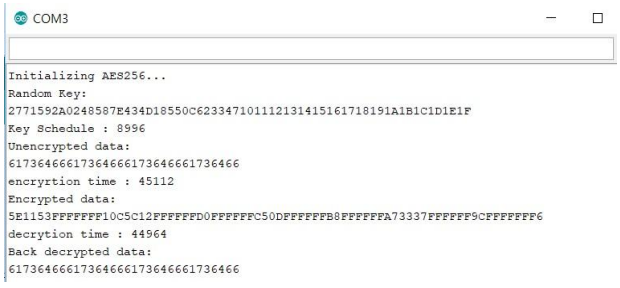

Fig. 8. AES256 process example in Arduino.
그림 8. 아두이노에서의 AES256 실행 예시

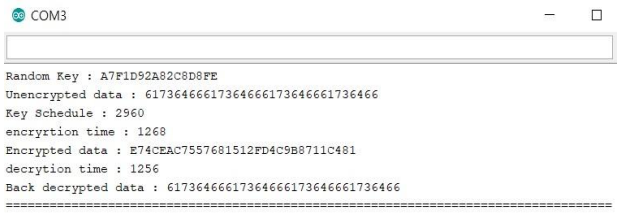

Fig. 9. LEA128 process example in Arduino.
그림 9. 아두이노에서의 LEA128 실행 예시

그림 10은 측정한 시간의 평균값을 나타내는 그래프이며 Key Sch는 키 확장 시간, Enc는 암호화 시간, Dec는 복호화 시간을 의미한다. 또한, 그래프의 왼쪽은 AES256, 오른쪽은 LEA128의 측정 결과를 나타낸다.

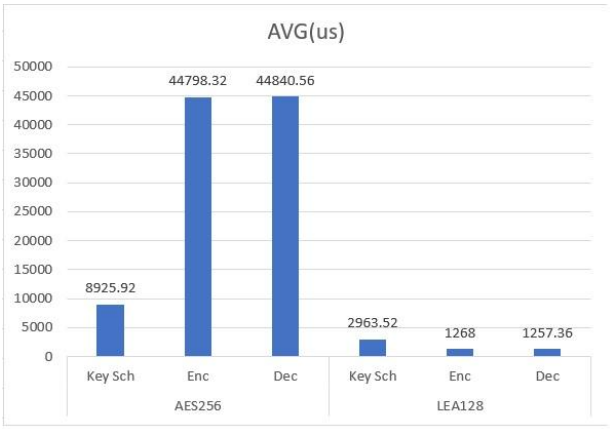

Fig. 10. Average times for key schedule, encryption and decryption(㎲)
그림 10. 암・복호화 및 키 확장 시간 평균값(㎲)

그림 11은 측정된 시간 값의 최빈값을 나타내는 그래프이며 표기 방식은 그림 10과 같다.

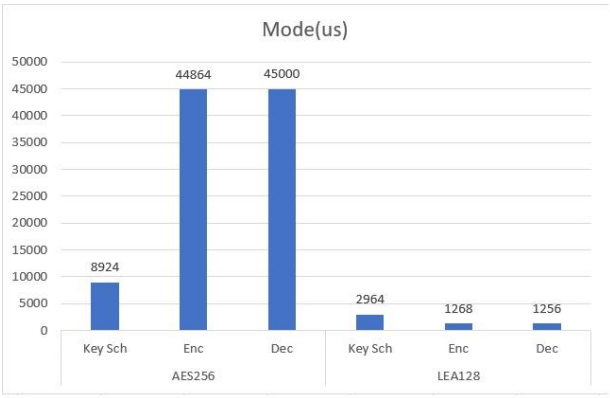

Fig. 11. Mode times for key schedule, encryption and decryption(㎲)
그림 11. 암・복호화 및 키 확장 시간 최빈값(㎲)

측정 결과 AES256이 LEA128에 비해서 키 생성은 약 3배, 암・복호화의 경우 약 35배의 시간을 더 소모하는 것을 확인하였다.

Ⅲ. 결론

본 실험을 통해 아두이노 동작 시 LEA128이 AES256에 비해서 약 30배 이상 빠르게 암·복호화 과정을 수행할 수 있음을 확인하였다. 물론, 본 실험은 아두이노와 공개 소스 소프트웨어를 사용하였기 때문에, 다른 방식으로 구현된 환경에서는 속도 결과에 차이가 있을 수 있다. 그러나, 대체로 LEA 알고리즘이 AES 알고리즘보다는 수행 속도가 빠를 것으로 예상할 수 있다. 따라서, 에너지 소모량에 많은 제한이 따르는 사물인터넷 환경에서는 AES 알고리즘보다 LEA 알고리즘이 좀 더 효율적이라고 생각할 수 있다.

국제 표준으로 인정되어 많이 활용되고 있는 알고리즘인 AES와는 달리, LEA 알고리즘의 보안성은 아직 엄밀하게 검증된 것은 아니므로, LEA 알고리즘을 적극적으로 활용하기 위해서는 좀 더 많은 연구와 실험 결과가 필요하다. 앞으로 LEA 알고리즘이 국제 표준 경량 블록 암호 알고리즘으로 인정받기 위해 계속 노력한다면, 국내 개발 알고리즘인 LEA가 더욱 많이 활용될 것으로 기대한다.

References

[1] Morris J. Dworkin, Elaine B. Barker, James R. Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, James F. Dray Jr, “Advanced Encryption Standard (AES),” *Federal Information Processing Standards (FIPS)-197*, 2001.
DOI: 10.6028/NIST.FIPS.197

[2] Hong, D., Lee, J.K., Kim, D. C., Kwon, D., Ryu, K. H. and Lee, D. G., “LEA: A 128-bit block cipher for fast encryption on common processors,” *International Workshop on Information Security Applications*, pp. 3-27, 2013.
DOI: 10.1007/978-3-319-05149-9_1

[3] NSR, “Block chiper LEA Specification,” https://seed.kisa.or.kr/kisa/skill/EgovLeaInfo.do

[4] KISA(KOREA INTERNET & SECURITY AGENCY) and Ministry of Science, ICT and Future Planning, “A Guide to the Use of Cryptographic Technology in the Internet of Things(IoT),” https://www.kisa.or.kr/public/laws/laws3_View.jsp?mode=view&p_No=259&b_No=259&d_No=84

[5] qistoph, “ArduinoAES256,” https://github.com/qistoph/ArduinoAES256

[6] Frank Kagan Gürkaynak, “GALS System Design: Side Channel Attack Secure Cryptographic Accelerators” Doctoral Dissertation, ETH Zurich, 2006. https://iis-people.ee.ethz.ch/~kgf/acacia/c3.html

[7] ARDUINO, “ARDUINO UNO REV3”, https://media.digikey.com/pdf/Data%20Sheets/Arduino%20PDFs/A000066_Web.pdF

BIOGRAPHY

Yeongjun Kwon (Member)

2020 : BS degree in Electronics and Electrical Engineering, Hongik University. (Expected)
2019 : Intern, Samsung Electronics Visual Display Business

Hyungsik Shin (Member)

2003 : BS degree in Electrical Engineering, Seoul National University.
2005 : MS degree in Electrical Engineering, Stanford University.
2011 : PhD degree in Electrical Engineering, Stanford University.
2011~2016 : Research Engineer, Docomo Innovations, Inc.
2016~ : Assistant Professor in School of Electronic and Electrical Engineering, Hongik University.