블록암호 LEA RUST 구현

김상원

https://youtu.be/Szr94hEHTmE

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

LEA란

키 스케줄 함수

암·복호화 함수

Q & A

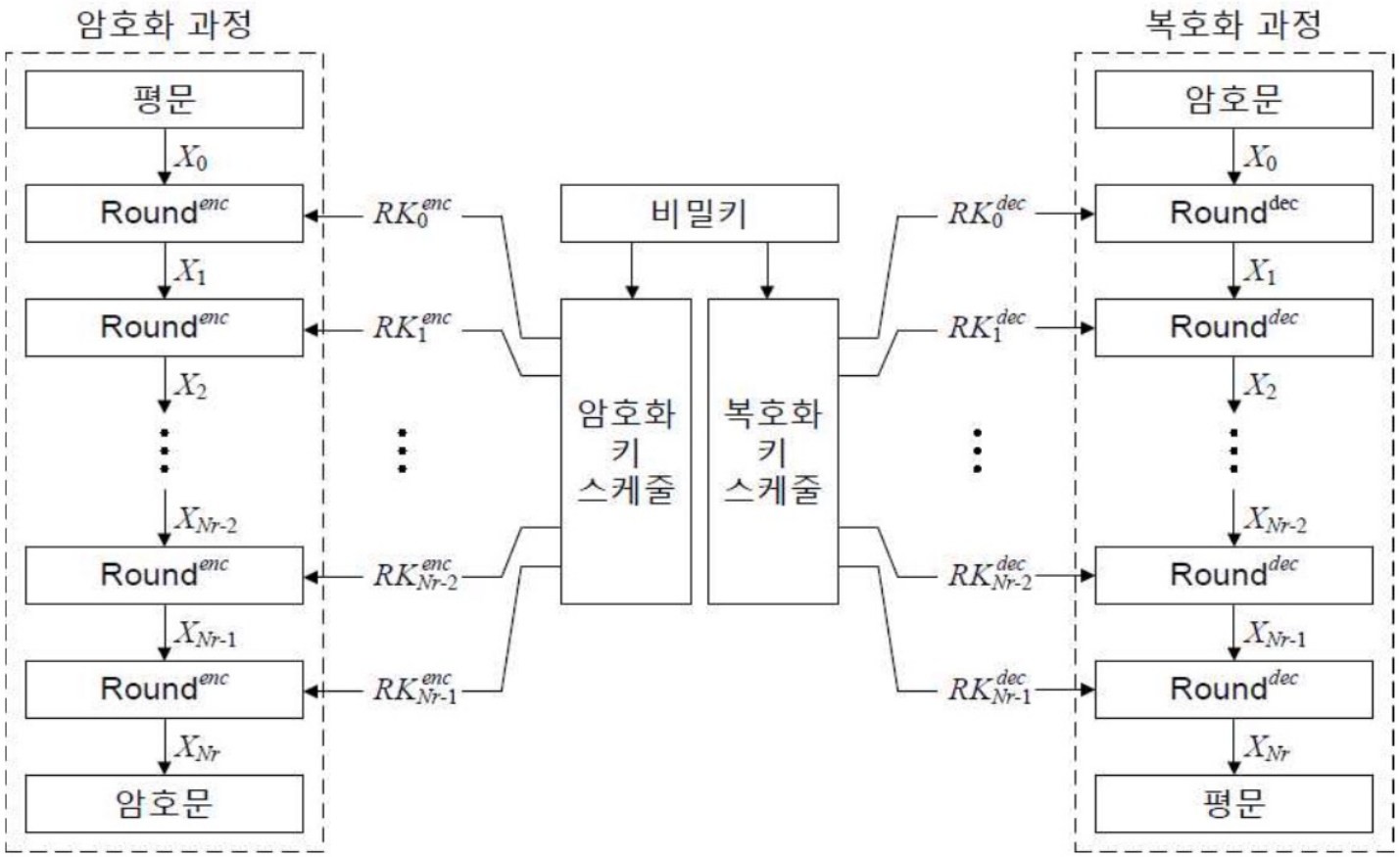
LEA란

- LEA (Lightweight Encryption Algorithm)는 경량 블록 암호 알고리즘
- 주로 자원이 제한된 환경에서의 사용을 목적으로 설계됨
- 효율적이고 빠른 암호화를 제공하기 위해 고안됨
- 주로 소프트웨어 구현에 최적화되어 있음
- LEA의 설계 목적은 AES (Advanced Encryption Standard)와 같은 기존의 암호화 알고리즘과 비교했을 때, 더 낮은 자원 사용과 빠른 속도를 제공하는 것

LEA란

- 블록 크기: 128 bit
- 키 크기: 128, 192, 256 bit
- 라운드 수: 24, 28, 32 Round
- 알고리즘 구조: ADD, SUB, XOR, ROR, ROL
- 응용 분야: IoT 장치, 스마트 카드, 무선 네트워크 등

LEA란



(그림 5-1) 암호화 및 복호화 과정

<표 5-1> LEA 규격

구분	Nb	Nk	Nr
LEA-128	16	16	24
LEA-192	16	24	28
LEA-256	16	32	32

Nb 평문 또는 암호문을 구성하는 바이트의 개수. LEA에 대하여 Nb = 16으로 고정됨

Nk 비밀키를 구성하는 바이트의 개수. LEA에 대하여 Nk = 16, 24, 또는 32로 사용됨

Nr 라운드 수. Nk에 따라 결정됨. LEA에 대하여 Nr = 24, 28, 또는 32로 사용됨

암호화 키 스케줄 함수

알고리즘 3 LEA-128 암호화 키 스케줄 함수: $(RK_0^{enc}, \cdots, RK_{23}^{enc}) \leftarrow KeySchedule_{128}^{enc}(K)$

입력: 128비트 비밀키 K

출력: 24개의 192비트 암호화 라운드키 RK_i^{enc} $(0 \le i \le 23)$

1: $T \leftarrow K$

2: for i = 0 to 23 do

3: $T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$

4: $T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$

5: $T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$

6: $T[3] \leftarrow ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$

7: $RK_i^{enc} \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$

8: end for

키스케줄에 사용되는 32비트 상수

$\delta[0] = c3efe9db$, $\delta[1] = 44626b02$,

$\delta[2] = 79e27c8a$, $\delta[3] = 78df30ec$,

$\delta[4] = 715ea49e$, $\delta[5] = c785da0a$,

$\delta[6] = e04ef22a$, $\delta[7] = e5c40957$.

암호화 키 스케줄 함수

```rust
fn enc_round_key_gen(mk: &[u32; 4], rk: &mut [u32; 192]) {
    let mut temp: [u32; 4] = [0; 4];

    for i in 0..4 {
        temp[i] = mk[i];

    }
    for i in 0..24 {
        temp[0] = rol(temp[0].wrapping_add(rol( x: KEY_CONST[i % 4], i as u8)), n: 1);
        temp[1] = rol(temp[1].wrapping_add(rol( x: KEY_CONST[i % 4], n: (i + 1) as u8)), n: 3);
        temp[2] = rol(temp[2].wrapping_add(rol( x: KEY_CONST[i % 4], n: (i + 2) as u8)), n: 6);
        temp[3] = rol(temp[3].wrapping_add(rol( x: KEY_CONST[i % 4], n: (i + 3) as u8)), n: 11);
        rk[i * 6] = temp[0];
        rk[i * 6 + 1] = temp[1];
        rk[i * 6 + 2] = temp[2];
        rk[i * 6 + 3] = temp[1];
        rk[i * 6 + 4] = temp[3];
        rk[i * 6 + 5] = temp[1];
    }
    println!();
}
```

1
2
3
4
5
6
7

알고리즘 3 LEA-128 암호화 키 스케줄 함수: $(RK_0^{enc}, \cdots, RK_{23}^{enc}) \leftarrow KeySchedule_{128}^{enc}(K)$

입력: 128비트 비밀키 K

출력: 24개의 192비트 암호화 라운드키 RK_i^{enc} $(0 \le i \le 23)$

1: $T \leftarrow K$
2: for i = 0 to 23 do
3: $T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$
4: $T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$
5: $T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$
6: $T[3] \leftarrow ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$
7: $RK_i^{enc} \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$
8: end for

복호화 키 스케줄 함수

알고리즘 8 LEA-128 복호화 키 스케줄 함수: $(RK_0^{dec}, \cdots, RK_{23}^{dec}) \leftarrow KeySchedule_{128}^{dec}(K)$

입력: 128비트 비밀키 K

출력: 24개의 192비트 복호화 라운드키 RK_i^{dec} $(0 \le i \le 23)$

1: $T \leftarrow K$
2: for i = 0 to 23 do
3: $\quad T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$
4: $\quad T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$
5: $\quad T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$
6: $\quad T[3] \leftarrow ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$
7: $\quad RK_{23-i}^{dec} \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$
8: end for

키스케줄에 사용되는 32비트 상수

$\delta[0] = c3efe9db$, $\delta[1] = 44626b02$,
$\delta[2] = 79e27c8a$, $\delta[3] = 78df30ec$,
$\delta[4] = 715ea49e$, $\delta[5] = c785da0a$,
$\delta[6] = e04ef22a$, $\delta[7] = e5c40957$.

복호화 키 스케줄 함수

```rust
fn dec_round_key_gen(mk: &[u32; 4], rk: &mut [u32; 192]) {
    let mut temp: [u32; 4] = [0; 4];

    [redacted]

1   for i in 0..4 {
        temp[i] = mk[i];

        [redacted]
    }
2   for i in 0..24 {
3       temp[0] = rol(temp[0].wrapping_add(rol( x: KEY_CONST[i % 4], i as u8)), n: 1);
4       temp[1] = rol(temp[1].wrapping_add(rol( x: KEY_CONST[i % 4], n: (i + 1) as u8)), n: 3);
5       temp[2] = rol(temp[2].wrapping_add(rol( x: KEY_CONST[i % 4], n: (i + 2) as u8)), n: 6);
6       temp[3] = rol(temp[3].wrapping_add(rol( x: KEY_CONST[i % 4], n: (i + 3) as u8)), n: 11);
        rk[138 -i * 6] = temp[0];
        rk[139 -i * 6] = temp[1];
        rk[140 -i * 6] = temp[2];
7       rk[141 -i * 6] = temp[1];
        rk[142 -i * 6] = temp[3];
        rk[143 -i * 6] = temp[1];
    }
```

알고리즘 8 LEA-128 복호화 키 스케줄 함수: $(RK_0^{dec}, \cdots, RK_{23}^{dec}) \leftarrow KeySchedule_{128}^{dec}(K)$

입력: 128비트 비밀키 K

출력: 24개의 192비트 복호화 라운드키 RK_i^{dec} $(0 \le i \le 23)$

1: $T \leftarrow K$
2: for i = 0 to 23 do
3: $T[0] \leftarrow ROL_1(T[0] \boxplus ROL_i(\delta[i \bmod 4]))$
4: $T[1] \leftarrow ROL_3(T[1] \boxplus ROL_{i+1}(\delta[i \bmod 4]))$
5: $T[2] \leftarrow ROL_6(T[2] \boxplus ROL_{i+2}(\delta[i \bmod 4]))$
6: $T[3] \leftarrow ROL_{11}(T[3] \boxplus ROL_{i+3}(\delta[i \bmod 4]))$
7: $RK_{23-i}^{dec} \leftarrow (T[0], T[1], T[2], T[1], T[3], T[1])$
8: end for

암호화 함수

알고리즘 1 암호화 함수: $C \leftarrow Encrypt(P, RK_0^{enc}, RK_1^{enc}, \cdots, RK_{Nr-1}^{enc})$

입력: 128비트 평문 P, Nr개의 192비트 라운드키 $RK_0^{enc}, RK_1^{enc}, \cdots, RK_{Nr-1}^{enc}$

출력: 128비트 암호문 C

1: $X_0 \leftarrow P$
2: for i = 0 to (Nr - 1) do
3: $X_{i+1} \leftarrow Round^{enc}(X_i, RK_i^{enc})$
4: end for
5: $C \leftarrow X_{Nr}$

알고리즘 2 암호화 과정의 i번째 라운드 함수: $X_{i+1} \leftarrow Round^{enc}(X_i, RK_i^{enc})$

입력: 128비트 내부상태 변수 X_i, 192비트 라운드키 RK_i^{enc}

출력: 128비트 내부상태 변수 X_{i+1}

$x \boxplus y$ 두 32비트열 x와 y에 대해 IntToBit(BitToInt(x) + BitToInt(y) mod 2^{32})로 정의되는 연산

1: $X_{i+1}[0] \leftarrow ROL_9((X_i[0] \oplus RK_i^{enc}[0]) \boxplus (X_i[1] \oplus RK_i^{enc}[1]))$
2: $X_{i+1}[1] \leftarrow ROR_5((X_i[1] \oplus RK_i^{enc}[2]) \boxplus (X_i[2] \oplus RK_i^{enc}[3]))$
3: $X_{i+1}[2] \leftarrow ROR_3((X_i[2] \oplus RK_i^{enc}[4]) \boxplus (X_i[3] \oplus RK_i^{enc}[5]))$
4: $X_{i+1}[3] \leftarrow X_i[0]$

암호화 함수

```rust
fn enc(x: &mut [u32; 4], rk: &[u32; 192]) ->[u32;4] {
    println!("Enc start : ");
    println!("_____________________________ ");
    let mut temp;


    for i in 0..24 {

        let temp1 = x[0] ^ rk[i * 6];
        let temp2 = x[1] ^ rk[i * 6 + 2];
        let temp3 = x[2] ^ rk[i * 6 + 4];
        temp = x[0];
      1 x[0] = rol(temp1.wrapping_add(x[1] ^ rk[(i * 6) + 1]), n: 9);
      2 x[1] = ror(temp2.wrapping_add(x[2] ^ rk[(i * 6) + 3]), n: 5);
      3 x[2] = ror(temp3.wrapping_add(x[3] ^ rk[(i * 6) + 5]), n: 3);
      4 x[3] = temp;
    }
        *x
}
```

```rust
let mut encrypted_text :[u32; 4] = enc(&mut plain_text, &mut enc_round_key);
```

$$
\begin{aligned}
1:&\quad X_{i+1}[0] \leftarrow ROL_9((X_i[0] \oplus RK_i^{enc}[0]) \boxplus (X_i[1] \oplus RK_i^{enc}[1])) \\
2:&\quad X_{i+1}[1] \leftarrow ROR_5((X_i[1] \oplus RK_i^{enc}[2]) \boxplus (X_i[2] \oplus RK_i^{enc}[3])) \\
3:&\quad X_{i+1}[2] \leftarrow ROR_3((X_i[2] \oplus RK_i^{enc}[4]) \boxplus (X_i[3] \oplus RK_i^{enc}[5])) \\
4:&\quad X_{i+1}[3] \leftarrow X_i[0]
\end{aligned}
$$

복호화 함수

알고리즘 6 복호화 함수: $P \leftarrow Decrypt(C, RK_0^{dec}, RK_1^{dec}, \cdots, RK_{Nr-1}^{dec})$

입력: 128비트 암호문 C, Nr개의 192비트 라운드키 $RK_0^{dec}, RK_1^{dec}, \cdots, RK_{Nr-1}^{dec}$

출력: 128비트 평문 P

1: $X_0 \leftarrow C$
2: for i = 0 to (Nr - 1) do
3: $X_{i+1} \leftarrow Round^{dec}(X_i, RK_i^{dec})$
4: end for
5: $P \leftarrow X_{Nr}$

알고리즘 7 복호화 과정의 i번째 라운드 함수: $X_{i+1} \leftarrow Round^{dec}(X_i, RK_i^{dec})$

입력: 128비트 내부상태 변수 X_i, 192비트 라운드키 RK_i^{dec}

출력: 128비트 내부상태 변수 X_{i+1}

x 田 y

두 32비트열 x와 y에 대해 IntToBit(BitToInt(x) − BitToInt(y) mod 2^{32})로 정의되는 연산

1: $X_{i+1}[0] \leftarrow X_i[3]$
2: $X_{i+1}[1] \leftarrow (ROR_9(X_i[0]) \boxplus (X_{i+1}[0] \oplus RK_i^{dec}[0])) \oplus RK_i^{dec}[1]$
3: $X_{i+1}[2] \leftarrow (ROL_5(X_i[1]) \boxplus (X_{i+1}[1] \oplus RK_i^{dec}[2])) \oplus RK_i^{dec}[3]$
4: $X_{i+1}[3] \leftarrow (ROL_3(X_i[2]) \boxplus (X_{i+1}[2] \oplus RK_i^{dec}[4])) \oplus RK_i^{dec}[5]$

복호화 함수

```
fn dec(x: &mut [u32; 4], rk: &[u32; 192])->[u32;4]  {
```

```
let mut decrypted_text :[u32; 4] = dec(&mut encrypted_text, &mut dec_round_key);
```

```
    for i in 0..24 {
        let temp0 = x[0];
        let temp1 = x[1];
        let temp2 = x[2];
        let temp3 = x[3];
1       x[0] = temp3;
2       x[1] = (ror(temp0, n: 9).wrapping_sub(x[0]^rk[i*6])) ^rk[i*6 + 1];
3       x[2] = (rol(temp1, n: 5).wrapping_sub(x[1]^rk[i*6 + 2 ])) ^rk[i*6 + 3];
4       x[3] = (rol(temp2, n: 3).wrapping_sub(x[2]^rk[i*6 + 4 ])) ^rk[i*6 + 5];

    }
    *x
}
```

알고리즘 7 복호화 과정의 i번째 라운드 함수: $X_{i+1} \leftarrow Round^{dec}(X_i, RK_i^{dec})$
입력: 128비트 내부상태 변수 X_i, 192비트 라운드키 RK_i^{dec}
출력: 128비트 내부상태 변수 X_{i+1}
1: $X_{i+1}[0] \leftarrow X_i[3]$
2: $X_{i+1}[1] \leftarrow (ROR_9(X_i[0]) \boxminus (X_{i+1}[0] \oplus RK_i^{dec}[0])) \oplus RK_i^{dec}[1]$
3: $X_{i+1}[2] \leftarrow (ROL_5(X_i[1]) \boxminus (X_{i+1}[1] \oplus RK_i^{dec}[2])) \oplus RK_i^{dec}[3]$
4: $X_{i+1}[3] \leftarrow (ROL_3(X_i[2]) \boxminus (X_{i+1}[2] \oplus RK_i^{dec}[4])) \oplus RK_i^{dec}[5]$

Q & A