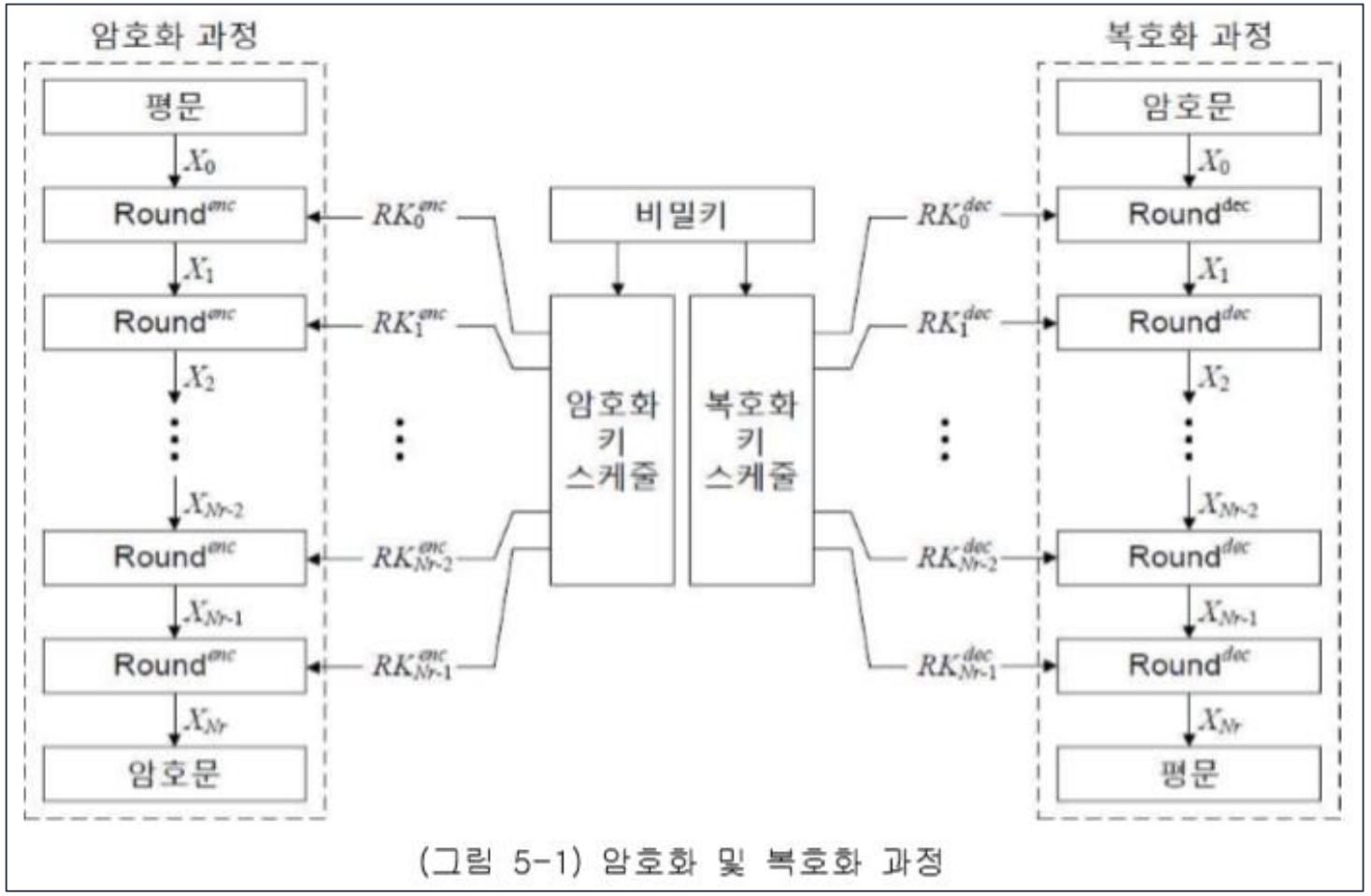
Chapter1
LEA 암호 알고리즘

정보보안암호수학과 20182209 김수빈

LEA

<표 5-1> LEA 규격

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



(그림 5-1) 암호화 및 복호화 과정

LEA.h

```
#ifndef _LEA_H_
#define _LEA_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef unsigned char BYTE; //1byte
typedef unsigned long int WORD; //4byte

int ROL(int i, WORD value);
int ROR(int i, WORD value);
void KeySchedule_128(BYTE *K, WORD *RK);
void KeySchedule_192(BYTE *K, WORD *RK);
void KeySchedule_256(BYTE *K, WORD *RK);
void Encrypt(int Nr, WORD *RK, BYTE *P, BYTE *C);
void Decrypt(int Nr, WORD *RK, BYTE *D, BYTE *C);

#else
#endif
```

LEA.c

LEA 키 스케줄링

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

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} $(0 \le i \le 23)$

```
T ← K
for i = 0 to 23 do
    T[0] ← ROL_1(T[0] ⊞ ROL_i(δ[i mod 4]))
    T[1] ← ROL_3(T[1] ⊞ ROL_{i+1}(δ[i mod 4]))
    T[2] ← ROL_6(T[2] ⊞ ROL_{i+2}(δ[i mod 4]))
    T[3] ← ROL_11(T[3] ⊞ ROL_{i+3}(δ[i mod 4]))
    RK_i^enc ← (T[0], T[1], T[2], T[1], T[3], T[1])
end for
```

알고리즘 4 LEA-192 암호화 키 스케줄 함수: $(RK_0^{enc}, \cdots, RK_{27}^{enc}) \leftarrow KeySchedule_{192}^{enc}(K)$

입력: 192비트 비밀키 K

출력: 28개의 192비트 암호화 라운드키 RK_i^{enc} $(0 \le i \le 27)$

```
T ← K
for i = 0 to 27 do
    T[0] ← ROL_1(T[0] ⊞ ROL_i(δ[i mod 6]))
    T[1] ← ROL_3(T[1] ⊞ ROL_{i+1}(δ[i mod 6]))
    T[2] ← ROL_6(T[2] ⊞ ROL_{i+2}(δ[i mod 6]))
    T[3] ← ROL_11(T[3] ⊞ ROL_{i+3}(δ[i mod 6]))
    T[4] ← ROL_13(T[4] ⊞ ROL_{i+4}(δ[i mod 6]))
    T[5] ← ROL_17(T[5] ⊞ ROL_{i+5}(δ[i mod 6]))
    RK_i^enc ← (T[0], T[1], T[2], T[3], T[4], T[5])
end for
```

알고리즘 5 LEA-256 암호화 키 스케줄 함수: $(RK_0^{enc}, \cdots, RK_{31}^{enc}) \leftarrow KeySchedule_{256}^{enc}(K)$

입력: 256비트 비밀키 K

출력: 32개의 192비트 암호화 라운드키 RK_i^{enc} $(0 \le i \le 31)$

```
T ← K
for i = 0 to 31 do
    T[6i mod 8] ← ROL_1(T[6i mod 8] ⊞ ROL_i(δ[i mod 8]))
    T[(6i+1) mod 8] ← ROL_3(T[(6i+1) mod 8] ⊞ ROL_{i+1}(δ[i mod 8]))
    T[(6i+2) mod 8] ← ROL_6(T[(6i+2) mod 8] ⊞ ROL_{i+2}(δ[i mod 8]))
    T[(6i+3) mod 8] ← ROL_11(T[(6i+3) mod 8] ⊞ ROL_{i+3}(δ[i mod 8]))
    T[(6i+4) mod 8] ← ROL_13(T[(6i+4) mod 8] ⊞ ROL_{i+4}(δ[i mod 8]))
    T[(6i+5) mod 8] ← ROL_17(T[(6i+5) mod 8] ⊞ ROL_{i+5}(δ[i mod 8]))
    RK_i^enc ← (T[6i mod 8], T[(6i+1) mod 8], T[(6i+2) mod 8], T[(6i+3) mod 8],
                T[(6i+4) mod 8], T[(6i+5) mod 8])
end for
```

LEA.c

LEA 키 스케줄링

```c
#include "lea.h"

WORD delta[8] = {
    0xc3efe9db,
    0x44626b02,
    0x79e27c8a,
    0x78df30ec,
    0x715ea49e,
    0xc785da0a,
    0xe04ef22a,
    0xe5c40957
};

//32비트 비트열 x의 i비트 좌측 순환이동
int ROL(int i, WORD value)
{
    return (value << i) | (value >> (32 - i));
}

//32비트 비트열 x의 i비트 우측 순환이동
int ROR(int i, WORD value)
{
    return (value >> i) | (value << (32 - i));
}
```

```c
void KeySchedule_128(BYTE *K, WORD *RK)
{
    WORD T[4];

    memcpy(T, K, 16); //8*16 = 128

    int i;
    for (i = 0; i < 24; i++)
    {
        T[0] = ROL(1, T[0] + ROL(i, delta[i % 4]));
        T[1] = ROL(3, T[1] + ROL(i + 1, delta[i % 4]));
        T[2] = ROL(6, T[2] + ROL(i + 2, delta[i % 4]));
        T[3] = ROL(11, T[3] + ROL(i + 3, delta[i % 4]));
        RK[i * 6 + 0] = T[0];
        RK[i * 6 + 1] = T[1];
        RK[i * 6 + 2] = T[2];
        RK[i * 6 + 3] = T[1];
        RK[i * 6 + 4] = T[3];
        RK[i * 6 + 5] = T[1];
    }
}
```

LEA.c

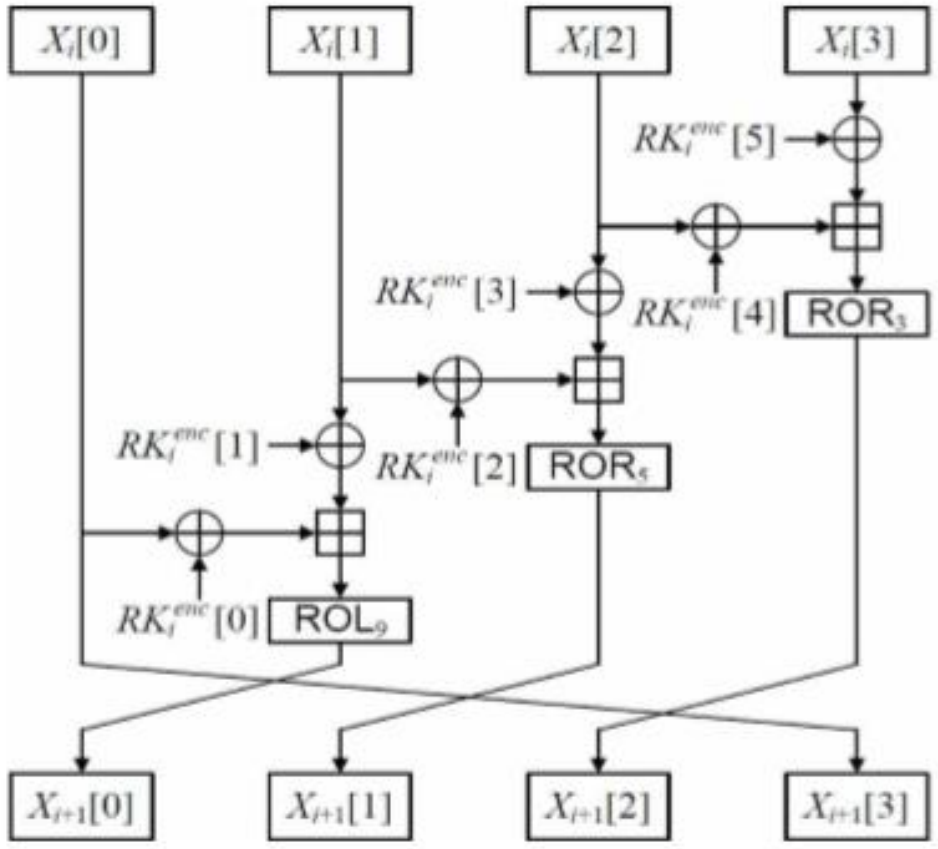
LEA 키 스케줄링

```
void KeySchedule_192(BYTE *K, WORD *RK)
{
    WORD T[6];

    memcpy(T, K, 24); //8*24 = 192

    int i;
    for (i = 0; i < 28; i++)
    {
        T[0] = ROL(1, T[0] + ROL(i, delta[i % 6]));
        T[1] = ROL(3, T[1] + ROL(i + 1, delta[i % 6]));
        T[2] = ROL(6, T[2] + ROL(i + 2, delta[i % 6]));
        T[3] = ROL(11, T[3] + ROL(i + 3, delta[i % 6]));
        T[4] = ROL(13, T[4] + ROL(i + 4, delta[i % 6]));
        T[5] = ROL(17, T[5] + ROL(i + 5, delta[i % 6]));
        RK[i * 6 + 0] = T[0];
        RK[i * 6 + 1] = T[1];
        RK[i * 6 + 2] = T[2];
        RK[i * 6 + 3] = T[3];
        RK[i * 6 + 4] = T[4];
        RK[i * 6 + 5] = T[5];
    }
}
```

```
void KeySchedule_256(BYTE *K, WORD *RK)
{
    WORD T[8];

    memcpy(T, K, 32); //8*32 = 256

    int i;
    for (i = 0; i < 32; i++)
    {
        T[(6 * i + 0) % 8] = ROL(1, T[(6 * i + 0) % 8] + ROL(i, delta[i % 8]));
        T[(6 * i + 1) % 8] = ROL(3, T[(6 * i + 1) % 8] + ROL(i + 1, delta[i % 8]));
        T[(6 * i + 2) % 8] = ROL(6, T[(6 * i + 2) % 8] + ROL(i + 2, delta[i % 8]));
        T[(6 * i + 3) % 8] = ROL(11, T[(6 * i + 3) % 8] + ROL(i + 3, delta[i % 8]));
        T[(6 * i + 4) % 8] = ROL(13, T[(6 * i + 4) % 8] + ROL(i + 4, delta[i % 8]));
        T[(6 * i + 5) % 8] = ROL(17, T[(6 * i + 5) % 8] + ROL(i + 5, delta[i % 8]));
        RK[i * 6 + 0] = T[(i * 6 + 0) % 8];
        RK[i * 6 + 1] = T[(i * 6 + 1) % 8];
        RK[i * 6 + 2] = T[(i * 6 + 2) % 8];
        RK[i * 6 + 3] = T[(i * 6 + 3) % 8];
        RK[i * 6 + 4] = T[(i * 6 + 4) % 8];
        RK[i * 6 + 5] = T[(i * 6 + 5) % 8];
    }
}
```

LEA.c

LEA 암호화



(그림 5-2) 암호화 과정의 i번째 라운드 함수 ($0 \le i \le (Nr-1)$)

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: X0 ← P
2: for i = 0 to (Nr - 1) do
3:     Xi+1 ← Round^enc(Xi, RKi^enc)
4: end for
5: C ← XNr
```

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}

```
1: Xi+1[0] ← ROL9((Xi[0] ⊕ RKi^enc[0]) ⊞ (Xi[1] ⊕ RKi^enc[1]))
2: Xi+1[1] ← ROR6((Xi[1] ⊕ RKi^enc[2]) ⊞ (Xi[2] ⊕ RKi^enc[3]))
3: Xi+1[2] ← ROR3((Xi[2] ⊕ RKi^enc[4]) ⊞ (Xi[3] ⊕ RKi^enc[5]))
4: Xi+1[3] ← Xi[0]
```

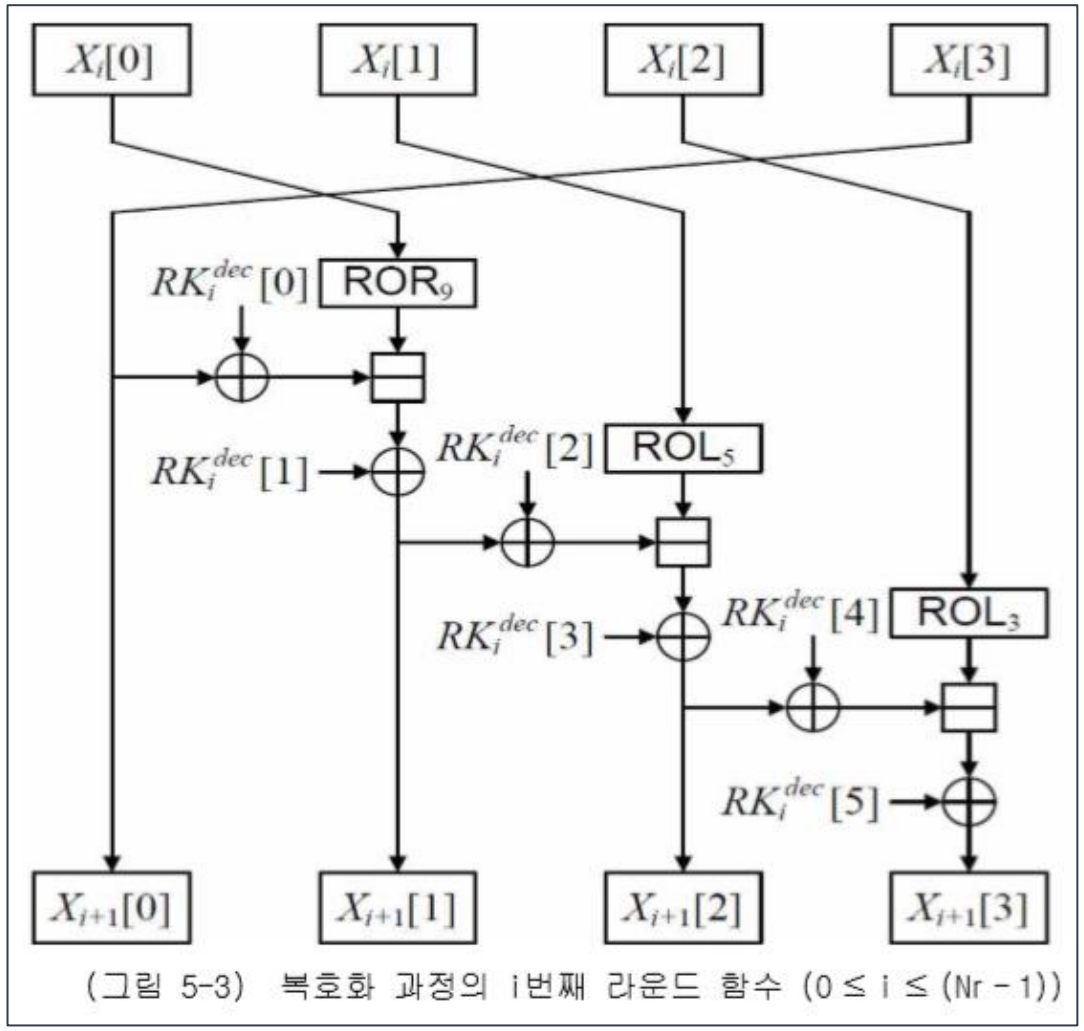
LEA.c

LEA 암호화

```
void Encrypt(int Nr, WORD *RK, BYTE *P, BYTE *C)
{
    WORD X_Round[4];
    WORD X_NextRound[4];

    memcpy(X_Round, P, 16);

    int i;
    for (i = 0; i < Nr; i++)
    {
        X_NextRound[0] = ROL(9, (X_Round[0] ^ RK[i * 6 + 0]) + (X_Round[1] ^ RK[i * 6 + 1]));
        X_NextRound[1] = ROR(5, (X_Round[1] ^ RK[i * 6 + 2]) + (X_Round[2] ^ RK[i * 6 + 3]));
        X_NextRound[2] = ROR(3, (X_Round[2] ^ RK[i * 6 + 4]) + (X_Round[3] ^ RK[i * 6 + 5]));
        X_NextRound[3] = X_Round[0];

        memcpy(X_Round, X_NextRound, 16);
    }

    memcpy(C, X_NextRound, 16);
}
```

LEA.c

LEA 복호화



(그림 5-3) 복호화 과정의 i번째 라운드 함수 ($0 \le i \le (Nr-1)$)

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: X0 ← C
2: for i = 0 to (Nr - 1) do
3:     Xi+1 ← Round^dec(Xi, RKi^dec)
4: end for
5: P ← XNr
```

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

$$RK_i^{dec} = RK_{Nr-i-1}^{enc} \quad (0 \le i \le (Nr-1))$$

LEA.c

LEA 복호화

```
void Decrypt(int Nr, WORD *RK, BYTE *D, BYTE *C)
{
    WORD X_Round[4];
    WORD X_NextRound[4];

    memcpy(X_Round, C, 16);

    int i;
    for (i = 0; i < Nr; i++)
    {
        X_NextRound[0] = X_Round[3];
        X_NextRound[1] = (ROR(9, X_Round[0]) - (X_NextRound[0] ^ RK[((Nr - i - 1) * 6) + 0])) ^ RK[((Nr - i - 1) * 6) + 1];
        X_NextRound[2] = (ROL(5, X_Round[1]) - (X_NextRound[1] ^ RK[((Nr - i - 1) * 6) + 2])) ^ RK[((Nr - i - 1) * 6) + 3];
        X_NextRound[3] = (ROL(3, X_Round[2]) - (X_NextRound[2] ^ RK[((Nr - i - 1) * 6) + 4])) ^ RK[((Nr - i - 1) * 6) + 5];

        memcpy(X_Round, X_NextRound, 16);
    }

    memcpy(D, X_Round, 16);
}
```

main.c

```c
#include "lea.h"

int main()
{
    int i;
    int Nk;
    int Nr;

    //K = 16, 24, 32 byte
    printf("LEA 키의 바이트 길이를 입력하세요.(16, 24, 32)₩n");
    scanf("%d", &Nk);

    BYTE K[16] =
    {
        0x0f, 0x1e, 0x2d, 0x3c, 0x4b, 0x5a, 0x69, 0x78, 0x87, 0x96, 0xa5, 0xb4, 0xc3, 0xd2, 0xe1, 0xf0
    };

    //RoundKey = 144, 168, 192 bit
    WORD RoundKey[144] = { 0, };
    BYTE P[16] = { 0 };
    BYTE C[16] = { 0 };
    BYTE D[16] = { 0 };
```

main.c

```
    if (Nk == 16)
    {
        Nr = 24;
        KeySchedule_128(K, RoundKey);
    }
    if (Nk == 24)
    {
        Nr = 28;
        KeySchedule_192(K, RoundKey);
    }
    if (Nk == 32)
    {
        Nr = 32;
        KeySchedule_256(K, RoundKey);
    }

    printf("Nk = %d₩n", Nk);
    printf("Nr = %d₩n₩n", Nr);
    printf("Key : ");
    for (i = 0; i < Nk; i++)
    {
        printf("0x%02x ", K[i]);
    }
    printf("₩n₩n");
```

```
    WORD tmp = 0;
    printf("Write plaintext : ");
    for (i = 0; i < 16; i++)
    {
        scanf("%x", &tmp);
        P[i] = tmp & 0xff;
    }

    printf("Plaintext : ");
    for (i = 0; i < 16; i++)
    {
        printf("0x%02x ", P[i]);
    }
    printf("₩n₩n");

    Encrypt(Nr, RoundKey, P, C);
    printf("₩n");
```

```
    printf("Ciphertext :");
    for (i = 0; i < 16; i++)
    {
        printf("0x%02x  ", C[i]);
    }
    printf("₩n₩n");

    Decrypt(Nr, RoundKey, D, C);
    printf("₩n");

    printf("Plaintext : ");
    for (i = 0; i < 16; i++)
    {
        printf("0x%02x  ", D[i]);
    }

    return 0;
}
```