2024 암호분석경진대회

6번 문제

빅데이터에 대한 암호화 및 인증코드를 생성하기 위해서는 블록암호 운영모드를 활용하는 것이 일반적이다. 하지만 빅데이터에 대한 암호화에는 많은 연산량이 요구되기 때문에 빅데이터를 활용한 서비스 가용성이 떨어질 수 있다. 이러한 문제점을 해결하기 위해 블록암호 운영모드에 대한 고속 구현이 필요하다. 아래에는 고속 암호화 및 인증코드 생성을 지원하는 하나의 블록암호 운영모드에 대한 C언어 코드를 나타내고 있다. 해당 C 코드를 기반으로 64-비트 Intel 프로세서 상에서 고속으로 블록암호 운영모드를 수행하는 최적화 코드를 제안하시오.

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>

#include <stdlib.h>
#include <time.h>

int64_t cpucycles(void)
{
    unsigned int hi, lo;

        __asm__ __volatile__ ("rdtsc\n\t" : "=a" (lo), "=d"(hi));

        return ((int64_t)lo) | (((int64_t)hi) << 32);
}

//BENCH ROUND
#define BENCH_ROUND 1000

// round of block cipher
#define NUM_ROUND 80

// basic operation
#define ROR(x,r) ((x>>r) | (x<<(8-r)))
#define ROL(x,r) ((x<<r) | (x>>(8-r)))

// constant :: cryptogr in ASCII
#define CONSTANT0 0x63
#define CONSTANT1 0x72
#define CONSTANT2 0x79
#define CONSTANT3 0x70
#define CONSTANT4 0x74
#define CONSTANT5 0x6F
#define CONSTANT6 0x67
#define CONSTANT7 0x72

// constant :: shift offset
#define OFFSET1 1
#define OFFSET3 3
```

```
#define OFFSET5 5
#define OFFSET7 7

// constant :: nonce value
#define NONCE1 0x12
#define NONCE2 0x34
#define NONCE3 0x56
#define NONCE4 0x78
#define NONCE5 0x9A
#define NONCE6 0xBC
#define NONCE7 0xDE

//
void key_scheduling(uint8_t* MK, uint8_t* RK){
    uint32_t i=0;

    //initialization
    for(i=0;i<8;i++){
        RK[i] = MK[i];
    }

    for(i=1;i<NUM_ROUND;i++){
        RK[i*8 + 0]= ROL( RK[(i-1)*8 + 0], (i+OFFSET1)%8) + ROL (CONSTANT0, (i+OFFSET3)%8);
        RK[i*8 + 1]= ROL( RK[(i-1)*8 + 1], (i+OFFSET5)%8) + ROL (CONSTANT1, (i+OFFSET7)%8);
        RK[i*8 + 2]= ROL( RK[(i-1)*8 + 2], (i+OFFSET1)%8) + ROL (CONSTANT2, (i+OFFSET3)%8);
        RK[i*8 + 3]= ROL( RK[(i-1)*8 + 3], (i+OFFSET5)%8) + ROL (CONSTANT3, (i+OFFSET7)%8);

        RK[i*8 + 4]= ROL( RK[(i-1)*8 + 4], (i+OFFSET1)%8) + ROL (CONSTANT4, (i+OFFSET3)%8);
        RK[i*8 + 5]= ROL( RK[(i-1)*8 + 5], (i+OFFSET5)%8) + ROL (CONSTANT5, (i+OFFSET7)%8);
        RK[i*8 + 6]= ROL( RK[(i-1)*8 + 6], (i+OFFSET1)%8) + ROL (CONSTANT6, (i+OFFSET3)%8);
        RK[i*8 + 7]= ROL( RK[(i-1)*8 + 7], (i+OFFSET5)%8) + ROL (CONSTANT7, (i+OFFSET7)%8);
    }
}

//
void ROUND_FUNC(uint8_t *intermediate, uint8_t *RK, uint8_t index, uint8_t loop_indx, uint8_t
offset){
    intermediate[index] = RK[loop_indx*8 + index] ^ intermediate[index];
    intermediate[index] = RK[loop_indx*8 + index] ^ intermediate[index-1] + intermediate[index];
    ROL(intermediate[index], offset);
}



//
void block_encryption(uint8_t* PT, uint8_t* RK, uint8_t* CT){
    uint32_t i=0;
    uint32_t j=0;
```

```
    uint8_t intermediate[8]={0,};
    uint8_t tmp=0;

    for(i=0;i<8;i++){
        intermediate[i] = PT[i];
    }

    for(i=0;i<NUM_ROUND;i++){
        for(j=7;j>0;j--){
            ROUND_FUNC(intermediate,RK,j,i,j);
        }

        tmp = intermediate[0];
        for(j=1;j<8;j++){
            intermediate[j-1] = intermediate[j];
        }
        intermediate[7] = tmp;
    }

    for(i=0;i<8;i++){
        CT[i] = intermediate[i];
    }

}

//
void CTR_mode(uint8_t* PT, uint8_t* MK, uint8_t* CT, uint8_t num_enc){
    uint32_t i=0;
    uint32_t j=0;
    uint8_t intermediate[8] ={0,};
    uint8_t intermediate2[8] ={0,};
    uint8_t ctr = 0;

    uint8_t RK[8* NUM_ROUND]={0,};

    //key schedule
    key_scheduling(MK, RK);

    //nonce setting
    intermediate[1] = NONCE1;
    intermediate[2] = NONCE2;
    intermediate[3] = NONCE3;
    intermediate[4] = NONCE4;
    intermediate[5] = NONCE5;
    intermediate[6] = NONCE6;
    intermediate[7] = NONCE7;
```

```
    for(i=0;i<num_enc;i++){
        //ctr setting
        intermediate[0] = ctr++;
        block_encryption(intermediate,RK,intermediate2);
        for(j=0;j<8;j++){
            CT[i*8+j] = PT[i*8+j] ^ intermediate2[j];
        }
    }
}

//
void POLY_MUL_RED(uint8_t* IN1, uint8_t* IN2, uint8_t* OUT){
    uint64_t* in1_64_p = (uint64_t*) IN1;
    uint64_t* in2_64_p = (uint64_t*) IN2;
    uint64_t* out_64_p = (uint64_t*) OUT;

    uint64_t in1_64 = in1_64_p[0];
    uint64_t in2_64 = in2_64_p[0];
    uint64_t one = 1;

    uint64_t result[2] = {0,};

    int32_t i=0;

    for(i=0;i<64;i++){
        if( (( one<<i ) & in1_64) > 0  ){
            result[0] ^= in2_64<<i;
            if(i!=0){
                result[1] ^= in2_64>>(64-i);
            }
        }
    }

    // reduction
    result[0] ^= result[1];
    result[0] ^= result[1]<<9;
    result[0] ^= result[1]>>55;
    result[0] ^= (result[1]>>55)<<9;

    out_64_p[0] = result[0];
}

//
void AUTH_mode(uint8_t* CT, uint8_t* AUTH, uint8_t num_auth){
    uint8_t AUTH_nonce[8] = {0,};
    uint8_t AUTH_inter[8] = {0,};
```

```
    uint32_t i, j;

    //nonce setting
    AUTH_nonce[0] = num_auth;
    AUTH_nonce[1] = num_auth ^ NONCE1;
    AUTH_nonce[2] = num_auth & NONCE2;
    AUTH_nonce[3] = num_auth | NONCE3;
    AUTH_nonce[4] = num_auth ^ NONCE4;
    AUTH_nonce[5] = num_auth & NONCE5;
    AUTH_nonce[6] = num_auth | NONCE6;
    AUTH_nonce[7] = num_auth ^ NONCE7;

    POLY_MUL_RED(AUTH_nonce, AUTH_nonce, AUTH_inter);

    for(i=0;i<num_auth;i++){
        for(j=0;j<8;j++){
            AUTH_inter[j] ^= CT[i*8 + j];
        }
        POLY_MUL_RED(AUTH_nonce, AUTH_inter, AUTH_inter);
        POLY_MUL_RED(AUTH_inter, AUTH_inter, AUTH_inter);
    }

    for(i=0;i<8;i++){
        AUTH[i] = AUTH_inter[i];
    }
}

void ENC_AUTH(uint8_t* PT, uint8_t* MK, uint8_t* CT, uint8_t* AUTH, uint8_t length_in_byte){
    uint8_t num_enc_auth = length_in_byte / 8;

    CTR_mode(PT, MK, CT, num_enc_auth);
    AUTH_mode(CT,AUTH,num_enc_auth);
}

void ENC_AUTH_IMP(uint8_t* PT, uint8_t* MK, uint8_t* CT, uint8_t* AUTH, uint8_t length_in_byte){
    //uint8_t num_enc_auth = length_in_byte / 8;

    //CTR_mode(PT, MK, CT, num_enc_auth);
    //AUTH_mode(CT,AUTH,num_enc_auth);
}


//PT range (1-255 bytes)
#define LENGTH0 64
#define LENGTH1 128
#define LENGTH2 192
```

```
int main(int argc, const char * argv[]) {
    uint8_t PT0[LENGTH0]={
        0x42,0xFB,0x9F,0xE0,0x59,0x81,0x5A,0x81,0x66,0xA1,0x0E,0x5C,0x4E,0xB4,0xDA,0xEC,
        0x2F,0xF5,0x60,0x7E,0x8A,0xED,0x3B,0xCA,0x2B,0xD5,0x82,0x69,0x1D,0xC3,0x84,0x13,
        0x0E,0xA6,0x6A,0x10,0xB3,0x3C,0xB4,0x4E,0x9A,0x80,0x4F,0x61,0x06,0x82,0x17,0xF4,
        0xCA,0x76,0xBA,0x84,0xE2,0xDC,0xC9,0x66,0x4F,0xA5,0x07,0x8C,0x8E,0x36,0xD1,0x97};
    uint8_t PT1[LENGTH1]={
        0x4E,0xE2,0xB3,0x54,0x05,0x90,0xB0,0xFD,0x87,0x9B,0x30,0xAB,0x19,0xC4,0x66,0x8F,
        0x2F,0x22,0x30,0xA8,0x5E,0x23,0x5B,0x0B,0xB1,0xEB,0xD6,0xAD,0x10,0x0F,0x33,0x25,
        0x90,0x66,0xC5,0x82,0xE7,0x1B,0x47,0xCA,0xBE,0x61,0xA3,0x91,0xDB,0xC2,0x19,0x97,
        0x04,0x6A,0x73,0x02,0x08,0x70,0x28,0x44,0x38,0x69,0xB5,0xCE,0x55,0x95,0xCB,0x90,
        0xD3,0x8A,0xE2,0x60,0x89,0x2A,0x15,0xCA,0x36,0x9B,0x73,0xEC,0xEF,0xD0,0x43,0x0B,
        0xA7,0xFC,0xDA,0x4B,0xAB,0xE7,0xB3,0xC9,0xB7,0xF5,0xD8,0x86,0xA2,0xC5,0x41,0x5D,
        0x18,0xC3,0x0C,0x30,0xDB,0xC2,0xFE,0x68,0x42,0x3D,0x33,0xFA,0x6D,0xA0,0xD3,0x6F,
        0x03,0x1F,0x87,0x75,0x3C,0x1E,0x81,0x58,0x88,0xAA,0xF4,0x90,0x56,0xA1,0x93,0x64};
    uint8_t PT2[LENGTH2]={
        0xA7,0xF1,0xD9,0x2A,0x82,0xC8,0xD8,0xFE,0x43,0x4D,0x98,0x55,0x8C,0xE2,0xB3,0x47,
        0x17,0x11,0x98,0x54,0x2F,0x11,0x2D,0x05,0x58,0xF5,0x6B,0xD6,0x88,0x07,0x99,0x92,
        0x48,0x33,0x62,0x41,0xF3,0x0D,0x23,0xE5,0x5F,0x30,0xD1,0xC8,0xED,0x61,0x0C,0x4B,
        0x02,0x35,0x39,0x81,0x84,0xB8,0x14,0xA2,0x9C,0xB4,0x5A,0x67,0x2A,0xCA,0xE5,0x48,
        0xE9,0xC5,0xF1,0xB0,0xC4,0x15,0x8A,0xE5,0x9B,0x4D,0x39,0xF6,0xF7,0xE8,0xA1,0x05,
        0xD3,0xFE,0xED,0xA5,0xD5,0xF3,0xD9,0xE4,0x5B,0xFA,0x6C,0xC3,0x51,0xE2,0x20,0xAE,
        0x0C,0xE1,0x06,0x98,0x6D,0x61,0xFF,0x34,0xA1,0x1E,0x19,0xFD,0x36,0x50,0xE9,0xB7,
        0x81,0x8F,0xC3,0x3A,0x1E,0x0F,0xC0,0x2C,0x44,0x55,0x7A,0xC8,0xAB,0x50,0xC9,0xB2,
        0xDE,0xB2,0xF6,0xB5,0xE2,0x4C,0x4F,0xDD,0x9F,0x88,0x67,0xBD,0xCE,0x1F,0xF2,0x61,
        0x00,0x8E,0x78,0x97,0x97,0x0E,0x34,0x62,0x07,0xD7,0x5E,0x47,0xA1,0x58,0x29,0x8E,
        0x5B,0xA2,0xF5,0x62,0x46,0x86,0x9C,0xC4,0x2E,0x36,0x2A,0x02,0x73,0x12,0x64,0xE6,
        0x06,0x87,0xEF,0x53,0x09,0xD1,0x08,0x53,0x4F,0x51,0xF8,0x65,0x8F,0xB4,0xF0,0x80};

    uint8_t CT_TMP[LENGTH2]={0,};

    uint8_t CT0[LENGTH0]={
        0x08,0x41,0x8B,0x3C,0x6F,0xAB,0x1A,0xBD,0x68,0x56,0x33,0xD9,0x9B,0x3D,0xC7,0xDF,
        0x69,0x01,0x9E,0xB4,0x46,0xC9,0x55,0xE4,0xA1,0x2C,0x85,0xD2,0x5E,0x58,0x8F,0x46,
        0x4C,0x10,0xCA,0x10,0xB9,0xBA,0x20,0xC6,0xCC,0xC3,0xD6,0x58,0x0F,0x27,0x76,0x8B,
        0x14,0xB6,0xA0,0x2A,0x42,0x7C,0xDB,0x7C,0xFD,0x40,0xE4,0xE3,0xF9,0x41,0xDE,0x06};
    uint8_t CT1[LENGTH1]={
        0x60,0x50,0xF3,0x58,0xB7,0xFA,0xBC,0xB1,0xA1,0x14,0xF3,0xDE,0x1C,0xE7,0xE3,0xC4,
        0xC1,0x8E,0xDE,0x32,0x92,0x5B,0x51,0xF9,0x1F,0x92,0xEF,0xDE,0x37,0x3E,0xC8,0xCC,
        0x9E,0xB0,0x91,0x5A,0x91,0x7D,0x47,0x5A,0xB8,0x22,0x94,0xC0,0x02,0x4D,0x60,0xD8,
        0xFA,0x9A,0x81,0x34,0x98,0xB4,0x36,0x52,0xC6,0x24,0x78,0x11,0xAE,0xC8,0xE4,0xED,
        0xAD,0xE0,0xFA,0xC4,0xF3,0x88,0x71,0xEE,0xA0,0x7C,0x08,0xA1,0x62,0x0B,0x3E,0x08,
        0xF9,0x18,0x9C,0x99,0x7F,0x57,0xD1,0x43,0x29,0x04,0xE9,0x8D,0x8D,0x6C,0x52,0xBC,
        0x86,0x0D,0x60,0x00,0x85,0xDC,0x26,0x60,0xB4,0xE6,0xFC,0xF3,0xCC,0xE7,0x22,0x68,
        0x8D,0x77,0x2D,0x1B,0x64,0x02,0x37,0x76,0x26,0x8F,0xB1,0x07,0x35,0x14,0xD4,0xB1};
    uint8_t CT2[LENGTH2]={
```

```
        0x59,0xD7,0x41,0x1E,0xA8,0x96,0xB8,0x6E,0x97,0xE8,0xA3,0xCA,0x59,0x31,0xA2,0x60,
        0xC5,0xF9,0x6E,0x8A,0xAB,0xD1,0xA3,0x57,0xB0,0xE2,0xB2,0x07,0xFF,0xC2,0xB6,0xE3,
        0x1E,0x31,0x9E,0xC1,0x3D,0x2F,0xF7,0x79,0x93,0x91,0x6E,0xE3,0x04,0x76,0xD9,0x08,
        0xA8,0xC1,0x23,0xAB,0xEC,0x1C,0x66,0x6C,0xDC,0x17,0x07,0xFA,0xB1,0x53,0x56,0x55,
        0x37,0x3B,0xD1,0xFC,0x46,0xB3,0xF2,0xDD,0xCF,0x90,0x3A,0xC1,0x3A,0x93,0xA8,0xEA,
        0xA1,0xDE,0x73,0x13,0x29,0xBB,0x5F,0xBE,0x93,0xF5,0x4D,0x6A,0x5E,0x4F,0xA7,0x37,
        0xFA,0xBB,0x22,0x00,0xCB,0x0B,0xF3,0x30,0xED,0x47,0xBE,0x5E,0xB7,0xAF,0xC4,0x3C,
        0x2B,0x43,0xA1,0x78,0xFE,0x43,0x6A,0x1A,0xC4,0x4E,0xFF,0x1D,0x18,0xB1,0xE2,0x37,
        0x00,0xC4,0x9E,0x11,0x18,0x22,0xBF,0xBD,0x6B,0xFD,0xAC,0xB2,0x2B,0x7C,0x53,0x56,
        0x32,0x76,0xBE,0x99,0xC3,0xDE,0xAA,0x80,0x8F,0xB0,0x37,0xC6,0x66,0xCD,0xD6,0xCF,
        0xAD,0xB0,0xF9,0x92,0xD8,0x34,0xB8,0x68,0x42,0xC7,0x25,0x59,0xCA,0x75,0x41,0x75,
        0xCC,0x83,0xC5,0x49,0x31,0x25,0x4A,0x8D,0xEF,0xE2,0xD5,0xE8,0xE4,0x5D,0x33,0x2D};

    uint8_t AUTH_TMP[8]={0,};

    uint8_t AUTH0[8]={0x1B,0x82,0x84,0xE5,0xE1,0x1D,0x78,0x4A};
    uint8_t AUTH1[8]={0x48,0x80,0x25,0x8E,0x7D,0xA6,0x71,0xD2};
    uint8_t AUTH2[8]={0x52,0x8E,0xF0,0x79,0xD4,0xD9,0x05,0x05};

    uint8_t MK0[8]={0xF5,0xD3,0x8D,0x7F,0x87,0x58,0x88,0xFC};
    uint8_t MK1[8]={0x47,0x33,0xC9,0xFC,0x8E,0x35,0x88,0x11};
    uint8_t MK2[8]={0xD8,0x99,0x28,0xC3,0xDA,0x29,0x6B,0xB0};

    uint32_t i=0;

    long long int cycles, cycles1, cycles2;

    printf("--- TEST VECTOR ---\n");

    ENC_AUTH(PT0, MK0, CT_TMP, AUTH_TMP, LENGTH0);

    for(i=0;i<LENGTH0;i++){
        if(CT_TMP[i] != CT0[i]){
            printf("wrong result.\n");
            return 0;
        }
        CT_TMP[i] = 0;
    }
    for(i=0;i<8;i++){
        if(AUTH_TMP[i] != AUTH0[i]){
            printf("wrong result.\n");
            return 0;
        }
        AUTH_TMP[i] = 0;
    }

    ENC_AUTH(PT1, MK1, CT_TMP, AUTH_TMP, LENGTH1);
```

```
for(i=0;i<LENGTH1;i++){
    if(CT_TMP[i] != CT1[i]){
        printf("wrong result.\n");
        return 0;
    }
    CT_TMP[i] = 0;
}
for(i=0;i<8;i++){
    if(AUTH_TMP[i] != AUTH1[i]){
        printf("wrong result.\n");
        return 0;
    }
    AUTH_TMP[i] = 0;
}

ENC_AUTH(PT2, MK2, CT_TMP, AUTH_TMP, LENGTH2);

for(i=0;i<LENGTH2;i++){
    if(CT_TMP[i] != CT2[i]){
        printf("wrong result.\n");
        return 0;
    }
    CT_TMP[i] = 0;
}
for(i=0;i<8;i++){
    if(AUTH_TMP[i] != AUTH2[i]){
        printf("wrong result.\n");
        return 0;
    }
    AUTH_TMP[i] = 0;
}

printf("test pass. \n");

//
printf("--- BENCHMARK ---\n");
cycles=0;
cycles1 = cpucycles();
for(i=0;i<BENCH_ROUND;i++){
    ENC_AUTH(PT2, MK2, CT_TMP, AUTH_TMP, LENGTH2);
}
cycles2 = cpucycles();
cycles = cycles2-cycles1;
printf("Original implementation runs in ..... %8lld cycles", cycles/BENCH_ROUND);
```

```
    printf("\n");

    cycles=0;
    cycles1 = cpucycles();
    for(i=0;i<BENCH_ROUND;i++){
        ENC_AUTH_IMP(PT2, MK2, CT_TMP, AUTH_TMP, LENGTH2);
    }
    cycles2 = cpucycles();
    cycles = cycles2-cycles1;
    printf("Improved implementation runs in ..... %8lld cycles", cycles/BENCH_ROUND);
    printf("\n");

    return 0;
}
```

주의사항

1) 구현 타겟 플랫폼은 64-비트 Intel 프로세서이며 C언어 만을 사용한다. 특히 기본적인 C언어만을 사용하기 때문에 아래 기능은 사용하지 않는다.

- AVX, SSE와 같은 SIMD 명령어 셋
- Carry-less Multiplication과 같은 polynomial 곱셈 가속화 명령어 셋
- assembly (인라인 어셈블리) 및 intrinsic 명령어 셋
- AES 및 SHA 가속기 명령어셋

2) 위의 C 코드에서 빨간색으로 표현된 부분만 수정이 가능하다. 즉 ENC_AUTH_IMP함수의 입출력 형식만을 유지하고 해당 함수 내부에 들어가는 함수와 변수들은 새롭게 정의 및 구성가능하다. 추가적으로 main함수도 그대로 유지한다.

3) 프로그램 컴파일과 실행에는 다음 명령어들을 사용한다. Linux 상에서 gcc를 활용하여 컴파일해야 하며 Windows 및 Mac 환경 상에서의 코딩은 허용하지 않는다. 결과물은 contest.c 하나만을 받으며 파일 쪼개기와 makefile 그리고 파일 형식 변환은 허용하지 않는다.

```
>> gcc -o contest contest.c
>> ./contest
```

현재 프로그램은 최적화 구현이 안되어 있기에 실행 결과는 아래와 같다. 즉 실제 결과에서는 두번째 항목 (Improved implementation runs in ....)에 결과가 도출되어야 한다.

```
--- TEST VECTOR ---
test pass.
--- BENCHMARK ---
Original implementation runs in .................   278083 cycles
Improved implementation runs in .................        4 cycles
```

4) 본 레퍼런스 코드에 대한 테스팅 환경은 아래와 같다. vmware에서 테스트를 수행해도 되며 Linux 환경을 실제 설치하여 수행해도 상관없다. 단 결과물은 Linux 환경에서 동작하는 C 코드만을 허용한다.

- Ubuntu 22.04 LTS (https://ubuntu.com/download/desktop)
- vmplayer (https://www.vmware.com/go/getplayer-win)

5) 결과물은 다음 2종을 포함한다.

- C 코드 (테스트 벡터 확인 과정, 벤치마크 과정)
- 문서 (구현 기법 상세, 테스트 벡터 확인 결과, 벤치마크 결과)

7) 평가방법은 다음과 같다.
- 테스트 벡터 통과 (30점, 절대 평가)
- 문서화 (30점, 절대평가)
- 벤치마크 결과 (40점, 상대평가)