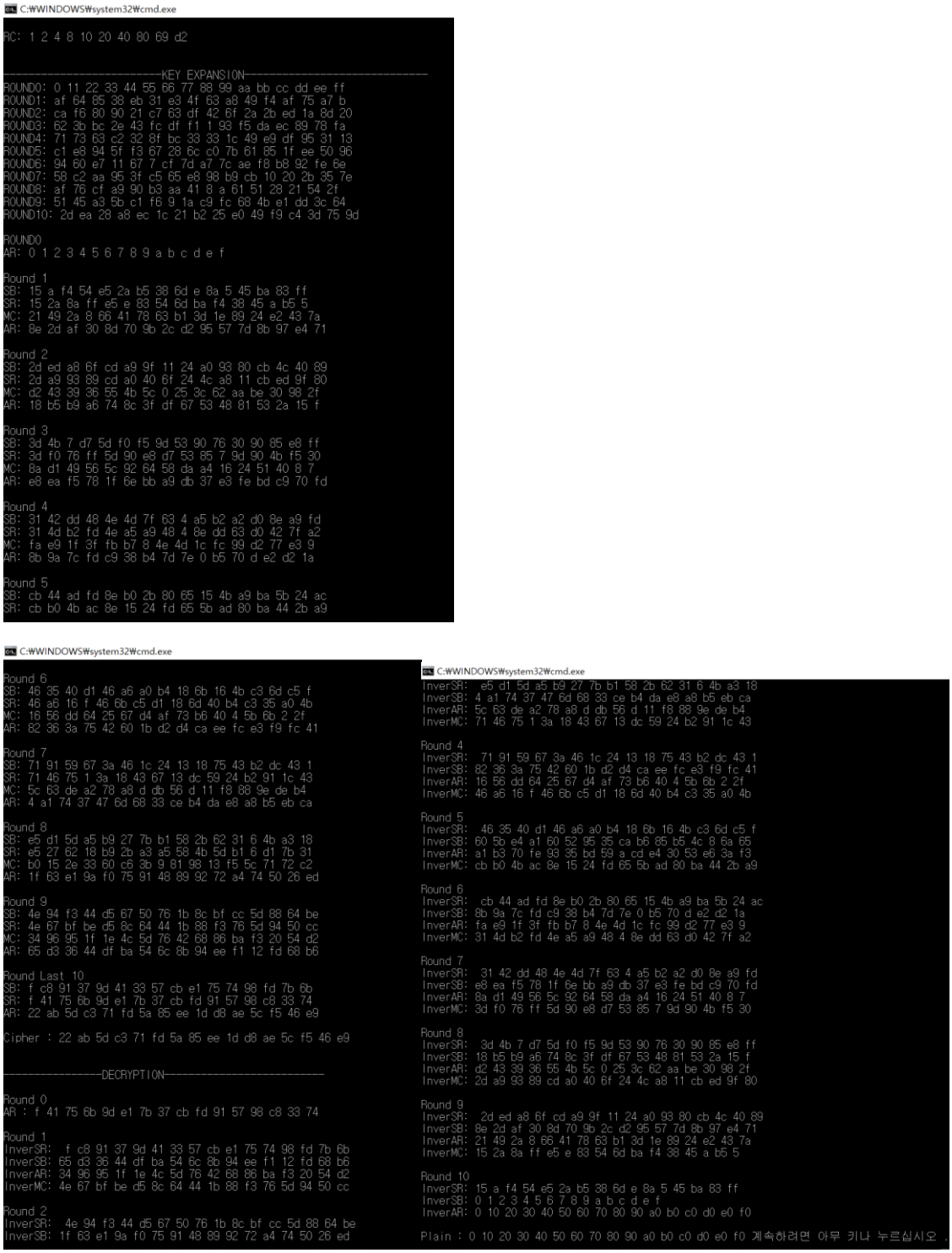
Modified AES 구현

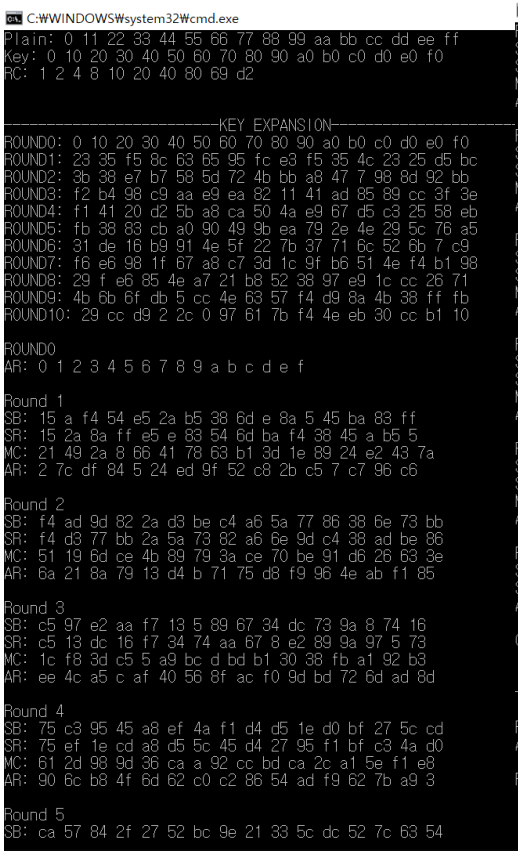
12141595 이용준

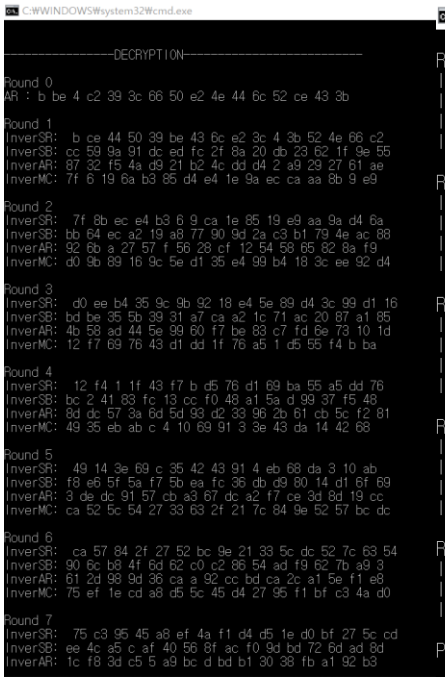
구현 언어 : C++,C

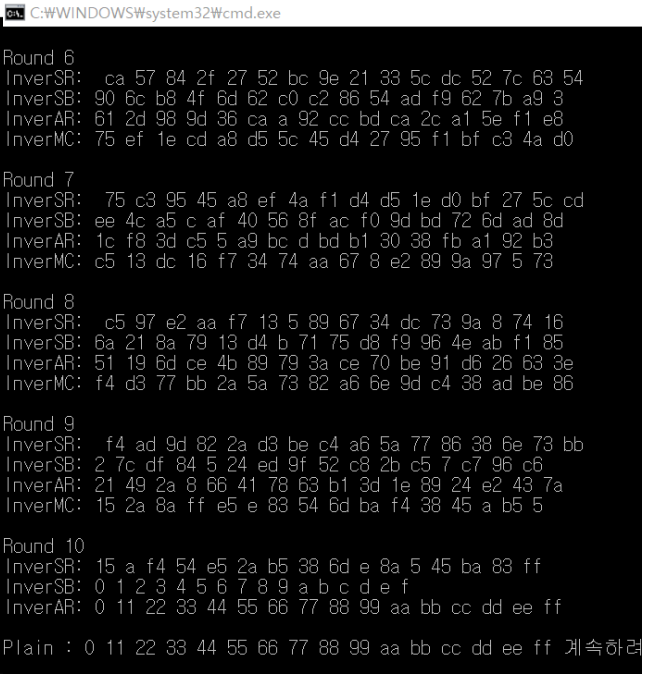
사용 IDE : Visual Studio 2017

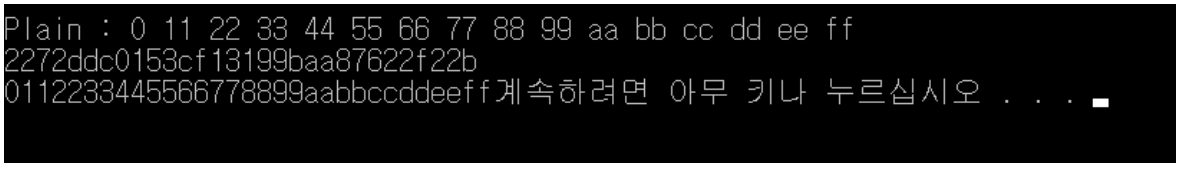


<주어진 key와 plain으로 돌렸을 때 주어진 결과와 일치>









<주어진 bin 파일을 실행한결과> <마지막은 bin에 정상적으로 입력되었음>

```
>uint8_t galoidadd(int a, int b) // galois field 위에서의 덧셈은 xor
{
        return a ^ b;
}
>uint8_t gmul(uint8_t a, uint8_t b) {
        uint8_t p = 0;
        uint8_t counter;
        uint8_t hi_bit_set;
        for (counter = 0; counter < 8; counter++) {
                if ((b & 1) == 1)
                        p ^= a;
                hi_bit_set = (a & 0x80);
                a <<= 1;
                if (hi_bit_set == 0x80)
                        a ^= 0x69;
                b >>= 1;
        }
        return p;
}
```

모든 숫자들이 8bit단위이므로 uint8_t 8bit 정수형 데이터 타입을 사용했다. 캐리가 일어나도 크게 결과에 영향이없다.

핵심적으로 galois 필드위에서 연산을 정의하여야 했다. 덧셈은 단순하게 xor연산이다. 곱셈의 경우 쉬프트 하면서 연산을 해주는데 작은 수의 숫자중 1로 set된 자리수 만큼 쉬프트한다.irr polynominal을 고려해 최상위 비트를 넘어갈 때 mod연산은 xor로 해주었다.

```
>uint8_t ExtendedEuclideanAlgo(uint8_t a) // 주어진값의 inverse 를 반환
{
	int a3 = 361; // irr polynominal 0x169 or x^8+x^6+x^5+x^3+x+1
	int b3 = (int)a;
	int c1 = 0, a2 = 0, b2 = 0;
	int flag = 0;
	if (a == 1) return 1;
	while (b3 > 1)
	{
		int  q = 0;
		int tmp = b3;
		int q1;
		while ((a3>tmp) || (msb(a3) >= msb(tmp)))
		{
			q1 = 1;
			while (!(msb(b3) == msb(a3)))
			{
				b3 <<= 1;
				q1 *= 2;
			}
			q += q1;
			a3 ^= b3;
			b3 = tmp;
		}
		b3 = a3;
		a3 = tmp;
		if (!flag) b2 = 1;
		int n = msb(q);
		tmp = b2; int c2 = 0;
		while (n >= 1)
		{
			if (((1 << (n - 1)) & q) != 0)
			{
				c1 = (b2 << (n - 1));
				c2 ^= c1;
			}
			n--;
		}
		b2 = c2 ^ a2;
		a2 = tmp;
		flag++;
	}
	return (uint8_t)b2;
}

//가장 중요한 함수이다. 주어진 irr다항식 아래에서 곱셈에 대한 역원을 찾는 확장 유클리드
알고리즘이다. 주어진 매개변수가 역원을 알고싶은 다항식이다. 큰 다항식에서 작은 다항식을 계속
mod연산을 해가야하므로 최상위비트를 저장하는 msb함수를 정의하였다.
int msb(int x) //most set bit
{
	int a = x;
	int tmp = 0;
	while (1){if (a == 0) break;a >>= 1;tmp++;}
```

```
        return tmp;
}

>void sboxgenerate()//sbox inverse-sbox generate function
{
        uint8_t input, raw = 0, column = 0; int num;
        uint8_t output[8];
        uint8_t b[8];
        uint8_t m = 0, n = 0;
        for (int k = 0; k < 256; k++) //00 ~ FF까지
        {
                num = 0;
                input = ExtendedEuclideanAlgo(k); // multiply matrix by inverse of given
number
                column = k % 16; raw = k / 16; // sbox raw,column
                for (int j = 0; j<8; j++)
                {
                        b[j] = input % 2; // inverse num to binary
                        input = input / 2;
                }
                int t;
                for (int i = 0; i < 8; i++)
                {
                        output[i] = b[i] ^ b[(i + 4) % 8] ^ b[(i + 5) % 8] ^ b[(i + 6) % 8] ^
b[(i + 7) % 8] ^ sconstan[i]; // matrix multiple and add changed constant
                        t = (int)output[i];
                        num += t * pow(2, i); //sbox
                }
                sbox[raw][column] = num;
        }
}
```

//확장 유클리드 알고리즘을 이용해 sbox를 생성한다. 2진수로 표현하여 연산하였다.

```
>void SubstituteBytes()
{
        for(int i=0;i<4;i++)
                for (int j = 0; j < 4; j++)
                {
                        state[j][i] = sbox[state[j][i]/16][state[j][i] % 16]; // read s box
                }

        for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                        printf("%x ", (int)state[j][i]);
}
```

SubstituteBytes함수는 16진수로 표현될 때 앞자리는 raw를 뒷자리는 column 을 읽는다.

```
>void inverseSboxgenerate()
{
        uint8_t input, raw = 0, column = 0; int num;
        uint8_t output[8];
        uint8_t s[8];
        uint8_t b[8];
        uint8_t m = 0, n = 0;
        for (int k = 0; k < 256; k++) //00 ~ FF까지
        {
                num = 0;
                input = k; // multiply matrix by inverse of given number
                column = k % 16; raw = k / 16; // sbox raw,column
                for (int j = 0; j<8; j++)
                {
                        b[j] = input % 2; // inverse num to binary
                        input = input / 2;
                }
                for (int j = 0; j < 8; j++)
                {
                        b[j] ^= sconstan[j];
                }
                int t;
                for (int i = 0; i < 8; i++)
                {
                        output[i] =b[(i+2)%8] ^ b[(i + 5) % 8] ^  b[(i + 7) % 8]; // inverse
of affine transform
                        /* inverse matrix of s box matrix

                        */
                        t = (int)output[i];
                        num += t * pow(2, i);
                }
                num = (int)ExtendedEuclideanAlgo(num); // multiplitive inverse
                inversesbox[raw][column] = num;
        }
}
```

inverseSubstitute함수는 연산의 순서가 거꾸로이다. 그러므로 주어진 상수를 먼저 xor한후 아핀변환의 역변화을 연산한후에 확장유클리드알고리즘으로 곱셈에 대한 역원을 반환한다. 마찬가지로 16진수진 표현되었을 때 앞자리는 raw뒷자리는 column이다.

```
>void MixColumns()
{
        uint8_t tmp1, tmp2, tmp3, tmp4;
        for (int i = 0; i < 4; i++)
        {
                tmp1 = state[0][i];
                tmp2 = state[1][i];
                tmp3= state[2][i];
                tmp4= state[3][i];
                state[0][i] =gmul(2,tmp1)^gmul(3,tmp2)^tmp3^tmp4 ;
                state[1][i] = tmp1^gmul(2,tmp2)^gmul(3,tmp3)^tmp4;
                state[2][i] = tmp1^tmp2^gmul(2,tmp3)^gmul(3,tmp4);
                state[3][i] = gmul(3,tmp1)^tmp2^tmp3^gmul(2,tmp4);
        }
        for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                        printf("%x ", (int)state[j][i]);
}
```

정의한 갈루아필드에서의 곱셈연산을 수행한다.

```
>void ShiftRows()
{
        uint8_t tmp[4];
        for (int i = 1; i < 4; i++)
        {
                for (int j = 0; j < 4; j++)
                        tmp[j] = state[i][j];
                for(int k=0;k<4;k++)
                        state[i][k] = tmp[(k + i)%4];
        }
        for (int i = 0; i < 4; i++)
                for (int j = 0; j < 4; j++)
                        printf("%x ", (int)state[j][i]);
}
```

두번째 행은 왼쪽으로 한칸, 세번째 행은 왼쪽으로 두칸, 네번째 행은 왼쪽으로 세칸 혹은 오른쪽으로 한칸 쉬프트한다.

```
>void keyexpansion()
{
        for (int i = 0; i < 4; i++)
        {
                for (int j = 0; j < 4; j++)
                {
                        word[i][j] = key[0][4 * i + j];
                }
        }
        for (int i = 4; i < 44; i++)
        {
                uint8_t temp[4];
                if (i % 4 == 0)
                {
                        for (int j = 0; j < 4; j++)
                        {

                                if (j == 0)
                                        word[i][0] = word[i - 4][0] ^ sbox[word[i - 1][1] /
16][word[i - 1][1] % 16] ^ RC[(i/4)-1];
                                else
                                {
                                        word[i][j] = word[i - 4][j] ^ sbox[word[i - 1][(j +
1) % 4] / 16][word[i - 1][(j + 1) % 4] % 16];
                                }
                                key[i / 4][(i%4)*4+j]=word[i][j];
                        }

                }
                else
                {
                        for (int j = 0; j < 4; j++)
                        {
                                word[i][j] = word[i - 1][j] ^ word[i - 4][j];
                                key[i / 4][(i % 4) * 4 + j] = word[i][j];
                        }
                }
        }

}
```

//주어진 state의 열이 하나의 단위를 이룬다. 원래는 cogetination으로 하나의 자료에 넣으려고 했으나 roundfunction을 쉽게 사용하기 위해 word도 2차원 배열로 저장했다. 하지만 이 때문에 index가 매우 복잡하게 되었다. RoundConstant의경우 바로 전 index에 2배를 하면서 생성할 수 있다. 역시 정의한 갈루아 연산으로 쉽게 구현이 가능하다.

```
for (int i = 1; i < 10; i++)
        {
                RC[i] = gmul(RC[i - 1], 2);
                m = (int)RC[i];
                printf("%x ", m);
        }
```

암호화단계

```
AddRoundKey(key[0]);
for (int i = 1; i < 10; i++) //Round 1 to 9
{

        printf("₩n₩nRound %d₩nSB: ",i);
        SubstituteBytes();
        printf("₩nSR: ");
        ShiftRows();
        printf("₩nMC: ");
        MixColumns();
        printf("₩nAR: ");
        AddRoundKey(key[i]);
}
//Round 10 without shiftrows.
printf("₩n₩nRound Last 10₩nSB: ");
SubstituteBytes();
printf("₩nSR: ");
ShiftRows();
printf("₩nAR: ");
AddRoundKey(key[10]);
printf("₩n₩nCipher : ");
```

//메인함수의 일부분이다. 암호화과정은 다음과 같이 진행되고 복화과정은 정확히 반대방향으로 같은 횟수만큼 진행한다.

복호화단계

```
inverseSboxgenerate(); // inverse sbox 생성
printf("Round 0₩nAR : ");
AddRoundKey(key[10]);
for (int i = 9; i > 0; i--)
{
        printf("₩n₩nRound %d₩nInverSR:  ", 10 - i); // 1~9 Round
        InverseShiftRows();
        printf("₩nInverSB: ");
        InverseSubstituteBytes();
        printf("₩nInverAR: ");
        AddRoundKey(key[i]);
        printf("₩nInverMC: ");
        InverseMixColumns();
}
printf("₩n₩nRound 10₩nInverSR: ");
InverseShiftRows();
printf("₩nInverSB: ");
InverseSubstituteBytes();
printf("₩nInverAR: ");
AddRoundKey(key[0]);
```

inverse함수의 경우 shiftrow는 반대방향으로 shift를 한다. Mixcolumn은 주어진 inverse행렬을 갈루아연산으로 처리한다. Addroundkey는 xor연산이므로 특별한 가공없이 다시한번 roundkey를 xor한다.

파일입출력

```
FILE *cipher;
	uint8_t t2[16];
	cipher = fopen("cipher.bin", "wb"); //cipher.bin 파일에 입력
	for (int i = 0; i < 4; i++)
		for (int j = 0; j < 4; j++)
		{
			t2[4 * i + j] = state[j][i];
			printf("%x ", (int)state[j][i]);
		}
	fwrite(t2, sizeof(uint8_t), 16, cipher);
```

암호가 저장된 배열일 state같은 경우 열을 우선으로 읽으므로 파일입출력에 번거로움이있어 일차원 배열에 순서대로 저장후 파일에 쓰기(입력)를 하였다.

```
FILE *plain;

if ((plain = fopen("plain.bin", "r")) == NULL) return -1;

while ((temp = fgetc(plain)) != EOF)  //read plaintext
	{
		if (m == 4 && n == 3) break;
		if (m > 3) { m = 0; n++; }
		state[m][n] = temp;
		printf("%x ",temp);
		m++;
	}
```

File stream을 통해 입력을 받았다.

해더파일

```
#include <iostream>
#include <fstream>
#include <stdint.h>
#include <algorithm>
#include <vector>
#include <math.h> // pow함수등을 사용
#include <stdio.h>
#include <stdlib.h>
#pragma warning(disable:4996) // c언어의 파일스트림의 경우 비주얼 스튜디오에서 사용하기위해서 sercure warning을 해제
```