한양대학교 최성인

AES, RSA 구현
및
SIMULATION

Hanyang Univ.

崔　聖　仁

AES,RSA 구현
및
SIMULATION

Hanyang Univ.

崔 聖 仁

Choi SungIN
Hanyang University

목차

Ⅱ.그림목차

1. 서론

AES(Advanced Encryption Standard)는 대칭키 암호시스템의 하나로, 2001년 미국 표준 기술 연구소(NIST)에 의해 제정된 암호화 방식이다. NIST가 기밀문서를 안전하게 보호하기위해 DES를 대체할 AES 알고리즘 공모전을 열었고, 5년의 표준화 과정을 거치며 이 과정에서 15개의 알고리즘이 경쟁, Rijndael 암호가 가장 적합한 알고리즘으로 선정되었다. Rijndael은 알고리즘의 개발자인 Vincent Rijmen과 Joan Daemen의 이름을 따서 지은 것으로 AES 표준은 여러 Rijndael 알고리즘 중 블록 크기가 128비트인 알고리즘을 말한다. AES는 ISO/IEC 18033-3 표준에 포함되어 있으며 여러 암호화 패키지에서 사용되고 있다. AES는 또한 미 국가안보국에 의해 1급비밀(Top Secret)에 사용할 수 있도록 승인된 알고리즘 중 최초로 공개되어 있는 알고리즘이다.

RSA는 공개키 암호시스템의 하나로, 암호화뿐만 아니라 전자 서명이 가능한 최초의 알고리즘으로 알려져 있다. RSA가 갖는 전자서명 기능은 인증을 요구하는 전자 상거래 등에 RSA의 광범위한 활용을 가능하게 하였다. 1978년 Ron Rivest, Adi Shamir, Leonard Adleman의 논문에 의해 발표되었으며, 자신들의 이름 앞 글자를 따 RSA라 명명하였다. 1973년 영국 GCHQ에 근무하던 수학자가 RSA 방식을 제일 먼저 개발하였으나 내용은 GCHQ에서 비밀로 취급되었으며, 이후 1997년 세상으로 발표되게 된다. RSA 암호체계의 안정성은 큰 숫자를 소인수 분해 하는 것이 어렵다는 것에 기반을 두고 있다. 1993년 피터 쇼어는 양자 컴퓨터를 이용하여 임의의 정수를 다항 시간 안에 소인수분해 하는 쇼어 알고리즘을 발표하였다. 따라서 양자 컴퓨터가 본격적으로 실용화 될 경우 RSA 암호시스템의 안정성은 크게 떨어질 것으로 예상된다.

이 글에서는 AES 암호와 RSA 암호를 직접 구현하고 이 암호들을 이용하여 평문을 암호문으로 변환하여 본다.

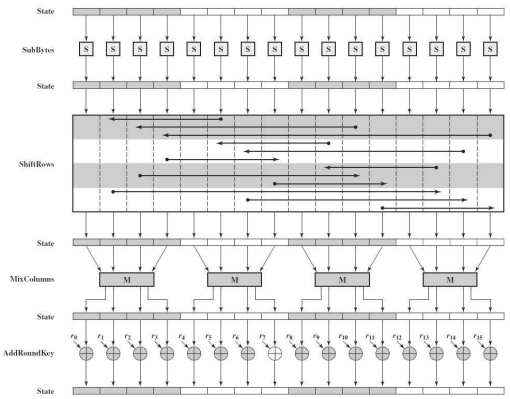

그림 1 AES 암호화 라운드

2. 이론적 배경

가)AES

블록 암호는 고정된 크기의 블록 단위로 암·복호화 연산을 수행하며 각 블록의 연산에는 동일한 키가 이용된다. Shannon의 암호 이론에 의하면 전치와 환자를 반복시켜 암호화하면 평문의 통계적 성질이나 암호 키와의 관계가 나타나지 않아 안전한 암호를 구성할 수 있다. AES는 대표적인 블록 암호 시스템으로 특정 계산 함수를 반복하여 암호화 한다. 이 때, 각 과정에 사용되는 함수를 라운드 함수라고 부르며 라운드 함수에 적용되는 키는 라운드 키라고 부른다.

라운드 키를 매 라운드마다 독립적으로 생성 할 수 없으므로, AES에서는 키 스케쥴링 과정을 진행한다. 입력받은 키를 이용하여 라운드 함수를 수행 할 때 사용할 서브키를 뽑아내는 과정이며 키 확장이라고도 한다. AES 알고리즘은 128/192/256-비트 키 길이를 제공하고, 라운드 수는 키 길이에 따라 각각 10, 12, 14이다. 128bit 키의 경우 10 라운드를 수행해야 하므로 10번의 서브키를 출력해야 한다.

라운드 함수의 경우 다음의 네 단계를 이용하는데, 한 번의 치환과정과 세 번의 대체과정이 들어가 있다. SubBytes 단계는 아래의 S-box라는 표를 이용하여 바이트 단위 형태로 블록을 교환한다. ShiftRows는 바이트의 행을 Shift 연산해준다. 1바이트를 한 칸 단위로 하여, 첫 번째 행은 왼쪽으로 0칸, 두 번째 행은 한 칸, 세 번째 행은 두 칸, 네 번째 행은 세 칸 Shift해준다. 단, 128비트, 192 비트 또는 256 비트에 따라 열의 숫자가 달라진다. MixColumns는 열에 속한 모든 바이트를 순환 행렬을 사용해 함수로 열에 있는 각 바이트를 대체하여 변화시킨다. AddRoundKey는 확장된 키의 일부와 현재 블록을 비트별로 XOR한다.

암호와 복호를 위해서 라운드 키 더하기 단계에서 시작하고, 이후 4단계를 모두 포함하는 라운드들을(128 bit의 경우 9라운드를) 수행하고, 마지막엔 MixColumns 단계가 빠진 3단계로 구성된 라운드를 수행한다.

오직 라운드 키 더하기 단계에서만 키를 사용한다. 그래서 암호와 복호 과정의 시작과 끝은 항상 라운드 키 더하기 단계이다. 시작이나 끝에 수행되는 다른 단계는 키 없이 역방향 계산이 가능하기 때문에 보안을 강화시키는 데는 아무 역할도 하지 못한다. 라운드 키 더하기 단계는 그 자체가 강력하지 못하는 대신 다른 세 단계와 같이 작동하여 비트를 뒤섞는 역할을 한다. 하지만 다른 세 단계들은 키를 사용하지 않으므로 보안성을 제공하는 것은 아니다. 이 암호 단계를 살펴보면 블록

에 XOR 암호화(라운드 키 더하기)를 하고, 그 다음 블록을 뒤섞고(다른 세 단계), 그 뒤에 다시 XOR 암호화를 하는 것으로 이를 번갈아서 적용하는 것을 볼 수 있다. 이 구조는 효과적이고 보안성을 매우 강화시킨다.

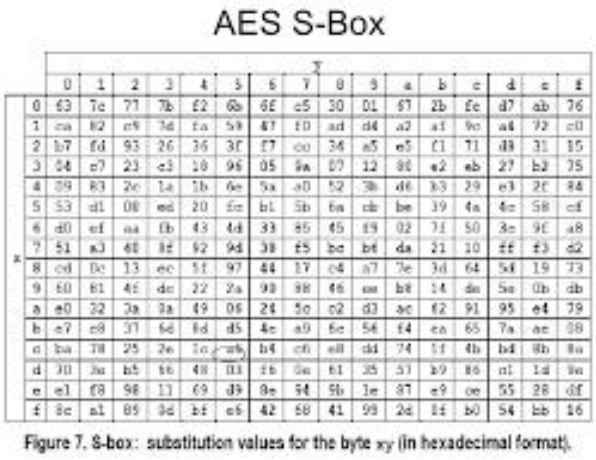

그림 2.1 AES S-Box

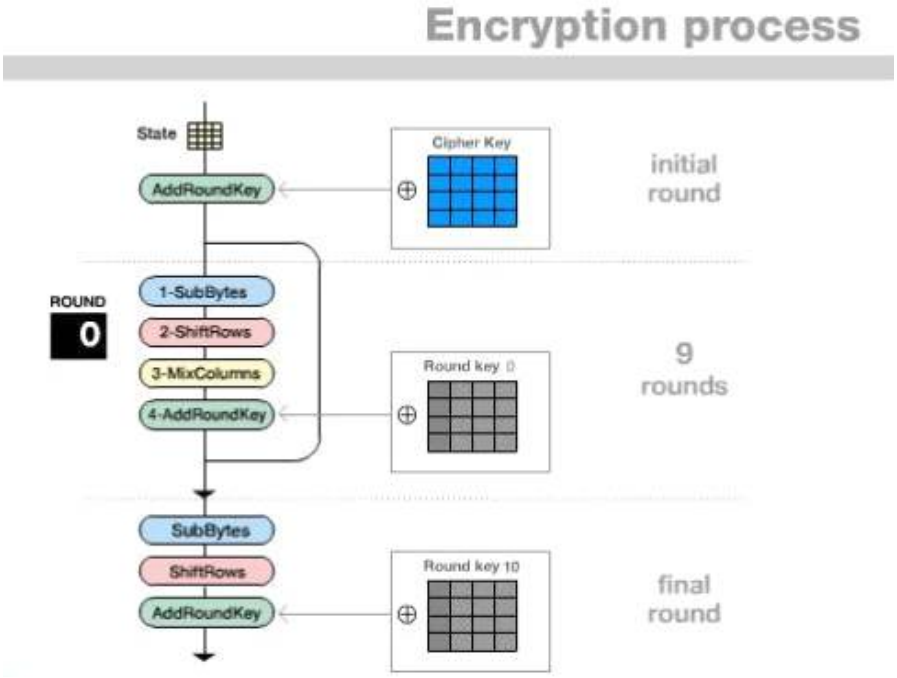

그림 2.2 AES 암호화 과정

나) RSA

RSA는 두 개의 키를 사용한다. 공개키(public key)는 모두에게 알려져 있으며 개인키(private key)는 공개되지 않고 비밀리에 사용한다. 키를 만들 때는 두 개의 큰 소수(보통 140자리 이상의 수)를 이용한다. 두 소수를 p와 q라 하자. 다음의 세 단계을 통하여 공개키와 개인키를 제작한다. 먼저, 두 수의 곱으로 n=pq을 구한다. 이 때, $\phi(n)=(p-1)(q-1)$ 이다. 두 번째는 다음을 만족하는 암호화 키 e를 선택한다. $1<e<\phi(n)$, $\gcd(e,\phi(n))=1$. 마지막으로 $ed \equiv 1 \bmod \phi(n)$ 을 만족하는 복호화 키 d를 계산한다. 이 때, 확장된 유클리드 호제법을 이용한다. 공개키는 {n,e}, 개인키는 {n,d} 이다. 여기서 p와 q의 보안은 매우 중요하다. 이를 가지고 d와 e의 계산이 가능하기 때문이다. 그리하여 공개키와 개인키가 생성이 된 후에는 이 두 숫자를 지워버리는 것이 안전하다.

RSA 암호화 과정은 다음과 같다. 암호화하고자 하는 평문을 숫자로 표시 했을 때 A라고 하자. 이 A를 n보다 작은 숫자로 변환한다. 이 변환법(padding scheme)은 암호를 받는 사람에게도 미리 알려져 있어야 한다. 예로는, 메시지를 토막 내어 하나의 메시지가 일정 수의 비트를 넘지 않게 하는 방법이 있다. 이렇게 변환된 숫자를 a라 하자. 암호를 보낼 때에는 $a^e \equiv c \mod n$ 을 계산하여 c를 보낸다. 이 암호문 c를 받은 사람은 복호화키 d를 이용하여 $c^d \equiv a \mod n$ 를 계산하고, a를 알아낸다. 위에서 설명 한 것처럼, a를 가지고 A를 알아내는 방법이 복호화하는 사람에게 알려져 있어야한다. 이러한 RSA를 통해 이러한 암호화, 복호화 과정이 가능한 이유는 다음과 같다. 오일러 정리에 의해, gcd(a,n)=1 이면, $a^{\phi(n)} \bmod n = 1$이다. 따라서 $\phi(n)=(p-1)(q-1)$ 이고 어떤 k에대해 $ed=1+k\phi(n)$ 이므로,

$c^d \bmod n \equiv a^{ed} \bmod n \equiv a \equiv {}^{1+k\phi(n)} \bmod n \equiv a^1 \bmod n \equiv a \bmod n$ 이다.

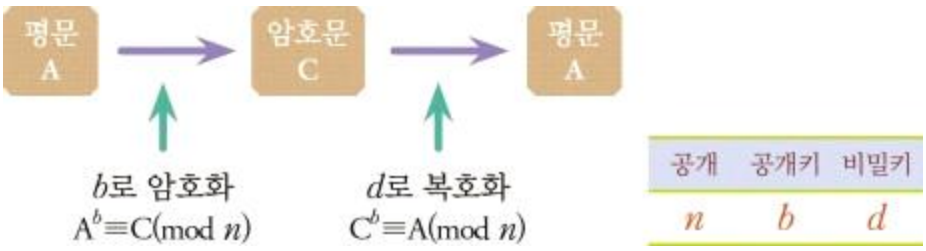

그림 2.3 RSA 암호화 및 복호화 과정

3. 구현 과정

가)AES

```
/*  <128비트AES 암복호화함수>
 *
 *   mode가ENC일경우평문을암호화하고, DEC일경우암호문을복호화하는함수
 *
 *  [ENC 모드]
 *  input    평문바이트배열
 *   output   결과(암호문)이담길바이트배열. 호출하는사용자가사전에메모리를할당하여파라미터로넘어옴
 *  key      128비트암호키(16바이트)
 *
 *  [DEC 모드]
 *  input    암호문바이트배열
 *   output   결과(평문)가담길바이트배열. 호출하는사용자가사전에메모리를할당하여파라미터로넘어옴
 *  key      128비트암호키(16바이트)
 */
void AES128(BYTE *input, BYTE *output, BYTE *key, int mode){

BYTE temp[16];
for (int i = 0; i < BLOCK_SIZE i++) {
temp[i] = input[i];
}
BYTE *roundKey;
roundKey = (BYTE*)malloc(ROUNDKEY_SIZE);
memcpy(output, input, BLOCK_SIZE);

    if(mode == ENC){

expandKey(key, roundKey);
        addRoundKey(temp,&roundKey[0]);


        for(int i =1 ; i<10 ;i++){ //9round 반복
          subBytes(temp,ENC);
shiftRows(temp,ENC);
mixColumns(temp, ENC);
addRoundKey(temp, &roundKey[i*KEY_SIZE]);
        }

subBytes(temp, ENC);
shiftRows(temp,ENC);
addRoundKey(temp, &roundKey[10* KEY_SIZE]);
```

AES의 뼈대골격으로, mode 가 ENC 일 때 평문을 암호화하고, mode가 DEC 일 때 암호문을 복호화 한다.
128bit 암호 key를 사용한다.

ENC mode 일 때, input은 평문 BYTE 배열을 파라미터로 Client가 호출 시 메모리를 할당하여 입력한다. Output은 암호문 BYTE 배열이다.

DEC mode 일 때, input은 암호문 BYTE 배열을 파라미터로 Client가 호출 시 메모리를 할당하여 입력한다.
입력된다. Output은 평문 BYTE 배열이다.

Round 는 10Round로 설정하였으며, round가 달라지는 경우에 대한 변수로 설정해두는 코딩은 설정하지 않았다.

for (int i = 0; i < BLOCK_SIZE i++) {
output[i] = temp[i];
}
 	
free(roundKey);
 	
}else if(mode == DEC){
 	
expandKey(key,roundKey);
 	
addRoundKey(temp,&roundKey[10*KEY_SIZE]);
 	
for(int i = 9 ; i>0 ; i--){
shiftRows(temp,DEC);
subBytes(temp,DEC);
addRoundKey(temp,
&roundKey[i*KEY_SIZE]);
mixColumns(temp, DEC);
 	
}
shiftRows(temp, DEC);
subBytes(temp, DEC);
addRoundKey(temp, &roundKey[0]);
 	
for (int i = 0; i < BLOCK_SIZE i++) {
output[i] = temp[i];
}
 	
}else{
fprintf(stderr, "Invalid mode!\n");
exit(1);
}
}	

void expandKey(BYTE *key, BYTE *roundKey){
 	
BYTE k[KEY_SIZE];
for (int i = 0; i < KEY_SIZE i++) {
k[i] = key[i];
}
 	
BYTE temp[KEY_SIZE/4];
int i;
for(i = 0 ; i < KEY_SIZE ; i++)
{
roundKey[i] = k[i];
	RSA Key Expension 함수이다.
	addRoundKey()에서 사용할 ExpandedKey 값을 생성해준다. 생성된 값들은 roundKey 배열에 저장되어, addRoundKey()에서 사용된다.
	
	KeySize 는 16 BYTE이며, roundKey는 176BYTE가 할당된다.
	
	expandKey에서 추가적으로 구현해야하는

}
 	
for(int i = KEY_SIZE/4 ; i < ROUNDKEY_SIZE/4; i++){
temp[0] = roundKey[4*i -4];
temp[1] = roundKey[4*i -3];
temp[2] = roundKey[4*i -2];
temp[3] = roundKey[4*i -1];
 	
if(i%4 == 0){
rotate(temp);
sub(temp);
 	
addByte(temp,Rcon(i/4),temp);
}
 	
roundKey[4*i+0] =roundKey[4*(i-4)+0]^temp[0];
roundKey[4*i+1] = roundKey[4*(i-4)+1]^temp[1];
roundKey[4*i+2] = roundKey[4*(i-4)+2]^temp[2];
roundKey[4*i+3] = roundKey[4*(i-4)+3]^temp[3];
}
}
	작업은 RotateWord 와 SubstitueWord 로, rotate() 와 sub() 함수로 구현하였다.

void rotate(BYTE *r) {
BYTE temp;
int i;
temp = r[0];
for(i=0 ; i<3 ; i++){
r[i] = r[i+1];
}
r[3] = temp;
}
	ExpandKey에 사용되는 rotate 함수 구현

void sub(BYTE *w) {
int i;
for(i = 0; i< 4; i++) {
w[i] = sbox[w[i]];
 	
}
}
	ExpandKey에 사용되는 rotate 함수 구현
	여기서 사용되는 SBox 배열은 생략하도록 하겠다.

BYTE* addRoundKey(BYTE *block, BYTE *rKey){
 	
for(int i = 0; i <BLOCK_SIZE ; i++){
block[i] = block[i] ^ rKey[i] ;
}
 	
return block
}
	16BYTE block 과 16BYTE roundKey를 더해준다.
	^(exclusive or) 연산을 통해 구현하였다.

BYTE* subBytes(BYTE *block, int mode){
 	
switch(mode){
case ENC:
 	
for(int i = 0; i<BLOCK_SIZE ; i++){
block[i] = sbox[block[i]];
}
 	
break
 	
case DEC:
 	
/* 추가구현*/
for(int i = 0; i<BLOCK_SIZE ; i++){
block[i] = inv_s_box[block[i]];
}
 	
break
 	
default:
fprintf(stderr, "Invalid mode!\n");
exit(1);
}
 	
return block
}
	암호화시에 사용되는 SBox 배열과 복호화시에 사용되는 inv_SBox 배열을 이용하여 구현하였다.

BYTE* shiftRows(BYTE *block, int mode) {
 	
switch (mode) {
 	
case ENC:
 	
/*
행단위 shift 연산
*/
 	
for (int i = 0; i < BLOCK_SIZE / 4; i++) {
for (int j = 0; j < i; j++) {
shiftLeft(block + i); //i번째행을i번왼쪽으로이동
}
 	
}
break
 	
case DEC:
	암호화시에는 ShiftLeft()함수를 이용하여 i(i=0,1,2,3) 번째 행을 I칸씩 왼쪽으로 옮겨주고, 복호화시에는 shiftLeft() 함수를 이용해 4-i 칸씩으로 왼쪽으로 옮겨준다.

for (int i = BLOCK_SIZE / 4; i > 0; i--) {
for (int j = 0; j < i; j++) {
shiftLeft(block + 4 - i); //i번째행을4-i번왼쪽으로이동
}
 	
}
 	
break
 	
default:
fprintf(stderr, "Invalid mode!\n");
exit(1);
}
 	
return block
}	

void shiftLeft(BYTE* block) { // 왼쪽으로1칸이동
BYTE temp;
temp = block[0];
for (BYTE i = 0; i < 3; i++) {
 	
block[4 * i] = block[4 * (i + 1)];
}
block[4 * 3] = temp;
}
	BYTE 배열을 4*4의 행렬모양이라 생각할 때,
	오른쪽으로 한 칸 옮긴다는 것은 index를 4만큼 더해준다는 것을 의미한다.(byte 배열을 행렬에 담을 때, 열방향 우선으로 채워주게된다) 이를 이용하여 shiftLeft 함수를 구현하였다.

BYTE* mixColumns(BYTE *block, int mode){ BYTE mixMat[16] = {2,3,1,1,1,2,3,1,1,1,2,3,3,1,1,2}; /* a(x) = {02} + {01}x + {01}x2 + {03}x3 2 3 1 1 1 2 3 1 1 1 2 3 3 1 1 2 */ BYTE invMixMat[16] = {14,11,13,9,9,14,11,13,13,9,14,11,11,13,9,14}; /* a(x) = {0e} + {09}x + {0d}x2 + {0b}x3 14 11 13 9 9 14 11 13 13 9 14 11	mixColunm 시에 갈루아필드(GF(2^n))의 기약다항식에 대한 곱셈연산이 시행되는데, 이 부분에 대한 연산을 행렬곱으로써 표현하여 해결한다. 번거로운 Code를 줄이기 위해, mixMat 배열과 복호화시 사용될 invmixMat 배열을 만들어 행렬 곱셈 함수로 해결하였다. tempMat= prodMat(invMixMat, block); >> 두 BYTE 4*4 행렬간의 prodMat() 함수 구현이 필요

11 13 9 14
*/
BYTE* tempMat;
 	
tempMat = (BYTE*)malloc(sizeof(BYTE) * 16);
 	
switch(mode){
 	
case ENC:
 	
tempMat= prodMat(mixMat, block);
for(int i = 0; i< BLOCK_SIZE ; i++){
block[i] = tempMat[i];
}
 	
break
 	
case DEC:
 	
tempMat= prodMat(invMixMat, block);
for(int i = 0; i< BLOCK_SIZE ; i++){
block[i] = tempMat[i];
}
break
 	
default:
fprintf(stderr, "Invalid mode!\n");
exit(1);
}
 	
return block
}	

BYTE* prodMat(BYTE* matA, BYTE* matB) {
BYTE mA[16], mB[16];
for (int i = 0; i < 16; i++) {
mA[i] = matA[i];
mB[i] = matB[i];
}
 	
BYTE result= 0x00;
int i,j,k;
BYTE rMat[16] = { 0, };
for(k = 0; k< col_size ; k++){
for(i = 0 ; i < col_size ; i++){
for(j = 0; j< col_size j++){
	두 행렬을 곱해주는 함수, 각각의 matrix는 16BYTE 배열로 주어지며, 4*4 행렬이라 생각하고 계산한다.
	이 때, BYTE 간의 곱셈은 일반 곱셈연산이 아닌, 갈루아필드(GF) 상에서의 곱셈이므로, 유의해야한다.
	이를 쉽게 계산하기 위해, bitwise 계산을 하는 gfmult()
	함수를 구현한다.

result = result ^ gfmult(mA[i*col_size + j] , mB[col_size*k+j]);
}
rMat[4*k+i] = result;
result = 0x00;
}
}
return rMat;
 	
}	

BYTE gfmult(BYTE a, BYTE b) { // GF(2^8) 곱
 	
BYTE p = 0, i = 0, top = 0;
 	
for (i = 0; i < 8; i++) {
if (b & 1) {//b의마지막bit가1 이라면
p ^= a // p에a를XOR
}
 	
top = a & 0x80; // top = a의최상위bit
a <<= 1;
if (top) a ^= 0x1b;
 	
/*
최상위bit가 1 이면
top을제외하고
modulo다항식0000 0001 0001 1011 과 XOR
*/
 	
b >>= 1;
}
 	
return (BYTE)p;
}
	GF(2^8)에서의 BYTE 기약다항식 곱셈의 modulo 결과를 계산해준다. 이 때, 실제 곱셈 및 모듈러 연산을 하지 않고, bitwise 연산(shift << 와 XOR ^)을 통해서 해결한다.
	기본적인 알고리즘은 다음과 같다.
	만약 이전 결과의 최상위 비트가 0이면 단지 이전 결과를 왼쪽으로 한 비트 이동한다.
	만약 이전 결과의 최상위 비트가 1이면 왼쪽으로 한 비트 이동하고, 최상위 비트를 제외하고 모듈로 다항식과 XOR한다.

나)RSA

uint mod(uint dividend, uint divisor)
{
uint result =
dividend - mul(divisor, divide(dividend, divisor));
if (result < 0) result += divisor
	RSA구현 중 도구로 사용될 기본적인 modulo 구현 부분이다.
	나눗셈의 몫과 제수를 사용하여 나머지를 계산해준다.

return result; }	나머지가 음수로 나오는 경우, 제수의 값을 더해줘서 정수론의 $\mathrm{mod} n$ 계산결과와 일치시킨다.

uint ModAdd(uint a, uint b, byte op, uint n) {
uint result = 0;
if (op == '+') {
result = a + b
}
else if (op == '-') {
result = a - b
}
result = mod(result, n);
 	
return result;
}
	$\mathrm{mod} n$에서의 덧셈 및 뺄셈을 구현하엿다.

uint ModMul(uint x, uint y, uint n) {
uint result = 0;
result = mod((x * y) , n) ;
if (result < 0) result = result + n
 	
return result;
}
	$\mathrm{mod} n$ 에서의 곱셈을 구현하엿다.

uint divide(uint dividend, uint divisor)
{
uint i = 0, sign = 0, div = 0;
sign = 0;
if (dividend < 0)
{
dividend = ~dividend + 1;
sign++;
}
if (divisor < 0)
{
divisor = ~divisor + 1;
sign++;
}
if (dividend < divisor)
{
div = 0;
}
else
{
for (i = 0; i<32; i++)
{
if (dividend < (divisor << i))
{
if (i > 0) i--;
break
	나눗셈계산함수 , 연산속도를 증가시키기 위해 bitwise 연산을 사용하였다.

}
else if (dividend == (divisor << i))
{
break
}
}
div = 0;
for (; i >= 0; i--)
{
if (dividend < divisor)
break
 	
if (dividend >= (divisor << i))
{
dividend -= (divisor << i);
div += (1 << i);
}
}
}
if (sign & 0x01)
{
div = ~div + 1;
}
return div;
}	

uint ModInv(uint a, uint m) {
uint x = a
uint y = m
uint result;
uint t1 = 1, t2 = 0;
uint p1 = 0, p2 = 1;
uint q;
uint r;
uint temp1, temp2;
 	
while (y != 1) {
q = divide(y , x);// y = x * q + r
	$\mathrm{mod}\, n$에서 역원값 a^{-1} 을 구하는 함수이다. $a \times a^{-1} \quad 1(\mathrm{mod} n)$ 을 만족한다.
	
	정수론의 Extended Euclide Algorithm(확장 유클리드 알고리즘)을 이용하여 구현하엿다. $(ax + by = 1(\mathrm{mod} n))$

r = (t1 - q* p1) * m + (t2 - q * p2) * a // t1 * a + t2* b =1 이면, gcd(a,b) = 1 이고, t2 = a^(-1) mod b 임을이용한다.
temp1 = p1;
temp2 = p2;
p1 = t1 - q * p1;
p2 = t2 - q * p2;
t1 = temp1;
t2 = temp2;
 	
y = x;
x = r;
}
result = mod(t2 , m);
if (result < 0) result = result + m
 	
return result;
 	
}	

uint GCD(uint a, uint b) {
  uint prev_a;
 	
  while(b != 0) {
  printf("GCD(%u, %u)\n", a, b);
  prev_a = a
  a = b
  while(prev_a >= b) prev_a -= b
  b = prev_a;
  }
  printf("GCD(%u, %u)\n\n", a, b);
  return a
}
	최대공약수를 구하는 함수인 gcd 함수이다.

uint sqMult(uint base, uint exp , uint n) {
if (exp == 1)
return base
else if (exp == 0) return 0;
else if (exp % 2 == 0) {
return ModMul(sqMult(base, exp / 2, n) , sqMult(base, exp / 2, n) , n);
}
else if (exp % 2 == 1)
return ModMul(base, ModMul(sqMult(base, exp / 2, n) , sqMult(base, exp / 2, n), n), n) ;
 	
}
	$\mathrm{mod} n$ 에서의 거듭제곱값(a^b , a = base, b = exp)을 구하기 위해 구현하였다.
	단순히 a를 b번 제곱하는 것은 연산시간이 오래걸리므로, sqare Multiplication(제곱 연산을 사용해 횟수를 줄임) 알고리즘을 이용하였다.
	  이 때 multiplication 은 위에서 구현된 Modular multiplication을 사용한다.

void InitWELLRNG512a(uint *init) {
int j;
state_i = 0;
for (j = 0; j < R j++) STATE[j] = init[j];
}
	난수생성을 위한 초기화 함수이다.
	seed = time(NULL)로 설정된 seed 값을 넣어 초기화한다.

double WELLRNG512a(void) {
z0= VRm1
z1= MAT0NEG(-16, V0) ^ MAT0NEG(-15, VM1);
z2= MAT0POS(11, VM2);
newV1 = z1 ^ z2;
newV0 = MAT0NEG(-2, z0) ^ MAT0NEG(-18, z1) ^ MAT3NEG(-28, z2) ^ MAT4NEG(-5, 0xda442d24U, newV1);
state_i = (state_i + 15) & 0x0000000fU;
return ((double) STATE[state_i]) * FACT
}
	두 개의 임의의 소수를 만들기 위해 난수를 생성하기 위한 함수이다. rand 함수 등의 난수 생성함수 보다 예측 불가능한 난수 생성함수를 만들기 위해 사용한다.

bool IsPrime(uint testNum, uint repeat) {
uint n = testNum
uint m = n - 1;
uint k = 0;
uint a,b;
srand(time(NULL));
while (m %2 == 0) { // 2^k * m = n-1
m = divide(m , 2);
k++;
}
 	
for (int i = 0; i < repeat i++) {
a = mod(rand() , n) + 1; // 1~n 사이의a를무작위로선택
if (GCD(a, n) != 1) {
return FALSE
}
b = sqMult(a, m, n); // b = a^m mod n
if (b == 1 || b == n - 1) { // 강한유사소수
continue
}
else {
for(int i = 0 ; i<k-1; i++)
if (mod((b*b) , n) == n - 1) { // 강한유사소수
continue
}
if (mod((b*b) , n) != n - 1) { // 합성수강한증거 FALSE 리턴
return FALSE
}
}
	밀러-라빈 소수 판별법 알고리즘을 이용하여 위에서 생성한 난수가 높은 확률로 소수임을 확인해주는 함수이다.
	
	rand함수를 통해 무작위로 뽑은 a값과 반복적으로 비교하여 testNum 이 강한유사소수인지, 합성수의 강한 증거가 있는지 확인한다.
	이 때 강한 유사소수로 판별되어 루프를 통과하면, 1-(1/4)^repeat 의 확률로 소수이다.

} return TRUE // 루프통과 }	

void miniRSAKeygen(uint *p, uint *q, uint *e, uint *d, uint *n) {
 	
srand(time(NULL));
uint r; //r 값생성
 	
uint phi_n;
*n = 0;
 	
while (*n < sqMult(2, 31, UINT_MAX))
// 2^31 <= n < 2^32 인n 을찾을때까지반복
{
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a());
// 0 ~ 2^31 보다 작은r값을 임의로 생성하여 p와q에할당
while (!IsPrime(r, 100)) {
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a());
}
*p = r;
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a());
while (!IsPrime(r, 100)) {
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a());
}
*q = r; // prime q 생성
*n = *p * *q
}
 	
phi_n = ModMul(*p - 1, *q - 1, *n);
 	
*e = (uint)(WELLRNG512a() * phi_n)+ 2 ;// 조건을만족하는1< e < phi_n 의e 생성
while (gcd(*e, n!=1)) {
*e = (uint)(WELLRNG512a() * phi_n) + 2;
}
*d = ModInv(*e, phi_n);
 	
}
	난수 생성함수 WELLRNG512a를 이용하여 난수를 생성한다. 이 때, WELLRNG512a 의 output값에 수를 곱하거나 더해주어, 생성되는 난수의 범위를 제한해주도록 구현하였다.
	
	생성된 난수 p,q 를 ISPrime 함수를 이용하여 소수인지 확인한다. p,q가 소수가 아니라면 소수일 때 까지 반복하여 소수 p,q 쌍을 얻어낸다.
	
	Euler Function $\phi(n)$ 값을 이용하여,
	1< e < $\phi(n)$ 의 e값을 임의로 생성한 뒤,
	
	ModInv() 함수를 이용하여 ed=1 ($\mathrm{mod} n$) 인 e의 역원 d를 구해준다.
	
	Public Key = { e, n }
	Private Key = { d, n }
	
	의 2개의 키 쌍을 얻어낸다.

uint miniRSA(uint data, uint key, uint n) {
uint result;
result = sqMult(data, key, n);
return result;
}
	생성된 key를 이용하여 암호화, 복호화를 진행하는 함수이다.

4. Simulation 및 결론

Visual Studio 2017을 이용하여 AES 암호화 알고리즘을 프로그래밍 한 후 simulation 하였다. 다음은 암호화 할 16BYTE의 평문 1 Block의 값이다.

그림 4.1 시작 값 - 평문 16BYTE , 1Block (iv 적용 전)

이 평문 1Block에 iv를 적용한다. 다음은 iv를 적용한 후의 Block의 값이다.

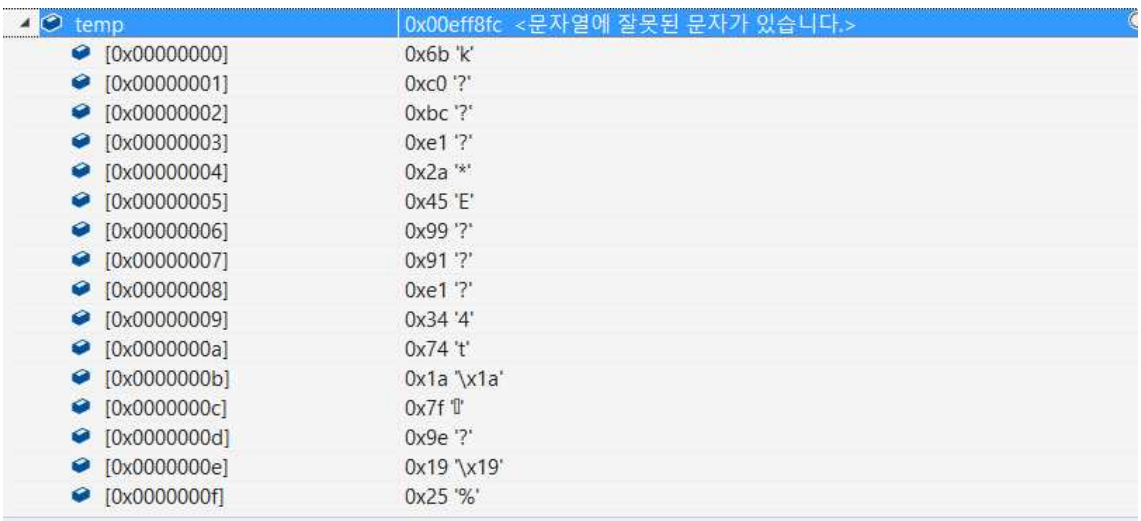

그림 4.2 temp 값(iv 적용 후)

다음은 여기에 addRoundKey() - subBytes() - ShiftRow() - mixColumns() 순서로 암호화 각 단계들을 순서대로 적용한 값들이다.

그림 4.3 addRoundKey() 적용 후

그림 4.4 subBytes() 적용 후

그림 4.5 ShiftRow() 적용 후

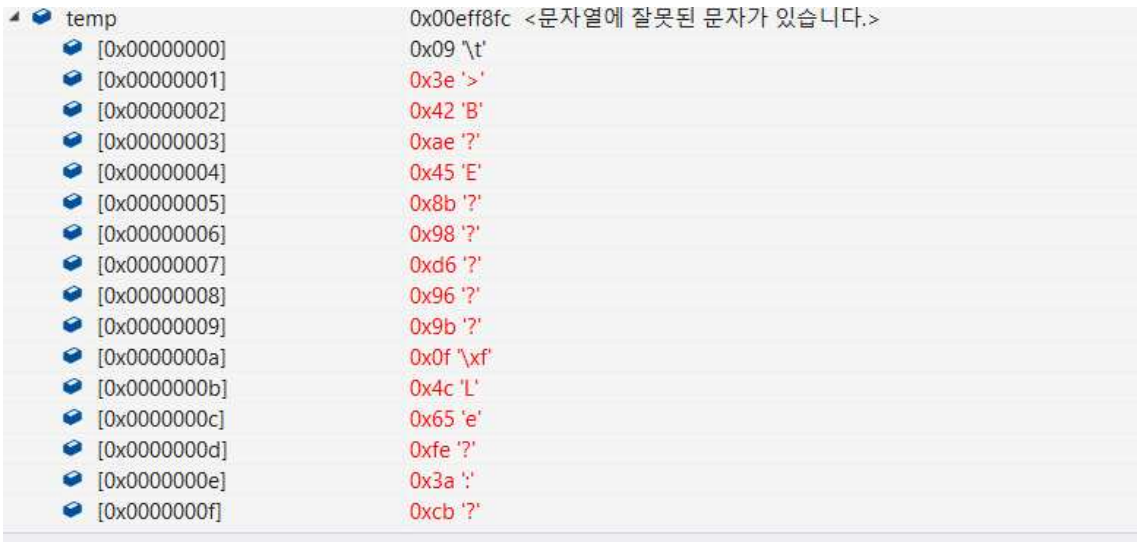

그림 4.6 mixColumns() 적용 후

각 라운드 단계들을 10라운드까지 계속 수행한 최종 암호화 결과 값은 다음과 같다.

그림 4.7 암호화 최종 결과화면

Ⅳ.참고문헌

[1] United States National Institute of Standards and Technology (NIST). *Announcing the advanced encryption standrad(AES)*, 2001

[2] Rivest, R. L.; Shamir, A.; Adleman, L, *A Method for Obtaining Digital*

Signatures and Public-key Cryptosystems, 1978

[3] Daemen, Joan; Rijmen, Vincent, ***AES Proposal: Rijndael,*** 2003

Ⅴ.부록

AES128.h

```
// 암호화모드
#define ENC 1
// 복호화모드
#define DEC 0

typedef unsigned char BYTE

// 128비트AES 암복호화인터페이스
void AES128(BYTE *input, BYTE *output, BYTE *key, int mode);
```

AES128.c

```
#include <stdio.h>
#include <stdlib.h>
#include "AES128.h"


#define KEY_SIZE 16
#define ROUNDKEY_SIZE 176
#define BLOCK_SIZE 16
#define col_size 4


BYTE R[] = { 0x02, 0x00, 0x00, 0x00 };

static BYTE sbox[256] = {
// 0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, // 0
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, // 1
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, // 2
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, // 3
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, // 4
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, // 5
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, // 6
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, // 7
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, // 8
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, // 9
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, // a
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, // b
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, // c
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, // d
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, // e
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };// f
```

```
static BYTE inv_s_box[256] = {
// 0     1     2     3     4     5     6     7     8     9     a     b     c     d     e     f
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb, // 0
0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, // 1
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e, // 2
0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25, // 3
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92, // 4
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84, // 5
0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06, // 6
0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b, // 7
0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73, // 8
0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e, // 9
0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b, // a
0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4, // b
0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f, // c
0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, // d
0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61, // e
0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d };// f



BYTE aes_xtime(BYTE x)
{
        return ((x << 1) ^ (((x >> 7) & 1) * 0x1b));
}
BYTE aes_xtimes(BYTE x, int ts)
{
        while (ts-- > 0) {
                x = aes_xtime(x);
        }

        return x;
}

BYTE gfmult(BYTE a, BYTE b) { // GF(2^8) 곱

        BYTE p = 0, i = 0, top = 0;

        for (i = 0; i < 8; i++) {
                if (b & 1) {//b의 마지막 bit가 1 이라면
                        p ^= a; // p에 a를 XOR
                        }

        top = a & 0x80; // a의 최상위 bit
        a <<= 1;
        if (top) a ^= 0x1b; //최상위 bit 가 1 이면 top을 제외하고 modulo다항식 0000 0001 0001 1011
        과 XOR
                b >>= 1;
        }

        return (BYTE)p;
}

        void coef_mult(BYTE *a, BYTE *b, BYTE *d) {

        d[0] = gfmult(a[0], b[0]) ^ gfmult(a[3], b[1]) ^ gfmult(a[2], b[2]) ^ gfmult(a[1], b[3]);
        d[1] = gfmult(a[1], b[0]) ^ gfmult(a[0], b[1]) ^ gfmult(a[3], b[2]) ^ gfmult(a[2], b[3]);
        d[2] = gfmult(a[2], b[0]) ^ gfmult(a[1], b[1]) ^ gfmult(a[0], b[2]) ^ gfmult(a[3], b[3]);
        d[3] = gfmult(a[3], b[0]) ^ gfmult(a[2], b[1]) ^ gfmult(a[1], b[2]) ^ gfmult(a[0], b[3]);
        }
```

```
        BYTE *Rcon(BYTE i) {

                if (i == 1) {
                        R[0] = 0x01; // x^(1-1) = x^0 = 1
                }
                else if (i > 1) {
                        R[0] = 0x02;
                        i--;
                        while (i - 1 > 0) {
                                R[0] = gfmult(R[0], 0x02);
                                i--;
                        }
                }

                return R;
        }

/* 기타 필요한 함수 추가 선언 및 정의 */
void addByte(BYTE* a, BYTE* b, BYTE* d) {

        d[0] = a[0]^b[0];
        d[1] = a[1]^b[1];
        d[2] = a[2]^b[2];
        d[3] = a[3]^b[3];
}




/*  <키스케줄링 함수>
 *
 *  key          키스케줄링을 수행할 16바이트 키
 *  roundKey     키스케줄링의 결과인 176바이트 라운드키가 담길 공간
 */

void rotate(BYTE *r) {
    BYTE temp;
        int i;
    temp = r[0];
    for(i=0 ; i<3 ; i++ ){
        r[i] = r[i+1];
    }
    r[3] = temp;
}

void sub(BYTE *w) {
        int i;
    for( i = 0; i< 4; i++) {
        w[i] = sbox[w[i]];

    }
}

void expandKey(BYTE *key, BYTE *roundKey){

        BYTE k[KEY_SIZE];
        for (int i = 0; i < KEY_SIZE; i++) {
                k[i] = key[i];
        }

    BYTE temp[KEY_SIZE/4];
        int i;
```

```
    for( i = 0 ; i < KEY_SIZE ; i++)
    {
        roundKey[i] = k[i];
    }

    for(int i = KEY_SIZE/4 ; i < ROUNDKEY_SIZE/4; i++){
        temp[0] = roundKey[4*i -4];
        temp[1] = roundKey[4*i -3];
        temp[2] = roundKey[4*i -2];
        temp[3] = roundKey[4*i -1];

        if(i%4 == 0){
        rotate(temp);
        sub(temp);

        addByte(temp,Rcon(i/4),temp);
        }

        roundKey[4*i+0] =roundKey[4*(i-4)+0]^temp[0];
                    roundKey[4*i+1] = roundKey[4*(i-4)+1]^temp[1];
                    roundKey[4*i+2] = roundKey[4*(i-4)+2]^temp[2];
                    roundKey[4*i+3] = roundKey[4*(i-4)+3]^temp[3];
    }
}


/*  <SubBytes 함수>
 *
 *  block    SubBytes 수행할 16바이트 블록. 수행 결과는 해당 배열에 바로 반영
 *  mode     SubBytes 수행 모드
 */
 BYTE* subBytes(BYTE *block, int mode){



    switch(mode){

        case ENC:

            /* 추가 구현 */
            for(int i = 0; i<BLOCK_SIZE ; i++){
                block[i] = sbox[block[i]];
                    }


            break;

        case DEC:

            /* 추가 구현 */
            for(int i = 0; i<BLOCK_SIZE ; i++){
                block[i] = inv_s_box[block[i]];
                    }

            break;

        default:
            fprintf(stderr, "Invalid mode!\n");
            exit(1);
    }

    return block;
}
```

```
/*  <ShiftRows 함수>
 *
 *  block    ShiftRows 수행할 16바이트 블록. 수행 결과는 해당 배열에 바로 반영
 *  mode     ShiftRows 수행 모드
 */

 void shiftLeft(BYTE* block) { // 왼쪽으로 1칸이동
          BYTE temp;
          temp = block[0];
          for (BYTE i = 0; i < 3; i++) {

                    block[4 * i] = block[4 * (i + 1)];
          }
          block[4 * 3] = temp;
 }


 BYTE* shiftRows(BYTE *block, int mode) {

          switch (mode) {

          case ENC:


                    /*
                    행 단위  shift 연산
                    */

                    for (int i = 0; i < BLOCK_SIZE / 4; i++) {
                              for (int j = 0; j < i; j++) {
                                        shiftLeft(block + i); //i번째 행을 i번 왼쪽으로 이동
                              }


                    }
                    break;

          case DEC:

                    for (int i = BLOCK_SIZE / 4; i > 0; i--) {
                              for (int j = 0; j < i; j++) {
                                        shiftLeft(block + 4 - i); //i번째 행을 4-i번 왼쪽으로 이동
                              }


                    }

                    break;

          default:
                    fprintf(stderr, "Invalid mode!\n");
                    exit(1);
          }

          return block;
 }


/*  <MixColumns 함수>
 *
 *  block    MixColumns을 수행할 16바이트 블록. 수행 결과는 해당 배열에 바로 반영
```

```
 *  mode     MixColumns의 수행 모드
 */

void mcol(BYTE* col) {
	BYTE t[4];
	for (int i = 0; i < col_size; i++) { //t is temp, temp에 copy
		t[i] = col[i];
	}
	col[0] = gfmult(0x02, t[0]) ^ gfmult(0x03, t[1]) ^ t[2] ^ t[3];
	col[1] = t[0] ^ gfmult(0x02, t[1]) ^ gfmult(0x03, t[2]) ^ t[3];
	col[2] = t[0] ^ t[1] ^ gfmult(t[2], 0x02) ^ gfmult(t[3], 0x03);
	col[3] = gfmult(0x03, t[0]) ^ t[1] ^ t[2] ^ gfmult(0x02, t[3]);

}

BYTE* prodMat(BYTE* matA, BYTE* matB) {
	BYTE mA[16], mB[16];
	for (int i = 0; i < 16; i++) {
		mA[i] = matA[i];
		mB[i] = matB[i];
	}

	BYTE result = 0x00;
	int i, j, k;
	BYTE rMat[16] = { 0, };
	for (k = 0; k< col_size; k++) {
		for (i = 0; i < col_size; i++) {
			for (j = 0; j< col_size; j++) {
				result = result ^ gfmult(mA[i*col_size + j], mB[col_size*k + j]);
			}
			rMat[4 * k + i] = result;
			result = 0x00;
		}
	}
	return rMat;


}


BYTE* mixColumns(BYTE *block, int mode){

    BYTE mixMat[16] = {2,3,1,1,1,2,3,1,1,1,2,3,3,1,1,2};
	/* a(x) = {02} + {01}x + {01}x2 + {03}x3
	2 3 1 1
	1 2 3 1
	1 1 2 3
	3 1 1 2
	*/
    BYTE invMixMat[16] = {14,11,13,9,9,14,11,13,13,9,14,11,11,13,9,14};
	/* a(x) = {0e} + {09}x + {0d}x2 + {0b}x3
	14 11 13 9
	9 14 11 13
	13 9 14 11
	11 13 9 14
	*/
	BYTE* tempMat;

	tempMat = (BYTE*)malloc(sizeof(BYTE) * 16);



    switch(mode){
```

```
        case ENC:


                            tempMat= prodMat(mixMat, block);
            for(int i = 0; i< BLOCK_SIZE ; i++){
                     block[i] = tempMat[i];
                    }



            break;

        case DEC:

            /* 추가 구현 */
                            tempMat= prodMat(invMixMat, block);
            for(int i = 0; i< BLOCK_SIZE ; i++){
                     block[i] = tempMat[i];
                    }
            break;

        default:
            fprintf(stderr, "Invalid mode!\n");
            exit(1);
    }

    return block;
}


/*  <AddRoundKey 함수>
 *
 *  block    AddRoundKey를 수행할 16바이트 블록. 수행 결과는 해당 배열에 반영
 *  rKey     AddRoundKey를 수행할 16바이트 라운드키
 */
BYTE* addRoundKey(BYTE *block, BYTE *rKey){



        for(int i = 0; i <BLOCK_SIZE ; i++){
            block[i] = block[i] ^ rKey[i] ;
    }


    return block;
}




/*  <128비트 AES 암복호화 함수>
 *
 *  mode가 ENC일 경우 평문을 암호화하고, DEC일 경우 암호문을 복호화하는 함수
 *
 *  [ENC 모드]
 *  input    평문 바이트 배열
 *  output   결과(암호문)이 담길 바이트 배열. 호출하는 사용자가 사전에 메모리를 할당하여 파라미터로 넘어옴
 *  key      128비트 암호키 (16바이트)
 *
 *  [DEC 모드]
 *  input    암호문 바이트 배열
 *  output   결과(평문)가 담길 바이트 배열. 호출하는 사용자가 사전에 메모리를 할당하여 파라미터로 넘어옴
```

```
 *  key      128비트 암호키 (16바이트)
 */
void AES128(BYTE *input, BYTE *output, BYTE *key, int mode){

          BYTE temp[16];
          for (int i = 0; i < BLOCK_SIZE; i++) {
                    temp[i] = input[i];
          }
          BYTE *roundKey;
          roundKey = (BYTE*)malloc(ROUNDKEY_SIZE);
          memcpy(output, input, BLOCK_SIZE);

    if(mode == ENC){

                    expandKey(key, roundKey);
        addRoundKey(temp,&roundKey[0]);


        for(int i =1 ; i<10 ;i++){ //9round 반복
          subBytes(temp,ENC);
                    shiftRows(temp,ENC);
                    mixColumns(temp, ENC);
                    addRoundKey(temp, &roundKey[i*KEY_SIZE]);
        }

                    subBytes(temp, ENC);
                    shiftRows(temp,ENC);
                    addRoundKey(temp, &roundKey[10* KEY_SIZE]);



                    for (int i = 0; i < BLOCK_SIZE; i++) {
                              output[i] = temp[i];
                    }

                    free(roundKey);


    }else if(mode == DEC){

        expandKey(key,roundKey);
        addRoundKey(temp,&roundKey[10*KEY_SIZE]);

        for(int i = 9 ; i>0 ; i--){
            shiftRows(temp,DEC);
            subBytes(temp,DEC);
            addRoundKey(temp, &roundKey[i*KEY_SIZE]);
                              mixColumns(temp, DEC);

        }
                    shiftRows(temp, DEC);
                    subBytes(temp, DEC);
                    addRoundKey(temp, &roundKey[0]);

                    for (int i = 0; i < BLOCK_SIZE; i++) {
                              output[i] = temp[i];
                    }



    }else{
        fprintf(stderr, "Invalid mode!\n");
        exit(1);
```

```
    }
}
```

test_AES128.c

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "AES128.h"

typedef unsigned char BYTE

BYTE iv[]  = {
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f
};

BYTE key[] = {
    0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c
};

BYTE plain[] = {
    0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
    0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
    0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
    0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10
};

BYTE cipher[] = {
    0x76, 0x49, 0xab, 0xac, 0x81, 0x19, 0xb2, 0x46, 0xce, 0xe9, 0x8e, 0x9b, 0x12, 0xe9, 0x19, 0x7d,
    0x50, 0x86, 0xcb, 0x9b, 0x50, 0x72, 0x19, 0xee, 0x95, 0xdb, 0x11, 0x3a, 0x91, 0x76, 0x78, 0xb2,
    0x73, 0xbe, 0xd6, 0xb8, 0xe3, 0xc1, 0x74, 0x3b, 0x71, 0x16, 0xe6, 0x9e, 0x22, 0x22, 0x95, 0x16,
    0x3f, 0xf1, 0xca, 0xa1, 0x68, 0x1f, 0xac, 0x09, 0x12, 0x0e, 0xca, 0x30, 0x75, 0x86, 0xe1, 0xa7
};

void printResult(char *keyword, uint8_t *data) {
    int i = 0;
    printf(" - %s : ", keyword);
    for(i = 0; i < 64; i++) {
        if(i % 16 == 0) printf("\n");
        printf("%02x ", data[i]);
    }
    printf("\n");
}

int main(){
    int i, j;
    BYTE result[64], tmpBlock[16];

    // 암호화테스트
    memcpy(result, plain, sizeof(BYTE)*64);

    printResult("Plain Text", plain);


    for(i = 0; i < 4; i++){
for (j = 0; j < 16; j++) {



if (i == 0)
result[j] = iv[j] ^ result[j];
```

```
else
result[i * 16 + j] = result[(i - 1) * 16 + j] ^ plain[i * 16 + j];
}
        AES128(result + i * 16, tmpBlock, key, ENC);
        memcpy(result + i * 16, tmpBlock, sizeof(BYTE) * 16);
    }

    printResult("Encrypted Plain Text", result);

    printf("==================================\n");
    printf("AES Encryption: %s\n", 0 == strncmp((char*) cipher, (char*) result, 64) ? "SUCCESS!" :
"FAILURE!");
    printf("==================================\n");

    // 복호화테스트
    memcpy(result, cipher, sizeof(BYTE)*64);

    printResult("Cipher Text", cipher);

    for(i = 0; i < 4; i++){
        AES128(result + i * 16, tmpBlock, key, DEC);
        for(j = 0; j < 16; j++)
            if(i == 0)
                result[j] = iv[j] ^ tmpBlock[j];
            else
                result[i * 16 + j] = cipher[(i - 1) * 16 + j] ^ tmpBlock[j];
    }

    printResult("Decrypted Cipher Text", result);

    printf("==================================\n");
    printf("AES Decryption: %s\n", 0 == strncmp((char*) result, (char*) plain, 64) ? "SUCCESS!" :
"FAILURE!");
    printf("==================================\n");

    return 0;
}
```

miniRSA.h

```
// WELL Random number generator 관련 매크로
#define W32
#define R16
#define P0
#define M113
#define M29
#define M35

#define MAT0POS(t,v)(v^(v>>t))
#define MAT0NEG(t,v)(v^(v<<(-(t))))

#define MAT3NEG(t,v)(v<<(-(t)))
#define MAT4NEG(t,b,v)(v^((v<<(-(t)))&b))

#define V0STATE[state_i]
#define VM1STATE[(state_i+M1) & 0x0000000fU]
#define VM2STATE[(state_i+M2) & 0x0000000fU]
#define VM3STATE[(state_i+M3) & 0x0000000fU]
#define VRm1STATE[(state_i+15) & 0x0000000fU]
#define VRm2STATE[(state_i+14) & 0x0000000fU]
#define newV0STATE[(state_i+15) & 0x0000000fU]
#define newV1STATE[state_i]
```

```
#define newVRm1STATE[(state_i+14) & 0x0000000fU]

#define FACT2.32830643653869628906e-10

#define RND_MAX0x00ffff
#define RND_MIN0x00b505
#define FALSE0
#define TRUE1

// mini RSA 관련타입
typedef unsigned char bool
typedef unsigned char byte
typedef unsigned int  uint

// WELL Random number generator 관련전역변수
static unsigned int state_i = 0;
static unsigned int STATE[R];
static unsigned int z0, z1, z2;

// WELL Random number generator 관련함수
void InitWELLRNG512a(uint *init);
double WELLRNG512a(void);

// mini RSA 관련인터페이스
uint modMul(uint x, uint y, uint mod);
uint modPow(uint base, uint exp, uint mod);
bool isPrime(uint testNum, uint repeat); // 밀러-라빈소수판별법알고리즘을이용}
uint gcd(uint a, uint b);
uint modInv(uint a, uint m);
void miniRSAKeygen(uint *p, uint *q, uint *e, uint *d, uint *n);
uint miniRSA(uint data, uint key, uint n);

// 난수생성을위한초기화함수
void InitWELLRNG512a(uint *init) {
int j;
state_i = 0;
for (j = 0; j < R j++) STATE[j] = init[j];
}

// 난수생성함수
double WELLRNG512a(void) {
z0= VRm1
z1= MAT0NEG(-16, V0) ^ MAT0NEG(-15, VM1);
z2= MAT0POS(11, VM2);
newV1 = z1 ^ z2;
newV0 = MAT0NEG(-2, z0) ^ MAT0NEG(-18, z1) ^ MAT3NEG(-28, z2) ^ MAT4NEG(-5, 0xda442d24U,
newV1);
state_i = (state_i + 15) & 0x0000000fU;
return ((double) STATE[state_i]) * FACT
}
```

miniRSA.c

```
/*
 * @file     rsa.c
 * @author   최성인/ 2012042201
 * @date     11/30
```

```
 * @brief    mini RSA implementation code
 * @details 세부설명
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "miniRSA.h"

uint p, q, e, d, n;


/*
 * @brief       모듈러덧셈연산을하는함수.
 * @param       uint a       : 피연산자1.
 * @param       uint b       : 피연산자2.
 * @param       byte op      : +, - 연산자.
 * @param       uint n        : 모듈러값.
 * @return      uint result : 피연산자의덧셈에대한모듈러연산값. (a op b) mod n
 * @todo        모듈러값과오버플로우상황을고려하여작성한다.
 */
uint ModAdd(uint a, uint b, byte op, uint n) {
uint result = 0;
if (op == '+') {
result = a + b
}
else if (op == '-') {
result = a - b
}
result = mod(result, n);

return result;
}
uint divide( uint dividend, uint divisor) {
uint i = 0, sign = 0, div = 0;

sign = 0;
if (dividend < 0)
{
dividend = ~dividend + 1;
sign++;
}
if (divisor < 0)
{
divisor = ~divisor + 1;
sign++;
}
if (dividend < divisor)
{
div = 0;
}
else
{
for (i = 0; i<32; i++)
{
if (dividend < (divisor << i))
{
if (i > 0) i--;
break
}
else if (dividend == (divisor << i))
{
```

```
break
}
}

div = 0;
for (; i >= 0; i--)
{
if (dividend < divisor)
break

if (dividend >= (divisor << i))
{
dividend -= (divisor << i);
div += (1 << i);
}
}
}
if (sign & 0x01)
{
div = ~div + 1;
}
return div;
}
uint mod(uint dividend, uint divisor)
{
uint result = dividend - mul(divisor, div(dividend, divisor));
if (result < 0) result += divisor
return result;
}


/*
 * @brief        모듈러곱셈연산을하는함수.
 * @param        uint x         : 피연산자1.
 * @param        uint y         : 피연산자2.
 * @param        uint n         : 모듈러값.
 * @return       uint result  : 피연산자의곱셈에대한모듈러연산값. (a x b) mod n
 * @todo         모듈러값과오버플로우상황을고려하여작성한다.
 */
uint ModMul(uint x, uint y, uint n) {
uint result = 0;
result = mod( (x * y) , n) ;
if (result < 0) result = result + n

return result;
}

/*
 * @brief        모듈러거듭제곱연산을하는함수.
 * @param        uint base    : 피연산자1.
 * @param        uint exp     : 피연산자2.
 * @param        uint n       : 모듈러값.
 * @return       uint result : 피연산자의연산에대한모듈러연산값. (base ^ exp) mod n
 * @todo         모듈러값과오버플로우상황을고려하여작성한다.
                'square and multiply' 알고리즘을사용하여작성한다.
 */
uint sqMult(uint base, uint exp , uint n) {
if (exp == 1)
return base
else if (exp == 0) return 0;
else if (exp % 2 == 0) {
return  ModMul( sqMult(base, exp / 2, n) , sqMult(base, exp / 2, n) , n );
}
```

```
else if (exp % 2 == 1)
return  ModMul( base, ModMul( sqMult(base, exp / 2, n) , sqMult(base, exp / 2, n), n ), n) ;



}

uint ModPow(uint base, uint exp, uint n) {
if (exp == 1) {
return mod(base,n);
}
else if (exp == 0) {
return 1;
}

//squre multiplication - recursive call을사용한다. 2로나누어지면결과값을제곱, 홀수이면1을빼서제곱* base

if (exp % 2 == 0) {
return (ModPow(base, exp / 2, n) * ModPow(base, exp / 2, n)) % n
}
else if (exp % 2 == 1) {
return (base * ModPow(base, (exp - 1) / 2, n) * ModPow(base, (exp - 1) / 2, n)) % n
}
}

/*
 * @brief        입력된수가소수인지입력된횟수만큼반복하여검증하는함수.
 * @param        uint testNum    : 임의생성된홀수.
 * @param        uint repeat     : 판단함수의반복횟수.
 * @return       uint result     : 판단결과에따른TRUE, FALSE 값.
 * @todo         Miller-Rabin 소수판별법과같은확률적인방법을사용하여,
                이론적으로4N(99.99%) 이상되는값을선택하도록한다.
 */
bool IsPrime(uint testNum, uint repeat) { // 밀러-라빈소수판별법알고리즘을이용

uint n = testNum
uint m = n - 1;
uint k = 0;
uint a,b;

while (m %2 == 0) { // 2^k * m  = n-1
m = divide(m , 2);
k++;
}

for (int i = 0; i < repeat i++) {
a = mod( rand() , n) + 1;  // 1~n 사이의a를무작위로선택
if (GCD(a, n) != 1) {
return FALSE
}
b = sqMult(a, m, n); // b = a^m mod n
if (b == 1 || b == n - 1) {  // 강한유사소수
continue
}
else {
for(int i = 0 ; i<k-1; i++)
if (mod((b*b) , n) == n - 1) { // 강한유사소수
continue
}
if (mod((b*b) , n) != n - 1) { // 합성수강한증거FALSE 리턴
return FALSE
}
}
```

```
}

return TRUE // 루프통과, 1-(1/4)^repeat 의확률로소수
}

/*
 * @brief        모듈러역값을계산하는함수.
 * @param         uint a       : 피연산자1.
 * @param         uint m       : 모듈러값.
 * @return       uint result : 피연산자의모듈러역수값.
 * @todo         확장유클리드알고리즘을사용하여작성하도록한다.
 */
uint ModInv(uint a, uint m) {
uint x = a
uint y = m
uint result;
uint t1 = 1, t2 = 0;
uint p1 = 0, p2 = 1;
uint q;
uint r;
uint temp1, temp2;

while (y != 1) {
q = divide(y , x);// y = x * q + r
r = (t1 - q* p1) * m + (t2 - q * p2) * a  // t1 * a + t2* b =1 이면, gcd(a,b) = 1 이고, t2 = a^(-1) mod
b 임을이용한다.
temp1 = p1;
temp2 = p2;
p1 = t1 - q * p1;
p2 = t2 - q * p2;
t1 = temp1;
t2 = temp2;

y = x;
x = r;
}
result = mod(t2 , m);
if (result < 0) result = result + m

return result;

}

/*
 * @brief      RSA 키를생성하는함수.
 * @param       uint *p    : 소수p.
 * @param       uint *q    : 소수q.
 * @param       uint *e    : 공개키값.
 * @param       uint *d    : 개인키값.
 * @param       uint *n    : 모듈러n 값.
 * @return     void
 * @todo        과제안내문서의제한사항을참고하여작성한다.
 */
void miniRSAKeygen(uint *p, uint *q, uint *e, uint *d, uint *n) {

srand(time(NULL));
uint r;  //r 값생성

uint phi_n;
*n = 0;
```

```
while (*n < sqMult(2, 31, UINT_MAX)) // 2^31 <= n < 2^32 인n 을찾을때까지반복
{
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a()); //  0 ~ 2^31 보다작은r값을임의로생성하여p와q에할당
while (!IsPrime(r, 100)) {
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a());
}
*p = r;
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a());
while (!IsPrime(r, 100)) {
r = (uint)(divide(UINT_MAX, 2)* WELLRNG512a());
}
*q = r; // prime q 생성
*n = *p * *q
}

phi_n = ModMul(*p - 1, *q - 1, *n);

*e = (uint)( WELLRNG512a() * phi_n )+ 2 ;// 조건을만족하는1< e < phi_n 의e 생성
while (gcd(*e, n!=1)) {
*e = (uint)(WELLRNG512a() * phi_n) + 2;
}
*d = ModInv(*e, phi_n);

}

/*
 * @brief       RSA 암복호화를진행하는함수.
 * @param       uint data    : 키값.
 * @param       uint key     : 키값.
 * @param       uint n       : 모듈러n 값.
 * @return      uint result : 암복호화에결과값
 * @todo        과제안내문서의제한사항을참고하여작성한다.
 */
uint miniRSA(uint data, uint key, uint n) {
uint result;
result = sqMult(data, key, n);
return result;
}

uint GCD(uint a, uint b) {
    uint prev_a;

    while(b != 0) {
        printf("GCD(%u, %u)\n", a, b);
        prev_a = a
        a = b
        while(prev_a >= b) prev_a -= b
        b = prev_a;
    }
    printf("GCD(%u, %u)\n\n", a, b);
    return a
}

int main(int argc, char* argv[]) {
    byte plain_text[4] = {0x12, 0x34, 0x56, 0x78};
    uint plain_data, encrpyted_data, decrpyted_data;
    uint seed = time(NULL);

    memcpy(&plain_data, plain_text, 4);

    // 난수생성기시드값설정
    seed = time(NULL);
    InitWELLRNG512a(&seed);
```

```
    // RSA 키생성
    miniRSAKeygen(&p, &q, &e, &d, &n);
    printf("0. Key generation is Success!\n ");
    printf("p : %u\n q : %u\n e : %u\n d : %u\n N : %u\n\n", p, q, e, d, n);

    // RSA 암호화테스트
    encrpyted_data = miniRSA(plain_data, e, n);
    printf("1. plain text : %u\n", plain_data);
    printf("2. encrypted plain text : %u\n\n", encrpyted_data);

    // RSA 복호화테스트
    decrpyted_data = miniRSA(encrpyted_data, d, n);
    printf("3. cipher text : %u\n", encrpyted_data);
    printf("4. Decrypted plain text : %u\n\n", decrpyted_data);

    // 결과출력
    printf("RSA Decryption: %s\n", (decrpyted_data == plain_data) ? "SUCCESS!" : "FAILURE!");

    return 0;
}
```