*모듈러 값과 오버플로우 상황을 고려하여 작성한다.

Mod() : 모드연산을 위한 함수

```
uint Mod(uint a, uint mod){
	if(a > mod){   while(a >= mod) a -= mod; result = a;}
	else {result = a;}
	return result;}
```

설명 : a가 mod값 보다 클 경우 mod값보다 작게 하기 위하여 mod값을 적절하게 빼셈한다.

ModAdd() : 모듈러 덧셈 연산을 하는 함수.

```
uint ModAdd(uint a, uint b, uint mod) {
	if( a+b <= a) { //overflow
		result = Mod((a-(mod-b)) ,   mod);   }
	else {
		result = Mod( (a+b), mod);}
	return result;
}
```

설명 : a+b값이 매우 커서 오버플로우가 생겨날 경우 처리를 해야 합니다. 모듈러 연산은 mod값으로 나누기 때문에 나누어지는 숫자인 a+b에 n을 더하거나 뺀다고 해서 나머지가 변하지는 않습니다. 다음과 같이 바꿀 수 있습니다. a+b≡a+b−mod = a + b - n == a - (n - b)입니다.

ModMul() : 모듈러 곱셈 연산을 위한 함수

```
uint ModMul(uint x, uint y, uint mod) {
        uint a = 0 ;
        uint b = Mod(x, mod);
        while (y > 0){
                if(y & 1 == 1){
                        a = ModAdd(a,b,mod);}41
                b =  ModAdd(b,b,mod);
                y = y >> 1; }
        result = Mod(a,mod);
        return result; }
```

설명 : ab mod n 은 다음이 성립합니다.

$$ab \bmod n = \begin{cases} (ak + ak) \bmod n, & b = 2k \\ (a + ak + ak) \bmod n, & b = 2k + 1 \end{cases}$$

곱연산은 Modadd연산으로 치환한 뒤 오버플로우를 해결 할 수 있습니다.

ModPow() : 모듈러 거듭제곱 연산을 하는 함수

```
uint ModPow(uint base, uint exp, uint mod) {
        uint x = 1;
        uint y = base;
        while (exp > 0){
                if (exp & 1 == 1){
                        x = ModMul(x,y,mod);}
                y = ModMul(y,y,mod);
                exp = exp >> 1 ;}
        result = x;
        return result;}
```

설명 : 거듭 제곱의 나머지를 구하는 것도 ModMul연산과 비슷합니다.

a^b mod n 또한 (a * a) mod n 와 같이 성립되기 때문입니다.

IsPrime() : 입력된 값이 소수인지 입력된 횟수만큼 반복하여 검증하는 함수

```
bool IsPrime(uint testNum, uint repeat) {       //Miller-Rabin primality test

        uint d = testNum - 1, s = 1;

        uint k, a, result;

        if(testNum == 1) return FALSE;

        while(((d >>= 1) & 1) ^ 1) s++;          //2^s * d

        while(repeat--){

                result = ModPow((uint)((WELLRNG512a() * (testNum - 2)) + 2), d, testNum);

                if(result == 1 || result == (testNum - 1)) continue;

                for(k = 0; k < s; k++) {

                        result = ModMul(result, result, testNum);

                        if(result == (testNum - 1)) break;

                }

                if(!(k ^ s)) {

                        printf("%u is not Prime.₩n₩n", testNum);

                        return FALSE;

                }

        }

        printf("%u may be Prime.₩n₩n", testNum);

        return TRUE;
```

설명 :

입력: n: 소수인지 검사할 숫자 , k: 소수판별법을 몇회 실행할지 결정하는 인자.
출력: n이 합성수이면 합성수이다, 아니면 아마 소수일 것 같다는 것을 반환한다.
$n-1$을 $2^s d$ 형태로 바꾼다.
다음을 k 번 반복한다.
    $[1, n-1]$에서 임의의 a를 선택한다.
    $[0, s-1]$의 모든 r에 대해 $a^d \bmod n \neq 1$이고 $a^{2^r d} \bmod n \neq n-1$이면 합성수이다.
위 조건을 만족하지 않으면 소수일 것 같다.

ModInv() : 모듈러 역 값을 계산하는 함수

```
uint ModInv(uint a, uint m) {
	uint mod = m;
	uint q, t, x0 = 0, x1 = 1;
	if (m == 1) return 0;
	while (a > 1) {
		q = 0;
		t = a;
		while(t >= m) {
			q++;
			t -= m; }
		t = m;
		while(a >= m) a -= m;
		m = a;
		a = t;
		t = x0;
		x0 = ModAdd(x1, ModMul(q, x0, mod), 0, mod);
		x1 = t;
	} return x1;}
```

설명 : 확장 유클리드 알고리즘을 사용하여 모듈러 곱의 역원을 계산합니다. (예시)

유클리드 알고리즘

576 = 31 × 18 + 18
31 = 18 × 1 + 13
18 = 13 × 1 + 5
13 = 5 × 2 + 3
5 = 3 × 1 + 2
3 = 2 × 1 + 1 ∴ gcd(576,31) = 1, 서로소 이다.

구한 식을 거꾸로 한식에 대입합니다.

확장 유클리드 알고리즘(빠르게 계산 하기 위해 바로 "1 = ~"으로 시작하겠다.)

1 = 3 – 2 × 1
  = 3 – [5 – 3 × 1] × 1
  = 3 – [5 – [13 – 5 × 2] × 1] × 1
  = 3 – [5 – [13 – [18 – 13 × 1] × 2] × 1] × 1
  = 3 – [5 – [13 – [18 – [31 – 18 × 1] × 1] × 2] × 1] × 1
  = 3 – [5 – [13 – [18 – [31 – [576 – 31 × 18] × 1] × 1] × 2] × 1] × 1

———————————이해를 위한 순서———————————┘

  = 1 = 3 – 2
  = 3 – (5 – 3)
=> 2 * 3 – 5 << 정리를 해야 나중에 원하는 모양이 나옴
  = 2 * (13 – 5 * 2) – 5
=> 2 * (13) – 5 * 5
  = 2 * 13 – 5 * (18 – 13)
=> -5 * (18) + 7 * 13
  = -5 * 18 + 7 * (31 – 18)
=> 7 * 31 – 12 * (18)
  = 7 * 31 – 12 * (576 – (31 * 18))
=> -12 * 576 + 223 * 31

———————————실제 계산 순서———————————┘

=> -12 × 576 + 223 × 31 ≡ 1 (mod 576)

∴ 역원(우리가 찾는 값)은 223 이다.

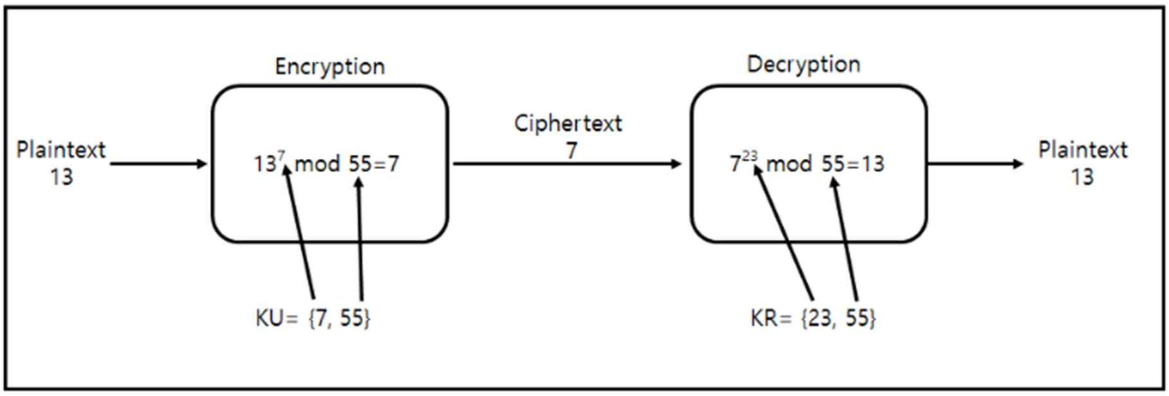
MRSAKeygen() : RSA 키를 생성하는 함수

```
void MRSAKeygen(uint *p, uint *q, uint *e, uint *d, uint *n) {
        printf("mRSA key generator start.₩n ");
        uint state[16]; uint temp = 0; uint qn;
        while(1){
                temp = (WELLRNG512a() * 100000);
                if(32768 < temp && temp < 65536){
                        printf("random_number1 %u selected ₩n",temp);
                        if (IsPrime(temp, 4)){
                                printf("%u may be Prime₩n",temp);
                                *q = temp; break;           //temp
                        }printf("%u is not Prime ₩n",temp);}}
        while(1){
                temp = (WELLRNG512a() * 100000);
                if(32768 < temp && temp < 65536){
                        printf("random_number2 %u selected ₩n",temp);
                        if (IsPrime(temp,4)){
                                printf("%u may be Prime₩n",temp);
                                *p = temp; break;
                        }printf("%u is not Prime₩n",temp); }}
        printf("finally selected prim q : %u   p : %u ₩n" ,*q , *p);
        *n = (*p) * (*q); qn = (*p -1 ) * (*q -1 );
        printf("thus, n = %u   qn =   %u ₩n" , *n ,qn);
        while(1){
                *e = (WELLRNG512a() * 10000000000);
                if(   1 < *e && *e < qn){printf(" e = %u selected₩n",*e);
```

```
                        if ( GCD(*e,qn) == 1){break;}
                }
        }
        *d = ModInv(*e ,qn);
        printf(" d = %u selected₩n",*d);
        printf(" e , d , n , qn : %u - %u - %u - %u ₩n",*e,*d,*n,qn);
        printf(" e*d mod qn : %u ₩n",ModMul(*e,*d,qn));
```

설명 :

예제 아래 그림은 RSA 알고리즘을 요약한 것입니다. 아래 그림의 내용을 가지고 키 생성 및 암/복호화를 진행해 보겠습니다. 예제에서는 작은 수의 소수를 선택하겠습니다.



- **키 생성**
  - 1) 서로 다른 큰 소수 p와 q를 선택한다. -> p=5, q=11 선택
  - 2) n= p*q를 계산한다. -> n= 5*11= 55
  - 3) φ(n)=(p-1)(q-1)를 계산한다. -> φ(n)= (5-1)(11-1)= 40
  - 4) φ(n)보다 작고 φ(n)과 서로소인 임의의 자연수 e를 선택한다. -> e= 7 선택
  - 5) 확장 유클리드 호제법을 이용하여 e mod φ(n)에 대한 곱의 역원, 즉 ed mod φ(n)=1인 d를 구한다.

- N = p*q는 2^31< N < 2^32 (32bit)의 높은 수로 선택되어야 합니다.

MRSACipher() : RSA 암복호화를 진행하는 함수

```
void MRSACipher(FILE *ifp, uint len, FILE *ofp, uint key, uint n) {
        printf("MRSACipher start. file len is %u ₩n",len);
        uint buf[len] ;
        uint buf2[len]; //buffer
        fread(buf,4,(len/4),ifp);
        for(int i = 0 ; i < (len/4) ; i++)
        {
                printf("len : %d ₩nptx : %u ₩nctx : %u ₩n"
                        ,(i+1) *4, buf[i] , ModPow(buf[i],key,n) );
                buf2[i] = ModPow(buf[i],key,n);
        }
        fwrite(buf2,4,(len/4),ofp);
        result = len;
        return result;
```

설명 : 암복화 값을 파일에 덮어 씁니다.