Randstate.c

Void randstate_init(uint64_t seed) {

Use gmp_randinit_mt to set the extern variable state to the Mersenne Twister algorithm

Use gmp_randseed_ui to set the seed to random

}

Void randstate_clear(void) {

Call to gmp_randclear() to clear the state.

}

Numtheory.c

//Pseudo code provided in asgn pdf

void pow_mod(mpz_t o, mpz_t a, mpz_t d, mpz_t n) {

```
POWER-MOD(a, d, n)
1  v ← 1
2  p ← a
3  while d > 0
4      if ODD(d)
5          v ← (v × p) mod n
6      p ← (p × p) mod n
7      d ← ⌊d/2⌋
8  return v
```

}

Bool isPrime(n, iters)

//Uses Miller-Rabin method to see if n is prime or not

//Write n - 1 = 2 ^ s * r such that r is odd

//Loop, start with s = 0 and r = n-1

while(r is even) {

Increment s by 1

Divide r by 2 (This should run until r is odd)

}

```
MILLER-RABIN(n, k)
 1  write n − 1 = 2^s r such that r is odd
 2  for i ← 1 to k
 3      choose random a ∈ {2, 3, ..., n − 2}
 4      y = POWER-MOD(a, r, n)
 5      if y ≠ 1 and y ≠ n − 1
 6          j ← 1
 7          while j ≤ s − 1 and y ≠ n − 1
 8              y ← POWER-MOD(y, 2, n)
 9              if y == 1
10                  return FALSE
11              j ← j + 1
12          if y ≠ n − 1
13              return FALSE
14  return TRUE
```

Void make_prime(mpz_t p, uint64_t bits, uint64_t iters)

Randomly generates numbers that should be *bits* long, and then makes sures the generated number is prime using *isPrime()* function.

Void gcd(mpz_t d, mpz_t a, mpz_t b)

Computes the greatest common divisor of a and b, storing the value of the computed divisor in d.

```
GCD(a, b)
1  while b ≠ 0
2      t ← b
3      b ← a mod b
4      a ← t
5  return a
```

void mod_inverse(mpz_t i, mpz_t a, mpz_t n)

```
MOD-INVERSE(a, n)
 1  (r, r') ← (n, a)
 2  (t, t') ← (0, 1)
 3  while r' ≠ 0
 4        q ← ⌊r/r'⌋
 5        (r, r') ← (r', r − q × r')
 6        (t, t') ← (t', t − q × t')
 7  if r > 1
 8        return no inverse
 9  if t < 0
10        t ← t + n
11  return t
```