numtheory.c

```
gcd()
While b != 0
    T = b
    B = a % b
    A = t
Return a

powermod()
V = 1
P = a
While d > 0
    If odd(d)
        V = (v * p) % n
    P = (p * p) % n
    D = foordiv(d / 2)
Return v

mof_inverse()
(r, rp) = (n, a)
(t, tp), = (0, 1)
While rp != 0
    Q = floordiv(r / rp)
    (r, rp) = (rp, r - q * rp)
    (t, tp) = (tp, t - q * tp)
If r > 1
    Return no inverse
If t < 0
T = t + n
```

```
Is prime()
Check edge cases

Init bitcount
While n-1 is even:
    Increment bitcount
    mpz_tdiv_q_2exp(r, r, 1);
For i = 1 to k
    Choose random a [2, n-2]
    Y = powermod()
    If y != 1 and y != n-1
        J = 1
        While j <= s - 1 and y != n - 1
            Y = powermod()
            If y == 1
                Return false
            J = j + 1
        If y != n - 1
            Return false
Return true

Make prime()
Do
    Generate prime()
While (its not prime)
```

keygen.c

```
Headers
Defaults
Flags

While getopt()...

Check flags

fopen(public)
fopen(private)
        assert(...)

fchmod(privatekeyfile), fileno(privatekeyfile)

randstate()

makepub()
makepriv()
genv(username)
write(private + pub keys)
Check verbose flag()
fclose(keys)
randstate_clear()
```

ss.c

```
ss_make_pub()
    make_prime()
    Generate random number from [nbits / 5, (2 * nbits) / 5]
    Calculate bits for q (nbits - 2 * pbits)
    Generate p, q, checking for conditions p % q - 1 != 0, vice versa

ss_write_pub()
    fprint(n)
    fprint(username)
    Format “%Zx”

ss_read_pub()
    Char []
    fscan(n)
    While != EOF
        fscan(character, add to char)
ss_make_priv()
    mod_inverse(n, lcm(p-1, q-1))

ss_write_priv()
    fprint(pq)
    fprint(d)

ss_read_priv()
    While != EOF
        fscan()
```

```
ss_encrypt()
E(m) = c = m^n % n

ss_encryptfile()
K = floordiv(logbase2(sqrtn) - 1) / 8)
malloc() uint8_t * k
Array[0] = 0xff
While != eof
    Read <= k -1 bytes, place bytes into array, starting at index 1
    mpz_import() convert read bytes into mpz_tm
    ss_encrypt(m)
    Write to outfile

ss_decrypt()
D(c) = m = c^d % pq

Ss_decryptfile
    Just reverse the steps of encryptfile()
```

randstate.c

```
randstate_int()
    gmp_randinit_mt(state)
    gmp_randseed_ui(seed)

randstate_clear()
    gmp_randclear(state)
```

decrypt.c

```
Headers
Defaults
Flags

While getopt()...

fopen(private)
read(private)
Check flags

ss_decrypt_file()
close(private file)
clear()
```

encrypt.c

```
Headers
Defaults
Flags

getopt()

fopen(pubkey)
        assert(...)
read(pubkey)
Check flags

encrypt_file()
close_file()
```