Public key cryptography:

- going to need GNU library for Multiple Precision Arithmetic
- going to implement certain number theory functions using this Precision library.
- need to test if generated numbers are prime. we do this using the Miller-Rabin test.
- need a function for gcd using Euclid's algorithm and a function for modular inverse.
- we use these functions to build an RSA library! we generate a Public key and a Private key for encrypted communication!

modular exponentiation

function takes a base, exponent, modulus

n = modulus

V = 1

P = exponent

while d > 0

V = V · P mod n if d is odd

P = P × P mod n

$d = \frac{d}{2}$

return V after loop!

testing for Prime: takes n and iteration for tests

$n-1 = 2^s r$ r is odd! (tests n)

for 1 to k as i

chose rand num from $\in \{2 \ldots n-2\}$

y= mod power (a, r, n)

if $y \neq 1$ and $y \neq n-1$

j=1

while $j \leq s-1$ and $y \neq n-1$

y= Power mod (y, 2, n)

if y=1 return false if $y \neq n-1$ return false

j= j+1

return true!

Use is_prime function to help generate prime numbers.

need to find gcd of a and b

GCD:

while $b \neq 0$

$t = b$

$b = a \bmod b$

$a = t$

at the end, return a copied into t

modular inverse: takes a and n.

$(r, r') = (n, a)$

$(t, t') = (0, 1)$

while $r' \neq 0$

$q = \frac{r}{r'}$

$(r, r') = (r', r - q \times r')$

$(t, t') = (t', t - q \times t')$

if $r > 1$
  no inverse

if $t < 0$
  $t += t + n$

at the end
return t!

rsa:

make_pub: generates two large primes using prev. functions, their product and exponent e.

write_pub: writes the public key to a file

read_pub: reads the pub key from file

make_priv: creates a private key w 2 large primes and exponent e. does inverse mod

write_priv: writes priv key into a file.

read_priv: reads private key from file

encrypt: encrypts using pub key

decrypt: decrypts using priv key

keygen: generates a public and private key using prev. functions.

encrypt: encrypts a file with the public key that can be decrypted with the private key

decrypt: decrypts a file that was encrypted with user's public key so it can be decrypted with the private key

$n-1 = 2^s r$ r is odd. miller rabin

start w $s=0$

$n-1 = 2^0 r$

$12 = 2^3 r$
$12 = 2^1 r$
$12 = 2^2 r$

increase s every loop!

go until r is odd!

while (r is even)
$r = n-1/2^s$
s++

is n is even and not 2
return false!

rsa.C

make_pub_key: call make_prime and calculate how many bits are needed for q & p and increase that number by one

make_private_key: need p & q from public key to generate key

read & write: use fscanf(gmp) and fprintf(gmp) respectively

encrypt & decrypt: use pow_mod

encrypt: $E(m) = C = m^e \bmod(n)$

decrypt: $D(c) = M = c^d \bmod(n)$

encrypt & decrypt file: read in a buffer, turn into number, encrypt and write to file.

decrypt: read in encypted line, decrypt it, fill buffer, write buffer to file

keygen: generate Public key make_pub
- Calculate private key make_priv
write to files (rsa.pub) (rsa.priv) default
free all MPZ_t's that were needed

encrypt.c:

open file to encrypt
read in public key! (username from keygen)
call encrypt file (with pub key)
close all files
clear variables

decrypt.c:

read in private key.

run decrypt file on encrypted file.

this will write the decrypted file to outfile.

close all files

clear all MPZ_t

making numbers nbits long

generate a random num $[0 - 2^{nbits-1} - 1]$

then add $2^{nbits-1}$

this will give you an nbit random number