Assignment 6 - Public Key Cryptography

Description: This program will demonstrate the encryption and decryption of private and public keys using the RSA algorithm. The keys represent shared data between two parties that can be shared publicly in the form of two keys. This program will generate public and private keys along with encoding and decoding the keys using GNU libraries of high math precision to be utilized. SSH key generation and sharing of keys allows for highly sensitive information to be shared in a secure manner by using power mod.

DELIVERABLES
Makefile:
• CC = clang must be specified.
• CFLAGS = -Wall -Wextra -Werror -Wpedantic must be specified.
• pkg-config to locate compilation and include flags for the GMP library must be used.
• make must build the encrypt, decrypt, and keygen executables, as should make all.
• make decrypt should build only the decrypt program.
• make encrypt should build only the encrypt program.
• make keygen should build only the keygen program.
• make clean must remove all files that are compiler generated.
• make format should format all your source code, including the header files.

README.md: This must use proper Markdown syntax and describe how to use your program and Makefile. It should also list and explain any command-line options that your program accepts. Any false positives reported by scan-build should be documented and explained here as well

DESIGN.pdf: This document must be a proper PDF. This design document must describe your design and design process for your program with enough detail such that a sufficiently knowledgeable programmer would be able to replicate your implementation.

decrypt.c: This contains the implementation and main() function for the decrypt program.

encrypt.c: This contains the implementation and main() function for the encrypt program.

keygen.c: This contains the implementation and main() function for the keygen program.

numtheory.c: This contains the implementations of the number theory functions.

numtheory.h: This specifies the interface for the number theory functions.
randstate.c: This contains the implementation of the random state interface for the RSA library and number theory functions.
randstate.h: This specifies the interface for initializing and clearing the random state.

rsa.c: This contains the implementation of the RSA library.
rsa.h: This specifies the interface for the RSA library

SOURCE FILE PSEUDOCODE:
randstate.c

randstate_init(uint64) initializes global random "state" with MT algorithm and using seed and a random seed. Calls gmp functions of randinit_mt and randseed_ui

randstate_clear (frees and clears memory of "state" and does so by calling gmp_randclear()

numtheory.c

```
is_prime (n,iters)
If n is 0 or 4 return false
If n is 2 or 3 return true
write n-1 = 2^(s)*r such that r is odd
While (r is even)
        s iterates +1 from 0
        r=(n-1)/2^(s)
Iterates through inters to set a random 'a' between [2:n-2]
        y= powermod (random,r,n)
        if( y!=1 and y!=n-1)
                j=1
                While (j<=s-1  and y!=n-1)
                        powermod(y,2,n)
                        If y=0 return false
                        j++
                If y!=n-1 return false
Return ture
```

make_prime(p,bits,iters)
Generates a prime number using mpz_randomb() with at least “bits” number of bits and is_prime to check the number. Uses iters to pass through is_prime()to test the number.

gcd(d,a,b)
Commutes greatest common denominator between a,b and stores it in d
Utilizes a while loop to take in b and send b to a mob b and continue to do so while b not equal 0.

mod_inverse(i, a ,n) computes the mod inverse i of modulo n
r,r',t,t' are stored as n,a,0,1
while r' == 0
q=floor[r/r']
(r,r')=(r',r-q*r'))
(t,t')=(t',t-q*t')
if r greater than 1 return 0
if t less than 0 return t+n
Return t

rsa.c

make_pub (p,q,n,e,nbits,iters)
set bits for p between [nbits/4:(3*nbits)/4] and q bits equal nbits-p
make_prime for p and q
n is product of p and q
totient of p and q is (p-1)(q-1)

find coprime of totient and set to e using mpz_urandomb to find a gdc that equals 1 between the totient and random

write_pub()
writes out the public key to pbfile. Formatted as n,e,s written as hex strings and then a username

read_pub()
reads RSA key from pbfile and the format of a public should be n,e,s then the username and utilized gmp formatted inputs for reading hex strings.

void rsa_make_priv(mpz_t d, mpz_t e, mpz_t p, mpz_t q)
creates a new RSA private key “d” computed with the inverse of e modulo $\phi(n) = (p-1)(q-1)$.

void rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile)
writes out n,d to pvfile

void rsa_read_priv(mpz_t n, mpz_t d, FILE *pvfile)
scans n,d from pvfile and stores it

void rsa_encrypt(mpz_t c, mpz_t m, mpz_t e, mpz_t n)
encrypts m,e,n using powermod and store in c
dynamically allocate array of uint8_t size k
sets zero element of kbytes to 0xff
while there are unread bytes in infile
read k-1 bytes into kbytes
import kbytes to mpz then encrypt and write out the encryption as hexstring to outfile

void rsa_encrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t e)
block size k equals bits of n -1 divided by 8

void rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n)
encrypt c,d,n using pow mod passing it to m

rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d)
encrypts m,e,n using powermod and store in c
dynamically allocate array of uint8_t size k
sets zero element of kbytes to 0xff
while there are unread bytes in infile
scan in a block of c hexstring
decrypt and export them into an mpz.
write out kytes starting from first element to outfile

void rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n)
signs m,d,n using power mod and passing it to s

bool rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n)
verifies that m is equal to the powermod of s,e,n

keygen.c

Getopt command
- Case b set bits
- Case i sets iterations
- Case n sets pubfile
- Case d sets privfile
- Case s sets seed for random
- Case v sets stats
- Case h prints helpful message

Created and opens file with reading and writing permissions
prints help message if pbfile is NULL
creates and opens file with reading and writing permissions
prints help message if pvfile is NULL
sets the permission of pv file to 0600
sets seed to randstate
makes pub key
makes priv key
gets user and passes it to username
sets username to mpz
signs using usr
writes out key pub to pbfile
writes out key to pvfile
prints out variables and stats if true
clears out mpz,randstate,and close pbfile and pvfile

encrypt.c

getopt commands for encryption
- Case ‘i’ opens specified infile for reading
- Case ‘o’ opens specified outfile for reading and writing
- Case ‘n’ sets name of pubfile
- Case ‘v’ sets print stats to true
- Case ‘h’ prints helpful message

opens pubfile and reads
prints helpful message if opening pubfile fails
prints helpful message if opening infile fails
prints helpful message if opening outfile fails

reads pubkey file and stores values n,e,s,username usign read_pub()
prints out states of variables with username if verbose is true
sets username to usr mpz
determines if user has permission to open file
encrypts file into hexstring
clears mpz,closes file

decrypt commands using getopt
- Case ‘i’ opens infile for reading
- Case ‘o’ opens outfile for reading and writing
- Case ‘n’ sets name of privkey file
- Case ‘v’ sets print varaible state to true
- Case ‘h’ prints helpful message

opens privkey file for reading
prints help message if privkey fails to open
prints help message if infile fails to open
prints help message if outfiel fails to open
passes in n and d from privkey file using read_priv
prints out stats of variables if verbose is true prints out decrypted message to outfile
clears mpz and closes files

Notes:
Uses assignment pseudocode provided by Professor Long
Is_prime used Euguene pseudocode explanation