Assignment 6: Public Key Cryptography

DESCRIPTION

This program implements the RSA method of encryption using numerical functions. There will be 3 executables: keygen, encrypt and decrypt. The keygen program handles key generation by producing public and private pairs of RSA keys. The encrypt program uses the generated public key to encrypt data, and the decrypt program uses the paired private key to decrypt the encrypted files. The RSA functions are written in rsa.c and use the mathematical functions in numtheory.c. A randstad, which can take an input seed, is used for random number generation.

PSEUDOCODE

numtheory.c

```
function gcd(mpz_t d, mpz_t a, mpz_t b)
    t = 0
    while b is not 0
        t = b
        b = a % b
        a = t
    d = a

function mod_inverse(mpz_t i, mpz_t a, mpz_t n)
    r = n
    r' = a
    t = 0
    t' = 1
    while r' is not 0
```

```
            q = r/r'
            r = r'
            r' = r - q x r'
            t = t'
            t' = q x t'
        if r > 1
            i = 0
        if t < 0
            t = t + n
        i = t

function pow_mod(mpz_t out, mpz_t base, mpz_t exponent, mpz_t modulus)
        if exponent is 0
            out = 1
        else
            out = 1
            for loop (i = 0, i < exponent, i += 1)
            out *=  base
        out = out % modulus

function is_prime(mpz_t n, uint64 iters)
        //conduct miller-rabin primality test on n using iters number of iterations
        solve n = 2^s r + 1 for s and r //involves a loop
        for loop (0 to k - 1 inclusive)
            a = random number {2, 3…, n - 2}
            y = pow_mod(a, r, n)
            if y is not 1 and y is not n - 1
                j = 1
                while j <= s - 1 and y is not n - 1
                    y = pow_mod(y, 2, n)
                    if y is 1
```

```
                    return false
                j += 1
            if y is not n - 1
                return false
    return true

function make_prime(mpz_t p, uint64 bits, uint64 iters)
    generate new prime number at least bits bits long
    check if its prime using is_prime(new prime number, iters)
    if its prime
        p = new prime number
```

randstate.c

```
function randstate_init(uint64 seed)
    gmp_randinit_mt(state);
    gmp_randseed_ui(state, seed);


function randstate_clear(void)
    gmp_randclear(state)
```

rsa.c

```
function rsa_make_pub(mpz_t p, mpz_t q, mpz_t n, mpz_t e, uint64 nbits, uint64 iters)
    make_prime(p)
    make_prime(q)
    decide number of bits going to each prime
    compute
    for loop (exits when coprime with totient is found)
        use mpz_urandomb() to generate random numbers around nbits
```

compute gcd() of the random number
if coprime with totient is found
set e to that value
terminate the loop

function rsa_write_pub(mpz_t n, mpz_t e, mpz_t s, char username, FILE *pbfile)
write() n, e, s, then username (each as hexstrings with trailing newlines) to pbfile

function rsa_read_pub(mpz_t n, mpz_t e, mpz_t s, char username, FILE *pfile)
read() n, e, s then username from the pbfile

function rsa_make_priv(mpz_t d, mpz_t e, mpz_t p, mpz_t q)
create new private key d using d = (p - 1)(q - 1)

function rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile)
write() n, then d (each as hexstrings with trailing newline) to pvfile

function rsa_read_priv(mpz_t n, mpz_t d, FILE *pvfile)
read() n, then d from pvfile

function rsa_encrypt(mpz_t c, mpz_t m, mpz_t e, mpz_t n)
compute c by encrypting message m using $c = m^e \pmod n$

function rsa_encrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t e)
calculate block size k
dynamically allocate array that can hold k bytes
set zeroth byte of the block to 0xFF
while there are still unprocessed bytes in infile
read k-1 bytes from infile into allocate block starting from 1
mpz_import() to convert read bytes into mpz_t m
ncrypt m with rsa_encypt

write encrypted number to outfile as hexstring with trailing newline

function rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n)

compute m by decrypting c using $m = c^d \pmod n$

function rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d)

calculate block size k

dynamically allocate array that can hold k bytes

while there are still unprocessed bytes in infile

scan a hexstring to mpz_t c

mpz_import() to convert c into bytes and stored into allocated block

j is the number of bytes converted

write j - 1 bytes from index 1 of the block to outfile

function rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n)

sign message s using $s = m^d \pmod n$

function rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n)

$t = s^e \pmod n$

if t is the same as m

return true

return false

help_print

encrypt.c, decrypt.c, and keygen.c all use help_print to print help messages. The functions are called when the command '-h' is detected in the switch statements.

encrypt.c

```
int main
        while loop
                switch
                        check for commands h, i, o, n, or v
                        if h then call help_print
                        if i, p, or o then set them to the input filenames
                        if v then enable verbose statistics

        rsa_read_pub(n, e, s, username, rop)
        mpz_set_str(rop, username, 52)

        if rsa_verify(rop,  s, e, n) is false
                print a helpful error message
                return
        return
if verbose is true
        print verbose statistics

rsa_encrypt_file(infile, outfile, n, e)
return
```

decrypt.c

```
int main
        while loop
                switch
                        check for commands h, i, o, n, or v
                        if h then call help_print
                        if i, p, or o then set them to the input filenames
                        if v then enable verbose statistics
```

```
        rsa_read_pub(n, e, s, username, rop)
        if verbose is true
                print verbose statistics
rsa_encrypt_file(infile, outfile, n, d)
return
```

keygen.c

```
int main
        while loop
                        switch
                                check for commands h, i, o, n, or v, d, b, s
                                if h then call help_print
                                if i, p, or o then set them to the input filenames
                                if v then enable verbose statistics
                                set seed to s
                                set min_bits to b


        fchmod(fileno(rsa_priv), 0600)
        rsa_make_pub(p, q, n, e, min_bits, iters)
        rsa_make_priv(d, e, p, q)

        get username with getenv("USER")
        rsa_sign(s, rop, d, n)

        rsa_write_pub(n, e, s, username, rsa_pub)
        rsa_write_priv(n, d, rsa_priv);

        if verbose == true
                print verbose stats
```

```
    return;
rsa_encrypt_file(infile, outfile, n, d)
return
```

ERRORS

- The primary source of errors is the command input when running the executables encrypt, decrypt, or keygen. However, if no parameters are specified, or invalid input is entered, the program will print a list of the correct commands for the user to try again.
- There are defaults set for certain parameters like the seed, and files to prevent errors.

FILES

- decrypt.c
    - Contains the main function for the decrypt program
- encrypt.c
    - Contains the main function for the encrypt program
- keygen.c
    - Contains the main function for the keygen program
- numtheory.c
    - Contains the implementations of the number theory functions
- numtheory.h
    - The header file which specifies the interface for the number theory functions
- randstad.c
    - Contains the implementation of the random state interface for the RSA library and number theory functions
- randstad.h
    - The header file which specifies the interface for initializing and clearing the random state.
- rsa.c

    - Contains the implementation of the RSA library
- rsa.h
    - The header file which specifies the interface for the RSA library
- Makefile
    - The makefile which builds executable programs using the .c and .h files.
- README.md
    - Contains program information regarding the basic description, building, and error handling
- DESIGN.pdf
    - The pdf which details a description of the program assignment, the structure and pseudocode, a list of files within the submission, and any errors or references.

REFERENCES

- I referred to the pseudocode in the assignment 6 document for the functions and mathematical equations
- I used the *C Programming Language* by Kernighan and Ritchie as reference for information on data types and functions
- I referred to TA Eugene's and TA Christian's pseudocode from their TA sections