cse13s asgn5 DESIGN.pdf

Lucas Lee; CruzID: luclee

2/4/2022

1 Program Details

We will create three functions: one that creates a public and private key pair using large prime numbers, one that encrypts a file using a public key, and one that decrypts the file using the private key. We will use a randomizer to generate the large primes for the keys.

2 functions and pseudocode

1. randstateinit(seed)
    (a) allocate memory for "state"
    (b) set srandom to seed
    (c) use "state" as a global variable to keep track of random state
2. randstateclear()
    (a) clear memory of state and free memory
3. powmod(out, base, exponent, mod)
    (a) temp = 1
    (b) temp2 = base
    (c) while exponent ¿ 0
        i. if exponent is odd
            A. temp = (temp x temp2) mod(mod)
        ii. temp2 = (temp2 x temp2) mod(mod)
        iii. exponent = exponent floor division by 2
    (d) out = temp

4. isprime(n, iters)
    (a) r = (n-1) / $2^s r$. r should be odd
    (b) for loop from 1 - iters
        i. a = random number from (2, n-2)
        ii. y = powmod(a, r, n)
        iii. if y is not 1 or n-1
            A. j = 1
            B. while j less than or equals s-1 and y not equals n -1
            C. if y == 1 return
            D. j = j + 1
            E. if y not equals n - 1. Return false
    (c) return true
5. makeprime(p, bits, iters)
    (a) generates prime number and stores in p
    (b) set number of bits of p to bits
    (c) test using isprime(p, iters)
6. gcd(d, a, b)
    (a) while b is not 0
        i. t = b
        ii. b = a mod b
        iii. a = t
    (b) return a
7. modinverse(i, a, n)
    (a) r = n
    (b) rp = a
    (c) t = 0
    (d) tp = 1

(e) while rp not equal to 0

i. q = r/rp

ii. r = rp

iii. rp = r - q x rp

iv. t = tp

v. tp = t - q x tp

(f) if r greater than 1: return no inverse

(g) if t less than 0: t = t + n

(h) return t

8. rsamakepub(p, q, n, e, nbits, iters)

(a) p, q = prime numbers, n = product of p and q, e = exponent

(b) makeprime() to make p and q

(c) log2(n) should be greater than nbits

(d) p bits in range (nbits/4, (3 x nbits) /4)

(e) q gets remaining bits from the calculation

(f) random number using random() and iters to check prime

(g) lambda(n) = lcm(p-1, q-1)

i. do this by calculating gcd(p-1, q-1) and comparing it to the product of p-1 and q-1

(h) get random numbers around nbits

(i) get the gcd of each random number to find lambda(n)

(j) coprime number lambda(n) = public exponent

9. rsawritepub(n,e,s, char username, file *pbfile)

(a) open pbfile for writing (if not already open)

(b) print n in hex with a newline

(c) print e in hex with a newline

(d) print s in hex with a newline

(e) print username with newline

(f) close pbfile

10. rsareadpub(n,e,s, char username, file *pbfile)

    (a) open pbfile for reading(if not already open)

    (b) scan each line to read then into variables

    (c) scan first line = n

    (d) scan second line = e

    (e) scan thrid line = s

    (f) scan fourth line = username

    (g) close pbfile

11. rsamakepriv(d,e,p,q)

    (a) d = private key to be created, e = public exponent, p and q = primes

    (b) d = inverse of e mod lambda(n)

12. rsawritepriv(n,d,file *pvfile)

    (a) open pvfile for writing (if not already open)

    (b) write n as a hexstring followed by newline

    (c) write d as a hexstring followed by newline

    (d) close pvfile

13. rsareadpriv(n,d, file *pvfile)

    (a) open pvfile for reading(if not already open)

(b) scan lines to assign values to variables

(c) n = scan first line

(d) d = scan second line

14. rsa encrypt(c,m,e,n)

    (a) $c = m^e (mod n)$

15. rsa encrypt file(file *infile, file *outfile, n, e)

    (a) encrypt in blocks from infile to outfile

    (b) block values cannot be 0 or 1

    (c) create block size $k = \lfloor (log_2(n) - 1/8) \rfloor$

    (d) malloc to allocate array that can hold k bytes as a uint8

    (e) set array at 0 to 0xFF

    (f) while unprocessed bytes in infile

        i. read k - 1 bytes from infile and place them into the allocated block array starting from 1

        ii. convert the read bytes including array(0) into mpzt m.

        iii. encrypt m using rsa encrypt() and write it to outfile as a hexstring with a newline.

16. rsa decrypt(m, c, d, n)

    (a) compute message m

    (b) $m = c^d (mod m)$

17. rsa decrypt file(file infile, file outfile, n, d)

    (a) allocate memory for block size similar to encrypt file

    (b) while unprocessed bytes in infile

        i. scan hexstring, save hexstring in variable.

ii. convert each hexstring back into bytes using mpxexport() j = number of bytes converted

iii. write out j-1 bytes starting from array(1) to outfile

18. rsa sign(s,m,d,n)

    (a) $s = m^d (mod n)$

19. rsa verify(m,s,e,n)

    (a) var t = $s^e (mod n)$

    (b) if t = m: return true

    (c) else: return false

3 main files and command line inputs

Key Generator

(a) -b = minimum bits for mod(n)

(b) -i = number of iterations to test prime numbers. Default 50

(c) -n pbfile = specifies file that has public key. Default rsa.pub

(d) -d pvfile = specifies file that has private key. Default rsa.priv

(e) -s = specifies random seed for random state. Default time(NULL)

(f) -v = verbose output

(g) -h = synopsis and usage

(h) set file permissions to 0600 using fchmod and fileno

(i) randstate init(s)

(j) make public and private keys

(k) getenv() to get username as string

(l) convert username using mpz set str() with a base of 62

(m) write public key to pbfile and private key to pvfile

(n) if verbose output:

i. print username with newline

ii. print signature s with newline

iii. print prime p with newline

iv. print prime q with newline

v. print mod(n) with newline

vi. print exponent e with newline

vii. print private key d with newline

viii. each of these lines should have number of bits for each and the decimal value that correspons to them

ix. randstate clear() and close/clear all files and variables

Encrypt

i. -i = input file to encrypt. Default = stdin

ii. -o = output file to encrypt to. Default = stdout

iii. -n = file containing public key. Default = rsa.pub

iv. -v = verbose output

v. -h = synopsis and usage

vi. open file and exit program if there is a problem opening the file

vii. read public key from pbfile

viii. if verbose:

A. print username with newline

B. print signature s with newline

C. print mod(n) with a newline

D. print exponent e with a newline

E. print each with their number of bits and their values as a decimal

ix. convert username to mpzt and verify it using rsa verify()

x. encrypt file using rsa encrypt file()

xi. close pbfile and clear mpz variables

Decrypt

i. -i = input file to decrypt. Default = stdin
ii. -o = output file to decrypt to. Default = stdout
iii. -n = specifies file containing private key. Default = rsa.priv
iv. -v = verbose output
v. -h = synopsis and usage
vi. open private key file. Print error if failed
vii. read private key from pvfile
viii. if verbose:
    A. print mod(n) with newline
    B. print private key e with newline
    C. both should print number of bits and their values in decimal
ix. decrypt file using rsa decrypt file()
x. close pvfile and clear mpz variables