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

for each function that needs random states, declare global variable state.

```
1. randstateinit(seed)
    (a) set Mersenne Twister initializer to state
    (b) set srandom to seed
    (c) set randseed with variable state to random seed
2. randstateclear()
    (a) clear memory of state
3. powmod(out, base, exponent, mod)
    (a) create mpz temp variables to store parameters so they do not change
    (b) v = 1
    (c) p = base
    (d) tempexpo
    (e) while tempexpo ¿ 0
```

```
            i. if tempexpo is odd
                A. v= (v x p) mod(mod)
           ii. p = (p x p) mod(mod)
          iii. tempexpo = tempexpo floor division by 2
      (f) out = v
      (g) clear mpz variables

4. isprime(n, iters)

      (a) return true if n is less than 4
      (b) return false if n is even
      (c) r = (n-1) / $2^s r$. r should be odd
            i. create mpz values s, r, temp to hold values in the above
               equation
           ii. s = 0, r = 1, temp = n-1
          iii. while temp is not 1
                A. floor divide temp by 2
                B. s += 1
                C. if temp is odd
                D. r = temp
                E. break loop
      (d) for loop from 1 - iters
            i. create mpz variables to store temporoary values for calcula-
               tions
           ii. a = random number from (2, n-2)
          iii. y = powmod(a, r, n)
           iv. 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, clear mpz vars and return
                D. j = j + 1
                E. if y not equals n - 1. clearn mpz vars and return false
      (e) clear all mpz vars used in function
      (f) return true
```

5. makeprime(p, bits, iters)

    (a) create mpz variables to store temporary values. bitcount, prime, randombits

    (b) bitcount = bits

    (c) bitcount = $2^(bits)$

    (d) set p = bitcount

    (e) set randombits = bits -1

    (f) set prime to a random number from $2^b + 2^b - 1$ if b = bits

    (g) test using isprime(p, iters)

        i. keep randomizing prime to a number in the same range to test for each iteration iters

    (h) clear mpz variables and p should be set to the prime number

6. gcd(d, a, b)

    (a) make temp variables so d, a, and b are not altered by function

    (b) while tempb is not 0

        i. t = tempb

        ii. tempb = tempa mod tempb

        iii. tempa = t

    (c) d = tempa

    (d) clear mpz vars

7. modinverse(i, a, n)

    (a) make temp mpz variables to store

    (b) r = n

    (c) rp = a

- (d) t = 0
- (e) tp = 1
- (f) 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
- (g) if r greater than 1: i = 0
- (h) if t less than 0: t = t + n
- (i) i = t
- (j) return i

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
        - ii. lcm(p-1, q-1) = absolute value(p-1 * q-1) / gcd(p-1, q-1)
    - (h) get random numbers around nbits

- (i) get the gcd of each random number to find lambda(n)
- (j) while gcd(e, lcm) is not 1 (not coprime)
    - i. randomize e from 0 - nbits
- (k) coprime number lambda(n) = public exponent e
- (l) clear mpz variables

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 = modinverse(e, lcm)

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) use powermod to compute c
    (b) $c = m^e (modn)$

15. rsa encrypt file(file *infile, file *outfile, n, e)
    (a) encrypt in blocks from infile to outfile
    (b) create block size $k = \lfloor (log_2(n) - 1/8) \lfloor$
    (c) malloc to allocate array that can hold k bytes as a uint8
    (d) set array at 0 to 0xFF
    (e) while unprocessed bytes in infile (using fread - bytes may not be numbers, so scan wont work)
        i. read k - 1 bytes from infile and place them into the allocated block array starting from 1 (fread)

ii. convert the read bytes including array(0) into mpzt m (use mpz import for this)

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

(f) close files (unless closed in functions) and free array

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

    (a) compute message m using powermod

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

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(use feof() to indicate when the end of the file is reached)

    i. scan hexstring, save hexstring in variable.

    ii. convert each hexstring back into bytes using mpzexport() j = number of bytes read

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

    (c) print newline to outfile for syntax

    (d) free array and close files (unless closed in functions)

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

    (a) calculate s by using power mod

    (b) $s = m^d (modn)$

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

    (a) calculate t by using power mod

    (b) var t = $s^e (modn)$

    (c) if t = m: return true

    (d) 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) fileno returns int, so store it in a variable and use it for fchmod

(j) fchmod(fileno integer, 0600 permissions)

(k) randstate init(s)

(l) make public and private keys

(m) getenv(USER) to get username as string

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

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

(p) if verbose output:

    i. sizeinbase(mpz, base2) to get bit numbers

    ii. print username with newline

    iii. print signature s with newline

    iv. print prime p with newline

    v. print prime q with newline

    vi. print mod(n) with newline

vii. print exponent e with newline

viii. print private key d with newline

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

x. 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 using set str() 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