ASGN #6 DESIGN

RSA:

public key: n, e <e, n>

private key: d

secret

p, q

d

$\varphi(n)$

public

n

e

random

random prime number: p, q. private

let $n = pq$. publish

$\varphi(n) = (p-1)(q-1)$

choose e : $2 < e < n$ && $\gcd(e, \varphi(n)) = 1$

comprime

side

$$\gcd(a,b) = \begin{cases} a & \text{if } a = b \\ \gcd(a, b-a) & \text{if } a < b \\ \gcd(a-b, a) & \text{if } a > b \end{cases}$$

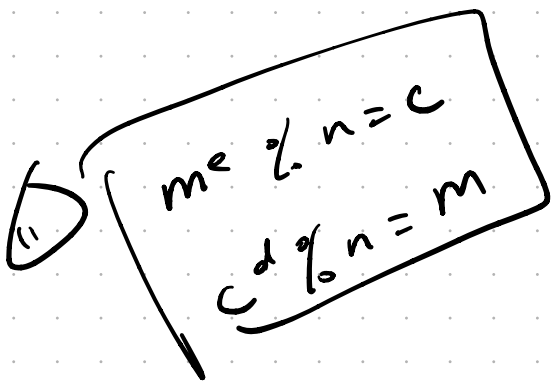
if p is prime.

$\varphi(p) = p - 1$

$\therefore \varphi(n) = \varphi(pq) = (p-1)(q-1)$

$d \times e \equiv 1 \pmod{\varphi(n)}$ => $d \times e \bmod \varphi(n) = 1$

$D(E(m)) = E(D(m)) = m$



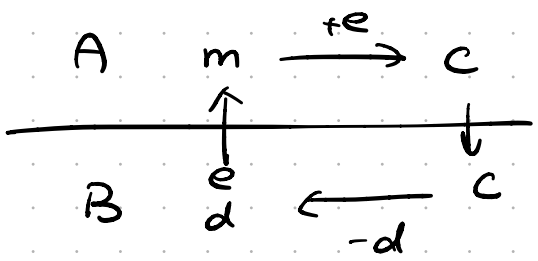
keygen :

p, q : prime

$n = pq$.

encrypt :

decrypt :



$m^e \% n = c$

$c^d \% n = m$

$\varphi(n) = \varphi(pq) = (p-1)(q-1)$

e . $\varphi(n)$ comprime

① remember to make a copy of input value in parameters !

② remember to do cal step by step !

randstate.c

- randstate_int (u64 seed)

  gmp_randseed_ui (state, seed) // use seed to make a pseudorandom

  gmp_randinit_mt(state) // get random value.
  ↳ result "return)

- randstate_clear

  gmp_randclear() // clear rand.

numtheory.c

- gcd (d, a, b) → return

  make a copy of each value first

$$\gcd(a,b) = \begin{cases} \text{if } a = b & a \\ \text{if } a < b & \gcd(a, b-a) \\ \text{if } a > b & \gcd(a-b, b) \end{cases}$$

```
If (a == b)
    d = a   (?)
else If (a < b)
        gcd( a, b-a)
else
        gcd (a-b, b)
```

- mod_inverse (t, a, n) → return

```
r = n
r' = a
t = 0
t' = 1
```
) order is not necessary ∴ not assigning the variable as a value

```
while r' ≠ 0
    q = ⌊r / r'⌋   ⇒ take the floor   (fdiv)
    temp = r
    r = r'
    r' = temp          ) swap r & r'

    temp = t
    t = t'
    t' = temp          ) swap t & t'

if r > 1                (cmp)
    return no inverse  // t = 0

if t < 0
    t = t + n

return t
```

- pow_mod (out, base, exponent, modulus) → return
  (a, d, ^)

```
v = 1
p = a
while d > 0
    if d is odd   → d % 2 == 1
        v = (v × p) % n
    p → (p × p) % n
    d = ⌊d / 2⌋
return v → out
```

— is_prime → : 0. 1. 3 are special cases ← if (x) check 0. 1. 3 first
// write $n-1 = 2^s r$ such that r is odd

× prime ↑ prime

```
mpz_t n
sub_ui(n1, n, 1)  // n-1
mpz_bitint_t s = 2 // s=2
// times 2 everytime
while (mpz_divisible_2exp_p(n1, s)){
    s++
}
s--
mpz_tdiv_q_2exp(r, n1, s) // r = (n-1)/2^s
```

```
for(i=0; i < iters ; i++){
    upper_bound = comp_n - 3   ← make 3 as a special case
    urandom(a, state, upper_bound  // choose random
    offset a by add 2  (a+2 => a ∈ {2, ..., n-2}
    y = power_mod(a, r, n)
    if  cmp(y, 1) != 0 &&  cmp(y, n-1) != 0
        j = 1
        while j <= s-1 && y != n-1
            y = power_mod(y, 2, n)
            if (y == 1)
                → false ←
            j = j+1
        if y != n-1
            false
true
```

give false positives.

MILLER-RABIN(n, k) → k is # of iterations
1 write $n-1 = 2^s r$ such that r is odd → s: guess as well r is odd to begin with ⇒ stop.
2 **for** i ← 1 **to** k ← guess & loop
3 choose random $a \in \{2, 3, \ldots, n-2\}$
4 y = POWER-MOD(a, r, n)
5 **if** $y \neq 1$ and $y \neq n-1$
6 $j \leftarrow 1$
7 **while** $j \leq s-1$ and $y \neq n-1$
8 $y \leftarrow$ POWER-MOD(y, 2, n)
9 **if** $y == 1$
10 **return** FALSE
11 $j \leftarrow j+1$
12 **if** $y \neq n-1$
13 **return** FALSE
14 **return** TRUE

$n-1 = 2^s r$
$nk - = n$
$r \times 1 = r$
$2^s = 1$
$s = 0$

initial: s=0 r=n-1
$n-1 = 2^s(n-1)$
s+=1 ← add 1 to s every loop

$\Lambda = 2^s r + 1$
r is odd

$s = 2^s r + 1$
$k = 2^s r$
s=2
r=1
r is odd

fresh memory.

while r is even
//

— make_prime

```
do {
    urandomb(p, state, bits)
    p += 2^n   // offset ← (∵ urandomb range : 0 ~ 2^n - 1
} while (! prime)                 if at least 2^n.
```

rsa.c

$\downarrow +2^n$ $\downarrow +2^n$

2^n $2\cdot 2^n - 1$

↓ at least bits

relatively prime

7 & comprime

4.2 x compirme

Alice: primes p, q

$n = pq$ ⇒ very large

$\varphi(n) = (p-1)(q-1)$

$\varphi(p) = p-1$ $\varphi(q) = q-1$ $\varphi(pq) = \varphi(n)$

$e = 65537$ (chose?)

$de \equiv 1$

$d = e^{-1} \bmod \varphi(n)$

$de \equiv 1 \pmod{\varphi(n)}$

- make_pub p, q, n, e : return

nbites : # bits goes into key

need to call make prime 2 times : p & q.

rand(x, y) ⇒ rand % (y-x) + x

pbites = random % (nbits/2) + nbits/4 // get a random for $p \in [\frac{nbits}{4}, \frac{3\,nbits}{4}]$

(+ nbits/4 : offset)

make_prime x 2 ⇒ get p & q

get (p-1) & (q-1)

$\varphi(n) = (p-1)(q-1)$

do{

mpz_urandomb(e, state, nbits) // get e.

gcd(e, φ(n)) // check e & φ(n) : coprime

}while = 1 ⇒ coprime ⇒ return

- write_pub:

input file

printf(pbfile, "%Zx\n", n); (Hex number! e) (var.)

d: decimal

("%s\n", username)

- read_pub

same as pub. but input → output

printf → scanf.

- make_priv d.e.p.q return d.

  get $\varphi(n) = (p-1)(q-1)$

  mod_inverse(d, e, $\varphi(n)$) // get d.

when use make_priv in main: need to call make pub first (∵ need to use p.q.e)

- write_priv

  Same as write pub

- read_priv

  Same as write priv

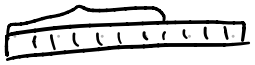
- encrypt

  $E(m) = C = m^e \% n$ pow mod

  pow_mod(C, m, e, n) //get C

- encrypt_file

1000101011000110011 ⇒ m



m<n ∴ need to %n

read 1 block / time.

bits ⇒ 1 ✓

1 → bit ✗

∵ ✗ know how many 0s in front

block size : $\left\lfloor \frac{\log_2 n - 1}{8} \right\rfloor$ use mpz_get_d_2exp OR mpz_sizeinbase cal $\log_2 n$.

// scan n bytes & convert to mpz_t ✓

how many

block ≠ 0.81

there is a $\log_2$ function!

$x\%5=2$ ✗

$7\%5=2$ ✗

$12\%5=2$

$\frac{\log_2(n)-1}{8} \leq \frac{\text{mpz\_sizeinbase}(n,2)-1}{8}$

mpz_get_ui

mpz_import(bytes) = m

r = rsa_encrypt(m)

steps:

block size

uint8_t * buff = malloc // Dynamically array

buff[0] = 0xFF ;

preperd set 1byte

while (! EOF) {

j = fread (buff+1, 1, mpz_get_ui(k-1), infile)

block size

the file need read.

how many blocks

return how many blocks.

buff+1 → need to know how many 0s in order to decrypt

mpz_import (m, j+1, 1, sizeof(buff[0]), 1, 0, buff)

return

include buff[0] ∴ need +1

size

given

the buff need to import

encrypt

printf (using %ex)

init buff

}

- decrypt

D(c) = m = $c^d \% n$ → pow_mod

from encrypt_file

- decrypt_file

get block size k (same as encrypt)

Dynamically array : buff

buff[0] = 0xFF

while (scanf() != EOF) {

decrypt

export (buff, &j, 1, size_of(buff[0]), 1, 0, m)

need to be changed

fwrite() // output into file

init buff

}

- sign

$S(m) = s = m^d \% n$ (pow_mod)

- bool verify

$V(s) = s^e \% n$ (pow_mod)

keygen.c

default:

b = 256
c = 50
pbfile = "rsa.pub"
pvfile = "rsa.priv"
seed = time(NULL)

Command line options: (include set -h & -v)

change default if have input

open file

change file to int ⇒ change permission for pvt file

↳ use fileno( )

init rand state

make pub
make priv
) make pub & priv keys.

getenv( ) ⇒ get user's name

mpz_set_str (username, 62) ↳ given num

get sign (rsa_sign)

output pub & priv keys into respective file

-v

close file
free memory

encrypt.c

default:
- pubfile = rsa.pub
- infile = stdin
- outfile = stdout
- pbfile = stdin

set username's size = _POSIX_LOGIN_NAME_MAX

command-line options. (include -h & -U)
remember to change the default when there is input through command line.

open files

rsa_read_pub to read pub from pbfile

-U

change username to mp2.

rsa_verify

encrypt_file

close files
free memory.

decrypt.c

default:
- infile: stdin
- outfile: stdout
- priufile: rsa.priv

command-line options (include -h & -U)
change default when there is inputs

open files

read priv key

-U

decrypt_file

close file
free memory.