Wichapas Pichetpongsa

Cryptography: key generator, encrypt and decrypt

Intro to the assignment:

RSA Algorithm…

RSA involves a public key and a private key.

Public key - can only be decrypted by using the private key. Integers **n** and **e** represent the public key. Private key is represented by the integer **d**.

Private key consists of the private exponent d, which must be kept secret; p, q, and (n) must also be kept secret since they are used to calculate d. In fact, the rest can be discarded after d has been computed.

We choose two large random primes p and q, publish the number n = pq

---

Your task:

They keygen will be in charge of producing RSA public and private key pairs. The encrypt program will encrypt files using public key, decrypt program will decrypt the encrypted files using corresponding private key

First file… randstate.c:

Need precision integers so we need the GNU multiple precision arithmetic library, referred to as GMP

Cannot use any GMP-implemented number theoretic functions. Must implement those functions yourself.

Void randstate_init(uint64_t seed):

Wichapas Pichetpongsa

Call gmp_randinit_mt and a call to gmp_randseed_ui

Initialize random state variable to pass it to any of the random integer functions in GMP

Number Theoretic Functions (6.1)

Given pseudo-code:

```
POWER-MOD(a, d, n)
1  v ← 1
2  p ← a
3  while d > 0
4       if ODD(d)
5            v ← (v × p) mod n
6       p ← (p × p) mod n
7       d ← ⌊d/2⌋
8  return v
```

From my understanding:

Takes in 3 arguments

Set v to 1

Set p to a

While d is greater than 0

Do this:

    If d is an odd number

        Set v to (v*p) mod n

    Set p to (p*p) mod n

    Set d to d/2

Return v

We are gonna be using this function later on. What is this for exactly? Exponentiation by squaring as shown above and reduce your results modulo n after each operation that is likely to yield a large result.

Primality Testing (6.2):

```
MILLER-RABIN(n, k)
 1  write n − 1 = 2^s r such that r is odd
 2  for i ← 1 to k
 3       choose random a ∈ {2, 3, ..., n − 2}
 4       y = POWER-MOD(a, r, n)
 5       if y ≠ 1 and y ≠ n − 1
 6            j ← 1
 7            while j ≤ s − 1 and y ≠ n − 1
 8                 y ← POWER-MOD(y, 2, n)
 9                 if y == 1
10                      return FALSE
11                 j ← j + 1
12            if y ≠ n − 1
13                 return FALSE
14  return TRUE
```

We are supposed to implement→

Void is_prime(mpz_t n, uint64_t iters)

Use the pseudo-code above, pass in iters to miller rabin to see if number n is prime

function is needed when creating the two large primes p and q in RSA, verifying if a large integer is a prime.

I should get these down first so I can implement it in other functions (Where I start my code).

Wichapas Pichetpongsa

Number theoric functions should be fine as long as I follow the instructions

---

RSA Library:
(need to nail this because this is quite important for the main files)

Rsa_make_pub():

Need to generate random p and q
Also make sure that they are in range such that nbits/4 and 3*nbits/4
Remaining q_bits is obtained from nbits - p
random(p)
random(q)

While true:

Totient = (p-1)(q-1)

Now find suitable public exponent

random(e)
gcd(random_variable, temp, totient)
if (random_variable == 1):

Break

Kind of just following the instructions, nothing crazy. However, need to be careful about the range for bits

rsa_encrypt_file():

Create the size of the block_k = (log_2(n)-1)/8

Allocate memory block = malloc ___

Block[0] = 0xFF

while(read(file,block+1,block_k-1) != 0):

import(m<-block)

Encrypt it

Print out the c

Pretty much following the instructions as well but have to make sure during read that you don't read the 0xFF and the size does not go too far because we had the 0xFF

Main Files:

Follow the instructions given from the assignment document

However,

Infile should be using fopen and set to "r"

For outfile however, fopen with "w+" if file doesn't exist it will create it

First complete the randstate.c (easiest)

Next, complete numtheory.c (fully supported pseudo code)

Next, complete rsa.c, totally depends on the numtheory.c to work

Mainfiles are just basically a combination of rsa.c and all the other .c files

FILES TO SUBMIT:

Decrypt.c

Encrypt.c

Keygen.c

Numtheory.c

Numtheory.h

Randstate.c

Randstate.h

Rsa.c

Rsa.h

README.md

DESIGN.pdf

MAKEFILE

Wichapas Pichetpongsa