Assignment 5 Public Key Cryptography

Katrina VanArsdale

February 28, 2023

1 Description of Program

This program generates a public key and a private key pair, encrypts files using a public key and decrypts encrypted files using the corresponding private key.

2 Files In Directory

- decrypt.c
  This contains the implementation and main() for the decrypt program
- encrypt.c
  This contains the implementation and main() for the encrypt program
- keygen.c
  This contains the implementation and main() for the keygen program
- numtheory.c
  This contains the implementations of the number theory functions
- numtheory.h
  This contains the interface for the number theory functions
- randstate.c
  This contains the implementation of the random state interface for the SS library and number theory functions
- randstate.h
  This contains the interface for initialixing and clearing the random state.
- ss.c
  This contains the implementation of the SS library
- ss.h
  This contains the interface for the SS library
- Makefile
  This file compiles all of the files and creates .o files for every .c file. It also cleans up all those files afterward and can clang format them. It can make each executable separately and uses pkg-config for the GMP library.
- README.md
  This markdown file will describe how to use my program and Makefile. It also lists and explains the command line options that my program accepts.
- WRITEUP.pdf
  This file will include things I learned about encryption and lessons I learned while working on the assignment
- DESIGN.pdf
  This file describes the design for this program with pseudo code. This is the file you are reading.

3 Pseudo Code

3.1 Random State

- Define global variable state
- Define functions
- randstate_init
  - Takes in a seed
  - Call srandom with provided seed
  - Call gmp_randinit_mt() and gmp_randseed_ui()
- randstate_clear
  - Clears and frees memory used for state
  - Simply calls gmp_randclear()

3.2 Number Theory Functions

- Define functions
- pow_mod
  - Takes in: o, a, d, n
  - a d and n are constants so we can't change them
  - base = a
  - exp = d
  - o = 1
  - While exp greater than 0
  - – if exponent is odd
  - — o = o * base mod modulus
  - – base = base * base mod modulus
  - – exp = exp / 2 floored
  - returns o
- is_prime
  - referenced from primes.py in the resources file
  - Takes in n and iters
  - if n less than 2 or (n not 2 and n % 2 == 0)
  - – return false
  - if n == 2 or n == 3
  - – return true
  - r = n-1
  - s = 0
  - while r is even
  - – r = r // 2
  - – s += 1
  - for loop in range 1 to iters
  - – a = random number from 2 to n-2

  - – y = pow_mod(a, r, n)
  - – if y is not 1 and not n - 1
  - —- j = 1
  - —- while j is less than s-1 and y is not n-1
  - —— y = pow_mod(y, 2, n)
  - —— if y = 1
  - ——— return false
  - —— j += 1
  - – if y not = to n-1 return false
  - return true

- make_prime
  - Takes in p, bits, iters
  - lower bound is 2 to the power of (bits-1)
  - upper bound is (2 to the power of bits)-1
  - while true
  - – p = random number between lower and uper bound
  - – if p mod 2 == 0
  - —- p += 1
  - – Tests p with is_prime(p, iters)
  - – returns p if it is a prime

- gcd
  - Takes in d, a, b
  - While b is not 0
  - –Define variable t equal to b
  - –b = a mod b
  - –a = t
  - –d = a
  - return

- mod_inverse
  - Takes in i, a, n
  - r = n
  - r’= a
  - t = 0
  - t’ = 1
  - i = 1
  - while r’ not 0
  - – q = r divided by r’ floored
  - – temp = r
  - – r = r’
  - – r’ = temp - q * r’
  - – temp = t
  - – t = t’

  - – t’ = temp - q * t’
  - i = t
  - if r greater than 1
  - – return no inverse
  - if t less than 0
  - – i = t + n
  - returns i

3.3 SS Library

- Define functions
- ss_make_pub
  - Generates components for new SS private key
  - Takes in p, q, n, nbits, iters
  - pbits = random number from [nbits/5, (2 * nbits)/5)
  - p = make_prime(p, pbits, iters)
  - qbits = pbits * 2
  - qbits = nbits - qbits
  - add 1 to pbits and qbits to make n have at least nbits
  - q = make_prime(q, qbits, iters)
  - while true
  - – d1 = gcd(p, q-1)
  - – d2 = gcd(q, p-1)
  - – if d1 == p or d2 == q
  - —- p = make_prime(p, pbits, iters)
  - —- q = make_prime(q, qbits, iters)
  - –else break
  - n = p * p * q
  - return
- ss_write_pub
  - print n into pbfile as a hextring followed by a newline
  - Find the length of username
  - user = alloc space for username
  - copy username into user
  - print user into pbfile followed by a newline
- ss_read_pub
  - fscanf pbfile
  - reads the ss key from pbfile
- ss_make_priv
  - Takes in p and q
  - d = (p - 1)(q -1) / gcd(p-1,q-1)
  - n = p * p * q

  - d = mod_inverse(n, d);
  - return

- ss_write_priv
  - fprint private ss key into pvfile
  - first p * q newline
  - then d newline
  - both should be hextrings

- ss_read_priv
  - fscanf pvfile to find pq and d

- ss_encrypt
  - pow_mod(c, m, n, n)

- ss_encrypt_file
  - Encryptes the contents of infile
  - Block size k = (log2(sqr(n)) -1)/8 floored
  - kbytes = Dynamically allocate an array for k bytes with sizeof uint8_t *
  - Set zeroth byte of block to 0xFF
  - While there are unprocessed bytes in infile
  - – set j to number of read bytes
  - – place j bytes from kbytes into m with mpz_import
  - – ss_encrypt(c, m, n)
  - – fprint c as a hextring into outfile with a newline
  - – clear array and set first bytes to 0xFF
  - free allocated space

- ss_decrypt
  - pow_mod(m, c, d, pq)

- ss_decrypt_file
  - Decrypts the contents of infile
  - Block size k = (log2(pq) -1)/8 floored
  - kbytes = Dynamically allocate an array for k bytes with sizeof uint8_t *
  - While there is still lines to decrypt
  - – scan in hexstring from in file as mpz_t c
  - – ss_decrypt(m, c, d, pq)
  - – Using mpz_export() convert m into bytes and put it into the allocated block and put the amount of read bytes into j
  - – Print 1 to j-1 bytes of the block into outfile
  - free allocated space

3.4 Key Generator

- Options bin:d:svh
- create a function that prints the usage
- Use getopt to parse options
- fopen public and private files
- Use fchmod and fileno to set person to user only with S_IRUSR — S_IWUSR
- randstate_init with default or provided seed
- make pub and make priv
- Use getenv for users name
- write pub and write priv
- If -v is enabled print: username, p, q, n, d, pq with bits
- Close files
- randstate_clear, clear mpz_t variables

3.5 Encrypt

- Options i:o:nvh
- create a function that prints the usage
- Use getopt to parse options
- fopen public file input file and output file
- read public key
- If -v is enabled print: username, public key n with bits
- encrypt file
- Close files
- clear mpz_t variables

3.6 Decrypt

- Options i:o:nvh
- create a function that prints the usage
- Use getopt to parse options
- fopen private file input file and output file
- read public key
- If -v is enabled print: username, private key d
- decrypt file
- Close files
- clear mpz_t variables

4 Credits

- Pseudo code is referenced from the asgn5.pdf
- I looked at many of the gmplib.org manual pages to learn about the gmp functions
- I referenced the mpz.pdf slides shown in class to see which functions to use
- I referenced the primes.py code in the resources to write the pseudo code for is_prime
- I looked through the discord to see if anyone else had the same problems as me and asked some questions
- I went to Jessie's tutoring zoom to ask questions about my is prime function
- I learned how to use fread through https://www.tutorialspoint.com/c_standard_library/c_function_fread.htm