Assignment 5

Public Key Cryptography
DESIGN Document

Description of the program

In the design document, I will be focusing on the implementation of the cryptographic algorithm, to secure the passage of information between two parties.

What is Cryprography:
In Computer Science, Cryptography refers to a study of securing information, through mathematical formulas, which would result in secure communication between two sources. Cryptography makes it so that only the sender and the receiver are able to view the information, to other parties it would be encrypted, therefore it would look different than the original message.

Files to be included

Source and header files:

1. **decrypt.c** contains the implementation and **main()** function for the *decrypt* program
2. **encrypt.c** contains the implementation and **main()** function for the *encrypt* program
3. **keygen.c :** contains the implementation and **main()** function for the *keygen* program
4. **numtheory.c**: specifies the implementation for number theory functions
5. **numtheory.h**: interface for number theory functions
6. **randstate.c:** Contains the implementation of the random state interface for the RSA library and number theory functions
7. **randstate.h:** Specifies the interface for initializing and clearing random state

8. **rsa.c**: Implementation of the RSA library
9. **rsa.h**: Specifies the RSA library.

Additional Files:

1. **Makefile:** formats all source code, including the header files.
2. **README.md:** Description of how to use my program and Makefile. It also includes any command-line option that my program accepts. Any false positives reported by scan-build should be documented and explained here as well. Note down any known bugs or errors in this file as well for the graders.
3. **DESIGN.pdf (*This file*):** The design document describes the preliminary design and design process for my program with sufficient detail for potential replication
4. **WRITEUP.pdf:** Analysis and description of the produced program, as well as graphs generated and/or information, gathered from outputs

Pseudocode/Structure:

---

randstate.c:

randstate_init(uint64_t seed):

Initialize the global random state though the Mersenne Twister algorithm.

randstate_clear(void):

Clears and frees all memory used by global random state name state

numtheory.c:

pow_mod(mpz_t out, mpz_t base, mpz_t exponent, mpz_t modulus):

Performs fast modular exponentiation

is_prime(mpz_t n, uint64_t iters):

Conducts Miller-Rabin primality test to indicate whether or not input *n* is prime user *inters* number of Miller-Rabin iterations.

make_prime(mpz_t p, uint64_t bits, uint64_t iters):

Generates a new prime number stored in p that will be at least *"bits"* number of bits long.

gcd(mpz_t d, mpz_t a, mpz_t b):

Computes the greatest common divisor of a and b

mod_inverse(mpz_t i, mpz_t a, mpz_t n):

computes the inverse i of module n

rsa.c

void rsa_make_pub(mpz_t p, mpz_t q, mpz_t n, mpz_t e, uint64_t nbits, uint64_t iters):

Creates a new RSA public key

rsa_write_pub(mpz_t n, mpz_t e, mpz_t s, char username[], FILE *pbfile):

Writes a public RSA key to a *pbfile*

rsa_read_pub(mpz_t n, mpz_t e, mpz_t s, char username[], FILE *pbfile):

Reads a public RSA key from *pbfile*

rsa_make_priv(mpz_t d, mpz_t e, mpz_t p, mpz_t q):

Creates a new RSA private key d given primes p and q and public exponent e

rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile):

Writes a private RSA key to *pvfile*

rsa_read_priv(mpz_t n, mpz_t d, FILE *pvfile):

Reads a private RSA key from *pvfile*

rsa_encrypt(mpz_t c, mpz_t m, mpz_t e, mpz_t n):

Preforms RSA encryption

rsa_encrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t e):

Encrypts the contents of infile

rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n):

Performs RSA decryption, computing message *m* by decrypting ciphertext c using private key d and public modulus *n*.

rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d):

Decrypts the contents of *infile*, writing the decrypted contents to *outfile*

rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n):

Performs RSA signing, producing signature s by using message m using private key d and public modulus n

rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n):

Performs RSA verification, returning true if signature s is verified, false otherwise.

key-gen.c - command-line options:

-b : specifies the minimum bits needed for the public modulus n.

-i [iterations]: specifies the number of Miller-Rabin iterations for testing primes (default: 50)

-n [pbfile]: Specifies the public key file (default: rsa.pub)

-d [pvfile]: Specifies the private key file (default: rsa.priv)

-s: Specifies the random seed for the random state initialization (default: the seconds since the UNIX epoch, given by time(NULL)).

-v: enables verbose output.

-h: display program synopsis and usage

<u>encrypt.c - command-line options:</u>

-i: specifies the input file to encrypt (default: stdin)

-o: specifies the output file to encrypt (default: stdout)

-n: specifies the file containing the public key (default: rsa.pub)

-v: enables verbose output

-h: display program synopsis and usage

<u>decrypt.c - command-line options:</u>

-i: specifies the input file to decrypt (default: stdin)

-o: specifies the output file to decrypt (default: stdout)

-n: specifies the file containing the private key (default: rsa.priv)

-v: enables verbose output

-h: display program synopsis and usage

Credit

- I took the pseudocode from the assignment 5 paper written by prof. Darrel Long