DESIGN DOCUMENT
PUBLIC KEY CRYPTOGRAPHY

randstate.c

GMP initializes a random state variable and passes it into any random GMP function. Along with that it should also clear any memory used by the initialized random state.

1. void randstate_init(uint64_t seed);
    a. Calls gmp_randinit_mt() and a call to gmp_randseed_ui()
2. void randstate_clear(void);
    a. Calls gmp_randclear()

numtheory.c

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

This function performs a fast modular exponentiation.

bool is_prime(mpz_t n, uint64_t iters)

This function takes in the parameter n and returns true if n is prime and returns false if n is not prime. For this function we use the highly efficient miller-rabin function.

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

This generates a prime number and checks if it's prime using the is_prime function. The number that is generated from make_prime should be at least a number of bits long.

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

This function computes the great common divisor of a and b.

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

This function computes the inverse i on the modular 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)

This function creates a new RSA public key. This function is important because it's used in the keygen.c when trying to create a public key for the use.

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

This function writes a public RSA key to pbfile. The values should be printed as a hex string

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

Reads a public RSA key from pbfile. They should be read as hex string.

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

Writes a private RSA key to pvfile.

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

Reads a private RSA key from pvfile.

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

Performs RSA encryption. It encrypts the message m using e and mod n.

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

This encrypts the contents of the infile, The data in the infile should be encrypted in blocks. So first we want to calculate the block size k then dynamically allocate memory for k, set the zeroth bit to 0xFF. While some bits are unprocessed. Read the utmost k-1 bytes. Then use mpz_import. Then encrypt using rsa_encypt.
void rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n)
Performs RSA decryption. Computing m using d and the public modulus n.
void rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d)
Decrypts the contents of the infile. First it calculates the block size k. Then dynamically allocates an array for k number of bytes. While there are unprocessed files we will scan the hexstring. Call mpz_export. And then write out j -1 bytes.

void rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n)
Performs RSA signing.

bool rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n)
Performs RSA verification. This checks if the signature s is verified or not. If it is verified then you return true or else you return false.

keygen.c
This function creates a public and private key. To do that first we have to make command line options based on what the user gives us. First we have to open the public and private key using fopen and then set the permissions of the file using fchmod and fileno to 0600. After that we must initialize the randstad_init so that the state is initialized. Then we call rsa_make_pub and rsa_make_priv to generate our public and private keys. Then we get the username from the user. Store the user input and convert it into an mpz. Then write the computed public and private keys to their files and close and free all the files and mpz's that you used.

Encrypt.c
First we do the command line options. Use fopen() to open the public key file. Then we have to read the public key from the opened public key file. Then print out the verbose if it's true. Convert the username to an mpz_t and then verify the signature using rsa_verify(). Now encrypt a file using rsa_encrypt_file(). Then close all the files and free all the memory.

Decrypt.c
First do the command line options. Use fopen to open the private key. Read the private key from the opened file. Print out the stats if verbose is true. Decrypt the file using rsa_decypt_file. Then close the files and free all the memory.