Shaan Mistry

Assignment 5: DESIGN.pdf

Description of Program:
This program uses an asymmetric cryptography which generates public private key pairs. The public keys are known by everyone while the private keys are only known by the owner. These keys are generated using the RSA algorithm which relies on the difficulty of factoring the product of large prime numbers.

Users can generate and store their public and private keys into designated files. In addition, they can use the generated keys to encrypt and decrypt files. Anyone can encrypt a message using the intended receiver's public key; however, the encrypted message can only be decrypted with the receiver's private key.

Files to be include in "asgn5":

1. decrypt.c
    - Parses command line options using getopt.
    - Inputs a file to be encrypted and output file to store the encryption.
    - Opens private key.
2. encrypt.c
    - Parses command line options using getopt.
    - Inputs a file to be encrypted and output file to store the encryption.
    - Opens public key.
3. keygen.c
    - Parses command line options using getopt.
    - Stores generated public key in a file.
    - Securely stores generated private key in a file.
    - Initializes the random state.
4. numtheory.c
    - Contains math functions to aid in the computation of the mathematics of RSA.
5. numtheory.h
    - Interface for the number theory functions.
6. randstate.c
    - Implements RSA library
    - Contains global randstate.
7. randstate.h
    - Interface for the RSA library.
8. rsa.c
    - Generates, stores, and reads public and private keys.
    - Encrypts and Decrypts files.
    - Performs RSA signing and verification.
9. rsa.h
    - Interface for the RSA library.
10. Makefile

- Directs compilation processes for all **C** files.
- Formats .c and .h files.
- Cleans object files and executables.

11. README.md
    - Markdown Format.
    - Contains information on how to build the program, how to run the program, and how to clean up and object files and binaries created from building the program. It can also contain information on possible bugs or errors in the program.
12. DESIGN.pdf
    - This file.
    - Contains pseudocode and explanations of the program.

Pseudocode / Structure:

ranstate.c

```
Create a global random state called state.
FUNCTION: randstate_init:
    PARAMETERS: seed
    Initialize the random state using inputted seed.
    Call srandom() with seed.

FUNCTION: randstate_clear:
    Free all memory occupied by state.
```

numtheory.c:

This file will be implemented using the provided pseudocode in the asgn5.pdf. At first, the functions will be created using 32-bit unsigned integers instead of mpz_t objects. After the functions work as intended with 32-bit integers, they will be converted to mpz_t objects from the GMP library.

rsa.c:

```
FUNCTION: rsa_make_pub:
    Create mpz_t out.
    Call make_prime using p and q, and store in out.
    Set number of bits for p to random number in range [nbits/4, (3×nbits)/4).
    Set number of bits for q to nbits – number of bits for p.
    Set number of Miller_Rabin iterations to iters.
    Call make_prime for both p and q using their respective number of bits and iters.
    Set n to the product of p and q.
    Calculate the greatest common denominator of p – 1 and q – 1 using gcd().
    Set lambda n to lcm(p – 1, q – 1) to the abs(p – 1, q – 1) / gdc(p – 1, q – 1).

    WHILE (true):
        Generate random_e of nbits using mpz_urandomb().
        IF gcd(random_e , lambda_n) == 1:
            free all memory used by mpz_t objects.
            Set e to random_e.
            Return.

FUNCTION: rsa_write_pub:
    FPRINT:
        n as a hexstring using gmp_prinf().
        e as a hexstring using gmp_prinf().
        s as a hexstring using gmp_prinf().
        username

FUNCTION: rsa_read_pub:
    Set n to fscanf() of first line.
    Set e to fscanf() of second line.
    Set s to fscanf() of third line.
    Set username to fscanf() of fourth line.

FUNCTION: rsa_make_priv:
    Set lambda n to lcm(p – 1, q – 1) to the abs(p – 1, q – 1) / gdc(p – 1, q – 1).
    Set d to mod_inverse(e, mod(lambda_n)).

FUNCTION: rsa_write_priv:
    FPRINT:
        n as a hexstring using gmp_prinf().
        d as a hexstring using gmp_prinf().

FUNCTION: rsa_read_priv:
    Set n to fscanf() of first line.
    Set d to fscanf() of second line.

FUNCTION: rsa_encrypt:
    Set c to pow_mod(m, e, n).

FUNCTION: rsa_encrypt_file:
    Set block size (k) to the floor value of $(\log_2(n)-1) / 8$.
    Allocate memory for an array of k bytes.
    Set the 0th index of the array to 0xFF.

    WHILE the EOF has not been reached :
        Set j to the number of bytes read from the block  with the function fread().
        IF j is greater than 0:
            Call mpz_import() to convert the read bytes.
            Call rsa_encrypt() to encrypt c.
            Use gmp_fprinf() to write c to the output file as a hexstring.
    Clear mpz_t variables.
    Free the memory allocated for the block.


FUNCTION: rsa_decrypt:
    Set m to pow_mod(c, d, n).

FUNCTION: rsa_decrypt_file:
    Set block size (k) to the floor value of ($\log_2$(n)-1) / 8.
    Allocate memory for an array of k bytes.
    Set the 0th index of the array to 0xFF.
    WHILE gmp_scanf() hasn't reached the end of file:
        Store the hexstring line scanned by gmp_scanf().
        Call rsa_decrypt to decrypt the hexstring line.
        Call mpz_export() to convert the hexstring to bytes and store them in the array.
        IF j is greater than 1:
            Use gmp_fprinf() to write j-1 bytes, excluding the prepended 0xFF.
    Clear mpz_t variables
    Free the memory allocated for the block.
FUNCTION: rsa_sign:
    Set s to pow_mod(m, d, n).

FUNCTION: rsa_verify:
    If pow_mod(s, e, n) equals the expected message:
        Return true.
    Return false.
```

keygen.c:

```
WHIILE: Get opt finds command line options:
    Case b: set bits to getopt argument.
    Case i set iterations to getopt argument.
    Case n: set file that contains public key to getopt argument.
    Case d: set file that contains private key to getopt argument.
    Case s: set seed to getopt argument.
    Case v: set verbose output to true.
    Case h: display program synopsis and usage.

Use fopen() to open the public and private key files.
Set the file descriptor to return value of fileno() with the private key file.
    Use fchmod() to set the file descriptor of the private key file to 0600.

Call randstate_init() using the specified seed.
Create public key using rsa_make_pub().
Create private key using rsa_make_priv().
Set username to the value from calling getenv().
Set username_string to mpz_set_str() using username and base 62.
Call rsa_sign with username_string.
Write the public key into the public key file with rsa_write_pub.
Write the private key into the private key file with rsa_write_priv.
IF verbose == true:
        PRINT: Using gmp_printf().
                Username.
                The signature s.
                The first large prime p.
                The second large prime q.
                The public modulus n.
                The public exponent e.
                The private key d.

Clear the random state using randstate_clear().
Close public and private key files and free all memory used by mpz_t objects.
```

encrypt.c:

```
WHIILE: Get opt finds command line options:
        Case i: set input file to getopt argument.
        Case o: set output file to getopt argument.
        Case n: set file that contains public key to getopt argument.
        Case v: set verbose output to true.
        Case h: display program synopsis and usage.

Open the input file with fopen().

Call rsa_read_pub() to read the public key from the public key file.

IF verbose output == true:
        PRINT using the values form reading the public key file:
                Username.
                The signature s.
                The public modulus n.
                The public exponent e.

Set expected value = mpz_t of username.
Call rsa_verify() with expected value to verify signature.
Call rsa_encrypt_file() to encrypt the output file.
Close public key file and free all memory used by mpz_t objects.
```

decrypt.c:

```
WHIILE: Get opt finds command line options:
        Case i: set input file to getopt argument.
        Case o: set output file to getopt argument.
        Case n: set file that contains private key to getopt argument.
        Case v: set verbose output to true.
        Case h: display program synopsis and usage.

Open the private key file with fopen().
Call rsa_read_priv() to read the private key from the private key file.

IF verbose output == true:
        PRINT using the values form reading the private key file:
                The public modulus n.
                The private key e.

Call rsa_decrypt_file() to decrypt the output file.
Close private key file and free all memory used by mpz_t objects.
```

Error Handling:
If the input files for the public or private keys are not valid, print an error and exit the program.
If the signature cannot be verified, report an error, and exit the program.