Assignment 6 - Public Key Cryptography

Sean Carlyle

November 11 2021

1 Introduction

This is my design document for Assignment 6 on public key cryptography. It is made up of 3 binary files called encrypt, decrypt, and keygen that encrypt files, decrypt files and create a private and public key respectively. There are also helper files rsa.c, numtheory.c, randstate.c that hold number theory functions, randstate helps to initialize and clear the randstate, and rsa holds the RSA library. These functions work together to make this encryption possible. The structure of this program is composed like this: the number theory functions are used in RSA to create functions to encrypt/decrypt and create keys. Then, RSA is used in encrypt, decrypt, and keygen to allow them to actually decrypt/encrypt and create keys.

2 Pseudocode with Explanation

2.1 keygen.c

keygen is one of the executables in the program. It creates two private keys and writes them to the pvfile, and pbfile respectively.

2.1.1 keygen.c pseudocode

```
Parse command line for arguments and put them into effect if needed
fopen() to open private and public keys
Use fchmod() and fileno() to make sure file permissions are correct
Initialize randstate with randstate_init() to set state seed
Make public and private key with rsa_make_pub() and rsa_make_priv()
Get user name as string using getenv()
convert username to mpz_t with mpz_set_str() with base 62, use rsa_sign() to compute signature
Write public and private key to respective files
Print verbose statements
Close public and private key files, clear mpz variables
```

2.2 encrypt.c

encrypt is one of the executables in the program. It encrypts an infile into ciphertext and writes it to the outfile.

2.2.1 encrypt.c pseudocode

```
Parse command line for arguments and put them into effect if needed
Use fopen() to open public key file
Read the public key file using rsa_read_pub
Print out verbose statements if needed
Test the signature with rsa_verify()
Encrypt the file with rsa_encrypt_file(), which also writes to outfile
close the public key and free mpz used
```

2.3 decrypt.c

decrypt is one of the executables in the program. It decrypts an infile from ciphertext to normal text and writes it to outfile.

2.3.1 decrypt.c pseudocode

```
Parse command line for arguments and put them into effect if needed
Use fopen() to open the private key file
Read in the private key
Call rsa_decrypt_file to decrypt the file and write to outfile
Print out verbose statements if needed
Close file and mpz's used
```

2.4 numtheory.c

Numtheory is the file that holds all of our function that compute things like primes, power_mods, inverses and more. It helps in RSA to encrypt/decrypt files and make primes for private/public keys.

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

Pow_mod takes the a number to a power and then takes the modulus of it. We do this by repeating squaring and modular reduction at each step.

2.4.2 pow_mod pseudocode

```
Set out to 1
Set p to base
```

```
while exponent is greater than 0
    if exponent is odd
        set out out to out * p mod n
    set p to p * p mod n
    d equals floor(d divided by 2)
```

2.4.3 bool is_prime(mpz_t n, uint64_t iters)

is_prime uses the miller-rabin primality test to determine whether a number is prime. The test can sometimes fail so we must do it over many iterations to make sure our numbers are actually prime.

2.4.4 is_prime pseudocode

```
find s and r such that n-1 = 2 to the s times r
    Choose a random number 2,3,n-2
    y = powermod(a,r,n)
    if y != 1 and y != n - 1
        j = 1
        while ¡= s-1 and y != n-1
            y = powermod(y,2,n)
            y == 1
                return false
            j = j + 1
        if y != n - 1
            return false
return true
```

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

This function creates a prime number of at least bits length and conducts miller-rabin with iters number of iterations.

2.4.6 make_prime pseudocode

```
Make random number between 0 and bits + 1
Find 2 to the bits + 1
If random number is less than 2 to the bits + 1
Multiply it by 2 to the bits + 1 so it is greater than minimum bits
Keep doing this until it passes is_prime()
```

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

gcd finds the greatest common denominator of two numbers.

2.4.8 gcd pseudocode

```
while b!= 0
    t = b
    b = a mod b
    a = t
return a
```

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

Finds the inverse of a mod n and returns it in i

2.4.10 mod_inverse pseudocode

```
r, r’ = n, a
t, t’ = 0, 1
while r’ != 0
    q = floor (r / r’)
    r, r’ = r’, r - q*r
    t, t’ = t’, t - q*t
r is greater than 1
    return no inverse
if t is less than 0
    t = t + n
return t
```

2.5 randstate.c

This file is just used to initiate the random state needed to find random numbers.

2.5.1 void rand_state_init(uint64_t seed)

Call gmp_randinit_mt() and gmp_randseed_ui()

2.5.2 void randstate_clear(void)

Call gmp_randclear()

2.6 rsa.c

2.6.1 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 the public key which consists of two primes, their product n, and an exponent.

2.6.2 rsa_make_pub pseudocode

```
Use make_prime() to make p and q
Let number of bits for p be random number between nbits/4 and 3xnbits/4
Let rest of bits be for q
Compute totient
Generate numbers using mpz_urandbomb()
Compute gcd of random numbers and totient
Stop if found a coprime, this will be public key
```

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

This function writes public file to pbfile

2.6.4 rsa_write_pub pseudocode

```
Simply print out each mpz and username to pbfile
```

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

This function reads in public file to variables.

2.6.6 rsa_read_pub pseudocode

```
Call fgets and sscanf each line into the mpz_t variables
sscanf the username into a char array
```

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

This function creates the private key.

2.6.8 rsa_make_priv pseudocode

```
Compute private key by computing the inverse of e mod (p-1)(q-1).
```

2.6.9 void rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile)

This function writes the private key to the pvfile.

2.6.10 rsa_write_priv pseudocode

```
Simply gmp print n and d to the pvfile.
```

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

This function reads the private key from pvfile into variables.

2.6.12 rsa_read_priv pseudocode

Simply fgets and sscanf in the variables.

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

This function encrypts a message into cyphertext

2.6.14 rsa_encrypt pseudocode

Find m to the e mod n and store in c.

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

This function encrypts an entire file into cyphertext. It outputs into outfile.

2.6.16 rsa_encrypt_file pseudocode

```
Calculate block size
Allocate array of k bytes
Set zero byte of block to 0xFF
while not all bytes read
    Place read bytes in block starting at index 1
    If we ever read 0 bytes just return
    Convert bytes to mpz
    Encrypt with rsa_encrypt()
    Print to outfile
```

2.6.17 void rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n)

This function decrypts ciphertext into a message

2.6.18 rsa_decrypt pseudocode

Find c to the d mod n

2.6.19 void rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d)

This function decrypts an infile into normal text and outputs to outfile.

2.6.20 rsa_decrypt pseudocode

```
Calculate block size
Allocate array of k bytes
while not reaching end of file
    Scan in hexstring
    Decrypt the hexstring
    Convert mpz_t in array message
    If the array message is ever 0, return
    Write out j -1 bytes from index 1 to outfile
```

2.6.21 void rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n)

This function produces a signature that we can verify later.

2.6.22 rsa_sign pseudocode

```
Find m to the d mod n and store in s.
```

2.6.23 bool rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n)

This function verifies our RSA verification by checking it the signatures are the same

2.6.24 rsa_verify pseudocode

```
Compute s to the e mod n.
If this is the same as m return true
Otherwise return false
```

3 Example of how the program runs

An example of how the program would run would look like this. We use keygen to create a public and private key which are written to two files. Then from there we encrypt the infile using the public key, having to read the public key to get it and using the key to encrypt the whole file. We also have to verify that the signature is correct from the public file. Finally to decrypt the file, we read in the private key which allows us to decrypt the entire file. This is the general way the program should be ran to encrypt/decrypt files.