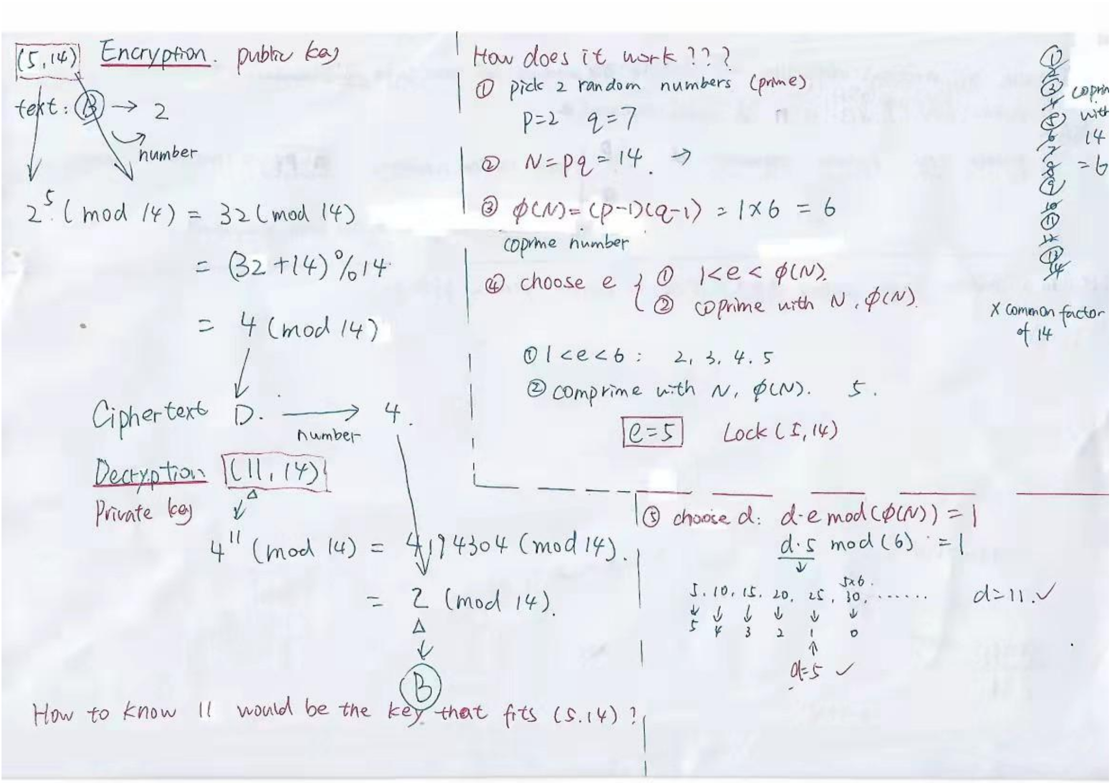
Assignment 5 DESIGN.pdf

Ning Jiang

Description of Program: For this assignment, we mainly need to Encrypt and decrypt files using RSA. The keygen program will be in charge of key generation, producing RSA public and private key pairs.The encrypt program will encrypt files using a public key, and the decrypt program will decrypt the encrypted files using the corresponding private key.

Basic rules of RSA:



Files to be included in directory “asgn5”:

1. decrypt.c: This contains the implementation and main() function for the decrypt program.
2. encrypt.c: This contains the implementation and main() function for the encrypt program.
3. keygen.c: This contains the implementation and main() function for the keygen program.
4. numtheory.c: This contains the implementations of the number theory functions.
5. numtheory.h: This specifies the interface for the number theory

functions.
6. randstate.c: This contains the implementation of the random state interface for the RSA library
and number theory functions.
7. randstate.h: This specifies the interface for initializing and clearing the random state.
8. rsa.c: This contains the implementation of the RSA library.
9. rsa.h: This specifies the interface for the RSA library.
10. Makefile
11. README.md
12. DESIGN.pdf

Structure & explanation / Pseudocode:

We first impliment two libraries and an random state module that will be used in keygen, encrypt and decrypt. The pseudocode will be peovided in the Pseudocode part.

Explanations for some of the equations are cited from asgn5.pdf.

randstate.c:
Global variable state

void randstate_init(uint64_t seed):
gmp_randinit_mt() and to gmp_randseed_ui().

Initializes the global random state named state.

void randstate_clear(void):
Clears and frees all memory used by the initialized global random state named state.

numtheory.c:

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

```
POWER-MOD(a, d, n)
v ← 1
p ← a
while d > 0
    if ODD(d)
        v ← (v × p) mod n
    p ← (p × p) mod n
    d ← ⌊d/2⌋
return v
```

Performs fast modular exponentiation, computing base raised to the exponent power modulo modulus, and storing the computed result in out.

bool is_prime(mpz_t n, uint64_t iters):

```
MILLER-RABIN(n, k)
write n − 1 = 2^s r such that r is odd
for i ← 1 to k
    choose random a ∈ {2,3,...,n−2}
    y = POWER-MOD(a, r, n)
    if y ≠ 1 and y ≠ n − 1
        j ← 1
        while j ≤ s − 1 and y ≠ n − 1
            y ← POWER-MOD(y, 2, n)
            if y == 1
                return FALSE
            j ← j + 1
        if y ≠ n − 1
            return FALSE
return TRUE
```

indicate whether or not n is prime.

void make_prime(mpz_t p, uint64_t bits, uint64_t iters):
Generates a new prime number stored in p. The generated prime should be at least bits number of bits long.

1. pick a random number which is at least nbits long.
2. Use is_prime to check whether it is prime or not.
3. If yes, return. Else, pick another random number.

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

```
GCD(a, b)
while b ≠ 0
    t ← b
    b ← a mod b
    a ← t
return a
```

Computes the greatest common divisor of a and b, storing the value of the computed divisor in d.

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

```
MOD-INVERSE(a, n)
(r, r′) ← (n, a)
(t, t′) ← (0, 1)
while r′ ≠ 0
    q ← ⌊r/r′⌋
    (r, r′) ← (r′, r − q × r′)
    (t, t′) ← (t′, t − q × t′)
if r > 1
    return no inverse
if t < 0
    t ← t + n
return t
```

Computes the inverse i of a modulo 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):

```
Create a random bits
make_prime(p)
make_primeq(q)
λ(n) = lcm(p−1,q−1) = abs((p-1)(q-1))/gcd(p−1,q−1)
While true:
   Create a random number
   Check whether it is coprime with λ(n)
   If true, return.
```

Creates parts of a new RSA public key: two large primes p and q, their product n, and the public exponent

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

```
If the file is valid:
   Print n, e, s, username to the pbfile.
```

Writes a public RSA key to pbfile.

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

```
If the file is valid:
   Read n, e, s, username from the pbfile.
```

Reads a public RSA key from pbfile.

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

λ(n) = lcm(p-1,q-1) = abs((p-1)(q-1))/gcd(p-1,q-1)
mod_inverse(e, l)

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

void rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile):
If the file is valid:
   Print n, d, username to the pvfile.

Writes a private RSA key to pvfile.

void rsa_read_priv(mpz_t n, mpz_t d, FILE *pvfile):
If the file is valid:
   Read n, d, username from the pbfile.

Reads a private RSA key from pvfile.

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

computing ciphertext c by encrypting message m using public exponent e and modulus n.

void rsa_encrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t e):
block size k= floor((log2(n)-1)/8)
Dynamically allocate an array: (arr) that can hold kbytes.
Arr[0] = 0xFF
While there are still unprocessed bytes in infile // use fread()
   mpz_import(), convert the read bytes, including the prepended 0xFF into an mpz_t m. 1 for most significant word first, 1 for the endian parameter, and 0 for the nails parameter.
   Encrypt m with rsa_encrypt(),
   encrypted number to outfile as a hexstring

Encrypts the contents of infile, writing the encrypted contents to outfile.

void rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n):
M = c^d (mod n).

computing message m by decrypting ciphertext c using private key d and public modulus n.

void rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d):
block size k= floor((log2(n)-1)/8)
Dynamically allocate an array: (arr) that can hold kbytes.
While here are still unprocessed bytes in infile // gmp_fscanf()
    Using mpz_export(), convert c back into bytes, storing them in the allocated block. 1 for most significant word first, 1 for the endian parameter, and 0 for the nails parameter.
    Write out j-1 bytes starting from index 1 of the block to outfile.

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

void rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n):
s= m^d (mod n)

producing signature s by signing message m using private key d and public modulus n.

bool rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n):
t= V(s) = s^e (mod n).
If t == m:
    Return true
Else:
    return false

returning true if signature s is verified and false otherwise.

keygen.c
This contains the implementation and main() function for the keygen program including make the public and private keys, get the current user's name, write the computed public and private key to their respective files.

The program should follow these steps:
1. Parse command-line options using getopt() and handle them accordingly.
2. Open the public and private key files using fopen(). Print a helpful error and exit the program in
the event of failure.
3. Using fchmod() and fileno(), make sure that the private key file permissions are set to 0600,
indicating read and write permissions for the user, and no

permissions for anyone else.
4. Initialize the random state using randstate_init(), using the set seed.
5. Make the public and private keys using rsa_make_pub() and rsa_make_priv(), respectively.
6. Get the current user’s name as a string. You will want to use getenv().
7. Convert the username into an mpz_t with mpz_set_str(), specifying the base as 62. Then, use
rsa_sign() to compute the signature of the username.
8. Write the computed public and private key to their respective files.
If verbose output is enabled print the following, each with a trailing newline, in order:
(a) username
(b) the signature s
(c) the first large prime p
(d) the second large prime q
(e) the public modulus n
(f) the public exponent e
(g) the private key d
All of the mpz_t values should be printed with information about the number of bits that constitute
them, along with their respective values in decimal. See the reference key generator program for
an example.
10. Close the public and private key files, clear the random state with randstate_clear(), and clear
any mpz_t variables you may have used.

encrypt.c
This contains the implementation and main() function for the encrypt program including read the public key from the opened public key file and encrypt the file.

The program should follow these steps:
1. Parse command-line options using getopt() and handle them accordingly.
2. Open the public key file using fopen(). Print a helpful error and exit the program in the event of
failure.
3. Read the public key from the opened public key file.
4. If verbose output is enabled print the following, each with a trailing newline, in order:

(a) username
(b) the signature s
(c) the public modulus n
(d) the public exponent e
All of the mpz_t values should be printed with information about the number of bits that constitute
them, along with their respective values in decimal. See the reference encryptor program for an
example.
5. Convert the username that was read in to an mpz_t. This will be the expected value of the verified
signature. Verify the signature using rsa_verify(), reporting an error and exiting the program if
the signature couldn't be verified.
6. Encrypt the file using rsa_encrypt_file().
7. Close the public key file and clear any mpz_t variables you have used.

decrypt.c

This contains the implementation and main() function for the decrypt program including read the private key from the opened private key file, decrypt the file.

The program should follow these steps:
1. Parse command-line options using getopt() and handle them accordingly.
2. Open the private key file using fopen(). Print a helpful error and exit the program in the event of
failure.
3. Read the private key from the opened private key file.
4. If verbose output is enabled print the following, each with a trailing newline, in order:
(a) the public modulus n
(b) the private key e
Both these values should be printed with information about the number of bits that constitute
them, along with their respective values in decimal. See the reference decryptor program for an
example.
5. Decrypt the file using rsa_decrypt_file().
6. Close the private key file and clear any mpz_t variables you have used.

Error Handing:

Credit: