Tanya Gyanmote
1887664
tgyanmot@ucsc.edu

Assignment 5 - Public Key Cryptography

Purpose:

The purpose of this program is to implement RSA encryption and decryption, and a key generator. All these programs together, securly encrypt, decrypt messages sent. The key generator creates a RSA public and private key pair. The encryptor will use the public key to encrypt files and the decrypter uses the private key to decrypt the encrypted file.

Files:

1. Decrypt.c
    a. This contains the implementation and main()function for the decrypt program.
2. Encrypt.c
    a. This contains the implementation and main()function for the encrypt program.
3. Keygen.c
    a. This contains the implementation and main()function for the keygen program.
4. Numtheory.c
    a. This contains the implementations of the number theory functions.
5. Numtheory.h
    a. This specifies the interface for the number theory functions.
6. Randstate.h
    a. This specifies the interface for initializing and clearing the random state.
7. Randstate.c
    a. This contains the implementation of the random state interface for the RSA library and number theory functions.
8. Rsa.c
    a. This contains the implementation of the RSA library.
9. Rsa.h:
    a. This specifies the interface for the RSA library.
10. Makefile
    a. Produces corresponding executibles for keygen,encrypt,decrypt
11. README.md
    a. Description of the program usage
12. DESIGN.pdf
    a. Describe design and design process of my program

Randstate.c

- randstate_init
    - Create a random seed using the seed which gets passed
    - Call gmp_randinit_mt with the state
    - Call gmp_randseed_ui with the state and the seed
- randstate_clear
    - Call gmp_randclear with gmp_randstate_t state
        - Clearing memory out of state

Numtheory.c

```
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
```

- Pow_mod
    - Using the puesdocode given
    - Creating temporary mpz values
    - Setting them to null
    - Initializing V to 1
    - Initializing P to a
    - Initializing temporary D to d
    - Have a while loop to check if temporary d is greater than 0
        - If the value of v is odd then set that value to v multiplied with v times p modulus n
        - multiply p with p and set it to p
        - Set p to p multiplied with p modulus n
        - Set d to d divided by 2
    - Setting my o variable to my v variable
    - Clearing all the values I created

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

- gcd
  - Using the puesdocode given
  - Creating temporary mpz values
  - Setting them to null
  - Initializing temporary a to a
  - Initializing temporary b to b
    - Having a while loop to check if temporary b is not equal to 0
    - Create a temporary variable for t
    - Set the value of t to b
    - Set temporary b value to a modulus of b
    - Set temporary a variable to t
  - Set temporary d variable to a
  - Clear all the varibles
- Make_prime
  - Create mpz_t variables
  - Setting them to null
  - Using mpz pow 2 to the power of bits
  - Have a do while
    - generates a new prime number using mpz_urandomb
    - While the value of temp and variables isn’t prime
  - Clear all mpz_t variables

```
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
```

- Mod_inverse using (Euclidean algorithm)
    - Using the puesdocode given
    - Creating temporary mpz values
    - Setting them to null
    - Initializing temporary r to n
    - Initializing temporary r prime to a
    - Initializing temporary t to 0
    - Initializing temporary t prime to 1
    - While r prime isn't 0
        - Creating temporary mpz values
        - Setting them to null
        - Setting temporary q to r divided by r prime
        - Seting temporary r value to the r prime
        - Seting temporary r prime value to r minus q times r prime
        - Seting temporary t value to the t prime
        - Seting temporary t prime to t minus q times t prime
        - Clear all temporary variables
    - If the statement for r is greater than 0
        - Set temporary t to 0
    - If the statement for that t is greater than 0
        - Set temporary t to t plus n
    - Set out to t
    - Clear temporary variables

```
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
```

- Is_prime
    - Using the puesdocode given
    - Create a function called find_totient for n-1 = 2^sr
        - Creating temporary mpz values
        - Setting them to null
        - Create a temp n and set it to n
        - Do mod of temp n by 2

        - Have a while loop to check if temp mod variable and 0 are equal to eachother
            - If they are then add 1 to S
            - Divide temporary n by 2
            - Moss temporary n by 2
        - Set r to temp n
        - Clear all mpz_t variables
    - Creating temporary mpz values
    - Setting them to null
    - check if the number is less than 4 then its prime so return true
    - check if the number is even then its not prime so return false
    - For loop starting at 1 until its iters
        - Have a range varible which is n minus 3
        - Using the random generator for the a value
        - Using power mod with inputs set to y
            - If y isn't 1 and n minus 1
                - Setting variable j to 1
                    - While j isn't equal to n miinus 1 and it's less than s minus 1
                        - Set power_mod of y,2,n all to y
                        - If y is 1
                            - return false
                        - Add one to the j value
                    - If the y variable isn't equal to n minus 1
                        - Return bool false
                    - Return bool true

Decrypt.c

- Create options i,o,n,v,h
- Initialize infile, outfile
- Initialize private key
- Using get opt
    - case i
        - input file to decrypt stdin
    - case o
        - decrypt with stdout
    - case n
        - Open the file with private key rsa.priv is the dault
        - If it doesnt open print a error message and exit

- case v
    - verbose output prints:
      - public modulus n and private key
      - number of bits and values in decimal
  - case h
    - prints help message
- call the fopen
  - private key file
  - If it doesn’t then print an error message plus return 0
- If verbose is called then print n,e and bits
- Decrypt to outfile form infile
  - call the rsa_encrypt_file
- clear all the mpz variables
- Close infile,outfile, and private key

Encrypt.c

- Create options
- Initialize infile, outfile
- Initialize public key
- Using get opt
  - case i
    - Open input file ,default is stdin
  - case o
    - Open output file, default is stdout
  - case n
    - the file with public key, the default is rsa.pub
  - case v
    - verbose variable becomes true
  - case h
    - prints synopsis
- call the fopen
  - public key file
  - If it doesn’t then print an error message plus return 0
- If verbose is true then print
  - public modulus n and private key
  - number of bits and values in decimal
- Initilzie char username
- set the correct username to an mpz
- if the RSA verify the username is false
  - print the reported error and return 0

- Encrypt infile to outfile
- Close infile,outfile, and public key
- Clear all the mpz variables changes/used in this file

Keygen.c

- Create options
- Initlialize public and private key file
- Using get opt
    - case b
        - specify the minimum bits needed
        - If not valid print help options
    - case i
        - specify the number of iterations for testing primes
        - If not valid print help options
    - case n
        - the file with public key with rsa.pub as the default
    - case d
        - the file with private key with rsa.priv as the default
    - case s
        - for random seed
    - case v
        - Verbose is true
    - case h
        - prints synopsis
- Using fopen to open both public, private key files
    - If it doesnt work return 0 and error
- Is verbose is true then print
    - username
    - signature
    - large prime, and the next large prime
    - the public modulus
    - public exponent
- Making sure the file permissions are 0600
- Calling init function in randstad to set seed
- Make a public and private key
- Initalize char username
- Set the username
- Set the username to mpz with base 62
- Write public key to public key file

- Write private key to private key file
- Using close to close both public and private files
- Clear all mpz variables

Rsa.c

rsa_make_pub
- calling make prime
- If the log is greater than nbits
    - P is the random number in range of (nbits/4,(3 times nbits) divided by 4
    - Q is the remaining
- Using random with iters to get a random value
- Go through the least common multiple with the greatest common denominator
    - The least common multiple
- For loop to call random numbers
    - Calling the mpz random function
    - Compute the gcd

rsa_write_pub
- writes public rsa to pbfile
- print using "%Zx\n" n,e,s username

rsa_read_pub
- reads public rsa to pbfile
- print using "%Zx\n" n,e,s username

rsa_make_priv
- creates rsa private key d
- compute the inverse of e with primes p and q

rsa_write_priv
- writes a private RSA key from pvfile
- print using "%Zx\n" n,d

rsa_read_priv
- reads a private RSA key from pvfile
- print using "%Zx\n" n,d

rsa_encrypt
- encrypting the message m
- using power mod with c,m,e,n

rsa_encrypt_file

- Get block size
- Allocate an array with k bytes
- Set the 0th byte to 0xFF
- While there is unprocessed bytes in the infile
    - Read at most k-1 bytes from infile
    - Using mpz_import convert read bytes
    - Encrypt m with rsa_encrypt

rsa_decrypt

- get message m
- encrypting the message m
- using power mod with c,m,d,n

rsa_decrypt_file

- Get block size
- Allocate an array with k bytes
- While there is unprocessed bytes in the infile
    - Scan the hexstring
    - Decrypt c into mpz_t using rsa_decrypt()
    - Use mpz_export to convert m back into bytes
    - Write j-1 bytes from index to outfile

rsa_sign

- calling power mod with s,m,d,n
- setting value of s to the value from power_mod

rsa_verify

- using power mod with t, s, e, n
- if value t and m are equal then true
- else false