Diwa Ashwini Vittala
CSE 13S - Fall 2021

⁰❀ Assignment 6: Design ❀ ₒ

PUBLIC KEY CRYPTOGRAPHY

About the program:

This program uses three executable files to (1) generate keys, then (2) encrypt a message so that only the intended end user can decrypt it, and then (3) to decrypt a valid file.

About the executables:

Decrypt.c

Inputs: The program takes in the following commands:

-h help, displays the program synopsis and usage
-i input file (default is stdin)
-o output file (default is stdout)
-n specifies the file containing the private key (default is rsa.priv)
-v verbose printing

Outputs: The message after decryption. This may be in the given output file, or stdout.

Encrypt.c

Inputs: The program takes in the following commands (from the assignment inst):

-h help, displays the program synopsis and usage
-i input file (default is stdin)
-o output file (default is stdout)
-n specifies the file containing the public key (default is rsa.pub)
-v verbose printing

Outputs: The message after encryption. This may be in the given output file, or stdout.

Keygen.c

Inputs: The program takes in the following commands (from the assignment inst):

-h help, displays the program synopsis and usage
-b specifies the minimum number of bits needed for the public key n (default is 256)
-i specifies the number of Miller-Rabin iterations for testing primes (default is 50)
-n specifies the public key file (default is rsa.pub)
-d specified the private key file (default is rsa.priv)
-s specifies the random seed for the random state initiation (default is time(NULL))
-v verbose printing

Outputs: The requested rsa keys are generated.

Other files involved:

Makefile, README.md, decrypt.c, encrypt.c, keygen.c, numtheory.[c & h], randstate.[c & h], rsa.[c & h], set.h

Pseudocode/information about each file:

decrypt.c:

```
Include the header files for all the necessary files
Message function:
    Prints the synopsis and usage
Enumerate the decrypt options
Main function:
    Create a set to contain command options
    Open the infile and outfile
    Set a path to rsa.priv and the private file, but do not open it yet
    Use getopt to parse command line options:
        If “h”:
            Print the help message
            Close all the files we opened
            End the program
        If “v”:
            Add verbose to the set
        If “n”:
            Set the private file path to the given input
        If “i”:
            Open the given file as infile
            If there was a problem opening the file:
                Print an error message and close other files
                End the program
        If “o”:
            Open the given file as outfile
            If there was a problem opening the file:
                Print an error message and close other files
                End the program
        Default:
            Print the help message
            Close all the files we opened
            End the program
    Open the private key file
    If we cannot open the private key file:
        Send error message
        Close all other opened files
        End the program
    Read private key from file
    If verbose printing is part of the set:
        Print the public modulus n and the number of bits
        Print the private key d and the number of bits
    Decrypt the file
    Close private key file and clear all mpz_t variables
```

encrypt.c:

```
Include the header files for all the necessary files
Message function:
    Prints the synopsis and usage
Enumerate the encrypt options
Main function:
    Create a set to contain command options
    Open the infile and outfile
    Set a path to rsa.pub and the public file, but do not open it yet
    Use getopt to parse command line options:
        If “h”:
            Print the help message
            Close all the files we opened
            End the program
        If “v”:
            Add verbose to the set
        If “n”:
            Set the private file path to the given input
        If “i”:
            Open the given file as infile
            If there was a problem opening the file:
                Print an error message and close other files
                End the program
        If “o”:
            Open the given file as outfile
            If there was a problem opening the file:
                Print an error message and close other files
                End the program
        Default:
            Print the help message
            Close all the files we opened
            End the program
    Open the public key file
    If we cannot open the public key file:
        Send error message
        Close all other opened files
        End the program
    Create needed mpz_t variables
    Create a username string (character array)
    Read public key from file
    If verbose printing is part of the set:
        Print the username
        Print the signature s and the number of bits
        Print the public modulus n and the number of bits
        Print the exponent e and the number of bits
    Set the username as an mpz_t variable
    If we can verify the signature:
        Encrypt the file
    Else:
        Print that there is an invalid key
    Close all the files and clear all mpz_t variables
```

keygen.c:

```
Include the header files for all the necessary files
Message function:
    Prints the synopsis and usage
Enumerate the keygen options
Main function:
    Create a set to contain command options
    Open the infile and outfile
    Set a path to rsa.pub and the public file, but do not open it yet
    Set a path to rsa.priv and the private file, but do not open it yet
    Use getopt to parse command line options:
        If “h”:
            Print the help message
            End the program
        If “v”:
            Add verbose to the set
        If “b”:
            Set the bits to the number that is input
            If there are less than 4 bits:
                Print an error message and end the program
        If “n”:
                Set the public file path to the given input
        If “d”:
                Set the private file path to the given input
        If “i”:
                Set the number of iterations to given input
        If “s”:
                Set the seed to given input
        Default:
                Print the help message
                End the program
Open the public key file
If we cannot open the public key file:
        Send error message
        End the program
Open the private key file
If we cannot open the private key file:
        Send error message
        Close the public file we opened
        End the program
Get the file number of the private file
Set the permissions for the file to 0600 so only the owner has access to it
Set the randomstate seed
Create needed mpz_t variables
Make public key file
Make private key file
Create the username string (character array)
Get the user’s name
Convert the string to an mpz_t type
Form the sign
Write the public and private keys to their files
If verbose printing is part of the set:
        Print the username
        Print the signature s and the number of bits
        Print the first prime p and the number of bits
        Print the second prime q and the number of bits
        Print the public modulus n and the number of bits
        Print the private key d and the number of bits
Clear the random state
```

```
    Close all the files and clear all mpz_t variables
```

numtheory.c:

```
Include the header files for all the necessary files
Greatest common divisor (gcd):
    Initialize needed mpz_t variables
While b (the third value given) is not 0:
        Store b into t (a temporary variable)
        Store a (the second value given) mod b into b
        Store t into a
    Store the value of a into d (the output value)
    Clear mpz variables
Mod inverse:
    Set temporary values r and r’ as n and a respectively
    Set temporary values t and t’ as 0 and 1 respectively
    While r’ is not 0:
        Store r/r’ into a value q
        Set the values r and r’ as r’ and r-qr’ respectively
        Set the values t and t’ as t’ and t-qt’ respectively
    If r > 1:
        There is no inverse, set i = 0
        Clear the mpz variables
    If t < 0:
        Add n to t
        Clear the mpz variables
    Set the value of the inverse i to the value of t
    Clear the mpz variables
Power mod (a^b mod n):
    Store 1 into a temporary variable v
    Store the base into the temporary variable p
    While the exponent is greater than 0:
        If the exponent is odd:
            Store (v * p) mod modulus into v
        Store (p * p) mod modulus into p
        Store (exponent/2) into the exponent variable
    Store the value of v into the output variable
Is prime (checks if something is prime):
    If the number is 2 or 3:
        The number is prime
    If the number is less than 2 or if number (mod 2) is 0:
        The number is not prime
    s = 0
    r = n - 1
    Let a variable first = 1, where “first” represents 2^s in the equation n-1=(2^s)*r
    While r is even:
        Add 1 to s
        Multiply the first part by 2
        Let r be the floor of n-1 / first
    From i=1 to k iterations:
        Chose a random value from 2 to n-2
        Let a value y be the power mod of the random variable, r, and n
        so that y = random number ^ r (mod n)
        If y is not 1 and y is not n=1:
            Let a variable j=1
            While j<= s-1 and y != n-1:
                Store the mod of y,2,n into y so that y = y^2 (mod n)
                If y is 1:
                    The number is not a prime number, clear variables
                Add one to j
            If after the while loop, y is not n-1:
                The number is not a prime number, clear variables
    If we could leave the leave the loop, that means the value must be a prime number
Make prime (makes a prime number of at least some number of bits):
    While a prime number had not been found:
        Make a random number that is the specified bits long
using a random number generator
Add 1 to the number so that it is within range
    If prime, we will be outside the loop.
    Store that value into p (the first number given)
```

numtheory.h:

Provided by professor, declarations of all the functions/variables/basic headers required for numtheory.c

randstate.c:

```
Include the header files for all the necessary files
Randstate initialize:
    Set the state using gmp
    Set the random seed to state using gmp
Randstate clear:
    Clear the random state
```

randstate.h:

Provided by professor, declarations of all the functions/variables/basic headers required for randstate.c

rsa.c:

```
Include the header files for all the necessary files
Rsa make public key:
    In a loop:
        Let pbits be a random number within the range [n/4, 3n/4)
        Let qbits be what is left over of nbits after pbits is calculated
        Calculate p and q prime numbers
        Multiply p and q to obtain n
        If the $\log_2(n)$ is >= nbits, then break
    Calculate $\varphi(n) = (p-1)(q-1)$
    In a loop:
        Create a random number of nbits
        Compute the gcd of the random number and computed totient
        If coprime with totient found, meaning the gcd is 1:
            The exponent is found, break the loop
    Clear the variables
Rsa write public key:
    Write the public rsa key to pbfile in the order:
    n, e, s, as hexstrings with newlines at the end of each (use gmp functions)
    Username as a string with newline at the end
Rsa read public key:
    Read the public rsa key from pbfile in the order:
    n, e, s, as hexstrings with newlines at the end of each (use gmp functions)
    Username as a string with newline at the end
Rsa make private key:
    Find the totient, and the mod inverse of e with the totient as the mod
    Store the inverse value into d
Rsa write private key:
    Write the private rsa key to pvfile in the order:
    n then d as hexstrings with newlines at the end
Rsa read private key:
    Read the private rsa key from pvfile in the order:
    n then d as hexstrings with newlines at the end
Rsa encrypt:
    Find the power mod of the message such that $c = m^e \pmod{n}$
```

```
Rsa encrypt file:
    Calculate a block k = floor of ($\log_2$(n) - 1 / 8)
    Dynamically allocate an array to hold k bytes of unsigned 8 bit integer pointers
    Let the 0th byte of the block be 0xFF
    While there are unprocessed bytes leftover:
        j = the actual number of bytes read after asking to read k-1 bytes in from infile
        Use mpz import to convert all the read bytes into an mpz_t m (the message),
        most significant word first, 1 for endian parameter, and 0 for the nails parameter
        Encrypt the message into a variable c that will contain the cipher text
        Print to the cipher text outfile
    Free the array and clear variables
Rsa decrypt:
    Find the power mod of the cipher text such that message = c^d (mod n)
Rsa decrypt file:
    Calculate a block k = floor of ($\log_2$(n) - 1 / 8)
    Dynamically allocate array that can hold k bytes of type unsigned integer 8 bits
    While it's not the end of the file:
        Scan a hexstring as mpz_t c
        Decrypt the cipher text into the message
        Use mpz_export to convert the read bytes
        Write out j-1 bytes to outfile
Rsa sign:
    Find the signature s so that s = m^d (mod n)
Rsa verify a signature:
    Find t = s^e (mod n)
    If the value of t is the same as m (the signature is correct):
        Clear variables and return true
    Otherwise:
        Clear variables and return false
```

rsa.h:

Provided by professor, declarations of all the functions/variables/basic headers required for rsa.c

set.h:

Provided by professor (from previous assignment in this class), contains functions that can be used for set operations