CSE 13S Assignment 5 - Public Key Cryptography

Jaren Kawai - jkawai

General Idea:

- Create a program that creates an RSA public key, and is capable of both encrypting and decrypting the key.

Power-Mod

- Make a variable that is equal to a for squaring purposes
- While d > 0, we can continue to carry out function
    - Check if d is odd
        - Change value of output variable to output variable multiplied by variable created earlier mod n
    - Set created variable to variable squared mod n
    - Update d to d/2 to carry out loop correct amount of times

Is-Prime

- Create a variable that is set to an odd number such that n -1 = 2^sr
- Iterate until iterator is equal to or greater than k
    - Create a random number
    - Set extra variable to check if power-mod gives a value of 1 or a value equal to n-1
        - Create new variable that equals 1
        - As long as j is less than or equal to s-1 and power-mod variable is not equal to n-1
            - Power_mod variable is updated
            - Check if power_mod variable is equal to 1
                - Number is not prime
            - Number that was set to 1 earlier is not incremented by 1
        - Check is power_mod variable is not equal to n-1
            - Number is not prime
- Number is prime

GCD

- As long as b is not equal to 0, we can set d to b and b to a mod b
- Set a to d

Mod-Inverse

- Extended Euclid's Theorem
- Carry out simulated parallel assignment with r and r prime
- Output is set to 0 if no inverse
- Divide r and r prime set to q
- Set temp variable to r

- Set r to r prime
- Multiply q and t
- Set another temp variable to t
- Set t to output variable - q times t
- Set output to temp variable that had t

Make-Prime

- Generate a prime number that is stored in p
- Run is_prime() test for the amount of iterations that are given as a parameter
- Stop when is_prime returns true

RSA Library

Rsa-make-pub

- Create prime numbers for p and q
- Number of bits that are given to p is decided by a random number using random
- Take rest of the bits and assign them to q
    - Miller-Rabin iterations for is_prime() is determined by the iters parameter
- Calculate lambda by taking the product of p-1 and q-1 and dividing it my the gcd of p-1 and q-1
- Find value for e which is the public exponent
    - Create a loop that generates random numbers
    - Computer gcd of create number and lambda
    - Stop loop when created number and lambda are coprime

Rsa-write-pub

- Create char pointer variables for n, e, and s
- Write each variable to file that is passed as a parameter

Rsa-read-pub

- Must be read in order of n, e, s, then username
- Use fscanf

Rsa-make-priv

- Take in two prime numbers and an exponent
- Calculate lambda as before
- Compute the inverse of e modulo lambda

Rsa-write-priv

- Write to file passed as parameter
- Use fprintf

Rsa-read-priv

- Use fscanf to read from file passed as parameter
- Must be read in the order n then d

Rsa-encrypt

- Use power_mod function written in numtheory file to find m^e mod n
- Result of calculation should be stored in c

Rsa-encrypt-file

- Write contents of passed in infile to the outfile
- Must transfer contents over in blocks due to restrictions placed by n which prevents blocks from being greater than n
- Blocks also cannot be of size 0 or 1
    - Determine the size of blocks
    - Dynamically allocate memory for the size of the block using malloc
    - Set the first element to 0xFF which will prepend the byte that is needed in the workaround method
    - Can read blocks up to size k-1 and keep a variable that keeps track of how many bytes that have been read
    - Start at index 1 to not overwrite the first element
    - While loop that continues while there are still bytes left
        - Use import function to convert read bytes to a variable m that will eventually be encrypted
        - encrypt
        - Write encrypted values to outfile
    - Free block

Rsa-decrypt

- Use power mod function to calculate m which is c^d mod n

Rsa-decrypt-file

- Write contents of passed in infile to the outfile
    - Determine the size of blocks
    - Dynamically allocate memory for the size of the block using malloc
    - Keep track of how many bytes have been read with the use of a temp variable
    - Start at index 1 to not overwrite the first element
    - While not at end of file
        - Decrypt
        - Use export function to convert read bytes to a variable m that will eventually be decrypted

        - Write decrypted values of m to an outfile
    - Free block

Rsa-sign

- Creates rsa signature by using power_mod function to calculate m^d mod n

Rsa-verify

- Find variable t such that t is equal to s^e mod n, use power_mod function for this
- If t is equal to m, return true
- Otherwise return false

Keygen

- Use getopt to take in arguments for bindsvh
- Check if integer version of optarg is able to be used, otherwise return error message or set default values if no value is given
- Use fopen() to open files for both the public and private key, if there is an error, display message
- Set permissions for key files to give read and write permissions to people who are allowed to have access to the keys
- Initialize a new random state
- Create the key pairs using the functions created above
- Use getenv() to get a username from the user that will eventually be written to files in some of the functions
- Convert user name to type mpz_t and create signature using signature function
- Write encrypted public and private keys to respective files
- If needed, print out message from assignment pdf
- Close key files and clear any mpz_t variables that were used

Encrypt

- Use getopt() to handle arguments inputted by the user
    - Here user specifies infiles and outfiles as well as whether or not they want certain information displayed and what file they want to use that has the public key
- Open key file, print error message if failure
- Read public key from keyfile that was opened
- Print output message if verbose output is enabled
- Verify signature using verify function created earlier
    - Return error if signature doesn't pass verify test
- Encrypt file using file encryption function created earlier
- Close public key file and clear any variables used

Decrypt

- Use getopt() to handle arguments inputted by the user

    - Here user specifies infiles and outfiles as well as whether or not they want certain information displayed and what file they want to use that has the private key
- Open key file, print error message if failure
- Read private key from keyfile that was opened
- Print output message if verbose output is enabled
- Decrypt file using file decryption function created earlier
- Close private key file and clear any mpz_t variables that were used