Assignment Five Design.pdf

By James Yim

Description of Program:

This program will be able to create keys for encryption and decryption. It will also be able to encrypt and decrypt messages using keys. We will use the fact that it is believed to be hard to factor large composite numbers into their constituent primes. With this we are able to have a message encrypted unknown to adversaries who do not have the proper key. If someone has the proper key they are able to decrypt the message and read it.

Files to be included in asgn4

- decrypt.c
    - This contains the code that will decrypt a message.
- ecrypt.c
    - This contains the code that will encrypt the message
- keygen.c
    - This contains the code that will generate keys for encrypt.c and decrypt.c
- numtheory.c
    - This contains the implementation of the numtheory library, used for decrypt.c, encrypt.c, and keygen.c

- numtheory.h
    - This contains the function headers for the numtheory library
- randstate.c
    - This contains the code for the randstate functions
- randstate.h
    - This contains the function headers for the randstate library
- rsa.c
    - This contains the implementation for the rsa library used for encrypting and decrypting and creating keys.
- rsa.h
    - This contains the function headers for the rsa library.
- Makefile
    - This contains the script that will compile the program.
- README.md
    - This describes how to use the program. It will explain command line options that the program accepts. It will also note bugs and errors that are in the program.
- DESIGN.pdf
    - This pdf will show the process of how to create this program. It will include pseudocode and it will credit sources where information was learned.

Pseudocode and Design

randstate.c

```
randstate_init(seed)
{
        initiate state with Mersenne Twister algorithm
        set the state seed with input seed

}

randstate_clear()

{

        clears the state using gmp_randclear()

}
```

numtheory.h

```
gcd(d, a ,b)
{
        while (b does not equal 0)
        {
                tempVariable = b
                b =  a % b
                a = tempvariable
        }
        return a
}
mod_inverse()
```

```
isprime(number, iterations)
{
	write n - 1 to equal 2^s times r where r is off
	for the amount of iterations
	{
		choose a random number from a to n - 2
		set y to the 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++
			if (y != n - 1)
			{
				return false
			}
		}
	}
	return true
}
```

Notes about code

-

Error Handling

-

Credit

-