# Assignment 6 - Public KEy Cryptography

Description: This program will demonstrate the encryption and decryption of private and public keys using the RSA algorithm. The keys represent shared data between two parties that can be shared publicly in the form of two keys. This program will generate public and private keys along with encoding and decoding the keys using GNU libraries of high math precision to be utilized. SSH key generation and sharing of keys allows for highly sensitive information to be shared in a secure manner.

Makefile:
• CC = clang must be specified.
• CFLAGS = -Wall -Wextra -Werror -Wpedantic must be specified.
• pkg-config to locate compilation and include flags for the GMP library must be used.
• make must build the encrypt, decrypt, and keygen executables, as should make all.
• make decrypt should build only the decrypt program.
• make encrypt should build only the encrypt program.
• make keygen should build only the keygen program.
• make clean must remove all files that are compiler generated.
• make format should format all your source code, including the header files.

README.md: This must use proper Markdown syntax and describe how to use your program and Makefile. It should also list and explain any command-line options that your program accepts. Any false positives reported by scan-build should be documented and explained here as well

DESIGN.pdf: This document must be a proper PDF. This design document must describe your design and design process for your program with enough detail such that a sufficiently knowledgeable programmer would be able to replicate your implementation.

decrypt.c: This contains the implementation and main() function for the decrypt program.

encrypt.c: This contains the implementation and main() function for the encrypt program.

keygen.c: This contains the implementation and main() function for the keygen program.

numtheory.c: This contains the implementations of the number theory functions.

numtheory.h: This specifies the interface for the number theory functions.

randstate.c: This contains the implementation of the random state interface for the RSA library and number theory functions.
randstate.h: This specifies the interface for initializing and clearing the random state.

rsa.c: This contains the implementation of the RSA library.
rsa.h: This specifies the interface for the RSA library

Randstad.c

Randstad_init(uint64) initializes global random "state" with MT algorithm and using seed and a random seed. Calls gmp functions of randinit_mt and randseed_ui

Randstate_clear (frees and clears memory of "state" and does so by calling gmp_randclear()

Numtheory.c
is_prime (n,iters)
Require iters to determine if the large generated number is prime

make_prime(p,bits,iters)
Generates a prime number that will be stored in p and the generated number should be at least "bits" number of bits. Uses iters to pass through is_prime()to test the number

Gdc(d,a,b)

Commutes greatest common denominator between a,b and stores it in d
Utilizes a while loop to take in b and send b to a mob b and continue to do so while b not equal 0.

mod_inverse(i, a ,n) computes the mod inverse i of modulo n

Rsa.c
make_pub (p,q,n,e,nbits,iters)
Creates a rsa pub key with p,q and their product n

write_pub(writes out the public key t pbfile. Formatted as n,e,s written as hex strings and then a username read_pub()reads RSA key from pbfile and the formate of a public should be n,e,s then the username and ulitized gmp formatted inputs for reading hex strings.

void rsa_make_priv(mpz_t d, mpz_t e, mpz_t p, mpz_t q) Creates a new RSA private key d given primes p and q and public exponent e. To compute d, simply compute the inverse of e modulo $\phi(n) = (p-1)(q-1)$.
void rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile) Writes a private RSA key to pvfile. The format of a private key should be n then d, both of which are written with a trailing newline. Both these values should be written as hexstrings.
void rsa_read_priv(mpz_t n, mpz_t d, FILE *pvfile) Reads a private RSA key from pvfile. The format of a private key should be n then d, both of which should have been written with a trailing newline. Both these values should have been written as hexstrings.

void rsa_encrypt(mpz_t c, mpz_t m, mpz_t e, mpz_t n) Performs RSA encryption, computing ciphertext c by encrypting message m using public exponent e and modulus n. Remember, encryption with RSA is defined as E(m) = c = me (mod n).

void rsa_encrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t e) Encrypts the contents of infile, writing the encrypted contents to outfile. The data in infile should be in encrypted in blocks. Why not encrypt the entire file? Because of n. We are working modulo n, which means that the value of the block of data we are encrypting must b

void rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n) Performs RSA decryption, computing message m by decrypting ciphertext c using private key d and public modulus n. Remember, decryption with RSA is defined as D(c) = m = c d (mod n). void rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d) Decrypts the contents of infile, writing the decrypted contents to outfile. The data in infile should be decrypted in blocks to mirror how rsa_encrypt_file() encrypts in blocks.

void rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n) Performs RSA signing, producing signature s by signing message m using private key d and public modulus n. Signing with RSA is defined as S(m) = s = md (mod n).

bool rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n) Performs RSA verification, returning true if signature is verified and false otherwise. Verification is the inverse of signing. Let t = V(s) = s e (mod n).