Tommy Chen
10/26/22
Prof Miller

Public Key Cryptography

Overview: Encrypt, and decrypt stuff

Helper Functions:

- *Void mpz_init(mpz_t x)*
  - Initializes mpz_t
- *Void mpz_inits(mpz_t x….NULL)*
  - Can initialize multiple mpz_t variables
- *Int mpz_cmp_ui(mpz_t x, long y)*
  - If x > y return 1. X < y return -1. x=y return0
- *Int mpz_odd_p(mpz_t x)*
  - Returns non-zero if odd, zero if no
- *Void mpz_mul(mpz x, mpz y, mpz z)*
  - X = y * z
- *Void mpz_mod(mpz x, mpz y, mpz z)*
  - X = y % z
- *mpz_cdiv_q_ui(mpz x, mpz y, long z)*
  - X = y/z
- *gmp_printf*
  - Prints using gmp variables
- *void mpz_urandomm (mpz_t rop, gmp_randstate_t state, const mpz_t n)*
  - Randoms
- *void gmp_randinit_mt (gmp_randstate_t state)*
  - Initialize state for a Mersenne Twister algorithm. This algorithm is fast and has good randomness properties.
- *void gmp_randseed_ui (gmp_randstate_t state, unsigned long int seed)*
  - Set an initial seed value into state.
- *void mpz_add (mpz_t rop, const mpz_t op1, const mpz_t op2)*
  - Function: void mpz_add_ui (mpz_t rop, const mpz_t op1, unsigned long int op2)
  - Set rop to op1 + op2.
- *Function: void gmp_randclear (gmp_randstate_t state)*
  - Free all memory occupied by state.
- *mpz_clears(mpz_t x…)*
  - Free all the mpz variables listed in the parameters

NumTheory:

- Functions:
    - *Pow_mod(mpz_t o, mpz_t a, mpz_t d, mpz_t n)*
        - Equ: o = (a^d) % n
        - Iterate until d is less than or equal to zero. Each iteration will cause d to be divided by 2. If d is odd, multiply o by the base of the exponent while also moding by n. Square p and mod by n each iteration too.
    - *isPrime(mpz_t n, uin64_t iters):*
        - Returns True it is a prime number and false otherwise
        - Pseudo:
            - Return False immediately if n < 3 or n is even
            - Find a *r* and *s* so it satisfies n-1 = (2^s) x r
            - Iterate *iter* times. Initialize a random number. Call power mod with y as the input, random number as the base, r as the exponent, and n being the modular value.
            - If the random value isnt one or one less than our number
                - Set j as 1
                - Iterate until j is less than s-1 and y is not one less than our number
                - Call pow_mod(y,y,2,n)
                - If y is ever one return false
                - Increment j by 1 each iteration
            - Return True if it finishes iterating
    - Gcd(mpz_t a, mpz_t)
        - Greatest common denominator
        - Iterate until b is equal to zero
            - Set temp equal to b
            - Set b = a % b
            - Set a = temp
            - Return a after looping
    - Prime(mpz_t p, uint64_t bits, uint64_t iters):
        - Generate a random prime number that is at least “bits” long
    - Mod inverse:
        - Set r, r1 equal to n,a
        - Set t and t1 to zero
        - Loop until r1 is 0
        - Set q equal to r divided by r1
        - Set r to r1
        - Set r1 to r minus q times r1
        - Set t to t1
        - Set t1 equal to t- q * t1
        - Return t if r < 1
        - other wise return no inverse

Randstate:

- Functions

- Void randstate_init(uint64_t seed)
    - Create an extern variable named “state” with the Mersenne twister algorthim using the inputted seed
- Void randstate_clear(void)
    - Clears all random state vars named state

RSA:

- Functions
    - void rsa_make_pub(mpz_t p, mpz_t q, mpz_t n, mpz_t e, uint64_t nbits, uint64_t iters)
        - Get pbits through the random function within the given bounds
        - Get qbits by subtracting pbits to nbits
        - Create p and q with make_prime() using their respective bits
        - Compute LCM using the equation given
        - Find a public exponent e. In a loop, generate random numbers around nbits usin randomb(). Compute the gcd of the random number and LCM

    - void rsa_write_pub(mpz_t n, mpz_t e, mpz_t s, char username[], FILE *pbfile)
        - Writes a public RSA key to a file using fprintf
    - void rsa_read_pub(mpz_t n, mpz_t e, mpz_t s, char username[], FILE *pbfile)
        - Reads the RSA key from the file using fscanf
    - void rsa_make_priv(mpz_t d, mpz_t e, mpz_t p, mpz_t q)
        - Create a new private key
        - Find LCM of (p-1) and (q-1)
        - Take inverse mod of it to get our private key
    - void rsa_write_priv(mpz_t n, mpz_t d, FILE *pvfile)
        - Write a private key using fprintf
    - void rsa_read_priv(mpz_t n, mpz_t d, FILE *pvfile)
        - Read the private key from the file using fscanf
    - void rsa_encrypt(mpz_t c, mpz_t m, mpz_t e, mpz_t n)
        - Encrypt the key
        - Use this equation to encrypt: E(m) = c = m^2 % n
    - void rsa_encrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t e)
        - Encrypts the content of infile.
        - Get block size
        - Iterate through the key with each iteration encrypting the blocksize
    - void rsa_decrypt(mpz_t m, mpz_t c, mpz_t d, mpz_t n)
        - Decrypt the key by calling pow mod
    - void rsa_decrypt_file(FILE *infile, FILE *outfile, mpz_t n, mpz_t d)
        - Decrypt the file in the file
        - Get the file
        - Read the blocks and decrypt it with the functions
    - void rsa_sign(mpz_t s, mpz_t m, mpz_t d, mpz_t n)
        - Signs a message using pow mod
    - bool rsa_verify(mpz_t m, mpz_t s, mpz_t e, mpz_t n)

- Decrypt the username and compares it to the input to verify rsa, return True or False depending on it

Keygen.c: Main function that takes the following command lines options

- Options
  - b : input minimum bits needed for the public mod n(Default = 1024)
  - -i: inputs number of iteration(default:50)
  - -n pbfile: specifies public key file(default:rsa.pub)
  - -d pvfile: specidies the private key file(defaul: rsa.priv)
  - -s specifies the rand seed for random state initialization(default: the seconds since the unix epoch, give by time(NULL)
  - -v: enables verbose output
  - -h: displays program synopsis and usage

Decrypt.c: main function of decryptor

Options:

- -i: specifies the input file to decrypt(default: stdin)
- o: specifies the output file to decrypt(default: stdout)
- -n: specifies file containing the private key(default: rsa.priv)
- -v: enables verbose output
- -h: displays program synopsis and usage

Encrypt.c

Options:

- -i: specifies the input file to decrypt(default: stdin)
- o: specifies the output file to decrypt(default: stdout)
- -n: specifies file containing the private key(default: rsa.priv)
- -v: enables verbose output
- -h: displays program synopsis and usage