Public Key Cryptography Design

Lucais Sanderson

February 26, 2023

1 Introduction

The purpose of this program is to realize the Schmidt-Samoa encryption algorithm by constructing the mathematical functions used by the algorithm, implementing the algorithm itself, and finally a user interface.

The program comprises three primary components: the key generator, encryptor, and decryptor. Additionally, the key generator requires mathematical functions which are described below.

2 Pseudocode

2.1 Keygen

See Section 7 on `ss_make_pub()` in the assignment document.

```
ss_make_pub(nbits, iters):
    # first determine bits in p
    pbits = random(nbits/5, 2*nbits/5)
    # determine qbits with remaining bits such pbits has two contributions
    qbits = nbits - 2 * pbits
    # create primes p and q
    while p !| q-1 and q !| p-1:
        p = make_prime(pbits, iters)
        q = make_prime(qbits, iters)
    # create public key n
    n = p * p * q
```

See Section 7 on `ss_make_priv()` in the assignment document.

```
ss_make_priv(p, q):
    pq = p * q
    # create private key d
    # note: lcm(a,b) = ab/gcd(a,b)
    d = mod_inverse(n, lcm(p-1, q-1))
```

2.2 Encryptor

See Section 7 on `ss_encrypt()` in the assignment document.

```
ss_encrypt(m, n):
```

```
    c = pow_mod(m, n, n)
```

See Section 7 on `ss_encrypt_file()` in the assignment document.

```
ss_encrypt_file(infile, outfile, n):
    # figure block size and allocate space
    k = (bits_in(n^0.5) - 1) / 8
    buffer = allocate uint8 * k
    # prepend so the block cannot be 1 or 0
    b[0] = 0xFF
    # iterate for all characters in infile and encrypt them
    for (i = 1, ch = getchar from infile, i++):
        buffer[i] = ch
        # make sure block no larger than k - 1
        # to account for prepended 0xFF
        if i == k - 1:
            import buffer to mpz in m
            c = ss_encrypt(m, n)
            write c to outfile
            # reset i and clear buffer
            i = 1
            for j = 1, j < k, j++:
                buffer[j] = 0x00
    # write any remaining data from buffer
    import buffer to mpz in m
    c = ss_encrypt(m, n)
    write c to outfile
```

2.3 Decryptor

See Section 7 on `ss_dycrypt()` in the assignment document.

```
ss_decypt(c, d, pq):
    m = pow_mod(c, d, pq)
```

See Section 7 on `ss_decrypt_file()` in the assignment document.

```
ss_decypt_file(infile, outfile, pq, d):
    # ensure block size here is greater than
    #   it is in encrypt_file; pq > n^0.5
    k = (bits_in(pq) - 1) / 8
    buffer = allocate uint8 * k bytes
    while (c = getline) not EOF:
        m = ss_decrypt(c, d, pq)
        export from mpz to buffer
        for (i = 1, i < j, i++):
            putchar in outfile from buffer[i]
```

2.4 Number Theory Functions

See Section 6.2 in the assignment document.

```
make_prime(bits, iters):
    while !(is_prime(p, iters)):
```

```
        p = random() # bits long
    return p
```

See Section 6.2 in the assignment document.

```
is_prime(n, iters):
    # need r and s such that n-1 = r2^s and r is odd
    for (r = n - 1, r is odd, r = r / 2):
        s++
    for iters:
        a = random(2, n - 2)
        y = pow_mod(a, r, n)
        if y != 0 and y != n - 1:
            j = 1
            while j <= s - 1 and y != n - 1:
                y = pow_mod(y, 2, n)
                if y == 1:
                    return false
                j++
            if y != n - 1:
                return false
    return true
```

See Section 6.1 in the assignment document.

```
pow_mod(a, d, n):
    v = 1
    p = a
    while d > 0:
        if d is odd:
            v = (v * p) mod n
        p = (p * p) mod n
        d = d / 2
    return v
```

See Section 6.3 in the assignment document.

```
mod_inverse(a, n):
    t = 0
    t' = 1
    r = n
    r' = a
    while r' != 0:
        q = r/r'
        # (r,r') = (r',r-q*r')
        temp = r
        r = r'
        r' = temp - q * r'
        # (t,t') = (t',t-q*t')
        temp = t
        t = t'
        t' = temp - q * t'
    if r > 1:
        return no inverse
    if t < 0:
        t = t + n
```

```
    return t
```

See Section 6.3 in the assignment document.

```
gcd(a, b)
    while b != 0:
        t = b
        b = a mod b
        a = t
    return a
```

3 Citations

- All number theoretic functions derived from Darrel Long in the assignment spec as well as some from other functions.
- Thanks to Audrey Ostrom for this template!