现代密码学 · Hw3

计01　容逸朗　2020010869

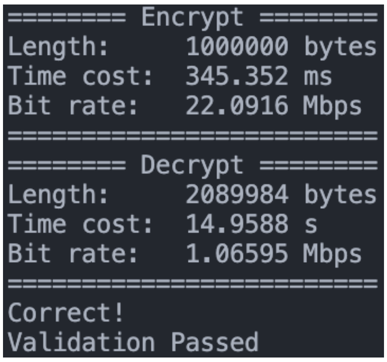
公钥密码算法实现

1. 运行方式

- 测试前请先进入算法对应文件夹：`cd rsa`
- 进入后按照 `README.md` 的指示，运行 `bash run.sh` 即可进行测试。

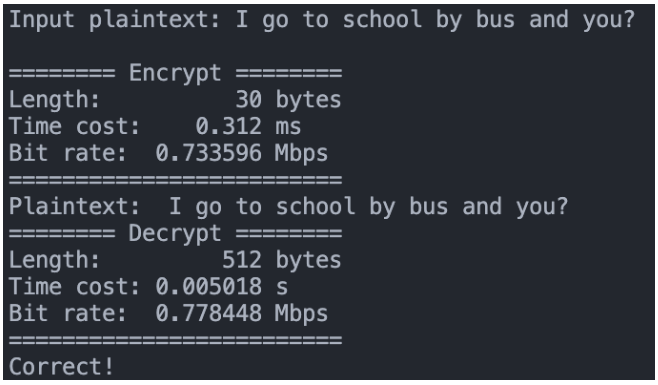
2. 实现效果

- 可以看见，加密速度约为 22 Mbps，解密的速度为 1 Mbps。



```
======== Encrypt ========
Length:     1000000 bytes
Time cost:  345.352 ms
Bit rate:   22.0916 Mbps
=========================
======== Decrypt ========
Length:     2089984 bytes
Time cost:  14.9588 s
Bit rate:   1.06595 Mbps
=========================
Correct!
Validation Passed
```

- 一个加解密的例子如下：



```
Input plaintext: I go to school by bus and you?

======== Encrypt ========
Length:           30 bytes
Time cost:     0.312 ms
Bit rate:   0.733596 Mbps
=========================
Plaintext:  I go to school by bus and you?
======== Decrypt ========
Length:          512 bytes
Time cost: 0.005018 s
Bit rate:   0.778448 Mbps
=========================
Correct!
```

- 注：为实现简便，此处不支援非 ascii 字符加密。

3. 实现简介

- **公私钥对生成：**

  首先生成两个大素数 p, q，然后计算 $n = pq$，再选取 $e = 65537$，计算 $d = e^{-1} \mod (p-1)(q-1)$。

```
RSA(int bits_ = Rsa_bits): bits(bits_),
    elen(bits / 8 - 11), dlen(bits / 4) {
    _gen_10k_prime();
    mpz_class p = _gen_prime(bits / 2);
    mpz_class q = _gen_prime(bits / 2);
    n = p * q;
    e = 65537;

    mpz_class phi = (p - 1) * (q - 1);
    mpz_class tmp;
    _exgcd(e, phi, d, tmp);
    if (d < 0) d += phi;
}
```

- **素数生成:**

  先生成一个 1024 位的大整数，若该数是偶数，则先加一，然后检查该数是否可能为素数，若否则加 2，然后重复检测之。

```
mpz_class _gen_prime(int bits = Prime_Bits) {
    bits--;
    mpz_class res, low;

    // scale the random number to the range [low, high]
    res = _gen_random_number(bits);
    mpz_ui_pow_ui(low.get_mpz_t(), 2, bits);
    res += low;

    // make sure the number is odd
    if (res % 2 == 0) res += 1;
    while (!_check_prime(res, bits)) res += 2;

    return res;
}
```

- **检测素数:**

  首先，我们可以先利用小于 10000 的素数做初筛，若这些素数都不能整除目标素数，则可以利用 miller-rabin 方法做进一步的检测：

```cpp
bool _check_prime(mpz_class tar, int bits = Prime_Bits) {
    // 1. use small primes to check
    for (int i = 0; i < Prcnt; i++) {
        if (Pr[i] * Pr[i] > tar)
            return true;
        if (tar % Pr[i] == 0)
            return false;
    }

    // 2. use miller-rabin to check
    return _miller_rabin(tar, bits);
}
```

接下来是 miller_rabin 算法的过程，此处不再复述算法内容：

```cpp
bool _miller_rabin(mpz_class tar, int bits = Prime_Bits, int rnd = 50) {
    // 1. factorization tar-1
    mpz_class r = tar - 1;
    int s = 0;
    while (r % 2 == 0) {
        r /= 2;
        s++;
    }

    // 2. test rnd rounds
    for (int t = 0; t < rnd; t++) {
        mpz_class a = _gen_random_number(bits - 1) + 2;
        mpz_class y;
        mpz_powm(y.get_mpz_t(), a.get_mpz_t(), r.get_mpz_t(), tar.get_mpz_t());
        if (y == 1 || y == tar - 1) continue;
        for (int j = 0; j < s; j++) {
            y = y * y % tar;
            if (y == 1) return false;
            if (y == tar - 1) break;
        }
        if (y != tar - 1) return false;
    }

    return true;
}
```

- **加密算法：**

加密算法的内核十分简单，只需要计算 $c = m^e \mod n$ 即可：

```cpp
mpz_class _encrypt(const mpz_class &m) {
    mpz_class res;
    mpz_powm(res.get_mpz_t(), m.get_mpz_t(), e.get_mpz_t(), n.get_mpz_t());
    return res;
}
```

考虑都加密的内容可能很长，因此我们还需要做分段处理，注意分段长应小于 $elen = \frac{2048}{8} - 11$，若长度不足，则从前端补零直至长度足够，同时，若加密后该块结果长度不足 $dlen = \frac{2048}{4} = 512$，则也需要在密文前补零：

```cpp
std::string encrypt(std::string &msg) {
    std::string res;
    int len = msg.length();

    std::string src = msg;
    // RSA - No padding
    while (len % elen != 0) {
        src = char(0) + src;
        len++;
    }

    for (int i = 0; i < len; i += elen) {
        std::string tmp = src.substr(i, elen);
        std::string enc = _encrypt(_convert2mpz(tmp)).get_str(16);
        while (enc.length() < dlen) enc = '0' + enc;
        res += enc;
    }

    return res;
}
```

- **解密算法：**

由于加密时已经完成了补零的动作，因此我们可以直接按长度把数据分割为不同的块，然后依次解密便可得到明文。

```cpp
mpz_class _decrypt(const mpz_class &m) {
    mpz_class res;
    mpz_powm(res.get_mpz_t(), m.get_mpz_t(), d.get_mpz_t(), n.get_mpz_t());
    return res;
}
```

```cpp

std::string decrypt(std::string &msg) {
    std::string res;
    int len = msg.length();
    for (int i = 0; i < len; i += dlen) {
        std::string tmp = msg.substr(i, dlen);
        mpz_class num;
        mpz_set_str(num.get_mpz_t(), tmp.c_str(), 16);
        res += _convert2str(_decrypt(num));
    }
    return res;
}
```

公钥密码算法计算

设 E 是 Z_{11} 上的椭圆曲线 $y^2 = x^3 + x + 6$。

1. 计算 E 上的所有点。

   首先，我们有：

x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
x^2	0	1	4	9	5	3	3	5	9	4	1
y^2 / x^3+x+6	6	8	5	3	8	4	8	4	9	7	4
y	-	-	4,7	5,6	-	2,9	-	2,9	3,8	-	2,9

   由此可知，在 E 上的点有 $(2,4),(2,7),(3,5),(3,6),(5,2),(5,9),(7,4),(7,7),(8,3),(8,8),(10,2),(10,9)$。

2. 证明 $\alpha = (2,7)$ 是本原元。

   首先，我们可以计算 2α ，由公式知：

   - $m_1 = (3x_\alpha^2 + a)(2y_\alpha)^{-1} \equiv (3 \times 2^2 + 1)(2 \times 7)^{-1} \equiv 13 \times 14^{-1} \equiv 13 \times 4 \mod 11 = 8$
   - $x_1 \equiv m_1^2 - 2x_\alpha \mod 11 \equiv 5$
   - $y_1 = y_\alpha + m_1(x_1 - x_\alpha) \mod 11 \equiv 7 + 8 \times (5 - 2) \mod 11 \equiv 9$

   得到 $2\alpha = (x_1, -y_1) = (5,2)$，然后计算 3α，由公式知：

   - $m_2 = (y_1 - y_\alpha)(x_1 - x_\alpha)^{-1} \equiv (2 - 7) \times (5 - 2)^{-1} \equiv 2$
   - $x_2 = m_2^2 - x_1 - x_\alpha \equiv 4 - 5 - 2 \mod 11 \equiv 8$
   - $y_2 = y_1 + m_2(x_2 - x_1) \equiv 2 + 2 \times (8 - 5) \equiv 8$

   得到 $3\alpha = (x_2, -y_2) = (8,3)$；

   同理得到 $4\alpha = (10,2)$，$5\alpha = (3,6)$，$6\alpha = (7,9)$，$7\alpha = (7,2)$，$8\alpha = (3,5)$，$9\alpha = (10,9)$，$10\alpha = (8,8)$，$11\alpha = (5,9)$，$12\alpha = (2,4)$，$13\alpha = 0$（无穷远点）；

   由此可知，$n\alpha$ 的集合组成了一个循环子群，而 $\alpha = (2,7)$ 是本原元。

3. 设 $\alpha = (2,7)$ 为基点，使用椭圆曲线上的 ElGamal 算法，完成对明文 $x = (5,2)$（为椭圆曲线上的点）的加解密（随机选择 $k = 3$）。

   加密:

   - 不妨取 $p = 1$ 为私钥，此时公钥 $Y = 1 \cdot \alpha = (2,7)$；
   - 加密时选取 $k = 3$，得到密文第一部分 $C_1 = 3\alpha = (8,3)$；
   - 然后密文的第二部分是 $C_2 = x + k \cdot Y = (5,2) + 3 \times (2,7) \equiv 2\alpha + 3\alpha \equiv 5\alpha \equiv (3,6)$；
   - 故密文为 $(C_1, C_2) = ((8,3),(3,6))$；

   解密:

   - $C_2 - p \cdot C_1 \equiv (3,6) - 1 \cdot (8,3) \equiv 5\alpha - 3\alpha \equiv 2\alpha \equiv (5,2)$

数字签名算法

假设 Alice 使用 ELGamal 签名方案，$p = 31847, \alpha = 5$ 以及 $\beta = 25703$。给定消息 $x = 8990$ 的签名 $(23972, 31396)$ 以及 $x = 31415$ 的签名 $(23972, 20481)$，计算 k 和 a 的值。

- 首先，由于两个签名使用了同一个随机数签名，故我们有:

$$\gamma \equiv \alpha^k \mod p \tag{1}$$

$$\delta_1 \equiv (x_1 - a\gamma)k^{-1} \mod p-1 \tag{2}$$

$$\delta_2 \equiv (x_2 - a\gamma)k^{-1} \mod p-1 \tag{3}$$

- $(2) - (3)$ 得到:

$$\delta_1 - \delta_2 \equiv (x_1 - x_2)k^{-1} \mod p-1$$

- 即:

$$(\delta_1 - \delta_2)k \equiv (x_1 - x_2) \mod p-1$$

- 代入数据有:

$$(31396 - 20481)k \equiv (8990 - 31415) \mod 31846$$

- 解得:

$$k = 1165$$

- 接下来对 (2) 式改写，有:

$$a\gamma \equiv (x_1 - \delta_1 k) \mod p-1 \tag{2}$$

- 代入数据有:

$$23972a \equiv 8990 - 31396 \times 1165 \mod 31846$$

- 解得:

$$a = 7459$$