实验二

(密码学与网络安全课程报告)

姓　　名：肖文韬

学　　号：2020214245

专 业 方 向：电子信息（计算机技术）

邮　　箱：xwt20@mails.tsinghua.edu.cn

二〇二一年四月十日

第 1 章　实验介绍

RSA 加密算法是应用最广泛的公钥加密算法，本次实验实现基于 RSA 算法的加解密以及数字签名功能，包含以下 4 种操作：生成密钥对、公钥加密、私钥解密和数字签名。本次实验中密钥长度为 2048 比特。实验使用 Rust 语言实现 RSA-2048 的密钥生成，加密解密，以及数字签名操作。

实验目的：

1. 熟悉 RSA 公钥加密算法的思路
2. 学习 RSA 实现上的技巧
3. 学习 rust 语言的基本用法

实验平台：

1. rust 语言

第 2 章　实验内容

2.1　生成密钥对

密钥对的生成过程包括选取随机数，对随机数进行素性测试，根据素数 p 和 q 计算 n，随机选择和 n 的欧拉函数互质的 e，计算 e 的逆元 d。选取的大素数 p 和 q 应当满足现有的安全性要求，且至少使用两种不同的算法进行素性测试，请在实验报告中说明你选择参数和算法的安全性以及效率。选取的 e 同样应当满足安全性要求，至少使用两种不同的算法进行计算逆元 d。

2.1.1　素性测试

本实现采用了两种最主流的概率素性测试算法:

1. **Miller-Rabin 测试**, 作为费马定理的扩展，每一轮 MR 测试的伪素数的可能性为 $\frac{1}{4}$，所以 k 轮通过仍然是伪素数的可能性为 4^{-k}。复杂度 $O(k \log^2 n)$ (k 为测试轮数)。
2. **Baillie–PSW 测试**, 结合了 Miller-Rabin 测试和强 Lucas 概率测试。复杂度为 $O(log^2 n)$，低于 MR 测试。

值得一提的是，素性测试的一些优化技巧:

1. 首先测试是否是常用的小素数的倍数，具体来说，RSA-2048 会首先判断随机数是否是自然数中前 384 个素数的倍数。该技巧和 384 的来源参考的是 OpenSSL 最新代码的 `bn_prime.c#L74:12`。
2. 如果当然随机数不是素数，会对随机数 +2 再判断是否是素数，因为按照素数定理，对于 RSA-2048，连续的数中出现素数的概率为 $\frac{1}{\log(2^{2048})} \approx \frac{1}{1418}$。最坏情况下尝试 700 多次就一定会遇到素数。

miller rabin 测试的代码:

```rust
fn miller_rabin_test(rnd: &Integer, iteration: u8, base_2: bool) {
    let n = Integer::from(rnd);
    let mut d: Integer = Integer::from(&n - 1);
    let mut r = 0;
    while !(&d.get_bit(0)) { d >>= 1; r += 1; }
    let mut rng = RandState::new();
    'witness_loop: for _ in 0..iteration {
        let mut n_sub = Integer::from(&n - 2u8);
        let a = if base_2 {
            Integer::from(2)
        } else {
            Integer::from(n_sub.random_below_ref(&mut rng)) + 2u8
        };
        let mut x = a.pow_mod(&d, &n).unwrap();
        n_sub += 1u8;
        if &x == &1 || &x == &n_sub {
            continue 'witness_loop;
        }
        for _ in 1..r {
            x = x.pow_mod(&Integer::from(2), &n).unwrap();
            if &x == &n_sub {
                continue 'witness_loop;
            }
        }
        return false;
    }
    true
}
```

因为囿于篇幅，具体的代码注释参见源码。

2.1.2　模逆运算

本实现中素数 p 和 q 均为 2048 位，是目前主流的 RSA-2048 实现，密钥长度符合安全要求。

对于 e 的选择，过小的 e（例如 3）会存在安全问题，同时短比特长度和小的 Hamming 权重能够使得加密的效率更加高，目前 OpenSSL 以及其他实现广泛采用的是 65537 (0x10001). 本实现参考主流实现，e 的选择也是 65537.

d 作为 e 的模 $\phi(n)$ 逆元，因为 n 为 4096 位，所以 d 的强度也能够得到保证。

本实现共有两种模逆的算法实现：

1. **扩展欧几里得算法**，算法效率 $O(2\log_{10}(\phi(n)))$（除法运算）。gcd 的计算复杂度推导很复杂，具体可以参考 TAOCP。
2. **平方幂算法（binary exponentiation）**，算法效率 $O(\log(\phi(n)))$。但是该算法只适用于 $\phi(n)$ 为素数的情况。

两种模逆的算法实现代码：

```rust
fn modular_inverse(e: &Integer, phi_n: &Integer, method: &str) {
    if method == "extend_gcd" {
        let mut t = Integer::from(0);
        let mut newt = Integer::from(1);
        let mut r = Integer::from(phi_n);
        let mut newr = Integer::from(e);
        let mut quotient = Integer::new();
        let mut tmp = Integer::new();
        while newr.significant_bits() != 0 {
            quotient.assign(&r / &newr);
            tmp.assign(&quotient * &newt);
            tmp *= -1; tmp += &t; t.assign(&newt);
            newt.assign(&tmp);
            tmp.assign(&quotient * &newr);
            tmp *= -1; tmp += &r; r.assign(&newr);
            newr.assign(&tmp);
        }
        if r > 1 { panic!("e is not invertible!"); }
        if t < 0 { t += phi_n; }
        return t;
    } else if method == "binary_exp" {
        if baillie_psw(phi_n, true) {
            println!("phi_n is prime, use binary_exp");
            return Integer::from((&e).pow_mod_ref(
                  &Integer::from(phi_n - 2), &phi_n).unwrap())
        } else {
            return Integer::from((&e).invert_ref(&phi_n).unwrap());
        }
    }
}
```

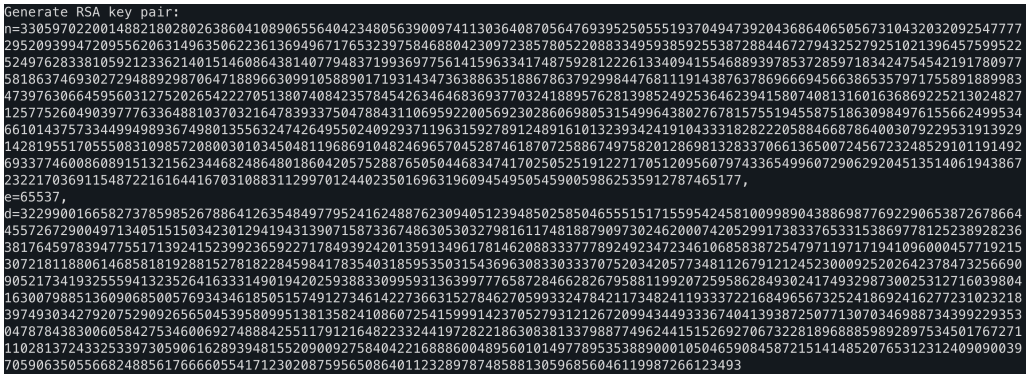

图 2.1　密钥生成运行结果

2.1.3　运行结果

RSA 的实现跟之前 AES 的实现都使用 cargo 组织项目代码。编译运行的代码:

```
cd target/release && cargo build --release && ./crypto
```

密钥生成的 n, e, d 如图 2.1 所示，其中 n 是两个 2048 位随机大素数 p, q 的乘积。

参考文献