readme

倪浚桐

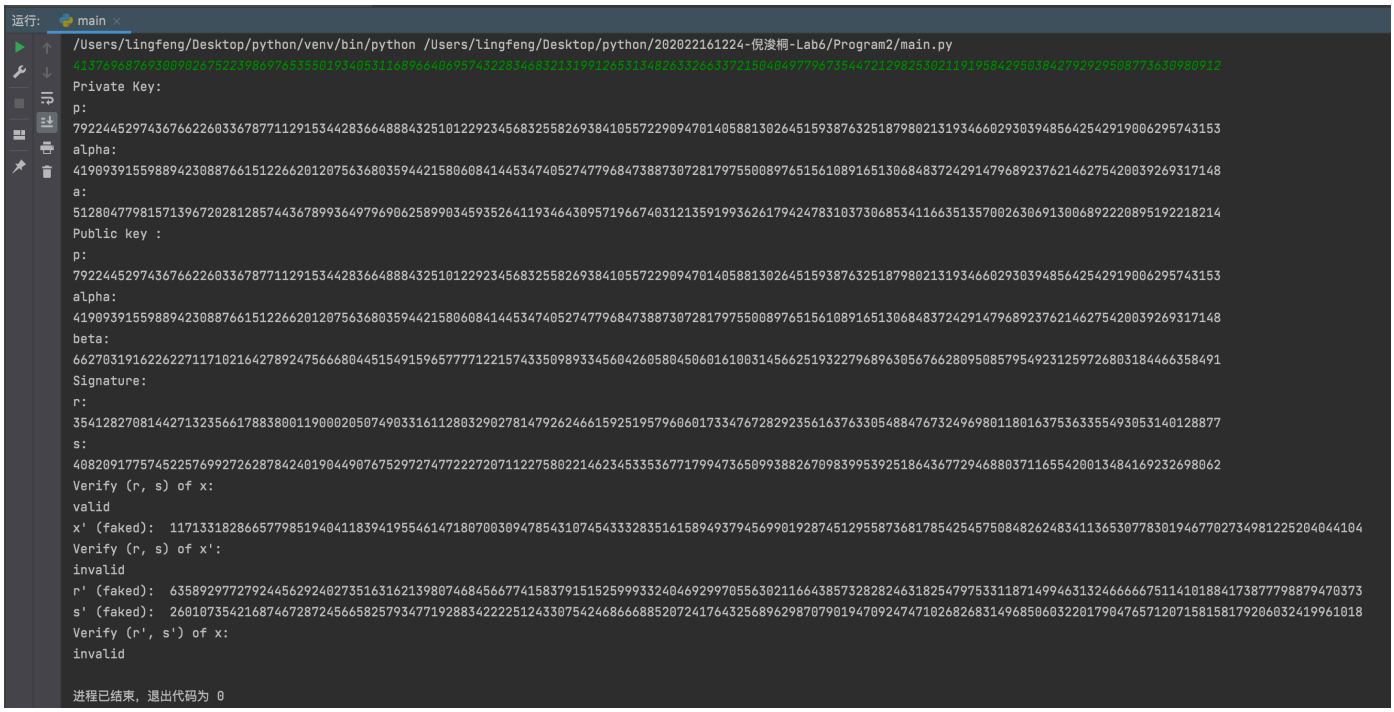
202022161224

Lab6-progarm2

macOS Monterey 12.0.1

Pycharm 11.0.12 x86-64

Python 3.9.5




```python
import random


# 求最大公约数
def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a % b == 0:
        return b
    else:
        return gcd(b, a % b)


# 快速幂+取模
```

```python
def power(a, b, c):
    ans = 1
    while b != 0:
        if b & 1:
            ans = (ans * a) % c
        b >>= 1
        a = (a * a) % c
    return ans


# 卢卡斯-莱曼素性检验
def Lucas_Lehmer(num) -> bool:  # 快速检验pow(2,m)-1是不是素数
    if num == 2:
        return True
    if num % 2 == 0:
        return False
    s = 4
    Mersenne = pow(2, num) - 1  # pow(2, num)-1是梅森数
    for x in range(1, (num - 2) + 1):  # num-2是循环次数，+1表示右区间开
        s = ((s * s) - 2) % Mersenne
    if s == 0:
        return True
    else:
        return False


# 大素数检测
def Miller_Rabin(n):
    a = random.randint(2, n - 2)  # 随机第选取一个a∈[2,n-2]
    # print("随机选取的a=%lld\n"%a)
    s = 0  # s为d中的因子2的幂次数。
    d = n - 1
    while (d & 1) == 0:  # 将d中因子2全部提取出来。
        s += 1
        d >>= 1

    x = power(a, d, n)
    for i in range(s):  # 进行s次二次探测
        newX = power(x, 2, n)
        if newX == 1 and x != 1 and x != n - 1:
            return False  # 用二次定理的逆否命题，此时n确定为合数。
        x = newX

    if x != 1:  # 用费马小定理的逆否命题判断，此时x=a^(n-1) (mod n)，那么n确定为合数。
        return False

    return True  # 用费马小定理的逆命题判断。能经受住考验至此的数，大概率为素数。


```

```python
# 扩展的欧几里得算法，ab=1 (mod m)，得到a在模m下的乘法逆元b
def Extended_Eulid(a: int, m: int) -> int:
    def extended_eulid(a: int, m: int):
        if a == 0:  # 边界条件
            return 1, 0, m
        else:
            x, y, gcd = extended_eulid(m % a, a)  # 递归
            x, y = y, (x - (m // a) * y)  # 递推关系，左端为上层
            return x, y, gcd  # 返回第一层的计算结果。
        # 最终返回的y值即为b在模a下的乘法逆元
        # 若y为复数，则y+a为相应的正数逆元

    n = extended_eulid(a, m)
    if n[1] < 0:
        return n[1] + m
    else:
        return n[1]


# 生成域参数p，长度大约为512bits
def Make_p() -> int:
    a = random.randint(10 ** 150, 10 ** 160)
    while gcd(a, 2) != 1:
        a = random.randint(10 ** 150, 10 ** 160)
    return a


# 生成域参数alpha
def Make_alpha(p: int) -> int:
    return random.randint(2, p)


# 生成一个小于p的素数作为私钥，长度大约为512bits
def Make_private_key(p: int) -> int:
    pri = random.randint(2, p - 2)
    while gcd(pri, p) != 1:
        pri = random.randint(2, p - 2)
    return pri


# 快速幂
def qpow(a: int, b: int) -> int:
    ans = 1
    while b != 0:
        if b & 1:
            ans = ans * a
        b >>= 1
        a = a * a
    return ans
```

```python


def Make_prime(key_size: int) -> int:
    while True:
        num = random.randrange(qpow(2, key_size - 1), qpow(2, key_size))
        if Miller_Rabin(num):
            return num


# 计算签名
def Sign(x, p, alpha, d) -> []:
    temp_key = random.randint(0, p - 2)
    while gcd(temp_key, p - 1) != 1:
        temp_key = random.randint(0, p - 2)
    r = power(alpha, temp_key, p)
    s = (x - d * r) * Extended_Eulid(temp_key, p - 1) % (p - 1)
    return r, s


# 签名验证
def Verify(x, p, alpha, beta, r, s):
    t = (power(beta, r, p) * power(r, s, p)) % p
    if t == power(alpha, x, p):
        return True
    else:
        return False


if __name__ == '__main__':
    x = int(input())
    if type(x) != int:
        raise ValueError("Must be an integer!")

    p = Make_prime(512)
    alpha = Make_alpha(p)
    a = Make_private_key(p)
    beta = power(alpha, a, p)

    r, s = Sign(x, p, alpha, a)
    Valid = Verify(x, p, alpha, beta, r, s)

    r_fake = random.randint(10 ** 150, 10 ** 160)
    s_fake = random.randint(10 ** 150, 10 ** 160)
    x_fake = random.randint(10 ** 150, 10 ** 160)

    print("Private Key: ")
    print("p:")
    print(p)
    print("alpha:")
```

```
    print(alpha)
    print("a:")
    print(a)
    print("Public key : ")
    print("p:")
    print(p)
    print("alpha:")
    print(alpha)
    print("beta:")
    print(beta)
    print("Signature: ")
    print("r:")
    print(r)
    print("s:")
    print(s)
    print("Verify (r, s) of x: ")
    if Verify(x, p, alpha, beta, r, s):
        print("valid")
    else:
        print("invalid")
    print("x' (faked): ", x_fake)
    print("Verify (r, s) of x': ")
    if Verify(x_fake, p, alpha, beta, r_fake, s_fake):
        print("valid")
    else:
        print("invalid")

    print("r' (faked): ", r_fake)
    print("s' (faked): ", s_fake)
    print("Verify (r', s') of x: ")
    if Verify(x, p, alpha, beta, r_fake, s_fake):
        print("valid")
    else:
        print("invalid")

```