Network Security Technology

Tutorial 5, Week 5 (March 23) Due Date: March 30

薛春宇 518021910698

---

1. 随机素数生成算法 (50 points)

(1) 实现PPT《网络安全技术5公钥密码学-2 数学基础》第39页所述的生成随机素数的算法，并生成至少2个32-bit的素数，不需要第三方大整数运算库。（需要学习与实现 perfect power 的判定）

Answer:

首先介绍依照本题要求实现的随机数生成算法，本模块严格按照PPT上所述的生成随机素数的算法进行实现，包含了如下子模块：

- 欧拉 ϕ 函数的计算：

```
# Euler Phi Function
# Input: a number n
# Output: Euler phi function \phi(n)
def euler_phi(n):
    m = int(pow((n + 0.5), 0.5))
    ans = n
    for i in range(2, m, 1):
        if n % i == 0:
            ans = int(ans / i * (i - 1))
        while n % i == 0:
            n /= i
    if n > 1:
        ans = int(ans / n * (n - 1))
    return ans
```

- 欧几里得算法判断两个数是否互素：

```
# Judge whether num1 and num2 are prime to each other, use Euclidean Algorithm
def are_prime_to_each_other(num1, num2):
    A = max(num1, num2)
    B = min(num1, num2)
    while B > 0:
        Remainder = A % B
        A = B
        B = Remainder
    return A == 1
```

- 重复模乘法以简化模指数计算：

```
# Repeated Modular Multiplication
# Input: a^u % n
# Output: the calculated value
def repeated_modular_multiplication(a, u, num):
    ret = a
    for i in range(1, u, 1):
        ret = (ret * a) % num
    return ret
```

- 平方-乘算法以简化模指数计算：

```
# Square and Multiply Algorithm
# Input: a^u % n
# Output: the calculated value
def square_and_multiply_algorithm(a, u, num):
    # Build dictionary
    count = 1
    remainders = {}
    while count <= u:
        if int(count / 2) in remainders.keys():
            remainders[count] = pow(remainders[int(count / 2)], 2) % num
        else:
            # First element
            remainders[count] = a % num
        count *= 2
    # Statistic
    ret = 1
    while count > 0 and u > 0:
        # Too big
        if count > u:
            count /= 2
            continue
        # Fit
        u -= count
        ret = (ret * remainders[count]) % num
```

```
        count /= 2
    return ret
```

上述子模块均集成在 **Utils.py** 文件内，以供 **main.py** 在素数生成与检测过程中的调用。

接下来，我们将基于上述实现的工具函数，介绍在 **main.py** 中实现***Miller-Rabin* 素数检测算法**，以及在此基础上实现生成32位随机素数的方法。

在素数检测算法的实现中，我们首先需要根据公式 $N-1=2^ru$，计算 *r* 和 *u* 的值：

```
# Compute r and u
    r = 0
    tmpNum = num
    while (tmpNum - 1) % 2 == 0:
        r += 1
        tmpNum = int((tmpNum - 1) / 2 + 1)
    u = int(tmpNum - 1)
```

之后，我们基于PPT中伪代码的思想如下实现素数检测算法的主体部分：

```
is_prime = False
    # Test for n times
    for idx in range(0, n, 1):
        # Random seed
        a = num
        while not (are_prime_to_each_other(a, num) and a != 1):
            # Must make sure that a is prime to num, so that we can use the Generation Statement of Euler Theorem
            a = secret_generator.randint(1, num - 1)

        # Time counter
        time_mark = time.process_time()
        # Use Generation Statement of Euler Theorem to simplify the calculation
        # Operate u
        new_u = u % euler_phi(num)
        for i in range(0, r, 1):
            new_pow_u = (pow(2, i) % euler_phi(num)) * new_u
            # Use Modular Exponentiation to simplify the pow calculation
            # Judge
            # if repeated_modular_multiplication(a, new_u, num) == 1 or repeated_modular_multiplication(a, new_pow_u, num) == -1:
            if square_and_multiply_algorithm(a, new_u, num) == 1 or square_and_multiply_algorithm(a, new_pow_u, num) == -1:
                is_prime = True
        print("Test", idx, ": a is set to be", a, "| time spent: ", round(time.process_time() - time_mark, 3), "seconds")
```

这里有几点注意事项：

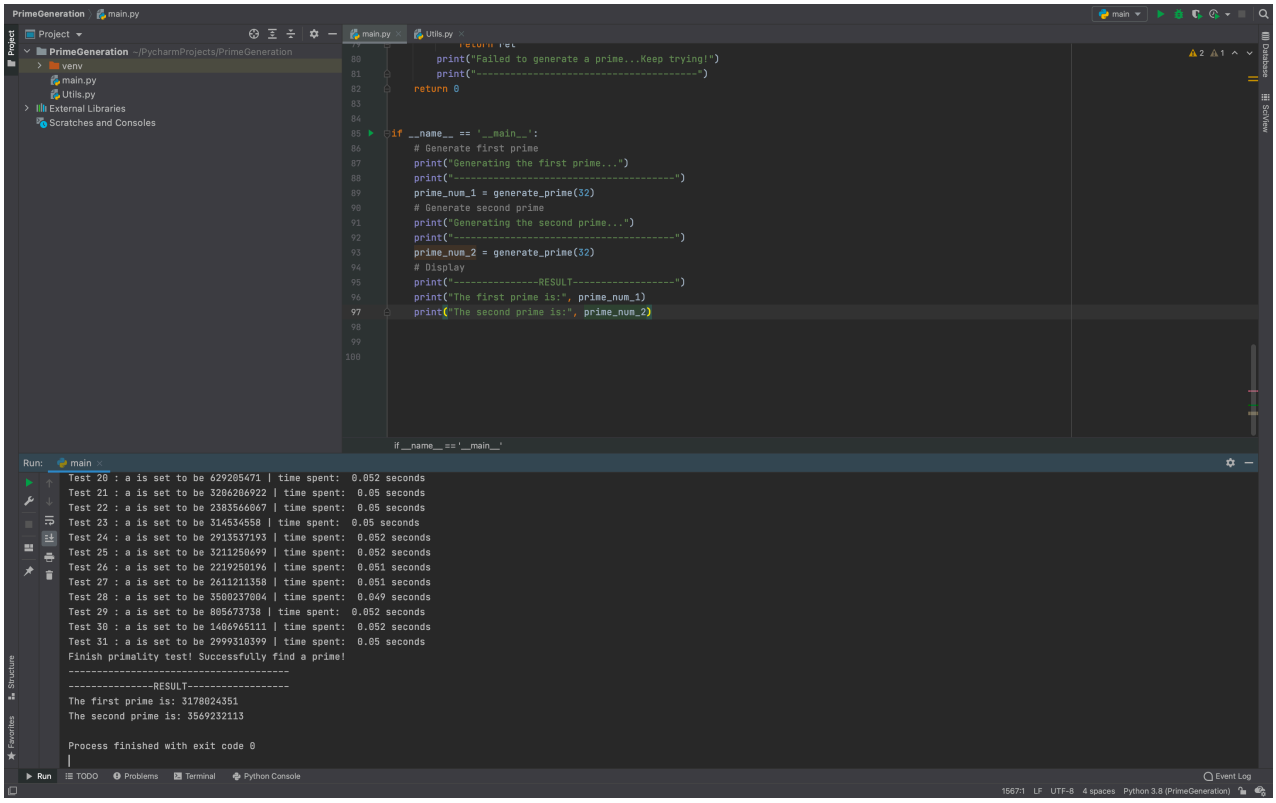
- 在第14行和16行中，我们使用**欧拉定理的推论**："对整数 a 和正整数 k ，n，若 a 和 n 互素，则：$a^k \bmod n = a^{k \bmod \phi(n)} \bmod n$ " 来简化模指数的计算。为了满足上述前提条件，我们使用之前实现的 *are_prime_to_each_other( )* 函数来保证随机生成的 *a* 与被测试的数 *num* 是互素的。
- 整个测试迭代次数与安全参数 *n* 保持一致。
- 我们使用 *python* 中的 *secrets* 扩展包来实现生成一定范围内的随机数（见代码第8行）
- 在第19行中（已被注释），我们使用**重复模乘法**来计算模指数，其效果要远差于在第20行中实现的**平方-乘算法**，且在安全参数达到28以上时，迭代速度已达到不可接受的慢。迭代速度可以从第22行中输出的迭代时间看出。

  接下来，我们介绍基于上述素数检测算法实现的素数生成算法：

```
1  # Generate number
2  # Input: n is the security parameter
3  def generate_prime(n):
4      # Set boarder
5      max_number = get_maximum_number(n - 1)
6      min_number = 0
7
8      # 3n^2 times at most
9      for i in range(0, 3 * pow(n, 2), 1):
10         print("Iteration", i, "for Prime Generation is processing...")
11         # Random algorithm
12         ret = secret_generator.randint(min_number, max_number)
13         # Add 1 at the head of the binary expression
14         ret += pow(2, n - 1)
15         print("Successfully generate random seed:", ret)
16
17         if primality_test(ret, n):
18             print("----------------------------------------")
19             return ret
20         print("Failed to generate a prime...Keep trying!")
21         print("----------------------------------------")
22     return 0
```

其中，*get_maximum_number(n)* 是在 **Utils.py** 中自主实现的用来获取指定 *n-bit* 能达到的最大整数，以为随机整数的生成设置上界。根据PPT中所述，我们最多重复生成 $3n^2$ 次素数，即可以很大概率获得一个素数。第17行便是调用我们之前实现的素数检测函数。

进行的一次素数生成实验结果如下：



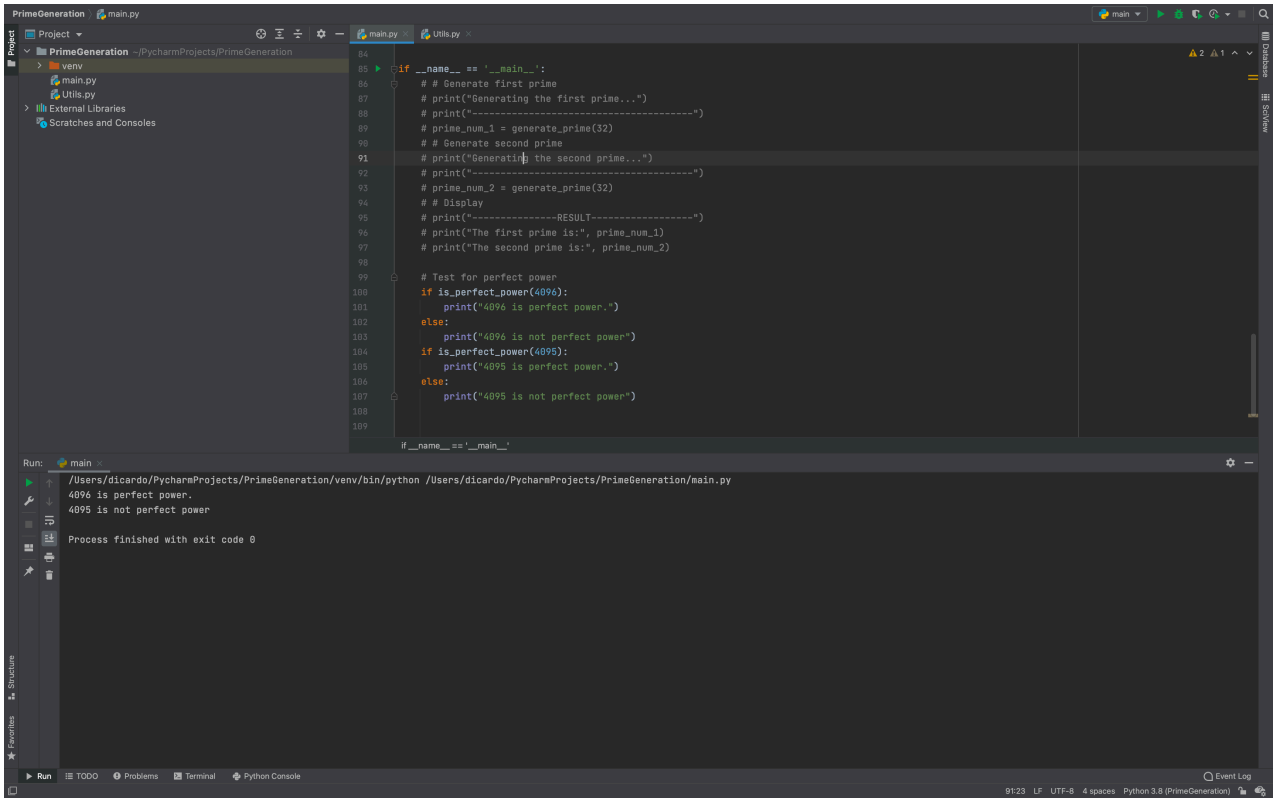
可以看到，我们的算法最终输出了3178024351和3569232113作为生成的两个大素数。为了检验算法的正确性，我们使用网上提供的素数判别器，对这两个素数的正确性进行验证：

根据上述结果，我们得以验证该素数生成算法的正确性。

此外，我们还实现了判断 *perfect power* 的函数：

```
# Judge whether num is a perfect power
def is_perfect_power(num):
    s = int(pow(num, 0.5))
    for i in range(2, s + 1, 1):
        k = 2
        while pow(i, k) < num:
            k += 1
        if pow(i, k) == num:
            return True
    return False
```

运行效果如下所示（成功判断4096是perfect power，而4095不是）：



---

(2) 学习开源库中已有的素数生成算法，撰写报告，阐述比我们讲的道理、比你的实现更加优化的地方。

Answer:

本次学习的对象是**大整数运算库GMP (GNU高精度算术运算库，GNU Multiple Precision Arithmetic Library)**。通过阅读其关于素数生成的核心源码可以得出如下结论：

- GMP中也使用了***Miller-Rabin* 素数检测算法**，其核心思想与本项目的实现基本一致。
- 相较于本项目的实现，GMP的核心优化方式在于针对素数强伪证的判定，增大了素数生成的能力；此外，通过调用硬件指令，GMP的程序运行速度要远比本项目中用python实现的程序要快。

---

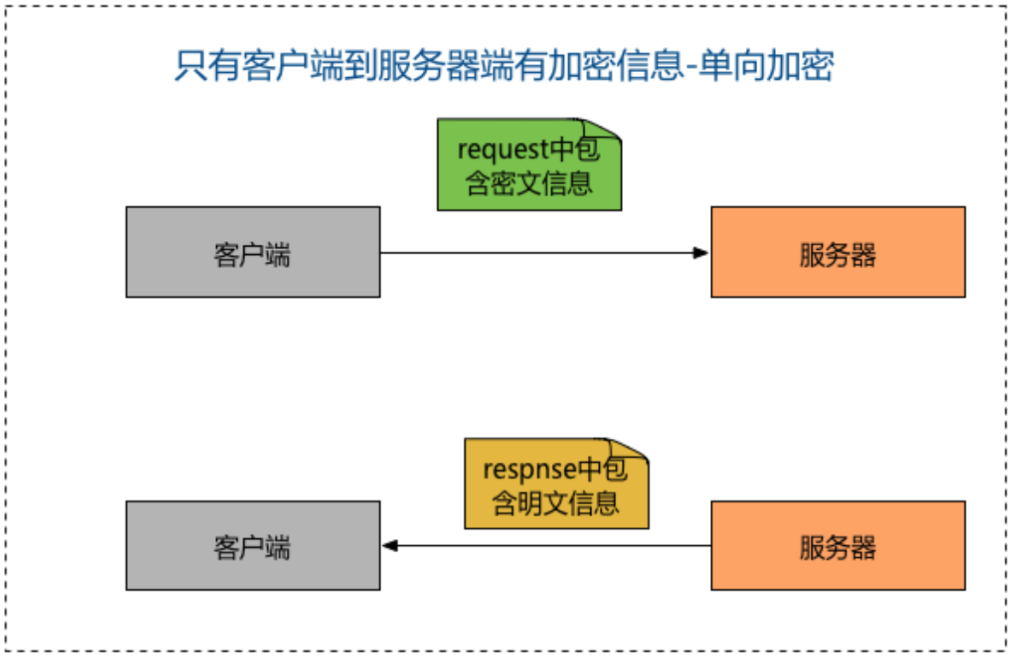
2. RSA算法 (50 points)

检索和阅读文献，写一篇简单的 survey，包括历史上提出的一些要得到实际中可以安全使用的 RSA 加密的尝试，以及目前产业界在实际使用的基于 RSA 的公钥加密方案。给出其中各方案的具体算法、优缺点、解决了的问题、存在的问题等。

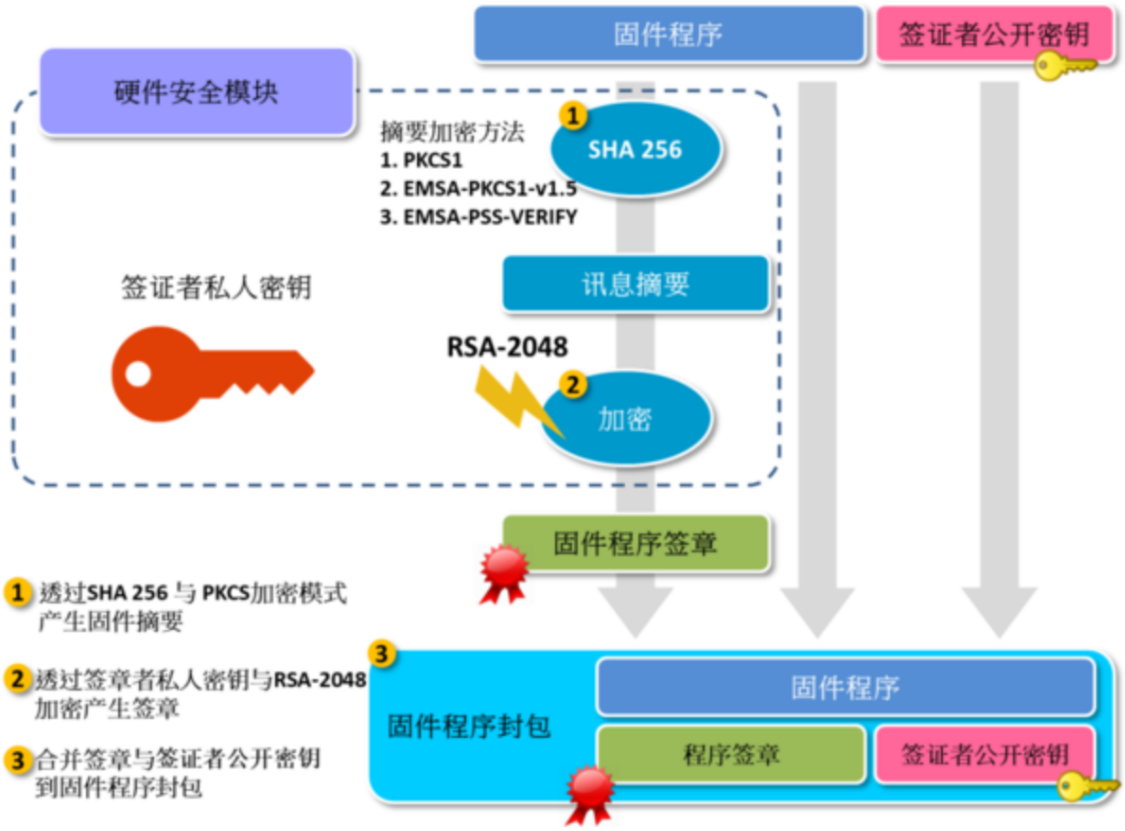
Answer:

- 历史上提出的RSA安全加密的尝试：**客户端与服务端的交互**
  - 具体算法：
    - 使用RSA工具生成公钥 - 私钥对，把私钥分发给客户端程序；

        - 客户端程序对参数进行MD5加密；
        - RSA对MD5值进行加密；
        - 客户端把请求参数发送到服务器端；
        - 服务器把MD5数据解密还原；
        - 服务器端对明文参数重复做一次MD5加密；
        - 比较客户端和服务器端的MD5值是否一致，若不一致则认为访问无效



    - 优点：
        - 使用MD5和RSA两种方案进行了双重加密，保证了数据传输的安全性
    - 缺点：
        - 产生密钥花费较大，且由于素数产生技术的束缚，因此很难做到一次一密
        - 分组长度太大，速度较慢
    - 解决的问题：
        - 主要解决了客户端与服务器端在数据交互时的安全性保障问题
    - 存在的问题：
        - 加密数据传输效率不高，分组长度难以控制
- 目前产业界使用RSA公钥加密方案的实例：**合肥“兆芯”RSA加密提高信息安全，防范SSD后门**
    - 具体算法：**固件程序信息加密认证**



固件程序信息加密认证

- 优点：
  - 使用RSA-2048算法进行加密，极大保证了加密程序的安全性
  - 与多种其他加密算法混合使用，具有很强的安全性
- 缺点：
  - 密钥生成算法花费较大，速度较慢
- 解决的问题：
  - 主要解决了SSD固态硬盘中固件认证的数字签章算法的实现
- 存在的问题：
  - 对相关配套硬件设备的要求较高，普及程度有待提高