

# 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  $\phi$  函数的计算：

```
1  # Euler Phi Function
2  # Input: a number n
3  # Output: Euler phi function \phi(n)
4  def euler_phi(n):
5      m = int(pow((n + 0.5), 0.5))
6      ans = n
7      for i in range(2, m, 1):
8          if n % i == 0:
9              ans = int(ans / i * (i - 1))
10             while n % i == 0:
11                 n /= i
12         if n > 1:
13             ans = int(ans / n * (n - 1))
14         return ans
```

- 欧几里得算法判断两个数是否互素：

```

1  # Judge whether num1 and num2 are prime to each other, use Euclidean
    Algorithm
2  def are_prime_to_each_other(num1, num2):
3      A = max(num1, num2)
4      B = min(num1, num2)
5      while B > 0:
6          Remainder = A % B
7          A = B
8          B = Remainder
9      return A == 1

```

- 重复模乘法以简化模指数计算:

```

1  # Repeated Modular Multiplication
2  # Input:  $a^u \pmod n$ 
3  # Output: the calculated value
4  def repeated_modular_multiplication(a, u, num):
5      ret = a
6      for i in range(1, u, 1):
7          ret = (ret * a) % num
8      return ret

```

- 平方-乘算法以简化模指数计算:

```

1  # Square and Multiply Algorithm
2  # Input:  $a^u \pmod n$ 
3  # Output: the calculated value
4  def square_and_multiply_algorithm(a, u, num):
5      # Build dictionary
6      count = 1
7      remainders = {}
8      while count <= u:
9          if int(count / 2) in remainders.keys():
10             remainders[count] = pow(remainders[int(count / 2)], 2) % num
11          else:
12             # First element
13             remainders[count] = a % num
14             count *= 2
15      # Statistic
16      ret = 1
17      while count > 0 and u > 0:
18          # Too big
19          if count > u:
20             count /= 2
21             continue
22          # Fit
23          u -= count
24          ret = (ret * remainders[count]) % num

```

```

25         count /= 2
26     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  $N - 1 = 2^r u$ ，计算  $r$  和  $u$  的值：

```

1  # Compute r and u
2      r = 0
3      tmpNum = num
4      while (tmpNum - 1) % 2 == 0:
5          r += 1
6          tmpNum = int((tmpNum - 1) / 2 + 1)
7      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：

```

1  is_prime = False
2      # Test for n times
3      for idx in range(0, n, 1):
4          # Random seed
5          a = num
6          while not (are_prime_to_each_other(a, num) and a != 1):
7              # Must make sure that a is prime to num, so that we can use
the Generation Statement of Euler Theorem
8              a = secret_generator.randint(1, num - 1)
9
10         # Time counter
11         time_mark = time.process_time()
12         # Use Generation Statement of Euler Theorem to simplify the
calculation
13         # Operate u
14         new_u = u % euler_phi(num)
15         for i in range(0, r, 1):
16             new_pow_u = (pow(2, i) % euler_phi(num)) * new_u
17             # Use Modular Exponentiation to simplify the pow calculation
18             # Judge
19             # if repeated_modular_multiplication(a, new_u, num) == 1 or
repeated_modular_multiplication(a, new_pow_u, num) == -1:
20                 if square_and_multiply_algorithm(a, new_u, num) == 1 or
square_and_multiply_algorithm(a, new_pow_u, num) == -1:
21                     is_prime = True
22                 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：

```
PrimeGeneration main.py
Project
  PrimeGeneration --(PycharmProjects/PrimeGeneration)
    venv
      main.py
      Utils.py
    External Libraries
    Scratches and Consoles

main.py
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

def generate_prime(n):
    while True:
        if is_prime(n):
            return n
        n += 1
    print("Failed to generate a prime...Keep trying!")
    print("-----")
    return 0

if __name__ == '__main__':
    # Generate first prime
    print("Generating the first prime...")
    print("-----")
    prime_num_1 = generate_prime(32)
    # Generate second prime
    print("Generating the second prime...")
    print("-----")
    prime_num_2 = generate_prime(32)
    # Display
    print("-----RESULT-----")
    print("The first prime is:", prime_num_1)
    print("The second prime is:", prime_num_2)

if __name__ == '__main__':
```

Run: main

```
Test 20 : a is set to be 629205471 | time spent: 0.052 seconds
Test 21 : a is set to be 3206200922 | time spent: 0.05 seconds
Test 22 : a is set to be 2383540047 | time spent: 0.05 seconds
Test 23 : a is set to be 314534558 | time spent: 0.05 seconds
Test 24 : a is set to be 2913537193 | time spent: 0.052 seconds
Test 25 : a is set to be 3211250699 | time spent: 0.052 seconds
Test 26 : a is set to be 2219250196 | time spent: 0.051 seconds
Test 27 : a is set to be 2611211358 | time spent: 0.051 seconds
Test 28 : a is set to be 3500237004 | time spent: 0.049 seconds
Test 29 : a is set to be 805673738 | time spent: 0.052 seconds
Test 30 : a is set to be 1406965111 | time spent: 0.052 seconds
Test 31 : a is set to be 2999310399 | time spent: 0.05 seconds
Finish primality test! Successfully find a prime!

-----RESULT-----
The first prime is: 3178024351
The second prime is: 3569232113

Process finished with exit code 0
```

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：

osgeo.cn

OSGeo中国 文档 资源 地图 数据 计算 教程 专题 公众号 用户

EN

当前位置： 计算 / 代数计算 / 在线判断质数（素数）

在线判断质数（素数）

分类： 代数计算 更新时间: 2021-01-26 Help edit

请输入数字:

3178024351

计算 还原

3178024351 是质数



PayPal

广告 一个账户，收款全球。0费用开户，享卖家保障，赢逾2亿用户。

PayPal

打开

APP说明

质数(prime number)又称素数。有无限个。一个大于1的自然数，除了1和它本身外，不能被其他自然数整除，换句话说就是该数除了1和它本身以外不再其他的因数；否则称为合数。

根据算术基本定理，每一个比1大的整数，要么本身是一个质数，要么可以写成一系列质数的乘积；而且如果不考虑这些质数在乘积中的顺序，那么写出来的形式是唯一的。最小的质数是2。

使用示例

请输入数字:55

点击“计算”，输出结果

55 它不是质数因为它可以被另一个数整除5。

评价

★★★★★ Five Stars

为了方便学习，提供离线版本的计算工具下载。

请输入验证码查看下载方式:

输入验证码

查看下载方式

关注本站微信公众号，回复“s2711”，获取验证码。



在微信里搜索“开源集思”或微信扫描下方二维码关注微信公众号。

相关App

两整数之间素数（质数）在线计算器

计算分类

代数计算 几何计算 三角函数计算 概率统计

OSGeo中国文档资源地图数据计算教程专题

EN

当前位置: 计算 / 代数计算 / 在线判断质数 (素数)

在线判断质数 (素数)

分类: 代数计算 更新时间: 2021-01-26 [Help](#) [Edit](#)

请输入数字:

3569232113

计算 还原

3569232113 是质数



PayPal  
广告 一个账户，收款全球。0费用开户，享卖家保障，赢逾2亿用户。  
注册PayPal企业账户 打开

APP说明

质数(prime number)又称素数，有无限个。一个大于1的自然数，除了1和它本身外，不能被其他自然数整除，换句话说就是该数除了1 和它本身以外不再有其他因数；否则称为合数。

根据算术基本定理，每一个比1大的整数，要么本身是一个质数，要么可以写成一系列质数的乘积；而且如果不考虑这些质数在乘积中的顺序，那么写出来的形式是唯一的。最小的质数是2。

使用示例

请输入数字:55

点击“计算”，输出结果

55 它不是质数因为它可以被另一个数整除5。

评价

★★★★★ [Free Stars](#)

为了方便学习，提供离线版本的计算工具下载。

请输入验证码查看下载方式:

输入验证码

查看下载方式

关注本站微信公众号，回复 **'s2711'**，获取验证码。



在微信里搜索 **'开源集思'** 或微信扫描上方二维码关注微信公众号。

相关App

两整数之间素数 (质数) 在线计算器

计算分类

代数计算 几何计算 三角函数计算 概率统计

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：

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

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

```
84
85 if __name__ == '__main__':
86     # Generate first prime
87     # print("Generating the first prime...")
88     # print("-----")
89     # prime_num_1 = generate_prime(32)
90
91     # Generate second prime
92     # print("Generating the second prime...")
93     # print("-----")
94     # prime_num_2 = generate_prime(32)
95     # # Display
96     # print("-----RESULT-----")
97     # print("The first prime is:", prime_num_1)
98     # print("The second prime is:", prime_num_2)
99
100     # Test for perfect power
101     if is_perfect_power(4896):
102         print("4896 is perfect power.")
103     else:
104         print("4896 is not perfect power")
105     if is_perfect_power(4895):
106         print("4895 is perfect power.")
107     else:
108         print("4895 is not perfect power")
109
110 if __name__ == '__main__':
```

Run: main

```
/Users/dicardo/PycharmProjects/PrimeGeneration/venv/bin/python /Users/dicardo/PycharmProjects/PrimeGeneration/main.py
4896 is perfect power.
4895 is not perfect power
Process finished with exit code 0
```

(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。

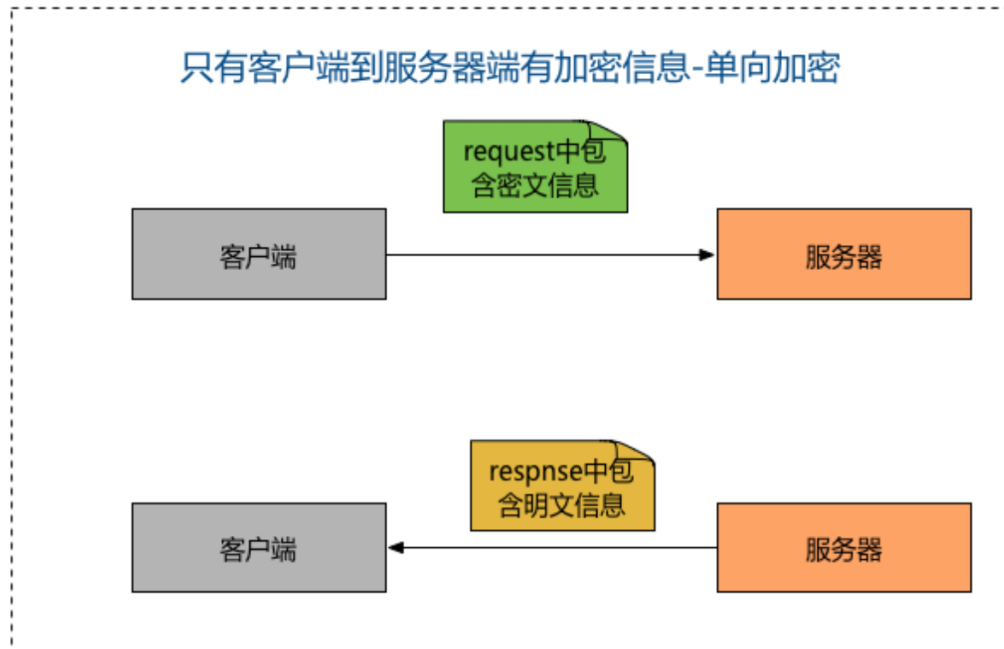
## 2. RSA算法 (50 points)

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:

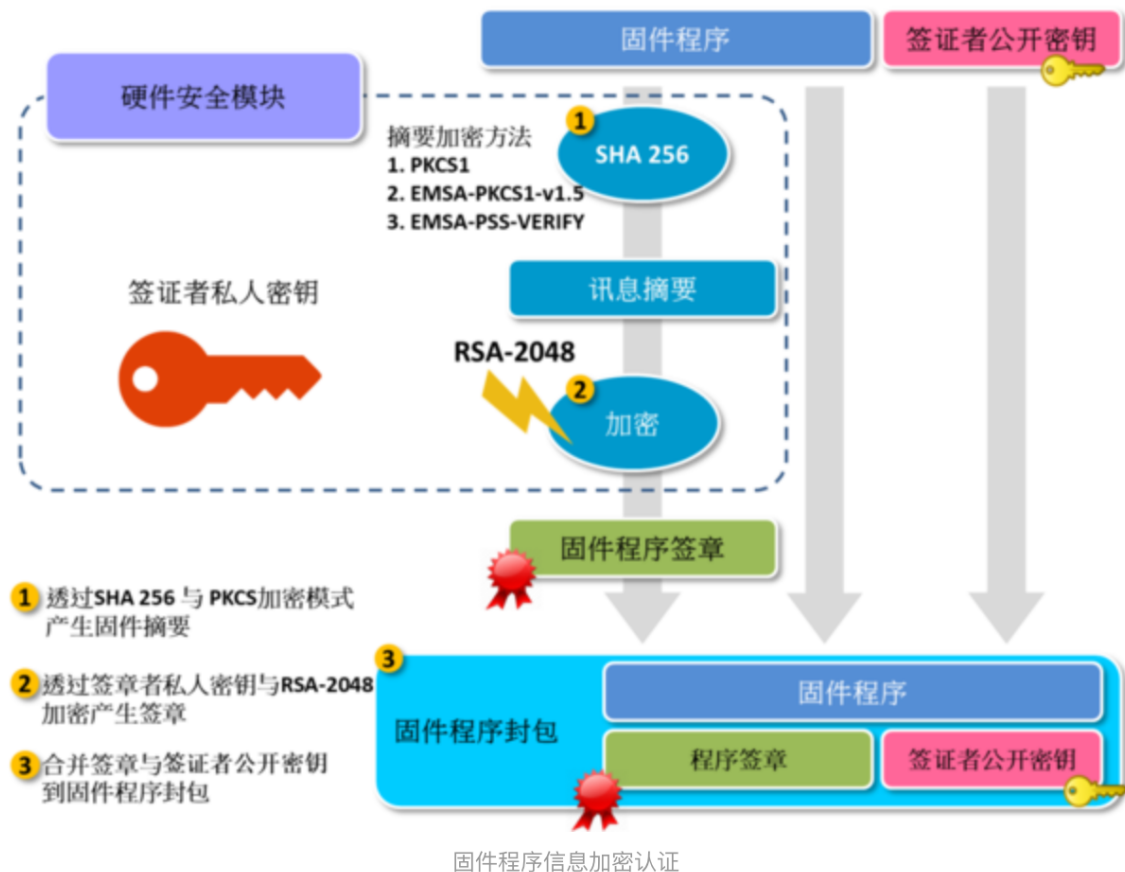
- 历史上提出的RSA安全加密的尝试：客户端与服务端的交互
  - 具体算法：
    - 使用RSA工具生成公钥 - 私钥对，把私钥分发给客户端程序；

- 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





- 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

