|  |
| --- |
| **Github（或者Coding）账号：observer-297** |
| **个人博客关于密码学实验的链接：** [**observer-297/cryptography**](https://github.com/observer-297/cryptography/tree/master) |
| **实验题目（中文）：**   1. **欧拉计划 问题182:** 2. **实用RSA实现** |
| **实验摘要（中文）：**  实验1：欧拉计划问题182  在RSA加密中，选择合适的公钥指数 e 是非常重要的。在选择 𝑒 时，我们希望最小化未加密信息的数量。问题182要求在给定两个素数𝑝=1009p=1009 和 𝑞=3643，q=3643 的情况下，找到所有满足 1<e<ϕ(n) 且 gcd(𝑒,𝜙(𝑛))=1的  𝑒，并且此时未加密信息的数目最小。最后，要求将所有这些 e 的总和求出。  实验2：实用RSA实现  实现一个简化的RSA加密和解密系统，主要关注密钥生成、加密和解密流程。通过这个实验，我们将实践RSA算法的核心部分，包括生成公钥和私钥、加密和解密整数。此外，还需要实现字符串的加密和解密。 |
| **题目描述（清楚描述题目中文，写出自己的理解，请勿复制原题目）**  **1．欧拉计划 问题182:**  **RSA加密基于以下流程：**  **生成两个不同的素数p和q。计算n=pq以及φ=(p-1)(q-1)。**  **找到整数e，满足1<e<φ，且gcd(e,φ)=1。**  **RSA系统能加密的信息是区间[0,n-1]中的整数。**  **因此，需要加密的文本首先需要转换成可加密的信息（即区间[0,n-1]内的某个整数）。**  **加密文本时，如果文本转换成的信息是m，则加密为c=me mod n。**  **解密文本时，需要以下流程：计算d满足ed=1 mod φ，如果加密的信息是c，则解密为m=cd mod n。**  **存在某些e和m使得me mod n=m。**  **我们称满足me mod n=m的这些m为未加密信息。**  **在选择e时，很重要的一点就是不应该有太多未加密信息。**  **例如，令p=19，q=37。**  **然后n=1937=703，φ=1836=648。**  **如果我们选择e=181，那么尽管gcd(181,648)=1，但是所有可能的信息m (0≤m≤n-1)都是未加密信息，只要我们计算一下me mod n就能发现。**  **对于任意的e，总是存在一些未加密信息。**  **使得未加密信息的数目为最小值是很重要的。**  **现在我们选择p=1009，q=3643。**  **找出所有e，满足1<e<φ(1009,3643)且gcd(e,φ)=1，并且此时未加密信息的数目为最小值。求出所有这些e的和。**  **实际上通过实验一我们需要明确e选取的原则，并计算出题设条件下可用的e。**  **2.实用RSA实现**  **实施 RSA 有两件令人讨厌的事情。他们俩 涉及密钥生成;RSA 中的实际加密/解密是 琐碎。**  **首先，您需要生成随机素数。**  **第二个是你需要一个 “invmod” 运算（乘法 inverse），它不是连接到 语言。算法只有几行，但我总是丢失一个 小时让它开始工作。**  **我建议你不要费心去 primegen，但要花时间去买 你自己的 EGCD 和 invmod 算法工作。**  **现在：**  **• 生成 2 个随机素数。我们将使用小数字开始，以便您 可以从 Prime 表中挑选他们。称它们为“p”和“q”。**  **• 设 n 为 p \* q。您的 RSA 数学取模 n。**  **• 设 et 为 （p-1）\*（q-1） （“totient”）。您只需要此值 Keygen 的 Key Gen 中。**  **• 设 e 为 3。**  **• 计算 d = invmod（e， et）.invmod（17， 3120） 是 2753。**  **• 您的公钥是 [e， n]。您的私钥是 [d， n]。**  **• 加密：c = me%n。解密：m = cd%n**  **• 用一个数字来测试一下，比如 “42”。**  **• 用 bignum 素数重复（保持 e=3）。**  **最后，要加密字符串，请执行一些俗气的操作，例如将 string 设置为 hex 并在其前面加上 “0x” 以将其转换为 数。数学并不在乎你给它喂字符串有多愚蠢。** |
| **过程（包括背景，原理：必要的公式，图表；步骤，如有必要画出流程图，给出主要实现步骤代码）**   1. **欧拉计划 问题182**   **Step1：计算n和n的欧拉函数,𝑛=𝑝×𝑞，其中 𝑝=1009 和 q=3643.ϕ(n)=(p−1)×(q−1) 是欧拉函数，用于密钥生成。**  **Step2：遍历 e 值并计算未加密信息的数量,对于所有的1<e< ϕ(n),我们需要检查每个e的未加密信息数量。未加密信息指的是满足m的e次幂在mod n意义下仍为m的m。**  **Step3：求出所有e值的总和。**  **import math**  **# 主函数，计算最终的结果**  **def compute\_final\_sum():**  **prime\_p = 1009**  **prime\_q = 3643**    **# 计算欧拉函数 φ(n) = (prime\_p - 1) \* (prime\_q - 1)**  **totient\_n = (prime\_p - 1) \* (prime\_q - 1)**    **# 获取 prime\_p 和 prime\_q 对应的所有未隐藏消息数量**  **unconcealed\_count\_p = calculate\_unconcealed\_for\_all\_exponents(prime\_p)**  **unconcealed\_count\_q = calculate\_unconcealed\_for\_all\_exponents(prime\_q)**  **# 找到 prime\_p 和 prime\_q 对应的最小未隐藏消息数量**  **min\_unconcealed\_p = min(unconcealed\_count\_p)**  **min\_unconcealed\_q = min(unconcealed\_count\_q)**  **# 计算满足条件的所有 e 的总和**  **total\_sum = sum(exponent for exponent in range(totient\_n)**  **if unconcealed\_count\_p[exponent % (prime\_p - 1)] == min\_unconcealed\_p and**  **unconcealed\_count\_q[exponent % (prime\_q - 1)] == min\_unconcealed\_q)**    **return str(total\_sum)**  **# 计算给定素数 prime 下所有可能的未隐藏消息数量**  **def calculate\_unconcealed\_for\_all\_exponents(prime):**  **unconcealed\_counts = []**  **for exponent in range(prime - 1):**  **# 只有当 exponent 与 (prime - 1) 互质时，才计算未隐藏消息数量**  **if math.gcd(exponent, prime - 1) == 1:**  **unconcealed\_counts.append(count\_unconcealed\_messages(prime, exponent))**  **else:**  **unconcealed\_counts.append(10\*\*20)**  **return unconcealed\_counts**  **# 计算给定模数 modulus 和指数 exponent 的未隐藏消息数量**  **def count\_unconcealed\_messages(modulus, exponent):**  **unconcealed\_message\_count = 0**  **for message in range(modulus):**  **# 如果 message^exponent ≡ message (mod modulus)，则 message 是未隐藏的**  **if pow(message, exponent, modulus) == message:**  **unconcealed\_message\_count += 1**  **return unconcealed\_message\_count**  **if \_\_name\_\_ == "\_\_main\_\_":**  **print(compute\_final\_sum())**  **运行结果**    **2.实用RSA实现**  **Step1生成两个素数 𝑝 和𝑞：**  **我们可以使用小素数开始，例如 𝑝=61 和 𝑞=53。稍后，我们可以使用更大的素数来增强安全性。**  **Step2计算 𝑛 和 𝜙(𝑛)**  **ϕ(n)：𝑛=𝑝×𝑞 是模数，RSA加密和解密操作都在模 𝑛 的范围内执行。𝜙(𝑛)=(𝑝−1)×(𝑞−1)，这是公钥生成过程中需要的值。**  **Step3选择公钥指数 𝑒**  **通常选择 𝑒=3，因为它是一个小的、常用的公钥指数，并且容易计算。（但是这里的e和 𝜙(𝑛)并不互素，所以取e=7）**  **Step4计算私钥指数 𝑑**  **私钥指数 d 通过求 𝑒在模 𝜙(𝑛)下的乘法逆元来计算，即 𝑑=invmod(𝑒,𝜙(𝑛))。**  **Step5加密和解密过程：**  **加密：对于明文 𝑚，加密后的密文为 𝑐=𝑚^𝑒 mod 𝑛。**  **解密：对于密文 𝑐，解密后的明文为 𝑚=𝑐^d mod 𝑛**  **Step6处理字符串加密：**  **将字符串转换为16进制表示，然后将其转换为整数进行加密。解密后，再将整数转换回字符串。**  **import random**  **# Step 4: Extended Euclidean Algorithm (EGCD) and Modular Inverse**  **def egcd(a, b):**  **""" Return g, x, y such that a\*x + b\*y = g = gcd(a, b) """**  **if a == 0:**  **return b, 0, 1**  **else:**  **g, x, y = egcd(b % a, a)**  **return g, y - (b // a) \* x, x**  **def modinv(a, m):**  **""" Return the modular inverse of a modulo m """**  **g, x, \_ = egcd(a, m)**  **if g != 1:**  **raise ValueError('Modular inverse does not exist')**  **else:**  **return x % m**  **# Step 6: RSA Encryption and Decryption**  **def rsa\_encrypt(m, e, n):**  **return pow(m, e, n)**  **def rsa\_decrypt(c, d, n):**  **return pow(c, d, n)**  **# Example with small primes**  **p = 61**  **q = 53**  **# Step 2: Compute n and totient**  **n = p \* q**  **totient = (p - 1) \* (q - 1)**  **# Step 3: Public exponent e**  **e = 7**  **# Step 4: Compute private exponent d**  **d = modinv(e, totient)**  **# Print public and private keys**  **print(f"Public Key: (e={e}, n={n})")**  **print(f"Private Key: (d={d}, n={n})")**  **# Step 5: Encrypt and Decrypt a message**  **message = 42**  **print(f"\nOriginal Message: {message}")**  **# Encrypt the message**  **ciphertext = rsa\_encrypt(message, e, n)**  **print(f"Encrypted Ciphertext: {ciphertext}")**  **# Decrypt the message**  **decrypted\_message = rsa\_decrypt(ciphertext, d, n)**  **print(f"Decrypted Message: {decrypted\_message}")**  **# Now test with larger primes using the same e = 3**  **def generate\_prime(bits=512):**  **""" Generate a random prime number of the given bit size """**  **while True:**  **prime\_candidate = random.getrandbits(bits)**  **# Ensure it's odd and greater than 1**  **prime\_candidate |= (1 << bits - 1) | 1**  **if is\_prime(prime\_candidate):**  **return prime\_candidate**  **def is\_prime(n, k=5):  # number of tests = k**  **""" Test if a number is prime using Miller-Rabin Primality Test """**  **if n <= 1:**  **return False**  **if n <= 3:**  **return True**  **if n % 2 == 0:**  **return False**  **# Write n-1 as 2^r \* d by factoring powers of 2 from n-1**  **r, d = 0, n - 1**  **while d % 2 == 0:**  **r += 1**  **d //= 2**  **# Miller-Rabin test**  **def miller\_test(a):**  **x = pow(a, d, n)**  **if x == 1 or x == n - 1:**  **return True**  **for \_ in range(r - 1):**  **x = pow(x, 2, n)**  **if x == n - 1:**  **return True**  **return False**  **for \_ in range(k):**  **a = random.randrange(2, n - 1)**  **if not miller\_test(a):**  **return False**  **return True**  **# Generate two large primes**  **p\_large = generate\_prime(512)**  **q\_large = generate\_prime(512)**  **# Compute n and totient for large primes**  **n\_large = p\_large \* q\_large**  **totient\_large = (p\_large - 1) \* (q\_large - 1)**  **# Compute private key exponent d for large primes**  **d\_large = modinv(e, totient\_large)**  **# Print new public and private keys**  **print(f"\nPublic Key (large primes): (e={e}, n={n\_large})")**  **print(f"Private Key (large primes): (d={d\_large}, n={n\_large})")**  **# Encrypt and Decrypt a larger message (string to number conversion)**  **message\_str = "Hello, RSA!"**  **message\_int = int(message\_str.encode('utf-8').hex(), 16)**  **# Encrypt the large message**  **ciphertext\_large = rsa\_encrypt(message\_int, e, n\_large)**  **print(f"Encrypted Ciphertext (large): {ciphertext\_large}")**  **# Decrypt the large message**  **decrypted\_message\_large = rsa\_decrypt(ciphertext\_large, d\_large, n\_large)**  **# Convert decrypted message back to string**  **decrypted\_message\_str = bytes.fromhex(hex(decrypted\_message\_large)[2:]).decode('utf-8')**  **print(f"Decrypted Message (large): {decrypted\_message\_str}")**  **运行结果** |
| **总结（完成心得与其它，主要自己碰到的问题和解决问题的方法）**  **遇到的问题**  **素数生成:**  **在手动选择素数时，确保选择的素数足够大，以确保安全性是一个挑战。我最初选择的素数较小，导致后续测试中容易被破解。**  **解决方法:**  **使用更大的随机素数，或利用现成的素数库，确保选择的素数在安全范围内。**  **扩展欧几里得算法（invmod）:**  **我在实现乘法逆元时遇到了困难，特别是在处理较大数时，计算效率低下、代码逻辑不清晰。**  **解决方法:**  **查阅更多关于扩展欧几里得算法的资料，改进了算法实现，使其更高效且易于理解。通过使用递归和迭代的结合，提高了代码的可读性和效率。**  **未加密信息的识别:**  **理解未加密信息的定义和计算方法较为复杂，尤其是在选择不同的𝑒时如何影响未加密信息的数量。**  **解决方法:**  **系统地分析了不同 𝑒值下未加密信息的数量，通过编写程序进行模拟，获得了直观的结果，帮助我更好地理解这一概念。**  **收获与反思**  **理论知识的学习与实践中的应用是相辅相成的。通过实际实现RSA算法，我对其工作原理有了更深刻的理解。**  **在实现算法的过程中，我的编程技能得到了提升，尤其是在算法优化和调试方面。**  **面对各种技术难点，我学会了如何更系统地分析问题，并寻找解决方案，这对我未来的学习和工作都非常有帮助。**  **理解RSA算法的安全性及其潜在风险，使我更加重视信息安全，意识到在实际应用中保护数据的重要性。** |
| **参考文献（包括参考的书籍，论文，URL等，很重要）**  [**Python的random（随机数）模块的使用\_pythonrandom随机数的用法-CSDN博客**](https://blog.csdn.net/qq_53810226/article/details/138687123)  [**用Python实现进制转换，这一篇教程就够了 - 知乎**](https://zhuanlan.zhihu.com/p/266777426) |