

整数分解与 RSA 的安全性

任彦冰

(西安电子科技大学网络与信息安全学院, 陕西 西安 710071)

摘要: 提出了 3 种大整数因数分解的方法, 并通过这些方法对 RSA 密码算法的安全性做出了界定。根据分析得知, 如果不考虑 RSA 中生成密钥的 2 个素数之间的差值, 单纯地增加 2 个素数的取值对提高安全性很可能是无效的。最后, 给出了两则在 RSA 密码算法下要想产生能提供足够安全强度的密钥应该遵守的建议。

关键词: 大整数; 因数分解; RSA; 密钥长度; 安全性

中图分类号: TN918.1

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2017.00166

Factorization of big integer and the security of RSA

REN Yan-bing

(School of Cyber Engineering, Xidian University, Xi'an 710071, China)

Abstract: Three kinds of methods for integer factorization were proposed and the security of RSA was demarcated. RSA is a well-known cryptographic algorithm, using the analysis result of those methods. Through the work, readers could easily realize that if merely enlarged two prime numbers but lost attention of the relevance of them, the security of this algorithm might been missed. In the end, two recommended tactics to choose prime numbers as key of this algorithm were given.

Key words: integer, factorization, RSA, key size, security

1 引言

RSA 密码算法的安全基于大整数分解的困难性^[1]。公认的能够破解 RSA 密码算法的方法是实现大整数的有效分解^[2,3], 即将一个大整数 n 分解为 $p_1 \times p_2$ (p_1 、 p_2 为大素数)。到目前为止, 人们依然无法快速分解大整数, 但随着计算能力的不断提高, 要想取得同等强度的安全性, RSA 的密钥长度也需要逐渐地增加^[4-6]。但是密钥增加到多少位才可以说是安全的? 对这个问题业界并没有一个严格的界定。一般的观点是密钥长度以待加密内容的密级而定^[7,8], 但这种说法依然十分模糊。例如, 一个以 RSA 算法提供 HTTPS 连接的网站可能会选择

2 048 bit 的 RSA 密钥, 这足够安全, 但却浪费了服务器的大量计算资源, 因为对于一般的民用网络来说, 1 024 bit 的 RSA 密钥就足够。因此, 本文的主要目的是为解决上述问题提供一种思路。本文试图用 3 种方式完成整数的因数分解, 并根据对这一问题的探讨给出 2 种 RSA 密钥参数的选取策略, 依据该策略用户可以生成相对安全的 RSA 密钥。但需要注意的是, 这种安全是基于本文所讨论的对因数分解这一问题的解决方案。

2 RSA 密码算法简述

1) 选取大素数 p_1 与 p_2 , 计算 $n = p_1 \times p_2$, $\varphi(n) = (p_1 - 1)(p_2 - 1)$ 。再选取正整数 e 作为公钥,

其中 e 满足 $(e, \varphi(n)) \equiv 1 \pmod{n}$; 再求出 d 作为私钥, 其中 d 满足 $ed \equiv 1 \pmod{n}$ 。

2) 将明文 M 加密: $C \equiv M^e \pmod{n}$, 其中 M 满足 $M < n$ 。

3) 将密文 C 解密^[9]: $M \equiv C^d \pmod{n}$, 其中 C 满足 $C < n$ 。

3 整数因数分解方法介绍

3.1 方法一

3.1.1 综述

注意到 2 个大素数一定都是奇数^[10], 则它们一定满足关系式 $(p_1 + p_2) \bmod 2 = 0$, 这个式子的意义是 p_1 与 p_2 之间一定存在一个唯一的中间值 p_3 , 且 $p_3 = \frac{p_1 + p_2}{2}$ 。如 $3 \times 7 = 21$, 3 与 7 的中间值是 5; $3 \times 17 = 51$, 3 与 17 的中间值是 10。

如果能够较快地找到 p_1 与 p_2 的中间值, 那么从这一点同时向左右两边遍历就能够找到 p_1 、 p_2 , 至此实现了大整数 $n = p_1 \times p_2$ 的分解。

3.1.2 进一步探讨

如何较快速地寻找 $p_3 = \frac{p_1 + p_2}{2}$? 一种直观的想法就是求解 $\text{sqrt} = \sqrt{n}$ ^[11], 并通过 sqrt 寻找 p_3 。但显然 $\text{sqrt} = \sqrt{n}$ 并不一定得到一个整数, 而 p_3 则一定是一个整数, 因此, 引入 2 个取整函数以及由它们计算所得的值。

$$\text{sqrt}_{\text{below}} = \lfloor \sqrt{n} \rfloor$$

$$\text{sqrt}_{\text{above}} = \lceil \sqrt{n} \rceil$$

此时, 如果发现 $\text{sqrt}_{\text{below}} = \lfloor \sqrt{n} \rfloor$ 正是所需的 $p_3 = \frac{p_1 + p_2}{2}$, 大整数分解的任务便已经完成了一大半, 但事实上, 随着 $|p_1 - p_2|$ 的增大, $|\text{sqrt} - p_3|$ 也会增大, 因此, 需要在已知 sqrt 的基础上求得 p_3 。

$p_3 - \text{sqrt}$ 随 p_1 、 p_2 在正整数范围内的变化趋势如图 1 所示。

图 1 中的 x 轴与 y 轴分别表示 p_1 、 p_2 , z 轴表示 $p_3 - \text{sqrt}$ 。可以看出, $p_3 - \text{sqrt}$ 的值随着 p_1 与 p_2 之间差值的增大而增大; 当 p_1 与 p_2 之间非常接近时, $p_3 - \text{sqrt}$ 的值随之减小, 即 p_3 与

sqrt 之间的接近程度与 p_1 与 p_2 之间的接近程度呈正相关。

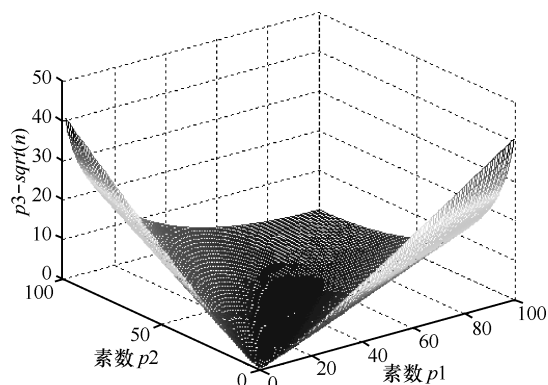


图 1 $p_3 - \text{sqrt}$ 随 p_1 、 p_2 的变化趋势

由图 1 可知, 对于任意正整数 n , 如果有 $\sqrt{n}$ 不是一个整数, 那么有 $\text{sqrt}_{\text{below}}^2 < n < \text{sqrt}_{\text{above}}^2$ 。因此, 首先选择令 $\text{sqrt}_{\text{below}}$ 为 p_1 与 p_2 的中间值, 然后向 $\text{sqrt}_{\text{below}}$ 的左右两边寻找 p_1 与 p_2 , 并使 $n = p_1 \times p_2$ 。但是, 在此过程中发现, 不论 i 取多少, $(\text{sqrt}_{\text{below}} - i) \times (\text{sqrt}_{\text{below}} + i)$ 的值一定小于 $\text{sqrt}_{\text{below}}^2$ 。也就是说, $\text{sqrt}_{\text{below}}^2$ 小于 n , 而 $(\text{sqrt}_{\text{below}} - i) \times (\text{sqrt}_{\text{below}} + i)$ 的值却比 $\text{sqrt}_{\text{below}}^2$ 更小, 即

$$(\text{sqrt}_{\text{below}} - i) \times (\text{sqrt}_{\text{below}} + i) < \text{sqrt}_{\text{below}}^2 < n$$

成立。事实上, 对于任何整数 a , 有 $(a - i) \times (a + i) = a^2 - i^2 \leq a^2$, 其中 i 可取任何整数。因此, 选择 $\text{sqrt}_{\text{below}}$ 作为中间值的尝试是没有必要的, 因为从 $\text{sqrt}_{\text{below}}$ 出发向两边取值的乘积永远不会等于 n 。基于此, 可以选择 $\text{sqrt}_{\text{above}}$ 作为中间值 p_3 的候选值。

以 $\text{sqrt}_{\text{above}}$ 为出发点向两边取值, 并求乘积的过程是可行的, 原因是 $\text{sqrt}_{\text{above}}^2 \geq n$, 且随着 i 的增大, $(\text{sqrt}_{\text{above}} - i) \times (\text{sqrt}_{\text{above}} + i)$ 的值逐渐减小, 并在当 i 增大到某个值时有 $(\text{sqrt}_{\text{above}} - i) \times (\text{sqrt}_{\text{above}} + i) = n$ 或 $(\text{sqrt}_{\text{above}} - i) \times (\text{sqrt}_{\text{above}} + i) < n$, 前一种情况表示已经找到了 p_1 、 p_2 满足 $n = p_1 \times p_2$, 即完成了大整数的分解; 后一种情况说明 $\text{sqrt}_{\text{above}}$ 依然不等于中间值 p_3 , 需要继续设定 p_3 的候选值, 并以同样的方法寻找 p_1 、 p_2 。为此, 将 $\text{sqrt}_{\text{above}} + 1$ 设置为新的 p_3 候选值, 并以它为基点向两边寻找, 依次类推。

3.1.3 优化方案

显然大素数都应该是奇数, 因此在确定了中

间值的探测值 $Test$ 以后, 可以在以此为中点向左右两边探测的过程中跳过 2 个偶数相乘的情况, 只对 2 个奇数相乘的情况进行计算比对。

3.1.4 算法

设已知大整数 n 是由 2 个素数相乘得到, 有 $n = p_1 \times p_2$, 且 n 已知, p_1 、 p_2 未知, 则对于给定的大整数 n , 欲求 p_1 、 p_2 , 有以下算法。

- 1) 求出 $sqr t = \sqrt{n}$ ^[12,13], 令 $sqr t_{above} = \lceil \sqrt{n} \rceil$, 转第 2) 步。
- 2) 定义变量 $Test$, 令其初值为 $sqr t_{above}$, 转第 3) 步。
- 3) 判断 $Test$ 的奇偶性, 若为奇数, 设 $i = 0$; 若为偶数, 设 $i = 1$, 转第 4) 步。
- 4) 计算 $Result = (Test - i) \times (Test + i)$, 转第 5) 步。
- 5) 若 $Result = n$, 则令 $p_1 = Test - i$, $p_2 = Test + i$, 算法结束; 否则, 转第 6) 步。
- 6) 若 $Result > n$, 计算 $i = i + 2$, 转第 4) 步; 否则, 转第 7) 步。
- 7) 若 $Result < n$, 计算 $Test = Test + 1$, 转第 3) 步。

3.1.5 效率

该算法其实分为 2 个维度: 第一个维度是寻找 p_1 与 p_2 的中间值 $p_3 = \frac{p_1 + p_2}{2}$, 实际操作是从 $sqr t_{above}$ 出发, 并不断地对这个值进行加 1 操作使其最终等于 p_3 , 因此这个维度可以用 $|p_3 - sqr t_{above}|$ 来衡量; 第二个维度是从所选择的中间值 $Test$ 开始, 不断地比较 $(Test - i) \times (Test + i)$ 是否等于 n 来判断 $(Test - i)$ 与 $(Test + i)$ 是否分别等于 p_1 与 p_2 ($i = 1, 2, 3, \dots$), 这个维度的计算次数随着 $Test$ 的增大而增大 (线性增大), 具体值取决于使 $Test^2 - i^2 < n$ 中 i 的大小。因此, 整个算法的总计算次数可以看成是这 2 个维度的计算次数构成的三角形的面积, 这个三角形的高是 $|p_3 - sqr t_{above}|$, 长是 $\left| p_2 - \frac{p_1 + p_2}{2} \right| = \left| \frac{p_2 - p_1}{2} \right|$, 因此总的基本计算次数约等于 $\frac{1}{2} \left| \frac{p_2 - p_1}{2} \right| |p_3 - sqr t_{above}|$, 可以看到用这种算法的计算次数与 n 无关, 只与 p_1 、 p_2 以及 $\sqrt{n}$ 有关, 但 $\left| \frac{p_2 - p_1}{2} \right|$ 的数量级是 $\sqrt{n}$ 级别,

$|p_3 - sqr t_{above}|$ 的数量级也是 $\sqrt{n}$ 级别, 因此, 整个算法的时间复杂度是 $O(n)$ 。

3.1.6 进一步优化

在上述讨论的算法中, 当出现 $(Test - i) \times (Test + i) < n$ 时, 将 $Test$ 自增 1, 并让 i 回到 0, 重新遍历所有可能的取值。事实上这是不需要的, 只需要在 $(Test - i) \times (Test + i) < n$ 时将 $Test$ 自增 1, 同时令 i 在当前值对的基础上继续增大即可。之所以可以这样, 是因为注意到 $(a - i) \times (a + i) = a^2 - i^2$, 则如果当前的 i 值满足 $a^2 - i^2 < n$ 且 $a^2 - (i - 1)^2 > n$, 则必有 $(a + 1)^2 - i^2 > n$ 。因此不必让 i 回复到 0 重新遍历, 而只需让它继续增大即可。

举例说明此问题, 假设现已知 $n = 57$, 要求 2 个素数 p_1 、 p_2 使 $n = p_1 \times p_2$ 。

根据算法, 此时有 $Test$ 的初值为 8, i 的初值为 0。算法执行过程如下所示。

- 1) $Test = 8$, $8^2 - 1 = 63 > 57$, $8^2 - 2^2 = 60 > 57$, $8^2 - 3^2 = 55 < 57$
- 2) $Test = 9$, $9^2 - 1 = 80 > 57$, $\dots$, $9^2 - 3^2 = 72 > 57$, $\dots$, $9^2 - 5^2 = 56 < 57$
- 3) $Test = 10$, $10^2 - 1 = 99 > 57$, $\dots$, $9^2 - 5^2 = 75 > 57$, $\dots$, $9^2 - 7^2 = 51 < 57$
- 4) $Test = 11$, $11^2 - 1 > 57$, $\dots$, $11^2 - 7^2 > 57$, $11^2 - 8^2 = 57$

因此, $p_1 = 11 - 8 = 3$, $p_2 = 11 + 8 = 19$, 算法结束。

为方便理解, 此算法依然沿用了每次触发判断条件时将 i 恢复成 1, 但从中也可以看出, 更有效率的做法是在 $(Test - i) \times (Test + i) < n$ 时不回置 i , 而是让它继续增大。

3.1.7 新算法

设已知大整数 n 由 2 个素数相乘得到, 即 $n = p_1 \times p_2$, 且 n 已知, p_1 、 p_2 未知。则对于给定的大整数 n , 欲求 p_1 、 p_2 , 有以下算法。

- 1) 求出 $sqr t = \sqrt{n}$, 令 $sqr t_{above} = \lceil \sqrt{n} \rceil$, 转第 2) 步。
- 2) 定义变量 $Test$, 令其初值为 $sqr t_{above}$, 转第 3) 步。
- 3) 判断 $Test$ 的奇偶性, 若为奇数, 设 $i = 0$; 若为偶数, 设 $i = 1$, 转第 4) 步。
- 4) 计算 $Result = (Test - i) \times (Test + i)$, 转第

5)步。

5) 若 $Result = n$, 则令 $p1 = Test - i$, $p2 = Test + i$, 算法结束; 否则, 转第 6)步。

6) 若 $Result > n$, 计算 $i = i + 2$, 转第 4)步; 否则, 转第 7)步。

7) 若 $Result < n$, 计算 $Test = Test + 1$, $i = i + 1$, 转第 4)步。

3.1.8 改进后的效率

改进后的算法避免了对 i 的重复取值, 将算法的整体复杂度从二维降到了一维。算法仍需要 $Test$ 进行猜测性遍历, 对 $Test$ 的遍历其实是测试大于 $\sqrt{p3}$ 、小于 $p3$ 的全部整数, 总的测试次数是 $|p3 - \sqrt{p3}|$, 时间复杂度是 $O(\sqrt{n})$ 。

3.2 方法二

3.2.1 综述

继续讨论 $n = p1 \times p2$, 在方法一的讨论中已经知道, $(\sqrt{p3} - i) \times (\sqrt{p3} + i)$ 随着 i 的增大会越来越小, 因此, 如果假设已经找到了 $p3 = \frac{p1 + p2}{2}$, 那么在 i 增大的过程中, $(p3 - i) \times (p3 + i)$ 会逐渐减小, 并在 i 取到某一个特定的值时满足 $(p3 - i)(p3 + i) = n$, 以下用实例来说明。

已知 $11 \times 23 = 253$, $\frac{11+23}{2} = 17$, 则

$$17^2 = 289$$

$$(17-1) \times (17+1) = 288$$

$$(17-2) \times (17+2) = 285$$

$$(17-3) \times (17+3) = 280$$

$$(17-4) \times (17+4) = 273$$

$$(17-5) \times (17+5) = 264$$

$$(17-6) \times (17+6) = 253$$

至此, 求出了 $p1 = 17 - 6 = 11$, $p2 = 17 + 6 = 23$ 。

如果对这个例子进行进一步的研究, 求出它们相邻项数之差, 即有下面各式。

$$17^2 - (17-1) \times (17+1) = 1$$

$$(17-1) \times (17+1) - (17-2) \times (17+2) = 3$$

$$(17-2) \times (17+2) - (17-3) \times (17+3) = 5$$

...

$$(17-5) \times (17+5) - (17-6) \times (17+6) = 11$$

可以看到, 随着 i 的增大, 相邻的 $(p3 - i) \cdot (p3 + i)$ 之差构成了一个公差是 2 的等差数列。事实上, 并不只有某些特定的数字符合这种情况, 对于所有非零整数 k , 都有 $(k+i)(k-i) - [k+(i+1)][k-(i+1)]$ 的值构成 $1, 3, 5, 7, 9, 11, \dots$ 的等差数列 ($r=0, 1, 2, 3, \dots$)。

作为测试, 随机选取 2 个素数 $p1 = 1033$ 与 $p2 = 1997$, 它们的中值 $p3 = \frac{p1 + p2}{2}$ 为 1515。绘制出当 i 取 $0, 1, 2, 3, \dots$ 时, $(p3 + i)(p3 - i) - [p3 + (i + 1)][p3 - (i + 1)]$ 取值的图形, 如图 2 所示。

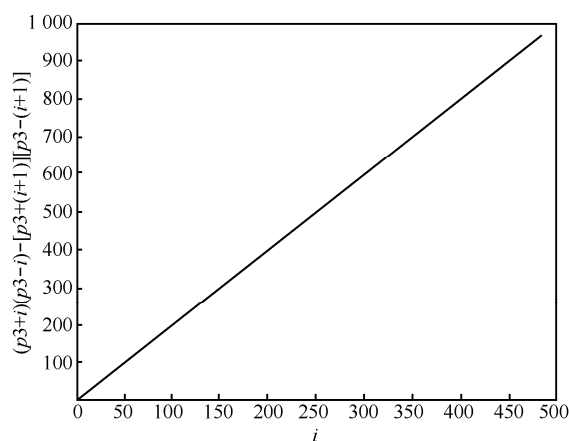


图2 相邻两项之差构成等差数列

图 2 中横坐标表示 i , 纵坐标表示 $(p3 + i)(p3 - i) - [p3 + (i + 1)][p3 - (i + 1)]$ 。可以看出, 相邻 $(p3 - i)(p3 + i)$ 的差值符合首项为 1、公差为 2 的等差数列。由此, 可以设计出另一种解决素数乘积问题的方法。

回到 $11 \times 23 = 253$ 的例子, 此时 253 是已知的, 并且知道它是由 2 个素数相乘得到的, 因此需要找到 11 与 23 来完成大整数问题的分解。假设已知 $\frac{11+23}{2} = 17$, 即有 $17 \times 17 = 289$, 由

$$289 - 1 - 3 - 5 - 7 - 9 - 11 = 253 \quad (1)$$

可知 253 是由 $(17-6) \times (17+6)$ 得到的, 即 $253 = 11 \times 23$ 。但在实际问题中, 仅已知 $n=253$, 而并不知道 $17 = \frac{11+23}{2}$; 此时可以对式(1)进行恒等变换, 有 $253 + 1 + 3 + 5 + 7 + 9 + 11 = 289$ 。在实际实现的过程中, 在计算 $253 + 1 + 3 + 5 + 9 + 11 + \dots$ 的过程中, 需要对所求得的中间值进行开根号的测试, 若 $253 + 1 + 3 + 5 + \dots + i$ 的值能够被开根号

为整数, 这个数即为所求的 p_3 。然后求出 $p_3 + i$ 、 $p_3 - i$ 即为 p_1 、 p_2 。

3.2.2 算法

设已知大整数 n 由 2 个素数相乘得到, 有 $n = p_1 \cdot p_2$, 且 n 已知, p_1 、 p_2 未知。则对于给定的大整数 n , 欲求 p_1 、 p_2 , 有以下算法。

- 1) 令 $Test = n$, $i = 1$, $count = 1$, 计算 $Sum = Test + i$ 。
- 2) 判断 $\sqrt{Sum}$ 是否是整数, 若是, 转第 4) 步; 若不是, 转第 3) 步。
- 3) 依次计算 $i = i + 2$, $Sum = Sum + i$, $count = count + 1$, 转第 2) 步。
- 4) 计算 $p_1 = \sqrt{Sum} - count$, $p_2 = \sqrt{Sum} + count$, 算法结束。

3.2.3 改进

以上算法虽然可以完成大整数分解的任务, 但是它相当于遍历了从 n 到 $(p_3)^2$ 的所有奇数, 而随着 n 的增大, 这个算法的计算量也是巨大的, 因此需要对这个算法进行改进。

注意到 3.2.2 节算法是在已知 n 的基础上累加一个等差数列的各项, 直到求和的结果能够被开根号为整数为止, 这个算法能够被优化是因为在之前的讨论中并没有对等差数列的特性进行讨论, 而当对等差数列本身加以关注, 并引入等差数列的求和公式后, 此算法的时间复杂性会有一个数量级的改进。

等差数列求和公式如下^[14,15]

$$S_k = a_1 k + \frac{k(k-1)}{2} d \quad (2)$$

其中, a_1 是首项, d 是公差, k 是项数, S_k 是各项之和。

在已知 a_1 与 d 的情况下, 给出任何一个数字 S_k , 都能够求出是否存在唯一的正整数 k 满足式(2), 在 $a_1 = 1$, $d = 2$ 的情况下, 式(2)变为

$$S_k = k^2 \quad (3)$$

此时, 便不需要用 n 依次加上 1, 3, 5, 7, 9, ... 去判断结果能否整开根号, 只需要迅速找到一个可能有用的值 $Test$, 然后计算 $Test - n$ 是否能使式(3)成立, 即 $Test - n$ 是否能被整开方即可。如果 $Test$ 能够满足上述条件, 那么 $\sqrt{Test}$ 即为 p_1 与 p_2 的中值。

举例说明这种改进, $7 \times 11 = 77$, $sqr_{above} = \lceil \sqrt{77} \rceil = 9$; $sqr_{above}^2 = 9^2 = 81$ 。

按照原始算法的思路, 需要计算 $77+1+3+\dots$, 直到计算结果可以被开根号为整数为止, 事实上, 这需要进行 2 次比对, 即先计算 $77+1=78$, $\sqrt{78}$ 不是整数, 再计算 $78+3=81$, $\sqrt{81}$ 是一个整数, 因此 $\sqrt{81}=9$ 即为生成该整数的 2 个素数的中值。改进后的算法首先计算 $sqr_{above} = \lceil \sqrt{77} \rceil = 9$, 然后计算 $sqr_{above}^2 - n$, 计算结果为 4, 因为 4 满足首项为 1、公差为 2 的等差数列的求和公式, 于是 $sqr_{above} = 9$ 即为所求, 然后令 9 分别减去与加上 $\sqrt{4}=2$, 即为素数 7 与 11 的数值。相比于原算法, 改进算法只运行了一轮基本步骤就求得了结果。

综上所述, 可以先令 $Middle = sqr_{above}$, 然后计算 $Middle^2$, 再求 $Distance = (Middle)^2 - n$ 的值, 如果这个值能够满足首项为 1、公差为 2 的等差数列求和公式, 则 sqr_{above} 就是所要寻找的 $p_3 = \frac{p_1 + p_2}{2}$, 反之, 令 $Middle = sqr_{above} + 1$, 直到找到满足条件的值为止。

3.2.4 改进后的算法

设已知大整数 n 由 2 个素数相乘得到, 即 $n = p_1 \times p_2$, 且 n 已知, p_1 、 p_2 未知。则对于给定的大整数 n , 欲求 p_1 、 p_2 , 有以下算法。

- 1) $Middle = sqr_{above}$, 转步骤 2)。
- 2) 计算 $Distance = Middle^2 - n$, 转步骤 3)。
- 3) 判断 $\sqrt{Distance}$ 是否是整数, 若是, 转步骤 5); 若不是, 转步骤 4)。
- 4) 计算 $Middle = Middle + 1$, 转步骤 2)。
- 5) 计算 $p_1 = Middle - \sqrt{Distance}$, $p_2 = Middle + \sqrt{Distance}$, 算法结束。

3.2.5 效率分析

该算法在初始时选取 $Middle = sqr_{above}$ 作为待定值, 随后 $Middle$ 的值逐渐增大, 并在 $Middle = p_3 = \frac{p_1 + p_2}{2}$ 时结束, 在此过程中算法的基本操作总共需要执行 $\frac{p_1 + p_2}{2} - sqr_{above}$ 次。可以看到, 方法二与方法一从不同的角度出发, 最终获得了相同的效率与时间复杂度。

3.3 方法三

3.3.1 综述

注意到任何 2 个正整数的乘积都可以表示为一个等差数列逐项求和, 其中等差数列的公差是 2。因为 2 个正整数相乘总可以表示为 $n = k(a+k)$, 其中 k 与 $a+k$ 分别为乘数与被乘数。如果将它们分别减去 $k-1$, 乘式变为 $1 \times (a+1)$, 此时, 令 $C = k \times (a+k) - 1 \times (a+1)$, 则 $n = k \times (a+k) = 1 \times (a+1) + C$, 其中, $C = k \times (a+k) - 1 \times (a+1) = (a+1+k)(k-1)$, 于是有

$$\begin{aligned} n &= 1 \times (a+1) + C \\ &= a+1 + (a+1+k)(k-1) \\ &= k(a+1) + (k-1)k \end{aligned}$$

此时, n 满足首项为 $a+1$, 公差为 2, 项数为 k 的等差数列的通项求和公式。

令 $a_1 = a+1$, 此时有 $n = ka_1 + (k-1)k$ 。

由此, 可以对 k 进行遍历, 并计算 $Test = n - (k-1)k$, 如果求出的 $Test$ 能够被 k 整除, 即求出了 a_1 与 k , 继而找到了整数 n 的 2 个因数。

3.3.2 算法

设已知大整数 n 由 3 个素数相乘得到, 有 $n = p_1 \times p_2$, 且 n 已知, p_1 、 p_2 未知。则对于给定的大整数 n , 欲求 p_1 、 p_2 , 有以下算法。

- 1) 初始时设 $k = 3$, 转步骤 2)。
- 2) 计算 $Test = n - (k-1)k$, 转步骤 3)。
- 3) 判断 k 能否整除 $Test$, 若可以, 转步骤 5); 反之, 转步骤 4)。
- 4) 令 $k = k + 2$, 转步骤 2)。
- 5) 计算 $a_1 = \frac{Test}{k}$, 则 $p_1 = k$, $p_2 = a_1 + k - 1$ 。

3.3.3 效率

这个算法的基本步骤是遍历 k , 又由于 $k = \min(p_1, p_2)$, 因此这个算法的基本运算大约执行 $\frac{\min(p_1, p_2)}{2}$ 次 (只需要遍历奇数), 时间复杂度是 $O(\sqrt{n})$ 。从效率上看, 这个算法与从 1 开始遍历所有奇数, 使用暴力破解的方法探测 $\min(p_1, p_2)$ 的值是一样的。在此给出这个方法主要是基于 2 点考虑: 一是探讨了任何正整数 n 都等于一个公差为 2 的等差数列逐项求和这一性质; 二是为本文后面探讨 RSA 的密钥选取策略提

供方便。

4 进一步探讨

众所周知, 公钥密码体系 RSA 密码算法的安全性是基于整数分解问题的困难性的, 但是 RSA 密码算法的挑战在于随着计算机计算能力的增大, 必须选取更加大的素数相乘作为密钥。但是如果不考虑 2 个数字之间的关系, 只是单纯地选取更大的数字, 有可能并不能起到预想的安全效果。本文讨论的算法为这种安全性做了一个界定。由方法一与方法二可知, 如果想让 RSA 密码加密文件更安全, 就需要在生成公私钥时选择相距尽可能远的素数, 因为只有这样, $|p_3 - \text{sqr}_{\text{above}}|$ 的值才会大到攻击者无法采用穷举方式进行破解; 而方法三与传统意义上的简单暴力破解则指出, 令 RSA 密码算法安全的必要条件是 p_1 与 p_2 中较小的一个 $\min(p_1, p_2)$ 足够大, 以至于攻击者无法通过枚举所有自然数的方式解决问题。

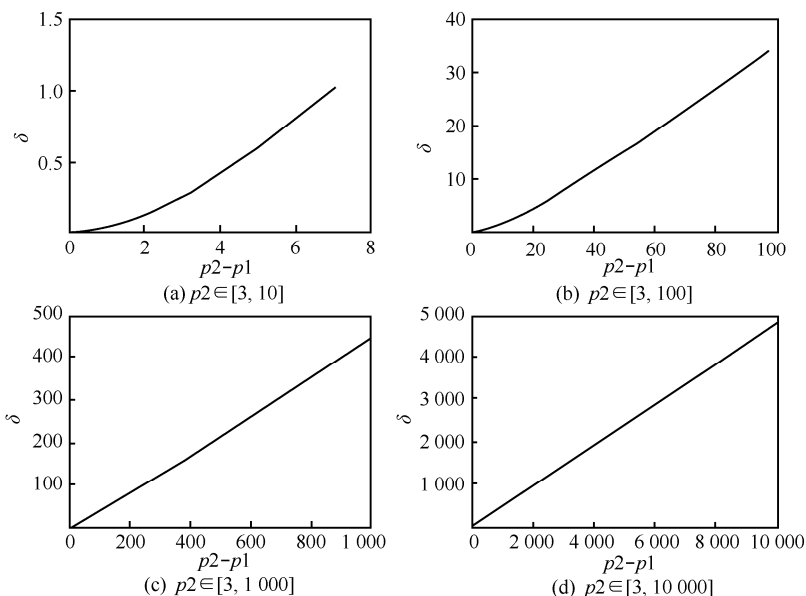
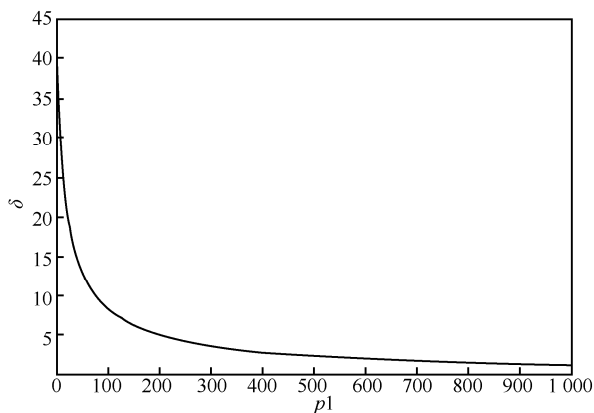
基于此, 可以定义一个衡量 RSA 安全性的参数 ζ , 且 $\zeta = \min(p_1, p_2, \delta)$, 其中, $\delta = |p_3 - \text{sqr}_{\text{above}}|$ 。这个值的物理意义是攻击者若采用本文给出的 3 种方法或是暴力破解中的任意一种, 要破解由参数 ζ 定义的 RSA 密码所最少需要的计算次数。

因为在方法的实现中, $\min(p_1, p_2)$ 相对容易确定且这也是大多数用户首先考虑到的安全指标^[16], 故而选择对 δ 做进一步探讨。

首先做一个实验, 令 p_1 取为常数 3, 通过 p_2 的变化来观察 δ 的变化如图 3 所示。

从图 3 中可以看出, 随着 p_2 的增大, δ 与 $p_2 - p_1$ 的关系越来越趋近于一条斜率为 $\frac{1}{2}$ 的直线。这也说明增加 p_1 与 p_2 的差值确实可以提高算法的安全性, 因为随着 δ 的增加, 分解整数 n 所需的运算量也越来越大。

但是差值到底要选为多大才算是真的安全, 这又是一个相对的概念^[17]。如果用户选择一个非常大的差值并将它运用在所有 p_1 与 p_2 的参数对中, 这也是不可取的。因此, 可以做另一个实验, 令 p_1 与 p_2 之间的差值固定, 此处固定为 100, 然后同步增大 p_1 与 p_2 , 得到如图 4 所示的 δ 关于 p_1 的图形。

图3 δ 随 $p1$ 与 $p2$ 差值的变化图4 固定 $p1$ 与 $p2$ 之间差值后 δ 的变化趋势

从图4中可以看出,随着 $p1$ 与 $p2$ 的增大,由这2个参数构成算法的安全性呈指数级减少。因此,仅确保将 $p1$ 与 $p2$ 之间的差值设定为一个固定的数值(或数量级)是不安全的,因为随着 $p1$ 与 $p2$ 的增大, δ 的值(或数量级)会随着图4中曲线所展示的方式趋于减小,这让分解整数 n 的因数变得简单。

5 RSA 中密钥选取

通过以上的分析,给出2种RSA中初始参数 $p1$ 、 $p2$ 的选取策略,第一种策略较为直观简便,第2种策略更加理论化。

5.1 策略一

该策略可以简单描述如下。

1) 确认系统需要的安全级别,判断需要选取

多少位的十进制数来作为自己的安全阈值。

2) 假设在步骤1)中,确认至少需要 k 位的十进制数来阻止攻击者的计算攻击,那么在这一步中,首先选择第一个素数 $p1$,且 $p1$ 满足 $p1 > 10^k$ 。

3) 选择 $p2$,要求 $p2 > p1$,且如果要在 k 位十进制整数中选择 $p2$,则 $p2$ 必须满足 $\frac{p2+p1}{2} - \sqrt{p2p1} > 10^k$ (即 $\delta > 10^k$);当然,也可以在任何 l 位整数中选取参数 $p2$, l 满足 $l > k$ 。

5.2 策略二

根据上文论述,依然取 ζ 作为衡量RSA的安全参数,并选取 η 作为用户能够接受的安全阈值。其中 η 是用户认定的安全计算次数,即用户认定攻击者无法承受 η 次的计算来破译密码。显然,算法参数的选取应该满足 $\zeta \geq \eta$ 。

根据上文的叙述, ζ 为 $p1$ 、 $p2$ 与 δ 当中的较小者,不妨设 $p2 > p1$,即算法参数的选取应该满足使 $p1$ 与 δ 均不小于 η 。又因为 $\delta = |p3 - \text{sqr}_{\text{above}}|$ 且当 $n \gg 1$ 时,有 $p3 > \text{sqr}_{\text{above}}$,所以在 $n \gg 1$ 时有

$$\begin{cases} \eta \leq \frac{p2+p1}{2} - \lceil \sqrt{p2p1} \rceil \\ \eta \leq p1 \end{cases}$$

即

$$\begin{cases} \eta \leq \frac{p2+p1}{2} - \sqrt{p2p1} \\ \eta \leq p1 \end{cases} \quad (4)$$

在选择参数 p_1 、 p_2 的过程中, 令 p_1 满足 $p_1 \geq \eta$ 是容易实现的; 而欲令 p_2 满足 $\frac{p_2 + p_1}{2} - \sqrt{p_2 p_1} \geq \eta$, 需要求出此关于 p_2 的一

元二次不定方程的解。对式(4)进行化简, 可得

$$(p_2)^2 - (2p_1 + 4\eta) \times p_2 + [(p_1)^2 - 4\eta p_1 + 4\eta^2] \geq 0$$

计算可知方程有解, 选取方程解中较大的一个即为应该选取的 p_2 的最小取值。

6 结束语

本文从整数的因数分解问题入手, 提出了3种能够完成因数分解问题的方法, 并对这些方法的效率做了简要的分析与度量。另外, 在本文的方法三中简要讨论了2个数的乘积与等差数列求和之间的关系问题。第4节对提出的算法所定义的RSA安全性做了进一步的讨论, 并在第5节给出了2种RSA密钥参数的选取策略, 以确保用户生成的整数 n 在因数分解问题上的困难性。

参考文献:

- [1] 陈恭亮. 信息安全数学基础[M]. 北京: 清华大学出版社, 2014.
CHEN G L. Mathematical basis of information security [M]. Beijing: Tsinghua University Press, 2014.
- [2] STALLINGS W. Cryptography and network security: principles and practice[C]//Prentice Hall. 2010.
- [3] STINSON D R. Cryptography: theory and practice[M]. Boca Raton: CRC Press, 2005.
- [4] 胡向东, 魏琴芳. 应用密码学教程[M]. 北京: 电子工业出版社, 2005.
HU X D, WEI Q F. Applied cryptography tutorial[M]. Beijing: Electronic Industry Press, 2005
- [5] BRENNER A E. Moore's law[J]. Science, 1997, 275(5306): 1551-1551.
- [6] ROMANELLI A. Intel extends moore's law[J]. Electronic News, 2002, 48(8):1.
- [7] 王恒青, 宋如敏. 公开密钥密码体制的安全取决于密钥的长度[J]. 科技信息: 学术研究, 2008(34):214-216.
WANG H Q, SONG R M. The security of public key cryptosystem depends on the length of the key[J]. Science And Technology Information: Academic Research, 2008 (34): 214-216.
- [8] 王恒青, 宋如敏. 密钥的长度影响着 RSA 体制的安全程度[J]. 电脑知识与技术, 2011, 7(10): 7104-7105.
WANG H Q, SONG R M. The length of the key affects the security of the RSA system [J]. Computer Knowledge and Technology, 2011, 7 (10): 7104-7105.
- [9] RIVEST R L, SHAMIR A, ADLEMAN L. A method for obtaining digital signatures and public-key cryptosystems[J]. Communications of the ACM, 1983, 26(2):96-99.
- [10] RIBENBOIM P. The book of prime number records[J]. Mathematical Gazette, 1988, 73(465):663-665.
- [11] 姜建国, 臧明相. 数论算法[M]. 西安: 西安电子科技大学出版社, 2014.
JIANG J G, ZANG M X. Number theoretic algorithm[M]. Xi'an: Xidian University Press, 2014.
- [12] BECKETT R, HURT J. Numerical calculations and algorithms[J]. Mathematics of Computation, 1968, 22(102): 266-269.
- [13] HUANG Y L. Square root algorithm: US20070112903[P]. 2007.
- [14] HONG S. Application of functions to get the sum and product of partial sequencing items in series of equal difference and proportion[J]. Journal of Nanyang Normal University, 2006, 26(5): 159-171.
- [15] KOTZ S, NADARAJAH S. Arithmetic progression[M]//Encyclopedia of Statistical Sciences. John Wiley & Sons, Inc., 2006:141.
- [16] KAHNEMAN D. Thinking, fast and slow[M]// Thinking, Fast and Slow. Farrar, Straus and Giroux, 2011:501-502.
- [17] SYNGE J L. Relativity: the general theory[J]. Physics Today, 2009, 14(10):50-52.

作者简介:



任彦冰 (1993-), 男, 甘肃武威人, 西安电子科技大学硕士生, 主要研究方向为网络与信息安全。