大数因子分解算法综述*

刘新星[1]，邹潇湘[2]，谭建龙[1]
（1. 中国科学院信息工程研究所，北京 100093；2. 国家计算机网络应急技术处理协调中心，北京 100029）

摘 要：大数因子分解不仅是非对称加密算法 RSA 最直接的攻击手段，也是 RSA 安全性分析最关键的切入点，对其研究具有极其重要的应用和理论价值。主要概括了大数因子分解的研究现状，回顾了当前主流的大数因子分解算法，介绍了它们的基本原理和实现步骤；此外，对比分析了现有大数因子分解技术在实现和应用上的优缺点；最后分析并展望了大整数分解未来的研究趋势。

关键词：大数因子分解；非对称加密；RSA；安全性
中图分类号：TP309.2 **文献标志码**：A **文章编号**：1001-3695(2014)11-3201-07
doi:10.3969/j.issn.1001-3695.2014.11.001

Survey of large integer factorization algorithms

LIU Xin-xing[1], ZOU Xiao-xiang[2], TAN Jian-long[1]
(1. *Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093, *China*; 2. *National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing* 100029, *China*)

Abstract: The large integer factorization is not only the most direct attacking method against RSA asymmetric encryption algorithm, but also the most important point to analyze the security of RSA. Study on the large integer factorization problem is of great value for theory and practice. This paper summarized the study on of the large integer factorization problem and reviewed modern popular integer factorization algorithms, and introduced their basic prinple and implementation steps. In addition, this paper made an analysis on existing large integer factorization techniques' advantage and disadvantage of implementation and application. At last, this paper stated the future prospect of large integer factorization.

Key words: large integer factorization; asymmetric encryption; RSA; security

RSA[1]是当前最重要的公开密钥算法，它广泛应用在各领域，其安全性决定于对大整数分解的难度。所谓整数因子分解，即任一大于 1 的自然数都可以写成质数的乘积，这些质因子按大小排列之后，表达形式唯一，但是具体的因子值不能直接得知。当合数所有的因子都很大时，采用强力方式得到具体的因子是很困难的，而这也正是 RSA 体制理论的核心。

整数因子分解与质性检测这两个问题非常类似，如今质性检测已被完全证明多项式时间可解[2]，而大数因子分解问题仍然悬而未决。几百年来，大数因子分解问题既未被证明是多项式时间可解的 P 问题，也未被证明是 NP 完备问题，对其研究具有极其重要的理论和应用价值。

本文首先归纳近些年大数因子分解研究工作进展和现有计算机与算法水平下的 RSA 模数分解能力。对若干重要大数分解算法进行回顾，并对它们进行分析、比较各自的优缺点，最后对大数分解未来进一步的研究趋势进行展望。

1 研究现状

为了鼓励更多的人加入研究大数分解问题的行列，跟踪整数分解研究进展，早在 1991 年，RSA Laboratories[3] 发起了 RSA 模数分解挑战赛，它们发布了十进制位数从 100～617 位的 54 个 RSA 模数，并称任何组织和个人，不管使用什么样的技术手段，只要分解了其中的任意一个数即可获取他们的高额奖金。20 年过去了，有 18 个数获得了分解结果，它们分别是 RSA-100～RSA-704 的最小的 17 个数以及 RSA-768。表 1 是 2000 年以来 RSA 模数分解情况。关于更多 RSA 模数分解，可以查看文献[3]。

表 1 RSA 模数分解情况

RSA number	decimal digits	binary digits	factored on	factored by
RSA-150	150	496	Apr-04	Kazumaro Aoki
RSA-170	170	563	Dec-09	Bonenberger, *et al*[4]
RSA-180	180	596	May-10	Danilov, *et al*[5]
RSA-190	190	629	Nov-10	Timofeev[6]
RSA-200	200	663	Mar-05	Jens Franke[7]
RSA-210	210	696	Sep-13	Propper, *et al*[8]
RSA-704	212	704	Jul-12	Bai Shi, *et al*[9]
RSA-768	232	768	Dec-09	Kleinjung, *et al*[10]

目前，大数分解已经完全脱离了暴力试除的时代，研究者们已经提出了 ρ 方法、$P-1$ 法、椭圆曲线算法、随机平方法、二次筛法和数域筛法等大数分解技术。在过去的十多年里，为了保证安全，RSA 密钥的二进制长度由 512 位增长到了现在的 1024，并且安全性级别高的要求使用 2048 位。

近些年来，对于该问题的研究主要集中在纯理论上，对实现技术的研究则相对较少。宏观来看，具体可以归纳为以下几个方面：

收稿日期：2014-02-18；**修回日期**：2014-04-24 **基金项目**：国家“863”计划资助项目(2012AA012502)；中国科学院战略性先导科技专项基金资助项目(XDA06030602)

作者简介：刘新星(1989-)，女，湖南衡阳人，硕士，主要研究方向为信息安全(liuxinxing@iie.ac.cn)；邹潇湘(1976-)，男，湖南望城人，高级工程师，博士，主要研究方向为网络安全；谭建龙(1974-)，男，湖南宁乡人，研究员，博导，博士，主要研究方向为网络安全.

a)算法创新与改进。迄今为止,大数因子分解问题既没有被证明多项式时间可解,同时也未证明 RSA 的破译与模数分解是否同等困难。目前已经出现了十几种大整数分解的算法,具有代表性的大数分解算法有试除法、二次筛法(quadratic sieve,QS)、椭圆曲线算法(elliptic curve method,ECM)以及数域筛法(number field sieve,NFS);在过去两年也有新的算法出现如文献[11,12]。当然,前者是基于量子计算机,后者时间复杂度未知,新算法的出现并没有取代已经熟知的算法。下文将一一介绍当前流行的、典型的整数算法。此外,关于对当前流行的算法的改进也是该问题的研究热点之一,研究最热的当属对数域筛法的改进,如文献[13],这只是对数域筛法多项式的改进。数域筛法目前最高效,但复杂又难以控制、选择合适的参数对性能影响很大,对它的研究与改进自然是最热的。具体的改进将在 3.3 节介绍。

b)分布式计算与并行处理。大整数的分解经过几个世纪的研究,人们还没有发现多项式可解的算法。近些年来,随着分布式处理技术的发展,人们开始希望从并行处理的角度加快大规模模数的分解,文献[14 ~ 16]等可以归纳为这一类。近年来,不管是采用网格技术还是 Hadoop,主要还是将二次筛法部署到分布式平台上,数域筛法颇为复杂,这也是没有选择数域筛法的原因。二次筛法和数域筛法的介绍详见后文。

c)基于硬件平台的设计与实现。由摩尔定律可知,硬件技术发展迅速,新的硬件平台层出不穷,将现有的整数分解算法移植到硬件平台从而加速整数分解,也是主要的研究点之一。文献[17 ~ 19]是对椭圆曲线算法的硬件化;文献[20]则是用硬件来加速二次筛法;文献[21]结合 Xilinx FPGA 实现了数域筛法,设计了专用整数分解设备,并且第一次给出了用硬件实现整数分解的实现和实验数据,成功分解了 RSA-128。硬件平台的多样化给这一问题带来了很多研究空间,当前椭圆曲线(ECM)硬件化的研究比二次筛法或者数域筛法都更热,这主要是因为 ECM 的算子更能适应流水线结构。关于 ECM 介绍见 2.4 节。

可以看出,如上展开的三个方面的研究主要是围绕以后的椭圆曲线算法、二次筛法和数域筛法三者进行,其中数域筛法被认为是目前最好的算法。在现有的十几种大数分解算法,根据能被分解整数是否具备特殊的形式,可将它们分为专用算法和通用算法;根据算法性能是否依赖于被分解数的光滑性,可将它们分为基于光滑性的算法和基于组合同余的算法两类。本文将按照后者分类方法展开对各个算法的介绍。

2 基于光滑性的算法

所谓基于光滑性的算法,表示这些算法的性能直接取决于待分解的数是否满足某些特殊的光滑性。例如,是否具有较小的质因子、是否具有质因子,使得能分解为一些较小质数的乘积或者与有关的群的阶是否具有一定的光滑性等。这一类算法包括试除法、ρ 方法、$p \pm 1$ 方法以及椭圆曲线算法。

2.1 试除法

试除法(trial division)也称为穷举法,当要分解的数 N 本身很小,或者 N 有一个很小的质因子时,可以采用这种方法。其基本思想非常简单,就是穷举搜索 N 所有可能的因子。需要指出的是,对于随机的 N,有 50% 的可能性能被 2 整除,有 33% 的可能性能被 3 整除,等等。一般地,随机给定整数,有小于 x 因子的概率为

$$P(x) = 1 - \prod_{p<x}(1 - \frac{1}{p})$$

也就是说,约 88% 的正整数有一个小于 100 的因子,约 91% 的正整数有一个小于 1000 的因子。这样,如果随机给定一个数 N 需要进行分解,用试除法是不错的选择。

最坏情况下,试除法的时间复杂度为 $O(\sqrt{N})$。假如只枚举从 2 到$\sqrt{N}$的所有质数 p 来对 N 进行试除,根据素数定理,定义 $\pi(x)$ 为素数计数函数,即不大于 x 的素数个数,给函数这样一个估计:

$$\pi(x) \approx \frac{x}{\ln x}$$

因此,只需要检查 $O\left(\frac{\sqrt{N}}{\ln N}\right)$个素数,就可以找到 N 的因子。所以使用试除法,最快可达到 $O\left(\frac{\sqrt{N}}{\ln N}\right)$。

当然,如果要分解的数 N 较大时,并且 N 没有小的因子,该方法就基本上无能为力了。

2.2 ρ 方法

ρ 方法[22],也叫蒙特卡罗方法(Monte Carlo),是 Pollard 于 1975 年提出的一种分解算法,Brent 于 1980 年对其进行了改进[23]。蒙特卡罗方法是一种基于随机数的方法,适合具有小因子的整数 N。

该方法的基本思想如下:假设 N 是某个需要分解的数,p 是其质因子,并且假设 p 相对于 N 是一个较小的数。x_0 是 Z/nZ 上的一个随机整数,$f(x) \in Z[x]$是多项式,建议使用$f(x) = x^2 + 1$ 或$f(x) = x^2 - 1, x_0 = 2$。考虑如下拟随机序列$\{x_0, x_1, x_2, \cdots, x_i, \cdots\}$,其中 $x_{i+1} = f(x_i) \bmod N$。如果 $x_i \equiv x_j (\bmod p)$,而根据序列的定义,这两个数模 N 不相等,即 $x_i \not\equiv x_j (\bmod N)$,这样 $g = \gcd(|x_i - x_j|, N)$就是 N 非平凡因子。

因此,该算法的实质就是要找到拟随机序列中两个模 p 同余的数。虽然事先并不知道 p,因此无法直接判断 $x_i \equiv x_j (\bmod p)$ 是否成立,但是可以通过计算 $g = \gcd(|x_i - x_j|, N)$,并判断 $1 < g < N$ 是否成立来间接知道这一点。

为了得到两个模 p 同余的数,需要生成多少个拟随机数呢? 这个问题与数学中著名的生日问题有关,该问题是说,随机选择 k 个人,那么 k 至少应该多大,才能使得其中有两个人是同一天生日的概率大于 50%? 通过简单的计算发现,$k = 23$ 即可保证有两人是同一天生日的概率超过一半,远远小于直觉所预料的值。同样地,为了使得有两个数模 p 同余的概率大于 50%,需要选择 k 个数,k 满足

$$P_2(k,p) = 1 - \frac{p-1}{p}\ \frac{p-2}{p}\ \cdots\ \frac{p-(k-1)}{p} > 50\%$$

有如下近似值:

$$P_2(k,p) = 1 - \frac{p!}{k!p^k} \approx 1 - e^{-k(k-1)/2p} \approx 1 - (1 - \frac{k}{2p})^{k-1}$$

大约选择了 $k = 1.117\sqrt{p}$(下面记为 $c\sqrt{p}$)个数后,得到两个数模 p 同余的概率超过 50%。

如何选择合适的 $f(x)$ 以便能更有效地找到因子,在 Pollard 的原始论文中,建议使用:

$$f(x) = x^2 + 1 \text{ 或 } f(x) = x^2 - 1, x_0 = 2$$

蒙特卡罗方法启发式运行时间是 $O(n^{1/4}\log^2 n)$,相对来说比试除法好。事实上,该方法只对有小质数因子的整数有效

率。基于蒙特卡罗方法,Brent 等人[24]在 1981 年成功地分解了第 8 个费马数,找到了一个 16 位数字的因子。

2.3 $P-1$ 法

Pollard[25]在提出了 ρ 方法之后不久,还提出了另一种方法,称为 $P-1$ 法。假设当 N 有质因子 p,并且 $p-1$ 是一个平滑数时,使用这种方法比较有效。

$P-1$ 方法基于费马小定理,该定理是说,对于质数 p 和整数 a,当 a 与 p 互质时,有 $a^{p-1}\equiv 1(\bmod p)$。根据这一定理,对于任意整数 k,有 $a^{k(p-1)}\equiv 1^k\equiv 1(\bmod p)$。因此对于 $p-1$ 的任意倍数 M,有 $a^M\equiv 1(\bmod p)$,即 p 整除 a^M-1,又 p 整除 N,因此 $p\mid \gcd(a^M-1,N)$,通过求 $\gcd(a^M-1,N)$ 就有可能得到 N 的因子。

构造 $p-1$ 的一个倍数 M 很简单,可以给定一个上界 B,将小于 B 的一些质数或质数的幂乘起来得到 M。这个方法还是有一定效果的,根据 Brent 的介绍,通过 $P-1$ 方法找到了 $2^{977}-1$ 的一个因子 p_{32},这是一个 32 位十进制数:

$$p_{32}=49858990580788843054012690078841$$

此时

$$p_{32}-1=2^3\times 5\times 13\times 19\times 977\times 1231\times 4643\times 74941\times 1045397\times 11535449$$

对于给定的上界 B,执行一次 $P-1$ 算法的时间复杂度为 $O(B\times\log B\times\log^2 N)$。$P-1$ 算法能否有效地找到 N 的因子,取决于 N 的因子 $p-1$ 是否是平滑的;最坏情况下,$(p-1)/2$ 是一个质数,此时 $P-1$ 法不会优于试除法。

2.4 椭圆曲线算法

椭圆曲线方法(elliptic curve method,ECM)是 Lenstra[26]于 1987 年提出来的一种亚指数级复杂度的整数分解方法。这种方法使用的想法与 $P-1$ 法类似,两个算法成功分解的前提是它们的群的阶平滑,与 $P-1$ 方法不同的是,ECM 用随机选择的椭圆曲线的加法群来取代 $P-1$ 方法中的乘法群 F_p。

ECM 使用了椭圆曲线的群结构,假设 N 是被分解的数,随机选择一条椭圆曲线 $E:y^2z=x^3+axz^2+bz^3$,如果 (x,y,z) 满足该方程,并且 $c\neq 0 \bmod p$,则 (cx,cy,cz) 也满足该方程,因此 (x,y,z) 和 (cx,cy,cz) 可以看成是等价的。用 $(x:y:z)$ 表示包含 (x,y,z) 一类等价点的等价类。

在这个群 $E_{a,b}$ 中的加法零元 O 是 $(0:1:0)$,此时 $z=0 \bmod p$,即 z 是 p 的倍数,因此在群 $E_{a,b}$ 中,如果某一步运算得到了加法零元 $O=(x,y,z)$,那么通过计算 $\gcd(N,z)$ 就能得到一个大于 1 的值,这样就可能将 N 分解。

设 $P_1=(x_1:y_1:z_1)$ 为椭圆曲线上的一点,$nP_1=P_1+P_1+\cdots+P_1$(共 n 个 P_1 相加),记 $P_n=nP_1=(x_n:y_n:z_n)$。对于一个大整数 r 来计算 P_r,一般地,r 是所有小于给定上界 B_1 的质数幂的乘积。群 $E_{a,b}$ 的阶为 g,如果 g 是 B_1 幂平滑的,那么显然有 $g\mid r$,而有限群的元素的阶一定是群的阶的因子,因此 P_r 等于加法零元,通过计算 $\gcd(N,z)$ 可以分解 N。

在 Pollard 的 $P-1$ 方法中,如果第一步处理中算法没有成功分解 N,即 $p-1$ 不是平滑的,那么可以转入第二步,即允许 $P-1$ 有一个质因子大于给定上界。基于同样的思想,Montgomery 等人提出可以在 ECM 方法上加第二阶段的处理,改进后的 ECM 方法允许群 $E_{a,b}$ 的阶 g 有一个大于 B_1 但小于 B_2 的因子,这里 $0<B_1<B_2$ 为预先给定的上界。

在 $P-1$ 方法中,乘法群 F_p 的阶是 $p-1$,这种方法只有 $P-1$ 平滑时才能分解 N。同样地,ECM 若能成功分解 N,要求加群的阶也是平滑的。如果某次尝试 ECM 没有成功分解 N,可以重复随机选择新的椭圆曲线来提高成功率,ECM 的期望运行时间是

$$O(\exp((\sqrt{2}+O(1))(\log p)^{1/2}(\operatorname{loglog} p)^{1/2}))$$

其中:p 为 N 的最小素因子。

这个界是所有前面介绍的大整数因子分解算法中最好的一个界,因此椭圆曲线是所有基于平滑性的分解算法中实际运行最快的。特别地,当 N 具有小素数因子时,ECM 运行更快。此外,ECM 还有一个显著的优点,即它占用的内存很少,而后文要介绍基于组合同余的算法需要大量的存储单元。

Brent[27,28]使用 ECM 分解了第十和第十一费马数。2012 年 Wagstaff 也使用 ECM 分解出了一个 79 位数的因子,这是 ECM 分解最好最新的记录[29]。

3 基于组合同余的因子分解算法

基于组合同余的思想为:得到平方同余式 $x^2\equiv y^2(\bmod N)$ 左右两边的整数 x 和 y,并通过计算 $\gcd(x\pm y,N)$ 来得到非平凡的因子。它们最终的目的都是找到两个整数 x 和 y,$x\not\equiv \pm y(\bmod N)$ 使得 $x^2\equiv y^2(\bmod N)$,最后计算 $\gcd(x+y,N)$ 和 $\gcd(x-y,N)$ 得到非平凡的因子。这种思想相对于试除、基于随机数和基于费马小定理来说是一个全新的突破。当前最优秀的二次筛法和数域筛法就是基于这样的思想。

3.1 基本概念

在介绍基于组合同余这一类算法前,先介绍几个基本概念:

a) 平滑性。数 n 说是平滑的(smooth),如果对于 n 的所有质因子 P_i,都有 $P_i\leqslant B$。

b) 因子库。质数的集合 $F=\{P_1,P_2,\cdots,P_m\}$ 称为因子库(factor base)。数 n 说是 F 上平滑的,如果 n 的所有质因子都出现在 F 中。

3.2 随机平方法

Dixon 随机平方法为了能够有效进行整数因子分解,得到同余式 $x^2\equiv y^2(\bmod N)$,基本思想为:给定一个因子库 $F=\{p_1,p_2,\cdots,p_m\}$,生成一系列的随机数 $r_i>\sqrt{N}$,使得 $f(r_i)\equiv r_i^2(\bmod N)$ 的左边在 F 上是平滑的。寻找足够多的这样的同余式,并选择这些同余式的一个子集 U,使得乘积中的每个质因子的指数都是偶数形式,就得到了满足平方模 N 同余条件的 x 和 y,接下来计算 $\gcd(x\pm y,N)$ 即可。

找到满足所需条件的集合 U,就是寻找向量之间的线性相关。许多重要的算法,包括二次筛法、数域筛法,最后都有寻找向量之间的线性相关的步骤,这种 01 矩阵是有限域 GF(2)上的大型稀疏矩阵,解决的办法可以采用高斯消元。1994 年 Coppersmith[30]提出了 Block Wiedemann 算法,1995 年 Montgomery[31]提出了 Block Lanczos 算法,这两种块形式的算法都有效缩减了高斯消元的复杂度。

3.3 二次筛法

二次筛法(quadratic sieve,QS)[32,33]是 20 世纪 80 年代和 90 年代初最为有效的一种方法,能够有效地处理十进制 50 ~ 100 位的整数,包括了在因子库上寻找平滑整数、通过高斯消除法求向量相关性等步骤,是一种相对独立成熟的方法。该算法关键的一步,是用因子库里的质数对某个区间内的二次函数

值进行筛选,这也是该算法被称为二次筛法的原因。

二次筛法的主要流程如下:

a)选择因子库 $F=\{p_1,p_2,\cdots,p_m\}$;

b)定义多项式 $Q(x)=(x+[\sqrt{N}])^2-N$,选择不同的 x 计算得到一系列的函数值 $Q(x_a),Q(x_b),Q(x_c),\cdots$;

c)采用筛法对这些函数值进行筛选,找出其中在 F 上平滑的函数值 $Q(x_1),Q(x_2),\cdots,Q(x_k)$;

d)求 01 矩阵的线性相关性,得到一系列平滑值 $Q(x_1),Q(x_2),\cdots,Q(x_k)$ 后,对于每个同余式 $Q(x_i)$,构造一个 m 维向量 v_i,表示 $Q(x_i)$ 在因子库 F 中的各个质因子的指数的奇偶性,如果能求得 $e_i\in\{0,1\}$,使得

$$v_1e_1+v_2e_2+\cdots+v_ke_k\equiv O(\bmod 2)$$

那么那些 e_i 为 1 所对应的 $Q(x_i)$ 的乘积,就是一个完全平方。这相当于求一组线性相关的向量 v_i,如果向量的个数 k 大于向量的维数 m,则一定能找到一组线性相关的向量,求解向量的相关性可以通过高斯消除法实现。

现在,假设得到 $Q(x_{i,1})Q(x_{i,2})\cdots Q(x_{i,r})$ 是一个平方值 y^2,取

$$x=(x_{i,1}+[\sqrt{N}])(x_{i,2}+[\sqrt{N}])\cdots(x_{i,r}+[\sqrt{N}])$$

那么 $x^2\equiv y^2(\bmod N)$,最后求 $\gcd(x\pm y,N)$ 即可。

寻找平滑值的过程中需要进行大量的除法运算,筛选是 QS 中计算量最大的部分,如图 1 所示,成为了整个算法的性能瓶颈。

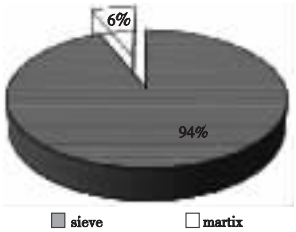

图1 筛选部分耗时比例

毫无疑问,若要提高整个算法的效率,必须有效减少筛选部分的计算量。在实际算法中,二次筛法使用了一些技巧来进一步减少除法运算。在筛选过程中,用 $\log(Q(x))$ 代替 $Q(x)$,用 $\log(Q(x))-\log p$ 代替 $Q(x)/p$ 即可。这里不必存储精确的对数值,只要存储估计值即可,最后扫描那些 $\log(Q(x))$ 接近于 0 的值。由于使用的是对数粗略值,因此会累积一些误差,$\log(Q(x))$ 可能不会严格等于 0。对于这些位置上的 $Q(x)$,用因子库里的质数进行试除,进一步判断其是否是 F 上的平滑数。同直接对所有 $Q(x)$ 进行试除不同,由于只有极少的 $Q(x)$ 可能是平滑的,这里试除的运算量几乎可以忽略不计。

在基本二次筛法中只使用一个二次多项式,有一种较好的改进,这种改进的方法称为多重多项式二次筛法(multiple polynomial quadratic sieve,MPQS)[34,35],是由 Montgomery 在与 Pomerance 的私人通信中提出来的[36]。MPQS 方法顾名思义,使用了多个多项式,这些多项式形如

$$Q(x)=ax^2+2bx+c,\quad \text{其中 } N\mid b^2-ac$$

这样

$$aQ(x)=a^2x^2+2abx+ac=$$
$$(ax+b)^2-(b^2-ac)\equiv(ax+b)^2\bmod N$$

1994 年 4 月,来自多个国家的 600 名志愿者花了将近 8 个月的时间,采用多重多项式二次筛法为 RSA-129[37] 找到了 64 位数和 65 位数两个素数因子。这就是历史上著名的 RSA-129 行动计划。

二次筛法的另一个改进是:允许 $Q(x)$ 有一个不在 F 中的质因子 q,q 小于给定的界限 B_2。这样,如果另外又找到了一个 $Q(x')$ 也是只有一个不在 F 中的质因子 q,那么将这两个式子乘起来得到

$$Q(x)Q(x')=q^2s$$

其中:s 在 F 是平滑的。由于 $Q(x)Q(x')$ 的因子分解式中 q 已经是平方的形式,这样就可以把 $Q(x)Q(x')$ 当成是 F 上平滑的来使用。二次筛法的这种变种可以节省约 50% 的计算时间,当然相应地会增大存储的需求。

二次筛法运行时间估计是

$$O(\exp((1+O(1))(\log N)^{1/2}(\log\log N)^{1/2}))$$

二次筛法的算法特点是能够很好地进行并行化。在区间 $[a,b]$ 进行筛选的过程可以在 n 个节点上并行处理,这只需要给每个节点分配一个互不相交的区间,每个节点的区间内期望包括 $|m/n|$ 个平滑的 $Q(x)$ 即可。但是,二次筛法还有另外一个瓶颈,就是矩阵部分产生的大规模的 01 稀疏矩阵,需要大量的内存,普通计算机无法满足。

3.4 数域筛法

数域筛法(number field sieve, NFS)[38,39] 由 Pollard 于 1993 年提出,针对不同的分解对象,NFS 分为特殊数域筛法(special number field sieve, SNFS)和通用型数域筛法(general number field sieve, GNFS)。特殊数域筛法(SNFS)分解的整数具有 $n=r^e\pm s$ 的特殊形式,其中 r 和 $|s|$ 比较小,e 可能很大。一般数域筛法(GNFS)则分解一般形式的整数。这里主要介绍 GNFS,NFS 一般指 GNFS,下文如果没有特别说明,NFS 都表示 GNFS。

GNFS 对二次筛法进行了扩展,允许更高次的多项式。当然,所付出的代价不能像二次筛法那样,可以直观地、直接地得到 Z/NZ 中的一个平方值。

假设 n 是将被分解的数。首先,选择一个合适的多项式 f 和有理整数 m,使得 $f(m)\equiv 0(\bmod n)$,α 是多项式 f 的一个复数根。可以定义如下一个从 $R=Z[\alpha]$ 到整数域的一个环同态:

$$\varphi:R\to Z_n,\text{即 }\varphi(\alpha)=m(\bmod n)$$

则有

$$X^2=\varphi(\beta)^2=\varphi(\beta^2)=\varphi(\prod_i(a_i+b_i\alpha))=$$
$$\prod_i(a_i+b_im)\equiv Y^2(\bmod n)$$

这样求得同余式的 X、Y 即为所需。

为了得到满足

$$\prod_i(a_i+b_i\alpha)=\beta^2$$
$$\prod_i a_i+b_im=Y^2$$

的整数对 (a,b),需要将 $Z[\alpha]$ 和 Z 上平滑的概念结合起来,寻找整数对 (a,b) 使得 $a+b\alpha$ 在 $Z[\alpha]$ 的"代数"因子库上平滑,同时 $a+bm$ 在 Z 的"有理"因子库上平滑。当找到足够的在两个因子库上同时平滑的整数对之后,在 $Z[\alpha]$ 和 Z 上同时产生一个平方值。

数域筛法可以分为如下几个步骤:

1)选取多项式 f_1 f_2

多项式的选取是该算法中的一个重要环节,它直接影响着后期关系对 (a,b) 的产量,对整个算法的运算时间影响很大。通常这一步选择得到两个互质的多项式 f_1、f_2,并且这两个多

项式模 n 时有相同的根。最简单的情况下使用 M-基方法,即

$$n=\sum_{i=0}^{d}c_i m^i,0\leqslant c_i<m$$

得到

$$f_1(x)=\sum_{i=0}^{d}c_i x^i,f_2(x)=x-m$$

可以发现,$f_1(m)\equiv f_2(m)\equiv 0 \bmod n$,$f_1$、$f_2$ 模 n 有相同的根 m。

如何选择好的多项式 f_1、f_2 对,减少筛选时生成关系对(a,b)的难度与耗时是近些年来的研究热点。现在的生成多项式的方法分成线性和非线性两大类算法。所谓线性算法,即多项式对 f_1、f_2 中含有一个线性多项式,上面介绍的 M-基方法就属于这一类。线性算法在数域筛法发展初期就开始有相应的研究[40],慢慢地 Montgomery 和 Murphy[41] 以及 Kleinjung[42, 43] 对多项式选择进行了改进,这些改进成就了很多创记录的大数分解,RSA-768 就是最好的例子。采用线性类算法会导致两个多项式指数差距大,因此寻找平滑数时在代数域上比有理数域上消耗的资源更多,会影响关系对的产生。与线性类算法正好相反,非线性算法[42, 44] 则选择两个指数相差不大或者一样的多项式,这样就平衡了两个多项式之间选择的平滑数的基和时间。实际上,由于这种方式并没有被过多的实际工程采用,而且限于目前的方法只能产生两个二次多项式,因此,非线性算法已经落后于线性算法了。

2) 筛选数对

在进行筛选工作前,需要确定三个基,$a+bm$ 对应的有理因子基 B_1 以及 $a+b\alpha$ 对应的代数因子基 B_2,以及检验是否是 $Z[\alpha]$ 中的一个完全平方值的二次特征基 B_3。

筛选出足够的(a,b)对,满足 $a+bm$ 和 $a+b\alpha$ 都很大可能在相应的因子基下完全分解。这一步进行了大量的除法操作,它是整个数域筛法中最耗时的部分。为了使得费时的除法运算降到最低,一个很好的主意是用快速的加减法[45] 取代通常的除法。在检验 B_1 上平滑性的时候,不是用除法运算,而是用加减法排除大部分不平滑的 $a+bm$ 值,只有当 $a+bm$ 几乎肯定是平滑的时候才使用试除法进行。所采用的策略是:在筛选数组中,存储的是 $\ln(a+bm)$ 的估计值,而非 $a+bm$。根据对数运算的基本性质,将 $a+bm$ 除以质数 p,等价于从 $\ln(a+bm)$ 中减去 $\ln p$,然后用 B_1 中的质数进行试除,检验这些 $a+bm$ 的平滑性。

这种一般的算法就是线筛,第一次被 Lenstra 等人[38] 提出。1993 年 Pollard[46] 在线筛的基础上进行了改进,提出了格筛的方法,上文介绍的降低除法运算的策略同样适用在格筛里,现在各大工程项目,比如文献[47]等,都是采用格筛。紧接着在 1994 年,Golliver 等人[48] 在 Pollard 的工作基础上加入了试除的操作,进一步改善了格筛的效率;此后在 2005 年,Franke 等人[49] 则提出了在格筛的基础上采用连分数来改善筛选的办法。

3)矩阵求解

与二次筛法一样,求解线性相关是为了得到平方数,不过这里需要得到有理数因子基 B_1 和代数因子基 B_2 下的两个平方数。求解方法与二次筛法一样。

定义 $\pi(B_i)$ 为因子基 B_1 的大小,那么三个因子基构成的向量大小为 $\pi(B_1)+\pi(B_2)+\pi(B_3)$,因此,至少要找到 $\pi(B_1)+\pi(B_2)+\pi(B_3)+1$ 组关系(a,b),才能使 $v_1e_1+v_2e_2+\cdots+v_ke_k\equiv O(\bmod 2)$ 有解。

被分解的模数过大时,矩阵将会变得很庞大,例如 RSA-768 分解过程中,使用的矩阵达到了 192,796,550×192,796,550,普通 PC 机无法满足需求,这一步对庞大内存的需求,也是所有基于组合同余算法的瓶颈。

目前数域筛法中,矩阵求解都是采用 Block Wiedemann 或者 Block Lanczos 的算法。RSA-768 团队大型稀疏矩阵 192,796,550×192,796,550 的求解就是采用 Block Wiedemann。

4)求平方根

平方根的求解涉及到代数域。当找到 $\prod_{(a,b)\in U}(a+b\theta)=\beta^2$,$\prod_{(a,b)\in U}(a+bm)=Y^2$ 后,Y 的值自然容易得出,X 的值需要求解 $Z[\theta]$ 上的代数平方根,然后利用同态 $\varphi(\beta)=X$ 得到。1993 年,Lenstra 等人[38] 介绍了求解平方根的步骤,Couveignes[50] 结合中国剩余定理,提出了新的求解平方根的方法。后者被使用得更广泛,不过它只适用多项式的次数为奇数的情况。

数域筛法被认为是当前最好的因式分解算法,自 1993 年被提出后,不断刷新分解记录,创建了一个又一个大数分解的里程碑。

1996 年,Cowie 等人[51] 使用数域筛法分解了 RSA-512,意味着从此 RSA-512 将不再安全;2009 年 12 月,RSA-768[10] 被成功分解,找到了两个二进制长度 384 位的因子,这是到目前为止整数分解最好的成果。

除了一般形式的模数分解之外,还有类似于 $n=r^e\pm s$ 这样特殊的数需要分解,通常采用特殊性数域筛法。2012 年,Childers[52] 介绍了 SNFS 分解的梅森数 $2^{1061}-1$,这是特殊数域筛法的一个里程碑,也是至今 SNFS 分解的最好的记录。其他比较好的成果有 2007 年 Aoki 等人[53] 分解的 $2^{1039}-1$。

数域筛法的时间复杂度为

$$O\left(\exp\left(\left(\frac{64}{9}+O(1)\right)^{1/3}(\log N)^{1/3}(\log\log N)^{2/3}\right)\right)$$

如同 QS 一样,NFS 不可避免地拥有两个瓶颈,一个是寻找平滑数的效率,该部分占有整个过程约 90% 的时间;另一个是内存。NFS 流程复杂,不像二次筛法和椭圆曲线算法那样简易,这也是近些年来;人们在分布式环境或者硬件环境下没有采用数域筛法的主要原因。

4　对比与分析

4.1　各类大数分解算法对比

如上介绍的两类算法中,椭圆曲线和二次筛法适合分解中等规模的整数,即二进制长度 100~110 位左右;数域筛法则适合分解大规模整数,这样的数一般大于 110 bit。试除法和其他算法当前使用不多,适合分解规模比较小的整数。表 2 对各个因式分解算法特点作了总结。结合各个算法的特点,它们的优缺点分析如下:

a)$P\pm1$ 法,实质上是假定对于 N 的某个质因子 p、$p\pm1$ 是平滑的,因此能够通过一些同余式的特点,在较小的搜索范围内找出这个因子来,从这个意义上来说,试除法就可以称为 P 法,因为该方法要求 N 具有一个较小的质因子 p,也就是说 p 是平滑的。RSA 公钥密码算法,选择参数 $N=pq$,要求 $p-1$,$p+1$,$q-1$,$q+1$ 应该有大质数因子,就是为了防止被 $P\pm1$ 法攻击。

b)椭圆曲线算法相对于其他中等规模整数分解算法而言,如 QS(或者 MPQS),具有结构简单的特点,内存使用量也

少,只不过通用性不够,不像 QS(或者 MPQS)那样任何整数都可以分解。

c)几乎所有的通过寻找形如 $x^2 \equiv y^2 \pmod N$ 形式平方差的大数因子分解方法,都使用了因子库、平滑、寻找 $Z/2Z$ 上向量相关性等基本概念。要想取得关键突破,必须用更少的时间在因子库上产生更多平滑的值。例如,QS 方法对 Dixon 方法的改进就是使用了不同的多项式,同时更重要的是改变了寻找平滑值的方法,通过使用筛选的过程来得到平滑值,这是整数分解技术方面的一个突破。GNFS 的一个重要突破,是认识到不必像 Dixon 方法和 QS 方法那样限制使用二次多项式,也许某些三次、四次、五次甚至更高次的多项式,能比二次多项式产生更多的平滑值。

根据上面的分析,可以对各个算法的优点和缺点进行总结如表 3 所示。

表 2 典型大数分解算法的特点

算法名称	时间复杂度	分解对象
试除法	$O(\sqrt{N})$	小规模整数
ρ 方法	$O(N^{\frac{1}{4}}\log^2 N)$	小规模整数
$P-1$ 方法	$O(B\times\log B\times\log^2 N)$	小规模整数
椭圆曲线算法(ECM)	$O(\exp((\sqrt{2}+O(1))(\log p)^{\frac{1}{2}}(\log\log p)^{\frac{1}{2}}))$	中等规模整数
二次筛法(QS)	$O(\exp((1+O(1))(\log N)^{\frac{1}{2}}(\log\log N)^{\frac{1}{2}}))$	中等规模整数
数域筛法(GNFS)	$O\left(\exp\left(\left(\frac{64}{9}+O(1)\right)^{\frac{1}{3}}(\log N)^{\frac{1}{3}}(\log\log N)^{\frac{2}{3}}\right)\right)$	大规模整数

表 3 典型大数分解算法优缺点比较

算法名称	优点	缺点
试除	简单、并行度好,适合分解随机选取的或者拥有小质数因子的整数	时间复杂度最高,只适用于分解小规模整数
$P\pm1$	$P\pm1$ 具有小质因子的数容易被分解	通用性不好,实用性不强
ECM	结构简单,内存使用量少,是所有基于平滑性的算法中运行最快的	通用性不够
QS	并行度高,通用性强	内存需求高
SNFS	能够分解特殊型的中等规模和大规模整数,并行度好	算法复杂,通用性不够,内存需求高
GNFS	分解大规模整数最快的算法,通用性高,并行度好,实用性强	算法复杂,内存需求高,工程难度大

4.2 分析与展望

图 2 介绍了过去十年,椭圆曲线算法(ECM)以及一般数域筛法(GNFS)和特殊数域筛法(SNFS)分解记录发展趋势。

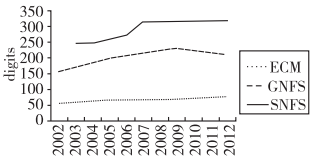

图 2 ECM和NFS分解记录

从历史分解记录来看,随着计算机计算能力的扩大,差不多每 10 年会多分解掉 100 digits 的数。可以估计,在实用型的量子计算机没有面世之前,只要拥有足够的人力和财力,NFS 可于 2020 年左右分解现在通行的 RSA-1024(309 digits)。从实际情况来看,目前所拥有的算法存在算法复杂性高、算法流程复杂、耗资巨大等实质性问题,硬件加速大数分解的研究也停留在初始阶段,较好的成果仅仅只是对数域筛法作的一些改进,从 1993 年至今都没有在算法或架构上跨越性的进展。分解 1024 bit 以上的 RSA number 仍然是一项耗资巨大的工程难题,大整数分解问题仍然被认为是困难的。

从总体上看,未来大数分解研究方向会在并行的前提下,尽可能提高系统效率,基于新的计算机体系结构设计出新的高度并行的适合硬件的算法,采用专用硬件设备加速大整数分解很有可能是未来整数分解新的发展方向。

5 结束语

对大整数因子分解的研究不仅具有挑战性还具有重要的理论和应用价值,它涵盖的领域包括计算机科学、数论、代数、密码学。几百年来,众多优秀学者加入其研究行列,在算法上取得了不错的进展,但仍然不清楚该难题属于哪个计算复杂性类。本文介绍了当前流行的大整数因子分解算法,详细分析比较了它们的优缺点,展望了大整数因子分解之路的发展趋势。

参考文献:

[1] RIVEST R L, SHAMIR A, ADLEMAN L. A method for obtaining digital signatures and public-key cryptosystems[J]. Communications of the ACM, 1978, 21(2): 120-126.

[2] AGRAWAL M, KAYAL N, SAXENA N. PRIMES is in P[J]. Annals of Mathematics, 2004, 160(2): 781-793.

[3] RSA Laboratories. The RSA factoing Challenge[EB/OL]. (2013). http://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-factoring-challengehtm.

[4] BONENBERGER D, KRONE M. Factorization of RSA-170[R]. Wolfenbiittel: Ostfalla Clniversity of Applied Sciences, 2010.

[5] DANILOV S, POPOVYAN I. Factorization of RSA-180[EB/OL]. IACR (2010-05-09). http://eprint.iacr.org/2010/270.pdf.

[6] POPOVYAN I, TIMOFEEV A. RSA-190 factored[EB/OL]. (2010-11-10). http://maths-people.anu.edu.au/~bai/factor/rsa190.html.

[7] KLEINJUN T. We have factored RSA200 by GNFS[EB/OL]. (2005-05-09). http://www.loria.fr/~zimmerma/records/rsa200.

[8] PROPPER R. RSA-210 factored[EB/OL]. (2013-10-17). http://www.mersenneforum.

[9] BAI Shi, THOMÉ E, ZIMMERMANN P. Factorisation of RSA-704 with CADO-NFS[EB/OL]. (2010-05-09)[2012-07-05]. http://eprint.iacr.org/2012/369.pdf.

[10] KLEINJUNG T, AOKI K, FRANKE J, *et al*. Factorization of a 768-bit RSA modulus[C]//Advances in Cryptology. Berlin: Springer, 2010: 333-350.

[11] 付向群, 鲍皖苏, 周淳, 等. 具有高概率的整数分解量子算法[J]. 电子学报, 2011, 39(1): 35-39.

[12] DENG Ying-pu, PAN Yan-bin. An algorithm for factoring integers[BE/OL]. (2012-02-29). http://eprint.iacr.org/2012/097.pdf.

[13] BAI Shi,ZIMMERMANN P. Size optimization of sextic polynomials in the number field sieve[EB/OL]. (2012-12-03). http://hal. inria. fr/docs/00176103/31/PDF/sopt. pdf.

[14] 李骏. 分布式计算环境下大整数分解的研究[D]. 上海:上海交通大学, 2007.

[15] GHEBREGIORGISH S T. Quadratic sieve integer factorization using Hadoop[D]. Stavanger:University of Stavanger,2012.

[16] TORDABLE J. MapReduce for integer factorization [EB/OL]. (2010-01-04). http://www. javiertordable. com/files/MapreduceForIntegerFactorization. pdf.

[17] ZIMMERMANN R, GUNEYSU T, PAAR C. High-performance integer factoring with reconfigurable devices[C]//Proc of International Conference on Field Programmable Logic and Applications. Washington DC:IEEE Computer Society, 2010: 83-88.

[18] ATANASSOV E, GEORGIEV D, MANEV N L. ECM integer factorization on GPU cluster[C]//Proc of the 35th International Convention. 2012:328-332.

[19] LI Lei, HAN Wen-bao. Hardware implemention of the pipeline architecture of integer factorization based on elliptic curve method[J]. Journal of Information Engineering University, 2012,13(1):13-17.

[20] ARCHER C. GPU Integer factorisation with the quadratic sieve[D]. Bath:University of Bath, 2010.

[21] IZU T, KOGURE J, SHIMOYAMA T. CAIRN: dedicated integer factoring devices[C]//Proc of the 13th International Conference on Network-based Information Systems. Washington DC:IEEE Computer Society, 2010: 558-563.

[22] POLLARD J M. A Monte Carlo method for factorization[J]. BIT Numerical Mathematics, 1975,15(3):331-334.

[23] BRENT R P. An improved Monte Carlo factorization algorithm[J]. BIT Numerical Mathematics,1980,20(2):176-184.

[24] BRENT R P, POLLARD J M. Factorization of the eighth Fermat number[J]. Mathematics of Computation, 1981,36(154):627-630.

[25] POLLARD J M. Theorems on factorization and primality testing[J]. Mathematical Proceedings of the Cambridge Philosophical Society,1974, 76(3): 521-528.

[26] Jr LENSTRA H W. Factoring integers with elliptic curves[J]. Annals of Mathematics,1987,126(3):64-73.

[27] BRENT R D. Factorization of the tenth Fermat number[J]. Mathematics of Computation of the American Mathematical Society, 1999,68(225):429-451.

[28] BRENT R. Factorization of the tenth and eleventh Fermat numbers [R]. Canberra:Australian National University, 1996.

[29] ZIMMERMANN P. 50 largest factors found by ECM [EB/OL]. [2013]. http://www. loria. fr/ ~zimmerma/records/top50. html.

[30] COPPERSMITH D. Solving homogeneous linear equations over GF(2) via block wiedemann algorithm[J]. Mathematics of Computation, 1994,62(205):333-350.

[31] MONTGOMERY P L. A block Lanczos algorithm for finding dependencies over GF(2)[C]//Advances in Cryptology . Berlin:Springer, 1995:106-120.

[32] POMERANCE C. The quadratic sieve factoring algorithm[C]// Advances in Cryptology. Berlin:Springer,1985:169-182.

[33] LANDQUIST E. The quadratic sieve factoring algorithm[C]//Advances in Cryptology. Berlin:Springer,1985:169-182.

[34] SILVERMAN R D. The multiple polynomial quadratic sieve[J]. Mathematics of Computation, 1987,48(177):329-339.

[35] GUAN D J. Experience in factoring large integers using quadratic sieve[R]. Kaohsiung: National Sun Yat-Sen University,2005.

[36] POMERANCE C. A tale of two sieves[J]. Notices of the American Mathematical Society,1996,43(1):1473 – 1482.

[37] ATKINS D, GRAFF M, LENSTRA A K,*et al*. The magic words are squeamish ossifrage[C]//Advances in Cryptology. Berlin: Springer, 1995:261-277.

[38] LENSTRA A K,Jr LENSTRA H W. The development of the number field sieve[M]. Berlin:Springer,1993.

[39] LANDQUIST E. The number field sieve factoring algorithm[D]. Kutztown:Kutztown University, 2002.

[40] BUHLER J P,Jr LENSTRA H W, POMERANCE C. Factoring integers with the number field sieve[M]//The Development of the Number Field Sieve. Berlin:Springer,1993:50-94.

[41] MURPHY B A. Polynomial selection for the number field sieve integer factorisation algorithm[D]. Canberra:Australian National University, 1999.

[42] KLEINJUNG T. On polynomial selection for the general number field sieve[J]. Mathematics of Computation, 2006,75(256):2037-2047.

[43] KLEINJUNG T. Polynomial selection[C]//Proc of CADO Workshop on Integer Factorization. 2008.

[44] PREST T, ZIMMEMANN P. Non-linear polynomial selection for the number field sieve[J]. Journal of Symbolic Computation,2012,47(4):401-409.

[45] CAVALLAR S, DODSON B, LENSTRA A K,*et al*. Factorization of a 512-bit RSA modulus[C]//Advances in Cryptology- EUROCRYPT. Berlin:Springer,2000:1-18.

[46] POLLARD J M. The lattice sieve[M]//The Development of the Number Field Sieve. Berlin:Springer, 1993:43-49.

[47] BAI Shi, GAUDRY P, KRUPPA A,*et al*. CADO-NFS, an implementation of the number field sieve[EB/OL]. (2011-10-27). http://cado-nfs. gforge. inria. fr.

[48] GOLLIVER R A, LENSTRA A K, McURLEY K S. Lattice sieving and trial division[C]//Proc of the 1st International Symposium on Algorithmic Number Theory. Berlin:Springer, 1994:18-27.

[49] FRANKE J, KLEINJUNG T. Continued fractions and lattice sieving [C]//Proc of the 1st Workshop on Special-Purpose Hardware for Attacking Cryptographic Systems. 2005.

[50] COUVEIGNES J M. Computing a square root for the number field sieve[M]//The Development of the Number Field sieve. Berlin: Springer,1993:95-102.

[51] COWIE J,DODSON B,ELKENBRACHT-HUIZING R,*et al*. A world wide number field sieve factoring record: on to 512 bits[C]//Advances in Cryptology. Berlin:Springer, 1996: 382-394.

[52] CHILDERS G. Factorization of a 1061-bit number by the Special Number Field Sieve[EB/OL]. (2012-08-06). http://eprint. iacr. org/2012/444. pdf. 2012:444.

[53] AOKI K, FRANKE J, KLEINJUNG T,*et al*. A kilobit special number field sieve factorization[C]//Advances in Cryptology. Berlin:Springer,2007:1-12.