计算机安全技术丛书

BigNum Math：加密多精度算法的理论与实现

[美] Tom St Denis 编著

尹浩琼 等译

学习如何实现有效的多精度算法

- 逐步构建概念体系
- 完全覆盖Karatsuba乘法、Montgomery缩减和模幂乘等内容
- 伪代码和真正可现场移植的C源代码示例

中国水利水电出版社
www.waterpub.com.cn

计算机安全技术丛书

BigNum Math：加密多精度算法的理论与实现

[美] Tom St Denis 编著

尹浩琼 等译

中国水利水电出版社

内 容 提 要

大数运算是加密和安全领域必不可少的一部分，要想实现它，既需要相应的数学理论知识，又需要一定的编程技巧。对于每一个初学者，要想掌握它，必定要花费大量时间查阅数学书本和C语言教程（也可能是别的语言）。

本书作者为了方便初学者学习及业内人士使用，开发了一个免费的大数运算库，即LibTomMath项目。结合LibTomMath库，由浅入深对各种大数运算的算法进行了阐述。对每一种运算一般都列出多种算法，并对其性能进行比较。

本书适合于对算法、IT安全、加密领域感兴趣的读者阅读。

北京市版权局著作权合同登记号：图字 01-2006-7277

图书在版编目（CIP）数据

大数运算：加密多精度算法的理论与实现 /（美）圣丹尼斯（Denis, T.S.）编著；尹浩琼等译. —北京：中国水利水电出版社，2008

（计算机安全技术丛书）

书名原文：BigNum Math：Implementing Cryptographic Multiple Precision Arithmetic

ISBN 978-7-5084-5022-3

Ⅰ. 大… Ⅱ. ①圣…②尹… Ⅲ. 电子计算机－算法理论 Ⅳ. TP301.6

中国版本图书馆CIP数据核字（2007）第155128号

书　　名	BigNum Math：加密多精度算法的理论与实现
作　　者	[美] Tom St Denis　编著
译　　者	尹浩琼　等译
出版　发行	中国水利水电出版社（北京市三里河路6号　100044） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）63202266（总机）、68331835（营销中心）、82562819（万水）
经　　售	全国各地新华书店和相关出版物销售网点
排　　版	北京万水电子信息有限公司
印　　刷	北京蓝空印刷厂
规　　格	787mm×1092mm　16开本　15印张　397千字
版　　次	2008年1月第1版　2008年1月第1次印刷
印　　数	0001—4000册
定　　价	30.00元

凡购买我社图书，如有缺页、倒页、脱页的，本社营销中心负责调换

前　言

本书的诞生是我人生的一个有趣的经历。那段时期我从一个涉世未深的年轻人成长为一个周游列国、结识了无数新朋友和同事的软件开发者。这一切开始于2001年12月，大约5年前，我开始了一个后来名为LibTomCrypt的项目，该项目被业内广泛使用。

我从事LibTomCrypt项目的最初目的是想把我的精力集中在某些有建设性的事情上，同时学习一些新技能。从事该项目的第一年，我学会了如何组织产品、归档文件，以及如何进行后续的支持和维护。大约在2002年冬季，我开始寻求另一个项目以打发时间。由于意识到LibTomCrypt缺乏数学支持，于是着手开发一个新的数学库。

这样LibTomMath项目就诞生了。它最初只是现有项目的补丁集，很快就发展为一个单独的项目。从头编写这个数学库是生产一个稳定和独立的产品的基本原则。我还学会了哪些算法可用来进行如模幂等这样的运算。这个库仅经过几个月的开发就非常稳定和可靠，并且立即投入使用。

2003年夏，我又开始寻找另一个项目。由于意识到只实现这些数学例程不足以真正理解它们，于是我开始尝试自己对它们进行诠释。在此过程中，我最终掌握了这些算法背后的概念。这些知识正是我想传递给读者们的。本书实际上取材于我自己的网站（www.libtomcrypt.com）上的一些公开素材。

当我向别人提起LibTom项目（共有6个）并且告诉他们我准备公开发布时，他们都觉得很诧异。他们问我为什么这么做，尤其是为什么还要继续免费进行下去。我能做出的最好的解释就是“因为我能够”。这对于成人间的对话来说有些奇怪，并且过于简单。我通常进一步解释为“我有这个能力，并且我愿意”，这也许更能说明问题。我是第一个承认我所做的并无特殊之处的人，也许还有其他人也这么看，那么我们就可以组成一个令人自豪的小团体。我的LibTom项目是我正在以工具和知识的形式回馈社会、能给他人带来帮助的东西。

我编写本书是因为它能进一步实现我的学术开放的目标。LibTomMath的源代码本身很容易领悟和学习。但有很多时候，纯粹的C代码不能恰当地解释本书中的算法。本书首先解释该库的基础，然后再介绍更复杂的算法。伪代码和源代码的同时使用提供了全世界计算机专业学生更容易接受的“理论”和“实践”的双重结合。我从未太偏离相对直接易懂的代数，并且希望本书能成为有价值的学习资料。

如果没有大量好心人在时间、资源以及友善的言辞等方面的帮助，就不可能有本书以及大部分LibTom项目现在的这个样子。编写一部长书（以及源代码）是一个累人且冗长的过程。目前，LibTom已面世5年了，它有成千上万的使用者，包含了10万多行源代码、TEX和其他资料。Mads Rassmussen和Greg Rose从一开始就鼓励我完成这项工作，有时候别人的认可能极大提升我继续完成该项目的勇气。当然，我的父母也给了我很大帮助，在2003年的数月里为我提供食宿。

Greg和Mads是该项目早期阶段的重要支持者。2003年8月发表的本文初稿是几个月

专职工作的结晶。长时间的工作，以及还需要去学校读书持续不断地消耗了我的能量，如果没有他们的支持，我不可能坚持下去。

当然如果没有各种 LibTom 项目的成功，也就没有这本书。它们的成功不仅仅是我努力工作的结果，还包含了其他成百上千人的贡献。有 Colin Percival、Sky Schultz、Wayne Scott、J Harper、Dan Kaminsky、Lance James、Simon Johnson、Greg Rose、Clay Culver、Jochen Katz、Zhi Chen、Zed Shaw、Andrew Mann、Matt Johnston、Steven Dake、Richard Amacker、Stefan Arentz、Richard Outerbridge、Martin Carpenter、Craig Schlenter、John Kuhns、Bruce Guenter、Adam Miller、Wesley Shields、John Dirk、Jean-Luc Cooke、Michael Heyman、Nelson Bolyard、Jim Wigginton、Don Porter、Kevin Kenny、Peter LaDow、Neal Hamilton、David Hulton、Paul Schmidt、Wolfgang Ehrhardt、Johan Lindt、Henrik Goldman、Alex Polushin、Martin Marcel、Brian Gladman、Benjamin Goldberg、Tom Wu 和 Pekka Riikonen 在项目开发的各个阶段花费了宝贵的时间，贡献了很多想法、更新、补丁，或者是鼓励。我要感谢这些年我认识的很多朋友，感谢他们为我带来的美好时光以及鼓励。我希望通过本项目向他们表达敬意。

感谢 Syngress 出版社的编辑们，他们在短短的一周内审阅了 300 多页的稿子，并且纠正了很多问题。我还要感谢我未曾提到的很多朋友，他们总能给我鼓励，并且能为我带来欢乐。感谢我的朋友 J Harper、Zed Shaw 和 Simon Johnson，他们在我交稿之前对本书进行了审阅。感谢 Secure Science Corporation 的 Lance James 以及 Elliptic Semiconductor 的全体同仁们，他们为我提供了大量的后期开发时间，把我送到了 Toorcon，并且给我介绍了很多现在我已经认识的人。

开放源代码、开放学术、开放思想。

Tom St Denis
多伦多，加拿大
2006 年 5 月

※　　※　　※

本书由尹浩琼翻译，在翻译过程中得到了张波、欧阳宇、易磊、盛海燕、安晓梅、徐红霞的帮助，其中张波审校了全文，在此一并致谢。

目　　录

第 1 章　引言

1.1　多精度算术

1.1.1　什么是多精度算术

在进行连续算术运算时，如加法或乘法，很少意识到我们已经不自觉地提高或降低了正在处理的数值的精度。比如，在十进制中，我们几乎可以立即算出 6 乘以 7 等于 42。但是，42 是两位数，而乘数和被乘数只有 1 位。另外，42 再乘以 3，结果是一个更大的数 126。在上述几个例子中，我们处理的数有多个精度。尽管精度不同，但是可以设计一个算法子集[①]去适应它们。

比较后发现，在各种运算中固定精度或单精度运算会损失精度。例如，在固定精度的十进制系统中，6 乘以 7 等于 2。

本质上，基于计算机的多精度算术与在学校学习的手算的加减乘除一样。

1.1.2　为什么需要多精度算术

最需要多精度算术（通常被称为“大数”数学）的当属公钥加密算法。像 RSA[10]和 Diffie-Hellman[11]这样的算法需要很大的整数来抵御已知的密码攻击。比如，一个典型的 RSA 模数至少大于 10^{309}，但像 ISO C [17]和 Java [18]这样的现代编程语言仅支持相对较小且单精度的整数。

ISO C 语言[②]提供的最大的数据类型只能表示最大为 10^{19} 的数值，如图 1.1 所示。C 语言本身并不能解决目前遇到的大数问题。利用目前的一般桌面计算机可轻松找出 10^{19} 数量级的 RSA 模数的因子[③]，从而使基于该算法的所有协议变得非常不安全。多精

① 不定期地进行优化。

② 虽然 ISO C 标准是这么定义的，但每个编译器供应商可根据自己需要进行扩充。

③ Pollard-Rho 因式分解法只需要进行 2^{16} 次运算。

度算法使用单精度数据类型扩展了可表示的整数范围，从而解决了上述问题。

数据类型	范围
char	-128～127
short	-32768～32767
long	-2147483648～2147483647
long long	-9223372036854775808～9223372036854775807

图 1.1 C 编程语言的典型数据类型

快速多精度算术领域中的大部分进步都来源于对更快更高效加密本原序列的需求。快速的模减和模幂算法，比如已经出现在各种加密期刊上的 Barrett 约简算法，可使像 RSA 和 Diffie-Hellman 这样的算法更高效。实际上，像 RSA Security、Certicom 和 Entrust 几个大公司已经将其整个产品线构建在高效算法的实现和部署上。

但是，密码学并不是唯一从快速多精度整数例程中受益的研究领域。多精度整数的另一个辅助用途是用于高精度浮点数据类型。基本的 IEEE [12]标准浮点类型由整数型的尾数 q、指数 e 和符号位 s 组成。数字以 $n = q \cdot b^e \cdot -1^s$ 的形式表示，其中 b=2 为 IEEE 最常用的基。由于 IEEE 浮点数在硬件中实现，所以其尾数的精度通常很小（23、48 和 64 位）。尾数只是一个整数，因此可用一个多精度整数创建比硬件本身支持的精度要高得多的尾数。这种方法在科学计算应用程序中非常有用，因为它需要在经过冗长的计算后，使输出结果的误差最小。

大整数的另一个用途是构建大特征值的多项式（比如，寻找伽罗华域 GF(p)[x]的大 p）。实际上，本文讨论的库就已经用于组建一个多项式基础库①。

1.1.3 多精度算术的优势

多精度表示法相对于单精度或固定精度表示法的优势在于，当运算所需精度超出范围时，其结果的表示精度不受损失。比如，两个 n 比特整数相乘，至少需要 2n 比特的精度才能真实地反映结果。多精度算法会增加中间变量的精度，以满足结果所需的精度，而单精度系统会截断多余的位数，以保持固定的精度。

用固定精度算法实现需要大整数的算法也是可能的。比如，常用于智能卡的椭圆曲线密码学（ECC），将整数的精度固定在系统所需的最大值上。这种方法是一种提供

① 详情请见http://poly.libtomcrypt.org。

主机平台都无法提供的整数的非常简单的算法[①]。尽管这种方法可能很有效，但是其源代码的灵活性较差。当运行时的输入值大小超出设计者预期时，它不能自动适应。

多精度算法比其他任何算法的开销都要大。虽然通过精心规划，大部分开销都可以保持在最小水平，但总的来说，它不适合消耗内存的平台。但是，就输入值的范围来说，多精度算法的灵活性最高。也就是说，同样基于多精度整数的算法，不需要设计者预先考虑即可适应任何大小的输入。这就降低了代码的拥有成本，因为它只需编写和测试一次。

1.2　本书目的

本书的目的是告诉读者如何实现高效的多精度算法。更准确地说，解释了算法背后的一小部分理论，以及被其他同样题材书籍的作者忽略的各种“零碎的”东西。有几本书（见参考文献[1]、[2]）详细介绍了这些算法的理论知识，但是却很少提到具体实现方面的内容。

大部分情况下，一种算法的理论解释和实际实现是两个完全不同的概念。比如，Handbook of Applied Cryptography（HAC）一书第 594 页的算法 14.7 提供了一种相对简单的多精度整数加法，但它没有考虑到两个整数输入的数量级可能不同的情况。结果，其实现就不像它力图让读者相信的那样简单。同样，其中一个除法例程（598 页的算法 14.20）没有说明如何处理符号或被除数在主循环中逐渐减小的问题（step #3）。

这两本书也没有提到所需的几种关键的最优算法，比如，“Comba”和 Karatsuba 乘法以及快速模逆算法，我认为这是实践上的疏忽。这些最优算法对于在非平凡应用程序上获得任何有用性能非常重要。

为了解决该问题，本书的重点放在多精度整数包的实际实现方面。作为一个案例，LibTomMath 包[②]被用来演示算法的真正实现[③]，其中这些算法已经通过现场测试并工作正常。LibTomMath 库在 Internet 上对所有用户免费开放，本书讨论了该库大部分的内部机理。

本书提供的算法通常包括至少一个“伪代码”描述，然后紧跟着该算法的 C 语言

① 比如，一般的智能卡处理器有一个 8 比特的累加器。

② 可在 http://math.libtomcrypt.com 上找到。

③ 用 ISO C 实现。

实现。伪代码用来帮助读者用适合自己的其他编程语言实现该算法。

本书还可作为从头创建多精度算法的一个演练，它告诉读者这些算法如何结合在一起，以及从何处启动各项任务。

1.3 讨论和表示法

1.3.1 表示法

n 位多精度整数可表示为 $x=(x_{n-1},\cdots x_1,x_0)_\beta$，它表示整数 $x\equiv\sum_{i=0}^{n-1}x^i\beta^i$。数组中的 x 表示以 β 为基的每一位的值。比如，$x=(1,2,3)_{10}$ 表示整数 $1\cdot10^2+2\cdot10^1+3\cdot10^0=123$。

mp_int 是一个复合结构，它包含其所表示的整数的每一位值，以及操作这些数据所需的辅助数据，2.2.1 节中将进一步讨论这些附加数据。在本书中，“多精度整数”和 mp_int 具有同一含义。当一个算法被指定接受一个 mp_int 变量时，意味着同时还存在着辅助数据。一个 variablename.item 类型的表达式暗含着它应该求出该变量的名为 item 的成员的值。比如，一个字符串可能有一个 length 成员，它的值为该字符串中的字符数。如果该字符串 a 为"hello"，那么紧跟着它的就应该是 a.length=5。

在某些讨论中，为了帮助读者理解解决问题的最终算法，提供了更通用的算法。当说一个算法接受整数输入时，意味着输入是一个不带额外多精度数据的普通整数。也就是说，使用整数而不是 mp_ints 作为输入的算法不关心内部操作，如内存管理。这些算法将建立起相关理论，以描述后面解决同一问题的多精度算法。

1.3.2 精度表示法

变量 β 表示多精度整数中每一位数的基，它必须表示为 q^p 的形式，其中 $q,p\in\mathbb{Z}^+$。单精度变量必须能表示范围 $0\leqslant x<q\beta$ 的整数，而双精度变量必须能表示范围为 $0\leqslant x<q\beta^2$ 的整数。额外的基 q 因子使得在进行加减运算时不需要截取进位。由于所有现代计算机都是二进制的，所以假定 q 等于 2。

在为每个算法提供的源代码中，数据类型 mp_digit 表示一种单精度整数类型，而数据类型 mp_word 表示双精度整数类型。在其中几个算法中（尤其是 Comba 例程），临时结果就存放在双精度类型的数组 mp_words 中。在本书中，x_j 表示单精度数组的第 j 位，$\hat{x}_j$ 表示双精度数组的第 j 位。当一个表达式被赋给一个双精度变量时，其中所

有单精度变量在赋值过程中都被提升为双精度。赋给单精度变量的表达式，其变量都被截断，以适应单精度数据类型。

比如，如果 $\beta=10^2$，单精度数据类型可表示范围在 $0\leqslant x<10^3$ 的值，而双精度数据类型可表示范围 $0\leqslant x<10^5$ 的值。令 a=23 和 b=49 为两个单精度变量，则单精度乘积表示为 $c\leftarrow a\cdot b$，而双精度乘积表示为 $\hat{c}\leftarrow a\cdot b$。在上例中，$\hat{c}=1127$，$c$=127。单精度数据类型容纳不下乘积的最高位，因此 $c\neq\hat{c}$。

1.3.3　算法输入和输出

在算法描述中，所有变量都被指明是单精度或双精度的标量。唯一的例外是当变量被指定为 mp_int 类型时。这个区别非常重要，因为标量通常被用作数组的下标以及其他各种计数值。

1.3.4　数学表达式

括号 $\lfloor\ \rfloor$ 表示将一个整数截断为不大于该表达式的整数，比如 $\lfloor 5.7\rfloor=5$。同理，括号 $\lceil\ \rceil$ 表示将表达式舍入为不小于该表达式的整数，比如 $\lceil 5.1\rceil=6$。通常，除法符号/表示结果被截断的除法，比如 5/2=2。为清楚起见，上式常写作 $\lfloor 5/2\rfloor=2$。当表达式写作分数形式时，表示实数除法，比如 $\frac{5}{2}=2.5$。

多精度整数的范数，如 $\|x\|$ 表示该整数的位数。例如，$\|123\|=3$，$\|79452\|=5$。

1.3.5　算法的效率

本书用修改过的大 O 表示法衡量算法的效率。在该系统中，所有单精度运算被认为具有相同的代价[①]。也即单精度的加、乘和除所用时间相等。这样做只是为了简化讨论，实际中情况并非如此。

有些算法比其他算法略微好些，因此有些常数在表示法中没有被去掉。比如，正常的基线乘法（5.2.1 节）需要 $O(n^2)$ 的运算量，而基线平方（5.3 节）需要 $O\left(\frac{n^2+n}{2}\right)$ 的运算量。在标准的大 O 表示法中，这两者都等于 $O(n^2)$。而在本书中并不是这样，

① 除非明确说明。

因为输入变量的数量级通常相当小。因此，算法效率中很小的常数因子就可能导致相当明显的算法效率差异。

本书提供的所有算法都有一个用多项式表示的运算量，也即 $O(n^k)$ 的形式，其中 $n,k \in \mathbb{Z}^+$。这有助于比较各种算法的运算速度，以及了解各种优化如何最终体现出优势。

1.4 练习

在某些章节里，专门留出一小节为读者提供一些与正在讨论的内容有关的具有挑战性的练习。这些练习的目的并不是搞有奖竞答，而是想开拓读者的思路。所有问题尽可能设计得有前瞻性，在后面章节里都能找到答案。希望读者完成这些练习，以更好理解相关主题。

我已经说过，练习的目的在于巩固对相关知识的掌握。尤其希望学生读者在继续阅读之前确认能够回答这些问题。

与[1, pp. ix]中描述的练习类似，这些练习根据问题难度不同，具有不同的分数等级。但是，和[1]不同的是，问题不是很难。这些练习的分数从 1（最简单）到 5（最难）不等。图 1.2 总结了所用的评分标准。

[1]	简单问题，只需要读者花费几分钟即可解决。通常不需要通过计算机来解决
[2]	简单问题，需要花费少量计算机时间。通常需要编写一段代码来解决
[3]	中等难度问题，需要大量时间解决。通常需要进行大量研究，并从学生的观点提出新理论
[4]	中等难度问题，需要下很大功夫才能解决。问题的解决表明读者对相关内容已经基本掌握
[5]	较难问题，涉及一些初学者难以理解的概念。解决这些问题表明已经完全掌握相关内容

图 1.2 练习的评分系统

第一等级的问题非常简单，读者不需要编程或提出新理论即可解决。这些问题是一些快速测验，想看看读者是否理解相关内容。第二级别的问题也较简单，但是需要编程或者使用某种算法来解决。这两个级别的问题都是入门级问题。

第三级问题比前两级都难。答案通常很显而易见，但需要一些思考和一些技巧才能找到正确答案。解决这些问题通常需要一种新算法或者对之前提到的算法进行改进。

能够回答这些问题的读者能够理解问题背后的概念。

第四级别与第三级别的问题类似，但是需要更多的钻研才能完成。读者通常不能立即知道答案，文中也没有提供详细答案，但是在后面的章节中可以找到答案。

第五级的问题相对于本章中的其他所有问题来说是最难的。能够回答这些问题说明读者已经完全掌握了相关内容。

通常问题是互相关联的。目的是与后面章节中要讨论的内容联系起来。希望读者能够回答后面的问题，并试着找出其相关性。

1.5　LibTomMath 简介

1.5.1　什么是 LibTomMath

LibTomMath 是一个免费和开源的多精度整数库，完全用可移植的 ISO C 编写。可移植是指该库不包含依赖计算机平台或者容易在某些平台上出现问题的代码。

该库已经在很多操作系统上测试通过，包括 UNIX[①]、Mac OS、Windows、Linux、Palm OS 以及像 Gameboy Advance 这样的单机硬件上。该库的功能非常完善，足以开发出像公钥密码系统这样的应用程序，并且占用相对较少的空间。

1.5.2　LibTomMath 的目标

效率高的库很少用像 C 这样的高级语言编写。虽然该库完全用 ISO C 编写，但在库内的算法实现优化方面下了很大功夫。尤其是已编写的代码可在 x86 和 ARM 处理器平台上用 GNU C 编译器（GCC）编译，并且尽可能地提供了像 Karatsuba 乘法、滑动窗求幂以及 Montgomery 缩减这样的高效算法，以便使该库更加高效。

应用程序编程接口（API）尽可能地简单，即使对于几乎最优和专门的算法也是如此。通常，通用的占位符例程无需开发者特别留意就会自动使用专门算法。比如，通用乘法算法 mp_mul()会根据输入大小和库的配置自动使用 ToomCook、Karatsuba、Comba 或基线乘法。

高效并不是 LibTomMath 项目的唯一目标。理想情况下，该库应该与其他流行的

① 所有商标都属于相应的所有者。

库在源代码上兼容，从而使它对开发人员更有吸引力。这种情况下，使用 MPI 库作为所有基本函数的 API 模板。选择 MPI 是因为它是特别适合 LibTomMath 的另一个库。虽然 LibTomMath 使用 MPI 作为函数名和参数传递约定的模板，但是这个 MPI 是由 Tom St Denis 重写的。

该库也可用作学生的学习工具。从逻辑上说，不存在一个易懂的“大数”库，它能教会计算机系的学生如何编写快速可靠的多精度整数算法。为此，该库的源代码中注释很少，也很少有算法讨论点。

1.6 为什么选择 LibTomMath

选择 LibTomMath 作为本书的案例不仅因为这两者都是同一人所做，还有其他更重要的原因。其他像 GMP[13]、MPI[14]、LIP[16]和 OpenSSL[15]这样的库也有多精度整数算术例程，但不是本书的最佳选择，具体原因将在下面详细说明。

1.6.1 代码基

LibTomMath 代码基是完全可移植的 ISO C 源代码。这意味着代码中不包含依赖于平台的条件性语句。这种干净整洁的库对开发者来说，不需要知道哪些是条件性代码，从而更容易洞察源代码的真正含义。

LibTomMath 的代码基结构良好。每个函数都包含在一个单独的源文件中，使读者可快速找到指定函数。每个源文件平均有 76 行代码，使得源代码清晰易读。与之形成鲜明对比的是，MPI 和 LIP 只有一个源文件，使得阅读起来非常困难。GMP 中包含很多条件代码段，这也影响了它的易读性。

在 x86 处理器上使用 GCC 编译，且对速度进行优化时整个库大约只有 100KiB[①]，与 GMP（超过 250KiB）相比非常小。LibTomMath 编译后比 MPI（大约 50KiB）略大，但是速度更快，内容更完整。

1.6.2 API 简单易懂

LibTomMath 在 MPI 库之后设计，并且它有着一样的 API 设计。通常，使用 MPI

① KiB 表示 2^{10} 个 8 位码元（译者注：1K 字节），同理，MiB 指 2^{20} 个 8 位码元（译者注：1M 字节）。