DOI: 10.3969/j.issn.1671-0673.2021.01.014

一种基于行公因子提取的改进 Coppersmith 算法

王云飞,李光松

(信息工程大学,河南 郑州 450001)

摘要: 1996 年欧密会上,Coppersmith 提出一种对单变元模方程求小根的多项式时间算法,该算法对公钥密码系统的安全性分析具有重要意义。结合 Coppersmith 算法中格基矩阵的结构特点和元素性质提出一种改进算法,通过逐次提取格基矩阵不同块中行向量的公因子,有效降低了 Coppersmith 算法的求解时间。同时通过实验证明了此改进算法可有效兼容一种预处理算法,通过将这两种算法结合,进一步提高了 Coppersmith 算法的求解效率,实验表明较原 Coppersmith 算法最高可提升 22.64%。

关键词: Coppersmith 算法; LLL 算法; 格基约化; 公因子提取

中图分类号: TN918.1 **文献标识码:** A **文章编号:** 1671-0673(2021)01-0081-06

Improved Coppersmith Algorithm Based on Extraction of Row Common Factor

WANG Yunfei, LI Guangsong

(Information Engineering University, Zhengzhou 450001, China)

Abstract: In a seminal work at Eurocrypt1996, the polynomial-time Coppersmith algorithm has been proposed for finding small roots of univariate modular equations. This algorithm is of great significance to the security analysis of public-key cryptosystems. In this paper, an improved Coppersmith algorithm is proposed based on the structural characteristics and element properties of lattice basis matrix. This algorithm can effectively reduce the running time of the Coppersmith algorithm by successively extracting the common factors of the row vectors in different blocks of the lattice basis matrix. At the same time, the compatibility of this improved algorithm with a preprocessing algorithm is proved, and combining these two algorithms further improves the efficiency of Coppersmith algorithm, which can be up to 22.64% higher than the original Coppersmith algorithm.

Key words: Coppersmith algorithm; LLL algorithm; lattice reduction; common factor extraction

Coppersmith 在 1996 年的欧密会[1]上提出了一种高效求解单变元模方程小根的算法,通过构造模方程簇、构造格基矩阵、LLL 算法[2]约化、还原多项式等步骤将模方程转化到整数上求解。Coppersmith 算法的结果可叙述如下:给定一个首一 δ 次多项式 $f(x)\in\mathbb{Z}[x]$ 以及整数 N,N 的分解未知,Coppersmith 算法可以在 $(\log N,\delta)$ 的多项式时间内求解所有满足 $|x_0|<N^{1/\delta}$ 和 $f(x_0)\equiv 0(\bmod N)$ 的整根 x_0。

Coppersmith 算法通过寻找相应格中 LLL 约化基[2]将模方程转化到整数上求解,这组基可以通过 LLL 算法[2]得到。虽然 LLL 算法是一个多项式时间算法,但它的理论复杂度上界为高阶多项式,且实际应用中高维格上的 LLL 算法难以在短时间内结束,因此 Coppersmith 算法的整体时间主要取决于 LLL 算法时间。对于维数为 n 的格基方阵,

收稿日期: 2020-09-14; 修回日期: 2020-11-02

基金项目: 国家自然科学基金群体创新项目(61521003)

作者简介: 王云飞(1995-),男,硕士生,主要研究方向为信息安全、算法设计。

LLL 算法的复杂度为 $O(n^6\log^3 C)$，其中 C 为格基矩阵中行向量欧几里得模的上界。经过优化算法结构和降低运算浮点精度要求的 L^2 算法[3]将 LLL 算法的复杂度降为 $O(n^5\log^2 C+n^6\log C)$。文献[4]对使用 L^2 算法作为约化算法的 Coppersmith 算法的复杂度进行了分析，得到 Coppersmith 算法的复杂度上界为 $O(\delta^5\log^9 N)$。

Coppersmith 算法被提出以来被广泛应用到各领域实际问题的求解中，如对密码学领域中的 RSA 部分比特泄露问题的求解[5]。1997 年的欧密会上，文献[6]改善了 Coppersmith 算法中格基矩阵的构造方式，此构造方式极大降低了格基矩阵构造的难度，被广泛应用。2014 年，文献[7]在公钥密码学会议(PKC)上提及一种预处理算法，此预处理算法能够约化矩阵中非对角线元素至小于本列对角线元素。

本文首先提出了一种改进的 Coppersmith 算法，通过逐次提取格基矩阵不同块中公因子，缩短了 Coppersmith 算法求解时间，同时本文证明了该算法与上述预处理算法的兼容性，并将两种算法结合，进一步提高了 Coppersmith 算法求解效率，最后通过实验验证了本文所提算法的有效性及高效性。

1 预备知识

本文首先约定如下。符号 $\|\cdot\|$ 表示向量的欧几里得模，符号 $\langle\cdot,\cdot\rangle$ 表示两个向量的内积，$\boldsymbol{B}[i]$ 表示矩阵 $\boldsymbol{B}$ 的第 i 行向量，$\boldsymbol{B}[i][j]$ 表示矩阵 $\boldsymbol{B}$ 中的第 i 行第 j 列元素，称 Coppersmith 算法中构造的格基矩阵为 Coppersmith 矩阵，令 a 表示与 $a\in\mathbb{R}$ 最接近的整数(若 a 小数部分为 0.5，则取最接近整数中大者)。

1.1 格的基本概念

1840 年前后，高斯为解决开普勒猜想提出了格的概念，经过近 200 年的发展，格的理论得到极大丰富。格的研究与代数、几何、分析、数论等学科有着紧密的联系，同时在密码学、通信、计算机等领域有着广泛的应用。

定义 1 m 维线性空间 $\mathbb{R}^m$ 上的 n 个线性无关向量 $\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n$ 的整系数组合构成一组格，其可形式化的表示为 $L(\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n)=\{\sum_{i=1}^{n}x_i\boldsymbol{b}_i \mid x_i\in\mathbb{Z},i=1,\cdots,n\}$，称 $\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n$ 为格 L 的一组基，n 为格 L 的秩，m 为格 L 的维数。若 $n=m$，则称格 L 为满秩的。

令 $\boldsymbol{B}$ 表示以格基为行向量构成的矩阵，那么格 L 的行列式定义为 $\det(L)=\sqrt{\det(\boldsymbol{B}\boldsymbol{B}^T)}$。若格 L 为满秩，由格基构成的矩阵 $\boldsymbol{B}$ 则为方阵，并且有 $\det(L)=|\det\boldsymbol{B}|$，本文中 Coppersmith 矩阵即为方阵。

定义 2[8] 给定格 L，找一个非零格向量 $\boldsymbol{v}$，满足对任意非零向量 $\boldsymbol{u}\in L$，$\|\boldsymbol{v}\|\leqslant\|\boldsymbol{u}\|$，称此问题为最短向量问题(SVP)。

格 L 中的最短向量可能有多个，若某算法能够找到其中一个就称其解决了 SVP 问题。

1.2 LLL 算法

文献[2]在 1982 年提出的 LLL 算法是求解 SVP 问题的一个近似算法，其可在多项式时间内返回一组与格中最短向量指数近似的格基，称为 LLL 约化基。LLL 算法中两个关键的步骤为 Size-reduction 和 Lovász 条件判断。Size-reduction 通过长向量减去适当倍数短向量将向量的长度减小，即利用短向量约化长向量；Lovász 条件用于判断前后两个向量是否进行交换。Gram-Schmidt 正交化为 LLL 算法中的关键步骤，其中一组正交基 $\boldsymbol{b}_1^*,\boldsymbol{b}_2^*,\cdots,\boldsymbol{b}_n^*$ 由一组格基递归得到：$\boldsymbol{b}_i^*=\boldsymbol{b}_i-\sum_{1\leqslant j<i}\mu_{ij}\boldsymbol{b}_j$，其中 $\mu_{ij}=\dfrac{\langle\boldsymbol{b}_i,\boldsymbol{b}_j^*\rangle}{\|\boldsymbol{b}_j^*\|^2}$。$\mu_{ij}$ 是 Size-reduction 步骤中的倍数参数。

定义 3 设 $\{\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n\}\in\mathbb{R}^m$ 是格 L 的一组基，若满足：

① $\mu_{ij}\leqslant 1/2$，$1\leqslant j<i\leqslant n$；

② $\delta\|\boldsymbol{b}_{i-1}^*\|^2\leqslant\|\boldsymbol{b}_i^*\|^2+\mu_{i,i-1}^2\|\boldsymbol{b}_{i-1}^*\|^2$，$i=2,3,\cdots,n$，$1/4<\delta\leqslant 1$；

则称这组有序的基 $\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n$ 是以 δ 为参数的 LLL 约化基。

LLL 约化基有如下性质。

定理 1 设 $\{\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_n\}\in\mathbb{R}^m$ 是格 L 中以 3/4 为参数的 LLL 约化基，那么 $\|\boldsymbol{b}_1\|\leqslant 2^{(n-1)/4}\det(L)^{1/n}$。

1.3 Coppersmith 算法

Coppersmith 算法解决的是模方程 $f(x)=0\pmod N$ 求小根的问题，其有如下结果。

定理 2[9] 令 $f(x)=0\pmod N$ 为一个首一 δ 次模 N 的模方程，N 的分解未知。令 X 满足 $X\leqslant N^{1/\delta}$，则 Coppersmith 算法可以在 $(\log N,\delta)$ 的多项式时间内找到所有满足 $f(x_0)\equiv 0\pmod N$ 和 $|x_0|\leqslant X$ 的根。

下面叙述 Coppersmith 算法的主要过程，注意

到其中构造格基矩阵步骤中采用了 Howgrave-Graham 变体[8]。

首先构造一组含有 $n=h\delta$ 个元素的多项式簇:

$$g_{i,j}(x)=x^jN^{h-1-i}f^i(x)$$

其中 $0\leqslant i<h$,$0\leqslant j<\delta$,$h\geqslant 2$ 为整数。不妨设 x_0 为原模方程 $f(x)\equiv 0 \pmod N$ 的一个根,即 $f(x_0)\equiv 0 \pmod N$,则 $g_{i,j}(x_0)\equiv 0 \pmod{N^{h-1}}$。

根据 $g_{i,j}(Xx)$ 的各项系数构造格基矩阵,此时的格基矩阵为一个下三角的方阵。根据 LLL 算法约化的结果,可以得到一个上述多项式簇的整系数线性组合 $g(x)=\sum_{i,j}a_ig_{i,j}(x)$,$a_i\in\mathbb{Z}$,满足 $\|g(Xx)\|<\dfrac{N^{h-1}}{\sqrt{n}}$。

定理 3[8] (Howgrave-Graham)　令 $g(x)\in\mathbb{Z}[x]$ 为一个至多 n 项的多项式,假设 $g(x_0)=0 \bmod M$,其中 $|x_0|\leqslant X$ 且 $\|g(Xx)\|<\dfrac{M}{\sqrt{n}}$,那么 $g(x_0)=0$ 在整数上成立。

由定理 3 可知,$g(x_0)=0$ 在整数上成立。通过 Berlekamp-Zassenhaus 算法[9-10] 可以在整数上求解 $g(x)=0$,从而得到模方程 $f(x)\equiv 0 \pmod N$ 所有满足 $x<X$ 的根。

1.4　预处理算法

Coppersmith 算法构建格基矩阵后,可以通过预处理算法降低 LLL 算法输入的格基矩阵的元素规模。预处理算法的思想与 HNF(Hermite Normal Form) 算法[11] 原理类似,不同之处是 HNF 算法为右乘幺模矩阵对列进行变换使得原矩阵的行中元素小于本行对角线元素,而预处理算法为保持格的不变性对矩阵行向量进行变换使得原矩阵的列中元素小于本列对角线元素。由于 Coppersmith 矩阵为下三角矩阵,因此预处理的算法步骤为由 Coppersmith 矩阵右下角向左上角约化,具体算法步骤见算法 1。

算法 1　预处理算法

输入: 原始 Coppersmith 矩阵 $\boldsymbol{B}$

输出: 元素比特规模较小的 Coppersmith 矩阵

```
1.  for(i=n; i>0; i--)
2.      for(j=i+1; j≤n; j++)
3.          令 MU=⌊B[j][j]/B[i][i]⌋//MU 为整数
4.          B[j]=B[j]-MU*B[i]
```

2　Coppersmith 算法改进

本节首先对 Coppersmith 矩阵的构造方式进行研究,并在此基础上分析矩阵的结构特点、结合格基矩阵中元素性质提出改进的 Coppersmith 算法。

如 1.3 节所述,多项式簇的构造方式为 $g_{i,j}(x)=x^jN^{h-1-i}f^i(x)$,其中 $0\leqslant i<h$,$0\leqslant j<\delta$,$h\geqslant 2$ 为整数。那么在具有相同 i 的多项式中,系数必然有公因子 N^{h-1-i},由 Coppersmith 矩阵的构造方式可知,相应的格基中也具有公因子 N^{h-1-i},Coppersmith 矩阵的具体形式如下所示,空白处元素为 0。

$$\begin{pmatrix}
N^{h-1} & & & & & & & & & & & & \\
 & N^{h-1}X & & & & & & & & & & & \\
 & & \ddots & & & & & & & & & & \\
 & & & N^{h-1}X^{\delta-1} & & & & & & & & & \\
\hline
\ddots & \ddots & \ddots & \ddots & \ddots & & & & & & & & \\
\hline
- & - & \cdots & - & \cdots & NX^{h\delta-2\delta} & & & & & & & \\
 & - & \cdots & - & \cdots & - & NX^{h\delta-2\delta+1} & & & & & & \\
 & & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & & & & & \\
 & & & - & \cdots & - & - & \cdots & NX^{h\delta-\delta-1} & & & & \\
\hline
- & \cdots & \cdots & - & \cdots & \cdots & - & \cdots & \cdots & X^{h\delta-\delta} & & & \\
 & - & \cdots & - & \cdots & \cdots & - & \cdots & \cdots & - & X^{h\delta-\delta+1} & & \\
 & & \ddots & \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \\
 & & & - & \cdots & \cdots & - & \cdots & \cdots & - & - & \cdots & X^{h\delta-1}
\end{pmatrix}$$

将 Coppersmith 矩阵 $\boldsymbol{B}$ 按行进行 h 等分,每一部分称为一“块”,并且分别用 $1,\cdots,h$ 对它们进行标记,称为第 i 块,每一块中有 δ 行元素。注意到 Coppersmith 矩阵具有如下性质。

性质 1　第 i 块中的所有元素具有公因子 N^{h-i}。

由性质 1 可得,第 1 块中元素公因子最大,随着块号逐次增加,公因子会逐次减少因子 N。由此可在 LLL 约化前提取 Coppersmith 矩阵各块中公因子以提高 LLL 约化的效率,进而提高 Coppersmith

算法求解效率，具体算法步骤如算法2所示。

算法2 改进的Coppersmith算法

输入： 大整数 N，首一 δ 次模方程 $f(x)=0 \pmod N$，小根上界 X

输出： $f(x)=0 \pmod N$ 所有满足 $|x|<X$ 的根

1. 构造多项式簇 $g_{i,j}(x)=x^j f^i(x)$
2. 根据 $g_{i,j}(Xx)$ 的各项系数构造 Coppersmith 矩阵
3. for($i=1;i<h;i++$)
4. 以第 i 块首行格向量为起始向量对前 i 块元素进行 LLL 约化
5. 前 i 块元素中各个元素均乘以 N
6. 以第 h 块首行格向量为起始向量对 Coppersmith 矩阵进行 LLL 约化
7. 在整数上求解 $\boldsymbol{B}[1]$ 对应的多项式 $g(x)$ 的所有根
8. 输出上一步所求根中所有满足 $f(x_0)=0 \pmod N$ 的根 x_0

算法2在构建Coppersmith矩阵时直接将 N 的方幂略去，在前 $i(i<h)$ 块逐次完成LLL约化后再将前 i 块所有元素乘以 N，其中第6步保证了此步骤的输出为LLL约化基，从而保证了整个算法可正确求解。

算法2在前 $i(i<h)$ 块LLL约化时降低了复杂度中 $\log C$ 这一项的大小，提升了LLL算法的约化效率，但没有降低Coppersmith算法的理论复杂度。下面从约化步数和每步运算效率两方面给出LLL约化过程的效率分析。当LLL约化到阶段 k 时，利用 $b_i(0<i<k)$ 约化 b_k，称为一步约化。不妨设对格基矩阵 $\boldsymbol{B}$ 进行LLL约化后得到矩阵 $\boldsymbol{B}^R$，则有如下引理。

引理1 若 $\boldsymbol{B}$ 为一格基矩阵且有 $\boldsymbol{B}=c\tilde{\boldsymbol{B}}$，其中 c 为矩阵 $\boldsymbol{B}$ 中元素公因子，那么 $\boldsymbol{B}^R=c\tilde{\boldsymbol{B}}^R$，且 $\boldsymbol{B}$，$\tilde{\boldsymbol{B}}$ 进行LLL约化时约化步数相同。

证明 利用数学归纳法。第一步约化中由 $\boldsymbol{b}_1^*=\boldsymbol{b}_1$，$\boldsymbol{b}_i=c\tilde{\boldsymbol{b}}_i(i=1,2)$ 易得 $\mu_{21}=\tilde{\mu}_{21}$，$\boldsymbol{b}_2^*=c\tilde{\boldsymbol{b}}_2^*$，因此 $\boldsymbol{B}$，$\tilde{\boldsymbol{B}}$ 在第一步约化时Size-reduction的倍数和Lovász条件判断结果是相同的。不妨设约化进行到阶段 $k+1$ 时仍有 $\boldsymbol{B}=c\tilde{\boldsymbol{B}}$，此时前 k 个格基为LLL约化基，下面证明阶段 $k+1$ 约化时Size-reduction的倍数和Lovász条件判断结果相同。由 $\boldsymbol{B}=c\tilde{\boldsymbol{B}}$ 以及Gram-Schmidt正交化过程可得 $\boldsymbol{b}_i=c\tilde{\boldsymbol{b}}_i$，$\boldsymbol{b}_i^*=c\tilde{\boldsymbol{b}}_i^*$ $(0<i\leqslant k)$。由 $\mu_{k+1,i}=\dfrac{\langle \boldsymbol{b}_{k+1},\boldsymbol{b}_i^*\rangle}{\|\boldsymbol{b}_i^*\|^2}(0<i\leqslant k)$，$\boldsymbol{b}_{k+1}^*=\boldsymbol{b}_{k+1}-\sum_{1\leqslant i\leqslant k}\mu_{k+1,i}\boldsymbol{b}_i$ 可知 $\mu_{k+1,i}=\tilde{\mu}_{k+1,i}$，$\boldsymbol{b}_{k+1}^*=c\tilde{\boldsymbol{b}}_{k+1}^*$，因此阶段 $k+1$ 约化时Size-reduction的倍数和Lovász条件判断结果相同。综上可得，$\boldsymbol{B}$，$\tilde{\boldsymbol{B}}$ 约化过程中每步的Size-reduction的倍数和Lovász条件判断结果相同，也就意味着约化的总步数相同。

定理4 算法2与原Coppersmith算法中LLL约化步数相同。

证明 令 $\boldsymbol{B}_i$ 表示前 i 块元素构成的矩阵，那么有 $\boldsymbol{B}_i=N^{h-i}\tilde{\boldsymbol{B}}_i$。由引理1可得，矩阵 $\boldsymbol{B}$，$\tilde{\boldsymbol{B}}$ 在每块LLL算法约化完成时相应的约化步数相同，末块约化完成时约化步数也相同，因此约化总步数相同。

LLL算法中的运算主要由浮点数运算与大整数运算组成，浮点数运算一般较为高效。Coppersmith矩阵中元素比特规模大多在上万比特，而大整数运算一般由大量整数运算组合实现（如GMP库），因此大整数比特规模越大，运算效率越低。算法2中通过提取大整数的公因子，降低了前 $i(i<h)$ 块中每步约化时的大整数比特规模，提高了大整数运算的效率。又因为算法2与Coppersmith算法中约化总步数相同，因此算法2效率更高。

本节将算法2与第2节所述预处理算法结合，进一步提高Coppersmith算法的效率，具体步骤如算法3所示。下面定理证明了算法2与预处理算法的兼容性。

定理5 在算法2中LLL约化步骤前应用算法1对Coppersmith矩阵进行预处理，不会影响算法2中Coppersmith矩阵每块所提取公因子的大小。

证明 注意到预处理算法是由Coppersmith矩阵的右下角向左上角进行约化，意味着其中的每步约化都可形式化的表达为 $\boldsymbol{B}[j]=-MU^*\boldsymbol{B}[i](j>i)$。由于 $\boldsymbol{B}[j]$ 的公因子总是不小于 $\boldsymbol{B}[i]$ 的公因子，预处理算法不会影响Coppersmith矩阵每一块中公因子的大小。因此在算法2中LLL约化之前，应用算法1对Coppersmith矩阵进行预处理，不会影响算法2中Coppersmith矩阵每块所提取公因子的大小。

算法3 首先以Howgrave-Graham方式构建Coppersmith矩阵，并且在完成预处理算法之后再将各块中的公因子提出，而算法2则是在构建Coppersmith矩阵时直接将N的方幂略去，这是两种算法建格时的不同之处。算法3通过应用预处理算法提前对格基进行了一部分约化，降低了LLL算法输入矩阵的元素比特规模大小，进一步

提升了 Coppersmith 算法的求解效率。实验结果也表明,预处理算法的加入会提升 Coppersmith 算法效率。

算法 3　与预处理算法结合的改进 Coppersmith 算法

输入: 大整数 N,首一 δ 次模方程 $f(x)=0 \pmod N$,小根上界 X

输出: $f(x)=0 \pmod N$ 所有满足 $|x|<X$ 的根

1. 构建 Coppersmith 矩阵
2. 对 Coppersmith 矩阵应用预处理算法
3. for($i=1; i<h; i++$)
4. 　提取第 i 块中公因子 N^{h-i}
5. 　以第 i 块首行格向量为起始向量对前 i 块元素进行 LLL 约化
6. 　前 i 块元素中各个元素均乘以 N
7. 以第 h 块首行格向量为起始向量对 Coppersmith 矩阵进行 LLL 约化
8. 在整数上求解 $\boldsymbol{B}$ [1] 对应的多项式 $g(x)$ 的所有根
9. 输出上一步所求根中所有满足 $f(x_0)=0 \pmod N$ 的根 x_0

3　实验结果及分析

本节在 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz 的硬件平台上实现了原 Coppersmith 算法及本文中的 3 种算法。本节实验中 Coppersmith 算法及其改进算法均采用 Howgrave-Graham 变体,且选用 L^2 算法进行 LLL 约化。为保持算法实验结果对比的一致性,所有算法均利用 GMP 库中 MPZ 以及 MPF 模块进行 C 语言实现,且实验结果均取 10 次实验的平均值。

参照文献 [12] 所述问题,本节选取了次数为 3、模数为 2 048bits 的 RSA 型模数的首一模方程进行求解实验。此问题中 Coppersmith 矩阵维数为 $n=3*\ h$,考虑到实际求解时间,本节分别选择 $h=8,10,12,14,16,18,20$ 进行实验,并且在不同 h 时选取相同的 X,所有实验均正确的求出了小根,具体的实验结果如表 1 所示,表中时间单位均为秒,其中“预 Coppersmith 算法”表示仅应用预处理的 Coppersmith 算法。表 1 中“性能提升 1”和“性能提升 2”两项分别表示算法 2、算法 3 相比于原 Coppersmith 算法的性能提升效果,可见算法 2、算法 3 的效率比原 Coppersmith 算法分别最高提升 16.28%、22.64%。

表 1 结果显示,预 Coppersmith 算法在维数较小时优于算法 2,随着维数的增长,算法 2 的效率逐渐优于预 Coppersmith 算法,说明本文提出的改进 Coppersmith 算法在高维时表现较好。算法 3 在低维、高维时都明显优于预 Coppersmith 算法、算法 2,说明本文中改进 Coppersmith 算法与预处理算法的结合是有效的。

表 1　不同算法实验结果对比

h	算法				性能提升 1(%)	性能提升 2(%)
	原 Coppersmith 算法	预 Coppersmith 算法	算法 2	算法 3		
8	30.58	28.03	29.72	26.82	2.81	12.30
10	121.85	114.14	114.30	104.85	6.20	13.95
12	390.86	366.84	357.49	329.18	8.54	15.78
14	1 006.41	958.76	920.00	854.72	8.59	15.07
16	2 338.19	2 209.16	2 081.20	1 935.18	10.99	17.24
18	5 207.64	4 751.57	4 359.59	4 028.40	16.28	22.64
20	9 687.30	9 126.15	8 464.80	7 865.60	12.62	18.80

本节进一步分析了 $h=18$ 时从 LLL 算法开始至每一块完成 LLL 约化的时间,结果如图 1 所示,其中 t_0, t_1, t_2, t_3 分别表示原 Coppersmith 算法、预 Coppersmith 算法、算法 2、算法 3 完成相应块约化的时间。本节截取图 2 中的部分数据形成表 2,以更详细地分析各算法的约化过程。表 2 中 4 种算法约化第一块的时间都为 0 s,这是因为第一块中格基为正交基,并且易证其满足定义 3 为 LLL 约化基。表 2 数据还显示,采用行公因子提取方法的算法 2、算法 3 在前几块 LLL 约化的时候速度明显优于原 Coppersmith 算法、预 Coppersmith 算法,例如完成第 6 块约化时算法 2 是原 Coppersmith 算法速度的 3 倍左右,这是因为在前期约化时提取的行公因子较大。虽然由原 Coppersmith 算法和算法 2 的对比结果易见不同块约化的时间差在不断累积,但是随着块号的增加,格基数量增加,约化时间增多,公因子减小,算法的性能提升效果降低,最终算法 2 完成整个 LLL 约化时相比于原 Coppersmith 算法性能提升了 16.34%。

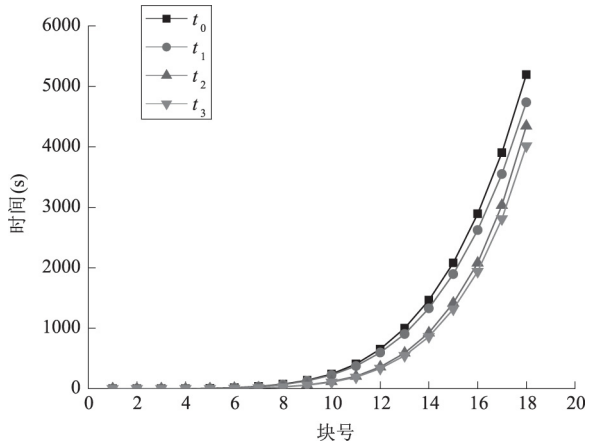

图1 各算法LLL约化过程

表2 不同块号LLL约化实验结果对比

块号	t_0	t_1	t_2	t_3
1	0.00	0.00	0.00	0.00
4	1.51	1.30	0.32	0.30
6	15.44	14.05	5.11	4.69
9	137.09	127.27	61.40	56.09
12	652.94	596.29	361.73	332.72
15	2 080.03	1 894.95	1 415.09	1 316.11
18	5 193.95	4 737.16	4 345.90	4 014.67

4 结束语

本文提出了一种Coppersmith算法的改进算法,通过提取格基矩阵不同块中相应的公因子来提高Coppersmith算法的求解效率。同时本文证明了此改进算法与一种预处理算法的兼容性,结合两种算法进一步提高了Coppersmith算法的求解效率。实验结果表明本文提出的改进Coppersmith算法可有效提高求解单变元模方程的算法效率,并且在维数较高时表现较好。下一步工作将根据Coppersmith矩阵列的性质提高Coppersmith算法的求解效率。

参考文献:

[1] COPPERSMITH D. Finding a small root of a univariate modular equation [C]//International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1996: 155-165.

[2] LENSTRA A K, LENSTRA H W, LOVÁSZ L. Factoring polynomials with rational coefficients [J]. Mathematische Annalen, 1982, 261(4): 515-534.

[3] NGUYEN P Q, STEHLÉ D. An LLL algorithm with quadratic complexity [J]. SIAM Journal on Computing, 2009, 39(3): 874-903.

[4] MAY A. Using LLL-reduction for solving RSA and factorization problems [M]. The LLL algorithm. Springer, Berlin, Heidelberg, 2009: 315-348.

[5] 刘向辉,韩文报,孙杰. 基于离散比特的RSA私钥泄漏攻击 [J]. 信息工程大学学报,2012,13(4):385-388.

[6] HOWGRAVE-Graham N. Finding small roots of univariate modular equations revisited [C]//IMA International Conference on Cryptography and Coding. Springer, Berlin, Heidelberg, 1997: 131-142.

[7] BI J, CORON J S, FAUGÈRE J C, et al. Rounding and chaining LLL: Finding faster small roots of univariate polynomial congruences [C]//International Workshop on Public Key Cryptography. Springer, Berlin, Heidelberg, 2014: 185-202.

[8] MICCIANCIO D, GOLDWASSER S. Complexity of lattice problems: a cryptographic perspective [M]. Springer Science & Business Media, 2012.

[9] BERLEKAMP E R. Factoring polynomials over finite fields [J]. Bell System Technical Journal, 1967, 46(8): 1853-1859.

[10] CANTOR D G, ZASSENHAUS H. A new algorithm for factoring polynomials over finite fields [J]. Mathematics of Computation, 1981, 36(154): 587-592.

[11] WILLIAMS H P. Integer and combinatorial optimization [J]. Journal of the Operational Research Society, 1990, 41(2): 177-178.

[12] COUPE C, NGUYEN P Q, STERN J, et al. the Effectiveness of lattice attacks against low-exponent RSA [C]//Public Key Cryptography, 1999: 204-218.

(编辑:颜峻)