Advances in Applied Mathematics 应用数学进展, 2020, 9(12), 2244-2255
Published Online December 2020 in Hans. http://www.hanspub.org/journal/aam
https://doi.org/10.12677/aam.2020.912262

Hans汉斯

BKW算法求解多元含错方程组

王艺航[1]，梁天元[2]

[1]69224部队，吉林 延边
[2]32124部队，西藏 阿里
Email: liangty0327@163.com

收稿日期：2020年11月21日；录用日期：2020年12月22日；发布日期：2020年12月29日

摘　要

LWE是近年来设计后量子密码和全同态加密算法的热门数学问题。对这些算法的分析，关键在于求解相应的LWE问题。一般的LWE问题求解的困难性甚至高于格上SVP困难问题。本文研究了目前求解LWE问题较为有效的算法——BKW算法。首先详细总结了BKW算法的主要步骤和原理，针对不同参数的LWE问题，分析了BKW约化技术的合理选择，进一步对约减变元数、错误率的变化、方程数量的膨胀、时间复杂度等进行了深入研究。最后通过实验仿真对整个算法进行了实现，对Z/(5)上40个变元、10000个样本，错误率为2%、4%的LWE实例进行了成功求解。

关键词

LWE问题，含错方程组，BKW算法，算法实现

BKW Algorithm Solve Fault-Tolerant Equations

Yihang Wang[1], Tianyuan Liang[2]

[1]69224 Unit, Yanbian Jilin
[2]32124 Unit, Ali Tibet
Email: liangty0327@163.com

Received: Nov. 21st, 2020; accepted: Dec. 22nd, 2020; published: Dec. 29th, 2020

Abstract

LWE is a popular mathematics problem in designing post-design quantum cryptography and full homomorphic encryption algorithms in recent years. The key to the analysis of these algorithms is

文章引用: 王艺航, 梁天元. BKW 算法求解多元含错方程组[J]. 应用数学进展, 2020, 9(12): 2244-2255.
DOI: 10.12677/aam.2020.912262

to solve the corresponding LWE problem. The difficulty of solving the general LWE problem is even higher than that of the SVP problem. This paper studies the BKW algorithm, which is the most effective algorithm for solving LWE problem. Firstly, the main steps and principles of BKW algorithm are summarized in detail. According to the LWE problem with different parameters, the reasonable selection of BKW reduction technology is analyzed, and the reduction of the number of variables, the error rate, the expansion of the number of equations, and the time complexity are further analyzed. Finally, the whole algorithm is implemented by experimental simulation. The LWE example with 40 variables and 10000 samples on Z/(5) and error rate of 2% and 4% is successfully solved.

Keywords

LWE Problem, Fault-Tolerant Equations, BKW Algorithm, Implementation

1. 绪论

1.1. 引言

随着量子计算机的快速发展，传统密码的安全性受到挑战，为了信息的安全，我们需要建立抗量子攻击的加密算法。目前已知的抗量子攻击的算法有很多。基于格的密码算法体制是其中较为热门的一类算法，占所有抗量子算法的50%以上。LWE问题是基于格的密码算法衍生出来的一种算法，其解决难度相当于解决格问题的难度，从数学本质上来说还是格问题，这使得基于LWE问题的密码算法在量子时代依然有着安全性保证。并且基于LWE问题的密码算法在其他密码方向也有蓬勃的生机，比如：LWE目前在同态加密方向有着重要的应用。针对LWE问题的求解也有着很多方法，其中BKW算法[1]是现在较为流行的一种方法。自从BKW算法提出到现在，有很多专家学者对BKW算法进行研究，改进，使得该算法能够更加快速、有效地解决问题。在前人的基础上研究这一算法也能够给我们关于LWE问题更多的启示。

LWE被认为是LPN问题的推广。现已被证明是构造加密算法的非常有用的工具。基于LWE的加密算法目前受到广泛关注有几个原因。一个是构造的简单性，有时会产生非常有效的实现，运行速度比替代解决方案快得多。另一个原因是关于格问题的完善理论，它给出了对LWE问题的难度的深入了解。从最坏情况下的格问题到平均情况下，在理论上减少了LWE问题的难度[2]。第三个原因是，基于LWE的加密是量子计算机尚未能攻破的领域之一，这一特点在后量子时代注定会使得该算法大放异彩。

针对LWE问题已经提出许多不同的方法。在解决LWE的组合算法方面做了很多工作，所有这些都以著名的BKW算法为基础。BKW算法类似于瓦格纳的通用生日方法，最初作为解决LPN问题的算法[3]。这些组合算法的优点是可以以标准的方式分析它们的复杂性，并且可以为LWE问题的不同实例化的复杂性获取显式值。这种方法往往给出了一些重要参数选择的最佳性能的算法。基于BKW算法的一个可能的缺点是，它们通常需要大量的内存，通常与时间复杂度的顺序相同[4]。最近在这个方向的工作目标是通过从可能的口令的字典中快速查找口令来支持基于彩虹的预计算。

1.2. 问题简介

n 是正整数，q 为奇素数，X 是在 Z_q 上服从离散高斯分布的错误分布。$\boldsymbol{s}$ 是 Z_q^n 上的秘密向量，秘密向量服从均匀分布，$L_{s,x}$ 是 $Z_q^n * Z_q$ 上的可能分布，通过随机地在 Z_q^n 上选择 a 和 Z_q 上服从 X 分布的错误 e，最终得到 $Z_q^n * Z_q$ 上的：

$$(a,z)=\left(a,\langle a,s\rangle+e\right) \tag{1-1}$$

LWE 问题就是利用 $L_{s,x}$ 中给出的大量的例子也就是方程，找到秘密向量 $\boldsymbol{s}$ [5]。

LWE 的参数选择通常与某些实际意义有关。素数 q 被选择为 n 中的多项式，而离散高斯分布 X 的平均值为零和标准偏差 $\sigma=\alpha*q$ (α 为较小的数)。例如，在某些示例中，$q\approx n^2,\alpha=1/\left(\sqrt{2}*\pi*n*\log_2 n\right)$。

上述为搜索 LWE 问题的定义。为了便于说明问题，我们重新修改搜索 LWE 问题。假设我们从 LWE 的分布 $L_{s,x}$ 中得到 m 个例子(方程)，表示为

$$(a_1,z_1),(a_2,z_2),\cdots,(a_m,z_m) \tag{1-2}$$

设：

$$\boldsymbol{z}=\left(z_1,z_2,\cdots,z_m\right) \tag{1-3}$$

$$\boldsymbol{y}=\left(y_1,y_2,\cdots,y_m\right)=\boldsymbol{s}*\boldsymbol{A} \tag{1-4}$$

其中

$$\boldsymbol{A}=\left[a_1^{\mathrm{T}},a_2^{\mathrm{T}},\cdots,a_m^{\mathrm{T}}\right] \tag{1-5}$$

可以知道

$$\boldsymbol{z}=\boldsymbol{y}+\boldsymbol{e}=\boldsymbol{s}*\boldsymbol{A}+\boldsymbol{e} \tag{1-6}$$

$$z_i=y_i+e_i=\langle s_i,a_i\rangle+e_i \tag{1-7}$$

e_i 是服从 X 分布的噪声[6]。我们看到这个问题被重新改写为解码问题。矩阵 $\boldsymbol{A}$ 作为线性代码的生成器矩阵，即方程的系数矩阵，它位于 Z_q 上，Z 是接收到的数据。查找数据：

$$\boldsymbol{y}=\boldsymbol{s}*\boldsymbol{A} \tag{1-8}$$

找到满足距离 $\|\boldsymbol{y}-\boldsymbol{z}\|$ 为最小的 $\boldsymbol{y}$ 的值对应的 $\boldsymbol{s}$ 值，即为秘密向量。

在 n 维欧氏空间，向量 $\boldsymbol{x}=(x_1,\cdots,x_n)$，的长度由范数定义：

$$\|\boldsymbol{x}\|=\sqrt{\sum_{i=1}^{n}x_i^2} \tag{1-8}$$

对于两个向量之间的欧氏距离表示为 $\|\boldsymbol{x}-\boldsymbol{y}\|$。对于给定的一组向量 $\boldsymbol{L}$，利用最小二乘法得到与 $\boldsymbol{I}$ 的最小距离，使得 $\|\boldsymbol{x}-\boldsymbol{I}\|$ 最小。

如果秘密向量 s 服从均匀分布，则可以应用一个简单的转换，我们可以通过高斯消除变换把 $\boldsymbol{A}$ 转换成系统形式。假定第一个 n 列是线性独立的，并形成矩阵 $\boldsymbol{A0}$。如果前 n 列非线性独立则可以通过列置换将矩阵前 n 列转换成线性独立。

定义：

$$\boldsymbol{D}=\boldsymbol{A0}^{-1} \tag{1-9}$$

随着变量的变化

$$\boldsymbol{s}'=\boldsymbol{s}*\boldsymbol{A0}-\left(z_1,z_2,\cdots,z_n\right) \tag{1-10}$$

我们得到一个等价的问题，由

$$\boldsymbol{A}' = \left(\boldsymbol{I}, a_{n+1}'^{T}, a_{n+2}'^{T}, \cdots, a_{m}'^{T}\right)，\text{其中 } \boldsymbol{A}' = \boldsymbol{D} * \boldsymbol{A} \tag{1-11}$$

我们计算

$$\boldsymbol{z}' = \boldsymbol{z} - \left(z_1, z_2, \cdots, z_n\right) * \boldsymbol{A}' = \left(\boldsymbol{0}, z'_{n+1}, z'_{n+2}, \cdots, z'_m\right) \tag{1-12}$$

在这个初始步骤之后，秘密向量 $\boldsymbol{s}'$ 中的每一个元素都符合 X 分布。便于后面的求解。

需要注意的是，将 $\boldsymbol{A}$ 做变换成 $\boldsymbol{A}'$ 之后，$\boldsymbol{s}$ 变换成 $\boldsymbol{s}'$，此时我们穷举 $\boldsymbol{s}'$ 的所有可能结果，最终得到的正确结果并不是最终的结果，此时的 $\boldsymbol{s}'$ 符合 X 分布，要得到 $\boldsymbol{s}$ 需要做如下运算：

$$\boldsymbol{s} = \boldsymbol{s}' + \left(z_1, z_2, \cdots, z_n\right) * \boldsymbol{D} \tag{1-13}$$

此时得到的 $\boldsymbol{s}$ 即为最终所求。

2. BKW 算法原理及流程

在这一章节，我们将讲述一下 BKW 算法的具体实现及复杂度。算法流程如下：

LWE 解决算法：

输入：n 行 m 列的矩阵 $\boldsymbol{A}$，长为 m 的向量 $\boldsymbol{z}$，迭代次数 t，l，d

伪代码如下：

Repeat

Step 1：高斯消元将未知秘密向量 $\boldsymbol{s}$ 软换成符合 X 分布的向量

Step 2：进行 t 步 BKW 运算消除后 $t*b$ 个元素

Step 3：对每一个可能的答案进行穷举

Until 找到正确答案。

2.1. 高斯消元

此步骤的目标是将机密向量的分布转换为噪声的分布将矩阵进行变形，使得新的未知变量满足离散高斯分布，便于下一步工作。

下面我们介绍一下离散高斯分布：

让 $x \in Z$。x 服从 $\boldsymbol{Z}$ 上的离散高斯分布，其平均值为 0 和方差 σ^2，表示为 $\boldsymbol{D}_{z,\sigma}$。在 Z_q 上方差为 σ^2 的 $\boldsymbol{D}_{z,\sigma}$ 模 q 得到 X 分布，即在每个剩余类中的所有整数上累积概率函数的值。同样，我们定义了离散高斯在 Z_n 与方差 σ^2，表示 $\boldsymbol{D}\left(z^n, \sigma\right)$，作为 n 个独立的 $\boldsymbol{D}_{z,\sigma}$ 的样本[7]。

一般而言，离散高斯分布并不完全继承连续事例中的通常属性，但在我们考虑的情况下，它将足够接近，我们将使用连续大小写中的属性，因为它们是近似正确的。例如，如果从 $X_{\sigma 1}$ 中抽取 $\boldsymbol{Y}$，从 $X_{\sigma 2}$ 中抽取 $\boldsymbol{X}$，那么我们考虑 $\boldsymbol{X} + \boldsymbol{Y}$ 是服从

$$\boldsymbol{X} + \boldsymbol{Y} \sim \sqrt{\sigma_1^2 + \sigma_2^2} \tag{2-1}$$

消元操作如下：

对矩阵 $\boldsymbol{A}$ 进行初等列变换，变成：$(\boldsymbol{I} \mid \boldsymbol{A0})$。其中 $\boldsymbol{I}$ 是 n 阶单位矩阵，此过程具体如下：

1) 如果矩阵 $\boldsymbol{A}$ 的前 n 列是线性相关的，可以先对行向量做一个随机的置换，使得系数矩阵的前 n 列是线性无关的，这样做只是相当于将秘密向量 $\boldsymbol{s}$ 中的元素做置换，得到结果后还原即可，不会影响结果正确性。这一操作的目的在于找到一个 $n * n$ 的线性独立的仿真，最终能够得到我们要的矩阵格式。得到仿真之后，进行下步操作：

2) 对该矩阵求逆，得到的矩阵记为 d，得到

$$\boldsymbol{d} * \boldsymbol{A} = (\boldsymbol{I} \mid \boldsymbol{A0}) \tag{2-2}$$

这一步骤的复杂度为：

$$C_0 = (m-n')*(n+1)*\frac{n'}{b-1} < m(n+1)*\frac{n'}{b-1} \tag{2-3}$$

其中 $n' = n - tb$。

2.2. BKW 消元

BKW 算法是由百隆等人提出的，最初是针对 LPN 问题。但是，它也被用于 LWE 问题，与瓦格纳的广义生日算法一样，BKW 方法在系数矩阵 $\boldsymbol{A}$ 中的列上使用迭代碰撞过程，该步骤逐步减少 $\boldsymbol{A}$ 的维数。将碰撞在某些位置子集中的列汇总在一起，并将它们作为新矩阵中的列，这样做可以减小矩阵的维数，但会增加噪声[8]。

该算法过程如下：

1) 分组，即将所有的列按照后 b 个元素是否可按下列的策略进行运算进行分组。

首先，在 $\boldsymbol{A}$ 中搜索与某一列的最后 b 项相同的两列的所有组合。假设一个发现两列，则对他们进行消元运算，效果如下

$$a_{i_1} - a_{i_2} = (* * \cdots * 0\, 0 \cdots 0) \quad (0 \text{ 的个数为 } b \text{ 个}) \tag{2-4}$$

$*$意味着任何值。

然后形成一个新的向量：

$$a_i = a_{i_1} - a_{i_2} \tag{2-5}$$

对应于这个向量，形成：

$$z_1^{(2)} = z_{i_1} - z_{i_2} \tag{2-6}$$

如果

$$y_1^{(2)} = \left\langle s, a_1^{(2)} \right\rangle \tag{2-7}$$

那么有

$$z_1^{(2)} = y_1^{(2)} + e_1^{(2)} \tag{2-8}$$

此时

$$e_i = e_{i_1} - e_{i_2} \tag{2-9}$$

又有：e_i 服从高斯分布与方差为 σ，那么

$$e_1^{(2)} = e_{i_1} - e_{i_2} \tag{2-10}$$

同样服从离散高斯分布方差为 $2\sigma^2$。

还有第二种明显的碰撞方法，在向量的选取上有所区别。这种方法是将两列向量后 b 个元素相加为 0 的进行消元，形式与上述雷同，再次不再赘述。

2) 消元，对组内的元素按照分组策略进行消元运算。

其中消元方法有两种，分别为 LF1 和 LF2：

LF1 是在每个集合中，随机取一个向量并与剩余向量消元；LF2 是在集合中对任意两个向量消元，设初始的向量总数为 n，如果方程系数的分布是均匀的，t 轮约化之后，LF1 将会使方程总数减少：

$$m - t * 2^b \tag{2-11}$$

LF2 使方程总数增加到

$$m' = \frac{m^{2^t}}{2^{\left(2^t-1\right)(b+1)}} \tag{2-12}$$

可以发现利用 LF1 能够有效的较少矩阵规模，而 LF2 能够增加矩阵规模，根据已有的数据量规模，可以选用不同的方法[9]。

到现在为止我们完成了 BKW 算法的一步。

然后对 $i = 2,3,\cdots,t$ 进行相同的迭代，选取新的大小冲突集 $q^b - 1$，并查找的冲突列，重复上述操作。

对于每次迭代，噪音都增加了。在 t 步 BKW 步骤后，噪声与每列的噪声相关为：

$$e = \sum_{j=1}^{2^t} e_{i_j} \tag{2-13}$$

总噪声是高斯分布，方差为 $2t\sigma^2$

我们将 $\boldsymbol{A}$ 减小到一个较小的实例，其中现在的秘密向量的长度是 $n' = n - tb$，但噪音有方差 $2t \cdot \sigma^2$。

操作过程如下：

我们首先执行 t 步标准 BKW 步骤来平衡两个噪声部分，即通过合并和编码引入的噪声增加噪声。此步骤完成后，矩阵的每列的后 $t * b$ 位均为零。我们现在给出细节：

已经得到高斯消元之后的结果，即 $\boldsymbol{z}'$和 $\boldsymbol{A}' = (\boldsymbol{IL0})$，我们只处理 $\boldsymbol{L0}$ 部分。类似于其他 BKW 过程，在每个步骤中，我们将向量按最后 b 个未处理的向量进行分组，从而将总样本划分为

$$\frac{q^b - 1}{2} \tag{2-14}$$

类，然后，我们将同一类中的那些合并(添加或减去)到零，形成新的样本作为输入到下一个 BKW 步骤。

消元过程中可根据原有数据的多少选择 LF1 和 LF2 进行挑选，若给定数据量过多，则可使用 LF1 方法进行，会使得数据量有所减少，减轻运算和内存压力，反之，则可使用 LF2，增多数据更容易求出答案。

此步骤的输出是矩阵$(\boldsymbol{I}|\boldsymbol{Lt})$，其中 $\boldsymbol{Lt}$ 的每列中的所有后 $t * b$ 项的都是零。将 $\boldsymbol{Lt}$ 的前 $n - t * b$ 行重新生成一个矩阵，此时我们得到了一系列维度为 $n - t * b$ 的 LWE 样本[10]。此步骤的复杂性是

$$C_1 = \sum_{t=1}^{t1} \left(n + 1 - tb\right)\left(m - \frac{i\left(q^b - 1\right)}{2}\right) \tag{2-15}$$

2.3. 遍历秘密向量

进行上述步骤之后我们得到的时进行消元之后的系数矩阵 $\boldsymbol{A}$，假设我们得到了 k 个方程。此时矩阵的变元数为

$$n' = n - t * b \tag{2-16}$$

此时我们可以手动对结果进行初步的计算，即初步确定算法复杂度：

$$C = q^{(n-b*t)} * \left(n - b * t\right)^k \tag{2-17}$$

3. 章实验仿真

3.1. 参数选取

对与参数的选取，主要涉及到 BKW 算法中的 b 为每次消元个数，t 为迭代次数[11]。

在我们的模拟中选定奇素数 q 为 5，变元个数 n 为 40 [12]，为了能够对最终得到的新矩阵进行运算，我们需要将先矩阵的维度限制在 14 以内[13]，具体原因后文会说明，在最初的模拟时我们为了验证程序正确性，对 1024 个无错方程组进行求解，此时我们令 $b = 10$，$t = 1$，将求得的结果与正确答案相对照，结果很容易得到了正确答案，而后引入错误，错误率达到 1%发现也能得到正确答案。我们将错误率提升到 2%，此时需要的总方程数已经有些略少，我们将初始方程数提升到 10,000 个，并将 b 设置为 5，t 设置为 6，进行 6 轮消元之后剩余 10 个变元，利用穷举得到正确答案。之后其他的条件不变将错误率提升到 5%，发现也能狗解出答案，但是当错误率达到 10%之后，此时的剩余方程总数为 250,000 已经不足以解出问题，之后控制条件我们陆续将剩余方程数调至 500,000、700,000 均未能解出答案，由于时间有限，为继续向下工作。

事实上，如果初始方程数过少时我们可以将 b 的值适当减少，同时增加 t 的值，同时采用 LF2 消元方法，则可以获取足够的方程，如果方程数很多，则可以采用 LF1，降低方程数，也可以才用 LF1 与 LF2 交换使用，同时按照情况选择校园个数 b 的方法进行消元[14]。

3.2. 算法实现

实验环境：win 10

实现平台：visual studio 2017

语言：C++

编程实现：我们知道 C++有丰富的库可以调用，大大简化了我们的工作，而且 NTL 库在有限域上的矩阵运算有很好的包装实现，所以本次我们凭借 NTL 库实现该算法[15]。

1) NTL 库的配置

① NTL 文件编译

具体库可以在 http://www.shoup.net/ntl/上获得，我们采用的是 9.7.0 版本，下载完成之后，在 vs 上新建一个 static 工程，将 NTL 文件夹的 src 子文件夹所有文件添加到工程中，将 NTL 文件夹的 include 子文件夹添加到工程属性 → 链接器 → 常规的附加库目录中(如图 1)。

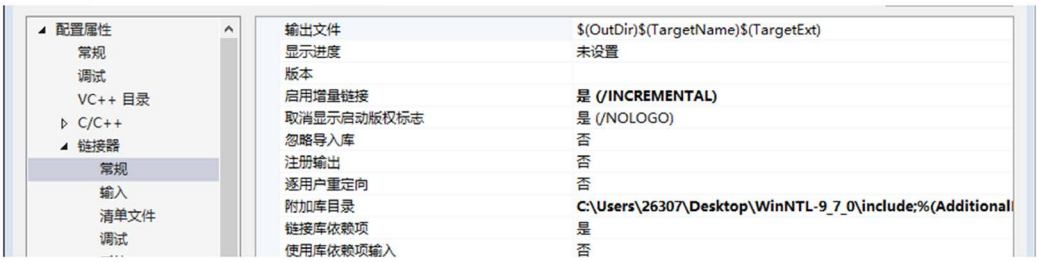

Figure 1. Add the include subfolder to the additional library directory
图 1. 将 include 子文件夹添加到附加库目录中

而后直接编译即可，编译之后可在工程文件加中找到 lib 文件。

② 运行程序

将刚刚的 lib 文件添加到工程属性 → 链接器 → 输入的附加依赖项中(如图 2)。

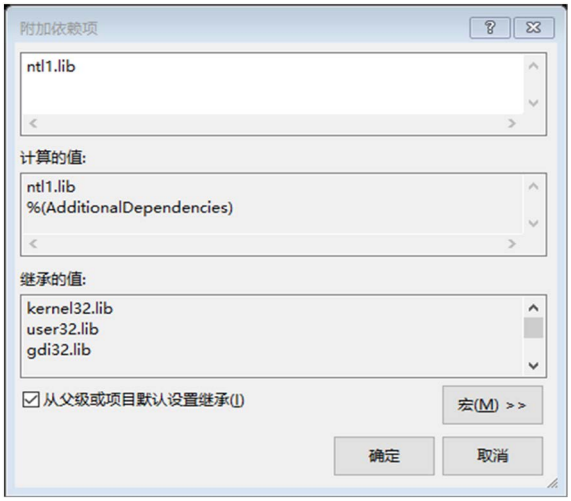

Figure 2. Add the lib file to the additional dependencies
图 2. 将 lib 文件添加到附加依赖项中

NTL 文件夹的 include 子文件夹添加到工程属性 → 链接器 → 常规的附加库目录中(如图 3)。

Figure 3. Add the include subfolder to the regular additional library directory
图 3. 将 include 子文件夹添加到常规的附加库目录中

而后运行 cpp 文件即可。

2) 程序解读

程序依照论文主要有以下几个程序，

① **GA**USS PROESS

输入为：系数矩阵 $\boldsymbol{G}$，含错常数向量 $\boldsymbol{z}$

输出为：$\boldsymbol{G}$ 做变换之后得到的矩阵 $\boldsymbol{L0}$，变换之后的常数向量 $\boldsymbol{z}$；前 n 行现行独立的 $n * n$ 方阵 $\boldsymbol{A0}$ 及他的逆矩阵 d；

关键步骤：

给定的系数矩阵 $\boldsymbol{G}$ 前 n 行可能非线性独立，需要对前 n 行的向量进行初等置换。得到 $\boldsymbol{A0}$ 之后对 $\boldsymbol{A0}$ 进行求逆的 d，最后 $\boldsymbol{G}' = d * \boldsymbol{G}$，得到 $\boldsymbol{G}'$的形式为$(\boldsymbol{I}|\boldsymbol{L0})$，最后得到的 $\boldsymbol{G}$ 为 $\boldsymbol{L0}$

② BKW-reduction

输入为之前得到的 $\boldsymbol{G}$，参数 b，t；

输出为最终的方程数目 counter；

关键步骤：

首先我们需要对刚刚获得矩阵 $\boldsymbol{G}$，按照后 b 个比特进行分类，相同的分成一类，这样的可以分成 q^b 项，对于分类方法我们采用将后 b 个比特的数据按照进制 q 进制转化成 10 进制，最终如果两个数按照数值进

行分类，分成 q^b 个二维数组。对于分类之后的数据我们会根据需要进行消元操作即 LF1 或者 LF2，对于何时采用 LF1 何时采用 LF2，我们在对每个数组进行完操作之后会对目前已有的总方程数进行判断当方程个数大于某一阈值采用 LF1，否则采用 LF2，采用这一方法主要是因为当方程数量过大时，若一直采用 LF2，会使得内存溢出，若一直采用 LF1，会导致得到的方程个数较少，无法得到正确答案。

③ LF1 及 LF2

输入：分组之后的二维向量 $\boldsymbol{V}$，变元个数 n，系统参数 b；

输出：运算之后的二维向量 $\boldsymbol{G}$；

关键步骤：

按照论文所说，LF1 随机选出一个向量与其他向量进行运算，LF2 将二维数组中的每两个向量进行运算。

④ Console

输入：消元后的矩阵 $\boldsymbol{G}$，消元后的常数向量 $\boldsymbol{z}$，变元个数 n，系统参数 b，t，剩余方程总量 $counter$；

输出：每一个可能答案的符合率，及最高符合率对应的向量。

关键步骤：

该子程序主要完成验证性功能，即在完成 BKW 消元后，对剩下的变元 s'进行穷举求解，分别计算对应的 zi 如下图：

$$\begin{bmatrix} a_{11} & \cdots & a_{1(n-bt)} \\ \vdots & \ddots & \vdots \\ a_{counter1} & \cdots & a_{counter(n-b*t)} \end{bmatrix} * \boldsymbol{s'} = \boldsymbol{Zi} = \left(zi_1, zi_2, \cdots zi_{counter}\right) \tag{3-1}$$

如果 $zi_j = z_j$，那么计数器累加，最后得到每个向量对应的

$$\text{符合率} = \frac{\text{最终计数器值}}{\text{总的方程数目}} \tag{3-2}$$

遍历所有的向量，找到符合率最高的项，而后验证是否为正确答案。

3.3. 问题与解决办法

当我们尝试解决 2%的错误率时遇到了些许问题，如果变元数目过多，那么将会导致数据量过大，导致无法遍历所有的位置向量 $\boldsymbol{s'}$ (假定有 20 个变元，则需要遍历 $5^{20} \gg 2^{40}$)同时如果消元数目过多则会导致得到的方程数目太少，因此我们选择的参数为 $b = 5$，$t = 6$。

最终执行的时候我们还是遇到了一些问题，在中间过程中，如果一直使用 LF2 进行消元会导致剩余方程数快速上涨，一般来说 4 轮 BKW 消元之后方程数目已经达到百万级别，导致内存空间不足，运行失败，而且即便运行成功，最后的遍历求解过程也会花费大量时间。

因此我们在消元上进行了一定量的工作，我们建立了一下几种方案：

1) 在分组时对放方程数目进行限制，利用 LF2 进行消元时，如果组内的方程数很多，进行消元之后会增加很多，大约为：

$$m' = \frac{m^{2^t}}{2^{\left(2^t-1\right)(b+1)}} - m \tag{3-3}$$

因此控制组内的方程个数是一个很好的方法，具体操作是当组内的方程个数达到一定数目之后将后来的方程舍弃，不在进行向组内添加方程。这样的方法会将原来方程中的很多有用的信息漏掉，采用需慎重，如果有更好的办法，不宜采用。

2) 第二个方法与上个方法异曲同工，不过是对总方程数进行控制，具体方法是在方程个数达到一定数目时，将舍弃后面为进行操作的组，不在进行消元，转而进行下一轮消元。此方法比第一种舍弃的方程数更为集中，不宜采用，

3) 第三个方法时将两种方法结合起来，在组内控制组内方程个数，在组外控制总的方程个数，经过调试，如果组内方程个数控制的好，可以达到较为理想的结果，但弊端还是同上一样，对信息利用不充分。

4) 第四个方法是每对一组做完组内消元之后判断当前方程数，如果方程数过多，则选取 LF1，会使得方程数减少，如果过少则选取 LF2，会使得方程数增加，但是需要注意的是选取 LF2 消元，方程数会以很快速度增大，LF1 消元，方程数较少速度较慢。所以选取的阈值需要经过测试一般来说，最终剩余的方程数是都会多于阈值，大约为阈值的 5 倍左右。表 1 我们列举了我们有关方程数与阈值的一些实验数据(见表 1)。

Table 1. Experimental data of different thresholds and equation numbers
表 1. 不同阈值与方程数的实验数据

阈值	最终方程数
150,000	683,963
100,000	476,693
50,000	256,464

最终我们决定采用第四中方法与第一种方法相结合的方式来控制方程数，控制组内方程数主要是因为组内方程过多，使用 LF2 增长速度过快，不容控制，将组能方程数控制在可控范围内，是必要的。

关于确定方程数数目的问题：

事实上，由于方程是含错的，由于错误率不同，我们为了得到正确的答案所需要的方程数也不同，为了得到正确的向量，我们应该在允许范围内，尽量得到更多的方程，同时，由与如果方程数目过多，我们所需要的运算量会大大增加，那么我们可能没有足够的能力处理这些方程，此时我们选择减少最终的方程数量，那么我们得到的符合率最高的向量就可能不是正确答案，那么我们就应该确定一个范围，使得符合率在这个范围内的向量就可以待定为正确答案，进行下一步确认，如果能够达到这样的目的，我们可以认为此时的方程数量是恰当的。

3.4. 实验结果

为了验证程序的正确性，我们选取不同的参数做了以下的实验，结果如下(见表 2)。

Table 2. Experimental results under different parameters
表 2. 不同参数下的实验结果

变元个数	方程个数	错误率	每次消元数	消元轮数	剩余方程数	最高符合数	符合率	是否正确
40	1024	0%	10	1	10,000	10,000	1	正确
40	1024	1%	10	1	10,000	10,000	1	正确
40	10,000	2%	5	6	25,000	7227	0.288	正确
40	10,000	5%	5	6	25,000	5202	0.208	正确
40	10,000	10%	5	6	25,000	4800	0.192	错误
40	10,000	10%	5	6	50,000	9800	0.196	错误

结果分析：

首先我们可以看到一个显而易见的结论，错误率越少，得到结果所需方程数越少，越容易得到正确答案，但是还有一些内容，单凭表格上的内容我们得不到，因此，我们将仅选取第 3，4 组的符合率分布做出了图表(此图表仅显示了符合率大于 0.2 的备选答案，见图 4)，可以较为直观的看到当错误率较小时，非正确答案分布满足基本在 0.2 左右，正确答案为 0.28，此时剩余方程数能够去别处正确答案，当错误率达到 5%时，非正确答案依然分布在 0.2 周围但是已经有部分备选答案略大于 0.2，此时虽然能够得到正确答案，但是正确答案的优势已经不那么明显了，而当符合率达到 10%时，可以从表中看出，上述的方程数已经无法去别出正确答案了，需要增加方程数。

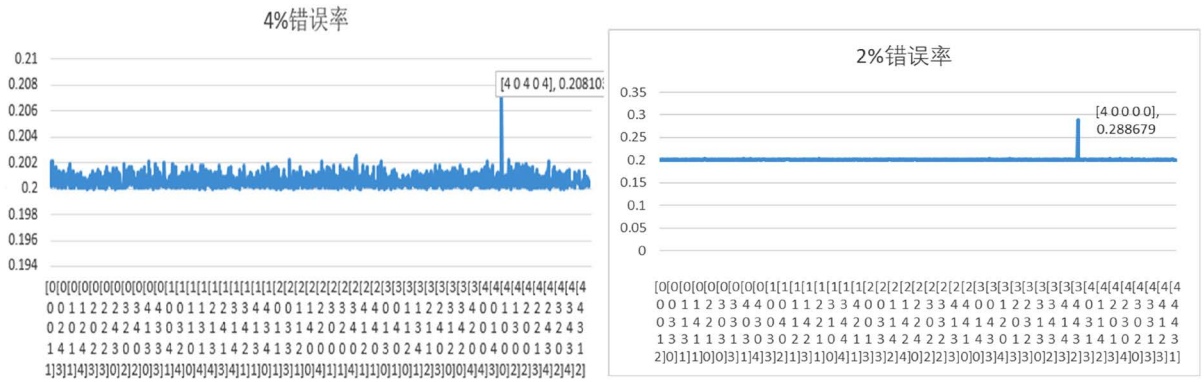

Figure 4. Experimental results under 4% error rate and experimental results under 2% error rate
图 4. 4%错误率下的实验结果和 2%错误率下的实验结果

4. 总结与改进

4.1. 结论

在本次论文中，我们学到了很多新的内容，在研究阶段我们接触到的关于解线性方程问题只有高斯消元法，此次研究，我们主要完成了以下几个问题：

1) 完成了对 LWE 问题的理论学习，由于这类问题国内还没有成熟的论文，所以相关文献大都为英文文献。虽然之前接触过英文文献，但在最初学习的时候依然很不适应，对文章的叙述方式及部分专业用语不熟悉，但是学习了一段时间之后，渐渐能够适应英文，相信会为以后的工作学习打下一个良好的基础。

2) 完成大数据处理，含错方程组的规模有 10,000 个，并且在经过多轮 BKW 消元之后，如果不加控制，方程数目甚至会达到上百万，最终导致内存溢出，这是需要我们自己多加思考，如何有效地控制方程规模，好在 BKW 算法有两种消元策略，一种会使得方程数增加，一种会使得方程数较少，通过判断当前方程数，判断采用哪种策略，有效地控制住方程数目。

3) 利用 NTL 库简化编程复杂度，NTL 库能够有效地对有限域下的矩阵运算，对矩阵求逆，矩阵乘法加法有封装好的函数。这次研究使得我们又认识到了一个新的很强大的库，这也是一个非常巨大的收获。

4.2. 未来的工作

由于时间较为紧张，原本计划的内容未能全部实现，在以后的工作中将在以下几个方面进行完善：

1) 减小初始方程数目，现在初始方程数目为 10000，所需要的方程数目有些多，我们要在下一步的

工作中减少初始方程数。

2) 目前的仿真数据较少，在接下来的工作中，将进一步确定程序的效率，检查程序的漏洞，争取将这个程序加以完善优化算法效率，由于时间问题，编写的程序仅仅能解决问题，但效率有些低，在以后的工作中，会进一步检查方程优化程序。。

3) 改进解决方案，之前的方案是在最终求解 s’采取穷举的方法，效率十分低级，在接下来的工作中，将继续学习穷举多项式方向知识，初步规划是学习傅里叶变换和线性码结合解决问题。

参考文献

[1] Albrecht, M.R., Cid, C., Faugere, J.C., Fitzpatrick, R. and Perret, L. (2013) On the Complexity of the BKW Algorithm on LWE. *Designs, Codes and Cryptography*, **74**, 26. https://doi.org/10.1007/s10623-013-9864-x

[2] Albrecht, M.R., Farshim, P., Faugere, J.-C. and Perret, L. (2011) Polly Cracker, Revisited. In: Lee, D.H. and Wang, X., Eds., *Asiacrypt* 2011, LNCS, Vol. 7073, Springer, Heidelberg, 179-196. https://doi.org/10.1007/978-3-642-25385-0_10

[3] Albrecht, M.R., Faugere, J.-C., Fitzpatrick, R. and Perret, L. (2014) Lazy Modulus Switching for the BKW Algorithm on LWE. In: Krawczyk, H., Ed., *PKC* 2014, LNCS, Vol. 8383, Springer, Heidelberg, 429-445. https://doi.org/10.1007/978-3-642-54631-0_25

[4] Applebaum, B., Cash, D., Peikert, C. and Sahai, A. (2009) Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In: Halevi, S., Ed., *CRYPTO* 2009, LNCS, Vol. 5677, Springer, Heidelberg, 595-618. https://doi.org/10.1007/978-3-642-03356-8_35

[5] Arora, S. and Ge, R. (2011) New Algorithms for Learning in Presence of Errors. In: Aceto, L., Henzinger, M. and Sgall, J., Eds., *ICALP* 2011, Part I, LNCS, Vol. 6755, Springer, Heidelberg, 403-415. https://doi.org/10.1007/978-3-642-22006-7_34

[6] Baigneres, T., Junod, P. and Vaudenay, S. (2004) How Far Can We Go beyond Linear Crypt-Analysis? In: Lee, P.J., Ed., *ASIACRYPT* 2004, LNCS, Vol. 3329, Springer, Heidelberg, 432-450. https://doi.org/10.1007/978-3-540-30539-2_31

[7] Blum, A., Kalai, A. and Wasserman, H. (2003) Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model. *Journal of the ACM*, **50**, 506-519. https://doi.org/10.1145/792538.792543

[8] Brakerski, Z. (2012) Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Safa-vi-Naini, R. and Canetti, R., Eds., *CRYPTO* 2012, LNCS, Vol. 7417, Springer, Heidelberg, 868-886. https://doi.org/10.1007/978-3-642-32009-5_50

[9] Brakerski, Z., Langlois, A., Peikert, C., Regev, O. and Stehle, D. (2013) Classical Hardness of Learning with Errors. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, ACM, New York, 575-584. https://doi.org/10.1145/2488608.2488680

[10] Brakerski, Z. and Vaikuntanathan, V. (2011) Efficient Fully Homomorphic Encryption from (Standard) LWE. *Proceedings of the* 2011 *IEEE* 52*nd Annual Symposium on Foundations of Computer Science*, Palm Springs, 22-25 October 2011, 97-106. https://doi.org/10.1109/FOCS.2011.12

[11] Chen, Y. and Nguyen, P.Q. (2011) BKZ 2.0: Better Lattice Security Estimates. In: Lee, D.H. and Wang, X., Eds., *ASIACRYPT* 2011, LNCS, Vol. 7073, Springer, Heidelberg, 1-20. https://doi.org/10.1007/978-3-642-25385-0_1

[12] Cohen, G., Honkala, I., Litsyn, S. and Lobstein, A. (1997) Covering Codes, Vol. 54. Elsevier, Amsterdam.

[13] Conway, J. and Sloane, N. (1982) Voronoi Regions of Lattices, Second Moments of Polytopes, and Quantization. *IEEE Transactions on Information Theory*, **28**, 211-226. https://doi.org/10.1109/TIT.1982.1056483

[14] Conway, J.H., Sloane, N.J.A., Bannai, E., Leech, J., Norton, S., Odlyzko, A., Parker, R., Queen, L. and Venkov, B. (1993) Sphere Packings, Lattices and Groups, Vol. 3. Springer, New York.

[15] Cover, T.M. and Thomas, J.A. (2012) Elements of Information Theory. Wiley, New York.