桂林電子科技大學
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

本科毕业设计（论文）

题目： 关于隐私保护分布式统计的算法研究

学　　号：	1800710238
姓　　名：	王智坚
学　　院：	数学与计算科学
专　　业：	信息与计算科学
指 导 教 师：	张必山
指导教师职称：	副教授

2022 年 5 月 26 日

摘　要

在分布式统计计算中，若自身的设备无法处理庞大的计算量，可交由第三方服务计算商代为处理.若数据为隐私、机密数据，直接发送可能会造成数据的泄露，引发不可估量的后果.所以将数据加密后，第三方服务计算商对加密后的密文进行运算，接着将结果返回这样保证了数据的隐私性、正确性和机密性.

本文通过 BFV 加密方案对数据进行加密，但其只能对整数进行加密，为了适应统计计算的特点，引入 CKKS 方案的编码方法，将明文空间扩大至浮点数.同时为了达到分布式的特点，将 BFV 的私钥作为联合私钥，之后将其分解，分发给各计算参与方，形成多用户多密钥的模型.接着对加密方案的正确性，安全性和累加局限性进行分析，最后通过仿真实验，计算统计中的求和时间代价.得出了一下结论.

1、多密钥的设定使得攻击者在获取部分私钥时，仍无法解密，之后获取全部的私钥，才能将数据解密.这使得数据得到了安全保证.

2、在累加局限性分析中，随机噪声的添加使得干扰因素增加，保证了加密的安全，但其不断累加的系数也使得操作解密的缩放值，致使解密失败，不能进行任意次同态加法操作.

3、仿真实验中，各环节时间代价都趋于 10ms，结果良好.

关键词：同态加密；安全多方计算；半诚实敌手模型；全同态加密

Abstract

In distributed statistical computing, if your own equipment cannot handle the huge amount of computation, it can be handled by a third-party service computing provider. If the data is private and confidential, sending it directly may cause data leakage and cause immeasurable damage. Consequences. Therefore, after encrypting the data, the third-party service computing provider operates on the encrypted ciphertext, and then returns the result, which ensures the privacy, correctness and confidentiality of the data.

In this paper, data is encrypted by BFV encryption scheme, but it can only encrypt integers. In order to adapt to the characteristics of statistical calculation, the coding method of CKKS scheme is introduced to expand the plaintext space to floating point numbers. At the same time, in order to achieve distributed characteristics, the private key of BFV is used as a joint private key, and then it is decomposed and distributed to each computing participant to form a multi-user multi-key model. Then the correctness, security and accumulation limitation of the encryption scheme are analyzed. Finally, the summation time cost in statistics is calculated through simulation experiments. came to a conclusion.

1. The setting of multiple keys makes it impossible for the attacker to decrypt the data when he obtains some private keys, and then obtains all the private keys to decrypt the data. This makes the data secure.

2. In the accumulation limitation analysis, the addition of random noise increases the interference factors and ensures the security of encryption, but the continuously accumulated coefficients also make the scaling value of decryption operate, resulting in decryption failure and the inability to perform any number of homomorphic addition operations. .

3. In the simulation experiment, the time cost of each link tends to be 10ms, and the result is good.

Key words: Homomorphic encryption; Secure multi-party computation; Secure computing protocol; Fully homomorphic encryption

目　　录

1 绪论

1.1 研究背景及意义

信息大爆炸的今天，海量的数据扑面而来.面对这些激增的数据，如果自己的设备无法处理，那么就可以交由有能力的第三方代行计算，这时就涉及到了数据的隐私与机密的问题.例如，一款软件若要在本地处理数据，若用户设备性能低，在本地处理会大大消耗设备资源，造成使用卡顿，用户体验性极差.而将这些数据提交到云服务器上去计算，则能大大解决这一问题.这时迎来了有一个问题，若将用户的隐私数据直接提交给云服务器，就会造成数据的泄露，这就是大数据下的隐私问题.而将数据进行加密，再安全传输协议发送至云服务器，由云服务器代为处理，这也许是一个好的办法.

1.2 研究现状

针对密文计算问题，如果能将数据进行加密，并且对加密后的数据执行同样的加减乘除操作，若是此时将结果进行解密仍能得到相同的结果，那么这就叫做同态加密，一种先计算后解密等同于先解密后计算的方案——同态加密，是由 Rivest 等人[1]在上世纪七十年代提出的.这使得数据能由多方提供给云服务器进行密文计算，对数据的机密性，完整性和隐私性得到了有效的保护.同态加密方案支持对密文的计算，并能由密文的计算结果解密得到正确的明文计算结果.在后来的同态加密方案中，有些只支持加法同态，如 Paillier 方案[2]和 Goldwasser-Micali 方案[3]；有些只支持乘法同态，如 Unpadded-RSA 方案[4]和 ElGamal 方案[5].而能同时支持加法和乘法运算的加密方案则较少.Boneh[6]等人提出了 Boneh-Goh-Nissim 加密方案，能够支持任意次的加法操作，但只能支持一次乘法操作.Centry[7]提出了一种基于理想格的加密方案，并利用提提出的“Bootstrappable”技术，通过对密文的重加密使密文支持任意次数的同态运算，是真正意义上的全同态加密，也称为全同态加密的研究奠定重要基础.后续的很多同态加密方案[8~14]也都基于 Centry 的“Bootsrappable”技术，这些方案对文献[7]都有不同程度的改进.

对于同态加法的安全性来说，满足选择明文不可区分(IND-CPA)安全是十分重要的.上述的 ElGamal 和作为应用最广的 Pailler[15]加密系统满足此安全性，但 EliGamal 的密文会成倍扩张，这是它的不足之处.Goldwasser-Micali[16]在二次剩余困难问题上也同样满足此安全，但作为异或同态加密系统，每次操作只能为单比特，由此加密效率大打折扣.渐渐的，后续在安全的基础上出现了许多的浅同态加密方案，作为代表的 Boneh 在理想成员判定困难假设下设计的加密方案，能执行若干次加法运算和解决 2NF 问题，但只能进行一次乘法运算，且解密时需要搜索解密，由此基于 2NF 的计算协议效率可以说是很低.时间往后接着推移，身处 IBM 的研究员 Craig Gentry[17]发表一篇论文于

STOC，他解决了一项棘手的数学问题，该问题自几十年前公钥加密发明以来，一致困扰着科学家们.他基于“理想格 idea lattice”的数学对象，使人们可以充分操作加密状态的数据，即可以在不解密的情况下对加密数据进行任何可以在明文上的计算，对加密后的信息仍能深入和无限的分析，保证数据的隐私性、完整性和保密性.这个加密技术被称为全同态加密(full homomorphic encryption).

目前对密文计算在分布式统计方面研究较少，马飞，蒋建国 [18]在分布式计算的基础上，运用 Pailier 同态加密提出了在 SMH 模式下，采用 CClient，KClient，SServer 的拓扑结构进行统计分析的计算方案.本文想运用 BFV 全同态加密算法在分布式统计情况下，研究其算法的正确性，安全性和局限性.

2 理论与本文工作

2.1 数据通过何种方式加密.

BFV 算法加密方案来源于文章 "Somewhat Practical Fully Homomorphic Encryption"，它是基于 RLWE (Ring-Learning With Errors)难题的全同态加密方案.密文形式为两个模为$mod\ d$的多项式，依据多项式相同幂级项相加的特性，可应用于分布式系统，交由多个计算参与方计算，且计算参与方之间互不干扰.

2.2 本文结构

第三章部分本文的相关理论基础.第二部分分析计算协议对数据的安全.第三部分对方案正确性，安全性，复杂性以及局限性进行分析.

对于本文工作.介绍了在分布式情况下，基于半诚实模型，数据所有者于计算参与方之间时如何完成委托计算的过程，以及数据所有者在保护数据隐私的情况下，如何对数据进行加密，以至于保证数据不被泄露的 BFV 全同态加密算法.最后模拟在多个计算参与方进行多次同态加法之后，对算法进行正确性，安全性，局限性分析.

3　理论基础

3.1 背景知识

同态加密是一种加密形式，具有额外的评估能力，可以在不访问密钥的情况下计算加密数据.这种计算的结果仍然是加密的.同态加密可以看作是公钥密码学的扩展.同态是指代数中的同态，加密和解密函数可以被认为是明文空间 M 和密文空间 L 之间的同态.同态加密的方案中，有密钥生成算法 Gen_ζ，加密算法 Enc_ζ，解密算法 Dec_ζ

1）密钥生成算法$Gen_\zeta: M \to b_{key}$.Gen_ζ根据明文空间 M 的随机数生成密钥b_{key}.

2）加密算法$Enc_\zeta: (b_{key}, p_\zeta) \to c_\zeta$.$p_\zeta$表示明文空间，$c_\zeta$表示密文空间，$Enc_\zeta$利用密钥$b_{key}$加密明文$p_\zeta$，返回密文$c_\zeta$.

3）解密算法$Dec_\zeta: (b_{key}, c_\zeta) \to p_\zeta$.$Dec_\zeta$利用密钥$b_{key}$解密密文$c_\zeta$，返回明文$p_\zeta$.

对于明文空间 M 上的多项式

$$f(a_0, a_1, a_2, a_3, \cdots, a_n) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n, a_n (n = 1, 2, 3, \cdots, n) \subset M, \quad (3\text{-}1)$$

以及数据 $p_1, p_2, p_3, \cdots, p_n$，使用密钥生成算法 Gen_ζ，数据 $p_1, p_2, p_3, \cdots, p_n$ 经过加密算法 Enc_ζ 加密后，得到密文 $c_1, c_2, c_3, \cdots, c_n$.这时候第三方对密文 $c_1, c_2, c_3, \cdots, c_n$ 进行计算处理，得到加密后的结果 e，第三方将结果 e 返回给数据拥有者 F，数据拥有者 F 通过私钥 g，结合解密算法 Dec_ζ 就可以将结果还原为明文形式，即

$$p + q = Dec(Enc(b, p) + Enc(b, q)). \quad (3\text{-}2)$$

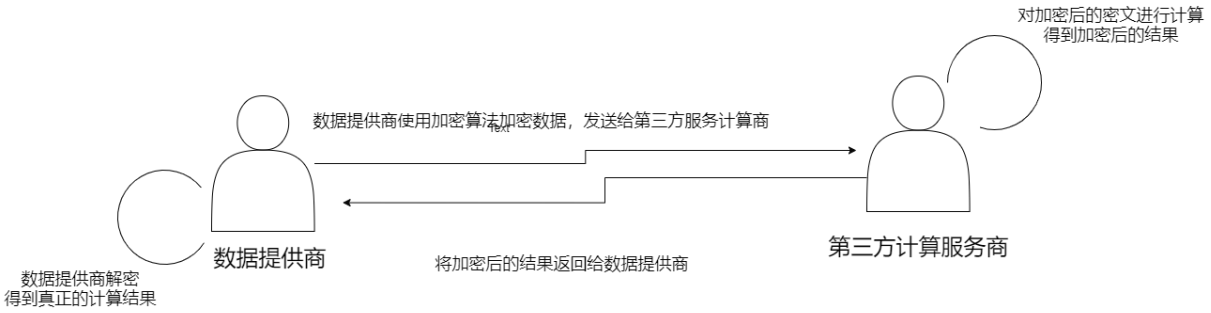

图 3.1 加密解密操作流程图

加密后进行加法运算得到的结果解密后与明文运算的一致，称为加法同态加密.同理满足减法、乘法的称为减法同态加密，乘法同态加密，而能够对密文进行任意计算的称为全同态加密.

3.2 加密算法原理

BFV[19]加密是主流的基于容错学习问题的全同态加密方案之一，其主要思想是将明文放在密文的高位，通过整除的方式，去除低位的噪声.比如有一个 BFV 的密文 ct，

私钥s，则 BFV 的解密为$m+e=c\times s$，其中e是解密噪声，显然，当噪声足够小的时候，我们可以通过除以t取整获得明文m.

3.2.1背景介绍

对于一个简单整系数多项式

$$4x^2+2x+1. \tag{3-3}$$

对系数取模$\bmod t$.假设$t=12$，如 9 加 6 得到 3，这就好比一个 12 月份的日历，经过一年，但观察其月份仍在 12 月之中.之后多项式中的所有系数都是这样处理的.

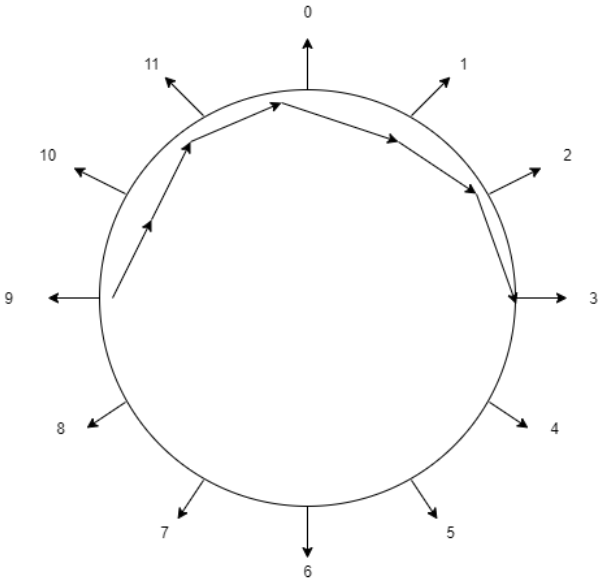

图 3.2 $\bmod t=12$的多项式环

由于计算机取模的特性，一个负数-3 对$t=12$取模，得到的是 9，若我们想拥有负数的系数空间，那么可以自行定义其取模方法，将数字处于-5 到 6 之间，那么我们就可以对负数进行求解.但是这只是一个方便系数，对于余数-1 和余数 11 之间是没有区别的.同时，若两个多项式相乘，那么多项式的最高次的次数将会增加，若进行多次相乘，则多项式的规模想不断增大，对此，余数的思想也同样可以使用到多项式本身.

这时，由 BFV 加密方案，定义一个多项式模，它是由一个特殊多项式组成，如同多项式的系数模一般，次数为 4 的多项式与次数为 14 的多项式相乘，对次数为 16 的多项式取模，得到了一个次数为 2 的多项式.而定义的多项式模的形式为x^d+1，那么对于上述的多项式模可知取$d=16$，并且规定了$d=2^n$，因此多项式为$x^{16}+1$.

接下来讨论对于多项式模$x^{16}+1$之后得到的余数，若多项式存在幂为 16 或更高次的幂，那么将其对多项式模取模之后都会消去，那么我们讨论的幂也就只有从x^0到x^{15}的多项式.运用数学公式可以表示为$x^{16}\equiv -1(\bmod\ x^{16}+1)$，这说明将多项式的幂归约到了幂$d\in[0,16)$之间，而$x^{16}$可以被$-1$替换.

所以接下来考虑的多项式都是这种形式的

$$a_{15}x^{15}+a_{14}x^{14}+a_{13}x^{13}+a_{12}x^{12}+a_{11}x^{11}+a_{10}x^{10}+a_9x^9+a_8x^8 \\ +a_7x^7+a_6x^6+a_5x^5+a_4x^4+a_3x^3+a_2x^2+a_1x+a_0. \tag{3-4}$$

其中这 16 系数（即 a_i ）中的每一个范围都是从 0 到 $d-1$，我们可以用系数的环面来说明，如下所示

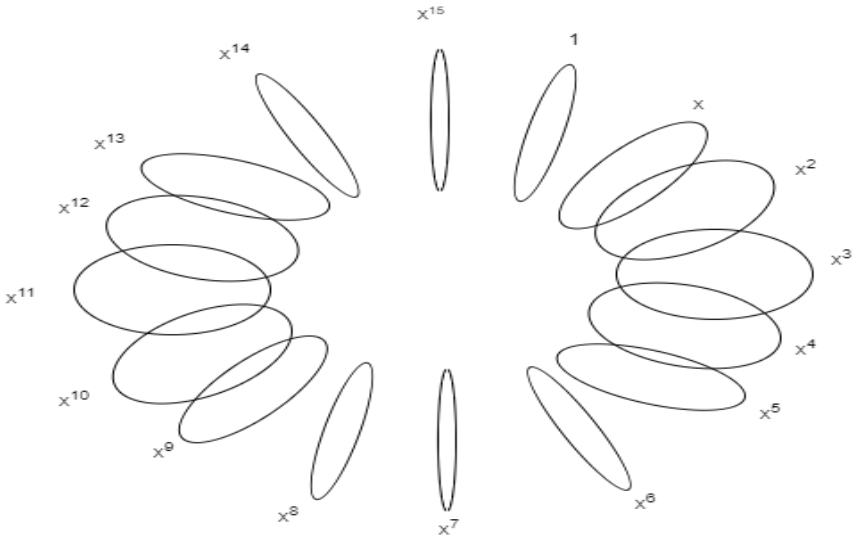

图 3.3 mod $d=16$ 系数环面

当我们执行加密方案时，涉及到多项式的乘法计算，会将两个 x 的幂次相乘，例如将 $2x^{14}$ 与 x^4 相乘会得到结果 $2x^{18}$.由 BFV 加密方案规定，需要将得到的多项式对多项式模进行取模，若多项式模定义为 x^{16}，那么余数只需要在系数环面上从 x^{14} 转过 x^0 直到 x^2 即可.但是根据 BFV 加密方案，定义的多项式模 $x^{16}+1$——如上所示，常数项 1 使得余数的系数发生了变化，这有助于进一步干扰乘法得到的结果.

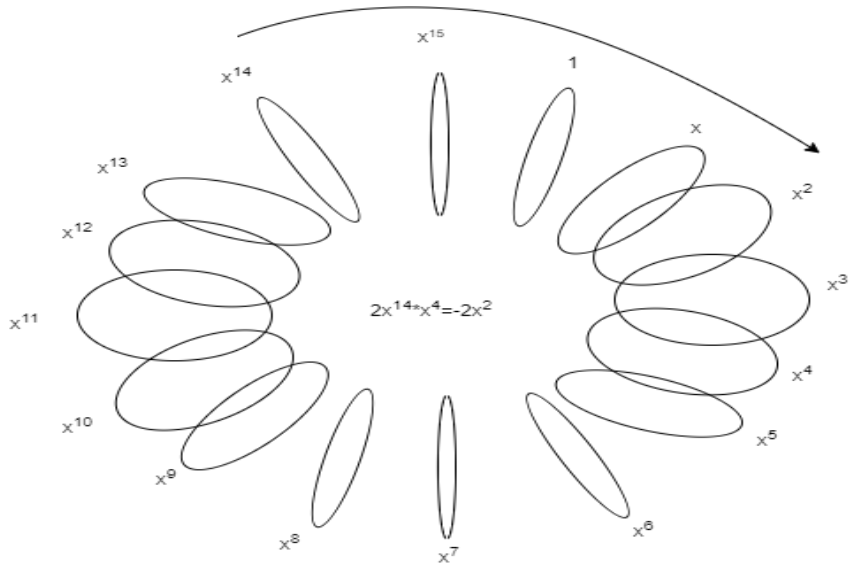

图 3.4 环面变化

由图 3.4 可知，$2x^{14}$ 因其幂次加上 4，环面由此向前旋转四个到达 x^2，此时我们得到结果 $-2x^2$，又因为系数模 $t=24$，则最后的结果为 $22x^2$（若我们想得到存在负数的结果，那么取值范围可以是-12 到 11，也就是说 $-2x^2$ 也是成立的）.

3.2.2使用环上的多项式加密

由前文所述，我们了解了 BFV 加密方案中的多项式系数环面的定义，接下来了解加密方案中的加密与解密的原理.由密码学基础知识可知，加密解密需要一对密钥（公钥和私钥），那么我们先了解其是如何生成私钥与公钥的，接着能更好的理解其加密与解密过程.

(1)私钥和公钥

若将明文进行加密使其转换为密文，那么需要生成一个私钥，接着利用私钥派生出一个公钥，最后使用公钥将明文进行加密.

对于明文空间的多项式，其经过多项式模 x^d+1 的取模，且其系数在系数模 t 之中.明文通过私钥生成的公钥加密之后，得到两个环上的多项式，且具有相同的多项式模，得到的两密文多项式与明文不相同的是，其系数模为 q，且 q 远大于 t.

如果我们假设 $d=4096$，无论是明文还是密文，那么就会存在 $d=4096$ 个系数，为了计算的安全，需要系数模 t 取得大，令 $t=290764801$，同时，密文多项式的系数就需要比系数模 t 还要大，如 $q=9214347247561474048$ 或更大.

为了表述直观但又保证一定的安全性，我们使 $d=20$、$t=20480$ 和 $q=26214400$.

对于私钥，它是由系数为-1、0 和 1 组成的多项式，接下来我们随机生成一个私钥，用 s 表示.

$$s=-x^{18}-x^{17}+x^{16}+x^{14}+x^{13}-x^{12}-x^{11}-x^{10}+x^9-x^7+x^6-x^5-x^4+x^3-x-1. \tag{3-5}$$

为了接下来生成公钥，还需生成一个随机多项式，但其系数模需仍为 q，生成的多项式用 a 表示.

$$\begin{aligned} a=&6232991x^{19}+4468759x^{18}+8071583x^{17}-9205735x^{16}+2480145x^{15}\\ &+7880745x^{14}+664117x^{13}-10042664x^{12}+700992x^{11}+6068675x^{10}\\ &+1080889x^9+4403536x^8-11676926x^7-5038114x^6-6847784x^5\\ &+389783x^4+8303932x^3+7824347x^2-10936607x+8220445. \end{aligned} \tag{3-6}$$

此外，为了加密的安全，引入了一个“小”的噪声多项式来扰乱其密文系数分布.噪声多项式的系数将从离散的高斯分布中提取，但是此多项式仅使用一次，不必保存.

$$\begin{aligned} e=&5x^{18}+x^{17}+x^{16}+4x^{15}+3x^{14}-x^{13}-4x^{12}-x^{11}-3x^{10}\\ &-4x^9+2x^8+3x^7+2x^6+3x^5-2x^2+2x-3. \end{aligned} \tag{3-7}$$

由 BFV 加密方案的规定，生成的公钥 $pk=([-as+e]_q,a)$ 由两个多项式组成，且具有多项式模和系数模 q 的.

对于上面给出的示例，公钥的第一个多项式被构造为

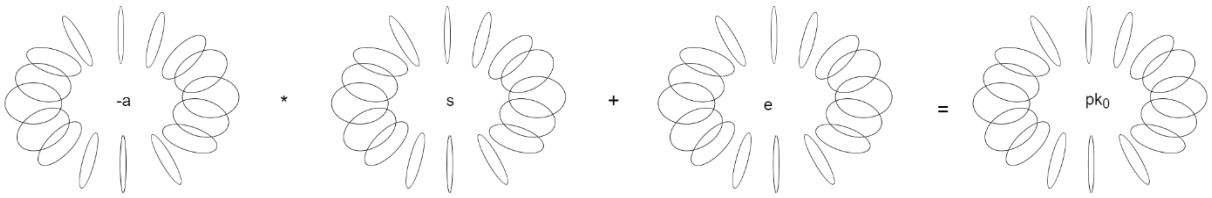

图 3.5 公钥生成

$$\begin{aligned} pk_0 = {} & 7770124x^{19} + 11873819x^{18} - 10547374x^{17} - 4160027x^{16} + 7970426x^{15} \\ & + 4383494x^{14} - 11926778x^{13} - 5351847x^{12} + 8805182x^{11} - 12993620x^{10} \\ & + 6085770x^{9} + 12168500x^{8} - 2814833x^{7} + 12795407x^{6} - 10238010x^{5} \\ & - 2547658x^{4} + 12185838x^{3} - 10264067x^{2} + 9206842x + 691008. \end{aligned} \tag{3-8}$$

由上图可知，将随机多项式a取负号，接着与系数为-1，0 和 1 的私钥s相乘，同时执行加密方案的取模操作，接着再与噪声多项式e相加，同理，一样需要取模.而取模操作有效的打乱了a的所有系数，由由于噪声多项式的加入，使得公钥的中的私钥得到有效的隐藏，保护.

对于公钥中噪声多项式e的加入，让$-as$的系数得到有效的保护，此时若从公钥中找的私钥的可能性将很难.若选择去掉噪声多项式，就会很容易计算出私钥，而噪声多项式系数的大小又是一个难题，太大使得通过私钥解密出错误的结果，太小不能保护，所以噪声多项式的系数选择是十分关键的.

(2)加密

将明文加密为密文，这个过程如同生成公钥一般，将一个系数模为t的多项式通过引入噪声多项式扰乱系数的分布，最后得出一对系数模为q的多项式.接下来，我们尝试使用一个简单的多项式作为明文（称为消息）——$m = 4x^2 + 2x + 1.$

同样的，这次我们引入三个“小”的多项式，两个噪声多项式的系数都取自离散高斯分布，另一个多项式同私钥一般，系数只有-1、0 和 1，我们将其称为u.

$$\begin{aligned} e_1 = {} & -3x^{19} + 3x^{18} + 2x^{16} - 2x^{15} - 3x^{14} + 2x^{13} - 3x^{12} - 4x^{11} + 5x^{10} \\ & - x^{9} - 2x^{8} + x^{7} + 4x^{6} - 4x^{5} - x^{4} + 3x^{3} - 3x^{2} + 2x - 3. \end{aligned} \tag{3-9}$$

$$\begin{aligned} e_2 = {} & -x^{19} + 3x^{18} + x^{17} + 3x^{16} - x^{15} - 4x^{14} + x^{12} + x^{11} - x^{10} \\ & - 3x^{9} + 2x^{8} - 3x^{7} - 3x^{6} + 4x^{5} - 3x^{4} + x^{3} + 4x^{2} + 3x - 2. \end{aligned} \tag{3-10}$$

和

$$\begin{aligned} u = {} & x^{19} + x^{18} + x^{17} - x^{16} - x^{14} + x^{13} - x^{12} - x^{10} \\ & + x^{9} + x^{8} - x^{7} + x^{6} - x^{5} + x^{4} - x^{3} + x^{2} + x + 1. \end{aligned} \tag{3-11}$$

这些多项式只在加密过程中使用，然后丢弃.

密文是由两个多项式组成的，通过如下计算得到

$$ct = ([pk_0u + e_1 + \frac{qm}{t}]_q, [pk_1u + e_2]_q). \tag{3-12}$$

通过观察密文的第一个多项式可以发现，上述给定的消息 m 其系数模为 t，而在密文的计算中，将其缩放为了 q/t (即 1280)，让其处于 mod q 的范围之中，对于后续的同态加法，同态减法能保持一致性，同时又能将消息进行掩盖. u 与公钥的第一个多项式相乘，而 u 的随机性又增加了密文的安全性，使得每此加密后都能产生不同的密文.

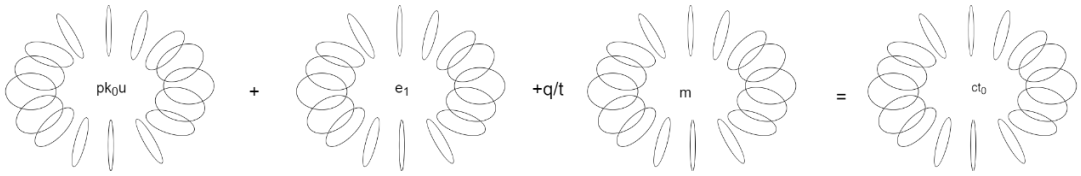

图 3.6 通过公钥加密明文为密文

由于消息只与系数模 q 和 t 相乘，以比例的形式存在于密文之中，噪声多项式掩盖了其系数分布，使得同态加法与同态乘法得以实现.

使用上面给出的多项式显式地计算密文的第一个元素

$$\begin{aligned} ct_0 = &-6704816x^{19} - 3387669x^{18} - 1263755x^{17} - 7566503x^{16} - 5553801x^{15} \\ &+ 10480374x^{14} - 2983465x^{13} + 8685054x^{12} - 8427950x^{11} - 12902530x^{10} \\ &+ 11941655x^9 - 5799749x^8 - 7949675x^7 + 12500833x^6 - 1304873x^5 \\ &- 5505880x^4 - 10725454x^3 + 12010517x^2 + 4880686x - 3589928. \end{aligned} \tag{3-13}$$

将密文中的公钥展开，我们可以看到密文的第一个元素为

$$ct_0 = (e_1 + eu - aus + \frac{qm}{t})_q. \tag{3-14}$$

由公式(3-14)可知， aus 是“大”的多项式，能将消息 m 进行有效的隐藏

密文的第二个元素是这样计算的:

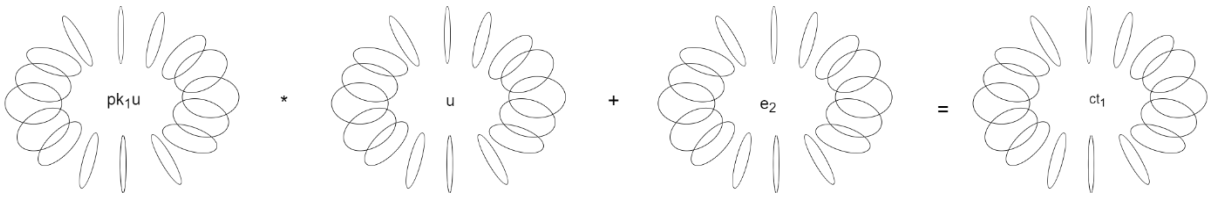

图 3.7 密文多项式的第二项

$$\begin{aligned} ct_1 = &-3163925x^{19} + 7314912x^{18} - 2859765x^{17} - 8189681x^{16} - 10348910x^{15} \\ &+ 8949536x^{14} - 7163151x^{13} + 10404596x^{12} + 4234890x^{11} + 9542582x^{10} \\ &- 6915366x^9 - 9859055x^8 - 8420035x^7 + 1103403x^6 + 1514767x^5 \\ &+ 4765440x^4 - 10020300x^3 + 10549635x^2 + 7785811x + 5495934. \end{aligned} \tag{3-15}$$

将密文中的公钥展开，观察密文的第二个多项式 $ct_1 = [au + e_2]_q$ 可知，若其乘以私

钥 s，那么我们就可以得到 $ct_1 s = [aus + e_2 s]_q$，其中的 aus 与密文中的第一个多项式相似，所以当我们解密时，可以消去这一干扰因素.

综上所述，密文可以用公钥(public key)、私钥(private key)、掩码(mask)、噪音(noise) 和消息(message)表示为

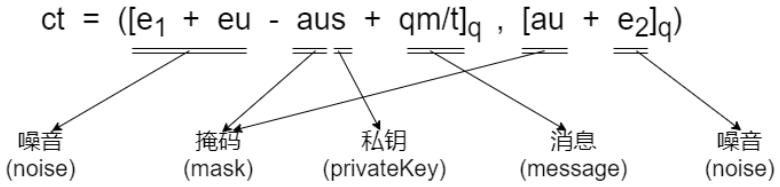

图 3.8 密文的组成

(3)解密

如上所述，将密文中的第二个多项式乘以私钥我们可以得到与第一个多项式相同的项，所以，在解密时，我们计算 $[ct_0 + ct_1 s]_q$，消除“大”的多项式的影响，只剩下了 $[qm/t + e_1 + e_2 s]_q$，此时若噪声多项式的系数不太大，我们就可以通过缩放系数将消息还原.

明确地，

$$\begin{aligned} ct_1 s + ct_0 = {} & 5x^{19} + 13x^{18} - 8x^{17} + 10x^{16} + 22x^{14} - 18x^{13} - 32x^{12} - 5x^{11} + 8x^{10} + 11x^9 \\ & + 33x^8 + 12x^7 - 12x^6 - 7x^5 + 5097x^4 + 4x^3 + 2558x^2 + 2x + 1292. \end{aligned} \tag{3-16}$$

通过观察多项式可以发现，除了我们给定的消息的系数外，其余的所有系数都小于与 $q/t = 1280$，此时我们将多项式系数缩放至 $mod\ \ t$ 范围内，那么我们就得到

$$\begin{aligned} & \frac{5x^{19}}{1280} + \frac{13x^{18}}{1280} - \frac{8x^{17}}{1280} + \frac{10x^{16}}{1280} + \frac{22x^{14}}{1280} - \frac{18x^{13}}{1280} - \frac{32x^{12}}{1280} - \frac{5x^{11}}{1280} + \frac{8x^{10}}{1280} + \frac{11x^9}{1280} \\ & + \frac{33x^8}{1280} + \frac{12x^7}{1280} - \frac{12x^6}{1280} - \frac{7x^5}{1280} + \frac{5097x^4}{1280} + \frac{4x^3}{1280} + \frac{2558x^2}{1280} + \frac{2x}{1280} + \frac{1292}{1280}. \end{aligned} \tag{3-17}$$

将系数进行四舍五入取整，多项式就会变为我们的明文消息 $m = 4x^4 + 2x^2 + 1$，由此来恢复我们的信息.

把它们放在一起，我们通过如下计算来解密密文

$$m' = [[\frac{t}{q}[ct_0 + ct_1 s]_q]]. \tag{3-18}$$

[]表示舍入到接近的整数(四舍五入).

可以发现，当我们对系数进行缩放操作的时候，如果噪声多项式的系数过大，缩放之后导致解出一个与正确值不同的整数，接着解密就会悄无声息的失败了.在本次例子中，最大的噪音为 $33/1280$，所有本次将会正确的解密出消息.

3.3 CKKS 方案的编码与解码

编码可让浮点数转变为整系数多项式，使得其符合 BFV 方案对整系数多项式加密的特点，且 CKKS 方案[20]也是 BFV 方案的衍生.

3.3.1编码

若此时拥有$n/2$维浮点数向量m，将其扩展至复数域，向量m增广为n维向量，其中前$n/2$项为原向量，后$n/2$项为共轭复数，我们将这一增广后的向量称为m'.

接下来考虑多项式模的根.由其表达式可知多项式模x^n+1拥有n个复数根ζ_i，将这些复数根组成n维向量ζ,以向量ζ为x，消息向量m'为y使用拉格朗日插值法构造出多项式f，满足

$$f(\zeta)=\Delta\bullet m'. \tag{3-19}$$

其中，Δ为放大因子，Δ的大小决定了浮点数的精度.但过大的Δ会导致多项式的系数不断变大，最终出现超出系数模的情况.

经过编码后得到了一个实数域上的整系数多项式，这就完成了编码的操作，加入了浮点数的明文空间.

3.3.2解码

解码的过程本质就是编码的逆过程，首先将放大因子Δ消去，接着根据多项式模的根向量还原出原来的消息.

3.4 半诚实敌手模型

半诚实敌手模型应用于多方安全计算中，而对于安全多方计算拥有两种安全模型，一是半诚实敌手模型(The Semi-Honest Model)，二是恶意敌手模型(The Malicious Model).

按照协议的规定并诚实的执行下去的计算参与方，这就是半诚实敌手模型.但是，执行协议的过程中计算参与方的输入输出和运行过程中得到的信息可能会被窃取，被恶意攻击者监听到.在生活中，往往被动攻击是占攻击的大部分，根据成本和开销，主动攻击相对来说更复杂和困难，由此，很多协议都是基于半诚实敌手模型下的安全而建立的，且满足半诚实敌手模型是安全多方计算的基础条件.

4 实验概述

通过 matlab 软件编写私钥、随机噪声多项式、公钥、加密、解密函数代码，接着通过 matlab 的 deploy 命令讲编写的函数打成 jar 包，最后在 Java 语言平台上运用 apache 开源的 mina 框架中的 ServerSocket 与 ClientSocket 模拟分布式情况，观察分析其同态操作效果.

4.1 实验环境

4.1.1环境要求和配置

硬件配置：Dell 笔记本电脑；Windows 10 家庭中文版；Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40GHz.

代码语言：matlab，Java.

编译工具：matlab2017R，IntelliJ IDEA 2021.2.

4.1.2实验数据参数

模多项式次数$d = 16$，模系数$q = 896$以及系数模$t = 7$.

4.1.3分析依据

通过数百次的模拟，统计计算出其在分布式情况下的加密后的密文在多次同态操作后的解密成功率，之后分析其正确性，安全性以及局限性.

5 算法的分析

5.1 同态加法

在分布式系统下，假设我们作为数据的提供者，这时我们生成一个私钥 s，根据计算参与方的数量，将私钥 s 分成与计算参与方相同的份数 n，且分开后的私钥相加在一起后等于私钥 s，即

$$s=\sum_{i=1}^{n}s_i. \tag{5-1}$$

接着在生成公钥时，根据式子 $pk=([-as+e]_q,a)$，每一个不同的私钥 s 都选择相同的多项式 a，而噪声多项式 e 仍保留其随机生成并使用后丢弃的特性.此时我们拥有 n 份私钥 s_i 生成的 n 份对应的公钥 pk_i，接着利用公钥 pk_i 加密明文块为密文 ct_i，将得到的密文 ct_i 发送给对应的计算参与方，让其进行同态加法操作.而此方案的同态加法与四则运算中的加法相同，故而两密文直接相加即可.各个计算参与方将进行运算后的密文交由云服务器，云服务器将所有密文进行汇总，得到最终的结果，最后将结果交由我们自己.我们利用自己手中的私钥，解开最后的密文，即可达到分布式下的同态加法操作.

5.2 正确性

由前文所述，结合公式(3-12)并将公钥 pk 展开，每个计算参与方 P_i 得到的密文是这种形式的

$$ct=([-as_iu+eu+e_1+\frac{qm}{t}]_q,[au+e_2]_q). \tag{5-2}$$

在计算参与方 P_1 进行多次同态加法之后，密文为

$$\begin{aligned}ct=&([-as_1(u+u_1+\cdots+u_n)+e(u+u_1+\cdots+u_n)+(e+e_1+\cdots+e_n)\\&+\frac{q(m_1+m_2+\cdots+m_n)}{t}]_q,\\&[a(u+u_1+\cdots+u_n)+(e^{'}+e^{'}_1+\cdots+e^{'}_n)]_q).\end{aligned} \tag{5-3}$$

接着，计算参与方 P_1 将计算好的结果发回云服务器后，服务器整合各个计算参与方 P_i 的密文计算结果

$$\begin{aligned}ct=&([-a(s_1+s_2+\cdots+s_n)\sum u+\sum e\sum u+\sum e+\frac{q\sum m}{t}]_q,\\&[a\sum u+\sum e^{'}]_q).\end{aligned} \tag{5-4}$$

由解密算法可得，密文的第二个多项式乘以总的私钥 s 多项式与第一个多项式相加，即可消除“大”多项式 a 对消息的掩盖，通过缩放即可获得运算之后的明文.

5.3 安全性

由前文正确性可知，若攻克了一个计算参与方，那么得到的私钥是不完整的私钥，是无法获得全部的结果，例如获得了私钥s_2，云服务器此时拥有的密文为

$$\begin{aligned} ct = ([-a(s_1+s_2+s_4)\sum u+\sum e\sum u+\sum e+\frac{q\sum m}{t}]_q, \\ [a\sum u+\sum e^{'}]_q). \end{aligned} \tag{5-5}$$

此时根据解密算法，仍然无法消除“大”多项式$-a(s_1+s_2+s_4)\sum u$对消息的掩盖，通过缩放多项式系数再四舍五入取整，这与接下来分析的局限性中的噪声过大相似，那么就会产生错误的解密，从而保证了计算的安全.唯一的弱点是攻击者获得了所有计算参与方的私钥，那么就可以通过解密算法解密出所有的数据，致使数据泄露.

5.4 局限性

对 BFV 加密算法而言，其本质是通过对多项式取模，对系数取模，接着添加噪声多项式来掩盖真实的消息，最后对密文多项式缩放，四舍五入取整确定系数.那么，由于添加噪声多项式的关系，且加法具有加法交换律，那么密文

$$\begin{aligned} a = ([pk_0u_1+e_1+qm_1/t]_q,[pk_1u_1+e_2]_q), \\ b = ([pk_0u_2+e_3+qm_2/t]_q,[pk_1u_2+e_4]_q). \end{aligned} \tag{5-6}$$

相加，得到新密文

$$\begin{aligned} &a+b \\ &=([pk_0(u_1+u_2)+(e_1+e_3)+q(m_1+m_2)/t]_q,[pk_1(u_1+u_2)+(e_2+e_4)]_q). \end{aligned} \tag{5-7}$$

而根据解密算法，解密的多项式为

$$[\frac{q(m_1+m_2)}{t}+(e_1+e_3)+e(u_1+u_2)+(e_2+e_4)s]_q. \tag{5-8}$$

随着同态加法操作的不断增加，噪声多项式的系数不断累积，当对解密多项式乘以t/q消去消息(m_1+m_2)的系数，噪音多项式的系数可能会出现有一个介于$k[q/t,q/(2t)]$的值，那么对系数四舍五入取整时，会将噪音多项式e的系数给加上，那么解密就会解出错误的结果.

针对此问题，将噪音分为三类

$$\begin{aligned} e_5 = e_1+e_3, \\ eu_3 = e(u_1+u_2), \\ e_6s = (e_2+e_4)s. \end{aligned} \tag{5-9}$$

为了直观感受与分析，分别取$q=896, t=7, d=16$对这三种噪音多项式类型分别添加了 1，5 和 30 个噪音多项式，观察其系数分布.

$e_5 = e_1 + e_3$类型噪音多项式.

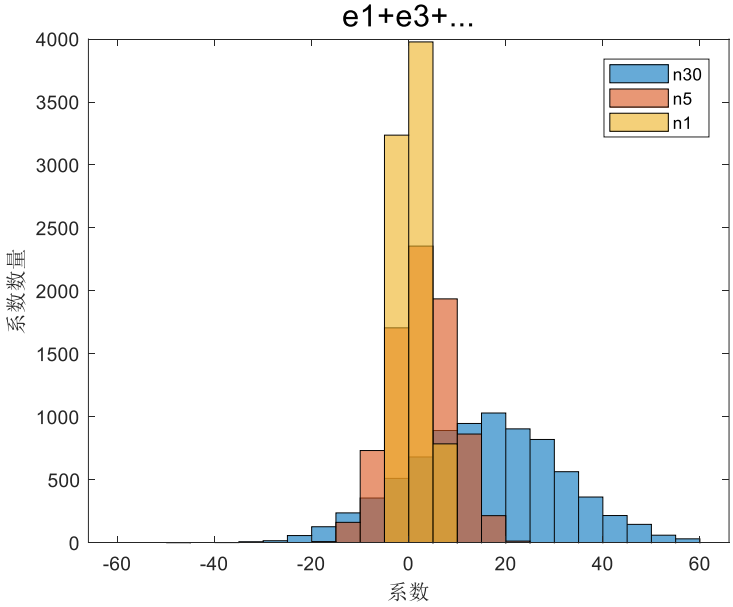

图 5.1 $e_5 = e_1 + e_3$类型各噪声多项式系数分布

当我们添加 30 个噪音多项式时，某些系数有可能会大于 64，即超过了 $q/t = 128$ 的一半，所以解密不会产生正确的结果.

另外两项表示不同的情况——第二项是一个噪音多项式乘以一些“小的多项式”(系数为-1、 0 或 1)的总和.这种乘法会产生更大的噪音.这意味着这个噪音与多项式的最高次的平方根 $\sqrt{n}$ 一致.

对这一项绘制与上面相同的分布可以看出，它比第一项大得多，而且即使对于我们示例中的参数，也存在错误解密的危险，即使只是添加了几个参数.

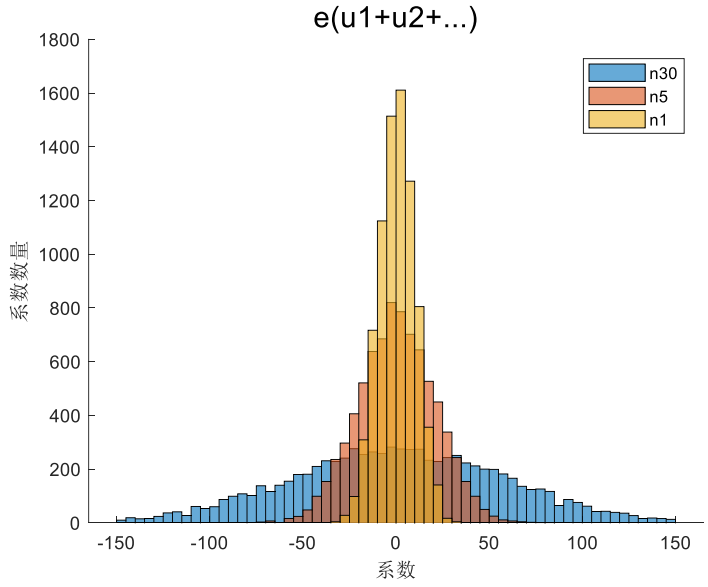

图 5.2 $eu_3 = e(u_1 + u_2)$ 类型各噪声多项式系数分布

第三项是类似的——一组噪音多项式之和，乘以一个“小的多项式”.它的噪音分布是这样的:

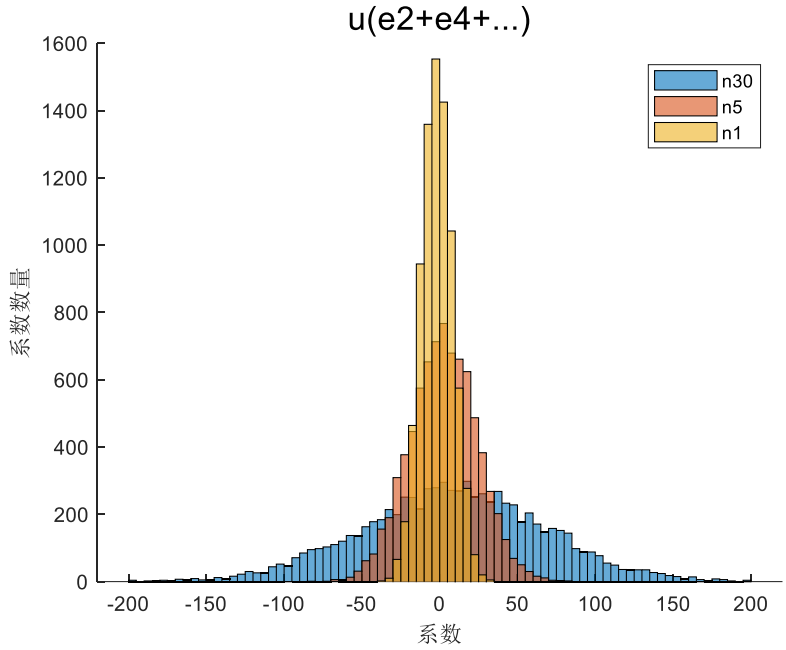

图 5.3 $e_6 s = (e_2 + e_4)s$ 类型各噪声多项式系数分布

这表明，对于我们所选择的参数，由两个以上加法产生的密文，解码错误的概率很高，而且两次加法失败的概率也很高.这是因为有时最大错误大于 64，当 q/t = 128 时，会导致不正确的解密，就像这里一样.为了给这样的操作提供更多的空间，我们需要使用更大的 q/t 比值， 这可以应对通常由所执行的操作数量引入的噪音量.

5.5 性能分析

测试方案在计算“求和”的时间代价.

实验环境通过同一台设备开设不同的端口而进行连接，忽略了设备间的通信成本.为保证语义安全，设置参数 $d = 30, t = 2048, q = 2621440$.

符号定义：

t_s：生成私钥时间；

t_{pk}：生成公钥时间；

t_{ct}：加密为密文时间；

t_{sum}：求和时间；

t_m：解密时间；

表 5.1 计算求和的时间代价

	Max/ms	Min/ms	Mean/ms
t_s	78.1	1.2	1.4
t_{pk}	93.8	0.6	9.3
t_{ct}	0.3	0	0.1
t_{sum}	62.5	0.9	0.6
t_m	0.3	0	0.1

通过上诉表格可以看到，私钥生成与公钥生成所需时间存在偶然性，使得单次时间过长，与整体时间不符.观察全部样本，得到的时间都趋于 10ms，效果较好.

6 结论

本文在半诚实模型下，关于数据在分布式计算下的隐私性对全同态加密的 BFV 方案进行分析，对方案进行了正确性，安全性与局限性的分析.能支持对数据的加密与解密， 保证数据的隐私不被泄露.

通过仿真模拟发现，当同态操作达到一定次数时，由于噪声多项式的累加影响，会对最后的解密成功率产生影响，使得解密失败，这也就导致了 BFV 方案不能进行任意次数的同态加法操作.引入了 CKKS 方案，加入编码操作，使得明文空间扩大至浮点数，但同时放大因子的加入也需要我们考虑精度问题，放大因子越大精度越高，但同样的也使得多项式的系数越大，导致计算溢出，需要更大的计算参数，使方案运算时间增加.

本文研究的是其同态加法，未来希望能加入 CKKS 的重缩放性，使得密文间的乘法得以实现，实现真正的全同态.

谢 辞

经过此次毕业论文设计，让我领略了学术的魅力， 完整的逻辑框架搭建，数据处理，结果的分析，一步一个脚印的走下去，最后直到成果的完成，这是自豪的，这是有成就感的.

这次的过程中，困难是必不可少的，但我们要迎难而上，要有“千淘万漉虽辛苦，吹尽狂沙始到金”的精神，打倒路上的纸老虎.但一个人的力量终究是弱小的，幸而我们在学校中，有同学的陪伴，有老师的支持，以及家中父母给予的鼓励.同学之间相互帮助，拥有的信息互相交流，基础算法的相互帮助，这让我们的毕设进度都能得到前进.遇到难题，进入死胡同时，父母发来的鼓励让我心中产生一丝慰藉，我不是一个人在战斗，父母永远是我身后最有力的后盾.

其中感受最深的是张必山老师，让我体会到了“山重水复疑无路，柳暗花明又一村”，给予了我建议，给了我论文指导了方向，以至于不用走太多的弯路.他的博学多才，一眼看出我论文的问题所在，非常认真负责，真诚地提出批评和指正，在此由衷的感谢张必山老师.对老师的感激，我将铭记于心.

大学时光就随着论文的结束而匆匆离去，路上我遇到了很多可爱的同学，敬业的老师，感谢你们的一路陪伴，大学的时光有你们而璀璨.

参考文献

[1] Rivest R L, Adleman L, Dertouzos M L. On data banks and privacy homomorphisms[A]. DeMillo RA Foundations of Secure Computation[C]. NY, USA: Academic Press, 1978. 169-180.

[2] Pailier P. Public-key cryptosystems based on composite degree residuosity classes[A]. Proc of the Advances in Cryptology(EUROCRYPT99)[C]. Prague, Czech Republic, 1999. 233-238

[3] Goldwasser S, Micali S. Probabilistic encryption[J]. Journal of Computer and System Sciences, 1984, 28(2): 270-299.

[4] Rivest R L, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems[J]. Comm unications of the ACM, 1978, 21(2): 120-126.

[5] Elgamal T. A public-key cryptosystem and a signature scheme based on discrete logarithms[J]. IEEE Transactions on Information Theory, 1985, 31(4): 469-472.

[6] Bonch D, Goh E J, Nissim K. Evaluating 2-DNF formulas on ciphertexts[A]. Second Theory of Cryptography Conference (TTC 2005)[C]. Cambridge, MA, USA, 2005. 325-341.

[7] Gentry C. A Fully Homomorphic Encryption Scheme[D]. California, USA: Stanford University, 2009.

[8] Smart P N, Vercauteren F. Fully homomorphic encryption with relatively small key and ciphertext sizes[A]. Proc of the Public Key Cryptography (PKC 2010)[C]. Paris, France, 2010. 420-443.

[9] Dijk V M, Gentry C, Halevi S, et al. Fully homomorphic encryption over the integers[A]. Proc of the Advances in Cryptology (EUROCRYPT 2010)[C]. Riviera, France, 2010. 24 43.

[10] Coron J, Mandal A, Naccache D, et al, Fully homomorphic encryption over the integers with shorter public keys[A]. Proc of the Advances in Cryptology (CRYPTO 201 1)[C]. Santa Barbara, California, USA, 2011. 487-504.

[11] Brakerski Z, Vaikuntanathan V. Fully homomorphic encryption from ring-lwe and security for key dependent messages[A]. Proc of the Advances in Cryptology (CRYPTO 201 1)[C]. Santa Barbara, Califomia, USA, 2011. 505-524.

[12] Gentry C, Halevi S. Implementing Gentry's fully-bomomorphic encryption scheme[A]. Proc of the Advances in Cryptology(EUROCRYPT 2011)[C]. Tallinn, Estonia, 2011. 129-148.

[13] Gentry C. Fully homomorphic encryption using ideal lattices[A]. Proc of the 41st ACM Symposium on Theory of Computing(STOC 09)[C]. Bethesda, Maryland, USA, 2009. 169-178.

[14] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in IEEE Transactions on Information Theory, vol. 31, no. 4, pp. 469-472, July 1985,

[15] Paillier P . Public-key cryptosystems based on composite degree residuosity classes[J]. Advances in Cryptology Leurocrypt, 2004.

[16] Goldwasser, S.; Micali, S.; Rivest, R. L. (1988). "A Digital Signature Scheme Secure Against

Adaptive Chosen-Message Attacks". SIAM Journal on Computing. 17 (2): 281.

[17] STOC '09: Proceedings of the forty-first annual ACM symposium on Theory of computingMay 2009 Pages 169–178.

[18] 马飞，蒋建国．具有隐私保护的分布式协作统计计算方案[J]. 计算机工程与设计, 2015, 36(09): 2383-2387.

[19] Fan J, F Vercauteren. Somewhat practical fully homomorphic encryption[J]. Iacr Cryptology Eprint Archive, 2012.

[20] Cheon J H , Kim A , Kim M , et al. Homomorphic Encryption for Arithmetic of Approximate Numbers[J]. Springer, Cham, 2017.

附 录

私钥生成

```
1. function s=genPrivateKey(d)
2.     rand('state',sum(100*clock));
3.     %生成随机 0，1，-1 矩阵
4.     S=ceil(3*rand(d))-2;
5.
6.     %随机选取其中一行作为私钥
7.     s=S(round(7),:);
8. end
```

公钥生成

```
1. function pk=genPublicKey(q,d,s,D)
2.     pk=zeros(2,length(s));
3.
4.     %随机生成一个多项式，用于生成公钥
5.     a=myMod(randsample(q+1,d)'-1,q);
6.
7.     %生成密文空间内的随机多项式
8.     a=genRandomPolynomial(q,d);
9.
10. %生成噪声多项式 e
11. e=genNoisePolynomial(d);
12.
13. pk(1,:)=BFV_multi(-a,s,D,q)+e;
```

```
14.
15. pk(2,:)=a;
16.
17. end
```

加密算法

```
1.  function ct=Enc(pk,q,t,d,D,m)
2.      %加密过程
3.      %随机生成两个噪声多项式和一个（0，1，-1）多项式
4.      %这三个多项式，用完即丢弃
5.      e1=genNoisePolynomial(d);
6.
7.      e2=genNoisePolynomial(d);
8.
9.      u=genPrivateKey(d);
10.     ct0=BFV_multi(pk(1,:),u,D,q)+e1+conv(m,q)/t;
11.     ct1=BFV_multi(pk(2,:),u,D,q)+e2;
12.     ct=[round(ct0);round(ct1)];
13. end
```

解密算法

```
1.  function m=Dnc(ct,q,t,D,s)
2.      %开始解密
3.      ct0s_ct1=BFV_multi(ct(2,:),s,D,q)+ct(1,:);
4.      myMod(ct0s_ct1,q);
```

```
5.      m=round(deconv(conv(ct0s_ct1,t),q));
6.  end
```

CKKS 编码

```
1.  function res=Encode(message,X,delta)
2.      [n,m] = size(message);
3.      X=myRoot(roots(D));
4.      Y=message;
5.      m=round(sym2poly(simplify(myLagrange(X,Y,delta))));
6.  end
```

Java 仿真模拟

服务器

```
1.  private static List<IoSession> sessionList = new ArrayList<>();
2.  private static List<Message> messageList = new ArrayList<>();
3.  private static int index = 0;
4.
5.  private static int userNum = 2;
6.  private static int curUser = 0;
7.
8.  private static int[][] sum = new int[2][16];
9.
10.
11.
12. public static int[][] add(int[][] ct){
```

```
13.         System.out.println(Arrays.toString(ct[0]));
14.         return BFVUtils.add(sum, ct);
15. }
16.
17.
18.
19.
20. @Override
21. public void messageReceived(IoSession ioSession, Object o) throws Exception {
22.          System.out.println(ioSession.getId()+"进来了");
23.          if(o instanceof  List){
24.                System.out.println("is List");
25.                messageList.addAll((ArrayList<Message>)o);
26.                sessionList.add(ioSession);
27.                ioSession.write("数据发送成功");
28.                System.out.println("数据接收完毕");
29.          }
30.           if(o instanceof  String){
31.                System.out.println(o instanceof String);
32.                String info = (String)o;
33.                if("ok".equalsIgnoreCase(info)){
34.                      sessionList.get(0).write(new Message(sum));
35.                      System.out.println("整合完毕");
36.                }
37.                f("connect".equalsIgnoreCase(info)){
```

```
38.                 curUser++;
39.                 sessionList.add(ioSession);
40.                 oSession.write(new String("连接成功"));
41.                  System.out.println(ioSession.getId()+": 连接成功");
42.             }
43.             if("getCT".equalsIgnoreCase(info)){
44.                 if(curUser == userNum){
45.                     if(index == messageList.size()){
46.                         ioSession.write("没啦，别要了");
47.                     }else{
48.                         List<Message> list = new ArrayList<>();
49.                         Message message = messageList.get(index++);
50.                         Message message2 = messageList.get(index++);
51.                         list.add(message);
52.                         list.add(message2);
53.                         ioSession.write(list);
54.                     }
55.                 }else{
56.                     ioSession.write(new String("等一下，人没到齐."));
57.                 }
58.             }
59.         }
60.          if(o instanceof Message){
61.             Message info = (Message) o;
62.             sum = add(info.getCt());
```

```
63.             System.out.println("接收到("+ioSession.toString()+")发送的密文:");
64.             ioSession.write("感谢你的计算");
65.         }
66. }
```

客户端

```
1.  private  int[][] sum = new int[2][16];
2.
3.  public  int[][] add(int[][] ct){
4.          System.out.println(Arrays.toString(ct[0]));
5.          return BFVUtils.add(sum, ct);
6.  }
7.
8.  @Override
9.  public void messageReceived(IoSession ioSession, Object o) throws Exception {
10.         System.out.println(o instanceof List);
11.         if(o instanceof  List){
12.             List info = (ArrayList<Message>) o;
13.             for(int i = 0 ; i < info.size() ; i++){
14.                 Message message = (Message) info.get(i);
15.                 sum = add(message.getCt());
16.             }
17.             Message totalMessage = new Message(sum);
18.             ioSession.write(totalMessage);
19.         }
```

```
20.     if(o instanceof String){
21.         System.out.println("Echo: "+(String)o);
22.     }
23. }
```

局限性分析

```
1.  % 参数说明
2.  % q：多项式系数模
3.  % d：模多项式的次数
4.  % m：自定义实验的次数
5.  % n：同态加法次数
6.
7.  function res=e5(q,d,m,n)
8.    %声明临时变量，用于存储多次实验的结果
9.    res=zeros(d,m);
10.   for i=1:m
11.     %声明临时变量，存储一次的计算结果
12.     temp_res=zeros(1,d);
13.     for j=1:n
14.       %模拟多次加法的操作
15.       temp_res = myMod(temp_res+genNoisePolynomial(d),q);
16.     end
17.     %将结果存储到 res 中去
18.     res(:,i)=temp_res';
19.   end
```

```
20. end
```

```
1.  % 参数说明
2.  % q：多项式系数模
3.  % d：模多项式的次数
4.  % m：自定义实验的次数
5.  % n：同态加法次数
6.
7.  function res=eu3(q,d,m,n)
8.      %声明临时变量，用于存储多次实验的结果
9.      res=zeros(d,n);
10.     %噪声多项式 e
11.     e=genNoisePolynomial(d);
12.     %模多项式 D
13.     D=genPolynomial(d);
14.     for i=1:m
15.         %声明临时变量，存储一次的计算结果
16.         temp_res=zeros(1,d);
17.         for j=1:n
18.             %模拟多次加法的操作
19.             temp_res = temp_res+genPrivateKey(d);
20.         end
21.
22.         temp_res=BFV_multi(temp_res,e,D,q);
23.
```

```
24.       %将结果存储到 res 中去
25.       res(:,i)=temp_res';
26.    end
27. end
```

```
1.  % 参数说明
2.  % q：多项式系数模
3.  % d：模多项式的次数
4.  % m：自定义实验的次数
5.  % n：同态加法次数
6.
7.  function res=e6s(q,d,m,n)
8.     %声明临时变量，用于存储多次实验的结果
9.     res=zeros(d,n);
10.    %私钥 s
11.    s=genPrivateKey(d);
12.    %模多项式 D
13.    D=genPolynomial(d);
14.    for i=1:m
15.       %声明临时变量，存储一次的计算结果
16.       temp_res=zeros(1,d);
17.       for j=1:n
18.          %模拟多次加法的操作
19.          temp_res = temp_res+genNoisePolynomial(d);
20.       end
```

```
21.
22.         temp_res=BFV_multi(temp_res,s,D,q);
23.
24.         %将结果存储到 res 中去
25.         res(:,i)=temp_res';
26.     end
27. end
```