毕业论文(设计)

论文（设计）题目：基于同态加密的单服务器隐私信息检索方案

姓　　名 谢钟萱

学　　号 201800150063

学　　院 网络空间安全学院（研究院）

专　　业 网络空间安全

年　　级 2019 级

指导教师 陈宇

2023 年　5 月　30 日

摘　　要

隐私信息检索（PIR）是现代密码学的重要应用之一，在以数据库为中心的各类应用中起着关键的用户隐私保护作用。然而 PIR 安全目标直接导致了线性计算量的理论下限，因此现有的 PIR 协议的性能表现被通信量、计算量两方面严重限制，使得这些协议在现实世界中的部署远不实用。

Carlos 等人在 2016 年首次提出了一个现实可用的基于同态加密的 PIR 方案，打破了 Sion 和 Carbunar 在 2007 年文章中提出的 PIR 不可能实用的假想。随后多篇相关论文沿其思路进行深入研究，给出了多种方案构造和优化手段。

本文首先对 2016 年以后的各种基于同态加密的 PIR 方案进行深入分析，提炼出各方案的核心思想和主要优化手段，并从通信量、计算量、预存储量等方面对比各种方案的优劣。同时本文还基于 Regev 加密体系给出了一个 PIR 方案构造，通过公共预计算线索和较小的客户端存储量进行计算加速，达到了较好的通信-计算-存储量权衡，并给出了在原方案上通过 Batch Code 进行批量询问处理的变种，另外本文还讨论了现实数据库和理论方案的差异，并给出了可能的解决方式。除了理论构造，本文还给出了方案的工程化实现和效率测试，使得该方案具有实用意义。

关键词：隐私信息检索；同态加密；batch code

ABSTRACT

Privacy Information Retrieval (PIR) is one of the key applications of modern cryptography, playing a critical role in protecting user privacy in a wide variety of database-centric applications. However, the security goals of PIR lead directly to a theoretical lower bound on the amount of linear computation. Therefore, the performance of existing PIR protocols is severely limited by both the amount of communication, and the amount of computation, making these protocols far from practical for real-world deployment.

Carlos et al. first proposed a realistically usable PIR scheme based on homomorphic encryption in 2016, breaking the hypothesis that PIR could not be practical, as proposed by Sion and Carbunar in their 2007 paper. Several related papers subsequently followed their ideas in depth, giving a variety of scheme constructions and optimisation.

In this paper, we first analyse various PIR schemes based on homomorphic encryption after 2016, extract the core and compare the advantages and disadvantages of various schemes in terms of communication volume, computation volume and pre-storage capacity. The paper also presents a PIR scheme construction based on the Regev encryption system, which achieves a better communication-computation-storage trade-off by accelerating the computation with common precomputation hints and small client-side storage. And then we present a batch variant of the original scheme. In addition, differences between the real-world database and the theoretical solution are discussed and possible solutions are given. In addition to the theoretical constructs, engineering implementations and efficiency tests of the scheme are given, making the scheme of practical interest.

Key words: privacy information retrieval, homomorphic encryption, batch code

目　录

第 1 章　绪论

1.1　研究背景与意义

根据 2022 年发布的第 50 次《中国互联网络发展状况统计报告》，中国的网民规模已达到 10.51 亿，互联网普及率达 74.4%。互联网给人们提供了便捷、互通、开放等特性，其影响力在 21 世界无远弗届。然而，人们在互联网上所留下的足迹却不可避免地给个人隐私带来了损害。随着信息茧房、数据“杀熟”、隐私外泄等一系列风险和危害的出现，隐私保护已经成为了当今互联网治理和发展的必答题。

隐私保护计算正是通过密码学、数据科学等多个领域的交叉来解决上述问题的技术手段之一。这项技术的目的是为了提供对数据计算过程和数据计算结果的安全保护能力，其中又包含安全多方计算、机密计算、同态加密、差分隐私等多个领域。

隐私信息检索（Private Information Retrieval，PIR）是现代隐私保护系统中的一个典型应用。在当下云数据库、集中式数据存储和服务器检索依旧是互联网运作的主要模式，而这些检索本身可能会泄露一些敏感信息。例如对于医疗数据的检索可能会泄露公民的健康信息、对于金融数据的检索模式可能会泄露投资策略等。而 PIR 技术的目的，就是允许客户端对一个服务器上的数据库中的某条数据进行检索，而不对服务器泄露客户端具体检索了哪条数据。通过 PIR 技术，客户端可以将敏感数据存储在不被信任的服务器上，同时还能保证自身的隐私安全。

虽然 PIR 技术有着很强的安全价值，但是它的代价却极其昂贵，且很难降低——如果想要对服务器隐藏客户端的询问索引，那么即使客户端仅发送了一个询问，服务器也必须对数据库中的每一项条目进行操作。如果只对部分条目进行操作，服务器就能知道被遗漏的那部分条目并不是客户端的询问对象，这也就违背了 PIR 的安全保证。高昂的计算代价通常伴随着同等昂贵的通信代价，这使得大部分的 PIR 方案很难被实际应用。

1.2 研究现状

Chor 等人在[1]中首次提出了关于隐私信息检索（private information retrieval，PIR）概念。假设一个服务器持有一个公开数据库 DB ，而客户端向服务器发送询问来检索 DB 上的特定元素。举例来说，在 DNS 询问场景下，每台主机向服务器发送一个特定域名，而服务器返回其对应的 ip 地址。不失一般性的，假设这个数据库中元素的索引为 $\{1,2,\dots,n\}$，而客户端的询问索引 $i \in \{1,2,\dots,n\}$ 。 虽然这个数据库是公开的，客户端依然想要向服务器隐藏自己的询问索引 i 。

PIR 问题后来演化出了信息理论 PIR（Information theoretic PIR, IT-PIR）和计算性安全（Computational PIR, CPIR）两个分支。IT-PIR 关注数据库分布在几个不共谋的服务器上的情况：客户端对几个服务器上分别发起询问，并将回复在本地组合起来，其主要缺点则在于需要进行服务器不共谋的假设。CPIR 则关注数据库由单一服务器管理的情况，方案的设计都需要基于密码学中的困难假设，这种解决方案的主要缺点则是在通信和计算成本方面的巨大开销。

对 CPIR 方案的设计和寻找其实际应用的具体有效的构造一直是密码学中的一个活跃领域。Kushilevitz 和 Ostrovsky[2]于 1997 提出了第一个基于线性同态加密的单服务器的 CPIR 方案。Beimel 等人在[3]中证明，如果数据库没有存储任何冗余信息来辅助计算，服务器端总需要 $\Omega(n)$ 的计算量来回答一个客户端查询，其中 n 代表数据库中的元素数量。为了规避这个下限，他们引入了预处理的概念：在离线阶段运算所有不可避免的线性计算并生成辅助数据；而在线阶段则通过预处理的辅助数据来进行回复，使得在线计算成本可以达到 n 的亚线性级别。在此之后，一系列关于如何构建服务器或客户端上的辅助数据，从而降低 PIR 的通信和计算成本的成果被陆续提出。

在 CPIR 领域的最新研究成果几乎都使用了加法同态加密。2016 年 Melchor 等人首次构造了一个代价相对较小、现实可用的方案——XPIR[4]，该方案基于 Ring-LWE 问题，并使用多项式上的数论转换和中国剩余定理等方法来加速多项式乘法；同时通过为多次使用的特定操作数预先计算牛顿系数的方式进一步降低了

计算代价。Angel 等人随后在 XPIR 的基础上提出了 SealPIR[5]的方案，该方案提出了一种压缩询问技术，来降低通信量。

而另一条研究路线则是基于全同态[6]来构建 CPIR 方案，主要关注点集中在如何降低同态乘法带来的高噪音。Asra Ali 等人在[7]中提出基于乘法同态加密的 PIR 方案—— MulPIR，在具有大条目的数据库上有着较好的通信量表现。[8][9] 等方案通过结合不同的同态方案做到了噪音控制和密文大小更好的权衡。

除了从同态方案的自身性质来降低方案代价，还有一部分研究关注如何将 IT-PIR 的思想和 CPIR 相结合，利用编码的手段来规避 $\Omega(n)$ 的计算量下限。这类方案通常能和其他 CPIR 方案相容，在其之上增加对数据库和客户端询问的编码构造，能够达到亚线性的时间复杂度。

本文将在第 3 章详细分析现有的 CPIR 方案构造和具体性能表现。

1.3 本文贡献

本文首先对于目前基于单服务器的 PIR 方案进行整理，从最新研究中提炼核心思想和主要改进，并对其进行分类和总结。

本文给出了当下所有 CPIR 方案的发展路径和具体关系，将其分为基于加法同态、基于乘法同态和引入 IT-PIR 技术的混合构造三个主要部分。对于各部分中的不同方案设计，本文对其进行理论复杂度的整理，并对比了各个方案的在通信量、计算量以及预计算存储量上的优劣势。

在此之上，本文从[10]中对于 Regev 方案的最新高效构造入手，构建了一个带预处理的 CPIR 方案，将线性复杂度的计算量分散到离线阶段，提升了在线阶段服务器的吞吐量。同时本文还引入之前研究中和本方案相容的改进（多维数据库表示、密文分块、Batch Code）等手段，进一步降低了方案的各种复杂度。

另外本文还讨论了真实数据库和理论方案的出入，提出了三种理论方案无法直接处理的具体数据库（不均匀、动态、键值对），并给出了可能的解决方案。

第 2 章　预备知识

2.1　记号说明

在本文中，使用加粗小写字母（例如 $\boldsymbol{v}$）代表向量，加粗大写字母（例如 $\boldsymbol{M}$）代表矩阵；上标代表矩阵/向量维度（例如$\mathbb{Z}^n/\mathbb{Z}^{n\times m}$），下标代表模数（例如$\mathbb{Z}_p$）。

本文常用符号说明见表 2-1。

表 2-1 常用符号

符号	含义
$\mathbb{Z}$	整数环
$\mathbb{Z}_p$	模 p 整数环
$[N]$	序列 $\{0,1,2,\dots,N-1\}$
R_t	多项式环，明文模数为 t
$\|\|s\|\|$	无穷范数，向量中的所有元素中的最大值或多项式中绝对值最大的系数
$x \leftarrow S$	S 代表一个集合，x 是从 S 中获得的随机取样
$x \leftarrow \mathcal{X}$	$\mathcal{X}$ 代表一个分布，x 是符合分布$\mathcal{X}$的随机取样

2.2　带误差学习问题

带误差学习问题（Learning with Error, LWE）[11] 是大部分 CPIR 方案的安全性依赖。

定义 1（带误差学习 Learning with Error, LWE）LWE 问题基于整数环 $\boldsymbol{R}=\mathbb{Z}$。对于某个安全系数 $\boldsymbol{\lambda}$，分别取整数模数 $\boldsymbol{q}\leftarrow\boldsymbol{q}(\boldsymbol{\lambda})$，维度 $\boldsymbol{n}\leftarrow\boldsymbol{n}(\boldsymbol{\lambda})$ 和某个$\mathbb{Z}$上的分布 $\boldsymbol{\mathcal{X}}\leftarrow\boldsymbol{\mathcal{X}}(\boldsymbol{\lambda})$。记分布 1 为均匀取样$(\boldsymbol{\alpha_i},\boldsymbol{b_i})\in\mathbb{Z}_{\boldsymbol{q}}^{\boldsymbol{n}}\times\mathbb{Z}_{\boldsymbol{q}}$；分布 2 为均匀采样$\boldsymbol{s},\boldsymbol{\alpha_i}\in\mathbb{Z}_{\boldsymbol{q}}^{\boldsymbol{n}}$，$\boldsymbol{e_i}\in\boldsymbol{\mathcal{X}}$所得到的 $(\boldsymbol{\alpha_i},\langle\boldsymbol{\alpha_i},\boldsymbol{s}\rangle+\boldsymbol{e_i})$。$\mathrm{LWE}_{\boldsymbol{n},\boldsymbol{q},\boldsymbol{\mathcal{X}}}$问题假设即分布 1 和分布 2 是不可区分的。

定义 2（环上带误差学习 Ring Learning with Error, RLWE）RLWE 问题是基于某个分圆域 $\boldsymbol{R}=\mathbb{Z}[\boldsymbol{x}]/\boldsymbol{f}(\boldsymbol{x})$，其中 $\boldsymbol{f}(\boldsymbol{x})$ 是一个分圆多项式，通常取$\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{x}^{\boldsymbol{d}}+\boldsymbol{1},\boldsymbol{d}$为某个 2 的幂。对于某个安全系数 $\boldsymbol{\lambda}$，分别取整数模数 $\boldsymbol{q}\leftarrow\boldsymbol{q}(\boldsymbol{\lambda})$和某个$\boldsymbol{R}$上的分布 $\boldsymbol{\mathcal{X}}\leftarrow\boldsymbol{\mathcal{X}}(\boldsymbol{\lambda})$，则$\boldsymbol{R_q}=\boldsymbol{R}/\boldsymbol{qR}$。记分布 1 为均匀取样$(\boldsymbol{a_i},\boldsymbol{b_i})\in\boldsymbol{R_q^2}$；分布 2 为均匀采样$\boldsymbol{s},\boldsymbol{a_i}\in\boldsymbol{R_q},\ \boldsymbol{e_i}\leftarrow\boldsymbol{\mathcal{X}}$所得到的 $(\boldsymbol{a_i},\boldsymbol{a_i}\cdot\boldsymbol{s}+\boldsymbol{e_i})$。$\boldsymbol{R}\mathrm{LWE}_{\boldsymbol{d},\boldsymbol{q},\boldsymbol{\mathcal{X}}}$问题假设即分布 1 和分布 2 是不可区分的。

在大部分密码体系中，分布 $\boldsymbol{\mathcal{X}}$ 都采用高斯分布。

2.3 同态加密

2.3.1 同态加密基本定义

定义 1（同态加密 Homomorphic Encryption, HE） 同态加密方案通过密钥生成、加密、解密、估值四个算法来定义，即 $\mathcal{HE}=(\mathrm{KeyGen},\ \mathrm{Enc},\mathrm{Dec},\ \mathrm{Eval})$。对于 某些特定函数 $f\in\mathcal{F}$，密文 $c_1\leftarrow\mathrm{Enc}(pk,m_1),\ \cdots,c_t\leftarrow\mathrm{Enc}(pk,m_t)$，进行估值操作得到新的密文 $c\leftarrow\mathrm{Eval}(pk,f,c_1,\ldots,c_t)$，对其进行解密能够得到 $\mathrm{Dec}(sk,c)=f(m_1,\ldots,m_t)$。同时 Enc, Dec 算法需要符合普通公钥加密方案安全性假设。

由于函数 f 可以表示为一系列电路门，因此可以将其表示为以下操作。假设同态加密明文空间为 $\mathcal{P}$，对于任何$m_1,m_2,\lambda\in\mathcal{P}$ ，$\mathcal{HE}$方案支持密文操作+, ×, · ：

1. Sum: $\mathrm{Enc}(sk,m_1)+\mathrm{Enc}(sk,m_2)=\mathrm{Enc}(sk,(m_1+m_2)\bmod t)$
2. Mul: $\mathrm{Enc}(sk,m_1)\times\mathrm{Enc}(sk,m_2)=\mathrm{Enc}(sk,(m_1\times m_2)\bmod t)$
3. Absorb: $\mathrm{Enc}(sk,m_1)\cdot\lambda=\mathrm{Enc}(sk,m_1\cdot\lambda\bmod t)$

定义 2（部分同态加密 Partly homomorphic Encryption, PHE）如果仅支持定义 1 中的第 1 条（+操作）或第 2 条（×操作）性质，而不支持另外一条，则称为部分同态加密。特别的，支持+操作称为**加法同态加密**，支持×操作的则称为**乘法**

同态加密。

定义 3（Somewhat 同态加密 Somewhat homomorphic Encryption, SHE）如果同态加密方案既支持+操作，又支持×操作，但是仅进行有限数量的同态操作，那么称为 Somewhat 同态加密。

现有的所有 SHE 方案都会得到一个包含噪音的密文，而在密文上进行同态操作将会增加噪音等级。当进行了多次操作，导致增加的噪音超过方案给定的限度时，密文将不能被正确解密。通常密文乘法（即定义 1 中 Mul 操作）将会导致很高的噪音增长，而密文加法（Sum 操作）和明-密文乘法（Absorb 操作）的噪音增长相对较小。

定义 4（全同态加密 Fully homomorphic Encryption, FHE）支持无限次数的同态操作的同态加密方案称为全同态加密。

为了密文随机化，显然密文所在域是比明文要更大的。因此定义密文扩因子 F ，它在 PIR 方案中是影响方案表现的一个重要因素。

定义 5（密文扩张因子 Ciphertext Expansion Factor）：密文扩张因子 F 代表了密文大小和明文大小之间的比率，即密文比特数/明文比特数。

接下来将简要介绍几个后文中所使用的同态加密方案。

2.3.2 Regev 加密方案

Regev 加密方案是在[11]中提出的基于 LWE 困难问题的加法同态方案，并在[12] 中提出了由加密单比特延伸到加密矩阵的多明文打包方案。原始的 Regev 方案中加密的是一个标量，在后来的工作中拓展出了加密向量、矩阵的变种。当明文是一个标量 $m \in \mathbb{Z}_p$ 时，私钥是一个向量 $\boldsymbol{s} \in \mathbb{Z}_q^n$，而密文是向量 $(\boldsymbol{a}, c) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ 。以下是 Regev 加密体系中向量对称加密版本的具体算法：

1. KeyGen：从 $\mathbb{Z}_q^n$ 上均匀随机选择密钥 $\boldsymbol{s}$。
2. Enc：计算 $c \leftarrow \left[\boldsymbol{a}^T\boldsymbol{s} + e + \left\lfloor \frac{q}{p} \right\rfloor \cdot m\right]_q$，其中随机选取向量 $\boldsymbol{a} \in \mathbb{Z}_q^k$，错误向 $e \leftarrow \chi$。输出密文 $(\boldsymbol{a}, c) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$。

3. Dec：计算 $m' \leftarrow \text{round}\left((c - a\text{^}T\ \boldsymbol{s})/\left\lfloor\frac{q}{p}\right\rfloor\right)$，当 $\|\boldsymbol{e}\| < \frac{q}{2p}$ 时能够恢复 $\boldsymbol{m}$。

Regev 体系支持同态加法和较小的标量乘法，每次进行操作时噪音呈线性叠加。Regev 的密文扩张因子为 $F_{regev} = \frac{\log p}{\log q}\frac{k}{n+k}$。当选取 $k \gg n, q \approx p^{1+\epsilon}$时，$F_{regev}$ 能够小于 $\frac{1}{2}$。

2.3.3 BFV 加密方案

BFV 加密方案是定义在多项式环 $R_t = \mathbb{Z}_t[x]/x^N - 1$上的，其中 N 是多项式环的度数，通常是某个 2 的幂次；t 是**明文模数**，代表多项式中的每个系数都是 $\mathbb{Z}_t$ 上的一个元素。在 BFV 加密方案中，明文 m 是 R_t 上的一个多项式，而密文 (c_0, c_1) 则是R_q 上的两个多项式，其中 q 被称为**系数模数**。其具体算法如下：

1. KeyGen：从 R_q 上进行随机选取，输出多项式 $s \leftarrow R_q$。
2. Enc：计算 $(c_0, c_1) = (a, b + e + m)$，其中， e 是一个噪音多项式，其系数符合分布 $\mathcal{X}$，$b = a \cdot s + e$。
3. Dec：$\mu = c_1 - c_0 \cdot s = e + m$，由于噪音 e 较小，因此对 μ 取整就能够恢复 m 。

在以上的参数中，系数模数 q 决定了密文中能够包含多少噪音，但是更大的 q 能够会导致安全系数的下降；明文模数 t 越小，同态操作所带来的噪音增长就越小，然而由于系数空间减小，同样长度的数据就需要更多的明文来表现。在实际使用中，q 通常远大于 t。对于 BFV 方案来说，由于明密文多项式度数相同，而系数分别在$\mathbb{Z}_t$和$\mathbb{Z}_q$上，同时密文含有两个多项式，因此密文扩张因子的计算公式为：

$$F_{BFV} = \frac{2\log q}{\log t}$$

正如之前所说，不同的同态操作会带来不同的噪音增长，同时各操作的计算代价也不同。以下对 BFV 方案中的同态操作进行具体分析，此处的操作分别对应**定义 1** 中的Sum, Mul, Absorb记号。

1. BFV 密文加法（Sum）：已有$c_1 = \text{Enc}(s, p_1(x)), c_2 = \text{Enc}(s, p_2(x))$ ，且

$c_1, c_2 \in R/qR$，则能够计算两明文相加的加密结果：$c_1 + c_2 = \mathrm{Enc}(s, p_1(x) + p_2(x))$，其中$p_1(x) + p_2(x) \in R/tR$。在密文加法中噪音的增长是两个原有密文$c_1$，$c_2$ 中包含的噪音的累加值，而计算代价则是一次多项式加法。

2. BFV 密文乘法（Mul）：已有 $c_1 = \mathrm{Enc}(s, p_1(x))$，$c_2 = \mathrm{Enc}(s, p_2(x))$，且 $c_1, c_2 \in R/tR$，则能够计算两明文相乘的加密结果：$\frac{t}{q} c_1 \times c_2 = \mathrm{Enc}(s, p_1(x) \times p_2(x))$，其中 $m_1 \times m_2 \in R/tR$。该操作的噪音增长大约为 $2tN^2||s||$ 倍，计算代价为一次多项式乘法。密文乘法可能会导致密文多项式的增加，即$\frac{t}{q}\ c_1 \times c_2 = \frac{t}{q}\left[\left(c_1^{(1)} \cdot\ c_2^{(1)}\right),\ \left(c_1^{(1)} \cdot\ c_2^{(2)} +\ c_1^{(2)} \cdot\ c_2^{(1)}\right), c_1^{(2)} \cdot\ c_2^{(2)}\right]$。
3. BFV 明文-密文乘法（Absorb）：有$p_1(x), c_2 = \mathrm{Enc}(s, p_2(x))$，且$p_1(x) \in R/tR, c_2 \in R/qR$ 则能够计算两明文相乘的加密结果：$p_1(x) \times c_2 = \mathrm{Enc}(s, p_1(x) \times p_2(x))$，其中 $p_1(x) \times p_2(x) \in R/tR$。噪音增长大约为原来密文中噪音的$N||m_1||$倍，比密文乘法小很多。
4. BFV 密文重线性化：进行了密文乘法，后需要通过重线性化（Relinearization）来将密文多项式恢复为 2 个，否则密文数量和噪音将剧烈增加，影响同态操作的次数。

2.3.4 GSW 加密方案

GSW 方案是基于 LWE 问题的复杂性，加密过程包括将明文信息编码为一个格上的标量，向其添加少量的噪声，然后使用公钥对其进行加密。

为了增加冗余度，GSW 方案中使用了工具矩阵的方式，来将一个矩阵 $\boldsymbol{X}$ 转换成一个维度更大、其中每个元素更小的新矩阵 $\boldsymbol{Y}$。实际上工具矩阵的作用就是以某个底数对原矩阵做比特分解。

定义 6 (工具矩阵 Gadget Matrix) 将矩阵 $\boldsymbol{G} \in R_q^{m \times n}$ 称为工具矩阵。对于某个底数 B，通常如下设定 $\boldsymbol{G}$

$$\begin{cases} \boldsymbol{g} = \left(1, B, \dots, B^{\lfloor \log_B q \rfloor}\right) \in R_q^t, \ t = \lfloor \log_B q \rfloor + 1 \\ \boldsymbol{G} = \boldsymbol{I}_n \otimes \boldsymbol{g} \ = [I, zI, \dots, z^{t-1}I] \in R_q^{m \times n}, m = nt \end{cases}$$

对于一个维度为 $n \times c$ 的矩阵 $\boldsymbol{X}$，有 $\boldsymbol{G} \cdot \boldsymbol{Y} \equiv \boldsymbol{X} \pmod q$ ，记 $\boldsymbol{Y} = \boldsymbol{G^{-1}(X)}$ 为一个满足的 $m \times c$ 维矩阵。显然 $\boldsymbol{G^{-1}(X)}$ 总是一个在有理数域上满秩的矩阵，并且在 $\boldsymbol{G^{-1}(X)}$ 中每个元素的值都在 $\left[-\frac{z}{2}, \frac{z}{2}\right]$ 中。

在 GSW 加密方案中，明文是一个小标量 $m \in R_q$，密文则是一个矩阵 $\boldsymbol{C} \in R_q^{n \times m}$，具体加密方案如下：

1. KeyGen: 随机选择 $\boldsymbol{s'} \in \chi^n$，输出私钥 $\boldsymbol{S} \leftarrow (-I | S')$。公钥为随机矩阵 $\boldsymbol{P}$，满足 $\boldsymbol{SP} = \boldsymbol{E}$，其中 $\boldsymbol{E}$ 是一个范数较小噪音矩阵。
2. Enc: 随机选择范数较小的矩阵 $\boldsymbol{X} \in R^{m \times m}$，计算 $\boldsymbol{C} = m \cdot \boldsymbol{G} + \boldsymbol{P} \cdot \boldsymbol{X} \in R_q^{n \times m}$
3. Dec: 计算 $\boldsymbol{S} \cdot \boldsymbol{C} = m \cdot \boldsymbol{G} \cdot \boldsymbol{S} + \boldsymbol{P} \cdot \boldsymbol{X} \cdot \boldsymbol{S} = m \cdot \boldsymbol{G} \cdot \boldsymbol{S} + \boldsymbol{E'}$，由于 $\boldsymbol{E'} = \boldsymbol{E} \cdot \boldsymbol{X}$ 的范数也较小，所以能够能够被忽略。

GSW 体系支持同态加法和同态乘法。在进行同态乘法时，需要将第一个密文乘上第二个矩阵的冗余版本，即$\boldsymbol{C'} = \boldsymbol{C_1} \times \boldsymbol{G^{-1}(C_2)}$。而在同态乘法中，增长的噪音相对于两个原密文中的噪音时非对称的，具体公式为$noise\left(C_1 \cdot G^{-1}(C_2)\right) = poly(\lambda) \cdot noise(C_1) + m_1 \cdot noise(C_2)$。当如果密文 C_1 所加密的明文 $m_1 = (0,1)$ 时，同态乘法的噪音增长时**加法线性**的。

GSW 的密文扩展因子为 $F_{GSW} = \frac{1}{n \times m}$，也就是选择的矩阵大小。

2.4 隐私信息检索

2.4.1 隐私信息检索问题定义

在[1]中提出的对隐私信息检索问题的定义如下

定义 6 (隐私信息检索, PIR)：一个隐私信息检索协议的基本设置如下：服务器持有数据库 $D = \{D_1, \dots, D_n\}$，其中每个条目 D_i 长度为 l；客户端持有一个输入索引i。PIR 协议的目标是在保证服务器完全不知道i的情况下取出 D_i。一个 PIR 方

案中含有两个算法{Query, Eval}：

1 $q \leftarrow \text{PIR.Query}(i)$ **请求生成算法**，客户端通过索引i生成相应请求的算法。

2 $[\![D_i, \perp]\!] \leftarrow \text{PIR.Eval}([\![q, D]\!])$ **请求估值算法**，这是一个两方计算的协议，其输入是客户端编码的询问 q 和服务器的数据库 D；输出是对应的 D_i。大部分 PIR 方案都是用非交互的方式构建，因此可以将估值协议替换为如下两个算法：

2.1 $r \leftarrow \text{PIR.Response}(D, q)$ **回复生成算法**，服务器通过客户端生成的询问 q 和数据库 D，计算出包含条目 D_i 的加密回复 r。

2.2 $D_i \leftarrow \text{PIR.Extract}(r)$ **回复提取算法**，客户端通过服务器的回复 r，运行该算法来提取出预期条目 D_i。

2.4.2 PIR 安全要求

一个 PIR 方案应该满足以下两点要求：

1. 正确性：对于索引 i 和数据库 D 运行算法 $\text{Eval}(D, \text{Query}(i))$，所得到的条目 D_i 必须和客户端索引所对应的条目一致。
2. 隐私性：对于任意索引 $i, j \in [1, \cdots, n]$ 运行 Query 算法得到 q_i, q_j，记其长度为 l_q。取符合均匀分布的随机串 $r = \{0,1\}^{l_q}$，符合概率分布

$$\Pr(q_i = r) = \Pr(q_j = r)$$

第 3 章　同态 CPIR 方案构造及性能比较

3.1　基于加法同态加密的 CPIR 方案及改进

以加法同态加密作为基础构造是当前 CPIR 方案研究的主线。正如上文所提到的，同态密文乘法操作通常会带来很大的噪音及计算代价，因此大部分 CPIR 方案在设计时都尽量避免密文乘法。基于加法同态加密所构造的 CPIR 方案最新成果中的大部分方案都基于 Ring-LWE 困难问题假设和 BFV 加密。

3.1.1　基线 CPIR 协议

在 1998 年 Stern[13] 提出了支持多种加密体系的 1-out-of-t 的不经意传输方案，该方案是后来一系列基于加法同态加密方案构建的基础，也是其他 CPIR 方案进行性能比较的基线。基本思想是将客户端询问表现为 n 条同态加密密文，假设客户端索引为 i，则第 i 条密文为 $\mathrm{Enc}(s,1)$，其余为 $\mathrm{Enc}(s,0)$；而数据库中的 n 条数据则表示 n 条同态加密明文。每条询问和对应数据之间进行明文-密文乘法，得到的所有密文进行密文加法。

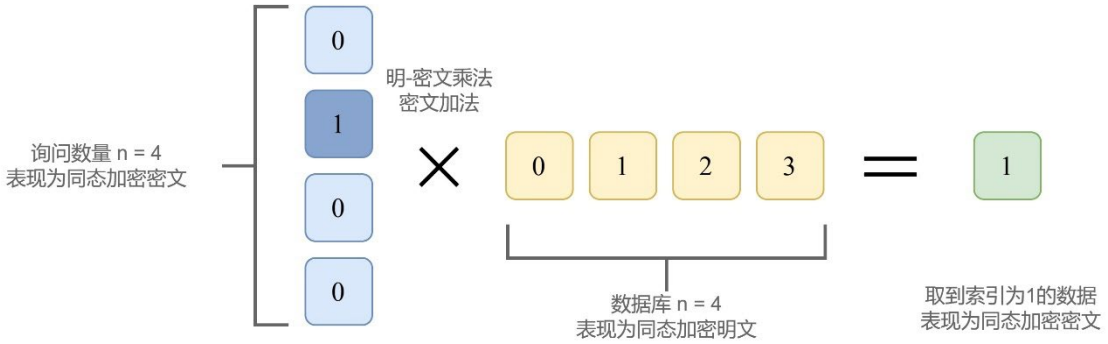

图 3.1 基线乘法图示

根据同态加密的性质，当密文加密的消息为 0 时有 $\mathrm{Absorb}(m,\mathrm{Enc}(s,0))=\mathrm{Enc}(s,m\times 0)=\mathrm{Enc}(s,0)$，则消息被抹去；反之若密文的加密值为 1 时有 $\mathrm{Absorb}(m,\mathrm{Enc}(s,1))=\mathrm{Enc}(s,m)$，则消息保留，因此客户端可以恢复原始消息。

算法 1：Baseline CPIR 协议

客户端 PIR.Query(i)：

1. **for** $i = 1$ to n **do**
2. **if** $(j \neq i)$： $q_j = \mathrm{Enc}(s, 0)$ ◁ 非目标索引为 0 的加密
3. **else**： $q_j = \mathrm{Enc}(s, 1)$ ◁ 目标索引为 1 的加密
4. **return** $q = \{q_0, q_1, \dots, q_n\}$

服务器 PIR.Response(D, q)：

➢ 假设密文可以吸收 l_0 个比特。

1. **for** $i = 1$ to n **do**
2. 将总长为 l 的消息 m_i 分为 l_0 长的块，记为 $m_{i,j}$, $j \in [1, \dots, \lceil l/l_0 \rceil)]$
3. **for** $j = 1$ to $\lceil l/l_0 \rceil$ **do**
4. **for** $i = 1$ to n **do**
5. $r_j = r_j + \mathrm{Absorb}(m_{i,j}, q_i)$
6. **return** $r = (r_1, \dots, r_{\lceil l/l_0 \rceil})$

客户端 PIR.Extract(r)

1. **for** $j = 1$ to $\lceil l/l_0 \rceil$ **do**
2. $p_j \leftarrow \mathrm{Dec}(s, j)$
3. $D_i \leftarrow (p_1, \cdots, p_{\lceil l/l_0 \rceil})$ ◁ 将解密结果进行顺序拼接

基线 CPIR 方案中，客户端生成的询问大小为 n 个同态加密的密文；服务器生成的回复则是 $\lceil l/l_0 \rceil$ 个同态加密的密文，其大小约为 $l \times F$。其中PIR.Response的步骤需要进行$n \times \lceil l/l_0 \rceil$ 次 Absorb 操作和 Sum 操作。

然而在基线方案中，客户端询问大小、服务器计算量和回复大小都和数据库成线性关系，这在实际使用中是不可接受的代价。接下来将叙述在基线方案上进行的一系列改进，从而使得计算量达到亚线性级别。

3.1.2 数据库递归表示

将数据库进行多维化递归表示是 CPIR 中普遍采取的优化方式，其主要目的在

于降低通信量。在算法 1 的行 1,2 中，实际上是将数据库和询问都表示成 n 长的向量，然后和同样为 n 长的询问进行向量内积预算得到询问结果。这样的数据库构造直接导致了询问的长度达到 $O(n)$ 级别，[13]中提出将数据库表示为 d 维矩阵，能够将客户端询问的通信量降低至 $O(\sqrt[d]{n})$。

具体来说，将数据库表示为 d 维矩阵后，其中的每个条目可以用 d 维坐标来表示。由于实践中性能表现最好的是二维数据库构造，因此此处以 $d=2$ 进行方案说明。将 D 表示成矩阵 $M_{k\times k}$，$k=\sqrt{n}$，则原先索引为 i 的条目可以用坐标 $m_{r,c}$ 来表示，其中 $r=\left\lfloor\frac{i}{\sqrt{n}}\right\rfloor$，$c=i-r\sqrt{n}$。而客户端发送两个询问向量 $q_r=(q_{r_1},\cdots,q_{r_k})$，$q_c=(q_{c_1},\cdots,q_{c_k})$。假设客户端询问条目为 $m_{a,b}$，则q_{r_a},q_{c_b} 为对 1 的随机加密，其余所有密文都为对 0 的随机加密。而服务器首先进行矩阵-向量乘法得到 n 长密文向量 $A_c=M\cdot q_r$，然后根据密文扩展因子将重新分为 F 个块，即$A_c=(A_{c,1},\cdots,A_{c,F})^T$，然后再次进行矩阵-向量乘法得到询问结果 $r=A_c^T\cdot q_c$。不失一般性地，这个方案可以很容易地通过递归的方式扩展到更高的维度。

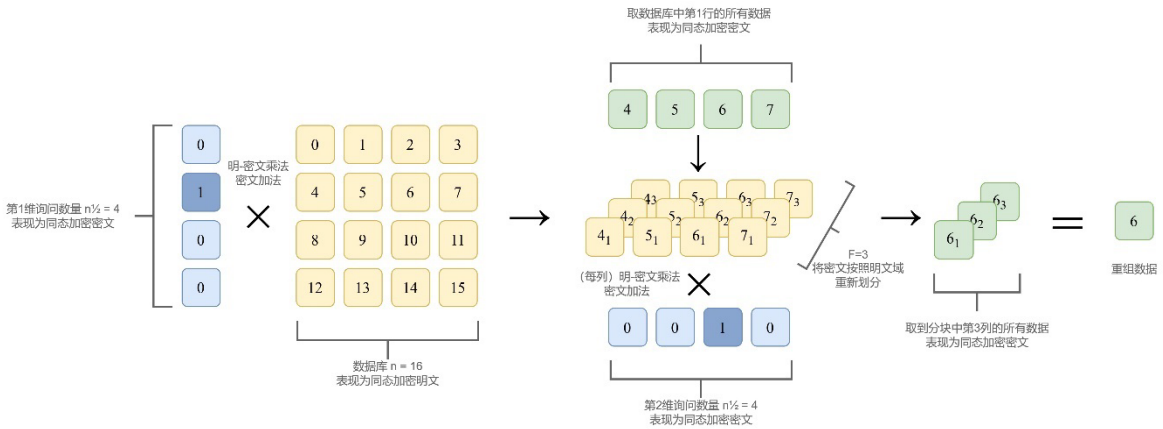

图 3.2 数据库递归表示图示

算法 2[①]：数据库递归表示 CPIR 协议(d = 2)

客户端 PIR.Query(i)：

1. $k \leftarrow \sqrt{n},\ a \leftarrow \lfloor i/k \rfloor,\ b \leftarrow i - a \cdot k$
2. **for** $i = 1$ to k **do**
3. **if** $(i \neq a)$: $q_{ri} \leftarrow Enc(s, 0)$ ◁ 非目标索引为 0 的加密
4. **else**: $q_{ri} \leftarrow Enc(s, 1)$ ◁ 目标索引为 1 的加密
5. **for** $j = 1$ to k **do**
6. **if** $(j \neq b)$: $q_{cj} \leftarrow Enc(s, 0)$ ◁ 非目标索引为 0 的加密
7. **else**: $q_{cj} \leftarrow Enc(s, 1)$ ◁ 目标索引为 1 的加密
8. **return** $q_r = \{q_{r1},\ q_{r2}, \dots, q_{rk}\},\ q_c = \{q_{c1},\ q_{c2}, \dots, q_{ck}\}$

服务器 PIR.Response(D, q)：

1. $d \leftarrow \sqrt{n},\ a \leftarrow \lfloor i/k \rfloor,\ b \leftarrow i - a \cdot k$
2. **for** $i = 1$ to d **do**
3. **for** $j = 1$ to d **do**
4. $A_{c_i} \leftarrow A_{c_i} + \text{Absorb}(m_{i,j}, q_{r_i})$
5. **for** $t = 1$ to F **do**
6. $A_{c_i} \rightarrow (A_{c_1,1}, \cdots, A_{c_1,F})$ ◁ 将密文分块
7. **for** $t = 1$ to F **do**
8. **for** $i = 1$ to d **do**
9. $r_t \leftarrow r_t + \text{Absorb}(A_{c_i,t}, q_{c_i})$
10. **return** $r = (r_1, \dots, r_F)$

客户端 PIR.Extract(r)：

1. **for** $t = 1$ to F **do**
2. $A'_{c_t} \leftarrow \text{Dec}(s, r_t)$
3. $A'_c \leftarrow (A'_{c_1}, \cdots, A'_{c_F})$ ◁ 将解密结果进行顺序拼接
4. **return** $\text{Dec}(s, A'_c)$

在算法 2 中的第一次矩阵-向量乘法后，所得到的结果 A_{c_i} 是同态加密的密文。由于密文乘法会带来很大的噪音和显著的长度增加，所以这些密文不能直接和数据库中的条目相乘。因此在行 14 中对密文按照 F 进行重新划分，并将划分后的密文当成新的明文对待，继续进行明文-密文乘法操作，从而能过**模拟同态乘法**。

[①] 为了可读性，算法 2 中假设每个密文都可以完全吸收一条消息。实际上和算法 1 类似，数据库库中的每条消息可能需要被划分到不同的密文中。

将数据库进行多维的表示，其优势在于降低了客户端的通信量：客户端发送 d 个询问向量，每个向量中包含 $\sqrt[d]{n}$ 个密文，复杂度从 $O(n)$ 降低到了 $O(d\sqrt[d]{n})$。而带来的缺点则是，在多维的情况下服务器需要递归地进行矩阵-向量乘法，回复通信量从原来的 $O(F)$ 密文量增长到 $O(F^{d-1})$。由于密文扩展因子 F 是比 n 小的多的常数，所以在 d 较小的时候通信量增长并不明显，因此 d 通常选取 2 或 3。

3.1.3 询问压缩和不经意扩展

即使是在进行了数据库多维表示的情况下，通信代价依旧非常高昂。[5]提出了一种将压缩询问的优化。该技术使得客户端能够发送**单个密文**，然后由客户端对其进行不经意扩展，恢复到 n 个密文（算法 1）或 $d\sqrt[d]{n}$ 个密文（算法 2）。

询问压缩 在之前的方案中客户端询问是对 1 或 0 的加密。而在询问压缩的方案中则更改为加密一个单项式，该单项式的次数为客户端期望索引 i，即 $q = \mathrm{Enc}(s, x^i), x^i \in R_t$。此处的限制是索引 i 必须小于多项式度数 N，后续会加以改进。

不经意扩展 为了进行不经意扩展，首先需要引入 BFV 密码体系支持的另一种同态操作

5. BFV 密文替换（Sub）：已有 $c = \mathrm{Enc}(s, p(x))$，其中$p(x) \in R/tR$，对于一个奇数 k，密文替换操作为 $\mathrm{Sub}(c, k) = \mathrm{Enc}(s, p(x^k))$。

密文替换操作是基于分圆环的自同构。在[14]中证明了当选取了第 i 个分圆多项式 $\phi_i = x^N + 1$ 时，如果 k 为和 i 互质的奇数，那么直接将密文多项式 $(c_0, c_1) = \mathrm{Enc}(s, p(x))$ 中的每个 x 替换为 x^k，并记结果为 (c_0^k, c_1^k)，则有 $(c_0^k, c_1^k) = \mathrm{Enc}(s^k, p(x^k))$。换言之，使用 s^k 来解密进行 Sub 后的密文，就能够得到预期的明文$p(x^k)$。

不经意扩展从密文环 R_q 上的一个元素 $c = x^i \in R_q (i \in [N])$ 开始，进行 $\log N$ 次迭代。每次迭代都将环上元素数量翻倍，同时保证在所有的元素中只有一

个是非 0 元素。

算法 3：EXPAND 算法

服务器 PIR.Expand($c = \mathrm{Enc}(x^i)$)： ◁ $i \in \{0, \dots, 2^l - 1\}$

1. list $= [c]$
2. **for** $j = 0$ to $l - 1$ **do** ◁ 每次迭代都会将 list 长度翻倍
3. $t = N/2^j + 1$
4. **for** $k = 0$ to $2^j - 1$ **do**
5. $c_0 \leftarrow \mathrm{list}[k]$
6. $c_1 \leftarrow c_0 \cdot x^{-2^j}$
7. $\mathrm{list}[k] \leftarrow c_0 + \mathrm{Sub}(c_0, t)$
8. $\mathrm{list}[k + 2^j] \leftarrow c_1 + \mathrm{Sub}(c_1, t)$
9. **return** list

密钥转换 注意到上面行 7, 8 中，进行了替换前后的密文被直接进行相加。根据以上对 Sub 操作的讨论，这两个密文的加密密钥分别为 s 和 s^t。因此这一步的相加还需要密钥转换的辅助。为了解决这个问题，需要引入[15]中的密钥转换操作。

3.2 基于全同态加密的 CPIR 方案

为了避免同态乘法指数级别增长的噪音，基于加法同态的方案通过将密文划分到明文域上的方式来模拟同态乘法，从而较好地控制了噪音。而这样做的代价就是服务器的回复通信量随着维数 d 呈**指数型增长**。

而在 GSW 方案中，同态乘法噪音的增长是不对称的。在对一个全新的密文做 k 次乘法的时候，噪音的增长是线性的。然而 GSW 方案每次加密一个标量都需要一个大矩阵，这造成了一个远大于 Regev/BFV 方案的密文扩展因子，从而导致了总体通信量的增加。

从这两方面来看，Regev/BFV 和 GSW 分别在密文口扩张因子、噪音控制有

着优势，如果能够综合二者的优势，就能够让同态乘法参与进 CPIR 方案的构建。本小节将详细叙述如何结合 Regev/BFV 和 GSW 方案，并在其基础上进行进一步优化。

3.2.1 使用外积操作打包密文

在[16]中提出了**外积（external product）**操作，能够在 Regev 密文和 GSW 密文之间进行乘法操作。即

$$\mathrm{Ext}(c_{Regev}, c_{GSW}) = \mathrm{Enc}_{Regev}(s, m_1 \cdot m_2)$$

[8]中提出了矩阵 Regev 的变种，使得对明文 $M \in R_q^{n\times n}$ 有 $S^T C_{Regev} = M + E_{Regev}$。而 GSW 对于某个比特 $\mu = \{0,1\}$ 则有 $S^T C_{GSW} = \mu S^T G + E_{GSW}$。如果两者采用相同的密钥，那么外积操作就是计算 $S^T C_{GSW} G^{-1}(C_{Regev}) \approx \mu S^T C_{Regev} \approx \mu M$，即 Regev 密文和 GSW 密文的相乘结果。如果 GSW 的密文加密的总是$\{0,1\}$，那么外积操作所带来噪音是**加法线性**的，因此能够将两方案的优势进行结合。

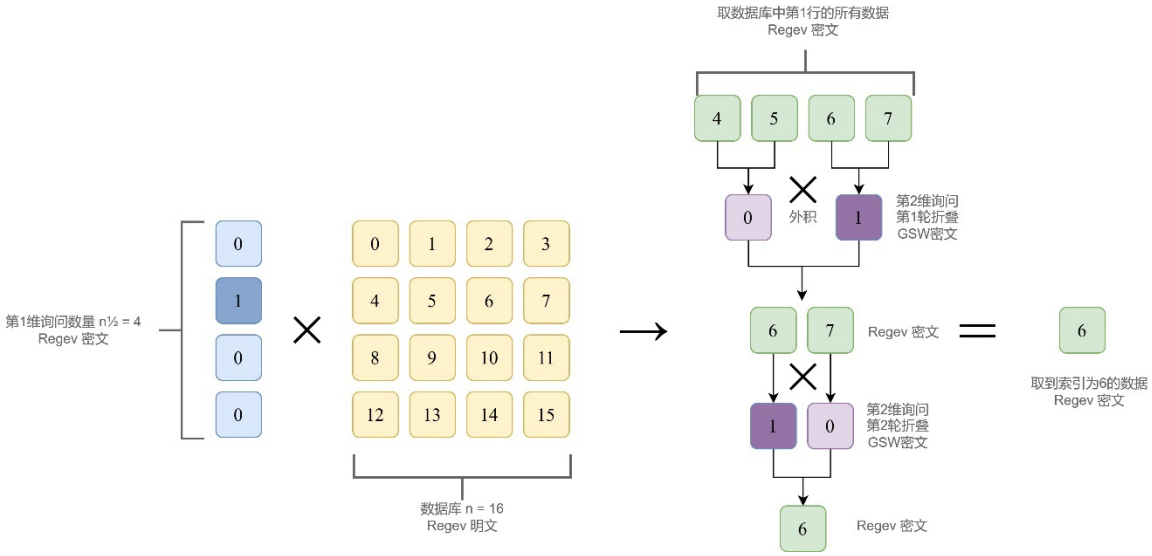

图 3.3 全同态方案图示

在大部分全同态 PIR 方案[8][9]中都采用了外积的操作，[8]中提出的方案为：将数据库使用两维表示，在客户端询问中，第一维询问使用矩阵 Regev 密文，第

二维则使用 GSW 密文。和加法同态 PIR 方案中类似的，进行第一维明文-密文乘法之后，得到临时 Regev 密文。接着在第二维的选择中，将临时 Regev 密文和一个加密了 0 或 1 的 GSW 密文进行外积，通过递归的方式选择到想要的条目。

这个方案的优势为，第一维的操作（Regev 标量乘）运算快，第二维使用新鲜的 GSW 密文控制了噪音，同时最终得到的是密文扩张因子较小的 Regev 明文。而其缺点是一个 GSW 的密文大小接近 4MB，这使得客户端通信代价变得很大。

3.2.2 GSW 密文压缩为 Regev 密文

和基于加法同态的 PIR 方案相比，[8]中客户端通信代价的劣势主要来源于没有进行询问压缩。回忆在 3.1.3 节中，询问压缩使用的是同态自同构的性质。Regev 类型的密码体系能够使用密钥转换操作来支持高效的密文替换操作，而 GSW 则没有这样的性质。

为了改进客户端通信量，[9]中所提出的方案则在第一维中使用 BFV 密文，第二维中使用 RGSW 密文，其基本架构和[8]类似，并提出了一种将所有询问向量压缩进单个密文中的操作。

算法 4：PACK/UNPACK 算法

客户端 PIR.Pack($q = \{q_1, \cdots, q_d\}$)：　◁ d 维明文询问向量

1. $x \leftarrow 0, p \leftarrow []$
2. **for** $i = 1$ to d **do**　◁ 对于每一维 i
3. **for** $j = 1$ to N_i **do**　◁ 对于第 i 维的每个元素
4. **for** $k = 1$ to f **do**　◁ 当 $i = 1$ 时 $f = 2$；其余$f = l$
5. $p[x] \leftarrow q_{i,j}[k]$　◁ $q_{i,j}$ 代表第 i 个向量中的第 j 个比特，[k]代表明文
6. $x \leftarrow x + 1$
7. **return** $\mathrm{Enc}_{BFV}(p)$

服务器 PIR.Unpack(c)：

1. **for** $j = 0$ to $N_1 - 1$ **do** ◁ 展开第一维向量
2. $C_{BFV_j}[0] = c[2j+1], C_{BFV_j}[1] = c[2j+2]$ ◁ 单个 BFV 密文
3. $x \leftarrow 2N_1$
4. **for** $i = 1$ to $d - 1$ **do**
5. **for** $j = 0$ to $N_{i+1} - 1$ **do**
6. **for** $k = 1$ to l **do**
7. $C_{RGSW}{}_j^i[k+l] \leftarrow c[x + jl + k]$ ◁ C_{RGSW}^i代表第 i 维询问
8. $C_{RGSW}{}_j^i[k] \leftarrow \mathrm{Ext}(A, c[x + jl + k])$ ◁外积，$A = \mathrm{Enc}_{RGWS}(-s)$
9. $x \leftarrow x + lN_i$
10. **return** $\left(C_{BFV}, \{C_{RGSW}^i\}_{i=1}^{d-1}\right)$

这个方案的优势在于能够将所有询问向量打包进一个密文中，大大降低了客户端询问的代价。然而注意到在这个算法中，扩展得到的密文实际上都是单个比特，也就是一个标量。而在 3.2.1 中的方案所使用的则是矩阵 Regev 密文，因此这使得密文扩张因子不能得到最好的表现。

在[17]中对这个方案进行了进一步拓展，提出了从标量 Regev 密文转换到矩阵 Regev 密文的方法，并且对服务器回复进行进一步压缩，达到了目前全同态 PIR 方案中最好的噪音控制和密文扩张因子。

3.3 亚线性时间 CPIR 设计

在 3.1、3.2 节中所叙述的方案通过不同手段来改进了客户端和服务器方的通信量。然而，这些 CPIR 方案却都有着昂贵的计算代价。直觉性的，如果想要对服务器隐藏客户端的索引，那么服务器在计算中必须扫描数据库中中的每个比特，否则就会泄露哪些记录是和客户端索引无关的，从而违背 PIR 的安全性目标。在[13]中提出，服务器的计算量的理论下界为 $\Omega(n)$，其中 n 为数据库大小。换言之，对于客户端的每个请求，服务器都需要**线性时间**来进行恢复。

为了使得 CPIR 方案实际可用，一些新的研究开始关注如何使用辅助手段降低

服务器计算量和回复时间。当前能够达到**亚线性时间**的方案主要分为批处理 PIR（Batch PIR）、预处理 PIR（PIR with preprocessing）和有状态 PIR（Stateful PIR）。在本节中将对这些方案的基本思想、现行研究进行叙述和优缺点分析。

3.3.1 Batch PIR

Batch PIR 的主要思想是：如果一个客户端预先知道自己有 k 个询问，那么服务器可以进行批处理，而不是运行 k 个独立的查询。对于足够大的 k，摊销时间能达到亚线性复杂度 $\tilde{O}\left(\frac{n}{k}\right)$。

Batch PIR 主要利用了 Batch Code[18]的思想。假设数据库中有 n 个元素，这个数据库被分布在 b 个设备上，一个用户从所有元素（可以是任何设备上的）取 k 个元素。一个 (n, m, k, b) 的 Batch Code 将这 n 个元素分布到 b 个桶中，也就是$\mathcal{B}$: $DB \rightarrow (C_0, \dots, C_{b-1})$，码字总长度将大于原来的元素个数 $m = \sum_{i=0}^{b-1}|C_i| \geq n$。这些码字有两个目标：（1）通过从这 b 个桶中分别取出至多 1 个码字，能够保证能够获取数据库中任意 k 个元素。（2）保证总的码字数量 $m \leq k \cdot n$。符合条件的 Batch Code 就可以用来均摊计算开销，方法是在 b 个桶中独立运行某个 CPIR 协议。假设在 n 个元素上运行某个 CPIR 协议的复杂度为 $\mathcal{T}(n)$，那么平凡运行 k 次的代价为 $k \cdot \mathcal{T}(n)$；使用 Batch Code 的复杂度则为 $\sum_{i=0}^{b-1}\mathcal{T}(|C_i|)$，而 CPIR 协议中随着数据库规模增长，时间复杂度至少呈现线性增加，因此有 $\sum_{i=0}^{b-1}\mathcal{T}(|C_i|) \leq \mathcal{T}(m) \leq k \cdot \mathcal{T}(n)$。

根据以上原理，Batch Code 的构造很大程度上决定了分摊的性能。在[5]中给出了一种基于布谷鸟哈希（Cuckoo Hash）的 Batch Code。

（1） 服务器运行一次性设置阶段：首先挑选 w 个哈希函数 $\{h_1, \dots, h_w\}$，并创造 b 个桶。对于数据库 $DB = \{D_1, \dots, D_n\}$ 中的每个元素并计算 $h_1(D_i), \dots, h_w(D_i)$，根据哈希值将 D_i 复制到对应的桶中（选取的哈希函数需要确保这 w 个哈希值对应的桶各不相同）。最终结果为分布在 b 个桶中的 wn 个元素。

（2） 客户端生成 k 个询问时，计算其中每个索引 i 对应的候选桶 $h_1(i), \dots, h_w(i)$，选择其中的空桶进行插入。如果没有空桶则按照布谷鸟哈希的

方式，随机插入其中的一个候选桶并对原先桶中的元素进行重新插入。所有元素插入结束后，对剩余空桶赋虚拟值（通常时 0），最终得到 b 个索引$\{q_1,\dots,q_b\}$。

（3） 服务器对第 i 个桶运行 CPIR 协议，检索 q_i。值得注意的是在步骤（1）中对每个元素计算对应桶时，对于冲突元素没有踢出原桶的操作，换言之索引 q_i 存在于任何一个候选桶中。因此 CPIR 检索必然成功。

该方案的优势为均摊了计算开销：对于单个询问，原有的计算代价 $\tilde{O}(n)$ 变为分摊代价 $\tilde{O}(\frac{wn}{b})$，在选取 $w=3,b=1.5k$ 参数的情况下该代价为$\tilde{O}(\frac{2n}{k})$。该方案的缺点则是更大的服务器存储量（$n\to wn$），以及对应更高的通信代价($k\to b$)。由于 Batch Code 的方法是独立于 CPIR 本身的构造以外的，几乎和任何方案相容，因此被后来的多个方案用于改进通信量。

3.3.2 预处理 PIR

预处理 PIR 中，服务器在预处理阶段对数据库进行编码，计算出 r 比特的公共线索，并使用这些线索来更快的回复客户端请求。[19]中证明批处理 PIR 的下界为 $tr=\Omega(n\log n)$。

预处理的概念最先是在[3]中被提出的，它基于多服务器的 IT-PIR 方案，使用 $O(n^{1+\epsilon})$ 个额外比特，将在线阶段计算量降低到 $O(\frac{n}{\epsilon^2\log^2 n})$。[20]首次关注了如何将 IT-PIR 技术和 CPIR 技术混合，得到亚线性复杂度的方案。在 [21]中通过不经意本地可解码码字（Oblivious locally decodable code，OLDC）给出了一个单服务器的预处理 PIR 模型，同样对数据库做了某种形式上的编码处理。其主要思想是通过码字来混淆索引信息的前提下，对码字集提前运行不可避免的线性计算，并沿途生成辅助数据或提示。预处理 PIR 的优势在于降低了在线阶段的计算复杂度，而其缺点则是增加了服务器端的存储量。

除了通过编码的方式，一些 CPIR 方案中还更为针对性的提出了存储某些预计算值使得计算更快速的方法，这并不属于预处理 PIR 的范畴，但是其思想有相似之处（同样是利用服务器存储公共辅助信息）。例如[4]中提出了对于 64×64 比

特的乘法，可用通过预计算牛顿系数来避免昂贵的整数除法。[5]中为了能够进行 Sub 操作，需要在服务器端针对每个用户存储各自的密钥转换矩阵信息。这些辅助信息虽然不能够降低整体的计算复杂度，但是却能够极大的降低计算时间，因此这也是工程实现中不可忽视的一部分。

3.3.3 有状态 PIR

有状态 PIR 又叫做离线/在线 PIR（offline/online PIR），在这种设置中客户端是带状态的，每个客户端将独立存储 r 比特的线索。[22]中证明上界为 $tr = O(n)$，如果使用 $\tilde{O}(\sqrt{n})$ 比特的客户端线索，那么在线计算时间能达到 $t = \tilde{O}(\sqrt{n})$。

在[22]中从 IT-PIR 的角度出发，提出了利用可穿刺伪随机集合（Puncturable Pseudorandom Sets, PRS）构造的一个通用的 2 服务器有状态 PIR 模型，然后使用加法同态或全同态的手段转换为单服务器的方案。随后[22][23][24]对其进行了进一步优化和应用拓展。

在该方案中，数据库中的每条数据都是 1 比特的，表示为 $X = \{x_1, \cdots, x_n\}, x_i \in \{0,1\}$。离线步骤分为两步：（1）客户端首先随机选择 m 个集合 $S_1, \ldots, S_m \in [n]$，每个集合的大小 $|S_j| = \sqrt{n}$，并将其发送给服务器 1。（2）服务器 1 对于每个集合计算对应数据集的奇偶性 $h_j = \sum_{l \in S_j} x_l \bmod 2$，发送给客户端。客户存储所有的 S_j, h_j 作为线索。在线步骤分为三步：（3）对于期望索引 $i \in S_j$，客户端发送集合中除了 i 的所有元素给服务器 2，即 $S' = S_j \backslash \{i\}$ 给服务器 2，集合 S' 被称为可穿刺伪随机集合。（4）服务器 2 和离线步骤（2）中相似地计算奇偶性 $a = \sum_{l \in S'} x_l \bmod 2$，并发送给客户端。（5）客户端简单计算异或和得到 $x_i = (h_j + a) \bmod 2$。

在这个方案中，两服务器不能是共谋的。如果服务器 2 知道了集合 S_j 的信息，很容易就能破解用户的期望索引 i。而将其转换为单服务器方案的方式是，使用加法同态加密或全同态加密来运算离线阶段。类似之前章节，该方案将数据库 X 表

示为 $\sqrt{n}\times\sqrt{n}$的矩阵。客户端将选择的集合 S_j 表现成向量 $v_j\in R^{\sqrt{n}}$，发送 $c_j=\mathrm{Enc}(s,v_j)$ 给服务器，服务器计算向量-矩阵乘法 $c_j{}^T\cdot X=\mathrm{Enc}(s,v_j^T)\cdot X=\mathrm{Enc}(s,v_j^T\cdot X)=\mathrm{Enc}(s,h_j)$。客户端解密后就得到和 2 服务器方案中相同线索 h_j，同时在该过程中没有向服务器泄露用户选择的集合信息，因此在线阶段可以沿用之前的步骤。如果使用全同态加密，离线阶段的通信量为 $2\sqrt{n}$ 个同态密文，计算量为 n 次明-密文乘法和 n 次密文加法；而在线阶段通信量为 $2\sqrt{n}$ 个比特，计算量仅 n 次比特操作。

该方案的优势在于，近乎所有的计算代价都被挪到了离线阶段，使得服务器在线回复时间能够达到 IT-PIR 方案的高效。而其缺点在于**客户端高计算代价**，通过使用安全系数为 λ 的随机种子伪随机动态生成 $\sqrt{n}$ 个集合，客户端的存储量为随机种子和服务器回复的 $\sqrt{n}$ 比特。然而这就需要客户端花费大量时间重构集合，并寻找 i 所在的集合 S_j，客户端计算代价达到 $\tilde{O}(n^{5/6})$。

3.4 CPIR 方案计算及通信代价分析

在本小节中将总结归纳和比较以上提到的所有 CPIR 方案的构造，主要分为无状态和有状态两个部分进行比较。

设数据库中共有 N 个条目，每个条目的长度为 w 比特，被表示为 d 维均匀矩阵。所采用的同态加密方案的密文扩张因子为 F，明文空间为 $\mathbb{Z}_p$，密文空间为 $\mathbb{Z}_q$，维度为 n，分别用 $\mathcal{A},\mathcal{P},\mathcal{M},\mathcal{S},\mathcal{X}$ 来代表一次同态加法/明-密文乘法/密文乘法/替换/外积操作的代价，$\mathcal{E}$ 则代表密文的大小。以下表 3-1 中我们给出了目前研究中基于加法同态方案中的具体计算、通信代价。

表 3-1 无状态 PIR 方案对比

方案基础	方案名称	计算代价		通信代价	
		$1\le d\le\frac{\log n}{\log F}$	$\frac{\log n}{\log F}\le d\le\log n$	询问	回复

	下载数据库	-	$O(N)$	$O(1)$	$O(N)$
加法同态	基线 PIR[2]	$N(\mathcal{A}+\mathcal{P})$	$N(\mathcal{A}+\mathcal{P})$	$N\mathcal{E}$	$N\mathcal{E}$
	XPIR[4]	$N(\mathcal{A}+\mathcal{P})$	$\sqrt[d]{N}F^{d-1}(\mathcal{A}+\mathcal{P})$	$d\sqrt[d]{N}\mathcal{E}$	$F^{d-1}\mathcal{E}$
	SealPIR[5]	$N(\mathcal{A}+\mathcal{P}+2\mathcal{S})$	$d\sqrt[d]{N}\cdot\mathcal{S}+\sqrt[d]{N}F^{d-1}(\mathcal{A}+\mathcal{P})$	$d\lceil\sqrt[d]{N}/n\rceil\mathcal{E}$	$F^{d-1}\mathcal{E}$
全同态	MulPIR[7]	$N(\mathcal{A}+\mathcal{P})+N^{\frac{d-1}{d}}\mathcal{M}$	$N(\mathcal{A}+\mathcal{P})+N^{\frac{d-1}{d}}\mathcal{M}$	$\mathcal{E}_B$	$\mathcal{E}_B$
	OnionPIR[9]	$N_1(\mathcal{A}+\mathcal{P})$	$\frac{N\mathcal{X}}{N_1}+\frac{N\mathcal{X}}{4N_1}+\cdots$	$\mathcal{E}_B$	$\mathcal{E}_B$
	Spiral[17]	$N_1(\mathcal{A}+\mathcal{P})$	$\frac{N\mathcal{X}}{N_1}+\frac{N\mathcal{X}}{4N_1}+\cdots$	$\mathcal{E}_R$	$\mathcal{E}_R$

总体来看，基于加法同态和基于全同态的方案都没有越过计算量 $\Omega(N)$ 的下界，但是基于加法同态的方案不需要进行同态乘法，相对而言**计算量代价更小**。然而模拟加法的操作需要进行密文分块，因此基于加法同态的方案的通信量通常受到密文扩展因子 F 的影响；与之相比全同态方案的**通信量更小**。

而有状态方案的具体设计较少，大部分论文中只给出了设计方向和理论上下限分析，在表 3-2 中给出了这几种方案的对比。有状态方案都能够达到亚线性的复杂度，而与之相对的是需要更大的存储空间。

表 3-2 有状态 PIR 方案对比

方案名称	预处理方式	额外存储量	计算量
Batch PIR	分块	w 倍（哈希函数个数）	$O(N/k)$
预处理 PIR	OLDC	r 比特服务器线索	$O(N\log N)/r$
有状态 PIR	PRS	r 比特客户端线索	$O(N)/r$

第 4 章　基于 Regev 加密的预处理 CPIR 方案设计

根据第 3 章中对不同 CPIR 方案思想的叙述、对比和总结，可以看出传统无状态 CPIR 方案主要的瓶颈集中在计算代价上，方案实用价值不高；而预处理、有状态等 CPIR 虽发展时间较短，但是通过将计算代价集中到一次性离线阶段中，有效降低了服务器回复时间，使得这些方案在现实世界中可用。

在此章节中将给出一个基于 Regev 加密方案的预处理 CPIR 方案设计和 C++ 工程实现，该方案主要关注点在于当前方案中的存在的几个问题：

- **预处理代价昂贵**：毋庸置疑地，在预处理阶段的计算依旧需要进行 $O(N)$ 复杂度的操作。但是在不同加密体系和不同预处理设置下，操作的代价差异很大。例如基于 GSW 加密的同态操作要显著慢于 Regev 加密体系。
- **存储量权衡**：预处理 PIR 和有状态 PIR 需要利用客户端和服务器的额外存储空间来换取更好的时间代价，然而这些存储量通常非常巨大（通常和数据库大小呈亚线性相关）。客户端存储量过大的方案实用意义通常不强， 因此如何权衡存储量、通信量和计算量也是目前研究的重要关注点。
- **缺乏高效工程实现**：大部分有状态 PIR 的设计[22] [23] [24]都只有理论实现，缺乏实际可用的代码、工程实现细节。而对于理论上接近最优的方案，还有向量化、矩阵快速乘等工程上能实现能够提升实际吞吐带宽。

4.1　方案设计

本方案的主要构建思想基于[10]中 7.8 节里提出的对 Regev 公钥加密系统的高效改进，其安全性依赖于 LWE 困难问题和该文章中对于重复使用矩阵 A 的安全性证明。最新文献[25][26]基于该思想分别进行了 CPIR 方案的构建，本文将延续他们所给出的构造做进一步改进。**图 4.1** 给出了该方案的整体架构，总共分为 5 个步骤，其中步骤 1、2 是服务器/客户端一次性预处理的过程，而 3、4、5 则是客户端每次发起询问时运行的在线步骤。在方案中采用上文中高维矩阵表现数据库的形式，图中流程给出了将数据库表现为 2 维时，第 1 维询问运行的全部流程。

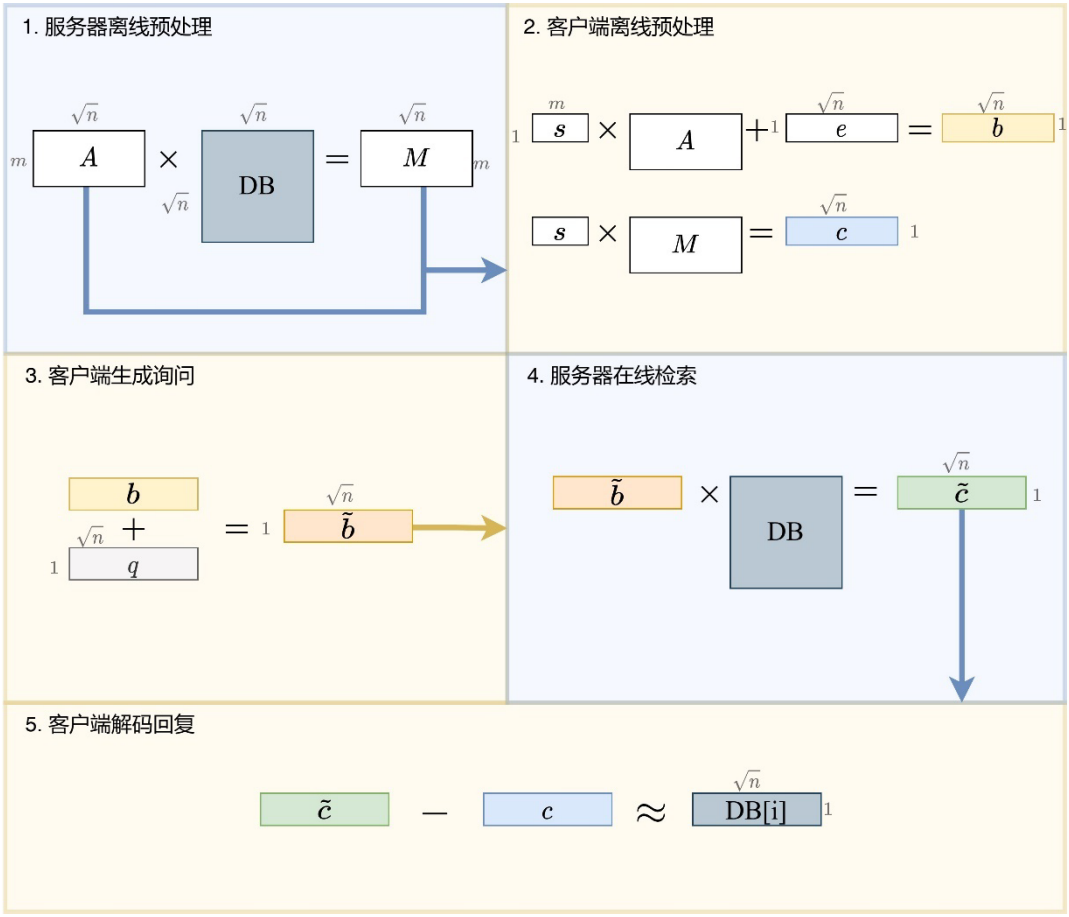

图 4.1 方案整体架构

4.1.1 预处理步骤

服务器持有 n 长的数据库 $DB = \{D_1, \dots, D_N\}$，其中每个条目的长度为 w。假设服务器设定的 Regev 初始化参数如下：维度为 n，模数为 q，分布为 $\mathcal{X}$，明文模数 $p \ll q$。服务器首先将数据库表现为 $\sqrt{N} \times \sqrt{N}$ 的矩阵，即 $\boldsymbol{DB} \in \mathbb{Z}_p^{\sqrt{N} \times \sqrt{N}}$。以下将从第 1 维询问的角度，对预处理进行详细叙述。

在基于 Regev 构建的基础 PIR 协议中，客户端发送的询问为一条 Regev 密文，其加密对象为 $\sqrt{N}$ 长向量 $\boldsymbol{m} \in \mathbb{Z}_p^{\sqrt{N}}$。加密过程如下：随机选取矩阵 $\boldsymbol{A} \leftarrow \mathbb{Z}_q^{\sqrt{N} \times n}$，私钥 $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$，错误向量 $\boldsymbol{e} \leftarrow \mathcal{X}^{\sqrt{N}}$。根据以下公式计算客户端询问 $\boldsymbol{Q} \leftarrow \mathrm{Enc}_{Regev}(m) = (\boldsymbol{A}, \boldsymbol{c}) = \left(\boldsymbol{A}, \boldsymbol{As} + \boldsymbol{e} + \left\lfloor \frac{q}{p} \right\rfloor \cdot \boldsymbol{m}\right) \in \mathbb{Z}_p^{\sqrt{N} \times (n+1)}$。而服务器收到询问之

后计算回复 $\boldsymbol{R} \leftarrow \boldsymbol{DB} \cdot \boldsymbol{Q} = (\boldsymbol{DB} \cdot \boldsymbol{A}, \boldsymbol{DB} \cdot \boldsymbol{c}) \in \mathbb{Z}_p^{\sqrt{N} \times (n+1)}$。客户端通过 Regev 解密来得到数据库中的一行。

基于以上公式中有如下几个重要观察：

- 客户端询问分为两个部分 $(\boldsymbol{A}, \boldsymbol{c})$，而服务器对这两部分进行的矩阵乘法是**独立**的。在没有收到 $\boldsymbol{c}$ 的情况下，服务器能够得到部分结果 $\boldsymbol{DB} \cdot \boldsymbol{A}$。
- 这两部分中 $\boldsymbol{A}$ 为 $\sqrt{N} \times n$ 矩阵， $\boldsymbol{c}$ 为 $\sqrt{N}$ 长向量，服务器计算量主要集中在前者中。
- 根据[10]中提出的结论，如果每次使用的私钥 $\boldsymbol{s}$ 能够使得 $\boldsymbol{As} + \boldsymbol{e}$ 中有 $l = O(n)$ 行和其他询问中的不同，那么可以安全地重复使用矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{e}$。而在 PIR 方案中，客户端每次询问都选择不同的私钥是一种常见设定。

具体到第 1 维下，如果让服务器来选取公共 $\boldsymbol{A_1}$ 并预计算相关部分，得到某个公共线索；而客户端在预处理阶段对其进行下载和存储，就能够将大部分的计算开销挪移到离线部分。在**图 4.2** 中，给出了对以上观察的说明，其中 $\boldsymbol{A_1}$ 和 $\boldsymbol{MR} \leftarrow \boldsymbol{DB} \cdot \boldsymbol{A_1}$ 为服务器公共线索，在预处理阶段被计算，并由每个客户端独立下载并进行存储。在进行询问时，客户端可以独立进行计算并得到 q_1，并由服务器计算回复 r_i。

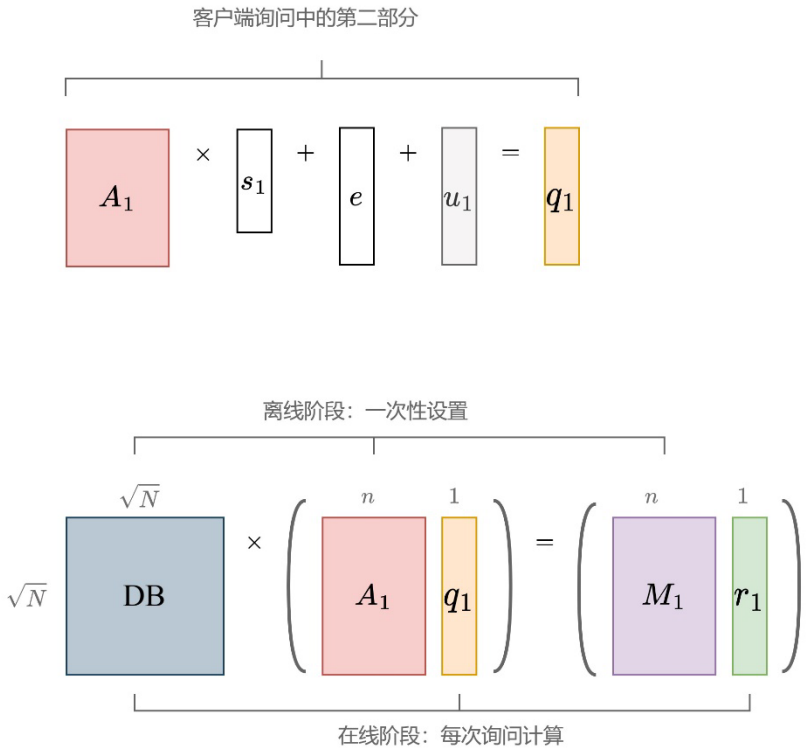

图 4.2 预处理步骤图示

进行预处理步骤后，在离线阶段中服务器的计算量为 Nn 次 $\mathbb{Z}_q$ 上乘法和 Nn 次加法；而在线阶段则只需要 N 次 $\mathbb{Z}_q$ 上乘法和 N 次加法。因此占比为 $\frac{n}{n+1}$ 的计算量发生在离线阶段，这将大大加快服务器回复速度。

而对于通信量来说，在预处理阶段需要发送矩阵 $\boldsymbol{A_1}, \boldsymbol{M_1}$，大小为 $2n\sqrt{N}$。如果使用 λ 长的伪随机种子来生成矩阵 $\boldsymbol{A_1}$，那么就能够将通信量进一步缩小到 $\lambda + n\sqrt{N}$，同时依旧 λ 比特的安全性。只要 λ 选取合理，安全性上的损失是可以忽略的。在通信复杂量上来说不仅没有显著提升，而且把原方案中需要每次询问的通信量转移到了一次性预处理阶段，使得在线阶段的通信量从原先的 $2\sqrt{N}(n+1)$ 降低到了 $2\sqrt{N}$。

4.1.2 基于加法同态的高维处理

在上一小节中以简化的第 1 维询问来介绍了预处理的想法，服务器中的回复

实际上是数据库中的一行。而对于更高维度的处理，在第 3 章中已经给出了处理的基本思想——基于加法同态和基于全同态。

首先从加法同态的角度出发对高维进行处理。类似 3.1.2 节中介绍的，服务器在进行第 1 维矩阵乘法之后，得到的向量 $\boldsymbol{r}_1$ 是一个 $\mathbb{Z}_q^{\sqrt{N}}$ 上的元素。为了再次和第 2 维的询问做乘法，需要将 $\boldsymbol{r}_1$ 根据密文扩张因子 F 进行划分，形成一个新的矩阵 $\boldsymbol{R}_1 \in \mathbb{Z}_q^{\sqrt{N}\times F}$。假设第 2 维的询问是正常 Regev 密文，那么最终得到的回复将是一个 $(n+1)\times F$ 的矩阵。

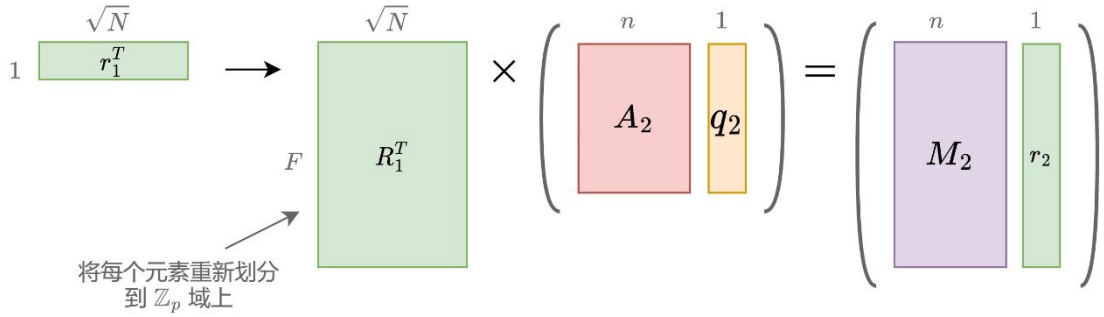

图 4.3 第 2 维询问中矩阵分解

分析用户对 $\boldsymbol{r}_2$ 进行解密的正确性，首先通过服务器回复中的 $\boldsymbol{M}_2$ 直接进行 Regev 解密，得到$\boldsymbol{r'}_2^T \leftarrow \left\lfloor \frac{p}{q} \right\rceil (\boldsymbol{r}_2 - \boldsymbol{M}_2 \cdot \boldsymbol{s}_2) \in \mathbb{Z}_p^F$。将其重组到 $\mathbb{Z}_q$ 上，能够得到单个值 $r'_2 \in \mathbb{Z}_q$ 。进一步根据 $\boldsymbol{MR}$ 中的第 i 行计算第 1 维解密，得到 $\boldsymbol{D}_{i,j} \leftarrow \left\lfloor \frac{p}{q} \right\rceil (r'_i - \boldsymbol{M}_1[i] \cdot \boldsymbol{s}_1[i]) \in \mathbb{Z}_p$。算法 6 中给出了基于加法同态的全部流程：

算法 6：基于加法同态下算法全过程

服务器 $\text{PIR.Prep}(n, q, p, \lambda, N, F, \boldsymbol{DB} \in \mathbb{Z}_p^{\sqrt{n}\times\sqrt{n}})$：　◁ 各参数和上文含义相同

1. $\boldsymbol{A}_1 \in \mathbb{Z}_q^{\sqrt{n}\times n} \leftarrow \text{PRG}(Seed_1^\lambda)$
2. $\boldsymbol{A}_2 \in \mathbb{Z}_q^{\sqrt{n}\times n} \leftarrow \text{PRG}(Seed_2^\lambda)$
3. $\boldsymbol{M}_1 \in \mathbb{Z}_q^{\sqrt{n}\times n} \leftarrow \boldsymbol{DB} \cdot \boldsymbol{AR}$

---

4. **return** $(Seed_1, Seed_2, \boldsymbol{M}_1)$

客户端 PIR.Quary($x \in [N]$)：◁ 询问索引为 x 的密文

1. $r_1 \leftarrow \lfloor x/\sqrt{n} \rfloor, r_2 \leftarrow x - r_1$ ◁ 进行维度分解
2. 取样 $\begin{cases} \boldsymbol{s_1}, \boldsymbol{s_2} \leftarrow \mathbb{Z}_q^n \\ \boldsymbol{e_1}, \boldsymbol{e_2} \leftarrow \chi^{\sqrt{n}} \end{cases}$
3. $\boldsymbol{A}_1 \in \mathbb{Z}_q^{\sqrt{n} \times n} \leftarrow \mathrm{PRG}(Seed_1^{\lambda}), \boldsymbol{A}_2 \in \mathbb{Z}_q^{\sqrt{n} \times n} \leftarrow \mathrm{PRG}(Seed_2^{\lambda})$
4. $\boldsymbol{q}_1 \leftarrow \boldsymbol{A}_1 \cdot \boldsymbol{s}_1 + \boldsymbol{e}_1 + \Delta \mathbf{u}_{\mathbf{r}_1}$ ◁ u_{r_1}为 r_1 处为 1，其余为 0 的向量。 $\Delta = \lfloor q/p \rfloor$
5. $\boldsymbol{q}_2 \leftarrow \boldsymbol{A}_2 \cdot \boldsymbol{s}_2 + \boldsymbol{e}_2 + \Delta \mathbf{u}_{\mathbf{r_2}}$
6. **return** $(\boldsymbol{q}_1, \boldsymbol{q}_2)$

服务器 PIR.Response($\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{DB}$)

1. $\boldsymbol{r}_1 \in \mathbb{Z}_q^{\sqrt{n}} \leftarrow \boldsymbol{DB} \cdot \boldsymbol{q}_1$
2. $\boldsymbol{R}_1 \in \mathbb{Z}_p^{F \times \sqrt{n}} \leftarrow \mathrm{Decomp}(r_i)$ ◁ Decomp代表按明文域进行划分
3. $(\boldsymbol{M}_2 | \boldsymbol{r}_2) \in \mathbb{Z}_q^{F \times (n+1)} \leftarrow \boldsymbol{R}_1 \cdot (\boldsymbol{A}_2 | \boldsymbol{q}_2)$
4. **return** $(\boldsymbol{M}_2, \boldsymbol{r}_2)$

客户端 PIR.Extract($\boldsymbol{M}_2, \boldsymbol{r}_{i,j}$)

1. $\boldsymbol{r'}_1^T \in \mathbb{Z}_p^F \leftarrow (\boldsymbol{r}_2 - \boldsymbol{M}_2 \cdot \boldsymbol{s}_2) / \Delta$
2. $r'_1 \in \mathbb{Z}_q \leftarrow \mathrm{Recomp}(\boldsymbol{r'}_1^T)$ ◁ Recomp代表按密文域重组
3. $\boldsymbol{D}_{i,j} \in \mathbb{Z}_p \leftarrow (r'_1 - \boldsymbol{M_1}[i] \cdot \boldsymbol{s}_1[i]) / \Delta$

通过加法同态的方式来进行更高维度的处理，在线阶段服务器计算量复杂度为 $N + NF$ 次 $\mathbb{Z}_q$ 上的加法和乘法，维持了总体复杂度不变。而在通信复杂度上则分别增加了第 2 维的询问和回复。在第 2 维中，矩阵 $\boldsymbol{A}_2$ 可以在离线阶段进行传输，同样能够以伪随机种子生成的方式将通信量降低到 λ。但是矩阵 $\boldsymbol{M}_2$ 依

赖用户询问进行计算，因此必须在在线阶段进行计算和传输，这使得回复通信量从原先的 $\sqrt{N}$ 变为 $F(n+1)$。注意此时通信量和数据库大小无关，因此整体性能得到了提升。

4.1.3 降低客户端线索大小

此处作出另一个重要观察：对于 $\boldsymbol{r}_{i,j}$ 的解密，只需要使用 $\boldsymbol{M}_1$ 中的第 i 行进行解密，就能得到原始数据库中的 $\boldsymbol{D}_{i,j}$。这也就意味着每次其实只需要原来公共中的 $\frac{1}{\sqrt{N}}$，就可以完成一次询问解密。而从 $\boldsymbol{M}_1$ 中提取第 i 行，正好是上一小节中描述的在第 1 维运行本 CPIR 方案所得到的结果！如果在预处理阶段不发送矩阵 $\boldsymbol{M}_1$，而第 2 维询问的时候将 $\boldsymbol{M}_1$ 当成令一个数据库矩阵，在其上递归运行一次之前的方案，就能够得到 $\boldsymbol{M}_1[i]$。由于线索矩阵 $\boldsymbol{M}_1$ 要远小于数据库矩阵，因此其性能开销不会影响总计算复杂度。更深入的，如果表示数据库矩阵的维度不平衡（例如 $l \times m \geq N$），那么就能在线索大小和通信大小中做不同的权衡。

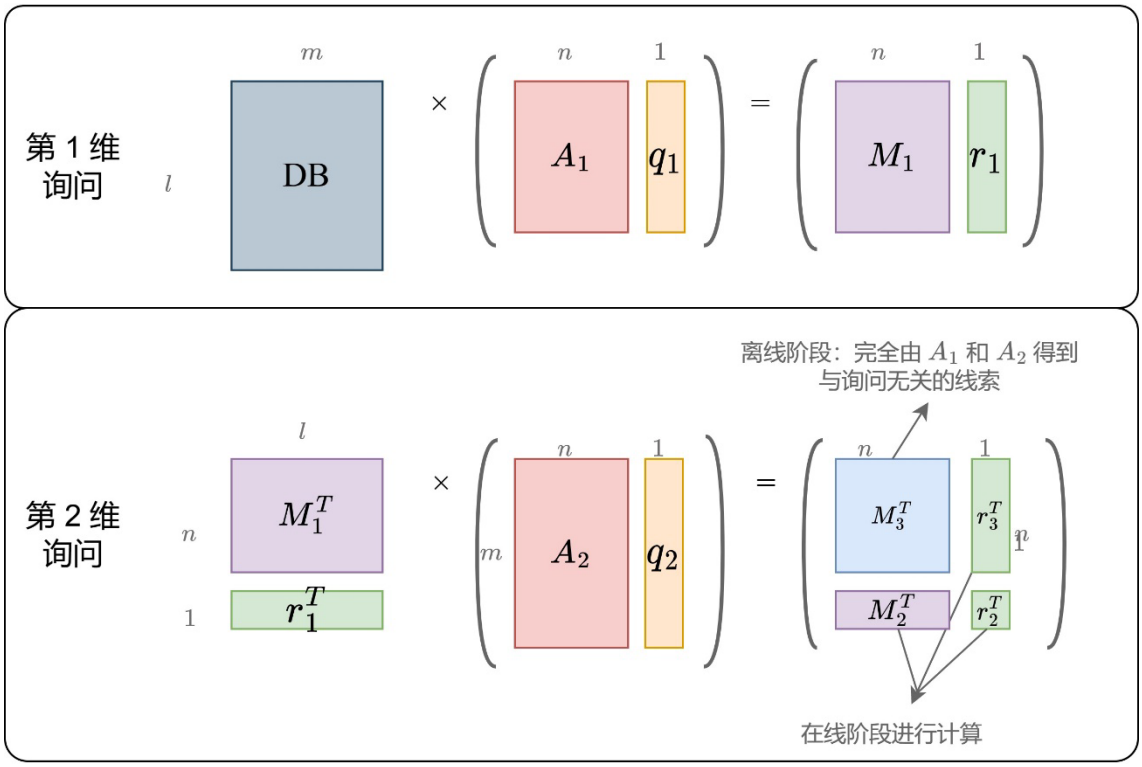

图 4.4 对线索矩阵进行递归

通过这个观察可以给出算法 6 的一个递归变种，图 4.4 省略简化了需要密文扩展来进行操作的部分，将其纳入考量后给出以下算法 7：

算法 7：基于加法同态下算法全过程（小线索版本）

服务器 PIR.Prep$(n, q, p, \lambda, N, F, \boldsymbol{DB} \in \mathbb{Z}^{l\times m})$：◁ 各参数和上文含义相同

1. $\boldsymbol{A_1} \in \mathbb{Z}^{m\times n} \leftarrow \mathrm{PRG}(Seed_1^{\lambda})$
2. $\boldsymbol{A_2} \in \mathbb{Z}^{l\times n} \leftarrow \mathrm{PRG}(Seed_2^{\lambda})$ ◁ 随机生成
3. $\boldsymbol{M_1} \in \mathbb{Z}^{l\times Fn} \leftarrow \mathrm{Decomp}(\boldsymbol{DB}\cdot\boldsymbol{A_1})$ ◁ Decomp代表按明文域划分
4. $\boldsymbol{M_3} \in \mathbb{Z}^{n\times Fn} \leftarrow (\boldsymbol{M_1^T}\cdot\boldsymbol{A_2})^{\boldsymbol{T}}$
5. **return** $\boldsymbol{M_3}$

客户端 PIR.Quary$(x \in [N])$：◁ 询问索引为 x 的密文

1. $r_1 \leftarrow \lfloor x/m \rfloor, r_2 \leftarrow x - r_1$ ◁ 进行维度分解
2. 取样 $\begin{cases} \boldsymbol{s_1}, \boldsymbol{s_2} \leftarrow \mathbb{Z}_q^n \\ \boldsymbol{e_1} \leftarrow \mathcal{X}^m, \boldsymbol{e_2} \leftarrow \mathcal{X}^l \end{cases}$
3. $\boldsymbol{A_1} \in \mathbb{Z}_q^{m\times n} \leftarrow \mathrm{PRG}(Seed_1^{\lambda}), \boldsymbol{A_2} \in \mathbb{Z}_q^{l\times n} \leftarrow \mathrm{PRG}(Seed_2^{\lambda})$
4. $\boldsymbol{q_1} \in \mathbb{Z}_q^m \leftarrow \boldsymbol{A_1}\cdot\boldsymbol{s_1} + \boldsymbol{e_1} + \Delta\mathbf{u}_{\mathbf{r_1}}$ ◁ u_{r_1}为 r_1 处为1，其余为0的向量。$\Delta = \lfloor q/p \rfloor$
5. $\boldsymbol{q_2} \in \mathbb{Z}_q^l \leftarrow \boldsymbol{A_2}\cdot\boldsymbol{s_2} + \boldsymbol{e_2} + \Delta\mathbf{u}_{\mathbf{r_2}}$
6. **return** $(\boldsymbol{q_1}, \boldsymbol{q_2})$

服务器 PIR.Response$(\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{DB})$

1. $\boldsymbol{r_1} \in \mathbb{Z}_q^l \leftarrow \boldsymbol{DB}\cdot\boldsymbol{q_1}$
2. $\boldsymbol{R_1} \in \mathbb{Z}_p^{l\times F} \leftarrow \mathrm{Decomp}(\boldsymbol{r_i})$ ◁ Decomp代表按明文域进行划分
3. $\boldsymbol{M_2} \in \mathbb{Z}_p^{F\times n} \leftarrow \boldsymbol{R_1^T}\cdot\boldsymbol{A_2}$ ◁ 类似算法 6 中的Response.3
4. $(\boldsymbol{R_3^T} \mid \boldsymbol{r_3}) \in \mathbb{Z}_p^{F\times(n+1)} \leftarrow (\boldsymbol{M_1^T} \mid \boldsymbol{r_1})\cdot\boldsymbol{q_2}$ ◁ 递归询问线索矩阵
5. **return** $(\boldsymbol{M_2}, \boldsymbol{R_3}, \boldsymbol{r_3})$

客户端 PIR.Extract$(\boldsymbol{M_2}, \boldsymbol{R_3}, \boldsymbol{r_3})$

1. $(\boldsymbol{M_1'} \mid \boldsymbol{r_1'}) \in \mathbb{Z}_q^{n+1} \leftarrow \mathrm{Recomp}\left(\left((\boldsymbol{r_3} \mid \boldsymbol{r_2}) - (\boldsymbol{M_3} \mid \boldsymbol{M_2})\cdot\boldsymbol{s_2}\right)/\boldsymbol{\Delta}\right)$ ◁ 对第 2 维的线索矩阵、询问进行解密和重组
2. $\boldsymbol{D}_{i,j} \in \mathbb{Z}_p \leftarrow \left(\boldsymbol{r_1'} - \boldsymbol{M_1'}^T\cdot\boldsymbol{s_1}\right)/\Delta$ ◁ 对第 1 维解密

在递归版本中，离线阶段服务器计算量为 $nN + Fn^2\sqrt{N}$ 次 $\mathbb{Z}_q$ 加法和乘法，预处理中的公共线索减小到 $2\lambda + Fn^2$，从原先和数据库大小成亚线性相关（$2\lambda + n\sqrt{N}$）变为和数据库大小**无关**。在线阶段，服务的计算量为 $N + (2n + 1) + F\sqrt{N}$ 次 $\mathbb{Z}_q$ 加法和乘法，通信量则为 $F(2n + 1)$ 。

值得一提的是，在之前的算法中保护的隐私信息只有用户的索引，而数据库中的内容在线索 $\boldsymbol{MR}$ 中被额外泄露给用户（通常表现为用户询问一个条目，能从服务器回复中恢复多个条目）。而算法 7 则对称保护了数据库的隐私，满足[1]提出的对称 PIR(Symmetric PIR, SPIR) 的安全性要求。

表 4-1 不同版本方案复杂度对比

方案	计算代价		在线通信代价		线索大小
	在线	离线	询问	回复	
算法 5	$2N$	$2nN$	$\sqrt{N}$	$\sqrt{N}$	$\lambda + n\sqrt{N}$
算法 6	$2N(1 + F)$	$2nN$	$2\sqrt{N}$	$(n + 1)F$	$2\lambda + n\sqrt{N}$
算法 7	$2N + 2(2n + 1)F\sqrt{N}$	$2nN + 2Fn^2\sqrt{N}$	$2\sqrt{N}$	$(2n + 1)F$	$2\lambda + Fn^2$

4.1.4 其他扩展和加速

询问压缩 Regev 密文可以采用中的询问压缩的方式。Regev 密文空间大小为 $\mathbb{Z}_q$，本可以存储 q 比特的信息。然而在 CPIR 方案中，一个询问中包含的 n 个密文里仅有 1 个加密值为 1，其余都为 0，换言之使用 np 的空间存储了 1 比特信息。询问压缩技术的本质就是在一个密文中加密更多比特的信息，通过同态操作的方式将其展开为原始询问。

计算加速 离线阶段的计算涉及到一个大矩阵乘法，处理代价较为昂贵。显然从矩阵乘法的角度能够对其进行进一步加速。而在线阶段进行的是向量-矩阵乘法，如果有 $\sqrt{N}$ 个询问，那么就可以将这些询问顺序组成一个矩阵，进行矩阵-矩阵乘法，这样就能够复用离线阶段的矩阵乘法加速。可以通过

SIMD 和向量化 SIMD（Single Instruction Multiple Data）即单指令流多数据流，通过 MMX、SSE、AVX 等指令集，可以将多个 32/64bit 的数据操作进行同时处理。本方案的计算主要集中在矩阵乘法方面，非常适合利用 SIMD 技术来进行打包计算优化。

4.2 Batch PIR

该算法在预处理步骤中已经很大程度上降低了计算代价，但是注意到该方案可以使用 Batch Code 来分摊开销的优化相兼容。因此只要将预处理步骤稍作修改，就可以进一步降低计算开销。本方案将沿用[5]中 Probabilistic batch codes (PBC) 的构造方法，来将上述方案转换为 Batch PIR 的版本。在本小节中方案指代 4.1.2 节中的算法 7。

首先给出在预计算情况下进行 Batch 处理的基本流程。

算法 8：带预处理的 Batch PIR 方案

服务器 $\mathrm{BPIR.Prep}(DB)$ ：

1. 计算 $C(DB) \leftarrow (DB_1, \dots, DB_b)$
2. **for** $i = 1$ to b **do**
3. $\quad \mathrm{Hint}_i \leftarrow \mathrm{PIR.Prep}(n, q, p, \lambda, N, F, DB_i)$
4. **return** $H \leftarrow (\mathrm{Hint}_1, \dots, \mathrm{Hint}_b)$

客户端 $\mathrm{BPIR.Query}(\{x_1, \dots, x_k\})$

1. $\big((a_1, b_1), \dots, (a_k, b_k)\big) \leftarrow \mathrm{BatchCode}(\{x_1, \dots, x_k\})$ ◁ 运行 BatchCode 算法，得到 x_l 在 DB_{a_l} 中的坐标 b_l
2. **for** $i = 1$ to m **do**
3. $\quad$ **for** $j = 1$ to k **do**
4. $\quad\quad$ **if** $k == a_j$ **do** ◁ 如果是一个合法数据库，那么运行对其的询问
5. $\quad\quad\quad D_{a_j} \leftarrow \mathrm{PIR.Query}(Hint_i, b_j, i)$
6. $\quad$ **else** $\mathrm{PIR.Query}(Hint_i, 1, i)$ ◁ Batchcode 算法可能会给出一些虚假填充，运行虚假询问
7. **return** $\{D_{a_1}, \dots, D_{a_k}\}$

服务器 BPIR.Response(b_j, i)

1. $A \leftarrow$ PIR. Response(DB_i, b_j) ◁ 在第 i 个数据库上查询索引b_j
2. **return** A

接下来进一步给出方案的具体构造。将本方案改造成 Batch PIR 最直接的想法如下：将含有 N 条记录的数据库随机分成 b 块，每块都表示成 $\frac{\sqrt{N}}{b} \times \sqrt{N}$ 的矩阵。如果用户询问的 $k(k \leq b)$ 条记录分别落在这 b 个不同块中，那么只需要对每个块分别进行询问，总代价为 $k \cdot O\left(\frac{N}{b}\right)$。也就相当于在更小的数据库上进行询问，性能自然提升。为了实现上述效果，需要解决的问题有两个：1. 服务器将数据库通过某个算法分为小块，并保证用户的 k 条记录落在不同块中， 2. 客户端将原来的询问索引转换为每个块上的索引，并通过新索引对不同块发起询问。

这两个问题的解决方式是使用 3.3.1 中介绍的 Batch Code。服务器选取三个哈希算法来组建一个布谷鸟哈希$H = \{h_1, h_2, h_3\}$，并预先设定参数将数据库分为 b 块，随后和每个客户端之间对哈希算法、设定参数达成共识。对于数据库中的每个条目 $DB = \{d_1, d_2, \ldots, d_n\}$，服务器计算 $h_1(d_i), h_2(d_i), h_3(d_i), 1 \leq i \leq n$ 并将对应条目复制到这三个哈希值所指向的桶中，如果同一个元素 d_i 的多个哈希值指向同一个桶，那么只需要复制一次元素。分块之后，新数据库的总量 $\leq 3N$。图 4.5 中展示了服务器分块算法的具体策略。

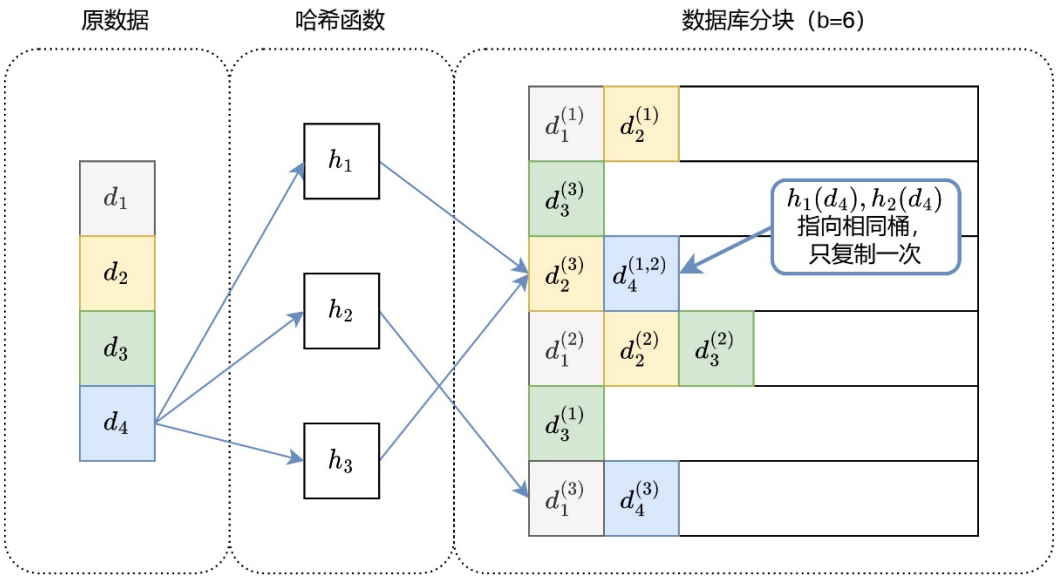

图 4.5 服务器分块策略

客户端在发起一个询问 $Q = \{q_1, \ldots, q_k\}$ 时，采用和布谷鸟策略。对于每个询问首先逐个计算哈希值 $h_j(q_i)$，如果当前桶为空，那么则进行复制，否则计算下一个哈希值 $h_{j+1}(q_i)$ 指向的桶。如果三个哈希值指向的桶都是满的，那么客户端将随机挑选一个哈希值对应的桶进行复制，并将其中原来的元素 q_k 踢出。随后对 q_k 递归运行以上算法，直到所有元素都能找到对应的空桶。**图 4.6** 中展示了客户端算法的具体策略。算法结束之后，客户端将得到每个询问 q_i 在 b 个桶中的对应位置，也就是 $Q' = \{q'_1, \ldots, q'_b\}$，其中必然有 $q'_l = h_j(q_i)$ 或 $q'_l = \perp$（换言之这 b 个 batch 询问中，每个元素都是原询问的某个哈希值或者随机填充数据）。注意由于服务器的分块策略，此时用户的某个有意义询问 $q'_l = h_j(q_i)$ 一定是三个哈希值之一，而服务器对于数据库中每个元素的哈希值都进行复制，那么必然有 q'_l 落在第 l 个桶中，这样就解决了上述的两个问题。

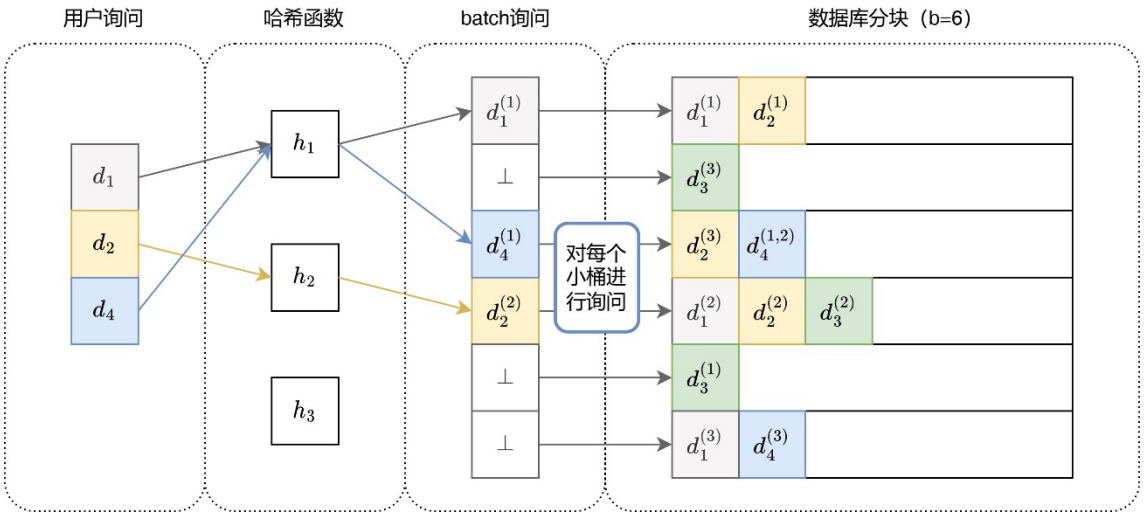

图 4.6 客户端询问策略

由于哈希函数的设计通常有着较好的负载，因此服务器的分块结果中，所有数据将均匀分布在每个块中，不产生堆积。这样设计所得到的分块均匀，且每个块数据量相对原数据库都更小，从而该 Batch Code 的整体设计性能较好。

4.3 真实数据库问题及处理

之前的 CPIR 论文中主要专注于理论最优化，简化忽略了各种现实因素。在本小节中将提出真实数据库和理论方案之间不匹配的几个重要问题，并给出对应的解决方案。

不均匀数据库 为了实现效率最优，这些 CPIR 方案通常要精心挑选 HE 方案的参数，而这些参数也决定了数据库中每条数据的大小。因此几乎所有论文中，数据库的每条条目都是一个**固定大小**。大部分方案中都将原始数据作为无意义数据流进行处理，进行计算的基本单位即数据库的每个条目（同态明文大小）。而在现实生活中，原始数据库中的每条数据经常是不均匀的，不能够简单的当成一个同态明文。

由于 PIR 方案中选择参数会很大程度影响方案表现，因此最好将数据库中的每个条目固定为理论最优值的大小，通过处理手段将原始数据库中的数据放置到这些固定大小的桶中。平凡的解决方式是将每条数据填充、切分成理论最优值，通过剥离多余数据和多条询问重组的方式来重构原始数据，然而这样显然会造成服

务器端存储空间的浪费，但是不会因为数据库重组而增加任何额外计算量。

一种可能的处理方式是按照数据库原始索引顺序，将尽可能多的条目放进一个同态明文中，附加上某些特征比特来对不同条目进行区分，这样就对同一个明文中的空间有了更多的利用。这样的做法缺点在于需要经常性地对原始数据中的一个条目进行划分，客户端可能需要更多次的询问来取回 1 个完整条目。而另外一个问题是用户很难知道希望询问的条目 i 具体放在数据库中的哪个位置。这就要求创建一个关于条目 i 具体放置位置的元数据库（相对较小），首先对元数据库进行检索来得到 PIR 方案中的具体索引。这个元数据库可以由客户端或者服务器进行存储。如果放置在服务器上，就需要首先对这个较小元数据库进行一次 PIR 检索，由于这个元数据库中每个条目大小相等，没有现实数据库中的不均匀问题，因此可以选择适当的 PIR 方案和参数，其代价相对较小。

另一种想法是增加哈希中间层，将索引 i 映射到某个哈希值 $h(i)$ 上，每个哈希值代表数据库中一个条目的位置，所有映射到同一个 $h(i)$ 上的索引都被放置在同一个密文中。客户端在发起询问时计算出 $h(i)$，并取回整个明文条目。由于哈希值本身隐含了映射到的位置信息，因此没有上一种做法中的位置元数据问题。这种做法适用于数据库中存在较多均匀小条目的情况，但是对于大条目需要切割的情况依旧需要进行额外处理。

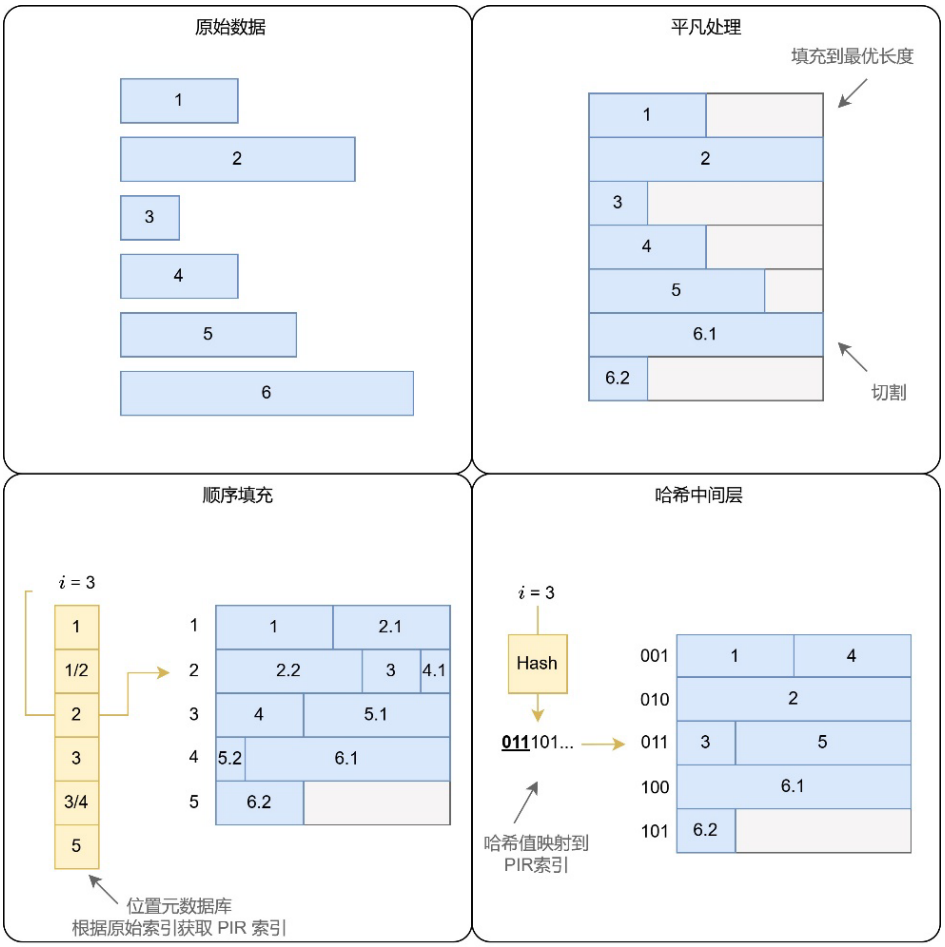

图 4.7 不均匀数据库策略

动态数据库 所有经过预处理的方案都需要对整个数据库进行计算，得到某些和整个数据库相关的线索。虽然这大幅降低了方案的计算复杂度，但是这不适用于动态数据库的场景。由于现实数据库通常不是一成不变的，如果考虑一个仅增长型数据库，传统无状态的方案可以很快速解决数据库更新问题，大部分方案只需要对数据库进行维度更新，在必要的时候扩大同态参数或者以更高维的形式表现数据库。而在预处理方案中，每次更新时都需要相对应重新运行预处理步骤，使得**一次性预处理代价变成多次**。

在本方案中的预处理需要计算 $\boldsymbol{DB} \cdot \boldsymbol{A}$，当进行数据库内容增加时，我们采取只在第 1 维上进行更新的手段，即 $\boldsymbol{DB} \in \mathbb{Z}_p^{l \times m}$ 增加 k 条数据后，数据库结构变

为 $\begin{pmatrix}\boldsymbol{DB}\\ \boldsymbol{DB}_{new}\end{pmatrix} \in \mathbb{Z}_p^{\left(l+\left\lceil\frac{k}{m}\right\rceil\right)\times m}$，其中 $\boldsymbol{DB}_{new} \in \mathbb{Z}_p^{\left\lceil\frac{k}{m}\right\rceil\times m}$。每次新增一行 $\boldsymbol{d} \in \mathbb{Z}_p^m$时，只需要在新增的部分运行预处理步骤 $\boldsymbol{m} \leftarrow \boldsymbol{d} \cdot \boldsymbol{A} \in \mathbb{Z}_p^n$，并将新增元素发送给客户端。经过这样处理，就不必将重新计算整个数据库矩阵的预处理 $\begin{pmatrix}\boldsymbol{DB}\\ \boldsymbol{DB}_{new}\end{pmatrix} \cdot \boldsymbol{A}$，能够使得在新增 1 个元素的预处理的代价和第 2 维长度 m 成**线性比例**。

和增加类似的，如果数据库中的某个条目 (i,j) 进行原地更新，那么也需要对所在行 $\boldsymbol{DB}[i]$ 进行重新计算，使用新计算的结果 $\boldsymbol{DB}'[i]$ 代替之前预计算矩阵中的第 i 行。

略有不同的是删除操作，为了重复利用之前预计算的结果，通常不对之前的数据库矩阵进行修改（换言之不会调整放置在被删除元素之后元素的索引），而是将其修改为某个特殊值，标记其被删除。然后同样按照原地修改的操作更新该行。

键值对数据库 在以上介绍的所有 CPIR 方案中，客户端询问都采用下标索引的方式，这隐含了一个假设——客户端知道自己的询问处于数据库中的具体位置。这个假设在某些场景下是可能的，例如比特币交易场景下，某个客户端希望根据某个交易号在数据库中查找交易细节。然而大部分现实数据库中，客户端询问是采取键值对询问的模式，通常采用将客户端的询问的“键”映射到某个索引上的方式来处理键值对数据库。

最基础的方式就是让客户端直接存储一个键到索引的映射表，由于这个表和数据库大小呈现线性关系，实用性不高。另一种想法是计算客户端询问键 k 的哈希值 $h(k)$，并直接用该哈希值作为索引。然而为了使得每条记录不冲突，通常需要比原来数据库条目 N 更大的空间，这将使得每条记录放置的位置分散在数据库中。[7] 中构造了一种使用布谷鸟哈希来处理索引的方式，使处理键值对数据库的代价大约为原来基于索引的 PIR 方案的 2 倍。

4.4 实现、效率分析及对比

本方案的测试环境为 Windows 11 系统，CPU 为 AMD Ryzen 7 6800H（2.5GHz），L1、L2、L3 级缓存大小分别为 512KB、4MB 和 16MB，内存大小 16GB。方案实现使用 C++ 语言，理论上在任何系统上安装了支持 C++20 及以上的编译器都能运行本方案代码。

本方案共包含数据库（Matrix/Database.cpp）、PIR 参数（Pir.cpp）、服务器（simplePirServer.cpp）、客户端（simplePirClient.cpp）四个主要组件。Matrix 结构体是用来表示数据库的形式，其中包含了行、列信息，数据库本体和一系列随机生成矩阵、矩阵计算、矩阵压缩等具体方法。Database 结构体有两个数据成员，分别是 DBInfo 和 Matrix，前者负责管理数据库大小、每个条目大小等元数据；后者则负责具体数据库信息。在给定了数据库的参数后，Database 结构体将会计算接近理论最优值的维度表示，随后 PirParams 结构体通过维度来选择对应的 PIR 参数。 simplePirServer 结构体中包含一个 Database，其中的两个主要算法函数是预处理（preprocess）和回复询问（reply）；simplePirClient 的主要方法则是发起询问（query）和解码回复（extract）。

对于原方案的实验结果如下

表 4-2 对不同数据库大小时间大小比较

数据库大小	计算时间（ms）			在线占比
（MB）	预处理	生成询问	回复询问	
1.24	1008	2	2	0,40 %
64	51964	14	48	0,12 %
128	105857	19	110	0,12 %
256	243055	28	351	0,16 %
512	522112	54	648	0,14 %
1210	-	73	1300	-

表 4-3 对不同数据库大小通信代价较

数据库大小	通信代价（KB）			在线占比
（MB）	预处理	生成询问	回复询问	
1.24	4096	4	4	0,20 %
64	29536	29	29	0,20 %
128	42400	41	41	0,19 %
256	60208	57	59	0,19 %
512	86800	82	85	0,19 %
1210	121520	117	119	0,19 %

在不同数据库大小下，在线过程中的时间、通信代价都只占据了 0.2%左右的代价，因此本方案的在线吞吐量极高。在一次性的离线设置结束后，用户每次询问的在线响应速度非常快。

第 5 章 结论

5.1 本文工作总结

1. 本文首先对基于同态加密所构造的 CPIR 方案进行了厘清，主要分为基于加法同态和基于全同态两个方向。并介绍了如何通过引入 IT-PIR 方案中的编码手段，通过额外空间代价使得计算时间达到亚线性的方法。通过第 3 章中的方案介绍、总结归纳和具体分析，给出了目前研究中不同方案的对比。

2. 基于上述分析和对比，本文整理了目前 CPIR 研究中的主要问题：服务器处理时间长、预计算代价昂贵、存储空间权衡等。根据这些观察，本文给出了一个基于 Regev 加密的预处理 CPIR 方案设计，该方案中通过一个**较小的公共预处理存储**，极大地**降低了服务器在线计算时间**。同时本文还给出了在这个预处理方案上通过加法同态来进行高维处理的具体算法，随后给出通过递归降低线索矩阵的方法。分别在计算量、通信量和存储量上有着不同的权衡。对比现有各种方案，此方案能够达到各方面较好的性能权衡。

3. 在该方案的基础上进行进一步延伸，通过 Batch Code 的手段将多个询问的计算量进行分摊。并且从工程实践的方向出发，对理论 PIR 方案中的数据库设置提出三个方向的问题，并针对性的给出了可能的解决手段。

5.2 未来方向展望

1. 通过客户端独立状态进一步提升效率：利用可穿刺伪随机集合的方式独立于本 PIR 方案的构造。在本方案中服务器并没有额外存储和每个客户端相关的信息，虽然能够从比例上大幅度降低计算代价，但是并没有跨越线性计算的边界。接下来可以构造二者的融合，达到效率更高的 PIR 方案。

2. 不同参数构造：PIR 方案的参数是方案表现的重要部分，然而现实数据库具有不同的特征，因而适用于不同参数。例如在视频数据库中，每个数据条目都较大，因而选择的同态明文大小将决定需要多少询问才能取回一个完整视频。因此需

要对更多的参数构造进行分析，包括明密文空间 $\mathbb{Z}_p, \mathbb{Z}_q$，同态维度 n，数据库递归维度 d等。

3. 应用场景实践和具体优化：CPIR 方案已经成熟到可以落地应用，近年来不断有工作将其应用到现实场景中。未来可以探索适合 CPIR 的场景并进行针对性工程化，例如具有固定大小的元数据库、相似大小的维基百科等具体数据库。另外还可以将 CPIR 作为底层工具构建集合成员判断等其他密码学协议。

参考文献

[1] Chor B, Kushilevitz E, Goldreich O, et al. Private information retrieval[J]. Journal of the ACM (JACM), 1998, 45(6): 965-981.

[2] Kushilevitz E, Ostrovsky R. Replication is not needed: Single database, computationally-private information retrieval[C]//Proceedings 38th annual symposium on foundations of computer science. IEEE, 1997: 364-373.

[3] Beimel A, Ishai Y, Malkin T. Reducing the servers computation in private information retrieval: PIR with preprocessing[C]//Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20. Springer Berlin Heidelberg, 2000: 55-73.

[4] Melchor C A, Barrier J, Fousse L, et al. XPIR: Private information retrieval for everyone[J]. Proceedings on Privacy Enhancing Technologies, 2016: 155-174.

[5] Angel S, Chen H, Laine K, et al. PIR with compressed queries and amortized query processing[C]//2018 IEEE symposium on security and privacy (SP). IEEE, 2018: 962-979.

[6] Gentry C. Fully homomorphic encryption using ideal lattices[C]//Proceedings of the forty-first annual ACM symposium on Theory of computing. 2009: 169-178.

[7] Ali A, Lepoint T, Patel S, et al. {Communication–Computation} Trade-offs in {PIR}[C]//30th USENIX Security Symposium (USENIX Security 21). 2021: 1811-1828.

[8] Gentry C, Halevi S. Compressible FHE with applications to PIR[C]//Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part II. Cham: Springer International Publishing, 2019: 438-464.

[9] Mughees M H, Chen H, Ren L. OnionPIR: Response efficient single-server PIR[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021: 2292-2306.

[10]Peikert C, Vaikuntanathan V, Waters B. A framework for efficient and composable oblivious transfer[C]//Advances in Cryptology–CRYPTO 2008: 28th

Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28. Springer Berlin Heidelberg, 2008: 554-571.

[11]Regev O. On lattices, learning with errors, random linear codes, and cryptography[J]. Journal of the ACM (JACM), 2009, 56(6): 1-40.

[12]Brakerski Z, Gentry C, Halevi S. Packed ciphertexts in LWE-based homomorphic encryption[C]//Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16. Springer Berlin Heidelberg, 2013: 1-13.

[13]Stern J P. A new and efficient all-or-nothing disclosure of secrets protocol[C]//Advances in Cryptology—ASIACRYPT'98: International Conference on the Theory and Application of Cryptology and Information Security Beijing, China, October 18–22, 1998 Proceedings. Springer Berlin Heidelberg, 1998: 357-371.

[14]Gentry C, Halevi S, Smart N P. Fully homomorphic encryption with polylog overhead[C]//Eurocrypt. 2012, 7237: 465-482.

[15]Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping[J]. ACM Transactions on Computation Theory (TOCT), 2014, 6(3): 1-36.

[16]Chillotti I, Gama N, Georgieva M, et al. TFHE: fast fully homomorphic encryption over the torus[J]. Journal of Cryptology, 2020, 33(1): 34-91.

[17]Menon S J, Wu D J. Spiral: Fast, high-rate single-server PIR via FHE composition[C]//2022 IEEE Symposium on Security and Privacy (SP). IEEE, 2022: 930-947.

[18]Ishai Y, Kushilevitz E, Ostrovsky R, et al. Batch codes and their applications[C]//Proceedings of the thirty-sixth annual ACM symposium on Theory of computing. 2004: 262-271.

[19]Persiano G, Yeo K. Limits of preprocessing for single-server PIR[C]//Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Society for Industrial and Applied Mathematics, 2022: 2522-2548.

[20]Gentry C, Ramzan Z. Single-database private information retrieval with constant communication rate[C]//Automata, Languages and Programming: 32nd

International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings 32. Springer Berlin Heidelberg, 2005: 803-815.

[21]Boyle E, Ishai Y, Pass R, et al. Can we access a database both locally and privately?[C]//Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II 15. Springer International Publishing, 2017: 662-693.

[22]Corrigan-Gibbs H, Kogan D. Private information retrieval with sublinear online time[C]//Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. Springer International Publishing, 2020: 44-75.Shi E, Aqeel W, Chandrasekaran B and Maggs B. Puncturable Pseudorandom Sets and Private Information Retrieval with Near-Optimal Online Bandwidth and Time. Advances in Cryptology – CRYPTO 2021.

[23]Corrigan-Gibbs H, Henzinger A, Kogan D. Single-server private information retrieval with sublinear amortized time[C]//Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part II. Cham: Springer International Publishing, 2022: 3-33.

[24]Kogan D, Corrigan-Gibbs H. Private blocklist lookups with checklist [C]: 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, 2021.

[25]Henzinger A, Hong M M, Corrigan-Gibbs H, et al. One server for the price of two: Simple and fast single-server private information retrieval[C]: 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA: USENIX Association, 2023.

[26]Davidson A, Pestana G, Celi S. Frodopir: Simple, scalable, single-server private information retrieval[J]//Proc. Priv. Enhancing Technol, 2023: 365-383.

[27]H. Chen, K. Han, Z. Huang, A. Jalali, and K. Laine. Simple encrypted arithmetic library v2.3.1. sealmanual.pdf (microsoft.com)

致　　谢

写至最后一章，本科学习便告一段落。这四年于我而言，不是一帆风顺的直行路，而是蜿蜒崎岖的登山道。回首来时路，沿途景已纳入下一程的行囊。

最应感谢的人是我的父母。无论我做出什么选择，总以信赖为我构建无尽的底气。在每个没有红绿灯的人生分岔路口上，我永远有勇气走向期望的道路。感谢我妹的出生，我们之间章鱼和蘑菇的友谊从来没有年龄的阻隔。我常见人人尽其所能追寻一条完美的人生道路，谨小慎微生怕踏错一步。而我的家人则引领我拥抱人生中所有的可能性，因此我才能成为现在这个坦荡坚定的我自己。

经师易求，人师难得。在此想感谢我的导师陈宇老师，他以对学术的热情、严谨深刻影响着我。如果没有遇到这样一位平易近人而有趣的老师，我也许不会选择密码学作为我的毕业论文。感谢胡思煌老师和王美琴老师在我申请留学的时候给予的帮助，让我能够进一步在学术道路上求索。感谢所有曾教导过我、给予我肯定的各位老师，让我拥有在学业上继续前行的信心。在网络空间安全学院的四年间，我所学会的东西远超于单纯的理论和技术。

还要感谢四年旅程中所有遇见的人。感谢养老院群里的五个朋友，我们在各奔东西的同时却又向着同一个方向，希望我们永远都会是无话不谈的朋友，永远能够是彼此支撑的脊梁，永远不生疏不离分。在未来的某一天，希望我们能真的住进同一家养老院。感谢端木浩杰和我一起共度的时光，这是我未曾预料的、最为奇迹的平凡日常。感谢四年间有趣又可爱的四任舍友，以及那些经常和我讨论问题的朋友。我一直是偏好独来独往的人，但同行的朋友成为了我和外界的桥梁，让我的大学精彩热烈。感谢在大学前就认识的瓜群亲友和江九，我永远不会忘记 2018 那个艳红的夏末。虽然我总是因为各种各样的意外搁浅了与你们的见面计划，但我们总能在未来的某个时间点汇合。

怕什么真理无穷，进一寸有进一寸的欢喜。写下致谢最后一段的时候，我已经订好了飞往赫尔辛基的机票，准备以勇气、独立和永远进取的心拥抱下一段航程。落笔为终，愿我未来如今日，一如往常向前走。

附　　录

附录 A 译文

在 PIR 中的通信-计算权衡

摘要

我们研究了不同隐私信息检索构造的计算和通行代价，以及他们可能的权衡方案。包括基于同台加密的方案和 Gernty-Ramazan PIR。

我们使用压缩技巧和一种全新的不经意扩展对 **SealPIR 方案**进行了改进，使得通行带宽降低了 80%并且几乎保持了同样的计算代价。然后我们提出了 **MulPIR 方案**，利用乘法的同态性来实现 PIR 中的递归步骤。虽然在以前的工作中已经考虑过使用乘法同构，但我们观察到，与我们的其他技术相结合，它引入了一个有意义的权衡。即当数据库有大条目时，大大减少了通信，代价是增加服务器的计算成本。对于一些应用，我们表明，这可以将服务器的总代价成本降低 35%。

在通信-计算频谱的另一端，我们仔细研究 **Gentry-Ramzan PIR**，一个具有渐进最优通信速率的方案。在这里，瓶颈是服务器的计算，我们设法大大减少了计算。我们的优化使我们能够在通信和计算之间进行可调整的权衡，这使我们能够将服务器的计算量减少 85%，但代价是查询量增加。

最后，我们介绍了在稀疏数据库上处理 PIR 的新方法（**keyword PIR**），基于不同的散列技术。我们实现了我们所有的结构，并比较了它们在几个应用场景中的通信和计算开销。

1 介绍

访问公共数据库往往会给查询者带来隐私方面的担忧，因为查询可能已经暴露了敏感的信息。例如，对医疗数据的查询可能会暴露敏感的健康信息，而对金融数据的访问模式可能会泄露投资策略。在这种隐私泄露有重大风险的情况下，客户

可能会对访问数据库望而却步。反过来说，数据提供者往往不希望访问敏感的客户查询，因为这些查询后来可能成为他们的责任。

私人信息检索（PIR）是一种加密原语，旨在解决上述问题，使客户能够**查询数据库而不向数据所有者透露任何关于他们查询的信息**。虽然这个基本原理的可行性已经解决了很长时间[14]，但为实际应用寻找具体有效的构造一直是一个活跃的再研究领域[4, 5, 9, 18, 23, 24, 30, 35, 46, 64]。在这种情况下，有几个参数和效率措施来描述 PIR 的设置，并决定什么解决方案可能最适合于特定的场景。

在这项工作中，我们深入研究了数据存储在**单一服务器**上的 PIR 环境。这种 PIR 模型是关于以下实用场景设置中的：**没有第三方可以协助数据存储和查询执行，而且人们不希望信任安全的硬件**。众所周知，非平凡的单服务器 PIR 构造需要计算假设[42]，与多服务器设置中可能存在的信息理论构造相比，这种解决方案在通信和计算成本方面都带来了巨大的开销[20]。虽然 PIR 的理论构造[42]实现了 poly-logarithmic 复杂度的通信，但大多数高效的单服务器 PIR 实现都没有达到这个目标，只是实现了具有更高渐进通信成本的构造的变体[4, 5, 35, 46]。

我们分析了不同的 PIR 构建方法所提供的通信-计算权衡，以及在实践中实现最佳渐进通信成本的障碍。这包括两种主要的 PIR 构造，它们依赖于概念上不同的技术。利用同态加密的 PIR，以及 Gentry 和 Ramzan[31]的 PIR 方法，利用具有隐藏平滑子群的群。第一类技术用于大多数现有的完全同态加密被提议作为建立 PIR 的工具[29]，现有的 PIR 实现[4, 5, 35, 46]利用仅依赖于**加法同态加密**的构造方法。这样的结构模拟了一种具有加法加密层的乘法同态的再限制形式。虽然这种方法允许最先进的协议，如 SealPIR[3，5]在计算方面表现良好，但它产生的密文扩展是**乘法计算深度的指数**，并且在实践中即使是少量的分层乘法，也有很大的通信开销。

我们提出了一种新的 PIR 结构，MulPIR，它在多个方面改进了这种技术状态。首先，我们表明 SealPIR 的**通信开销**可以在几乎没有计算成本的情况下大幅减少。我们进一步表明，通过使用底层 HE 方案的乘法同构性，我们可以进一步减少具有**大条目的数据库**的通信开销，这次是以增加的计算成本进行的。虽然使用乘法同构

性之前已经被考虑过，但我们首次表明，结合我们的其他改进，它能够在通信和计算之间进行有意义的权衡。在我们在谷歌云平台上的实验中，我们观察到，这可以将总的服务器成本开销降低 35%。

我们还重新审视了 Gentry-Ramzan PIR 方案[31]，该方案实现了最佳的通信，但计算开销很高。我们展示了如何有效地实现 Gentry-Ramzan PIR，即使是大型数据库中。并 i 企鹅提出了一个新的客户端辅助的变体，允许在通信和计算成本之间进行可调整的权衡。我们的经验表明，根据数据库的形状，无论是 MulPIR 还是客户端辅助的 Gentry-Ramzan PIR 都能使服务器的总成本最小化。

最后，我们转向 keyword PIR[13]，这是一个数据库大小比查询键域小得多的变体。常规的 PIR 结构假定数据库密集，因此它们的复杂性取决于索引域的大小，而 keyword PIR 旨在实现服务器的计算成本**只取决于实际的数据库大小**，而不是密钥域大小。我们提出了两种基于两种不同的散列方案的构造，并表明它们能够以另一种方式来交换通信和计算成本。

我们实现了我们所有的新型 PIR 方案以及替代方法，并在广泛的应用中进行了实验比较，包括匿名信息传递（如在以前的工作中使用的[5]），私人文件下载和密码检查[63]。由于篇幅限制，我们在附录中介绍了相关的工作。

1.1 我们的贡献

改进 SealPIR 通信量 近年来实现的最高效（安全）的单服务器 PIR 构造[4, 5, 18, 23, 24, 30, 35, 46, 51, 64]是基于同态加密（HE）技术，并实现了亚线性通信。其中，目前提供最佳实施性能的方案是 SealPIR[3，5]。虽然在理论上，这种构造在使用 d 层递归的时候，支持亚线性的通信量$O(d \cdot n^{\frac{1}{d}})$。但是在实际应用中，却有很大的通信量成本。这是由于分层加法同态加密方法的缘故：如果加密方案有密文扩展F，那么 PIR 回复将会包含F^{d-1}个密文（在[3]中$F = 10$）。这导致了$O(F^2)$的扩展通信，使得这个方案在有大条目的数据库上不可使用。

我们的第一个贡献是通过以下方法减小了 SealPIR 的通信量：使用（1）对称密钥加密来减小上传规模（2）使用模数转换约简技巧（在[9]中的 PIR 中被提出）

将F的值降低为$F=4$（3）介绍新的不经意扩展算法使得某些参数设置下上传大小可以进一步减半。因此，我们优化后的 SealPIR 可以减少 75%的上传通信量，以及 80%的下载通信量。

使用乘法同态 当 SealPIR 中使用递归时，下载通信量随着递归层数以指数形式递增（之前的贡献减少了指数的基础）。相反，我们建议在底层 HE 同时使用加法和乘法同态，方式是：通过在每个递归步骤中对加密值进行一次乘法。在之前的步骤中，上传和下载量是减少$O(\lceil d\cdot n^{\frac{1}{d}}/N\rceil+F^{d-1})$，其中$F$是需要被转换成单个 HE 密文的明文数量。而在经过这个优化后则降低到$O(\lceil d\cdot n^{\frac{1}{d}}/N\rceil+c(d))$，其中$c(d)$是支持 d 次连续乘法的密文的大小。综合上面提到的优化，乘法同态将会得到一个具有高通信量效率的 PIR 方案，我们将其称之为 MulPIR。对于有着大条目的数据库，它对于 SealPIR 的优势在低递归水平的时候是很显著的（下载通信量减少 60%）。并且我们观察到，对于我们所关系的数据库大小，$d=2$的时候效果是最优的。

Gentry‐Ramzan PIR：新的效率权衡 Gentry‐Ramzan PIR 构造对于几种设置达到了最优通信复杂度，但是却需要付出高昂的计算代价。因此，我们的贡献集中在如何减少这种计算开销上，其中包括将服务器的数据库编码为方案中计算所需的 CRT 形式的新的有效技术，回答每个查询所需的快速模块化指数化的新技术，以及在通信和编译之间进行权衡的客户端辅助 PIR 的技术。

在这种 PIR 协议中，服务器数据库$\{D_i\}_{i\in[n]}$需要被编码为$x=D_i\ mod\ \pi_i,\forall i\in[n]$，其中$\pi_i$为两两互素的整数。中国余数定理的朴素应用需要的计算量至少是数据库大小的四倍。我们利用分治模数插值算法[7]，使我们能够实现计算复杂度$\tilde{O}(nlog_2\ n)$。这种技术还允许预计算，可以重复用于使用同一组模子π_i的计算中。

服务器端每次查询的主要计算成本是模幂操作，其中指数是编码的数据库，基数和模数由客户端选择。我们的方法是将指数作为**生成元的预计算幂的乘积**来计算，并使用 Straus 的算法[62]来有效地完成这一工作。这就实现了一种客户端辅助技术，使我们能够以客户端（少量）的额外工作为代价来改善服务器的计算。特别

是，我们观察到在进行幂乘之前，通过使用模数的素因子分解，减少指数对群组顺序的影响（to reduce the exponent modulo the order of the group），能够使客户端更有效地预计算生成元的幂，。这为协议的计算和通信复杂性提供了一种新的权衡方式。在第 6 节中，我们展示了提供几个预计算的幂的证据，优化了服务器的工作。

对于稀疏数据库的 Keyword PIR 我们考虑稀疏数据库的集合，其中服务器的数据库在索引域中是稀疏的，因此客户端的查询对应于一个关键字的查找。我们提出了两种结构，它们都是基于散列方案的。第一种是基于简单的散列，其中数据库元素被分配到使用公共散列函数的桶中。然后客户端检索与他们的查询相对应的桶。虽然桶的大小通常会变得相当大，但在有些明文大小很大的方案中（如 MulPIR）这不是问题。我们的第二个构造以一种新的方式利用了 cuckoo hash[48]。与之前的工作[5]不同，cuckoo hash 法被用来将多个客户端查询批处理到一个，我们使用它将稀疏的服务器数据库压缩到一个密集的领域。cuckoo hash 保证了最多有一个元素被散列到任何一个桶中，代价是客户查询的数量增加（是常数）。这个变体对 Gentry-Ramzan PIR 特别有用，我们有小的明文，另外还可以使用 CRT 批处理[36]将多个客户端查询压缩成一个。

PIR 方案的对比和实用性估计. 我们对基于同态加密的 PIR 成本进行了全面比较。这包括对密本大小以及不同 HE 方案的加密、解密和同态操作的计算成本的详细具体估计。我们利用这些估计值来剖析使用相应方案的 PIR 构造在有递归和无递归情况下的效率成本。我们进一步介绍了这些 PIRs 在不同形状的数据库（记录数量和条目大小）中的实施情况的经验评估。我们的 benchark 的结果表明，在大多数情况下，**基于格的 HE 结构（也可以提供乘法同构性）在计算上优于其他加法 HE 方案。**在通信方面，当主要的通信成本是下载时，加法 HE 方案具有优势。例如，在具有大条目的小型数据库中没有递归的方案，因为这些加密提供了明文和密码文本之间的最佳比率。

我们评估了我们新的 PIR 结构，MulPIR，它使用了 somewhat-homomorphic 的加密（SHE），并实现了计算与通信的权衡，并与 SealPIR 进行了比较。

在我们的实验中，Gentry-Ramzan PIR 总是能实现最好的通信复杂度，但却伴

随着显著的计算成本，在某些情况下可能会被禁止。然而，我们表明，就货币成本而言，当数据库元素较少时，Gentry-Ramzan 可以胜过所有其他的 PIR 方法。

最后，我们将 keyword PIR 的结构应用于密码检查(password checkup)问题，在这个问题上，客户旨在检查他们的密码是否包含在一个泄露的密码数据集中，而不向服务器透露它。以前解决这个问题的方法[63]首先向服务器透露一个 k-匿名的标识符，以减少需要比较的候选密码的数量为 k，然后应用一个变种的私有集交集来比较当前密码和 k 个候选密码。我们对 Gentry-Ramzan 和 MulPIR 的实现使这种查找的通信量为 k 的次线性，因此在相同的带宽下可以实现更好的匿名性，或者相同的匿名性和更小的带宽。

附录 B 原文

Communication–Computation Trade-offs in PIR

Asra Ali
Google
asraa@google.com

Tancrède Lepoint
crypto@tancre.de

Sarvar Patel
Google
sarvar@google.com

Mariana Raykova
Google
marianar@google.com

Phillipp Schoppmann
Google
schoppmann@google.com

Karn Seth
Google
karn@google.com

Kevin Yeo
Google
kwlyeo@google.com

Abstract

We study the computation and communication costs and their possible trade-offs in various constructions for private infor- mation retrieval (PIR), including schemes based on homomor- phic encryption and the Gentry–Ramzan PIR (ICALP'05).

We improve over the construction of SealPIR (S&P'18) using compression techniques and a new oblivious expansion, which reduce the communication bandwidth by 80% while preserving essentially the same computation cost. We then present MulPIR, a PIR protocol additionally leveraging multi- plicative homomorphism to implement the recursion steps in PIR. While using the multiplicative homomorphism has been considered in prior work, we observe that in combination with our other techniques, it introduces a meaningful tradeoff by significantly reducing communication, at the cost of an in- creased computational cost for the server, when the databases have large entries. For some applications, we show that this could reduce the total monetary server cost by up to 35%.

On the other end of the communication–computation spec- trum, we take a closer look at Gentry–Ramzan PIR, a scheme with asymptotically optimal communication rate. Here, the bottleneck is the server's computation, which we manage to reduce

significantly. Our optimizations enable a tunable trade- off between communication and computation, which allows us to reduce server computation by as much as 85%, at the cost of an increased query size.

Finally, we introduce new ways to handle PIR over sparse databases (keyword PIR), based on different hashing tech- niques. We implement all of our constructions, and compare their communication and computation overheads with respect to each other for several application scenarios.

1 Introduction

Accessing public databases often brings privacy concerns for the querier as the query may already reveal sensitive in- formation. For example, queries of medical data can reveal sensitive health information, and access patterns of financial data may leak investment strategies. In settings where such privacy leakage has significant risk, clients may shy away from accessing the database. On the flip side, data providers often do not want access to sensitive client queries, as they could later become a liability for them.

Private information retrieval (PIR) is a cryptographic prim- itive that aims to address the above problem by enabling clients to query a database without revealing any infor- mation about their queries to the data owner. While the feasibility of this primitive has been resolved for a long time [14], the search for concretely efficient constructions for practical applications has been an active area of re- search [4, 5, 9, 18, 23, 24, 30, 35, 46, 64]. In this context, there are several parameters and efficiency measures that characterize a PIR setting and determine what solution might be most suitable for a particular scenario.

In this work, we take a deep dive into the setting of PIR where data is stored on a single server. This is the relevant PIR model in practical settings where no additional

party is avail- able to assist with the data storage and query execution and one does not wish to trust secure hardware. Non-trivial single server PIR constructions are known to require computational assumptions [42], and such solutions bring significant over- heads for both the communication and computation costs com- pared to information theoretic constructions that are possible in the multi-server setting [20]. While theoretical construc- tions for PIR [42] achieve poly-logarithmic communication, most efficient single server PIR implementations stop short of this goal and implement only variants of the construction with higher asymptotic communication costs [4, 5, 35, 46].

We analyze the communication–computation trade-offs that different PIR construction approaches offer and the hur- dles towards achieving the optimal asymptotic communica- tion costs in practice. This includes the two main types of PIR constructions that rely on conceptually different tech- niques: PIR leveraging homomorphic encryption, and the PIR approach of Gentry and Ramzan [31] leveraging groups with hidden smooth subgroups. The first type of techniques are used in the majority of existing PIR constructions. While fully homomorphic encryption has been proposed as a tool for building PIR [29], existing PIR implementations [4, 5, 35, 46] leverage constructions approaches that rely only on additive homomorphic encryption. Such constructions emulate a re- stricted form of multiplicative homomorphism with layers of additive encryption. While this approach allows state-of- the-art protocols such as SealPIR [3, 5] to perform well in terms of computation, it incurs a ciphertext expansion that is exponential in the multiplicative depth of the computation, and has a large communication overhead in practice even for small numbers of layered multiplications.

We present a new PIR construction, MulPIR, that improves on this state of the art in multiple ways. First, we show that the communication overhead of SealPIR can be significantly reduced at next to no cost in terms of computation. We further show that by using the multiplicative homomorphism of the underlying HE scheme, we can further

reduce the communi- cation overhead for databases with large entries, this time at an increased computation cost. While using the multiplicative homomorphism has been considered before, we are the first to show that, in combination with our other improvements, it enables a meaningful trade-off between communication and computation. In our experiments on Google Cloud Platform, we observed that this can reduce the total monetary server cost by up to 35%.

We also revisit the Gentry–Ramzan PIR scheme [31], which achieves optimal communication but has a high com- putation overhead. We show how to efficiently implement Gentry–Ramzan PIR even for large databases, and propose a new client-aided variant that allows for a tunable trade-off between communication and computation costs. We experi- mentally show that depending on the database shape, either MulPIR or client-aided Gentry–Ramzan PIR minimize the total server cost.

Finally, we turn to *keyword PIR* [13], a variant where the database size is much smaller than the query key domain. While regular PIR constructions assume dense databases and so their complexity depends on the index domain size, key- word PIR aims to achieve server computation cost that de- pends only on the actual database size as opposed to the key domain size. We present two constructions, based on two dif- ferent hashing schemes, and show that they enable another way to trade off communication and computation.

We implement all of our novel PIR schemes, as well as alternative approaches, and compare them experimentally on a wide range of applications, including anonymous messaging (as used in previous work [5]), private file download, and password checkup [63]. Due to space constraints, we present related work in Appendix A.

1.1 Our Contributions

Improving SealPIR communication. The most efficient (secure) single server PIR

constructions implemented in recent years [4, 5, 18, 23, 24, 30, 35, 46, 51, 64] are based on homomorphic encryption (HE) techniques and achieve sub-linear communication. Among those, the scheme that currently provides best implementation performance is

SealPIR [3, 5]. While theoretically this construction supports sub-linear communication complexity $O(d \cdot n^{1/d})$ leveraging d recursion levels, it comes with a large communication over- head in practice. This is due to the layered additive homo- morphic encryption approach: if the encryption scheme has

ciphertext expansion F, the PIR response will include F^{d-1} ciphertexts (where $F = 10$ in [3]). This yields communica- tion expansion of $O(F^2)$, which becomes unacceptable for databases with large entries.

Our first contribution reduces the communication of SealPIR by (1) using symmetric key encryption to reduce the upload size, (2) using modulus switching reduction tech- niques (as proposed for PIR in [9]) to reduce the value of F down to F 4, and (3) introducing a new oblivious expansion algorithm which can further halve the upload communication for some parameter sets. Therefore, our optimized SealPIR reduces by up to 75% the upload communication, and up to 80% the download communication.

Leveraging Multiplicative Homomorphism. When re- cursion is used in SealPIR, the download communication depends exponentially on the recursion level (the previous contribution reduced the basis of the exponential). Instead, we propose to use both the additive and *multiplicative* ho- momorphisms of the underlying HE scheme by doing one multiplication of encrypted values per recursion step. This reduces the size of the upload and download together from

$O(d \cdot n^{1/d}/N + F^{d-1})$ from the previous approach, where F is the number of

plaintexts needed to fit a single HE ciphertext, to $d \quad n^{1/d}/N \quad c(d)$, where $c(d)$ is the size of a ciphertext that supports d successive multiplications. Together with our

improvements to SealPIR mentioned above, the multiplica- tive homomorphism enables a highly communication-efficient PIR scheme, which we call MulPIR. For databases with large entries, its advantage over (optimized) SealPIR is already visible with low recursion level (download communication

reduced by 60%), and in fact we observe that $d = 2$ remains optimal for the database sizes we are interested in.

Gentry–Ramzan PIR: New Efficiency Trade-offs. The Gentry–Ramzan PIR construction [31] achieves optimal com- munication complexity for several settings but it pays with significant computational cost. Thus, our contributions here focus on ways to reduce this computation overhead, which includes new efficient techniques for encoding the server's database in CRT form needed for the computation in the scheme, new techniques for fast modular exponentiation needed to answer each query, as well as techniques for client- aided PIR that trade-off between communication and computation.

In this PIR protocol, the server database $\mathsf{D}_{i \ i \ [n]}$ needs to be encoded as $x = \mathsf{D}_i \bmod \pi_i$ for $i \ [n]$, where π_i are pair- wise co-prime integers. A naive application of the Chinese

Remainder Theorem requires computation at least quadratic in the size of the database. We leverage a divide-and-conquer modular interpolation algorithm [7] that enables us to achieve computation complexity $\tilde{O}(n \log^2 n)$. This technique also al- lows for pre-computation that can be reused for computations that use the same set of moduli π_i.

The main computation cost for each query on the server side is the modular

exponentiation, where the exponent is the encoded database, and the base and the modulus are chosen by the client. Our approach is to compute the exponentia- tion as a product of precomputed powers of the generator and to use Straus's algorithm [62] to do this efficiently. This enables a client-aided technique that allows us to improve the server's computation at the price of (small) additional work at the client. In particular, we observe that powers of the generator can be precomputed more efficiently by the client, by using the prime factorization of the modulus to reduce the exponent modulo the order of the group prior to exponen- tiating. This gives a new way to trade off computation and communication complexity for the protocol. In Section 6, we show evidence that providing several precomputed powers optimizes the server's work.

Keyword PIR for Sparse Databases. We consider the set- ting of sparse databases where the server's database is sparse in the index domain, and hence a client query corresponds to a keyword lookup. We present two constructions, both of which are based on hashing schemes. The first is based on simple hashing, where database elements are assigned to buckets us- ing a public hash function. The client then retrieves the bucket corresponding to their query. While the size of the buckets can generally get quite large, this is no concern in schemes that have a large plaintext size anyway, such as MulPIR. Our second constructions leverages cuckoo hashing [48] in a novel way. Unlike previous work [5], where cuckoo hashing was used to batch multiple client queries into one, we use it to compress the sparse server database into a dense domain. Cuckoo hashing guarantees that at most one element gets hashed to any bucket, at the cost of an increased (but constant) number of client queries. This variant is especially useful for Gentry–Ramzan PIR, where we have small plaintexts, and additionally can use CRT batching [36] to compress multiple client queries into one.

Comparison and Empirical Evaluation of PIR. We present a comprehensive comparison of the costs of PIR based on homomorphic encryption. This includes detailed concrete efficiency estimates for the ciphertext size and the computation costs

for encryption, decryption and homomor- phic operations of different HE schemes. We leverage these estimates to profile the efficiency costs of PIR constructions using the corresponding schemes when instantiated with and without recursion. We further present empirical evaluations of implementations of these PIRs with databases of different shapes (numbers of records and entry sizes). Our benchmarks demonstrate that for the majority of the settings constructions based on lattice based HE constructions, which could also offer multiplicative homomorphism, outperform in computa- tion other additive HE schemes. In terms of communication, additive HE solutions have advantage when the dominant communication cost is the download, e.g., in solutions with- out recursion for small databases with large entries, since these encryption provides best ratio between plaintext and ciphertext.

We evaluate our new PIR construction, MulPIR, that uses somewhat-homomorphic encryption (SHE) and enables a trade-off of computation for communication, and compare it against SealPIR.

In our experiments, Gentry–Ramzan PIR always achieves the best communication complexity but comes with a signifi- cant computation cost that can be prohibitive in some settings. However, we show that in terms of *monetary* cost, Gentry– Ramzan can outperform all other PIR approaches considered when database elements are small.

Finally, we apply our construction for keyword PIR to a *password checkup* problem, where a client aims to check if their password is contained in a dataset of leaked passwords, without revealing it to the server. Previous approaches to this problem [63] first reveal a k-anonymous identifier to the server to reduce the number of candidate passwords to compare against to k, and then apply a variant of Private Set Intersection to compare the current password against the k candidates. Our implementations of Gentry–Ramzan and MulPIR enable such lookups with communication *sublinear* in k, therefore either enabling better anonymity for the same bandwidth, or same anonymity and smaller bandwidth.