arXiv:2301.04370v1 [cs.CR] 11 Jan 2023

# 基于秘密共享的保序数据库加密

Dongfang Zhao 内
华达大学雷诺分校 美国

**摘要**

保序加密（OPE）问题最初由数据库社区于 2004 年提出，此前数据库即服务（DaaS）这一范式于 2002 年刚刚被命名。在过去二十年中，OPE 吸引了数据库、密码学和安全领域研究群体的极大兴趣；我们见证了 OPE 方案在理论和系统化方面取得的重大进展。所有现有的 OPE 方案都假设外包数据库被建模为一个单一的半诚实对手，该对手除了以可忽略的概率获知明文消息的顺序信息之外，不应了解任何其他信息。本文从新的视角研究 OPE 问题：我们不再将外包数据库建模为单一的半诚实对手，而是假设外包数据库服务由一个不串通服务器集群组成。这一假设具有现实可行性，因为所有主要云服务商都支持将多个数据库实例部署到相互隔离的子网络，甚至部署到不同的数据中心。该假设使我们能够通过在这些被假定不会串通的服务器之间采用秘密共享方案，设计一种新的无状态保序加密协议，即基于秘密共享的保序数据库加密（ODES）。我们将证明，ODES 能够保证最新的安全级别，即 IND-FAOCPA，并且其性能比当前最先进的方案高出数个数量级。

**ACM 引用格式：**
Dongfang Zhao。2023年。基于秘密共享的保序数据库加密。载于 *ACM* 计算机协会会议论文集（*Conference'17*）。ACM，纽约，美国纽约州，13 页。https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 引言

### 1.1 保序加密的起源

自具有开创意义的论文 [14] 发表于 ICDE'02 以来，二十年来，我们见证了数据库即服务（DaaS）的兴起与繁荣。截至本文撰写时，所有主要云计算服务商（例如亚马逊云科技、Google Cloud Platform、Microsoft Azure）都支持采用按需付费商业模式的 DaaS，使用户无需承担管理内部数据库的前期成本。从用户的角度来看，DaaS 可以被视为由云计算服务商维护的外包数据库 。与任何外包服务一样，外包数据库的安全一直是用户最关心的问题之一：外包数据库面临的威胁不仅来自外部攻击者，也来自内部对手，例如云计算服务提供商的开发人员和管理员。解决上述安全问题的一种研究途径，是在将用户的敏感数据上传到外包数据库之前对其进行加密。

数据即服务（DaaS）面临的一项关键挑战，是如何管理这些加密数据，例如构建索引。索引必须与明文关联，才能加快用户的查询和修改请求；然而，服务器能了解到的只有密文。为此，Agrawal 等人 [1]在 SIGMOD'04 中提出，在外包数据库中对明文进行编码，同时保留明文的数值顺序。论文展示了兼顾两者的可能性：所谓的保序加密（OPE）既能确保外包数据的机密性，也能保留其顺序。这项工作很快引起了数据库和安全／密码学领域的大量研究兴趣。

### 1.2 保序加密安全简史

在 EuroCrypt'09 上，Boldyreva 等人 [8] 提出了首个保序加密的安全定义。按照密码学领域的惯例，该定义基于加密方案的标准结构：(i) 安全目标是计算不可区分性；(ii) 威胁模型允许对手获取任意明文对应的多项式数量的密文，即所谓的选择明文攻击（CPA）；(iii) 通过基于模拟的归约证明可区分性是可忽略的。

遗憾的是，研究表明，在标准的选择明文攻击下，不可能实现不可区分性，因为选择明文攻击的定义过于严格：如果对手能够获知明文的顺序，就可以破坏该定义。好消息是，[8]提出了一种新的安全概念，即有序选择明文攻击下的不可区分性 （IND-OCPA）。除允许对手获知明文的顺序外，IND-OCPA 与 IND-CPA 同样严格。

IND-OCPA 提出后不久，Popa 等人 [26] 在 Oakland' 13 指出，IND-OCPA 无法抵御众所周知的频率攻击。频率攻击源于确定性密文；众所周知，在传统的选择明文攻击下，这类密文同样是不安全的。

随后，CCS'15 提出了一种更强的安全概念，即频率分析与有序选择明文攻击下的不可区分性 （IND-FAOCPA） [19]。此后有多种方案声称满足 IND-FAOCPA，例如 [27]。截至本文撰写时，IND-FAOCPA 仍是保序加密方案中最强的安全概念。

### 1.3 保序加密方案的系统研究

尽管密码学和安全领域的研究者投入了大量精力，从理论角度恰当地定义安全性并证明其安全，但数据库和系统领域的研究者同样

允许免费制作本作品全部或部分内容的数字或纸质副本，供个人或课堂使用，但不得为营利或商业利益而制作或分发副本，且副本必须附有本声明及首页的完整引文。对于非 ACM 所有的本作品组件，必须尊重其版权。注明出处后允许摘录。若要以其他方式复制、再版、发布到服务器上或分发给列表，须事先获得明确许可和/或支付费用。请向 permissions@acm.org 申请许可。
*Conference'17*，*2017*年7月，美国华盛顿特区© 2023年 计算机协会。ACM
ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

关注性能、成本等其他指标。如今，许多领先的云服务商都已开始支持加密数据库服务，例如 Microsoft Azure [2]，正如 SIGMOD'20 中所报道的那样。

VLDB'19 上的一篇评估论文 [7] 总结了截至 2019 年主要保序加密方案的优缺点。评估指标包括加密复杂度、比较复杂度、密文大小、I/O 成本和通信成本。根据论文报告的数据，尚无明显的赢家。

在 VLDB'21 上，Li 等人 [22] 提出了一种采用 128 位 AES 的新型频率隐藏保序加密方案（OPEA）。OPEA 在几乎所有方面都优于现有的保序加密方案：

- OPEA 具备 IND-FAOCPA 安全性；
- OPEA 只需在客户端与服务器之间进行固定次数的交互；
- OPEA 的插入和查询请求处理速度都明显快于其他方案；
- OPEA 需要 O($N$) 的客户端存储空间，其中 $N$ 表示不同明文的数量。

OPEA 唯一的局限在于客户端存储：作为一种有状态方案，OPEA 要求客户端维护一个本地表，以跟踪明文顺序。尽管重复明文只需存储一次，但其他方案（例如 POPE [27]）所需的客户端存储空间可能是常量。（不过，POPE 存在可能出现不可比较元素的问题）

## 1.4 本工作的动机与挑战

本工作的目标是消除 OPEA [22] 的缺点，同时保留其相较于现有方案的优势。这意味着我们希望同时实现高安全性和高性能。所谓高性能，是指较低的处理时间，这取决于计算时间、通信时间和 I/O 时间。显然，OPEA 的 O($N$) 客户端存储空间是一个性能瓶颈。

另一方面，利用当前可用的密码学基本组件以及我们关于系统优化的传统认知，同时实现强大的安全保证和高性能极具挑战性。如上文所述，由于客户端与服务器之间存在有状态协调，客户端存储是必需的，因为人们普遍认为，有状态机制在维护元组顺序方面能够实现更高的效率。截至本文撰写之时，我们仅了解到一项采用无状态方法的工作 [30]；但该工作未报告实验结果。此外，128-AES 被广泛认为是最高效且最安全的对称加密算法之一，

如今的加密方案；因此，通过升级或优化密码学子系统来提升 OPEA 的性能不太可能奏效。如果我们希望在不牺牲安全级别的前提下进一步提升性能，就需要采取更为彻底的措施。

## 1.5 拟议解决方案

本研究通过采用秘密共享原语，提出了一种新的保序加密方案。虽然从广义上说，秘密共享可以被视为一种加密方案，但其内部机制与单节点加密截然不同。我们不再将加密后的测试数据放在单台机器上，而是假设有一个不会串通的机器集群。密文现在被分散到多个

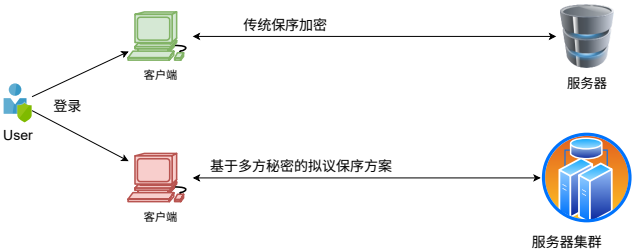


**图1：拟议的多方秘密方案与传统单节点加密方案。**

数据库服务器上，作为秘密份额；未经数据所有者授权，无法对其解密。因此，机密性得以实现。

为实现明文的顺序性，秘密共享原语应允许我们在不要求服务器共享其本地秘密份额的情况下比较相应的明文值。如果仅通过对秘密份额进行一些本地计算就能实现这种比较，那么得到的方案将是无状态的，并能节省一些输入/输出开销。

正如我们将在后续章节中展示的那样，某种特定类型的秘密共享方案能够同时实现编码数据的机密性和顺序性。为更具体地说明，图 1 展示了传统保序加密方案与我们提出的方案之间的整体差异，我们将后者命名为基于秘密共享的有序数据库加密 （ODES）。

## 1.6 贡献

总而言之，本研究在技术上作出了以下贡献：

- 我们提出了首个面向采用秘密共享的外包数据库的无状态保序加密方案，即基于秘密共享的有序数据库加密（ODES）；
- 我们证明了 ODES 能够保证较强的安全级别，即 IND-FAOCPA；
- 我们设计了多种数据库协议，以利用所提出的 ODES 方案；
- 我们基于 SQLite 数据库实现了 ODES 方案和数据库协议；并且
- 我们在 10 节点集群上，使用 TPC-H 基准测试和三个实际应用，将 ODES 与最先进的方案进行比较，从而对其进行了全面评估。

# 2 背景与相关工作

## 2.1 保序加密

保序加密（OPE）的概念最初由数据库社区提出 [1]。其动机显而易见：我们如何才能在外包数据库中同时实现敏感数据的机密性并保留其顺序？机密性这一部分显而易见，对顺序的要求也有充分的理由：数据库系统通常会构建索引，以加快查询和插入请求的处理；而能够对外包数据集进行排序或排列，是实现这一目标的关键。

早期实现这两个目标的方案相当直接：借助某种统计分布对明文进行编码，使编码后的值与明文保持相同的顺序。这种方法存在一些问题。首先，编码后的值是确定性的。这意味着，如果对手能够以某种方式获得某些选定明文的编码，即发动所谓的选择明文攻击（CPA），那么这种编码就无法保证安全。有人可能会认为，在外包数据库中，我们不需要达到选择明文攻击安全性那么高的安全级别；但从安全角度来看，实用的数据库系统至少应始终提供选择明文攻击安全性 [2]。

接下来，我们面临选择明文攻击安全性与有序编码之间的两难：选择明文攻击安全性意味着密文具有随机性，因而无法保留明文的顺序。解决方法是引入某种函数，使数据库无需依赖密文的原始值即可对加密元组排序，这种加密称为 揭序加密 （ORE） [9]。相应地，人们提出了一种新的安全概念，允许对手获知明文的顺序，由此产生了所谓的有序选择明文攻击下不可区分性（IND-OCPA）。显然，计算顺序值的方法有多种，但可分为两类：（i）有状态方案，客户端与服务器协同维护加密记录在数据库中的顺序信息 [26]，；（ii）无状态方案，顺序信息可即时获取 [30]。文献中的大多数保序加密（OPE）工作都侧重于有状态方法；正如我们将在后文看到的，所提出的 ODES 是一种无状态方案。

后来发现，揭序加密和 IND-OCPA 也面临新的问题：许多 IND-OCPA 方案 [8, 26] 容易受到利用查询访问模式的攻击。为此，研究者提出了一种更新的概念，即频率分析有序选择明文攻击下的不可区分性（IND-FAOCPA）。文献中已提出多种 IND-FAOCPA 方案，例如 [19, 22, 27]。一篇较新的评估论文 [7]报告了若干最热门保序加密方案的性能，其中也包括本文尚未提及的方案： [10, 11, 21]。截至本文撰写时， [22] 提出的保序加密方案在几乎所有指标（例如客户端存储、查询轮数）上都表现最佳，因此我们将主要把本文提出的 ODES 与 [22] 提出的协议进行比较。

## 2.2 秘密共享

秘密共享方案（SSS）的思想很简单：给定的明文 $pt$ 会被转换为一组编码字节 $ct$，只有特定的子集 才能重构原始 $pt$。SSS 的目标是降低明文泄露的风险；恶意对手无需攻破明文持有者，而是必须攻破多个实体，才能在任何份额持有者察觉攻击之前得手。即使面对较弱的攻击（假设对手仅为半诚实对手），将秘密份额分散给更多参与方，也能提高窃听攻击成功的门槛。

SSS 可以通过子集大小进行调整。形式上，a $t$-out-of-$n$ 门限秘密共享方案（TSSS）定义如下。

定义 1（门限秘密共享方案）。门限秘密共享方案由两个算法组成：

- 共享：一种随机化算法，以明文 $pt$ 为输入，并返回一个由份额组成的序列 $S = (s_1, \ldots, s_n)$。
- 重构：一种确定性算法，输入至少包含 $t$ 个份额的集合，并返回明文。

数值 $t$ 称为门限秘密共享方案的门限。设 $U$ 是 $n$ 个份额的子集，大小为 $t$， $|U| \geq t$ 且 $U \subseteq \{s_1, \ldots, s_n\}$，我们要求门限秘密共享方案满足以下性质：

$$Reconstruct(U) = pt.$$

正如我们将在下一节 §2.3 中看到的，门限秘密共享方案的定义会导致其安全定义与传统加密方案略有不同。

门限秘密共享方案的经典例子是 Shamir [29], 提出的方案，其中秘密通过一个 $(t-1)$ 次多项式揭示。从本质上说，每个份额都能通过拉格朗日多项式重构未知数中特定次数项的系数。除了 Shamir 的构造外，还有其他方案。Ito等人 [17] 提出了复制式秘密共享方案，该方案基于有限域，其中每个份额都是一个向量。复制式秘密共享的一个优点是其线性性：本地份额的加法和减法分别等于明文的加法和减法。复制式秘密共享的一个更简单的变体是加法秘密共享，其中每个份额都是一个标量值，且门限 $t$ 设为 $n$。事实上，这是我们提出的 ODES协议的一个构件；我们将在 §3.3 中看到，加法秘密共享的线性性如何使我们能够保留明文的顺序。

SSS 与安全多方计算（MPC） [3], 有着密切的联系，后者历史悠久 [32]。MPC 的目标比 SSS 更宏大：除了确保明文保密，我们还希望仅通过接触多个参与方持有的编码数据，计算原始明文的任意函数。所谓的混淆电路 [32],解决了这个问题，其思路相当简单：我们可以让每个参与方使用自己的私钥对输入进行编码，打乱加密后的密文，然后枚举所有密钥来解密结果。由于我们假设加密方案是安全的，因此只有将正确的私钥组合应用于某个混淆输出，才能揭示结果。这确实是一个可行的解决方案，至少在理论上如此；但在实际应用中，电路可能呈指数级增长，导致效率问题。还有许多效率更高的 MPC 方案，例如 [4, 20, 24, 33]。

## 2.3 可证明安全

在应用中采用加密方案时，最好能以可证明的方式展示其安全性。形式上，我们需要确定给定加密方案可证明安全所需的以下三个重要要素：安全目标、威胁模型和假设。安全目标阐明应用遭受攻击时所期望达到的效果；威胁模型说明对手能够如何实施攻击，例如能够收集哪些明文/密文信息，以及攻击在资源和时间上的限制；假设则列出对该密码方案的子系统或组件所作的具体假定，这通常是安全性证明（例如归约）中的重要构件。安全目标和威胁模型通常统称为安全定义 。

一种广受认可、兼顾效率与安全性的安全定义是：对手可以发起选择明文攻击 （CPA），定义如下。

定义 2（选择明文攻击）。给定安全参数 $n$，即密钥的比特串长度，对手最多可以获得 $poly(n)$ 组明文-密文对 $(m, c)$，其中 $m$ 由对手任意选择，$poly(\cdot)$ 是关于 $n$ 的多项式函数。利用这些信息，对手试图解密一个未包含在已知密文多项式数量中的 $c_t$。

多项式这一要求仅出于实际考虑，因为我们通常假设对手只能运行多项式时间算法，无法拥有无限资源。因此，我们希望设计具备选择明文攻击安全性的加密方案：即使对手 A 能够获得这些额外信息，A 解读密文的能力也不应比随机猜测好多少，至多只高出一个极小的概率。为量化这一极小概率的程度，可忽略函数定义如下。

定义 3. 如果对于所有多项式 $poly(n)$，不等式 $\mu(n) < \frac{1}{poly(n)}$ 在 $n$足够大时成立，则称函数 $\mu(\cdot)$ 是可忽略的。

为完整起见，我们列出以下关于可忽略函数的引理，这些引理将在后续章节中用到。我们陈述这些引理而不提供证明；证明可见于密码学或复杂性理论的入门教材。

引理 1（两个可忽略函数之和仍是可忽略函数）。设 $\mu_1(n)$ 和 $\mu_2(n)$ 均为可忽略函数。那么，$\mu(n) = \mu_1(n) + \mu_2(n)$ 是一个可忽略函数，其定义为 $(n) \stackrel{def}{=} (n) + (n)$。

引理 2（多项式函数与指数函数之商是可忽略函数）。$\frac{poly(n)}{2^n}$ 是一个可忽略函数。也就是说，$\exists N \in \mathbb{N}, \forall n \geq N$：$\frac{poly(n)}{2^n} < \frac{1}{poly(n)}$，其中 $\mathbb{N}$ 表示自然数。

证明所提出的加密方案（例如 IND-CPA）安全性的经典方法是通过归约 [23]。通常，将攻破该方案归约为一个困难数学问题，也就是说，如果可以攻击该方案，那么就能高效地求解这个数学问题。换言之，该加密方案并不比这个数学问题更容易解决。我们将该方案建模为一个子程序，并对其输入进行模拟，使对手无法分辨自己是在参与攻击，还是在帮助求解数学问题的子程序中。虽然也可以进行正向证明，但更常用的技巧是利用矛盾：假设对手能够以不可忽略的优势区分某些设计好的实验，那么归约就会以不可忽略的概率高效地求解一个被认为是难解的困难数学问题，从而导致矛盾。

遗憾的是，研究证明，在保序加密的背景下，传统的 IND-CPA 是不可能实现的 [8]。因此，密码学界提出了一种较宽松的概念，称为有序选择明文攻击下的不可区分性（IND-OCPA） [8]。然而，研究表明， [5] 可以利用访问模式发动有效攻击，以攻击 IND-OCPA 安全性。这个问题的根源在于早期保序加密方案中的确定性密文。事实上，众所周知，确定性加密方案不可能对选择明文攻击提供安全性。因此，现代保序加密方案都采用随机化，这意味着密文无法直接比较，必须通过间接方式进行比较。这种间接比较通常称为揭序加密（ORE） [9]，它将 OPE 的原始概念加以推广。从更广义的角度来看，人们提出了一些所谓的隐藏频率保序加密方案 [19, 27]。相应地，一种新的安全概念也随之提出，即频率分析有序选择明文攻击下的不可区分性（IND-FAOCPA） [19]。IND-FAOCPA 是该领域最新的安全定义，而我们提出的 ODES 方案满足 IND-FAOCPA 安全性。

上述对可证明安全的回顾假设密文是单一实体，没有考虑密文为集合的情形，而秘密共享正属于这种情形。秘密共享的可证明安全采用了略有不同的方法，因为其定义中额外假设对手无法访问所有份额。诚然，如果能证明整组份额是安全的，那么其中任意子集也安全；但证明秘密份额语义安全时，更常用的方法是通过可互换库 [28]。其核心思想是将方案建模为一个库，其输入可以是 $L$ 或 $R$ 明文输入；随后，证明将展示，通过一系列可互换操作，具有 $L$ 输入的库最终看起来与具有 $R$ 输入的库完全相同。我们将在 §3.5 中看到这种技术的应用。

## 3 保序数据库加密与秘密共享

### 3.1 概述

基于秘密共享实现保序的直观思路很简单：我们利用数据库服务器集群的多节点特性，在不泄露任何明文的同时，保留明文之间的比较顺序。也就是说，我们以某种方式将原始明文拆分为多个份额，并将每个份额分配给数据库集群中的不同服务器。假设集群中的节点彼此不串通；可以通过将服务器部署在不同的子网络或不同的数据中心来实现这一点。通过聚合每台服务器上的局部函数，可以恢复并更新明文的顺序。

更具体地说，图2通过一个过度简化的场景展示了这一思路：有两台服务器可用于存储编码后的有序数据。假设数据所有者想要将余额表保存到远程数据库服务中。该表十分简单，是一个键值存储，其中业务数据以年月为键，以美元金额为值。

两个数据库服务器分别存储一些看似随机的数字，以帮助对真实金额保密。我们必须能够从服务器存储的份额中重构出原始明文，否则用户就会丢失数据。在这个例子中，只需将不同数据库服务器上的份额相加，就能重构出原始余额。它

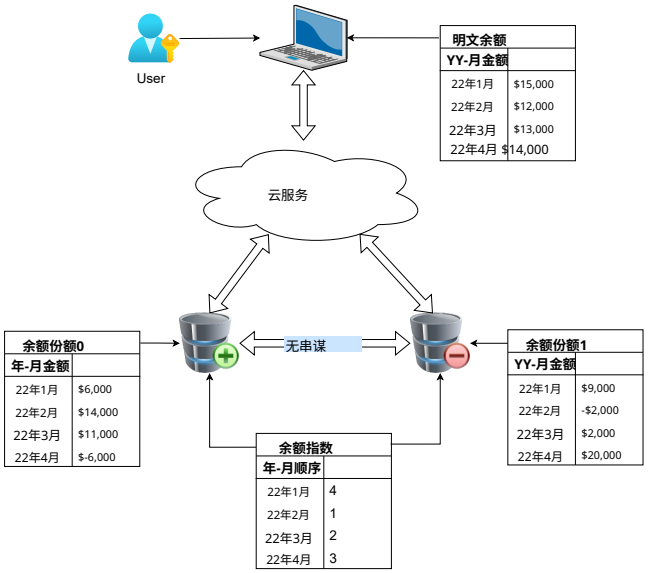


**图2：两个保序份额的示例。**

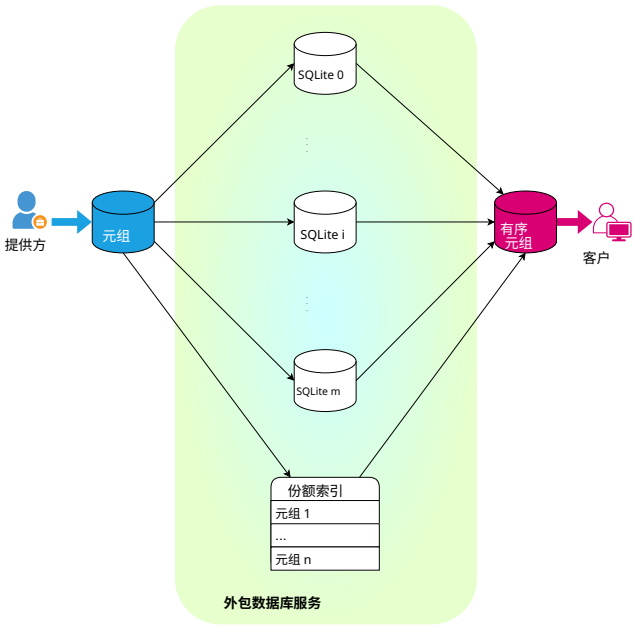


**图 3：在 SQLite 外包数据库中使用秘密份额保持元组顺序的所提架构。**

需要注意的是，重构只能在客户端进行，因为外包数据库并非完全可信。也就是说，数据库服务器不应共享其本地数据。我们稍后会正式定义"完全可信"的含义；但现在，我们先以非技术性的方式进行讨论。

另一项重要信息是用于跟踪明文顺序信息的索引元数据。插入新记录时，索引表将十分有用：它使我们能够通过二分查找，在数据库中找到新插入记录的正确顺序位置。此外，它还能提高与顺序相关的查询的效率。

这里再举一个例子，说明如何通过两组秘密共享份额进行明文比较。假设用户最近获得了一条新记录 (22年5月, $ 10,000)，并想知道余额是否高于上个月（22年4月）。请注意，ODES 是一种无状态协议，因此客户端不能直接将 $10,000 与 (22 年 4 月 , $14,000) 进行比较，因为后者在秘密共享给两台数据库服务器后便不复存在。客户端的做法是将 $11,000 拆分成两个随机数，例如 $3,000 和 $8,000，将它们与密钥组合后发送给两台数据库服务器。在图 2 的示例中，加法服务器收到 (22 年 5 月 , $3,000) 记录，减法服务器收到 (22 年 5 月 , $ 8,000) 记录。随后，两台服务器分别对 22年4月和 22年5月的记录进行本地计算；例如，在加法服务器上，它会计算以下增量 ($\delta_0$, 3000-(-6000) = 9000)。类似地，减法服务器会计算 ($\delta_1$, 8000-20000 = -12000)。然后，两台服务器都会将各自的本地 $\delta_j$ 广播给所有其他服务器。请注意，这种共享不会泄露订单以外的任何信息，因此是允许的（不同于秘密共享份额本身）。现在，每台数据库服务器都获得了所有 $\delta_j$，随后对它们进行聚合。例如，加法服务器计算 9000+(-12000) = -3000 < 0，这意味着 22年5月的余额低于 22年4月。服务器还可以通过在索引文件上进行二分查找，相应地更新索引元数据；在此例中，(22年5月, 1) 将被插入索引表，且一些现有订单的值将递增 1。

## 3.2 架构

如图 3 所示，我们设想有一个由 $m$ 外包数据库服务器组成的集群，这些服务器 (i) 不共享其本地数据，且 (ii) 可以访问用于秘密份额订单的索引。由于我们的系统原型是使用 SQLite 实现的，因此假设有 $m$ 个 SQLite 实例。只要至少有一个 $m$ SQLite 实例不与其他实例串通，外包数据就是安全的，从而解决了文献中所谓的 恶意多数 问题。这意味着数据库越多，安全级别越高；但代价是系统的计算和输入/输出开销也会增加。

该架构还区分了两种不同的用户角色：数据提供方（左侧）和数据客户（右侧）。两者的主要区别在于，提供方可以修改数据库集群中的秘密份额，也可能更新索引元数据；而客户只能执行只读查询。我们将看到，在所提出的协议中，份额索引将支持修改操作和只读查询。

## 3.3 基本组件

现在，我们可以正式介绍 ODES 中用于实现保序的基本组件。这些基本组件将在后续（§3.4）的各种协议中用作构建模块。我们假设明文是一个由 $l$ 位组成的字符串，并且总共有 $m$ 个数据库服务器。

共享。 我们首先介绍<style id='3'>共享 函数，该函数将给定的明文拆分为 $m$ 秘密份额。

定义 4（共享）。 共享 () 函数定义为

$$Share : \{0,1\}^l \to \left(\{0,1\}^l\right)^m,$$
$$pt \mapsto \{s_i\}, 0 \le i < m,$$

其中， $s_i$是一个随机数，当 $i \neq 0$ 时， $s_0$ 计算如下

$$s_0 := \sum_{i=1}^{m-1} s_i.$$

重构。 重构 函数是共享的逆过程。

定义 5（重构）。 重构()函数定义如下

$$Reconstruct : \left(\{0,1\}^l\right)^m \to \{0,1\}^l,$$
$$\{s_0, \ldots, s_{m-1}\} \mapsto \sum_{i=0}^{m-1} s_i.$$

比较。 由 $m$ 台服务器组成的集群旨在通过对两组秘密份额进行本地计算来比较两个明文。顾名思义，服务器不得泄露其本地份额。令$S^L$ def= $\left\{s_0^L, \ldots, s_{m-1}^L\right\}$表示第一个明文$L$的秘密份额，并令 $S^R$ def= $\left\{s_0^R, \ldots, s_{m-1}^R\right\}$表示第二个明文 $R$的秘密份额。

定义 6（比较）。比较是一个函数：

$$Compare : \left(\{0,1\}^l\right)^m \times \left(\{0,1\}^l\right)^m \to \{-1,0,1\},$$
$$(S^L, S^R) \mapsto \begin{cases} -1, & L < R \\ 0, & L = R \\ 1, & L > R \end{cases},$$

其中，*Compare* 被赋予与下方求和式相同的算术符号

$$\sum_{i=0}^{m-1} \left(s_i^L - s_i^R\right).$$

请注意，上述等式中的各项由每个数据库服务器独立计算。服务器将各自的局部差值广播给整个集群，以便每台服务器都能确定明文的序数信息。

## 3.4 协议

本节介绍三种 ODES 协议。这些协议会用到多种基本组件。为使协议内容完整自洽，我们会介绍这些基本组件的一些高层实现细节；在介绍时，我们会在伪代码中添加注释，提醒读者此处调用了某些基本组件（例如算法 1 中的第 3 行）。按照密码学约定，我们使用 := 表示确定性赋值，使用 ← 表示从随机化算法或分布中均匀采样得到的值。

*3.4.1* 初始化。系统首次部署时，我们假设数据提供方拥有一个初始数据集列表，这些数据集将被编码并上传到远程数据库服务。拟议的 ODES 系统投入运行前，必须完成以下几项任务：

- 数据提供方根据本地明文构建初始索引文件；
- 每个明文字段都被分解为一个份额列表；
- 相同明文的份额应进行随机化，并分发到不同的数据库服务器。

所有这些任务都在初始化阶段完成，如算法 1 所示。对于第 $i$ 条记录 $r$，客户端（即图 3 中的提供方）将其拆分为 $m$ 个部分（第 3–6 行）：最后的 $m-1$ 个部分$s_i[1: j]$ 是随机值，而第一个部分则计算为 $r$ 与 $\sum s_i[1$ 之差：$j]$。为了给编码后的密文增加更多随机性，我们还对 $m$ 个份额应用置换（第 7 行）。在第 9 行，客户端将每个 $m$ 份额发送到不同的数据库服务器；在第 10 行，服务器接收该份额并将其插入本地份额中。处理完记录 $r$ 后，在第 12 行更新索引元数据。完成所有 $n$ 条记录后，客户端在第 14–16 行将索引表广播给所有数据库服务器。

**算法 1：** ODES 初始化

**输入：** 一个基数为 $n$ 的关系 $R$，即有 $|R| = n$ 个明文元组；一组 $m$ 不串通节点 $N$；返回随机数的函数 $Rnd()$；

**输出：** 每个节点 $N_j$ 持有一组密文份额 $R_j$，$|R_j| = n$；索引 $idx$ 持有 $R$ 的订单；

```
1  对于 i = 0; i < n; i++ 执行
2      r = R[i]
3      对于 j = 1; j < m; j++ 执行                    // 共享()
4          s_i[j] ← Rnd()
5          s_i[0] := r − s_i[j]
6      end
7      置换元素于 s_i
8      对于 j = 0; j < m; j++ 执行
9          发送 s_i[j] 至 N_j
10         R_j[i] := s_i[j]                           // 在服务器 N_j 上
11     end
12     更新 idx
13 结束

14 对于 j = 0; j < m; j++ 执行
15     发送 idx 至 N_j
16 结束
```

算法 1 的复杂度如下。第 3–6 行需要 O($m$) 步，第 7 行需要 O($m!$) 步，第 8–11 行需要 O($m$) 步。因此，第 1–13 行需要 O($nm!$) 步。第 14–16 行显然需要 O($m$) 步，可以忽略。因此，算法 1 的总复杂度为 O($nm!$)。尽管渐近复杂度中的 O($m$) 因子看起来代价很高，但在实际应用中通常不是问题，因为 $m$ 的取值相对较小，例如 2、4 和 8。

*3.4.2* 插入。 插入协议假设数据库服务器集群已持有秘密份额，并将接收客户端发来的新记录。在此情境下，协议包含两个阶段：(i) 客户端准备新记录的秘密份额

并将其发送给服务器集群；(ii) 服务器更新其本地份额以及用于排序信息的索引元数据。我们在算法 2 中展示这两个阶段。

客户端协议。 与初始化协议类似，客户端在第1–4行将给定记录 $r$ 拆分为 $m$ 个份额。随后，客户端对这些份额进行置换，并将每个份额分别发送给不同的数据库服务器（第5–8行）。

服务器协议。 服务器始终会将收到的份额插入其本地表中，如第9行所示。第10–25行通过二分查找为新记录确定正确顺序，并据此更新索引。第13行计算收到的份额与对应明文为中位数的份额之间的差值。第14行使所有数据库服务器都能获知这些差值，从而决定移至排序后份额的较小一半还是较大一半（第16–19行）。如果数据库中存在重复值，则协议最终会执行到第21行（$\delta = 0$）；否则，第24行会设置新记录的顺序。

复杂度分析。 算法 2 的复杂度如下。第 1–4 行需要 O($m$) 步，第 5 行需要 O($m!$) 步，第 6–8 行需要 O($m$) 步。因此，客户端总共需要 O($m!$) 步。第 11–12 行和第 16–21 行意味着需要迭代 O($\log n$) 次。第 14 行需要 O($m$) 步。第 21 行和第 24 行需要常数步。因此，服务器总共需要 O($m \log n$) 步。

*3.4.3* 查询。 我们将顺序谓词表示为 $OP$，它是查询的一部分。查询协议分为三个阶段：

- 在查询协议的第一阶段，客户端（即图 3 中的提供方）只需将谓词广播给所有数据库服务器。
- 第二阶段在每台数据库服务器上进行，各服务器处理符合 $OP$ 谓词的本地份额，并将其发回客户端。
- 在第三阶段，客户端（即图 3 中的客户）收集所有符合条件的份额，并重构明文。

这三个阶段总结于算法 3 中。第 5–8 行检查每个已索引的订单，以确定本地份额是否符合条件；如果符合，服务器 $N_j$ 便将其本地份额 $R_j[k]$ 回复给客户端。第 13–19 行通过将份额聚合到各个元组中，重构明文列表。

算法 3 的复杂度如下。对于服务器，总体复杂度为 O($n$)（第 5–10 行）。对于客户端，第 1–3 行需要 O($m$) 步。第 11 行本身需要 O($nm$) 步，因为 $K \le n$。第 13–19 行也需要 O($nm$)。因此，客户端的总体复杂度为 O($nm$)。

*3.4.4* 删除与修改。 删除 ODES 比插入更为直接。第一阶段是要求每个数据库服务器删除相应的本地份额。第二阶段是在索引文件中删除待移除记录的顺序信息，并通过将顺序值减一来更新索引文件，其中减一操作针对顺序大于被删除记录顺序的记录。本会议论文省略了该协议的详细描述。

对 ODES 的修改可以通过先删除记录，再插入新值来轻松实现。本文不再正式描述修改协议。

**算法2：** ODES插入

**输入：** 一条新记录 $r$，其记录标识符为 $rid$，将被插入关系 $R$, $|R| = n$中； $R$ 无法直接访问，只能从 $R_j$ 重构，其中每个都存储在节点 $N_j$, $1 \le j \le m$上；一个全局索引 $idx$，其中保存了 $R$ 的顺序信息；

**输出：** $R_j$ 已更新，新增了一条记录； $Inx$ 是已更新，以反映新的顺序；

```
/* 在客户端                                                    */
1 对于j = 1; j < m; j++ 执行                                    // 共享()
2     s[j] ← Rnd()
3     s[0] := r − s[j]
4 结束
5 置换 s中的元素
6 对于j = 0; j < m; j++ 执行
7     发送 s[j] 至 N_j
8 结束

/* 在服务器 N_j */9 R_j := R_j ∪ {s[j]} // 更新数据共享
10 lo := 0, hi := n − 1, δ := 0 11 while lo < hi do // 二分查找
r 12     mid := ⌊(lo+hi)/2⌋ 13     δ_j := s[j] − R_j[idx[mid]] 的排
序位置 14 广播 δ_j 至 N 的 // 共享 δ 没问题，但 s[j]
15     δ := Σ_{j=0}^{m} δ_j 16 if δ > 0 then // 比较()
17         lo := mid + 1 18 else if δ < 0 then // 比较()
19         hi := mid − 1 20 else 21 insert(idx, mid, rid) //
在 R 中找到重复值 22 end 23 if δ ≠ 0 then 24 insert(
idx, low, rid) // 在 R 中未找到现有值 25 end
```

## 3.5 安全性分析

*3.5.1* 威胁模型。假设数据库服务器是半诚实的，这意味着它们无意发动主动攻击，例如窃取秘密份额。不过，这些服务器可能会窃听消息并被动共享信息；例如，服务器可能设法收集任意明文的秘密份额，这实质上构成了选择明文攻击（CPA）。

然而，在保序数据机密性的语境下，由于排序信息会让对手做出远胜于随机猜测的判断，因此不可能实现传统的选择明文攻击不可区分性（IND-CPA）。因此，我们假设对手服务器感兴趣的是了解明文订单以外的信息。这在文献中也称为 IND-OCPA。之后，有人提出了一种更强的安全概念，称为

**算法 3：** ODES 查询

**输入：** 一个顺序谓词 $OP(\cdot)$，基于该谓词，从一组记录从 $R$, $|R| = n$中返回；每个共享的 $R_j$ 存储在节点 $N_j$, $1 \le j \le m$;

**输出：** 客户端收到一组记录 $r$，其顺序 $r_{ord}$满足 $OP(r_{ord}) = True$;

```
/* 在客户端                                          */
对j = 0; j < m; j++ 执行
    将 OP(·) 发送到 R_j
结束

/* 在服务器端 N_j                                    */
从客户端接收 OP(·)
对于i = 0; i < n; i++ 执行
    如果 OP(i) 则
        k := idx[i]
        将 R_j[k] 发送给客户端
    end
结束

/* 在客户端                                          */
从每个 m 节点收集 K 值
结果 = []
对于k = 0; k < K; k++ 执行
    val := 0
    对于j = 0; j < m; j++ 执行                    // 重构()
        val += R_j[k]
    end
    res.append(val)
end
return res
```

频率分析有序选择明文攻击下的不可区分性（IND-FAOCPA）[19], ，这意味着访问模式不会泄露。

*3.5.2* 安全假设。 我们假设分组密码（例如 AES）在实际应用中可用作伪随机数生成器（PRG）。我们需要这一假设，因为随机数（即函数 $Rnd()$）对我们的协议至关重要（见算法 1–3）。这是应用密码学文献中广泛接受的假设。

*3.5.3* 两个参与方之间的不可区分性。令 $m_L$和 $m_R$表示两个长度相同的不同明文消息 $l$。根据算法 1，当加密 $m_L$时，$s_0 \leftarrow Rnd()$ 和 $s_1 := m_L - s_0$。对于 $m_R$，也可以类似地定义该函数。由于我们假设两个节点 $N_0$和 $N_1$不会串通，因此只能访问一个份额。形式化地，我们定义以下函数

$$ODES_L^2(m_L, m_R, i \in \{0,1\}) \stackrel{\text{def}}{=} \begin{cases} s_1 = Rnd(), & i = 1 \\ s_0 = m_L - s_1, & i = 0 \end{cases}$$

and

$$ODES_R^2(m_L, m_R, i \in \{0,1\}) \stackrel{\text{def}}{=} \begin{cases} s_1 = Rnd(), & i = 1 \\ s_0 = m_R - s_1. & i = 0 \end{cases}$$

目标是证明，当对手 A 向 ODES预言机提交两条消息 $m_L$ 和 $m_R$时，A 无法区分份额 $s_i$来自 $ODES_L^2$还是 $ODES_R^2$, $i \in \{0, 1\}$。如果访问的是 $s_1$，那么显然无法区分 $s_1$，因为 $ODES_L^2$和 $ODES_R^2$都会通过 $Rnd()$ 返回随机数。如果访问的是 $s_0$，则需要证明 $m_L - Rnd()$ 和 $m_R - Rnd()$ 不可区分。事实确实如此，因为 $Rnd()$ 被假定在消息空间中均匀分布，将其输出与 $m_L$或 $m_R$相加会使结果变得无法辨认。从技术上说，A 区分两个随机化数的概率为 $\frac{1}{2^l}$，这是关于 $l$（即明文的比特长度）的一个可忽略函数。换言之，由 $ODES_L^2$和 $ODES_R^2$生成的 $s_0$ 份额可以互换。

*3.5.4* 任意参与方之间的不可区分性。 在一般情况下， $t$个节点拆分明文，秘密可以在扩展函数中类似地定义：

$$ODES_L^t(m_L, m_R, i \in [0,t)) \stackrel{\text{def}}{=} \begin{cases} s_i = Rnd(), & i \neq 0 \\ s_0 = m_L - \sum_{i=1}^{t-1} s_i, & i = 0 \end{cases}$$

and

$$ODES_R^t(m_L, m_R, i \in [0,1)) \stackrel{\text{def}}{=} \begin{cases} s_i = Rnd(), & i \neq 0 \\ s_0 = m_R - \sum_{i=1}^{t-1} s_i. & i = 0 \end{cases}$$

当 $i \neq 0$时， $s_i = Rnd()$ 的值无法在 $ODES_L^t$ 和 $ODES_R^t$ 之间区分。当 $i = 0$时，重复的 $Rnd()$ 只能进一步混淆 $m_L$ 和 $m_R$ 消息（注意，对一组可忽略函数进行算术运算，结果仍然是可忽略函数），从而使 $s_0$ 可以互换。

*3.5.5 IND-FAOCPA*。 为了证明 ODES 满足 IND-FAOCPA，我们需要证明两个特性：（i）泄露访问模式并不能帮助对手区分两个密文；（ii）明文消息对序列始终按一定顺序排列，使对手无法获得有助于区分过程的额外信息。我们将证明 ODES 满足这两项特性。完整证明超出了这篇会议论文的范围，因此我们将分析限定在描述层面。

频率分析。 每当明文与数据库服务器集群交互时，都会通过*Share()* 原语（§3.3）对明文进行全新的分解。共享秘密重复出现的概率可以忽略不计，为 $\frac{1}{2^{l \cdot m}}$，其中 $l$ 表示消息空间的比特串长度， $m$ 表示数据库服务器的数量。直观地说，对于同一明文，生成一组重复份额的可能性极低。因此，对手无法通过研究频率或访问模式发动成功攻击。

有序明文。 请注意，所提出的 ODES 方案有一项假设：各方不串通。这意味着数据库服务器除了本地份额的序数信息之外，无法获知明文。然而，正如我们在 §3.5 中讨论的，这些份额在数据库服务器的本地表中与随机数无法区分。例如，在图2中，

尽管正值数据库服务器知道 22年3月的余额高于 22年2月（通过收集 $\delta$s），但服务器无法得知 22年3月或 22年2月的实际余额（除非两台服务器完整地交换本地份额，而 ODES 不允许这样做）。单台数据库服务器上的本地份额顺序并无帮助；例如，连服务器自身都无法达成一致（更不用说该顺序是否与明文一致了）：正值服务器认为 22年3月的本地份额（$11,000）低于 22年2月的（$14,000），而负值服务器却认为 22年3月的本地份额（$2,000）高于 22年2月的（$-2,000）。

# 4 评估

## 4.1 系统实现

我们使用约 1,200 行 Python 代码和 Bash 脚本实现了所提出的 ODES协议，代码将发布于 https://github.com/hpdic/odes。我们选择轻量级的 SQLite 作为本地数据库实例。请注意，SQLite 是基于文件的数据库，不支持网络访问。因此，我们通过 *paramiko* 库，在远程 SQLite 实例之间实现通信层，以支持安全数据传输和远程查询调用。其中一些最重要的库和依赖项包括：*Python* 3.8.0、*SQLite* 3.31.1、*numpy* 2.21.0、*paramiko* 2.12.0、*scp* 0.14.4 和 密码学 39.9.0。

## 4.2 实验设置

*4.2.1* 测试平台。我们在 CloudLab [13]托管的 10 节点集群上部署了所提出的 ODES协议和最新的保序加密方案 [22] 与 SQLite [15]。每个节点配备两颗 32 核英特尔至强金牌 6142 CPU、384 GB ECC DDR4-2666 内存和两块 1 TB 固态硬盘。操作系统映像为 Ubuntu 20.04.3 LTS，页大小为 4 KB。所有服务器均通过 1 Gbps 控制链路（戴尔 D3048 交换机）和 10 Gbps 实验链路（戴尔 S5048 交换机）连接。评估中我们仅使用实验链路。

具体而言，我们将集群中的 10 个节点命名为 *node0–node9*。客户端运行在 *node0* 上，ODES 服务器运行在 *node1–node8* 上，保序加密服务器运行在 *node9* 上。我们在所有 10 个节点上启用了无需密码的 SSH 连接，以便于通信，因为我们的评估关注性能指标而非安全性测量。图 7 展示了我们 10 节点集群的拓扑结构。

*4.2.2* 基准系统。保序加密（OPE）领域的相关文献主要来自安全和数据库两个社区。并非所有提出的构想都已实现；截至本文撰写时，最先进的 OPE 系统发表于 [22], , 该系统采用对称加密方案 128 位 AES [25] 作为构建模块。由于明文长度并非总是 128 位，我们的实现采用 PKCS7 [18] 填充方案。我们将用 OPEA 指代这一特定系统。

原始 OPEA 系统通过 MySQL 上的服务器端用户自定义函数和使用 Python 编写的客户端脚本实现。不过，为了进行公平比较，我们为 SQLite 重新实现了 OPEA 的服务器端和客户端功能；具体而言，我们采用了

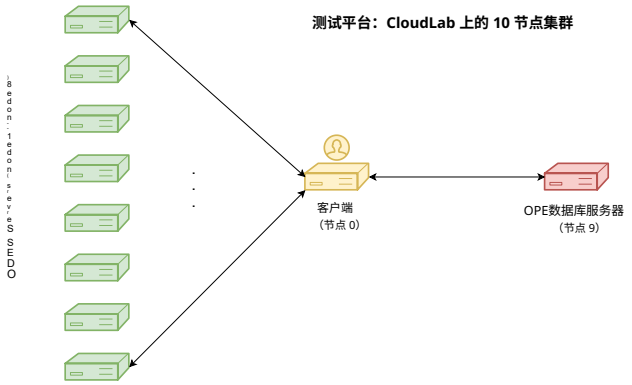


**图 4：CloudLab 上的 10 节点集群**

最先进的安全通信框架 *paramiko*和密码学软件包 *hazmat* 作为实现中的安全构建模块。

*4.2.3* 数据集。合成基准测试采用 TPC-H 3.0 版，这是标准的数据库基准测试。TPC-H 共包含八张表；我们选取其中四张来评估所提出的工作：供应商、客户、零件和订单。选择这四张表有两个原因。首先，它们都有单属性主键，可以直接编码到底层数据结构中。其次，它们都至少包含一个比文本属性更值得编码的数值属性。我们关注的属性包括：

供应商.S_Acctbal、客户.
C_Acctbal、零件.P_
Retailprice、订单.O_
Totalprice。

在规模为0.01的TPC-H中，上述列分别包含100、1,500、2,000和15,000个元组。

除合成基准测试外，我们还使用三个真实世界数据集评估了所提出的工作。

- COVID-19。第一个应用是美国2020年4月至2020年3月的全国COVID-19统计数据。该数据集包含341天的16项指标，例如死亡人数增量、阳性病例增量和住院人数增量。
- 比特币。第二个应用是自2013年2月首次在公开市场交易以来的比特币交易量历史数据。该数据按3天为一个周期，记录了从2013年2月至2022年1月累计的比特币交易量，总计1,086个大数值。• 人类基因组38号（hg38）。第三个应用是人类基因组参考序列， 38 [16], 通常称为 *hg38*，其中包含34,424行单一属性，例如转录位置、编码区和外显子数量，最后更新于2020年3月。

*4.2.4* 工作负载。如 [22], 中所述，我们主要关注保序数据库加密方案的两项性能指标：插入和查询。

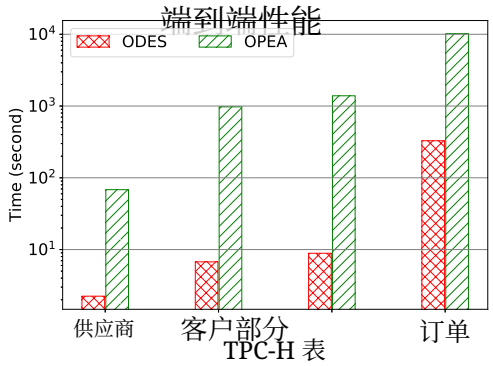


**图 5：ODES 和 OPEA 的整体性能**

- 插入工作负载的流程如下。客户端大致会执行以下 SQL，将一系列记录插入 SQLite 数据库：

  INSERT INTO <表名>VALUES <密文>;

  插入从零开始，即目标表为空。在插入过程中，会执行某种索引操作（具体取决于方案，例如 ODES、OPEA），以支持后续涉及排序操作的查询。本质上，插入不仅需要比较给定密文与表中已有的（加密）元组，从而产生搜索开销；新记录插入后，还需要维护顺序信息，从而产生更新开销。

- 查询工作负载比插入更简单。我们随机执行 $\log(n)$ 次逐点查询并比较其值，其中 $n$ 表示表中的记录总数，即基数。直观而言，如果要插入大量记录，查询时间应显著短于插入时间。

我们还将报告客户端产生的开销（如适用）。客户端开销包括明文编码的初始化和维护，以及其他可选的辅助信息，例如 OPEA 中的本地表 或 ODES 中的线性秘密共享 。

所有实验至少进行三次，并报告平均值。由于方差可以忽略不计，我们不绘制方差（即误差线）。如有需要，可提供完整的实验日志。除非另有说明，ODES 的默认份额数为两个。

## 4.3 TPC-H 上的端到端性能

我们首先报告所提出的 ODES 协议与传统 OPEA 结合时的端到端性能。图 5 展示了向空表中插入 $n$ 个元组后执行 $\log(n)$ 次查询的端到端执行时间，其中 $n$ 表示记录总数，详见 §4.2.3。实验涉及四个 TPC-H 表，所用时间除插入时间和查询时间外，还包括客户端开销。

根据图 5 的结果，ODES 在所有 TPC-H 表上的性能都比 OPEA 高出数个数量级。具体而言，对于 Customer 和 Part 表，性能提升约为 100×；对于 Supplier 和 Orders 表，加速比超过 10×。因此，我们认为，在 TPC-H 上，ODES 可使整体保序加密时间提升10×–100× 倍。接下来的三个小节将把整体运行时间分解为三个阶段：客户端开销、插入时间和查询时间。

## 4.4 ODES 在真实世界中的应用

我们报告 ODES 在三个不同真实世界数据集上的性能。我们将执行时间分解为客户端开销、插入时间和查询时间，以便更好地了解真实世界 ODES 应用中的开销分布。由于我们的目标是深入了解实际应用中的开销分配，因此不会将该指标与基线（即 OPEA）进行比较。

图 6 展示了这三个真实世界数据集的时间开销分布。其中有几点值得关注。首先，客户端开销微不足道，甚至可以忽略不计。这再次印证了我们此前关于 ODES 适用于物联网（IoT）或边缘计算应用中的轻量级设备的结论。其次，插入和查询之间的成本分配会动态变化。对于 COVID-19，插入成本几乎比查询成本低 10×；但 hg38 中则恰好相反，插入成本比查询成本高出 10× 。这主要可以用表的基数来解释：COVID-19 中有 341 行，而 hg38 中有超过 30,000 条记录。

## 4.5 客户端开销

请回想一下，在 OPEA 中，客户端会维护一个明文和密文的键值存储，以便在插入新记录时，客户端能够找到下界编码和上界编码对应的密文。这是因为 OPEA 必须使有状态客户端表与服务器端的记录编码保持同步。每个密文本质上都是一个 128 位字节串（因为 OPEA 使用 128 位 AES 进行加密），与明文长度无关[1]。假设一个明文数值以 4 字节（32 位）存储，则 OPEA 中本地表的扩展率为 $\binom{128+32}{32} = 5\times$。

相比之下，ODES 客户端是无状态且轻量的：（i）客户端只需生成 $m-1$ 随机数作为份额，并通过从明文中减去这些 $m-1$随机数来计算第 $m$ 个份额；（ii）客户端无需维护任何用于存储中间值的数据结构（严格来说，如果沿用 OPEA 的术语，扩展率为零）。因此，合理地预期 ODES 的客户端开销会低得多。

图 7 展示了 ODES 和 OPEA 在客户端的开销。事实上，我们发现，对于所有 TPC-H 表，ODES 的客户端开销比 OPEA 低约 3–4 个数量级。除了开销更低之外，结果还表明，ODES 对智能手机和智能手表等轻量级设备而言是更实用的解决方案，因为这些设备的计算能力和存储容量有限。

[1]显然，我们在此假设元组的表示长度小于 128 位。实际上，数据库应用中的数值几乎总是小于 $2^{128}$。

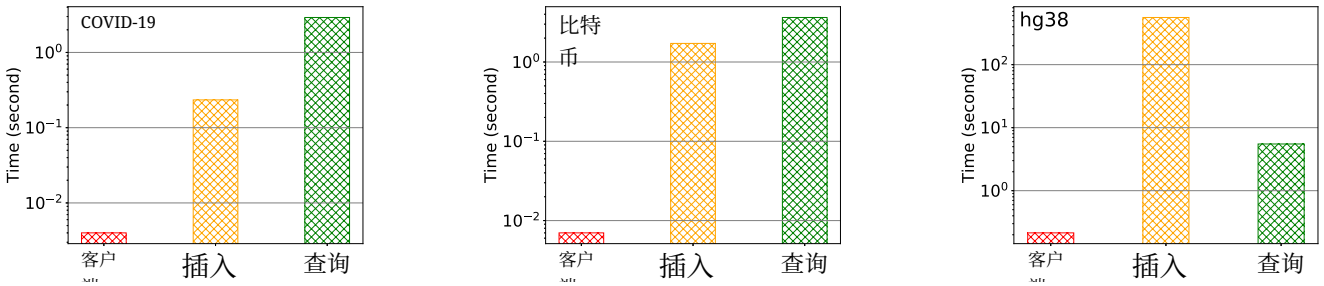


**图 6：面向实际应用的 ODES**

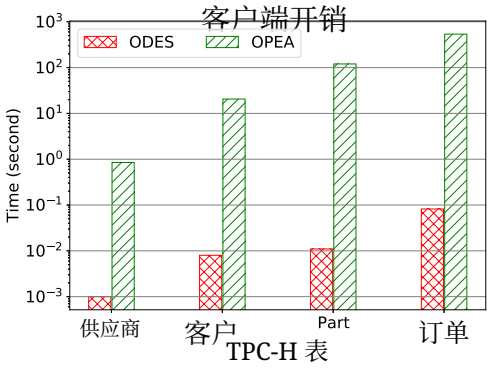


**图 7：ODES 和 OPEA 的客户端开销**

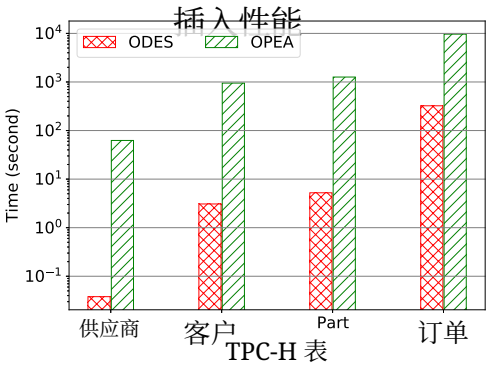


**图 8：ODES 和 OPEA 的插入性能**

## 4.6 插入性能

在 OPEA 中，插入一条记录的主要成本在于客户端与服务器之间的交互。客户端找到最接近下一个明文的较低和较高密文后，需要向服务器发出两次查询，以获取这两个密文的编码。为简单起见，可以将编码理解为明文的位置。粗略来说，每次 OPEA 插入都会额外产生两次数据库查询（一次查询较低邻项，另一次查询较高邻项）。也就是说，插入时查询次数会增加到原来的三倍。此外，客户端需要调整本地表，以纳入新插入的明文及其对应的密文、本地表中的条目位置，以及重复明文可能出现次数的计数器。

相比之下，ODES 插入记录的主要开销在于将秘密份额上传到多个服务器。不过，ODES 是一种无状态协议，这意味着客户端或服务器端都无需维护额外信息。也就是说，不同服务器上的查询可以并行执行。因此，尽管 ODES 的查询总数可能多于 OPEA（如果 ODES 服务器超过三个），执行这些查询的运行时间仍与单次查询大致相同。综合以上因素，我们预计，将记录插入数据库时，ODES 的运行速度远快于 OPEA。

图 9 展示了 ODES 和 OPEA 在 TPC-H 表上的插入性能。我们观察到，对于小型和中型数据集（Supplier、Customer、Part），ODES 的性能比 OPEA 高出两个数量级以上。对于大型 Orders 表数据集，加速比也十分显著，超过 10×。

## 4.7 查询性能

ODES 和 OPEA 在查询性能上的差距，明显小于记录插入时的差距，因为 OPEA 的排序信息以明文形式存储，因此可以利用高效的二进制排序。不过，如上所述，扩展密文（即 5×）所需的存储空间和网络传输量远大于明文。另一方面，ODES 在查询过程中处理 $m$× 明文，这些操作可以并行执行，其中 $m$ 表示节点数量（份额）。

图 9 展示了 ODES 和 OPEA 的查询性能。我们发现，就查询性能而言，ODES 仍优于 OPEA，但差距仅为约 35%–55%，而非我们在插入方面看到的几个数量级。不过，我们认为 35%–55% 的提升仍然显著，体现了所提出的 ODES 协议的优越性。

## 4.8 ODES 的可扩展性

我们感兴趣的最后一项 ODES 性能指标，是不同份额数量下的可扩展性。到目前为止，所有实验都假设有两个节点，每个节点

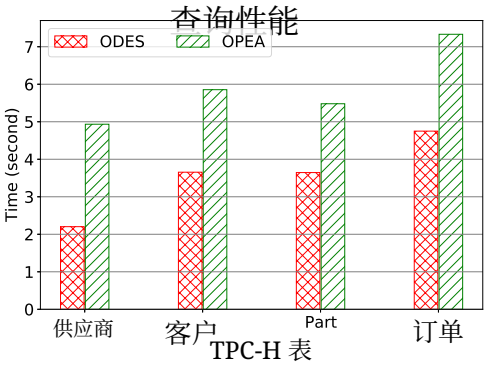


**图9：ODES 和 OPEA 的查询性能**

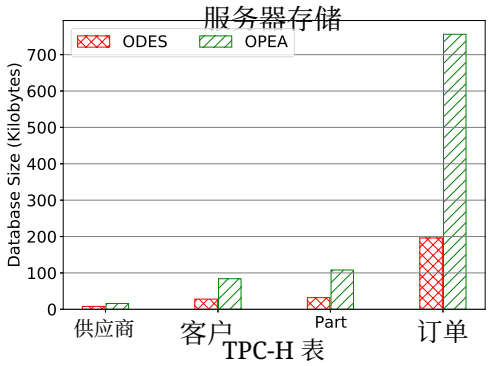


**图11：ODES 和 OPEA 的服务器存储**

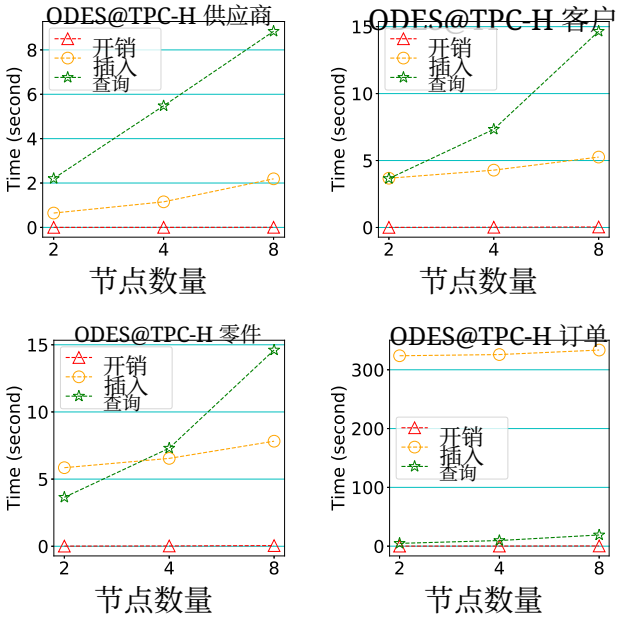


**图10：ODES 在四个 TPC-H 关系上的可扩展性：供应商、客户、零件和订单。**

持有明文的一个秘密份额。没有技术限制阻止用户使用更多节点来提高安全级别，代价可能是需要更多处理时间。也就是说，对手需要攻陷更多节点才能发动成功攻击。

图10展示了在四张 TPC-H 表上使用不同数量的份额（即节点）时 ODES 的可扩展性。我们观察到，在所有规模下，客户端开销都可以忽略不计，这与我们的预期一致。随着份额数量增加，插入成本略有上升，这可以用将明文拆分为更多份额所需的成本更高来解释。查询时间随节点数量增加而呈现更明显的上升趋势；其开销似乎与节点数量成正比。这种线性关系的主要原因是*paramiko* 库采用串行实现，该库是同步的，因此必须等一个远程查询完成后才能发起另一个查询。我们认为，客户端通过利用更多 CPU 核心和更高的网络带宽进行并行处理，可以大幅缓解这一问题；这将作为我们未来的工作方向。

## 4.9 存储成本

客户端存储。ODES 是一种无状态协议，这意味着 ODES 无需任何存储。另一方面，OPEA 的空间开销为 $O(\hat{n})$，其中 $\hat{n}$ 表示该列中的不同值数量。

服务器存储。图11报告了加密各种 TPC-H 表时 ODES 和 OPEA 的数据库大小。结果表明，对于小规模和中等规模的数据集，数据库大小并不显著。然而，对于订单表等较大的数据集，OPEA 的空间开销要高得多。这主要是因为 OPEA 使用了 AES 方案，会导致密文扩容。也就是说，出于安全方面的考虑，无论明文值有多小，加密后的文本长度始终固定（例如，128 位）。因此，明文越多，产生的空间开销就越大。

## 5 结论与未来工作

本文提出了一种新的无状态保序加密方案，即 ODES（基于秘密共享的有序数据库加密），该方案结合了秘密共享原语。ODES 支持最新的 IND-FAOCPA 安全性。本文基于 ODES 设计了一系列数据库协议。我们在由 10 节点集群中的 SQLite 数据库组成的环境中实现了 ODES 方案及数据库协议。实验结果表明，在 TPC-H 基准测试和多个实际应用中，ODES 的性能比最先进的方案高出数个数量级。

我们未来的工作将聚焦于效率更高的秘密共享原语，希望借此进一步缩短数据库应用中的插入和查询时间。另一个相对独立的研究方向是探索在数据库系统的保序加密中使用同态加密的可行性。

# 参考文献

[1] Rakesh Agrawal、Jerry Kiernan、Ramakrishnan Srikant 和 Yirong Xu。面向数值数据的保序加密。载于*2004* 年 *ACM SIGMOD* 数据管理国际会议论文集，SIGMOD '04，第 563–574 页，美国纽约州纽约市，2004 年。计算机协会。[2] Panagiotis Antonopoulos、Arvind Arasu、Kunal D. Singh、Ken Eguro、Nitish Gupta、Rajat Jain、Raghav Kaushik、Hanuma Kodavalla、Donald Kossmann、Nikolas Ogg、Ravi Ramamurthy、Jakub Szymaszek、Jeffrey Trimmer、Kapil Vaswani、Ramarathnam Venkatesan 和 Mike Zwilling。Azure SQL 数据库始终加密。载于*2020* 年 *ACM SIGMOD* 数据管理国际会议论文集，SIGMOD '20，第 1511–1525 页，美国纽约州纽约市，2020 年。计算机协会。[3] Diego F. Aranha、Anders P. K. Dalskov、Daniel Escudero 和 Claudio Orlandi。基于 LSS 同构的改进门限签名、主动式秘密共享和输入认证。载于 Patrick Longa 和 Carla Ràfols 编，《密码学进展——拉丁美洲密码学与信息安全第七届国际会议 LATINCRYPT 2021：哥伦比亚波哥大，2021 年 10 月 6–8 日，论文集》，《计算机科学讲义》第 12912 卷，第 382–404 页。Springer，2021 年。[4] D. Beaver、S. Micali 和 P. Rogaway。安全协议的轮数复杂度。载于第二十二届 *ACM* 计算理论年度研讨会论文集，STOC '90，第 503–513 页，美国纽约州纽约市，1990 年。计算机协会。[5] Vincent Bindschaedler、Paul Grubbs、David Cash、Thomas Ristenpart 和 Vitaly Shmatikov。隐私保护数据库中的推断之道。*VLDB* 学报，11(11):1715–1728，2018 年 7 月。[6] 比特币交易历史。https://www.blockchain.com/charts/trade-volume，访问于 2022 年。[7] Dmytro Bogatov、George Kollios 和 Leonid Reyzin。揭序加密方案与安全范围查询协议的比较评估。*VLDB* 学报，12(8):933–947，2019 年 4 月。[8] Alexandra Boldyreva、Nathan Chenette、Younho Lee 和 Adam O'Neill。保序对称加密。载于第 *28* 届密码学进展国际年会论文集——*EUROCRYPT 2009*——第 *5479* 卷，第 224–241 页，德国柏林、海德堡，2009 年。Springer-Verlag。[9] Dan Boneh、Kevin Lewi、Mariana Raykova、Amit Sahai、Mark Zhandry 和 Joe Zimmerman。语义安全揭序加密：无需混淆的多输入函数加密。载于 Elisabeth Oswald 和 Marc Fischlin 编，《密码学进展——EUROCRYPT 2015》，第 563–594 页，德国柏林、海德堡，2015 年。Springer Berlin Heidelberg。[10] David Cash、Feng-Hao Liu、Adam O'Neill、Mark Zhandry 和 Cong Zhang。参数隐藏的揭序加密。载于 Thomas Peyrin 和 Steven Galbraith 编，《密码学进展——ASIACRYPT 2018》，第 181–210 页，尚，2018 年。Springer 国际出版。[11] Nathan Chenette、Kevin Lewi、Stephen A. Weis 和 David J. Wu。具有有限泄漏的实用揭序加密。载于 Thomas Peyrin 编，《快速软件加密》，第 474–493 页，德国柏林、海德堡，2016 年。Springer Berlin Heidelberg。[12] 新冠肺炎数据。https://covidtracking.com/data/download/national-history.csv，访问于 2022 年。[13] Dmitry Duplyakin、Robert Ricci、Aleksander Maricq、Gary Wong、Jonathon Duerig、Eric Eide、Leigh Stoller、Mike Hibler、David Johnson、Kirk Webb、Aditya Akella、Kuangching Wang、Glenn Ricart、Larry Landweber、Chip Elliott、Michael Zink、Emmanuel Cecchet、Snigdhaswin Kar 和 Prabodh Mishra。CloudLab 的设计与运行。载于 *USENIX* 年度技术会议（*ATC*）论文集，第 1–14 页，2019 年 7 月。[14] H. Hacigumus、B. Iyer 和 S. Mehrotra。提供数据库即服务。载于第 *18* 届数据工程国际会议论文集，第 29–38 页，2002 年。[15] Richard D Hipp。SQLite，2020 年。[16] 人类基因组数据库。http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/，访问于 2022 年。[17] Mitsuru Ito、Akira Saito 和 Takao Nishizeki。实现一般访问结构的秘密共享方案。日本电子与通信（第三部分：基础电子科学），72(9):56–64，1989 年。[18] Burt Kaliski。PKCS #7：密码消息语法版本 1.5。RFC 2315，1998 年 3 月。[19] Florian Kerschbaum。频率隐藏保序加密。载于第 *22* 届 *ACM SIGSAC* 计算机与通信安全会议论文集，CCS '15，第 656–667 页，美国纽约州纽约市，2015 年。计算机协会。[20] Vladimir Kolesnikov。门评估秘密共享与安全的单轮双方计算。载于 Bimal Roy 编，《密码学进展——ASIACRYPT 2005》，第 136–155 页，德国柏林、海德堡，2005 年。Springer Berlin Heidelberg。[21] Kevin Lewi 和 David J. Wu。揭序加密：新构造、应用与下界。载于*2016* 年 *ACM SIGSAC* 计算机与通信安全会议论文集，CCS '16，第 1167–1178 页，美国纽约州纽约市，2016 年。计算机协会。[22] Dongjie Li、Siyi Lv、Yanyu Huang、Yijing Liu、Tong Li、Zheli Liu 和 Liang Guo。具有较小客户端存储的频率隐藏保序加密。*Proc. VLDB Endow.*，14(13):3295–3307，2021年10月。[23] Yehuda Lindell。如何模拟——模拟证明技术教程，第277–346页。Springer 国际出版，尚，2017年。[24] Moni Naor、Benny Pinkas 和 Reuban Sumner。保护隐私的拍卖与机制设计。载于第*1*届*ACM*电子商务会议论文集，EC '99，第129–139页，美国纽约，1999年。计算机协会。[25] 美国国家标准与技术研究院。高级加密标准。*NIST FIPS PUB 197*，2001年。[26] Raluca Ada Popa、Frank H. Li 和 Nickolai Zeldovich。一种理想安全的保序编码协议。载于*2013*年*IEEE*安全与隐私研讨会，第463–477页，2013年。[27] Daniel S. Roche、Daniel Apon、Seung Geol Choi 和 Arkady Yerukhimovich。POPE：部分保序编码。载于*2016*年*ACM SIGSAC*计算机与通信安全会议（*CCS '16*）论文集，美国纽约，2016年。计算机协会。[28] Mike Rosulek。密码学之乐。https://joyofcryptography.com。[29] Adi Shamir。如何共享秘密。*Commun. ACM*，22(11):612–613，1979年11月。[30] Ning Shen、Jyh-Haw Yeh、Hung-Min Sun 和 Chien-Ming Chen。一种面向外包数据库的实用、安全、无状态保序加密。载于*2021*年*IEEE*第*26*届环太平洋可靠计算国际研讨会（*PRDC*），第133–142页，2021年。[31] TPC-H 3.0.0。http://tpc.org/tpc_documents_current_versions/current_ specifications5.asp，访问于2022年。[32] Andrew C. Yao。安全计算协议。载于第*23*届计算机科学基础年会（*SFCS 1982*），第160–164页，1982年。[33] Samee Zahur、Mike Rosulek 和 David Evans。两半合为一体——使用半门减少混淆电路中的数据传输。载于 Elisabeth Oswald 和 Marc Fischlin 编，密码学进展——*EUROCRYPT 2015*——第*34*届密码学技术理论与应用国际年会，保加利亚索非亚，*2015*年*4*月*26*日至*30*日，论文集，第二部分，第9057卷《计算机科学讲义》，第220–250页。Springer，2015年。