arXiv:2305.03842v1 [cs.DB] 5 May 2023

Data Station：通过委托、可信且可审计的计算，借助数据托管机制促进数据共享联盟

Siyuan Xia、Zhiru Zhu、Chris Zhu、Jinjin Zhao、Kyle Chard、Aaron J. Elmore、Ian Foster、Michael Franklin、Sanjay Krishnan、Raul Castro Fernandez
芝加哥大学
{stevenxia,zhiru,chz,j2zhao,chard,aelmore,foster,mjfranklin,skr,raulcf}@uchicago.edu

摘要
汇集和共享数据能够提升其价值并扩大价值的惠及范围。但由于数据一经共享便无法撤回，出于监管、隐私和法律原因需要控制数据发布的场景，通常会选择不共享。由于难以有选择地控制要发布的数据，现有的少数数据共享联盟往往围绕数据共享协议建立，而这些协议需要经过漫长、繁琐且一次性的谈判才能达成。

我们推出 Data Station，这是一种旨在促成数据共享联盟成立的数据托管机制。数据所有者 将数据共享给托管方，并确信未经其同意，数据不会被发布。数据用户 将计算委托给托管方。数据托管方依靠委托计算，在不预先发布数据的情况下执行查询。Data Station 利用硬件安全飞地在参与者之间建立信任，并借助数据和计算的集中化生成审计日志。

我们在不可信中介上运行 Data Station，并使用机器学习和数据共享应用程序对其进行评估。除了重要的定性优势外，我们还表明，Data Station：i）在机器学习应用的准确率和运行时间方面优于联邦学习基线；ii）速度比其他安全数据共享框架快几个数量级；iii）在关键路径上仅引入少量开销。

1 引言

汇集和共享数据可以提升其价值并扩大价值受益范围。汇集数据的组织能够构建更强大的机器学习模型，并从中共同受益 [33]。相互共享数据的医疗组织能够改善患者护理 [41]。此外，共享实验数据的研究人员能够加速科学发现 [54]。尽管优势显而易见，实践中成立的数据共享联盟却寥寥无几。许多能够从数据共享中受益的组织面临监管、法律、隐私、激励和技术障碍，因此只能以受控方式发布选定的数据 [4, 13]。从技术角度来说，控制数据的使用方式并非易事，因此许多有益的数据共享联盟始终未能形成。即便有联盟成立，也往往围绕数据共享协议建立，而这些协议源于漫长、繁琐的一次性谈判，难以灵活应对数据使用方式的后续变化。

本文介绍 Data Station，这是一种中介式**数据托管**，也是一种计算与数据管理基础设施，旨在促成数据共享联盟的形成。数据所有者与托管方共享数据时，可以将其视为自身基础设施的延伸；也就是说，可以保证其数据始终保密，且未经其明确许可，任何人都无法访问这些数据（或任何衍生洞见）。希望从数据中提取洞见的用户会将计算委托给托管方执行，且只有在数据所有者许可的情况下才会执行该计算。托管方确保所有数据都受到保护，对威胁模型所作的假设很少，因此所有者和用户都可以信任它。最后，由于许多数据共享场景涉及监管和合规要求，平台上进行的所有计算都必须透明，以便第三方审计员和合规官能够审计这些联盟。

1.1 数据共享场景

我们通过真实应用案例构建了一些典型场景，以说明促成数据共享联盟所带来的机遇。

1.1.1 组织内部的数据共享：团队联盟。 分析师着手解决数据任务时，例如构建机器学习模型，或提取查询结果以完成报告，必须在组织内海量的数据源中找到相关数据 [17]。遗憾的是，许多数据源彼此孤立，由各个数据所有者分别管理，他们负责控制哪些人可以访问数据。这带来了一个重要挑战。在看到数据集之前，分析师未必知道它是否适用于自己的任务，因此他们必须与数据所有者进行耗时的一次性协商，以了解数据集并商定访问权限。此外，即使获得了数据集的访问权限，分析师也可能发现数据集不完整或描述不充分，或者还需要访问其他数据集，从而形成反复沟通的循环，延长获得洞见所需的时间。简而言之，即使在组织内部，分析师也必须预先投入大量成本来判断数据是否有用。数据所有者则不得不通过与分析师沟通，了解是否应授予数据访问权限。

在这种情境下，需要一个平台，允许分析师在数据所有者无需先释放数据的情况下，使用所有者的数据执行任务并进行评估。数据托管服务可以帮助所有者和用户解决这一问题，同时让合规官审计员工如何使用数据资产，并迅速发现数据滥用，例如使用敏感属性训练机器学习模型。

1.1.2 跨组织的数据共享：组织联盟。许多组织可以通过汇集各自的数据来训练更优秀的机器学习模型，从而实现互利共赢，但它们对共享原始数据集仍持谨慎态度。例如，化学工程组织可能愿意汇集获取成本高昂的模拟数据，以训练更强大的模型并寻找更优质的材料 [12],

但他们不希望其他组织看到自己所拥有的数据。他们愿意分享基于所有人的数据训练得到的模型所产生的推断结果，但目前尚无简便的解决方案，能够协调此类数据共享联盟，同时避免泄露各组织各自数据集的信息。尽管数据共享具有诸多益处，但上述风险导致组织不愿共享数据，使得数据的潜在价值无法得到充分发挥。

我们需要的是一个平台，它能够在不向任何人泄露数据的情况下，将各参与方的数据汇集起来训练模型，并有选择地允许参与方访问模型推断结果。这样的平台将帮助组织汇集数据，释放其价值。

1.2 DataStation 系统

主要贡献是 Data Station，这是一种新型数据托管系统，可实现委托计算、可信计算和可审计计算。

委托计算。 如今，数据访问与处理密不可分。要执行计算，必须先访问数据。然而，正如示例所述，数据共享受到种种障碍的限制，因此既无法进行计算，也无法从中提取价值。Data Station 充当托管方，数据所有者将数据发送给它，数据用户则将计算任务发送给它。将计算委托给 Data Station，意味着该平台可以承诺保护数据，即未经数据所有者明确同意，不会披露任何数据或派生数据产品。此外，这种模式使用户无需预先支付访问数据的费用，从而可以专注于获取查询结果。

可信计算。 只要数据所有者和用户都信任托管方能够确保其数据安全，引入托管机制就能实现数据共享。用户必须相信托管方能够安全地执行其计算任务，且不会将其泄露给其他参与者。数据所有者必须相信托管方会尊重其访问偏好。建立信任需要采用不同的机制，具体取决于威胁模型。Data Station 实现了两种模式：完全信任模式，适用于员工在运行第三方中介的组织内部执行数据发现等场景；以及近乎零信任模式，适用于独立组织希望通过第三方中介汇集各自数据等场景。为实现这些机制，Data Station 利用了安全硬件 enclaves 技术 [29] 和密码学技术。与旨在允许用户在第三方基础设施（例如云服务商）上对自己的数据执行计算的机密计算方法 [5, 6, 9, 34]不同，Data Station 旨在对来自多个参与方的数据执行计算。

可审计计算。 许多具有挑战性的数据共享场景受到监管，并受合规与审计规则约束 [15]。在这些情况下，即使数据所有者与用户彼此信任，他们在数据上执行的计算也必须对合规官和审计员等第三方透明。Data Station 利用平台对数据和计算进行集中管理的优势，将所有计算记录在防篡改不可变日志中。每次数据访问尝试、每项运行中的任务，以及数据所有者的数据访问偏好，都会存储在日志中。借助该日志，经授权的审计员可以验证 Data Station 运行的任务是否遵循管理这些数据的合规规则和法规。

贡献与评估结果。 据我们所知，Data Station 是首个为实现数据共享而设计的数据托管系统。在评估中，我们展示了由 Data Station 支持的两种共享应用程序。首先，我们表明，与联邦学习部署相比，Data Station 在多个学习任务中仅用很少的时间便实现了高得多的准确率。其次，我们展示了 Data Station 的开销最多比支持端到端加密的替代技术低两个数量级，从而支持更广泛的应用程序。最后，我们强调 Data Station 在定性方面的优势，并对其开销进行了全面评估。

本文重点。 要成功建立数据共享联盟，需要考虑隐私限制、法规、法律数据共享协议、参与者之间的激励机制等诸多方面。所有这些问题都很重要，但只有先有技术方案能够实现数据共享，它们才有意义。Data Station 旨在解决这一技术挑战。

本文其余部分安排如下。第2节概述 Data Station。第3节说明 Data Station 如何实现委托式和可审计计算的目标。第4节说明如何支持近零信任 模式。第5节介绍 Data Station 执行环境的设计与实现。第6节展示评估结果，第7节介绍相关工作，第8节总结全文。

2 DATA STATION 概述

第2.1节介绍 Data Station 使用的抽象，第2.2节介绍共享生命周期，第2.3节介绍计算生命周期。随后，第2.4节阐述 Data Station 作出的承诺，第2.5节概述其架构。

2.1 代理、数据实体、函数

代理 $a_i \in \mathrm{A}$ 是任何与 Data Station 交互的实体。代理分为三种类型。*所有者* 控制对其拥有的数据资产的访问权限。与其他代理共享这些数据资产可能会带来好处，因此所有者愿意将这些数据资产*注册* 到 Data Station。将数据资产注册到 Data Station，会把数据从所有者的基础设施复制到 Data Station 的基础设施。*用户*是希望对已注册到 Data Station 的数据运行计算的代理。最后，*运营者* 既不是所有者也不是用户，但他们希望了解 Data Station 内正在对哪些数据运行何种计算。审计员、合规官以及其他类型的监管人员都可能担任运营者。

任何已注册的数据资产在 Data Station 中都表示为数据实体 （DE），d_i。数据实体涵盖不同类型和粒度的数据，例如关系、数据库、文件、图像等。

Data Station 中的所有计算都通过 *函数* 表示，即 $f \in \mathrm{F}$。Data Station 提供一些基本函数，但大多数函数由计划组建共享联盟的所有者和用户提供，例如，化学工程师联盟可能会为分子数据提供专用索引和搜索功能。这些函数通过 $f \in \mathrm{F}$ 暴露在 Data Station 中。函数接收输入参数和数据实体作为输入，并生成其他数据实体以及可选的其他副作用，例如日志和临时文件；例如，训练函数接收数据实体作为训练数据并生成模型。用户无法看到任何函数副作用。因此，

函数应能够端到端运行。可以提供一套函数，利用托管服务准备和集成数据，但这需要解决本文范围之外的其他挑战。

2.2 策略与共享模式

所有者向 Data Station 注册数据实体，目的是最终允许某些计算在这些数据上运行。所有者通过策略 和共享模式，完全控制 Data Station 出于何种目的访问其注册的数据。

策略。 策略是一个三元组， a_i, f_i, d_i，表示代理 a_i可以在数据实体 d_i上运行函数 f_i。与访问控制策略不同，后者负责管理对文件的底层操作，例如 Unix 系统中的 MAC 所涉及的读取、写入和执行权限；Data Station 中的策略指明哪些函数可以在哪些数据实体上运行，因此它是一种更高层次的描述。

共享模式。 有三种共享模式 用于指明可用的数据访问类型：封闭、飞地、开放。在封闭模式下，除非有明确的策略允许访问，否则任何函数或 Data Station 都不能使用已注册的数据实体。若没有任何策略，所有者可以将以封闭模式注册的数据实体视为自有基础设施的单纯扩展，因为不会进行任何计算，且不会向任何人披露该数据实体的存在。

在飞地 模式下，Data Station 可以在数据实体上运行函数，但除非数据所有者明确同意（即明确制定允许发布结果的策略），否则不会发布任何输出。此模式允许 Data Station 执行创建索引、数据分析、模型训练等任务，同时确保不会向任何人泄露信息。

最后，当存在一项包含 a_i 和 d_j 的策略时，我们称数据实体 d_j 对给定代理 a_i 处于 开放 模式。

生命周期。 数据所有者最初可以在封存模式下注册一个数据实体，但所有者注册数据实体的目的是最终让用户能够从中受益。所有者可以将数据实体保留在封存模式下，并编写策略，以细粒度的方式描述谁可以对其数据执行哪些计算。例如，当允许第三方在其数据上测试一段软件，或通过中介机构进行安全数据交换时，这种方式非常有用。在其他情况下，所有者可以将数据实体设置为飞地模式，让 Data Station 在数据实体上运行计算，同时确保结果保持私密。例如，这有助于构建索引，使用户能够发现

相关数据实体，同时确保未经所有者明确同意，实际数据绝不会被释放。最终，若要将数据实体或衍生数据产品移出数据站，数据所有者必须明确制定书面策略，使该数据实体处于开放模式。

2.3 计算生命周期

用户调用在 Station 中预先注册的函数， $f_i \in \mathrm{F}$ 。函数调用会触发意图的创建，这些三元组的定义方式与策略类似 a_i, f_i, d_i，用于表示代理 a_i在数据实体 d_i上执行函数 f_i的意图。除非存在允许执行该函数的策略，否则绝不会针对封存的 数据实体创建意图。函数无法看到封存的 数据实体。

我们将函数大致分为两类：数据盲型函数和数据感知型函数。第一类函数不要求

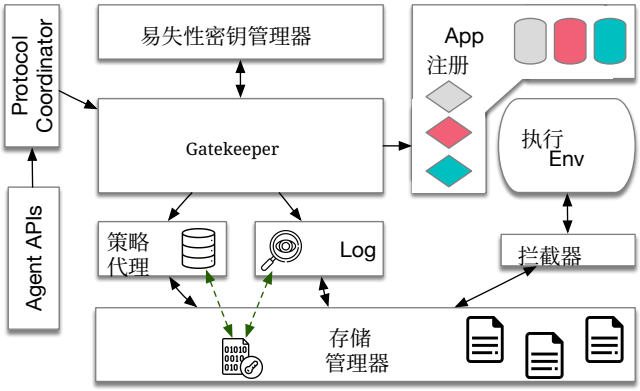


图 1：Data Station 架构概览

用户了解 Data Station 中的任何数据实体。例如，搜索和按示例查询函数接收来自用户的输入，而用户无需了解任何数据实体。相比之下，数据感知型函数需要以一组数据实体作为参数。例如，复制/下载函数需要指明要下载哪个数据实体。因此，任何用户都无法对（实际上不可见的）密封数据实体调用数据感知型函数。

派生数据产品。 当函数在数据实体上运行并生成输出时，我们称该输出为派生数据产品。派生数据产品是驻留在 Data Station 内的另一种数据实体。因此，数据实体既可以由其所有者上传，也可以由函数生成。派生数据产品是委托计算的关键组成部分，因为它们正是用户访问 Data Station 所要获取的结果。Data Station 必须解决的一项关键挑战是：将所有者针对已注册数据实体制定的策略应用于派生数据产品，即使这些产品可能源自不同所有者拥有的数据实体。

2.4 Data Station 的承诺与信任模式

Data Station 向所有者承诺，只有处于开放 模式的数据实体（包括派生数据产品）才会离开 Data Station。此外，Data Station 还向所有者承诺，涉及其所拥有数据实体的任何活动都会被记录，并可按需供其查看。

要维持这一保证，需要根据威胁模型采用不同的协议、算法，甚至基础设施。例如，当 Data Station 在组织内部运行，以便员工发现其他团队拥有的数据资产时，合理的威胁模型可能是：部署 Data Station 的基础设施并非恶意，数据实体会保留其原有的共享模式，且实现遵循所述承诺。相比之下，当 Data Station 在第三方基础设施上运行时，则必须确保在更具挑战性的威胁模型下兑现这一承诺。Data Station 通过不同的运行模式，在不同的威胁模型下保证兑现这一承诺。当代理认为 Data Station 在其目标威胁模型下能够兑现承诺时，便会将其视为可信赖的 。

2.5 架构概览

Data Station 架构概览如图 1 所示。现有应用程序可以通过应用注册器组件注册到 Data Station，该组件会告知 Gatekeeper 哪些函数可供执行。注册完成后，数据用户可以调用这些

函数，并通过使用 Agent API 组件的各种接口进行调用。所有函数调用均由 Gatekeeper 代理处理，Gatekeeper 会向策略代理确认能否根据相关数据的策略及相应共享模式继续执行。数据所有者使用 Agent API 编写这些策略。如果可以执行，Gatekeeper 就会创建任务来运行该应用程序；所有操作都在执行环境中进行，应用程序在其中隔离运行。对存储管理器的所有访问——无论是访问现有数据，还是存储派生数据产品或其他临时数据——均由拦截器组件进行中介。最后，为了支持在不同威胁模型下运行，Data Station 使用易失性密钥管理器和协议协调器，稍后我们将对此进行介绍。

3 D委托式、可审计计算 ION

我们介绍 Data Station 中支持委托式和可审计计算的主要组件。本节假设所有者和用户都信任基础设施、函数实现以及负责维护平台的管理员。我们将在第4节放宽这些假设。

目标。 Data Station 的主要目标是运行由用户调用、处理数据所有者提供的数据的函数，并以满足数据所有者共享偏好的方式实现这一点，也就是兑现 Data Station 的承诺。

3.1 深入了解 Gatekeeper

函数调用后，Data Station 会确定该函数可以访问哪些数据实体，包括派生数据产品。Gatekeeper 是所有函数调用的唯一入口。因此，它会协调提供函数调用服务所需的所有步骤，同时兑现 Data Station 的承诺。用户只能调用 Gatekeeper 开放的函数。函数通过应用程序注册流程（第5节）在 Gatekeeper 注册。确定函数可使用哪些数据实体时，对数据感知型函数和数据盲型函数采用不同的流程。

代理数据感知型函数。 数据感知型函数通过路径、名称或其他标识符请求访问特定的数据实体。在执行函数之前，Gatekeeper 使用调用方代理 ID a_i、被调用的函数 f_j以及被访问的数据实体 d_k，创建相应的意图。Gatekeeper 会为涉及的每个数据实体创建一个意图。然后，它使用策略代理来确定这些意图是否匹配策略。策略代理组件由一个数据库提供支持，该数据库包含系统中注册过的所有代理、函数、数据实体和策略。接着，给定一个意图（a_i， f_j， d_k），策略代理会创建一个查询，以检查是否存在允许 a_i执行 f_j于 d_k上的策略。为提高效率，如果创建了多个意图，则会将它们全部表示在一个查询中。最后，如果存在匹配的策略，Gatekeeper 就允许在该数据实体上调用函数。否则，它会阻止执行。

代理数据盲型函数。数据盲型函数不包含其需要访问的数据实体列表，这与数据感知型函数不同。Gatekeeper 通过组合以下两组数据实体，确定函数和代理可访问哪些数据实体：i）具有匹配策略的数据实体集合；ii）处于飞地模式的数据实体集合。这两组数据均从策略代理处获取。第一组通过查询符合包含调用代理 和函数 的谓词的数据实体获得，第二组则通过请求存储在飞地模式下的数据实体获得。

函数执行。确定函数可以访问哪些数据实体后，Gatekeeper 必须确保函数只能访问这些数据实体。为此，需要创建一个“受监禁”（即采用 Unix chroot [28] 机制）的执行环境，将函数与所有可用资源隔离，只有获得明确授权的资源才能访问。我们将在第5节介绍执行环境的设计。现在只需了解，这样的执行环境还会记录：i) 函数实际访问的数据实体；以及 ii) 函数生成的数据实体。首先请注意，授予函数访问权限的数据实体可能多于其实际需要访问的数据实体；例如，构建时空索引的函数只需要访问具有空间和时间属性的数据实体。其次，函数会生成并返回结果。Gatekeeper 必须管理函数调用所返回的结果。

暂存区与结果交付。 Gatekeeper 必须决定是否可以将函数调用的结果转发给调用代理。如果调用的函数是 数据感知型，则该函数确实已执行，因为针对所指示的特定数据实体存在匹配策略。如果存在匹配策略，就意味着该数据实体的数据所有者已指明 a_i可以执行 f_j，这意味着 a_i可以访问结果。因此，在这种情况下，Gatekeeper 将向调用代理返回结果。如果调用的函数是 数据盲型，则该函数可能既在具有可用策略的数据实体上执行，也在没有策略的飞地 数据实体上执行。如果结果仅取决于具有匹配策略的数据实体，Gatekeeper 就可以将结果返回给调用代理。如果结果还取决于飞地 数据实体，则必须事先获得飞地 数据实体的数据所有者的许可，否则无法返回结果。Gatekeeper 通过从执行环境中获取实际访问的数据实体列表，来确定结果是否仅取决于具有匹配策略的数据实体，具体如上所述。任何无法直接交付给 a_i的结果都会存储在暂存区中，并以飞地 数据实体的形式存在，等待允许其释放的匹配策略。

结果粒度和访问权限协调。 当一个飞地 数据实体 d_k 属于函数的结果时，Gatekeeper 会向数据所有者请求许可，以便将 d_k 发布给 a_i。数据所有者最初将该数据实体设置为飞地 模式，是为了允许 Data Station 执行某些计算，但并未规定可调用哪些函数以及由谁调用。数据实体成为某次函数调用结果的一部分后，Gatekeeper 会提供有关谁访问了该数据实体（ a_i）以及通过哪个函数访问（ f_j）的准确信息。随后，数据所有者可以决定是否向发起调用的代理授予访问权限，这相当于创建策略（a_i, f_j, d_k）。拒绝访问不会修改策略列表，并会将该数据实体从暂存区移除。最后，如果飞地 数据实体列表很大，访问权限协调需要数据所有者参与，可能会耗费较长时间。Data Station 为发起调用的代理提供了一种选项，可仅针对已有策略的数据实体集合调用函数。此选项适用于希望在结果可用后立即查看结果的发起调用代理。

来源与粒度。 作为所有函数的单一接入点，并同时观察输入（可访问的数据实体

数据实体）、从执行环境获取的实际访问数据实体列表，以及函数输出后，Gatekeeper便能获知每个派生数据产品的来源。来源主要用于判断 a_i是否可以访问结果，如上文所述。来源粒度取决于注册函数的应用程序的构建方式。例如，使用 Data Station API 访问数据实体的原生应用程序可以精确说明哪些数据实体用于生成不同的输出，以及具体如何生成。Data Station 允许执行未经修改的应用程序。本文重点讨论后一种情况。在此，当函数输出的数据实体是输入数据实体的子集时，Gatekeeper会检查是否存在匹配的策略。例如，搜索函数会返回包含某个关键词的数据实体。当函数返回由输入数据实体组合而成的结果时，这些新创建的数据实体按定义没有匹配的策略。例如，函数使用不同所有者提供的训练数据集来训练机器学习模型。Gatekeeper依赖一种额外机制来协助处理这些常见情况。

3.2 派生数据与依赖图

当意图引用由数据所有者上传的数据实体时，Gatekeeper 会检查是否存在匹配策略。但在许多场景中，函数需要访问由其他数据实体派生而来的数据实体 [4]。例如，搜索函数可能会使用一个索引，而该索引又可能由另一个函数根据一组可访问的数据实体创建。该索引是一个中间数据实体。因此，没有数据所有者会直接为该索引编写策略。当函数在中间数据实体上运行时，Data Station 需要一种机制来确定对数据实体的访问权限，并在数据所有者编写策略时为其提供便利。

函数依赖图。 依赖图是一个有向无环图（DAG）。节点表示函数，有向边表示这些函数之间以数据实体形式存在的依赖关系。例如，搜索 函数依赖于一个索引 函数，该函数本身根据数据实体创建索引。依赖图在应用注册过程中创建（第5节）。所有者了解函数之间的依赖关系。当所有者编写策略，允许某个函数在其拥有的数据实体上执行时，也就隐式地允许该函数的任何子函数访问该数据实体。因此，如果他们为搜索 函数编写策略，也就允许索引 访问其数据实体。

带依赖图的策略匹配。 为利用函数依赖图，在策略匹配过程中，Gatekeeper 必须检查该函数或其任何后代是否已有策略。如果没有依赖图，在没有匹配策略的情况下，只有当数据实体处于飞地 模式时，Gatekeeper 才会授予该函数的执行权限。之后还需要协调对函数结果的访问，从而造成不必要的延迟。相反，依赖图可以跳过不必要的策略处理，并大幅简化用户、所有者和 Gatekeeper 的任务。

3.3 可审计计算：日志

如果 Data Station 信守承诺，那么所有者就能确保其数据实体仅以其所制定的策略允许的方式被访问。在某些情况下，审计员、合规官和其他第三方需要了解 Data Station 的内部运作。由于 Data Station 集中处理计算和数据，因此会记录每次函数调用的来源。这些记录可以存储在日志中，并作为 Data Station 所执行操作的真实记录提供给运营人员。

为了便于操作人员使用，可审计日志必须记录 DataStation 执行的每一项意图、策略匹配检查以及数据实体结果交付。该日志必须允许数据所有者查阅与其拥有的数据实体相关的信息，并允许数据操作人员在参与的代理同意此类安排时访问这些信息。换言之：Data Station 中发生的任何计算都必须记录在日志中。

Gatekeeper 是管理此日志的理想人选，因为它已经充当中央角色，检查每个旨在访问数据实体的函数调用。因此，从架构上看，该日志位于 Gatekeeper 内。所有对此日志的写入操作都由 Gatekeeper 发起，而所有对日志的读取都通过 Gatekeeper 进行，就像其他任何函数调用一样。

该日志是一个数据实体，其数据所有者是 Data Station。当新所有者加入 Data Station 时，Data Station 会创建策略，允许所有者访问日志并查看与其拥有的数据实体相关的任何活动。随后，所有者可以通过调用相应的函数来检查日志。

日志条目。该日志存储在磁盘上，由一系列条目组成。每个条目包含一个代理 ID，对应于触发该条目创建的调用者，以及一个有效载荷。有效载荷可以有不同类型。它可以表明某个意图已创建、意图与策略匹配，或两者不匹配。它还会记录哪些数据实体可以离开 Station，包括派生数据产品。请注意，可以追踪任何中间数据实体由哪些数据实体贡献而来，因为 Gatekeeper 会保留来源，而来源本身也会记录在此日志中。所有与数据实体相关的历史记录都存储在日志中，这正是其可审计性的来源。

通过合同向第三方开放访问权限。 默认情况下，只有数据所有者才能查看日志，而且他们只能查阅与其拥有的数据实体相关的条目。若要允许第三方访问，Data Station 中所有参与的代理都必须生成一份合同。通过合同，Data Station 会为运营方代理创建一项策略，随后该代理便可以查阅日志。合同是一项策略（a_o, r, l），其中 a_o是运营方的 ID， r是用于读取日志的函数， l是指向该日志的数据实体。合同与策略的不同之处在于，合同必须由每一位参与者签署。若缺少任何一位参与者的签名，Data Station 就不会创建该策略，运营方也无法查阅日志。Data Station 依靠公钥密码学来允许代理签署合同；这些密码学基础原语将在下一节中介绍。

4 可信计算

我们介绍用于保护 Data Station 在不可信基础设施上运行时所作承诺的机制。当这些机制（在第 4.1 节中介绍）启用时，我们称 Data Station 以近零信任 模式运行。我们将在第 4.2 节介绍 Data Station 使用的加密协议，并在最后说明如何处理日志和数据库（第 4.3 节和第 4.4 节）。

威胁模型。 我们假设，好奇的运营者能够访问 Data Station 基础设施，并读取磁盘和内存内容。

为了兑现承诺，Data Station 不能向该运营者泄露任何数据实体。此外，Data Station 不能泄露数据库（其中包含所有代理、数据实体和策略）或可审计日志中的信息。攻击者可能会获取代理 ID 列表，但不应能够将这些 ID 与任何其他信息关联起来。我们不防范拒绝服务攻击。

4.1 近零信任原则

为确保 Data Station 中每个数据实体的机密性，数据从离开代理基础设施的那一刻起，直到函数在内存中访问该数据实体（即处理期间），始终进行端到端加密。为确保完整性，每条消息都使用其来源代理的私钥签名。为建立信任，Data Station 的节点会向代理证明其身份，并证明其运行的是原版 Data Station 软件，而非修改过的版本，从而避免后门攻击。

每个共享的数据实体都使用代理专属的对称密钥加密。随后，它通过 TLS [14] 安全通道传输至 Data Station。在 Data Station 中，数据实体静态存储时仍保持加密。当函数需要访问数据时，Data Station 会利用安全硬件飞地，使数据在内存中（即处理期间）保持加密；数据实体会被解密到加密内存中。函数可能需要在文件系统中存储中间结果，例如临时文件。由于其中一些中间结果可能泄露敏感信息，因此这些中间结果会以对应用透明的方式加密：在 Data Station 以 近零信任 模式运行时，任何函数都无法在磁盘上存储明文数据。我们将介绍安全硬件飞地，以及 Data Station 使用的两个关键属性：加密内存和远程证明。

4.1.1 安全硬件飞地简介。AMD 的安全加密虚拟化（SEV）和英特尔的软件防护扩展（SGX）利用专门构建的硬件，分别在称为安全飞地的区域内隔离虚拟机（节点） [29] 和应用程序 [25], , 以防止数据泄露给系统中权限再高的用户。这些技术带来了重要的权衡。

与加密节点全部工作内存的 SEV 相比，SGX 将总工作内存限制为 128MB [24]。其优势在于，使用 SGX 时，用户只需信任在飞地内运行的应用程序。相比之下，使用 SEV 时，用户还必须信任操作系统，因为最终由操作系统负责确保内存页经过加密。SGX 的另一个缺点是应用程序需要重写，而 SEV 则可以运行未经修改的软件。自 2022 年 1 月起，英特尔已弃用 SGX [11]。鉴于此，再加上加密全部内存更加方便，我们采用 AMD 的 SEV 来实现 Data Station。

加密内存。 AMD 的 SEV 通过加密操作系统对内存的所有写入操作来保证机密性。未经授权的用户（包括云环境中的虚拟机监控程序）无法读取明文数据，也就是说，转储进程的内存内容（例如 cat /proc/[pid]/maps）将显示密文。采用较新的 SEV-SNP （安全嵌套分页） [49], 后，可信计算基（ TCB ）（代理需要信任的所有硬件、固件和软件的集合）仅包含两个组件：AMD 硬件和固件，以及在节点中运行的操作系统映像。所有其他组件均不可信，包括 BIOS、虚拟机监控程序、其他映像（在多租户不中返云场景），以及外部 PCI 设备。总而言之，代理只需信任 AMD 的硬件实现正确，且操作系统映像正确实现了 SEV。

针对安全飞地实现的攻击与局限。在近零信任 环境中运行的 Data Station 会继承底层 SEV-SNP 实现的任何漏洞。解决 SEV 特定于实现的缺陷 [35, 36] 不在本文的讨论范围内；在撰写本文时，AMD 正积极致力于缓解这些缺陷。SEV 安全飞地可保护主内存中数据的机密性和完整性，但无法保护存储在外部设备上的数据，例如磁盘或 GPU。Data Station 可保护磁盘上的数据及其传输到内存的过程，但不支持在 GPU 上进行计算。安全飞地的另一项局限是性能较低：我们将在评估部分展示，这种开销不会显著影响应用程序的运行时间。最后，每家主要供应商都提供安全飞地技术，但截至 2022年5月，只有 AMD 提供 SEV 所具备的、使其适用于 Data Station 的特定安全保证，且该技术仅在 Google Cloud 中可用。

4.1.2 通过远程证明建立信任。为了让代理确信运行 Data Station 的基础设施已启用 SEV，且所运行的软件确实是 Data Station 软件，平台采用 AMD 硬件提供的远程证明。这使代理能够向节点请求一份包含唯一标识信息的报告。此外，代理可以确信节点运行的是正确实现了软件（包括 Gatekeeper）的 Data Station 版本，而不是包含后门的恶意篡改版本。如果软件重启，整个远程证明流程会重新执行，以防止攻击者替换软件版本。

4.2 端到端加密协议与密钥管理器

端到端加密协议必须确保所有数据实体始终处于加密状态，包括处理期间。同时，它必须允许具有匹配策略的代理访问这些数据实体。两个组件主要负责实现这一目标：使用大部分标准公钥密码学的协议协调器，以及用于保护密码密钥的机制易失性密钥管理器。接下来我们将分别介绍这两个组件。

预备知识。 每个代理和 Data Station 都拥有一对公钥和私钥。此外，每个代理还拥有一个与 Data Station 共享的对称密钥。数据在源端进行加密和签名（以证明身份），并在整个生命周期内始终保持加密状态。

协议协调器。 要处理数据实体，Data Station 首先使用数据实体所有者共享的对称密钥在内存中解密数据实体。由于飞地中的内存经过加密，数据仍受到保护，不会被外部观察者看到。要传输数据实体（例如，传输给匹配策略的代理），Data Station 会先在内存中解密数据实体，然后使用接收代理的对称密钥重新加密。由于这一处理过程始终在内存中进行，因此数据实体始终处于加密状态。

易失性密钥管理器。 Data Station 持有两类敏感信息：i) 代理用于加密数据实体的对称密钥；ii) 自身的私钥。如果其中任何密钥遭到泄露，整个系统的安全保障都会崩溃。为了保护这些密钥，密钥存储在位于

内存中的易失性密钥管理器中，因此是加密的。这是在 Data Station 运行期间确保密钥安全的一种简单方式。然而，如果 Data Station 重启（例如发生故障或进行维护），密钥将从密钥管理器中消失，因此需要一种恢复密钥的策略。某些飞地技术（例如英特尔的 SGX）支持对密封 数据，这意味着使用飞地专用密钥将内存中的数据加密到磁盘上。AMD SEV 不支持密封 [52]。因此，在必要时，Data Station 依靠代理重新发送其加密密钥，以恢复密钥管理器状态。

派生数据产品。 Data Station 负责在将派生数据产品存储到磁盘之前对其进行加密。请注意，无需使用任何特定密钥。与共享 DE 不同，派生数据产品之所以存在，是因为所有者允许 Data Station 对其调用函数。因此，Data Station 可以选择用于维持其磁盘加密状态的密钥。Data Station 使用调用代理的对称密钥，以避免在调用代理获准访问该数据实体时再进行一轮重新加密。

4.3 审计日志管理

如果审计日志以明文形式存储，那么获取节点访问权限的攻击者就会知道对哪些数据实体执行了哪些函数，以及由谁执行，从而可能了解到现有策略。尽管在某些应用程序中这可能并非关键问题，Data Station 仍通过在磁盘上加密存储日志来防范这一风险。日志由一系列 a_i，$E_{k_u}(l)$ 条目组成。代理 ID a_i 以明文形式存储，但只有能够访问数据库的人才能理解其含义。

只有数据所有者才能查看涉及其所拥有 DE 的日志条目，因此人们自然会认为这些条目是使用数据所有者的密钥加密的。然而，一个条目可能涉及许多 DE，例如，需要访问 DE 集合的函数；当 DE 数量较多时，使用每个 DE 的数据所有者的密钥加密该条目效率很低。相反，Data Station 使用触发这些条目的函数调用所属数据用户的对称密钥 k_u 加密日志条目。由于只有 Data Station 能访问审计日志，且每个条目恰由一个代理生成，因此这种方案效率更高。

4.4 数据库管理

攻击者若获得托管数据库的基础设施的访问权限，就会了解到所有策略、DE 位置（但无法访问 DE，因为这些信息已加密），以及在平台注册的任何代理的信息。为防范这种情况，数据库始终保留在内存中，因此由 SEV 加密。这样，即使攻击者获得物理机的访问权限，数据库内容也能得到保护。接下来的挑战是如何处理 Data Station 的故障和重启。

为解决这一问题，Data Station 使用了一种加密预写日志（EWAL），该日志独立于数据库系统内部使用的任何 WAL，并且位于其外部。数据库的所有更新操作都会首先以加密形式存储在 EWAL 中。每条日志记录均使用生成该更新操作的代理的对称密钥进行加密。每条记录都会与导致该更新操作的代理 ID 一同存储。代理 ID 的分配在数据库之外进行，以便将这些 ID 纳入 EWAL 日志记录中。重启后，数据库可以像传统关系数据库的恢复协议一样，通过日志进行恢复 [40]。为了解决.....

随着 EWAL 不断增长，数据库可以定期创建检查点并写入磁盘。在将检查点存储到磁盘之前，也必须先在内存中对其加密（见下文）。重启后，要解密 EWAL 条目，必须先从代理处收集密钥。重放 EWAL 后，Data Station 即视为已恢复。数据库恢复后，Data Station 随后即可恢复审计日志。

在没有密封机制的情况下加密数据库检查点。数据库检查点必须在存储到磁盘之前进行加密。使用密封机制时，Data Station 会使用飞地密钥加密检查点。在没有密封机制的情况下，我们会采用另一种解决方案。基本解决方案是选择代理的一个对称密钥来加密检查点。重新启动时，代理会将其密钥重新发送给 Data Station，Data Station 会尝试使用每个新接收到的密钥解密检查点，直到成功为止。由于在所有代理重新发送其密钥之前无法恢复操作，因此此过程不会引入额外延迟。为提高可靠性，即防止用于加密检查点的密钥所属代理无法重新连接到 Data Station，可以改为使用 m 代理的密钥加密检查点；这样，只要其中一个密钥能够解密数据库，Data Station 就会恢复运行。

4.5 其他保护措施与局限性

获得基础设施控制权的潜在攻击者无法修改数据库的内容，因为数据库已加密。因此，他们无法创建策略、代理或更改现有数据实体的共享模式。此外，当 *SEV-SNP* 可用时，这些针对内存内容完整性的攻击将被发现。与 Data Station 的所有交互都通过协议协调器进行协调，协议协调器还负责身份验证等事项。

虽然 EWAL 和审计日志在磁盘上经过加密，但攻击者仍可能对其进行篡改，例如添加随机字节。Data Station 不防御此类拒绝服务攻击。可通过扩展实现以支持复制服务，解决这一实现上的局限。

5 执行环境

要求。 执行环境包含 Data Station 的所有功能和组件，使应用程序能够运行。在设计 Data Station 时，我们希望现有应用程序无需修改即可运行，以便于部署。执行环境需要满足以下要求：

- 开发人员注册 现有应用程序时，无需更改应用程序的实现。他们只需提供一个简单的连接器，说明如何调用应用程序的功能。
- DataStation 为应用程序提供运行和处理函数调用所需的资源，同时确保应用程序与其他应用程序以及其无权访问的数据相隔离。
- 在执行期间，执行环境必须记录应用程序访问的数据实体，并在函数执行完毕后将其发送给 Gatekeeper。
- 在近零信任 模式下，数据在磁盘上经过加密。执行环境可确保应用程序透明地访问解密后的数据，因此在执行期间不会中断运行。

5.1 设计

函数注册。 应用程序开发者通过 Gatekeeper，借助连接器注册函数。连接器包含向 Data Station 暴露的函数，以及这些函数之间的依赖关系。这些信息足以让 Data Station 构建函数依赖图。系统初始化时，Data Station 会从开发人员处加载所有连接器，并向 Gatekeeper 注册这些函数。连接器中的每个函数都负责调用应用程序中的相应功能。

资源管理与隔离。 当 Gatekeeper 授予函数执行权限后，该函数将在一个隔离进程中实例化，并受到限制和控制，只能访问系统资源的指定部分。

拦截器中间件。 Data Station 在初始化时启动一个拦截器。该拦截器知晓 Gatekeeper 启动的所有进程，以及这些进程可访问的位置。它会拦截函数的隔离进程对存储发起的所有输入/输出调用。Gatekeeper 与拦截器中间件按以下客户端-服务器模式协同工作：

• Gatekeeper 通过函数的执行环境将可访问数据实体列表传递给拦截器。拦截器会过滤掉不可访问的数据实体，确保函数只能看到可访问的数据实体。

• 拦截器会记录函数实际访问的所有数据实体，并再次通过函数的执行环境将它们发送回 Gatekeeper。

• 在近零信任 模式下，数据在磁盘上以加密形式存储，且不得以明文形式存储函数的任何副作用（例如临时文件）。每当函数调用读取操作时，拦截器都会按需解密数据；同样，拦截器会使用数据所有者的对称密钥加密写入操作。

最后，Gatekeeper 从拦截器处获取函数实际访问的数据实体列表，并按照前文所述继续处理。

5.2 实现

执行器环境在本地文件系统上运行。将其扩展到其他设置不在本文的讨论范围之内。

Gatekeeper 使用 Docker 容器创建隔离进程。每个容器实际上都会创建一个监禁环境（从 chroot 的意义上来说），以限制函数可访问的资源。创建容器时会确保函数能够访问的所有路径都由拦截器拦截。此外，在 Docker 容器中实例化函数便于管理可用资源。

我们使用 FUSE [53], 和 libfuse 用户空间库 [2] 来实现拦截器中间件，该库用于在用户空间中创建文件系统。在 Data Station 的系统初始化期间，拦截器会将 DataStation 的存储挂载到指定的挂载点。此后，所有函数调用都将通过该挂载点访问数据实体，从而能够拦截所有输入/输出操作。

使用 FUSE 的一项关键优势是：由于文件系统是在用户空间中创建的，因此无需修改内核代码来拦截输入/输出调用并执行额外操作。避免修改内核对于缩小可信计算基至关重要。FUSE 文件系统提供了一种灵活的方式，可在不损害 Data Station 安全保障的情况下与其交互。

为支持并发函数调用，拦截器以服务器的形式运行，接收来自 Gatekeeper 的请求；Gatekeeper 会为每次函数调用启动一个隔离进程。拦截器会跟踪每个正在运行的函数所访问的相应数据实体。具体做法是，将每次数据访问与尝试访问数据的执行环境的身份相关联。同样，可访问的数据实体列表（以及在近零信任模式下对应的对称密钥）也与执行环境的身份相关联。Gatekeeper 和拦截器必须保持连接，以确保共享正确的进程上下文信息。

6 评估

我们展示评估结果，以回答两个问题：

RQ1：Data Station 支持哪些应用程序? 我们在 Data Station 中实现了两类应用程序：机器学习应用和文件共享应用。我们发现，Data Station 取得了显著更好的定量结果，并展现出许多不可估量的定性优势。

RQ2：Data Station 的设计决策能否构建出实用的系统? 我们研究了为确保可信性以及使用 SEV 而引入的开销。我们的目标是确定这些开销是否会成为应用程序运行时间的瓶颈。我们研究了 Data Station 中的各个组件，以及 SEV 引入的开销。我们表明，即使在近零信任模式下运行，开销也可以忽略不计。

概述。我们将评估分为三个小节，前两个小节回答**RQ1**，并展示两个应用程序的定量和定性结果。最后一个小节介绍 Data Station 开销的特征分析，从而回答**RQ2**。

6.1 机器学习联盟

我们考虑这样一种场景： $N = 8$ 代理希望汇集各自的数据集 D_i，以训练出更强大的机器学习模型，同时不允许其他代理查看其原始数据。仅允许访问联合训练模型的推理结果。使用 Data Station （处于近零信任 模式）时，各代理都会将数据注册到平台，因为他们知道数据将受到端到端保护。我们还考虑一种联邦学习场景，在这种场景中，代理的数据始终留在各自的机器上，模型以分布式方式训练。联邦学习是我们能够部署的、最接近实现预期目标的方案；本节后文将从定性角度比较联邦学习与 Data Station。

Baselines. We consider the following baselines:

• Base。这是一个集中式的不可信服务器，用于训练模型。它可作为与其他基线方案进行比较的参照。• DS。以近零信任 模式运行的 Data Station。

•Flower [10]。一种实现联邦平均聚合算法的先进联邦学习框架 [39]。我们在每轮之后从 2 个客户端中进行抽样，以加快收敛速度，这也是文档中的建议做法。我们在分布式环境中部署 Flower，每个客户端/代理均访问其自身的节点。• OpenFL [46]。一种实现与 Flower 相同聚合算法的先进联邦学习框架，但不会在每个训练周期进行抽样。其对分布式计算的支持尚不成熟；我们将所有客户端部署在单个节点上。

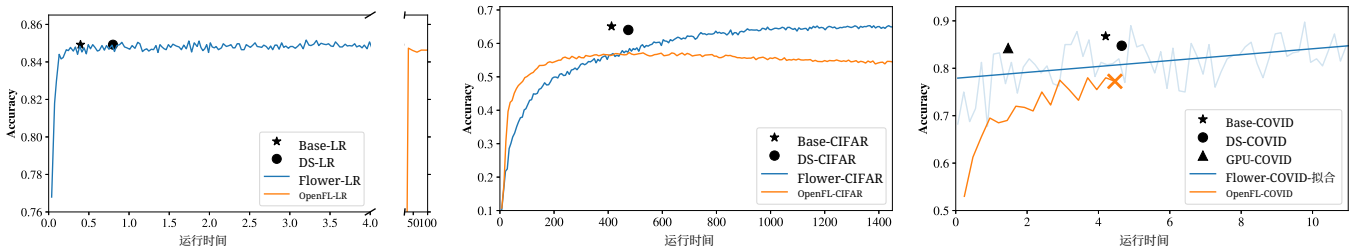


图 2：四种基线方法在成人收入（左）、CIFAR-10（中）和 COVIDx（右）数据集上的准确率和运行时间。

，不会在每个训练周期进行抽样。其对分布式计算的支持尚不成熟；我们将所有客户端部署在单个节点上。

在研究联邦学习基线方法时，我们也考虑了 Pysyft [65] 和 FATE [1]等其他方案。我们选择 Flower 和 OpenFL，是因为它们实现了最先进的联邦聚合算法，且已相当成熟：i）它们得到积极维护；ii）文档齐全；iii）教程中的示例均可运行。

机器学习任务。我们考虑 3 项机器学习任务：• 在 Income 数据集上进行 [55]逻辑回归。该任务旨在预测哪些人的薪资为 <$50K。在联邦学习基线中，每个代理都可以访问该数据集的均匀划分，每份包含 4.5K 个样本。

• 在 CIFAR-10 数据集上进行计算机视觉任务 [31]。每个代理都可以访问 6,250 个样本。
• 在 COVIDx CXR-3 数据集上进行计算机视觉任务 [60]。每个代理都可以访问 3810 张胸部 X 光图像。该任务旨在预测患者是否感染 COVID-19。数据集总大小为 14GB，所用模型为神经网络。

实验设置。我们使用 Google Compute Engine 的 n2d-highmem-8 实例（8 个虚拟 CPU 和 64GB RAM）。具体而言，对于 Flower，我们使用 8+1（客户端+服务器）个此类节点；对于 OpenFL，由于该框架在分布式环境中运行效果不佳，我们在单个节点上运行所有客户端和服务器。我们在单个节点上运行 Base 和 DS，该节点的规格与其他节点相同；运行 DS 时则启用 SEV，以支持近零信任模式。在各基线中，我们对预处理、训练、推理和评估采用相同的逻辑，但有一个例外。由于该框架对其他库的支持有限，OpenFL 在 Income 数据集上的实现使用 Keras 模型（而其他实现使用 sklearn）。

指标与基线。我们评估训练模型所需的时间，以及四种基线所达到的最高准确率。
结果。图 2a、图 2b 和图 2c 展示了三项任务的实验结果。Flower 和 OpenFL 的性能以收敛曲线呈现。当底层收敛行为呈尖峰状时，我们会拟合一条曲线，以提高可读性。对于 Base 和 DS，我们用点标示达到收敛的时刻。我们观察到几种趋势。

首先，在相同的训练时长下，集中式训练（Base 和 DS）的准确率高于联邦学习基线。其次，联邦学习基线需要多花 3 倍的时间（例如，图 2c 中为 11 小时，而 DS 为 4 小时），才能达到与 DS 相近的准确率。对于更复杂的深度网络模型也是如此：CI-FAR（图 2b）和 COVIDx（图 2c）。而在收入实验中使用的简单得多的逻辑回归模型中，集中式方法与 Flower 的表现同样出色；OpenFL 得出结果所需的时间稍长一些，但准确率相近。

其次，两种联邦学习框架的表现相近。在 COVIDx 实验中，OpenFL 稳定性较差，并且会耗尽内存（见图 2c）。请注意，我们以最佳情况运行联邦学习基线，即每个客户端都拥有训练数据的随机样本。若非如此（这在实际中很常见），联邦学习的效率就会降低。

第三，尽管 DS 在近零信任 环境下运行，并确保客户端数据端到端加密，但与 Base 相比，DS 在所有数据集上的运行时开销都极小。不过，在近零信任 环境下运行的 DS 无法在 GPU 上执行计算，因此无法利用 GPU 带来的性能优势。我们在图 2c 中展示了这一点：图中纳入了在 GPU 上运行的 Base，并显示其与 DS 的性能差异。

结果显示，集中化数据和计算在准确性和运行时间方面具有优势，并且与现成的（不可信的）模型训练相比，DataStation 的开销较低。

定性分析。 Data Station 与联邦学习之间存在多项重要的定性差异：
兼容性。 联邦学习仅支持能够合并的机器学习模型。Data Station 没有此类限制。同样，现有应用程序必须经过修改和调整，才能适应联邦学习框架。而 Data Station 只需编写一个简单的连接器，即可公开相关函数。

安全性与泄漏。 许多联邦学习框架会泄漏权重更新 [27]，，而这些更新可能被用来重构部分数据，从而破坏其承诺的安全保障。一种解决方案是将联邦学习与差分隐私 [58] 相结合。这需要对应用程序和算法进行进一步修改，并导致性能进一步下降。

性能差异。许多联邦学习算法无法保证达到与集中式实现相同的性能。我们已在上文对这一点进行了定量展示。

可扩展性。 尽管（原则上）联邦学习可随着数据贡献者数量的增加而扩展，但 Data Station 的当前设计需要进行调整，以支持多机环境。我们预计实现这一目标不会面临严重障碍。

6.2 安全数据共享

在此场景中，所有者将数据存储在第三方服务器上，并通过编写策略，有选择地允许其他代理访问这些数据。在此场景中，我们以近零信任 模式运行 Data Station，并且

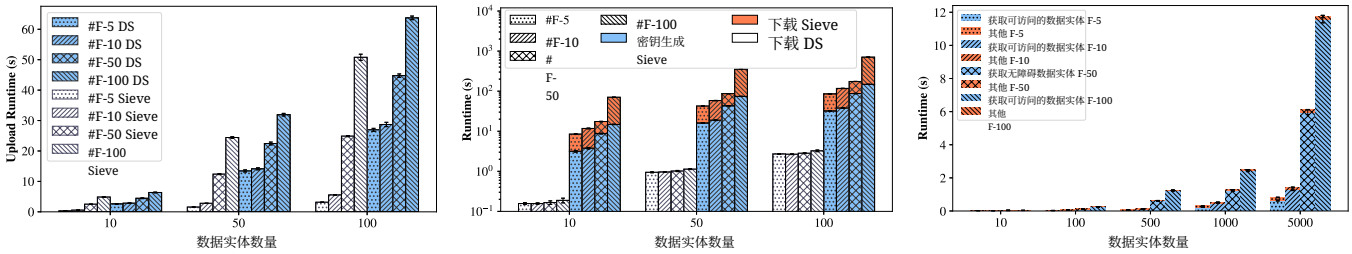


图3：左（中）：Data Station 和 Sieve 的上传（下载）时间。右：Data Station 对数据用户的开销。

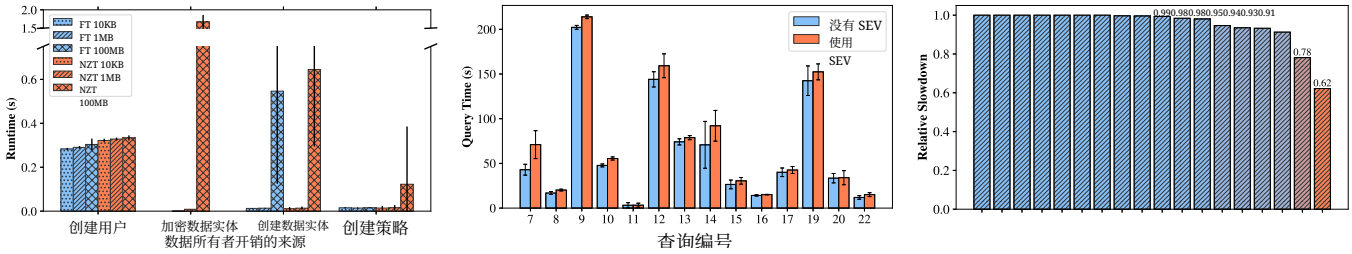


图4：左侧：开销　　面向数据所有者。中间：TPC-H 基准测试上的 SEV 开销。右侧：stress-ng 基准测试　　.

因此会受到飞地和加密协议所带来的开销影响。我们希望了解这些开销是否会导致平台运行速度慢到无法实际使用。为回答这一问题，我们测量运行时间，并将其与 Sieve 的 [57] 进行比较。

Sieve 入门。 Sieve 通过不可信中介实现密码学安全数据共享。与 Data Station 一样，它使用对称密钥加密文件。它使用基于属性的加密（ABE）[22] 加密每个文件的元数据，从而生成仅适用于选定属性的解密密钥。最后，它依赖同态加密 [20]来撤销对数据集的访问权限。与 Data Station 不同，Sieve 专为文件共享而设计。Sieve 是我们发现的唯一一种同时解决安全数据共享问题、采用开源方式，并使用密码学技术建立数据所有者信任的方案。在我们的评估中，Sieve 用于提供性能参考，以便解读 Data Station 在近零信任模式下运行时的性能。

实验设置。 Sieve 可以生成解密密钥，仅解密此前使用特定属性加密的数据。我们在 Data Station 中为每个已注册函数使用一个 Sieve 属性。假设有 n 个数据实体和 m 个函数。Sieve 使用对称密钥加密每个数据集，并使用基于属性的加密（ABE）加密元数据（即函数）。因此，想要调用函数的用户 $f \in m$ 可以在 Sieve 中下载数据集。Data Station 以近零信任模式运行，并实现了一个下载应用程序。download 接收一个数据实体作为输入参数，如果 Gatekeeper 允许，则会将该数据实体发送给用户。这复现了具有相同功能的 Sieve 设置，从而使我们能够比较这两种方案。

指标。 我们通过测量数据所有者上传数据集所需的时间，以及数据用户下载数据集所需的时间，评估数据共享的端到端性能。在 Data Station 中，我们创建 $n * m$ 策略，表示用户可以对任何数据实体调用任何函数。我们使用的文件均由 10KB 的随机字节串组成。我们展示 20 次运行的平均运行时间。

上传结果。 图 3a 展示了使用 Data Station 和 Sieve 上传数据的结果，其中横轴为数据实体数量，不同柱形样式表示已注册函数的数量。首先，当已注册函数的数量（#F-m）较少时，Data Station 的速度比 Sieve 快一个数量级。当函数数量增加到 50 个时，Data Station 比 Sieve 快 80%；达到 100 个函数时，Data Station 仍快 25%。其次，Data Station 和 Sieve 的扩展表现相似，均取决于数据实体和函数的数量。然而，至关重要的是，在 Data Station 中，我们测量的是最坏情况，即每个数据实体都可由任意函数下载。请注意，如果并非如此，Data Station 的开销会降低（取决于数据实体的数量以及实际获准使用的函数），而 Sieve 的开销则保持不变，因为它仍需为该属性加密一个值。最后，在 Data Station 中，主要开销来自创建策略，这需要写入 EWAL，因为它采用近零信任模式。在 Sieve 中，文件的对称加密开销仅随上传的数据实体数量变化。Sieve 最大的开销是基于属性的加密所需时间，在所有数据实体和属性的上传时间中占比超过 95%。即使在允许每个函数对每个数据实体执行的最坏情况下，Data Station 的表现仍优于 Sieve。

下载结果。 图 3b展示了在改变数据实体数量（x 轴）和已注册函数数量（不同柱形样式）时下载数据的结果。用户调用下载后，Gatekeeper 会检查其是否拥有访问权限；随后，函数将加密数据元素读入内存，使用数据所有者的对称密钥对其解密，再使用用户的对称密钥重新加密，并发送给用户。Sieve 的运行时间较长，原因在于使用了基于属性的加密。Sieve 的

数据所有者必须通过基于属性的加密，为所需属性生成解密密钥（密钥生成），如图 3b 底部的柱形所示。生成的密钥会发送给用户，用户使用该密钥解密数据；解密耗时占下载总耗时的一半以上。尽管 Sieve 的设计允许只为用户生成一次密钥，使其获得数据访问权限，但解密和下载的开销仍不可避免。因此，Data Station 相比 Sieve 有两项优势：i）用户下载时无需数据所有者参与；ii）运行时间缩短了 1 到 2 个数量级。

定性分析。除了定量结果外，我们还指出了各方法之间重要的定性差异。

信任模型。与 Data Station 不同，Sieve 不会向基础设施提供商隐藏文件元数据（即策略）。因此，好奇的提供商会了解到平台中有哪些可用函数，以及谁有权访问这些函数，尽管他们无法读取原始数据。Data Station 始终对数据库、EWAL 和审计日志进行加密，以保护这些信息。好奇的提供商可能会了解到代理 ID，但无法将其与用户关联起来。

委托计算。Sieve 不支持委托计算，它充当存储提供商。这意味着，一旦用户获得对某个数据实体的访问权限，Sieve 中的数据所有者就必须直接连接到该用户。这与 Data Station 不同：我们可以提供一个函数来执行此委托计算，在此情形下就是简单的下载函数；但一般而言，也可以提供更复杂的函数，例如上文机器学习应用中所示的函数。

撤销数据访问权限。在 Data Station 中，撤销数据访问权限意味着修改策略，这是一项快速操作，并且可以针对特定用户进行选择性处理。相比之下，在 Sieve 中，这需要更改先前上传数据的属性，然后在存储提供商处重新加密文件。由于无法先解密文件，Sieve 使用了一种巧妙的技术，借助同态加密在不解密的情况下对其重新加密。这会带来几项影响。首先，必须将新生成的密钥重新发送给仍有权访问该文件的用户。其次，由于上述技术，撤销过程本身比在 Data Station 中慢得多。

Sieve 是一个设计完善、工程实现良好的系统。我们认为，这些结果代表了完全基于密码学技术的解决方案，因此验证了 Data Station 的设计。

6.3 DataStation 性能分析

前两个实验表明，与应用程序运行时间相比，Data Station 引入的开销很小。在这里，我们使用一个空操作函数来了解 Data Station 为数据用户和所有者带来的固定 开销。最后，我们将在第 6.3.3 节中测量 AMD SEV 引入的开销。

6.3.1 数据用户的开销。用户调用函数后，潜在的开销来源包括：i) 从数据库获取用户 ID；ii) 从策略代理获取可访问的数据实体；iii) 初始化执行环境；iv) 在函数调用后收集实际访问的数据实体列表；以及 v) 将所有活动记录到审计日志中。

开销因素。在 noop 操作中，从未访问任何数据实体，因此数据实体的特征（例如数据类型）不会影响开销。

数据实体的大小会在需要解密时产生影响（在近零信任模式下运行时），但我们已在第 6.2 节研究过这种影响。这里，我们转而聚焦于上述 5 个步骤。其中，“获取可访问数据实体列表”占据了主要开销，而有两个因素会影响该开销：已注册数据实体的数量和已注册函数的数量。因此，我们展示了这两个因素变化时的结果。

实验设置。在改变已注册函数数量时，我们为每个数据实体和函数对创建一个策略，并将这些策略存储在数据库中。我们报告五次运行的平均端到端运行时间。数据实体的大小和类型均不会影响开销；我们使用代表文本文件的 1MB 数据实体。

结果。图 3c 显示了在改变数据实体数量（x 轴）以及函数数量不同时，每次函数调用的运行时间。我们报告了获取可访问数据实体列表所需的运行时间，并将其他开销来源合并，统称为“其他”。获取可访问数据实体的开销随策略数量线性增长。拥有 5000 个数据实体和 100 个函数（即 50 万条策略）时，开销仅为 12 秒。相比之下，训练上一节中那样的机器学习应用需要数分钟（甚至数小时），而且该应用只有少数几个已注册函数。我们得出结论，对于运行时间超过一分钟的应用程序而言，数据用户所承担的开销可以忽略不计。

6.3.2 数据所有者的开销。数据所有者的开销来源包括：i）在平台上注册；ii）（仅在近零信任模式下）加密待上传的数据实体；iii）上传数据实体；以及 iv）创建策略。与之前不同，由于开销来源 ii）和 iii），数据实体的大小会产生影响，因此我们在本实验中改变数据实体的大小。

结果。图4a展示了横轴所示的开销来源。我们展示了不同大小的数据实体（DE）的结果，以揭示加密数据实体（Encrypt DE）和上传数据实体（Create DE）所引入的开销。我们报告了100次运行的平均值。由于使用SEV并写入EWAL，CreateUser、Create DE和Create Policy的开销在近零信任模式下始终略高，但仍低于1秒。最大的开销来源是加密数据实体（仅在近零信任模式下），其开销取决于数据实体的大小。

6.3.3 AMD SEV 的性能开销。我们使用TPC-H 基准测试来测量AMD的SEV所引入的开销。TPC-H代表了一种读者熟悉的数据密集型工作负载；此外，我们还使用stress-ng [30], 这一用于对计算机系统施加压力、并曾在以往工作中用于了解飞地性能的工具 [21]。我们在一个100GB的数据库上运行DuckDB [45]，分别测试启用和未启用SEV的情况。图4b展示了每条查询运行10次后两种基线的平均运行时间。结果表明，SEV确实会引入开销；其中查询7最为明显，因为它会将大量驻留在磁盘上的数据写入（加密）内存。不过，考虑到所获得的安全保障，这一开销仍然适中。结果进一步印证了现代安全硬件飞地的优势，以及它们所带来的机遇。

stress-ng [30] 使用压力器 来了解系统性能。图 4c 显示了启用 SEV 时压力器的相对减速，测量单位时间内完成的操作数（减速 = 启用 SEV 时完成的操作数 未启用 SEV 时完成的操作数）。引入这些压力器的

最大开销（例如，walk-0a、swap、walk-1a）对应于执行随机访问、内存密集型任务的压力测试程序。walk-Xa 压力测试程序会强制从物理内存中读取数据，这解释了其较大的开销；而 swap 则交换两个不同内存位置的内容。

7 相关工作

Data Station 是首个专注于提供委托式、可信且可审计计算的数据托管系统。它建立在诸多既有研究方向的基础之上，我们将在下文加以介绍。

希波克拉底数据库 [3]。Data Station 与希波克拉底数据库的愿景有关。两者有重要的相似之处，也存在差异。关键在于，两篇论文都围绕目的定义访问控制；在 Data Station 中，目的由引用某个函数的策略指定。因此，与基于访问控制的隐私观 [59]相比，两者的愿景和系统都更倾向于追求情境完整性 [42] 视角下的隐私。另一个相似之处是都需要可审计计算：Data Station 通过审计日志实现，而希波克拉底数据库则采用审计追踪这一概念。两者的差异同样重要。希波克拉底数据库被设想为一种负责保护隐私的数据库系统。Data Station 则是一个用于实现其他应用程序的数据托管系统，其中包括关系数据库管理系统，也包括机器学习及其他基于分析的功能。由于 Data Station 将计算与数据解耦，建立用户和所有者双方的信任便成为至关重要的一环。

数据安全区 存储受限使用的数据，例如受隐私和法规约束的数据 [18]。飞地确保托管的数据受到保护，同时允许用户执行某些预先确定的计算，通常会在数据发布前进行审核 [32, 61]。这类安全飞地常见于研究组织，它们旨在促进数据驱动的研究，例如社会科学领域的 ICPSR [26], 和 NORC [43]。这些安全飞地是长期努力的成果。Data Station 旨在简化数据安全区的创建，包括在组织之间共享数据。

多方计算、同态加密、联邦学习和差分隐私。目前，越来越多的系统旨在支持对受限使用数据集进行协作分析。Shrinkwrap [7] 和 Saqe [8] 允许对来自多个组织的数据执行 SQL 查询，同时确保所得结果具有差分隐私，不会披露任何参与者的身份。Conclave [56], 、Cerebro [63], 和 Secrecy [37] 依靠多方计算实现类似目标。诸如同态加密等替代技术则允许直接对加密数据运行计算。这些技术专注于提供可信的数据处理。不同于现有解决方案和技术，Data Station 旨在解决上述三个要求，而其他解决方案均无法做到这一点。

机密计算项目一直由云服务商推动，旨在与将数据存储在云中的客户建立信任。Azure AlwaysEncrypted [5] 让用户能够在自己不拥有的基础设施（即云）上，对自己拥有的数据进行计算。类似地，Haven [9], SCONE [6], Key- stone [34], Ryoan [23], VC3 [48], 等研究方案可在数据托管于云端时保护数据处理，许多方案都侧重于保护数据库 [19, 51, 62, 64]。近期的一些研究提出了相关基础设施，以便对来自多个参与方的数据运行 SQL 查询 [13]。所有这些解决方案都利用了安全硬件飞地，如 Data Station。但与 Data Station 不同，它们都无法在多个参与方拥有的数据上实现委托式、可信且可审计的计算。

可审计性。 许多人指出了构建审计日志的重要性，例如在物联网环境中 [44], 使用区块链技术 [38, 47,50] 使日志无法被篡改，甚至基于这一抽象构建数据库 [16]。类似地，也有研究探讨如何在数据库环境中推理访问策略 [4]。这些方法所采用的技术与我们在 Data Station 中用于审计日志的技术互为补充。与这些方法不同，Data Station 利用数据和计算的集中化来简化问题。

8 结论

随着组织能够从数据汇集与共享中受益的场景日益增多，人们需要通过技术方案降低组建数据共享联盟的难度。我们介绍了 Data Station，这是一种数据托管系统，旨在通过实现受委托、可信且可审计的计算，促进所有者和用户共享数据并从彼此的数据中获益。Data Station 支持未经修改的应用程序，因此适用范围广泛。我们介绍了基于安全硬件飞地和密码学协议来赢得所有者和用户信任的机制。评估结果表明，与其他应用场景的强基线相比，该方法切实可行，其中包括机器学习训练联盟和安全数据共享。我们研究了其开销，并证明与应用程序运行时间相比，这些开销可以忽略不计。除了定量差异之外，我们还强调了数据托管设计的重要定性优势。

参考文献

[1] [n. d.]. FATE。https://fate.fedai.org/ 在线资源；访问于 2022 年 5 月 29 日。[2] [n. d.]. libfuse 的 Python-FUSE 接口。https://github.com/libfuse/python-fuse. 在线资源；访问于 2022 年 5 月 29 日。[3] Rakesh Agrawal、Jerry Kiernan、Ramakrishnan Srikant 和 Yirong Xu。2002。希波克拉底数据库。载于第 *28* 届超大型数据库国际会议。Elsevier，143–154。[4] Yael Amsterdamer 和 Osnat Drien。2020。通过主动对等探测实现细粒度数据访问控制。载于*EDBT*。403–406。[5] Panagiotis Antonopoulos、Arvind Arasu、Kunal D Singh、Ken Eguro、Nitish Gupta、Rajat Jain、Raghav Kaushik、Hanuma Kodavalla、Donald Kossmann、Nikolas Ogg 等。2020。Azure SQL 数据库始终加密。载于*ACM SIGMOD* 数据管理国际会议。1511–1525。[6] Sergei Arnautov、Bohdan Trach、Franz Gregor、Thomas Knauth、Andre Martin、Christian Priebe、Joshua Lind、Divya Muthukumaran、Dan O'keeffe、Mark L Stillwell 等。2016。SCONE：采用 Intel SGX 的安全 Linux 容器。载于第 *12* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 16*）。689–703。[7] Johes Bater、Xi He、William Ehrich、Ashwin Machanavajjhala 和 Jennie Rogers。2018。Shrinkwrap：私有数据联合中的差分隐私查询处理。*VLDB*基金会论文集 12，3（2018），307–320。[8] Johes Bater、Yongjoo Park、Xi He、Xiao Wang 和 Jennie Rogers。2020。Saqe：面向数据联合的实用隐私保护近似查询处理。*VLDB*基金会论文集 13，12（2020），2691–2705。[9] Andrew Baumann、Marcus Peinado 和 Galen Hunt。2015。借助 Haven 保护应用程序免受不可信云的侵害。*ACM*计算机系统汇刊 33，3，文章 8（8 月 2015 26 10 1145 2799647），页码。https://doi.org/ 。/ [10] Daniel J Beutel、Taner Topal、Akhil Mathur、Xinchi Qiu、Titouan Parcollet、Pedro PB de Gusmão 和 Nicholas D Lane。2020。Flower：一个友好的联邦

学习研究框架。arXiv 预印本 arXiv:2007.14390（2020）。[11] CCG CCE Tech Pubs - Intel Corp. 2022。第 12 代 Intel® CoreTM 处理器——数据手册，第 1 卷（共 2 卷）。
h
t
t
p
s://
w
ww.intel.com/content/www/us/en/products/docs/processors/core/core-technical-resources.html. 在线资源；访问于 2022 年 2 月 28 日。[12] Chris Clifton、Murat Kantarcio˘glu、AnHai Doan、Gunther Schadow、Jaideep Vaidya、Ahmed Elmagarmid 和 Dan Suciu。2004。隐私保护的数据集成与共享。载于第 9 届 ACM SIGMOD 数据挖掘与知识发现研究议题研讨会。19–26。[13] Ankur Dave、Chester Leung、Raluca Ada Popa、Joseph E Gonzalez 和 Ion Stoica。2020。利用硬件安全飞地实现隐蔽的竞合分析。载于第 15 届欧洲计算机系统会议。1–17。[14] Tim Dierks 和 Eric Rescorla。2008。传输层安全协议（TLS）1.2 版。（2008）。[15] Peter F Edemekong、Pavan Annamaraju 和 Micelle J Haydel。2018。《健康保险可携性和责任法案》。（2018）。[16] Muhammad El-Hindi、Carsten Binnig、Arvind Arasu、Donald Kossmann 和 Ravi Ramamurthy。2019。BlockchainDB：区块链上的共享数据库。VLDB基金会论文集 12，11（2019），1597–1609。[17] Raul Castro Fernandez、Ziawasch Abedjan、Famien Koko、Gina Yuan、Samuel Madden 和 Michael Stonebraker。2018。Aurum：数据发现系统。载于 IEEE 第 34 届国际数据工程会议。IEEE，1001–1012。[18] Ian Foster。2018。用于安全分析敏感数据的研究基础设施。《美国政治与社会科学学会年刊》675，1（2018），102–120。[19] Benny Fuhry、HA Jayanth Jain 和 Florian Kerschbaum。2021。Encdbdb：使用安全飞地实现可搜索加密、快速、压缩的内存数据库。载于 IEEE/IFIP 第 51 届可靠系统与网络国际年会。IEEE，438–450。[20] Craig Gentry。2009。一种全同态加密方案。博士论文。斯坦福大学。https://crypto.stanford.edu/craig/craig-thesis.pdf。[21] Christian Göttel、Rafael Pires、Isabelly Rocha、Sébastien Vaucher、Pascal Felber、Marcelo Pasin 和 Valerio Schiavoni。2018。硬件辅助内存保护机制在安全性、性能和能耗方面的权衡。载于 IEEE 第 37 届可靠分布式系统研讨会。IEEE，133–142。[22] Vipul Goyal、Omkant Pandey、Amit Sahai 和 Brent Waters。2006。用于加密数据细粒度访问控制的基于属性的加密。载于第 13 届 ACM 计算机与通信安全会议。89–98。[23] Tyler Hunt、Zhiting Zhu、Yuanzhong Xu、Simon Peter 和 Emmett Witchel。2018。Ryoan：用于在秘密数据上执行不可信计算的分布式沙箱。ACM计算机系统汇刊 35，4（2018），1–32。[24] Intel Corporation。2017。Intel® 软件防护扩展（Intel® SGX）安全飞地的飞地内存测量工具。https: //
w
w
w.
i
n
t
e
l.
c
o
m/
content/da
m/develop/external/us/en/documents/enclave-measurement-tool-intel-sgx-737361.pdf。在线资源；访问于 2022 年 2 月 24 日。[25] Intel Corporation。2021。Intel® 64 和 IA-32 架构软件开发人员手册，第 3D 卷：系统编程指南，第 4 部分。
https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html。在线资源；访问于 2022 年 2 月 23 日。[26] 政治与社会研究校际联盟。2022。ICPSR 数据安全区。
https://www.icpsr.umich.edu/web/pages/ICPSR/access/restricted/enclave.html。在线资源；访问于 2022 年 2 月 18 日。[27] Peter Kairouz、H Brendan McMahan、Brendan Avent、Aurélien Bellet、Mehdi Bennis、Arjun Nitin Bhagoji、Kallista Bonawitz、Zachary Charles、Graham Cormode、Rachel Cummings 等。2021。联邦学习的进展与未决问题。《机器学习基础与趋势》14，1–2（2021），1–210。[28] Poul-Henning Kamp 和 Robert NM Watson。2000。监禁环境：限制无所不能的 root 用户。载于第 2 届 SANE 国际会议，第 43 卷。116。[29] David Kaplan、Jeremy Powell 和 Tom Woller。2016。AMD 内存加密。白皮书（2016）。[30] Colin Ian King。[n. d.]。stress-ng。
https://github.com/ColinIanKing/stress-ng 在线资源；访问于 2022 年 5 月 29 日。[31] Alex Krizhevsky、Geoffrey Hinton 等。2009。从微小图像中学习多层特征。（2009）。[32] Julia Lane、Pascal Heus 和 Tim Mulcahy。2008。网络世界中的数据访问：利用网络基础设施。《数据隐私汇刊》1，1（2008），2–16。[33] 联邦学习。2017。无需集中式训练数据的协作式机器学习。发布日期：2017 年 4 月 6 日，星期四。[34] Dayeol Lee、David Kohlbrenner、Shweta Shinde、Krste Asanović 和 Dawn Song。2020。Keystone：用于构建可信执行环境的开放框架。载于第 15 届欧洲计算机系统会议。1–16。[35] Mengyuan Li、Yinqian Zhang 和 Zhiqiang Lin。2021。CROSSLINE：攻破 AMD SEV 中基于“崩溃安全”的内存隔离。载于 ACM SIGSAC 计算机与通信安全会议。2937–2950。[36] Mengyuan Li、Yinqian Zhang、Huibo Wang、Kang Li 和 Yueqiang Cheng。2021。{CIPHERLEAKS}：通过密文侧信道攻破 AMD SEV 上的恒定时间加密。载于第 30 届 USENIX 安全研讨会（USENIX Security *21)*. 717–732 .
[37] John Liagouris、Vasiliki Kalavri、Muhammad Faisal 和 Mayank Varia。2021 年。Secrecy：基于秘密共享数据的安全协作分析。arXiv 预印本 arXiv:2102.01048（2021）。[38] Sujaya Maiyya、Victor Zakhary、Mohammad Javad Amiri、Divyakant Agrawal 和 Amr El Abbadi。2019 年。区块链的数据库与分布式计算基础。在数据管理国际会议上。2036–2041。[39] H. Brendan McMahan、Eider Moore、Daniel Ramage、Seth Hampson 和 Blaise Agüera y Arcas。2016 年。利用去中心化数据进行深度网络的通信高效学习。（2016 年）。
https://doi.org/10.48550/ARXIV.1602.05629[40] Chandrasekaran Mohan、Don Haderle、Bruce Lindsay、Hamid Pirahesh 和 Peter Schwarz。1992 年。ARIES：一种支持细粒度锁定和部分回滚的事务恢复方法，使用预写日志。ACM 数据库系统汇刊（TODS）17，1（1992），94–162。[41] Nightingale Open Science。2022 年。
https://www.nightingalescience.org/。在线；访问日期：2022 年 2 月 25 日。[42] Helen Nissenbaum。2004 年。隐私即情境完整性。《华盛顿法律评论》79（2004），119。[43] NORC。2022 年。NORC 数据飞地。
https://www.norc.org/Research/Capabilities/ Pages/data-enclave.aspx。在线；访问日期：2022 年 2 月 18 日。[44] Nisha Panwar、Shantanu Sharma、Guoxi Wang、Sharad Mehrotra、Nalini Venkata-subramanian、Mamadou H Diallo 和 Ardalan Amiri Sani。2021 年。IoT 公证人：IoT 环境中可证明的传感器数据捕获。ACM 物联网汇刊 3，1（2021），1–30。[45] Mark Raasveldt 和 Hannes Mühleisen。2019 年。DuckDB：可嵌入式分析数据库。在数据管理国际会议上。1981–1984。[46] G Anthony Reina、Alexey Gruzdev、Patrick Foley、Olga Perepelkina、Mansi Sharma、Igor Davidyuk、Ilya Trushkin、Maksim Radionov、Aleksandr Mokrov、Dmitry Agapov、Jason Martin、Brandon Edwards、Micah J. Sheller、Sarthak Pati、Prakash Narayana Moorthy、Shih han Wang、Prashant Shah 和 Spyri-don Bakas。2021 年。OpenFL：一个开源联邦学习框架。arXiv:2105.06413 [cs.LG][47] Mark Russinovich、Edward Ashton、Christine Avanessians、Miguel Castro、Amaury Chamayou、Sylvan Clebsch、Manuel Costa、Cédric Fournet、Matthew Kerner、Sid Krishna 等。2019 年。CCF：构建机密可验证复制服务的框架。技术报告，Microsoft Research 和 Microsoft Azure（2019）。[48] Felix Schuster、Manuel Costa、Cédric Fournet、Christos Gkantsidis、Marcus Peinado、Gloria Mainar-Ruiz 和 Mark Russinovich。2015 年。VC3：使用 SGX 在云中进行可信数据分析。在 IEEE 安全与隐私研讨会上。IEEE，38–54。[49] AMD SEV-SNP。2020 年。通过完整性保护等机制强化虚拟机隔离。白皮书，1 月（2020 年）。[50] Alex Shamis、Peter Pietzuch、Miguel Castro、Edward Ashton、Amaury Chamayou、Sylvan Clebsch、Antoine Delignat-Lavaud、Cedric Fournet、Matthew Kerner、Julien Maffre 等。2021 年。PAC：CCF 的实用问责机制。arXiv 预印本 arXiv:2105.13116（2021）。[51] Yuanyuan Sun、Sheng Wang、Huorong Li 和 Feifei Li。2021 年。为实用的加密数据库构建原生飞地存储引擎。《VLDB基金会学报》14，6（2021），1019–1032。[52] SUSE。2022 年。AMD SEV 指南。
https://documentation.suse.com/sles/15-SP2/html/
SLES-amd-sev/art-amd-sev.html。在线；访问日期：2022 年 2 月 28 日。[53] Miklos Szeredi。2010 年。FUSE：用户空间中的文件系统。http://fuse. s ourceforge. net（2010）。[54] Carol Tenopir、Suzie Allard、Kimberly Douglass、Arsev Umur Aydinoglu、Lei Wu、Eleanor Read、Maribeth Manoff 和 Mike Frame。2011 年。科学家的数据共享：实践与认知。《PloS one》6，6（2011），e21101。[55] UCI。2022 年。成人收入数据集。
https://www.kaggle.com/wenruliu/adult- income-dataset。在线；访问日期：2022 年 3 月 1 日。[56] Nikolaj Volgushev、Malte Schwarzkopf、Ben Getchell、Mayank Varia、Andrei Lapets 和 Azer Bestavros。2019 年。Conclave：大数据上的安全多方计算。在第 14 届 EuroSys 会议上。1–18。[57] Frank Wang、James Mickens、Nickolai Zeldovich 和 Vinod Vaikuntanathan。2016 年。Sieve：针对不受信任云中用户数据的密码学强制访问控制。在第 13 届 USENIX 网络系统设计与实现研讨会（NSDI 16）上。611–626。[58] Kang Wei、Jun Li、Ming Ding、Chuan Ma、Howard H Yang、Farhad Farokhi、Shi Jin、Tony QS Quek 和 H Vincent Poor。2020 年。带差分隐私的联邦学习：算法与性能分析。《IEEE 信息取证与安全汇刊》15（2020），3454–3469。[59] Alan F Westin。1968 年。隐私与自由。《华盛顿与李法律评论》25，1（1968），166。[60] Alex Wong。[n. d.]。COVID-Net。
https://github.com/AlexSWong/COVID-Net 在线；访问日期：2022 年 5 月 21 日。[61] Jiaan Zeng、Guangchen Ruan、Alexander Crowell、Atul Prakash 和 Beth Plale。2014 年。用于非消费性文本使用的云计算数据胶囊。在第 5 届 ACM 科学云计算研讨会上。9–16。

[62] Wenting Zheng、Ankur Dave、Jethro G Beekman、Raluca Ada Popa、Joseph E Gonzalez 和 Ion Stoica。2017。Opaque：一个无意识加密分布式分析平台。载于第 *14* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 17*）。283–298。

[63] Wenting Zheng、Ryan Deng、Weikeng Chen、Raluca Ada Popa、Aurojit Panda 和 Ion Stoica。2021。Cerebro：一个用于{多方} 密码学协作学习的平台。载于第 *30* 届 *USENIX* 安全研讨会（*USENIX Security 21*）。2723–2740。

[64] Jinwei Zhu、Kun Cheng、Jiayang Liu 和 Liang Guo。2021。全加密：GaussDB 中的一种端到端加密机制。*VLDB*基金会会议录 14，12（2021），2811–2814。

[65] Alexander Ziller、Andrew Trask、Antonio Lopardo、Benjamin Szymkow、Bobby Wagner、Emma Bluemke、Jean-MickaelNounahon、Jonathan Passerat-Palmbach、Kritika Prakash、Nick Rose 等。2021。Pysyft：一个便于开展联邦学习的库。收录于 联邦学习系统。施普林格，111–139。