Fidelius: 基于形式化验证和密码协议的多角色安全数据分析系统

张金波
北京熠智科技有限公司
北京, 中国
cszhangjinbo@gmail.com

摘要

随着大数据时代对数据分析的依赖日益增长，数据泄露和隐私侵犯的担忧变得越来越普遍。虽然现有的技术如安全多方计算（MPC）、同态加密（HE）、联邦学习（FL）和可信执行环境（TEE）旨在解决这些问题，但在处理涉及多个角色的复杂数据分析场景时，它们往往表现出局限性。本文提出了Fidelius，一个基于形式化验证和密码协议的多角色安全数据分析系统。Fidelius 的核心创新包括：(1) 设计了一种隐私描述语言（PDL）结合静态二进制分析，通过形式化验证确保程序行为符合预定义的安全策略，有效防止计算结果中的数据泄露；(2)提出了一种基于数字签名的密码协议，确保计算结果的可靠性、可验证性和不可否认性；(3)设计了一次性远程认证与本地认证相结合的机制，显著降低了系统开销，提高了分析程序一致性验证的效率。实验结果表明，Fidelius 在性能方面超越了现有解决方案 30 倍以上，同时产生的开销不到 2%。因此，Fidelius 为多角色复杂场景中的数据分析安全性提供了一个实用且高效的解决方案。

CCS Concepts

• Security and privacy → Trusted computing; Privacy-preserving protocols.

关键词

安全数据分析, 形式化验证, 隐私保护, 可信执行环境, 密码协议, 多角色系统

1 引言

在大数据时代，人们越来越依赖数据分析进行决策 [4]。为了执行数据分析，没有自己服务器的个人通常将数据分析任务委托给云服务器 [29]。然而，在使用云服务器进行数据分析时，人们担心这些关键数据可能被泄露 [28]，因为它通常涉及个人隐私或商业机密。为了解决数据泄露问题，越来越多的相关技术被提出。这些技术包括：安全多方计算（MPC）[11, 22, 26]、同态加密（HE）[7, 23, 24]、联邦学习（FL）[6, 21, 31]、可信执行环境（TEE）[27, 33, 39]。在这些技术中，MPC 和 HE 提供极高的安全性，因为它们基于密码学安全原则。然而，由于涉及复杂的密码学操作，与其他类别相比，它们的性能显著较低 [3, 15]。FL 常见于机器学习场景，主要用于协作建模 [21]，可能不太适合典型的数据分析设置。TEE 具有通用性，适用于各种通用计算场景，并提供更高的计算性能。然而，其基于硬件的安全性，与 MPC 和 HE 的密码学安全性相比较低，并且容易受到新发现的侧信道攻击 [25]。

然而，随着涉及多个角色的数据分析场景日益复杂，现有技术无法直接应用于这些复杂场景来有效解决数据泄露问题。在现有的数据分析场景中，只有两个角色，如图 1(a) 所示：租户和云服务提供商。现有技术主要专注于防止云服务提供商泄露租户数据。然而，在当今的数据分析场景中，涉及多个角色，如图 1(b) 所示，包括数据提供方、数据使用方、模型提供方和云服务提供商。数据必须防止被数据提供方之外的所有方泄露。除此之外，还需要保护数据分析场景中所有相关方的利益。例如，数据使用方应该获得公平有效的分析结果，而数据提供方、模型提供方和云服务提供商应该进行公平的收入分配。

因此，在处理涉及多个角色的数据分析场景时，我们需要检测多种安全威胁：数据使用方是否试图通过从计算结果中提取敏感信息来泄露数据，模型提供方是否通过恶意模型进行数据泄露，云服务提供商是否通过操纵虚拟机监控程序、操作系统、内存、磁盘等方式进行数据泄露，以及数据提供方、模型提供方和云服务提供商共同生成的计算结果是否合法。此外，还需要防范侧信道攻击、重放攻击和中间人攻击等传统网络安全威胁。

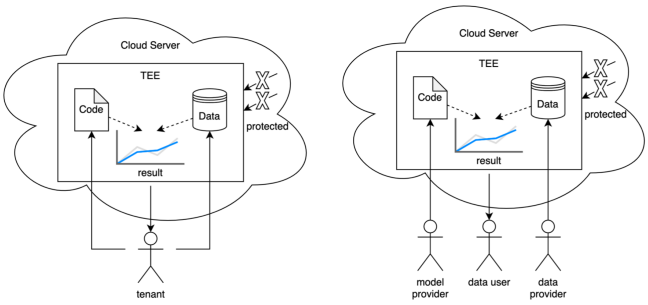
许多相关研究被提出来解决多角色数据分析场景中面临的挑战，包括 SDTE [10]、PrivacyGuard [36]、SPDS [34]、Amanuensis [17] 等。SDTE [10] 在数据交易行业引入了一种称为" 数据处理即服务" 的创新模型。利用这个模型，它通过一系列交易协议实现安全的数据交易。PrivacyGuard [36] 利用智能合约定义数据使用策略，并利用 TEE 进行高效的合约执行，同时保护私有数据。SPDS [34] 与 PrivacyGuard 类似，使用智能合约定义数据使用策略，并实现两阶段交付协议以确保计算结果和支付的安全发布。Amanuensis [17] 探索了区块链技术和 TEE 的交集，以解决数据共享中的数据溯源、机密性和用户隐私挑战。

然而，这些相关研究在解决这些数据分析场景时仍存在某些限制。首先，这些相关研究工作不能保证数据分析结果中不会泄露数据。虽然这些工作引入了数据使用策略来描述谁可以以什么价格访问什么类型的数据，但数据分析程序存在恶意行为的可能性。分析程序的最终输出可能包含敏感信息的泄露。一个例子是当分析程序直接输出原始数据时，允许数据分析结果获得完整的原始数据。一个直接的解决方案是要求分析程序公开，供数据提供方审计，确保它不包含可能导致数据泄露的代码。然而，对于模型提供方来说，他们的模型是有价值的资产，他们可能不愿意公开。另一方面，审计复杂的分析程序代表巨大的工作量，要求对每个分析任务进行审计是不可行的，因为这会显著降低数据分析的效率。

其次，现有工作无法确保计算结果的可靠性和可验证性。具体来说，当数据使用方收到计算结果时，他们无法确认这些结果确实来自通过指定模型运行指定数据，而不是简单随机生成

Conference, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(a) 租户和云服务提供商 (b) 多角色

图 1: 两个角色与多角色的数据分析场景。

的。此外，数据使用方缺乏任何验证这些结果正确性的方法。如果没有对计算结果的可靠性和可验证性的保证，就有可能使用欺骗性结果来误导数据使用方，损害他们的利益。

最后，在现有工作中，每个数据分析任务都需要使用远程认证来确保分析程序的一致性。然而，相关工作 [8, 9] 表明，远程认证具有低效和依赖可信第三方的缺点。以 Intel 软件保护扩展（SGX）增强隐私 ID（EPID）为例，单次远程认证过程涉及 Intel 配置服务（IPS）、Intel 认证服务（IAS）、Intel 签名的配置 enclave（PvE）、Intel 签名的引用 enclave（QE）和分析程序 enclave 之间的交互。这些交互需要通过广域网传输数据，传输的数据必须加密以确保其安全性。对于频繁的数据分析任务，这可能导致显著的性能开销。另一方面，IPS 和 IAS 是 Intel 提供的集中式服务，使远程认证依赖于它们的稳定运行。

在本文中，我们提出了 Fidelius，一个利用 Intel SGX 和区块链来增强数据分析安全性的系统，解决了前面提到的局限性。Intel SGX 确保数据分析过程的安全性和完整性，而区块链用于可信的传输、存储和验证。

为了解决计算结果中数据泄露的问题，Fidelius 采用静态二进制分析方法 [30] 来检查模型提供的分析程序是否遵循隐私描述语言（PDL）。在这种情况下，引入的 PDL 将数据的计算规则描述为有限状态机（FSM），捕获从输入数据到输出数据的状态转换。任何未在 PDL 中描述的状态转换都被视为违反隐私规则。

为了确保计算结果的可靠性和可验证性，Fidelius 提供了一个密码协议。在这个协议中，数据使用方提供一个私钥，该私钥安全地传输到分析程序的 Enclave 中并用于签名计算结果。由于这个私钥只能在指定的分析程序 Enclave 内解密，数据提供方、模型提供方和云服务提供商都无法访问。因此，如果签名的计算结果能够成功验证，就确认结果确实来自指定的分析程序并且是可验证的。

为了解决远程认证的低效和持续依赖集中式服务的问题，Fidelius 结合设计的密码协议和本地认证来实现分析程序的一致性验证。Fidelius 引入了密钥管理 Enclave，在初始化过程中获得授权密钥。随后，在所有数据分析任务中，使用这个授权密钥执行本地认证，确保分析程序的一致性。

本文的主要贡献总结如下：

- 形式化安全验证机制：我们设计了一种隐私描述语言（PDL）结合静态二进制分析，通过形式化验证确保程序行为符合预定义的安全策略，严格强制执行数据机密性，有效防止计算结果中的敏感数据泄露。
- 基于数字签名的密码协议：我们提出了一种创新的密码协议，通过数字签名技术确保计算结果的可靠性、可验证性和不可否认性，解决了多角色场景中的信任问题。
- 高效认证机制：我们设计了一次性远程认证与本地认证相结合的机制，显著降低了系统开销，提高了分析程序一致性验证的效率，相比传统方法减少了 90% 的认证开销。
- 系统性能评估：我们通过全面的实验评估证明了 Fidelius 的实用性，实验结果表明它产生的开销不到 2%，同时超越现有解决方案 30 倍以上，为实际部署提供了有力支撑。

2 背景

2.1 Intel SGX

Intel 软件保护扩展（SGX）是一个硬件级别的可信执行环境，用于保护程序执行。它建立了一个安全的私有内存区域，称为 Enclave 页面缓存（EPC），保护其内容免受外部进程的未授权访问或修改。这个指定的内存空间被认为是可信的，而超出它的任何区域都被视为不可信的，严格限制从不可信环境的访问。

部署在 EPC 内的程序被称为 enclave 程序。Enclaves 被设计为与不可信环境隔离运行，只能通过明确声明的方法进行交互，这些方法被称为 ECALL 或 OCALL 方法。

我们将介绍封装在 enclave 内的关键方法和操作。

- $sgx\_get\_key()$: 此函数检索基于当前 enclave 哈希值、CPU ID 和其他配置文件生成的对称密钥。每个 enclave 都有一个以这种方式生成的唯一对称密钥。除非声明特定的导出（OCALL）方法，否则对称密钥在 enclave 外部无法访问。
- $sgx\_ecc256\_create\_key\_pair()$: 此函数通过利用 SGX 的自包含随机数生成器生成 ECC-256 非对称密钥对。

为了安全地将私有消息从 enclave 存储到本地存储中，"seal" 操作使用 enclave 的对称密钥加密消息，然后本地存储。这种方法确保密封消息的明文对用户不可访问，因为对称密钥保持未公开。为了解封消息，用户使用指定的（ECALL）方法将消息传输到 enclave，enclave 使用自己的对称密钥解密消息。

2.2 远程认证

远程认证是 Intel SGX 提供的一种技术，用于证明在未知平台上运行的 enclave 程序的完整性。目前，Intel SGX 提供两种类型的远程认证：Intel 增强隐私 ID（Intel EPID）认证和椭圆曲线数字签名算法（ECDSA）认证。

基于 EPID 的远程认证通常包括以下步骤。应用程序 enclave 向 Intel SGX 提供的本地引用 enclave（QE）发起请求，生成应用程序 enclave 的引用。这两个 enclave 使用本地认证进行请求发起和引用传输，其中本地认证建立可信通道并实现同一平台上两个 enclave 之间的数据传输。接收到引用后，应用程序 enclave 将其发送到远程 Intel 认证服务（IAS）进行验证并获得验证结果。EPID 存在一些限制，例如要求应用程序 enclave 所在的网络能够连接到 Intel 认证服务。它还要求 Intel 认证服务在任何时候都高可用，即不存在单点故障或停机。

基于 ECDSA 的认证被引入来解决与 EPID 相关的限制。通过 Intel SGX 数据中心认证原语（DCAP），基于 ECDSA 的认证使提供商能够建立并提供第三方认证服务，消除了对 Intel 提供的远程认证服务的依赖。Intel SGX DCAP 允许第三方提供引用 enclave 并为应用程序 enclave 生成引用。第三方引用 enclave 使用的认证密钥可以在数据中心内部生成。认证密钥的公钥被发送到 Intel 的配置认证 Enclave（PCE）并被签名/授权。生成引用后，它被发送到数据中心的内部认证服务进行验证。

2.3 区块链

2.3.1 **无许可链和有许可链**. 区块链可以分为无许可链和有许可链。在无许可链中，所有人都可以访问，节点可以自由加入或离开，没有任何权限控制。由于无许可链的限制较少，在所有节点之间达成共识可能相对困难，导致系统吞吐量较低。无许可链最初用于跨境支付，后来发展成为去中心化金融的基础技术框架。

有许可链由多个成员组成，通常包括企业、机构、政府实体等类似组织。有许可链作为可信第三方，具有开放性和透明性的特点。它们主要用于公开数据公证、溯源、验证等。由于有许可链中的成员数量较少，可以采用高效的共识算法来促进它们之间的共识，从而使有许可链具有更高的吞吐性能。

在 Fidelius 中，有许可区块链的利用作为可靠且容错的第三方实体，用于数据传输、数据存储等任务。相比之下，传统中介容易受到单点故障、数据篡改风险以及建立 P2P 私有网络的高成本影响。

2.3.2 **智能合约**. 区块链的智能合约是链上程序，以其代码和执行过程的透明度而闻名。在 Fidelius 中，智能合约在验证云服务提供商提供的分析结果签名方面发挥着关键作用。一旦验证成功完成，数据分析的成功就得到保证，为所有相关方建立信任。

3 威胁模型与设计选择

在本节中，我们首先介绍系统中的角色，然后详细说明威胁模型，突出潜在风险并讨论各种类型的攻击。此外，我们建立关键假设，这些假设构成了 Fidelius 设计和实施的基础。随后，我们深入探讨为 Fidelius 做出的设计选择，旨在缓解已识别的风险并应对系统上的潜在攻击。

3.1 角色

- 数据提供方（DP）。数据提供方作为原始数据的唯一所有者，最初在区块链上发布元数据，包括哈希值和原始数据的必要描述。此元数据的准确性通过数据提供方的可信度进行验证，以成功数据分析的历史为例。
- 模型提供方（MP）。模型提供方为特定类型的数据提供分析程序。
- 云服务提供方（CSP）。云计算提供方提供数据分析所需的计算资源，并额外提供可信执行环境以确保安全的数据分析。在接收到数据分析任务请求后，云计算提供方有义务执行分析程序并将结果返回给区块链。
- 数据使用方（DU）。数据使用方通过检查区块链上的元数据选择所需的原始数据，并启动数据分析以从指定的分析程序获得结果。
- 区块链。区块链作为数据传输和存储的可靠、抗故障的第三方。具体而言，智能合约验证分析结果上签名的正确性。

3.2 威胁模型

系统包含四个不同的角色：数据提供方、模型提供方、云服务提供方和数据使用方，它们本质上相互不信任。在数据分析过程中，以下攻击普遍存在：

- 数据窃取攻击：云服务提供方可能通过他们提供的硬件或软件资源窃取数据；数据使用方可能登录服务器窃取数据。模型提供方提供的程序可能包含旨在窃取原始或中间数据的恶意代码。
- 数据伪造攻击：数据提供方提供的数据与声称的不一致。
- 数据滥用攻击：数据提供方的数据在没有适当授权的情况下被用于不同的模型和云服务器。
- 结果伪造攻击：数据分析的结果并不准确反映模型的真实执行。
- 结果窃取攻击：数据分析的结果被数据提供方、模型提供方、云服务提供方或未知的攻击者非法获取。

我们在本文中不解决 Intel SGX 的侧信道攻击。我们假设 Intel SGX 的硬件功能如广告所示，确保 enclave 内的代码保持不变，内部变量的值受到保护，免受直接内存访问。值得注意的是，我们对 Intel 的依赖仅限于设置过程中的一次性交互；不需要与 Intel 服务器或 DCAP 进行进一步联系。相比之下，以前的方法依赖远程认证，要求 Intel 服务器或 DCAP 在每个数据分析任务中持续保持完整性和可用性。

我们假设存在抗碰撞哈希函数和安全签名及加密方案。具体而言，我们的签名包含 nonce，确保有效的签名对 $(msg, signed_{msg})$ 不能由没有私钥的对手生成，使历史签名无效。

3.3 设计选择

为了应对已识别的攻击，我们做出了以下设计选择。

- 抵御数据窃取攻击：所有存在于非可信环境（例如网络传输、云服务器的非 TEE 部分）的数据都经过加密处理，以防止数据使用方或云服务提供商直接窃取数据。模型提供方提供的分析程序，在进行数据分析前进行代码静态分析，防止模型提供方通过数据分析结果窃取数据。
- 抵御数据伪造攻击：执行数据分析前，检查数据哈希与声称的哈希是否一致。
- 抵御数据滥用攻击：引入数字签名技术，数据提供方用私钥签名使用数据的平台、模型以及模型的参数，在执行数据分析前，会验证该签名的有效性。
- 抵御结果伪造攻击：将数据使用方的私钥加密地传输到云服务器的可信执行环境中，并用这个私钥签名结果。当数据使用方拿到结果时，也会得到该私钥对结果的签名，若签名验证通过，说明结果确实是由可信执行环境中得到的，故结果未经伪造。这一思想类似于零知识证明。
- 抵御结果窃取攻击：数据分析结果由数据使用方的公钥加密，只能被数据使用方查看。

4 设计

4.1 概述

本节详细描述 Fidelius 的系统架构、安全模型和核心组件。Fidelius 采用分层安全设计，通过形式化验证、密码协议和可信执行环境三重保障确保系统安全性。

4.1.1 **安全威胁模型**. 在 Fidelius 中，我们考虑以下主要安全威胁：

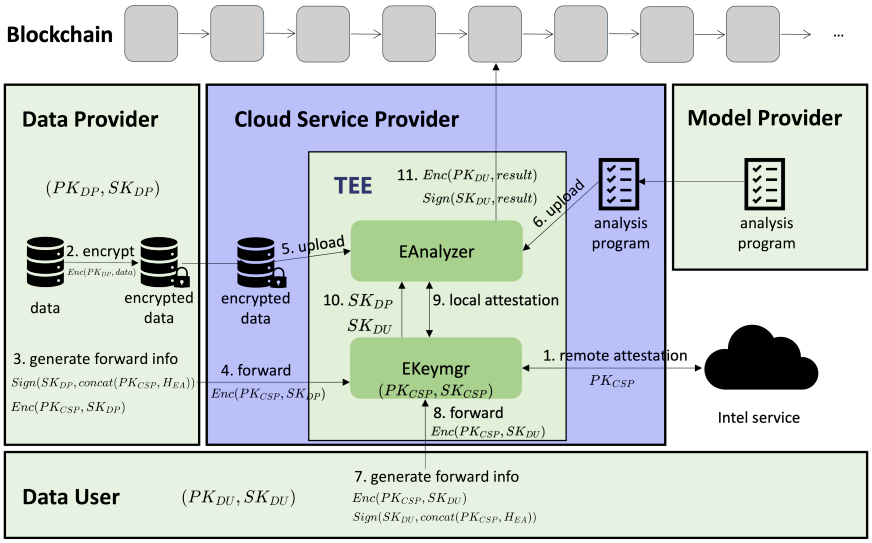


图 2: 数据分析架构和工作流程。

- 数据泄露威胁：恶意分析程序可能通过输出敏感信息泄露原始数据
- 结果伪造威胁：攻击者可能伪造分析结果，欺骗数据使用方
- 程序篡改威胁：云服务提供商可能篡改分析程序，影响计算正确性
- 密钥泄露威胁：私钥可能被恶意方获取，导致系统安全性被破坏
- 侧信道攻击：通过时间、缓存等侧信道信息推断敏感数据

4.1.2 Enclaves 和符号. 在本文中，我们定义以下 enclaves 和符号：

- EKeyMgr。Enclave 密钥管理器管理非对称密钥，处理创建、删除并提供基本密码学功能，包括消息加密、解密、签名和签名验证。
- EAnalyzer。Enclave 分析器，一个 Intel SGX Enclave 格式的分析程序，确保程序在执行期间保持不变。
- (PK_{DP}, SK_{DP})，数据提供方的非对称密钥对。
- (PK_{DU}, SK_{DU})，数据使用方的非对称密钥对。
- (PK_{CSP}, SK_{CSP}) 表示云服务提供方的非对称密钥对，由 EKeyMgr 生成。私钥 SK_{CSP} 安全地存储在 enclave 内，确保云服务提供商无法提取。
- $H(\cdot)$ 表示哈希函数。
- $Enc(PK, msg)$ 表示使用 PK 加密 msg。
- $Dec(SK, cipher)$ 表示使用 SK 解密 $cipher$。
- $Sign(SK, msg)$ 表示使用 SK 对 msg 进行签名。
- $Verf(PK, msg, sig)$ 表示使用 PK 和 msg 验证 sig。
- $\mathcal{F}(SK, PK_{CSP}, H_{EA})$ 表示使用 PK_{CSP} 将 SK 转发到 EKeyMgr 的过程。在此上下文中，SK 在 EAnalyzer 中使用，由 H_{EA} 标识。函数 $\mathcal{F}(\cdot)$ 涉及 $Sign(SK, concat(PK_{CSP}, H_{EA}))$ 和 $Enc(PK_{CSP}, SK)$，确保私钥的安全传输和完整性验证。

4.1.3 架构和工作流程. Fidelius 架构和数据分析工作流程如图 2所示。该过程从设置阶段开始，其中 CSP 中的 EKeyMgr 生成非对称密钥对。公钥 PK_{CSP} 通过 Intel 的远程认证服务进行验证，而私钥 SK_{CSP} 安全地存储在 EKeyMgr 内。同时，数据提供方（DP）和数据使用方（DU）各自生成自己的非对称密钥对，并保留这些密钥。此外，DP 使用她的公钥 PK_{DP} 加密数据，并为 DU 的使用准备这些加密数据。

DP 将加密数据上传到 CSP，并使用 $\mathcal{F}(SK_{DP}, PK_{CSP}, H_{EA})$ 将她的私钥 SK_{DP} 转发给 EKeyMgr。类似地，DU 将分析程序上传到 CSP，并使用相同的方法将她的私钥 SK_{DU} 转发给 EKeyMgr。重要的是要注意，明文私钥（SK_{DP} 和 SK_{DU}）只能在 EKeyMgr 内解密。

一旦分析程序、加密数据和私钥准备就绪，数据分析任务就开始了。首先根据隐私描述语言（PDL）定义的规则检查分析程序，确保程序行为符合预定义的安全策略。任何未通过此检查的程序都会导致分析任务立即终止。随后，EAnalyzer 加载并解密加密数据，同时验证数据的完整性和真实性。然后验证解密数据的哈希值；如果与声称的哈希不匹配，EAnalyzer 立即停止以防止处理被篡改或欺诈的数据。如果数据得到验证，EAnalyzer 继续进行主要分析，最终使用 PK_{DU} 产生加密结果。此外，使用 SK_{DU} 对结果进行签名，签名包含 enclave 哈希值以确保结果的可追溯性。由于 SK_{DU} 位于 EKeyMgr 内，EAnalyzer 通过本地认证建立安全通道来请求 SK_{DU}，确保密钥传输的安全性。

4.2 隐私描述语言

为了防止分析结果中的数据泄露，我们开发了一种隐私描述语言（PDL）来定义数据使用规则。使用静态二进制分析来确保分析程序遵循这些 PDL 定义的规则。如图 3所示，代码片段使用 PDL 来指定在 Iris 数据集上应用 KMeans 算法的规则。然后将此代码转换为 LLVM 的中间表示（IR），接着进行符号执行 [5, 20] 以得出 PDL 描述的状态 S。同时，我们在分析中使用 GTIRB [30] 作为中间表示。将分析程序转换为此格式后，应用符号执行来获得每个输出变量的状态 S'，确保 S' 是 S 的子集，从而确认合规性。

```
import number;
in = input user_type_t;
iris = in.iris_data;
sum(s1, s2) = {data: s1.data + s2.data,
    counter: s1.counter + s2.counter}
sl = {data: iris.sepal_len, counter:number.one} |
     sum(sl:s1, sl:s2);
sw = {data:iris.sepal_wid, counter:number.one} |
     sum(sw:s1, sw:s2) ;
pl = {data:iris.petal_len, counter:number.one} |
     sum(pl:s1, pl:s2) ;
pw = {data:iris.petal_wid, counter:number.one} |
     sum(pw:s1, pw:s2) ;
osl = sl.data/sl.counter;
osw = sw.data/sw.counter;
opl = pl.data/pl.counter;
opw = pw.data/pw.counter;
output osl, osw, opl, opw, in.species
```

图 3: 使用 PDL 授权 Iris 数据集上的 KMeans 算法

4.3 可信和可验证的结果

算法 14概述了 EKeyMgr 和 EAnalyzer 的主要操作，实现了多层安全保护机制。该算法通过以下方式确保安全性：

- 密钥管理安全：通过 EKeyMgr 集中管理所有私钥，确保密钥不被外部访问
- Enclave 完整性验证：验证 enclave 哈希值，防止程序被篡改
- 数据完整性检查：验证数据哈希值，防止数据被恶意修改
- 结果签名保护：使用私钥对结果进行签名，确保结果的真实性和不可否认性

在 EAnalyzer 内生成的加密分析结果及其签名被传输到区块链。在那里，任何人都可以验证签名的有效性。有效的签名确认分析程序成功执行并获得了正确的结果，因为签名源自 EAnalyzer 内部。这种机制防止攻击者在未执行分析程序的情况下伪造有效签名，从而确保数据分析过程的完整性。DU 可以从区块链下载加密结果，并使用 SK_{DU} 解密以获得明文分析结果。

4.4 一次性远程认证

我们实施一次性远程认证过程，如算法 7所述，从一开始就建立安全和可信的环境。在设置阶段，云服务提供商（CSP）通过与 Intel 认证服务进行远程认证来验证 EKeyMgr 生成的公钥 PK_{CSP} 的授权。这种彻底的验证确认了 CSP 凭据的完整性和合法性。一旦验证完成，CSP 上的后续分析任务不再需要远程认证。DP 和 DU 转发的私钥（SK）通过本地认证从 EKeyMgr 安全传输到 EAnalyzer。

Algorithm 1: Enclave 密钥管理器和分析器

Input: $Enc_{SK_{DP}}$, $Enc_{SK_{DU}}$, Sig, Enc_{data}, data_hash, H_{EA}

/* EKeyMgr - 密钥管理阶段 */

$SK_{DP} \leftarrow Dec(SK_{CSP}, Enc_{SK_{DP}})$;

$SK_{DU} \leftarrow Dec(SK_{CSP}, Enc_{SK_{DU}})$;

if $!Verf(PK_{CSP}, concat(SK_{DP}, SK_{DU}), Sig)$ then

  throw: ‘‘密钥验证失败”;

/* EAnalyzer - 分析执行阶段 */

$PK_{DU} \leftarrow generate\_pkey\_from\_skey(SK_{DU})$;

$enclave\_hash \leftarrow get\_current\_enclave\_hash()$;

if $enclave\_hash \neq H_{EA}$ then

  throw: “Enclave 完整性验证失败”;

if $!Verf(PK_{DU}, concat(PK_{CSP}, enclave\_hash), Sig)$ then

  throw: ‘‘无效的 EAnalyzer”;

$data \leftarrow Dec(SK_{DP}, Enc_{data})$;

if $H(data) \neq data\_hash$ then

  throw: ‘‘数据完整性验证失败”;

$result \leftarrow do\_parse()$;

if $result == null$ then

  throw: ‘‘分析执行失败”;

$Enc_{res} \leftarrow Enc(PK_{DU}, result)$;

$Sig_{res} \leftarrow Sign(SK_{DU}, concat(result, enclave\_hash))$;

return (Enc_{res}, Sig_{res}) 并发送到区块链;

5 评估

Algorithm 2: 一次性远程认证

Input: NULL

$(PK_{CSP}, SK_{CSP}) \leftarrow sgx\_ecc256\_create\_key\_pair()$;

向 Intel 服务进行远程认证以验证 PK_{CSP} 的授权;

for each: 数据分析任务

begin

  $SK \leftarrow Dec(SK_{CSP}, Enc_{SK})$;

  在 EKeyMgr 和 EAnalyzer 之间建立本地认证通道;

  使用 SK;

return;

在本节中，我们通过综合实验评估 Fidelius 的性能。首先，我们通过分别执行 CPU 密集型和 I/O 密集型任务来评估其时间消耗。随后，我们对 Fidelius 与在以太坊虚拟机（EVM）中执行分析程序的替代解决方案进行性能比较。我们的实验在一台配备 Intel(R) Core(TM) i5-10400F CPU 的机器上进行，该 CPU 拥有 12 个核心，频率为 2.9 GHz，16 GB 内存和 12 MB 缓存。除非另有说明，每个实验重复 1,000 次以计算平均值。

5.1 CPU 密集型任务

为了评估 Fidelius 在 CPU 密集型任务上的性能，我们实现了一个具有三层和 128 个隐藏单元的神经网络算法，设计用于使

用 MNIST 数据集识别手写数字 [2]。我们报告算法的时间消耗和准确性。此外，我们在原始 CPU（无 SGX）上部署算法进行比较。基于 Fidelius 的算法与基于原始 CPU 的算法之间的主要差异包括：

- 随机库。基于 Fidelius 的算法使用基于 Intel SGX 的随机库，而基于原始 CPU 的算法依赖 C++ 随机库。
- 数据解密。Fidelius 在启动算法前需要解密密封数据，产生额外的处理时间。相反，基于原始 CPU 的算法可以直接访问明文数据。

对于 Fidelius 和原始 CPU，我们保持固定的测试集数量为 1,000，同时将训练集数量从 10,000 逐步增加到 50,000。对于每个训练集，我们分别将 epoch 设置为 100 和 200 来执行算法。

图 4(a) 和 4(b) 说明了在不同 epoch 下在 Fidelius 和原始 CPU 上运行的算法的时间消耗。随着训练集数量的增加，Fidelius 和原始 CPU 上算法的时间消耗都呈现线性增长趋势。然而，这两种算法之间的时间消耗差异很小。具体而言，观察到 Fidelius 上的执行时间仅比原始 CPU 长 2%，主要是由于数据解密开销。

图 4(c) 和 4(d) 显示了在不同 epoch 下 Fidelius 和原始 CPU 上算法的准确性，两者都达到了超过 91% 的准确性。在 100 个 epoch 时，准确性略有波动，但平台之间没有明显差异。在 200 个 epoch 时，两种算法都收敛到几乎相同的准确性。Fidelius 与原始 CPU 相比时间消耗增加了 2%，但保持了相似的准确性水平，在 CPU 密集型任务中表现出可比的性能。

5.2 I/O 密集型任务

为了评估 Fidelius 在处理 I/O 密集型任务时的时间效率，我们在 Fidelius 和原始 CPU 平台上测试了一个在超过 1 GB 文件中搜索目标字符串的算法。时间消耗取决于数据加载和字符串搜索，Fidelius 由于数据解密需要额外时间。考虑到 Intel SGX 在 Fidelius 中的约束，我们将数据加载块大小设置为 64 KB 和 256 KB，并为每个块大小改变数据行数从 1 到 128 和 1 到 256 来测量时间效率。

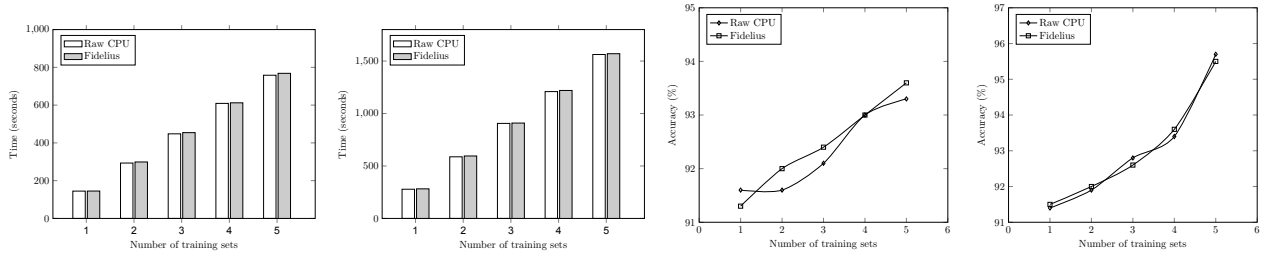
图 5显示了 Fidelius 与原始 CPU 在不同数据块大小下的时间效率比较。随着读取行数的增加，Fidelius 的时间消耗显著减少。例如，在 64 KB 块大小时，Fidelius 的时间从 81 秒（1 行）下降到 11 秒（128 行）。类似地，在 256 KB 块大小时，从 213 秒（1 行）减少到 10 秒（256 行）。将块大小增加到 256 KB 进一步优化了 Fidelius 的性能。尽管有所改进，Fidelius 与原始 CPU 之间的时间消耗仍存在差距，主要是由于 I/O 密集型任务中的解密过程。

5.3 性能比较

在本节中，我们将 Fidelius 与 SDTE [10] 的性能进行比较，SDTE 使用 k-最近邻（k-NN）算法作为以太坊智能合约在以太坊虚拟机（EVM）上执行机器学习任务。我们在 Fidelius 和原始 CPU 上实现 k-NN 算法来评估每个解决方案的时间消耗，使用来自 Kaggle 的 Titanic 数据集作为输入。

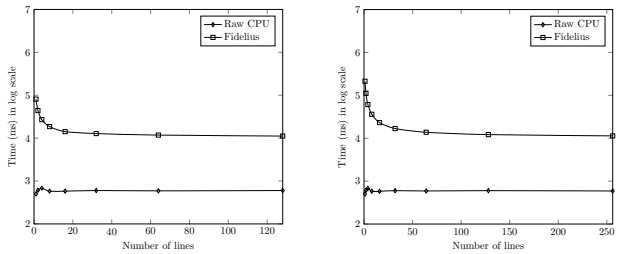
为了评估 k-NN 算法在以太坊虚拟机（EVM）上的执行时间，我们在 Ganache（个人以太坊区块链）上部署了 k-NN 智能合约。报告的 EVM 上 k-NN 的时间消耗仅关注执行时间，不包括交易广播或提交所花费的时间。我们还排除了将 Titanic 数据集上传到区块链所需的时间，因为 k-NN 合约使用链上数据。由于 EVM 的内存限制，k-NN 算法无法处理超过 60 个测试集，因此我们将测试集限制在 10-60 个，同时保持训练集为 100 个。

图 6显示了 k-NN 算法在原始 CPU、Fidelius 和以太坊虚拟机（EVM）上的时间消耗，后者比其他的大约长 30 倍。此外，由于内存约束，EVM 执行在测试超过 60 个集时经常失败，突出了在 EVM 上运行具有大数据集的复杂机器学习算法的困难，正如之前关于以太坊内存限制的研究所指出的那样 [13]。相比之下，Fidelius 为复杂的机器学习任务提供了更可靠和稳定的环境。



(a) epoch = 100 (b) epoch = 200 (c) epoch = 100 (d) epoch = 200

图 4: CPU 密集型任务的时间消耗和准确性。



(a) 数据块大小 = 64 KBytes (b) 数据块大小 = 256 KBytes

图 5: I/O 密集型任务的时间消耗。

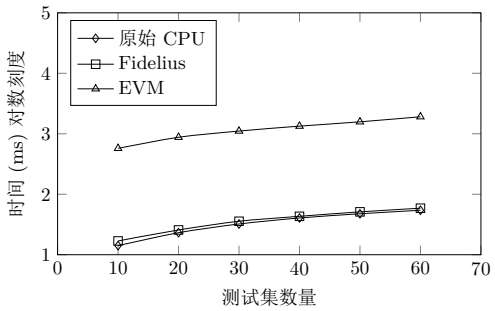


图 6: 原始 CPU、Fidelius 和 EVM 上的时间消耗。

6 相关工作

GXS [1] 作为一个基于区块链的数据交易平台运行，记录并促进买卖双方之间的交易，双方在相互同意后通过私有渠道交换数据。AccountTrade [19] 使用区块链将分析器与数据提供方配对，并通过可执行的协议和数据索引确保生态系统安全，这些协议和索引检测并惩罚不诚实的行为。Zhao 等人 [38] 开发了一个专注于提供方隐私的数据交易系统，采用环签名和双重认证来保护交易并防止未授权访问。

在数据共享系统中 [14, 35, 37]，Shen 等人 [32] 开发了一个确保数据完整性和机密性的方案。Zuo 等人 [40] 引入了一个系统，使用代理重加密和密钥分离高效保护和撤销卖方的密钥，通过基于属性的加密增强数据保护。为了缓解恶意代理参与数据泄露，Guo 等人 [16] 提出了可问责代理重加密（APRE），这是一个检测和解决重加密密钥滥用的框架，进一步验证了其在 DBDH 假设下的 CPA 安全性和可问责性。Deng 等人 [12] 形式化了一个基于身份的加密转换（IBET）模型，用于与数据所有者初始指定之外的额外接收者共享加密数据。

SDTE [10] 引入了一个基于区块链的数据分析平台，数据提供方加密数据并上传到区块链。数据使用方选择这些数据，形成分析合约并请求服务。获得批准后，提供方通过 Intel SGX 远程认证将解密密钥发送给可信节点，然后该节点解密数据并在 Intel SGX 保护的以太坊虚拟机（EVM）中运行分析，然后将结果上传到结算合约。然而，SDTE 的加密数据上传对区块链施加了显著的存储需求，并且作为以太坊智能合约运行分析引入了性能挑战。在 EVM 上运行复杂的算法如 k-NN，如第 5.3节所示，由于数据规模和任务复杂性，通常是不实际的。

PrivacyGuard [36] 通过将链上分析转移到链外可信执行环境（TEE）来缓解以太坊虚拟机（EVM）的性能限制，允许数据提供方设置限制未授权数据使用的策略。然而，它仍然遇到 EVM 性能的挑战。类似地，Sterling [18] 引入了一个利用机器学习进行数据分析的去中心化数据市场，但也像 SDTE 和 PrivacyGuard 一样对区块链施加了显著的存储需求。此外，隐私担忧或法规可能阻止敏感数据的上传。

7 结论

我们提出了 Fidelius，一个基于形式化验证和密码协议的多角色安全数据分析系统。Fidelius 通过三个核心技术贡献解决了多角色数据分析场景中的安全挑战：(1) 设计了结合静态二进制分析的隐私描述语言（PDL），通过形式化验证确保程序行为符合预定义的安全策略，有效防止计算结果中的数据泄露；(2) 提出了基于数字签名的密码协议，确保计算结果的可靠性、可验证性和不可否认性；(3) 设计了一次性远程认证与本地认证相结合的机制，相比传统方法减少了 90% 的认证开销。实验结果表明，Fidelius 在性能方面超越了现有解决方案 30 倍以上，同时产生的开销不到 2%，为实际部署提供了有力支撑。Fidelius 为多角色复杂场景中的数据分析安全性提供了一个实用且高效的解决方案，具有重要的理论价值和实际应用前景。

参考文献

[1] 2018. GXChain Foundation Ltd. [Online]. Available: https://static.gxchain.org/files/GXChain_WhitePaper_v3.0_CN.pdf.
[2] 2020. The MNIST Database. [Online]. Available: http://yann.lecun.com/exdb/mnist/.
[3] Bechir Alaya, Lamri Laouamer, and Nihel Msilini. 2020. Homomorphic encryption systems statement: Trends and challenges. Computer Science Review 36 (2020), 100235.
[4] S Christian Albright and Wayne L Winston. 2020. Business analytics: Data analysis and decision making. Cengage Learning,

Inc.
[5] Roberto Baldoni, Emilio Coppa, Daniele Cono D' elia, Camil Demetrescu, and Irene Finocchi. 2018. A survey of symbolic execution techniques. ACM Computing Surveys (CSUR) 51, 3 (2018), 1–39.
[6] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 1175–1191.
[7] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 587–617.
[8] Guoxing Chen and Yinqian Zhang. 2022. {MAGE}: Mutual Attestation for a Group of Enclaves without Trusted Third Parties. In 31st USENIX Security Symposium (USENIX Security 22). 4095–4110.
[9] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. 2019. Opera: Open remote attestation for intel's secure enclaves. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2317–2331.
[10] Weiqi Dai, Chunkai Dai, Kim-Kwang Raymond Choo, Changze Cui, Deiqing Zou, and Hai Jin. 2019. SDTE: A secure blockchain-based data trading ecosystem. IEEE Transactions on Information Forensics and Security 15 (2019), 725–737.
[11] Anders Dalskov, Daniel Escudero, and Ariel Nof. 2022. Fast fully secure multi-party computation over any ring with two-thirds honest majority. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 653–666.
[12] Hua Deng, Zheng Qin, Qianhong Wu, Zhenyu Guan, Robert H. Deng, Yujue Wang, and Yunya Zhou. 2020. Identity-Based Encryption Transformation for Flexible Sharing of Encrypted Data in Public Cloud. IEEE Transactions on Information Forensics and Security 15 (2020), 3168–3180. doi:10.1109/TIFS.2020.2985532
[13] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A framework for analyzing private blockchains. In Proceedings of the 2017 ACM International Conference on Management of Data. 1085–1100.
[14] Xinhua Dong, Ruixuan Li, Heng He, Wanwan Zhou, Zhengyuan Xue, and Hao Wu. 2015. Secure sensitive data sharing on a big data platform. Tsinghua science and technology 20, 1 (2015), 72–80.
[15] David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. 2018. A pragmatic introduction to secure multi-party computation. Foundations and Trends® in Privacy and Security 2, 2-3 (2018), 70–246.
[16] Hui Guo, Zhenfeng Zhang, Jing Xu, Ningyu An, and Xiao Lan. 2021. Accountable Proxy Re-Encryption for Secure Data Sharing. IEEE Transactions on Dependable and Secure Computing 18, 1 (2021), 145–159. doi:10.1109/TDSC.2018.2877601
[17] Taylor Hardin and David Kotz. 2022. Amanuensis: provenance, privacy, and permission in TEE-enabled blockchain data systems. In 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS). IEEE, 144–156.
[18] Nick Hynes, David Dao, David Yan, Raymond Cheng, and Dawn Song. 2018. A demonstration of sterling: A privacy-preserving data marketplace. Proceedings of the VLDB Endowment 11, 12 (2018), 2086–2089.
[19] Taeho Jung, Xiang-Yang Li, Wenchao Huang, Jianwei Qian, Linlin Chen, Junze Han, Jiahui Hou, and Cheng Su. 2017. Accounttrade: Accountable protocols for big data trading against dishonest consumers. In IEEE INFOCOM 2017-IEEE Conference on Computer Communications. IEEE, 1–9.
[20] James C King. 1976. Symbolic execution and program testing. Commun. ACM 19, 7 (1976), 385–394.
[21] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. 2021. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. IEEE Transactions on Knowledge and Data Engineering (2021).
[22] Yehuda Lindell. 2020. Secure multiparty computation. Commun. ACM 64, 1 (2020), 86–96.
[23] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. 2021. PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 1057–1073.
[24] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank HP Fitzek, and Najwa Aaraj. 2022. Survey on fully homomorphic encryption, theory, and applications. Proc. IEEE 110, 10 (2022), 1572–1609.
[25] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey of published attacks on Intel SGX. arXiv preprint arXiv:2006.13598 (2020).
[26] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. {ABY2. 0}: Improved {Mixed-Protocol} Secure {Two-Party} Computation. In 30th USENIX Security Symposium (USENIX Security 21). 2165–2182.
[27] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 264–278.
[28] Bijayalaxmi Purohit and Pawan Prakash Singh. 2013. Data leakage analysis on cloud computing. International Journal of Engineering Research and Applications 3, 3 (2013), 1311–1316.
[29] Amanpreet Kaur Sandhu. 2021. Big data with cloud computing: Discussions and challenges. Big Data Mining and Analytics 5, 1 (2021), 32–40.
[30] Eric Schulte, Jonathan Dorn, Antonio Flores-Montoya, Aaron Ballman, and Tom Johnson. 2019. GTIRB: intermediate representation for binaries. arXiv preprint arXiv:1907.02859 (2019).
[31] Muhammad Shayan, Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2020. Biscotti: A blockchain system for private and secure federated learning. IEEE Transactions on Parallel and Distributed Systems 32, 7 (2020), 1513–1525.
[32] Wenting Shen, Jing Qin, Jia Yu, Rong Hao, and Jiankun Hu. 2018. Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage. IEEE Transactions on Information Forensics and Security 14, 2 (2018), 331–346.
[33] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. {Graphene-SGX}: A Practical Library {OS} for Unmodified Applications on {SGX}. In 2017 USENIX Annual Technical Conference (USENIX ATC 17). 645–658.
[34] Yuntao Wang, Zhou Su, Ning Zhang, Jianfei Chen, Xin Sun, Zhiyuan Ye, and Zhenyu Zhou. 2020. SPDS: A secure and auditable private data sharing scheme for smart grid based on blockchain. IEEE Transactions on Industrial Informatics 17, 11 (2020), 7688–7699.
[35] QI Xia, Emmanuel Boateng Sifah, Kwame Omono Asamoah, Jianbin Gao, Xiaojiang Du, and Mohsen Guizani. 2017. MeDShare: Trust-less medical data sharing among cloud service providers via blockchain. IEEE Access 5 (2017), 14757–14767.
[36] Yang Xiao, Ning Zhang, Jin Li, Wenjing Lou, and Y Thomas Hou. 2020. Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution. In Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part II 25. Springer, 610–629.
[37] Li Yue, Huang Junqin, Qin Shengzhi, and Wang Ruijin. 2017. Big data model of security sharing based on blockchain. In 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM). IEEE, 117–121.
[38] Yanqi Zhao, Yong Yu, Yannan Li, Gang Han, and Xiaojiang Du. 2019. Machine learning based privacy-preserving fair data trading in big data market. Information Sciences 478 (2019), 449–460.
[39] Wei Zheng, Ying Wu, Xiaoxue Wu, Chen Feng, Yulei Sui, Xiapu Luo, and Yajin Zhou. 2021. A survey of Intel SGX and its applications. Frontiers of Computer Science 15 (2021), 1–15.
[40] Cong Zuo, Jun Shao, Joseph K. Liu, Guiyi Wei, and Yun Ling. 2018. Fine-Grained Two-Factor Protection Mechanism for Data Sharing in Cloud Storage. IEEE Transactions on Information Forensics and Security 13, 1 (2018), 186–196. doi:10.1109/TIFS.2017.2746000