计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202440489
61(10): 2467−2481, 2024

面向轻量级设备的云存储场景数据完整性校验方案

韩 冰[1] 王 昊[1] 方 敏[2] 张永超[3] 周 璐[1] 葛春鹏[4,5]

[1](南京航空航天大学计算机科学与技术学院 南京 211106)

[2](华东师范大学数据科学与工程学院 上海 200062)

[3](东南大学网络空间安全学院 南京 211189)

[4](山东大学软件学院 济南 250101)

[5](山东大学-南洋理工大学人工智能国际联合研究院 济南 250101)

(bling@nuaa.edu.cn)

Data Integrity Verification Scheme For Lightweight Devices in Cloud Storage Scenarios

Han Bing[1], Wang Hao[1], Fang Min[2], Zhang Yongchao[3], Zhou Lu[1], and Ge Chunpeng[4,5]

[1](*College of Computer Science and Technology*, *Nanjing University of Aeronautics and Astronautics*, *Nanjing* 211106)

[2](*School of Data Science and Engineering*, *East China Normal University*, *Shanghai* 200062)

[3](*School of Cyberspace Security*, *Southeast University*, *Nanjing* 211189)

[4](*School of Software*, *Shandong University*, *Jinan* 250101)

[5](*Joint SDU-NTU Centre for Artificial Intelligence Research* (*C-FAIR*), *Software School*, *Shandong University*, *Jinan* 250101)

Abstract Lightweight mobile devices with limited resources often alleviate their computational and storage burdens by outsourcing large-scale data to cloud storage servers. However, this cloud storage model is susceptible to the possibility of selfish cloud servers discarding data to conserve storage resources. Therefore, there is a need for effective integrity verification of cloud-stored data to ensure its correct and intact storage. Existing cloud storage integrity verification mechanisms lack a reliable approach to perform real-time, multiple verifications of data under the premise of data privacy protection. We propose an integrity verification mechanism based on a trusted execution environment. It generates trustworthy proofs in isolated areas to ensure that the cloud server remains unaware of the data and the entire proof generation process, thereby compelling honest assurance of data integrity throughout the process. To further enhance the security of the proposed solution, we introduce blockchain smart contracts to provide trustworthy storage and verification of proofs. Additionally, we address the issue of resource scarcity on the client side by proposing an efficient verification mechanism based on cuckoo filters. Experimental results demonstrate that this method can achieve high execution efficiency and practicality while ensuring the integrity verification of private data.

Key words integrity verification; cloud storage; trusted execution environment; blockchain; cuckoo filter; privacy-preserving

摘 要 资源受限的轻量级移动设备往往可以通过将大规模数据外包至云存储服务器中从而卸载自身的

收稿日期: 2024-05-31; **修回日期**: 2024-07-18

基金项目: 国家重点研发计划项目(2021YFB2700503); 国家自然科学基金项目(62071222, 62032025, U21A20467, U20A20176, U22B2030); 江苏省自然科学基金项目(BK20220075); 深圳市科学技术计划项目(JCYJ20210324134810028)

This work was supported by the National Key Research and Development Program of China (2021YFB2700503), the National Natural Science Foundation of China (62071222, 62032025, U21A20467, U20A20176, U22B2030), the Natural Science Foundation of Jiangsu Province (BK20220075), and the Shenzhen Science and Technology Program (JCYJ20210324134810028).

通信作者: 周璐(lu.zhou@nuaa.edu.cn)

计算和存储压力. 然而该云存储模式存在自私云服务器丢弃数据以节省存储资源的可能性. 因此需要能够对云储存数据进行有效的完整性校验以确保数据正确完好地存储着. 然而现有的云存储完整性校验机制在缺乏可靠且能够满足数据隐私保护的前提下对数据进行即时、多次校验的机制. 提出了一种基于可信执行环境的完整性校验机制, 通过在隔离区域中对数据产生可信证明, 保证了云服务器在全过程中对数据以及产生证明的全过程的不可见, 从而不得不诚实地保证存储数据的完整性. 为了进一步提高方案的安全性, 引入了区块链智能合约以提供证明的可信存证和验证. 此外, 还考虑到了端侧设备的资源不足问题, 提出了基于布谷鸟过滤器的高效验证机制. 实验结果表明, 该方法能够在保证隐私数据的完整性校验的基础上, 实现较高的执行效率和实用性.

关键词 完整性校验; 云存储; 可信执行环境; 区块链; 布谷鸟过滤器; 隐私保护

中图法分类号 TP309

端云场景中端侧设备往往是资源受限的轻量级设备, 比如移动手机、传感器、无人机等. 这些轻量级设备往往需要不间断地收集外部数据信息从而产生较大的算力和存储容量需求, 而其硬件配置不支持其提供相匹配的需求, 因此往往需要将端侧设备的数据传至中心化的云服务提供者处进行存储或计算以释放其存储负载. 然而, 这种托管模式存在着众多安全和隐私挑战. 首先, 在很多情况下, 来自端侧设备的数据具有隐私保护的需求, 不希望以明文的形式直接交付至云侧. 其次, 存放在云侧的数据的完整性需要得到保证. 自私的云服务器可能为了节省开销或者其他原因有意或者无意地丢弃部分或全部数据. 因此, 需要能够在数据隐私保护的前提下, 对云侧进行验证以确认数据被完好地存储. 再次, 轻量级端侧设备往往在数据托管后删除本地数据以释放空间, 这使得它们不再具有对数据信息的掌控力. 但矛盾的是, 往往它们有对存储数据进行多次验证的需求, 这种多次验证可能是定期的, 也可以是非定期的, 因此需要提供一种机制以确保用户在无需保留数据的前提下, 可以随时任意地对自己的数据进行校验.

第 1 个挑战可以轻易地以加密的形式解决, 用户对自己的数据进行加密后再发送给云. 第 2 个挑战可以用数字签名的方式来实现. 基于哈希函数的抗碰撞性, 对整个数据进行哈希, 一旦云存储的数据出现缺损, 则无法得到同样的哈希值. 然而, 实际的需求还需要满足第 3 个挑战, 即端侧设备应该有随时验证完整性的能力. 具体来说, 云端需要保证在提供存储服务期间, 能够对于来自端侧的任何形式的挑战提供完整性证明. 简单的签名机制, 即通过让云服务器提供数据的签名再由端侧进行验签的方式不再奏效, 因为端侧在发送了自己的数据之后, 本地往往不会继续存有原始数据, 而签名需要对比摘要是否一致, 从而得到摘要值, 自私的云端仅需要保存存储数据的哈希值即可通过验签挑战. 因此, 简单的签名机制不具备能够满足端侧多次任意的验证需求.

已有的方案也考虑到了这个问题, Li 等人[1]引入了基于区块链和梅克尔树的方案, 保证数据的哈希值在链上的可信数据结构中存储; Garg 等人[2]引入了基于 Schnorr 签名的第三方审计验证方案; Fan 等人[3]采用了基于身份加密和聚合签名的方案, 并于可信执行环境执行; Zhao 等人[4]利用 Lifted EC-ElGamal 密码体系、双线性配对和区块链技术, 提出了适用于 IoT 设备的验证方案. 这些方案中可能需要云或者第三方能够掌握一定的数据信息才能执行, 这对用户数据的机密性造成了破坏, 或者无法保证云或攻击者对证明进行伪造的恶意行为. 我们在表 1 中对相关工作展开了详细的对比. 因此, 如何在保护轻量级端侧数据机密性的前提下, 完成对云端数据完整性长期验证是一项艰巨的任务.

Table 1 Property Comparison of Data Integrity Check Schemes

表 1 数据完整性校验方案性质对比

策略	是否需要第三方	隐私保护级别	是否不可伪造	是否可信存证	性能是否增益	是否需要硬件	是否轻量级
文献 [1] 方案	否	高	否	是	否	否	否
文献 [2] 方案	是	低	是	否	否	否	否
文献 [3] 方案	否	高	是	否	是	是	否
文献 [4] 方案	是	低	是	是	否	否	否
本文方案	否	低	是	是	是	是	是

是否有方法可以让云服务器在无法获取到明文数据的情况下完成数据完整性验证计算? 可信执行环境(trusted execution environment, TEE)可以帮助我们做到这一点. 通过创造一个连云服务器本身都无

法访问的可信区域，将密文加载进去进行验证计算并产生计算过程的可信证明，既保证了数据的机密性，又防止云在验证计算过程中可能的作弊行为.

本文考虑了数据云存储中的隐私保护和数据完整性保证的需求，保证了在云服务器无法访问到明文数据的前提下，实现了对云存储数据的即时完整性验证. 具体来说，本文通过可信执行环境创造的隔离执行区域，将数据加载进去，并在该执行区域实现对明文数据的数字签名的计算过程，并将签名值以密文的形式传出，从而保证云服务器无法通过保存签名的方式通过下一次验证，端侧设备在获取到加密的签名后，进行解密并验签，从而判断数据是否完好地被存储. 通过这样的方式，云端不得不诚实地存储完整的数据，无法通过其他作弊的方式通过完整性验证.

进一步地，本文还考虑到轻量级端侧设备的算力和存储容量问题. 由于像传感器这样的终端设备需要源源不断地产生数据，随着数据量的增长，对指定数据的签名验证的检索效率较为低下. 为了提供快速高效的适配轻量级设备的验证机制，本文还设计了基于布谷鸟过滤器的端侧本地验证机制. 端侧本地保存托管数据的摘要值(哈希值)并添加进布谷鸟过滤器中，在验证时将解密后的摘要值输入进布谷鸟过滤器进行查找，如果结果返回查找到了或查找成功则证明验证成功. 除了较高的查找效率，相对于传统的存储数据结构，布谷鸟过滤器通常需要更少的内存空间来存储相似规模的数据集，也为端侧设备降低了存储负载.

本文的主要贡献包括 4 个方面：

1)提出了一种基于可信执行环境的数据完整性验证方案，通过在可信执行环境中执行产生密文数据的可信证明，在保证数据的隐私性的前提下，实现了对云侧存储数据的即时验证，保证了云只能诚实地保存完整数据而无法作恶；

2)提出了一种基于布谷鸟过滤器的端侧轻量级高效验证方案，显著提高了查找验证效率并降低了端侧的算力和存储空间负载；

3)利用了区块链进行完整性校验的审计和存证，保证了验证的可追溯和公平性，从而避免了潜在的纠纷和抵赖行为；

4)采用 Intel SGX 作为可信硬件实现了方案的原型系统并进行实验，实验结果表明，本文方案具有良好的实用性和可扩展性.

1 相关工作

在本文工作之前，已经有一些针对端云场景中数据存储的隐私性和完整性的研究工作.

首先，针对数据的隐私保护问题，Babitha 等人[5]针对公共云环境中的数据隐私和安全问题提出了一种解决方案，即采用 128 位高级加密标准 AES 对数据进行加密，以增强数据的安全性和保密性. 在数据上传至云端之前，使用 128 位 AES 加密算法对数据进行了加密处理，并将加密后的数据分散存储于云中，从而提高了数据的完整性. 为了防止未经授权的访问，引入了短信服务机制. Seth 等人[6]提出了一种混合加密协议，该协议部署了 Blowfish 和 Paillier 加密算法，并将它们与现有的混合 AES 和 RSA 技术进行了比较，还提出了分 2 个阶段实现安全数据存储协议的算法. 这种混合协议旨在通过减少计算时间和密文大小来提高云存储的能力. Sarkar 等人[7]提出了一种基于混合加密方案的新框架，该框架可以有效地对数据进行加密和检索. Morales-Sandoval 等人[8]提出了一种在完全基于属性加密的基础上构建的云中存储、共享和检索加密数据的安全方法，从而通过搜索访问控制实现对加密数据的访问控制机制，以及对信息检索任务的访问控制.

针对数据完整性校验，Yu 等人[9]提出了一种基于身份的远程数据完整性检查(identity-based remote data integrity checking, ID-RDIC)协议，旨在解决云存储中数据完整性验证的问题，同时确保数据的隐私性. 利用密钥同态加密原语来简化系统复杂性和 PKI 中公钥认证框架的建立与管理成本并且提出的 ID-RDIC 协议在 RDIC 过程中不会向第三方审计者泄露存储数据的任何信息. 新构造在通用群模型中被证明是安全的，并且对第三方审计者实现了零知识隐私. Ping 等人[10]提出了一种基于代数签名和椭圆曲线密码学的公共数据完整性验证方案，旨在解决云存储中数据外包服务的安全性问题. 该方案不仅允许第三方经过授权的第三方用户验证外包数据的完整性，还能够有效地抵御重放攻击、替换攻击和伪造攻击等恶意攻击. 除此之外该方案还采用了对称加密技术，以确保数据的隐私性. Zhu 等人[11]提出了一种基于 ZSS 签名的数据完整性验证方案，通过引入可信第三方的支持来实现隐私保护和公共审计. 该方案有效地降低了计算开销，通过减少签名过程中

哈希函数的开销,在假设 CDH 困难问题成立的情况下,该方案能够抵抗自适应选择消息攻击.该方案充分考虑了安全性、可扩展性和隐私保护,适用于物联网中大量聚合数据的数据分析应用;支持公共审计,采用随机掩码技术保护数据隐私,并基于 CDH 困难假设证明了其安全性.然而,与大多数现有云环境中的 BLS 签名方案相比,该方案在多副本环境下的数据完整性验证方面存在局限性.Garg 等人[12]提出了一种在云计算中进行数据完整性审计的有效方法.所提协议的目的是在审计协议的系统设置阶段最小化客户端的计算复杂性.基于双线性对的性质,所提出的协议是可公开验证的,并支持对数据的动态操作.所提出的协议的安全性依赖于随机 Oracle 模型中计算 Diffie-Hellman 问题的稳定性.

上述方案中,云或第三方通常需要访问一定的数据信息以执行验证操作,这可能会危及用户数据的机密性,并且用户必须默认第三方是可信的,这在实际情况中很难保证.因此,有必要探讨一种新的方法,即让云在不获取明文数据的情况下提供算力来完成验证计算.可信执行环境是一种有前景的解决方案.可信执行环境提供了一个安全的执行环境,保障敏感数据在执行过程中不被泄露给云服务提供商或其他第三方.利用可信执行环境,用户可以将验证计算的逻辑和代码放入可信执行环境中执行,从而实现对云端数据的完整性验证,而不泄露数据的明文.通过创造一个连云本身都无法访问的可信执行环境,将密文加载进去进行计算并产生计算过程的可信证明,既保证了数据的机密性,又防止云在计算过程中可能的作弊行为.

Fan 等人[3]提出了一种名为 SIBAS 的方案,该方案利用可信执行环境作为审计器,以检查本地侧外包数据的完整性.SIBAS 不仅能够验证外包数据的完整性,还能够通过 Shamir 的 (t, n) 阈值方案在可信执行环境中实现安全的密钥管理.在随机预言模型下,基于计算 Diffie-Hellman 假设,SIBAS 能够抵抗选择消息和目标身份的攻击者的攻击.SIBAS 方案不涉及 PKG 服务器和审计服务器,而是使用可信执行环境替代它们进行密钥生成和数据验证.Fan 等人[13]还提出了一种安全的重复数据消除方案,该方案利用可信执行环境进行密钥管理,并以此取代第三方服务器.具体而言,该方案限制权限不足的云用户通过工作量证明(PoW)协议来访问特权数据.在系统设置阶段,为每个用户分配一组权限.当用户试图将文件上传到云端时,需要在可信执行环境的协助下生成基于特权的身份验证码.根据验证码,云服务提供商确定用户是否有权执行重复检查或 PoW 操作;只有能够证明是拥有该文件的用户才能检索该文件.这种改变显著增强了对中间人攻击和野蛮力量猜测攻击的抵抗力.Kurnikov 等人[14]提出了一个名为 CKS 的系统,CKS 是一个在线服务,专门用于安全地生成、存储和使用个人加密密钥.该系统采用远程认证机制,以确保用户与合法的可信执行环境进行通信,并允许用户通过密码进行身份验证,以避免密码盗窃或网络钓鱼等安全风险.CKS 还支持基于策略的访问控制、密钥委托给其他 CKS 用户以及通过安全审计日志审计所有密钥使用情况.该系统模型涵盖了核心功能,如密钥生成、存储和使用,以及附加功能,如委托和审计.王惠峰等人[15]提出了一种基于文件属性和用户需求动态调整文件的审计方案,使得文件的审计需求和审计方案的执行强度高度匹配.

综上所述,尽管现有研究针对数据的完整性展开了大量研究,但据我们所知,缺乏一种面向轻量级设备的、能够兼顾数据隐私性和持续性完整性校验的机制.

本文提出了一种轻量级的支持多次验证的数据完整性校验机制,在保证数据云存储的全程隐私性的基础上,利用可信执行环境的支持,在不引入额外开销的情况下,支持对云存储数据的任意次即时验证,并且保证云无法通过伪造证明的方式通过验证,从而防止了数据丢失、被篡改等行为的发生.

2 背景知识

在本节中,我们将介绍一些与本文有关的相关概念.

2.1 可信执行环境

开发优先考虑数据机密性的应用程序给软件架构带来了许多挑战.即使在平台上运行的特权代码中存在一个很小的漏洞,也可能会无意中泄露敏感信息.为了解决这个问题,可信执行环境在中央处理器内提供了一个安全区域,确保加载到其中的代码以及数据的机密性和完整性.GlobalPlatform 给出了可信执行环境的定义:可信执行环境是一个与设备主操作系统并行但与之隔离的执行环境.它保护其资产免受一般软件攻击,可以使用多种技术实现,其安全级别也相应的不同,但大致需要满足 5 个安全性质:

1)安全启动.要求只有指定的代码段才能被加

载. 一旦检测到任何更改, 都将导致引导过程中断.

2)安全调度. 要求系统对可信执行环境的使用进行可行的资源分配调度, 以保证操作系统的正常响应.

3)跨环境通信. 要求可信执行环境能够通过接口与系统中的其他模块进行通信.

4)安全存储. 要求存储保证数据的保密性、完整性和新鲜度, 数据访问是经过授权的.

5)可信 I/O 路径. 要求确保可信执行环境与系统外设之间通信的真实性和部分机密性.

在本文中, 我们采用 Intel SGX(software guard extensions)作为所使用的可信执行环境实例. 它是一种由 Intel 公司推出的安全扩展指令集, 为用户级代码提供硬件级保护. 这使软件开发人员能够对敏感代码和数据的安全性进行控制. SGX 允许在一个被称为“enclave”的受保护地址空间内执行进程. enclave 通过保护进程免受特定形式的硬件攻击以及同一主机上的其他软件(包括操作系统)的攻击, 来保障进程的机密性和完整性. 总的来说, Intel SGX 提供了一种强大的安全机制, 允许开发者在不受信任的环境中处理敏感数据, 并保护关键代码免受恶意软件的攻击. 这种技术可以为本文的端云场景提供更高级别的安全性保障. 目前, 阿里云和腾讯云都提供支持 SGX 的云服务器实例, 比如阿里云的 g7t, c7t 等实例.

2.2 哈希函数

哈希函数是一个从消息空间到像空间的不可逆映射, 换句话说, 哈希函数可将任意长度的消息经过变换得到固定长度的像. 其过程可以表示为:

$$h = H(M),$$

其中, M是一个变长消息, H是哈希函数, h是定长的输出值.

哈希函数具有 3 个特性:

1)抗碰撞性. 如果一个哈希函数 H很难找到 2 个不同的输入a和b, 使得它们的哈希值相同, 即$a \neq b$但$H(a) = H(b)$, 那么该哈希函数就具有抗碰撞性.

2)抗第一原象性. 抗第一原象性指的是对于给定的哈希值 h, 在合理的时间内很难找到任何原始输入 x, 使得$H(x) = h$. 对于给定的h, 找到任意x使得$H(x) = h$是计算不可行的.

3)抗第二原象性. 抗第二原象性指的是对于一个固定的输入 x, 在合理的时间内很难找到一个不同的输入y, 使得 $H(x) = H(y)$. 对于固定的x和任意的$y \neq x$, 找到$H(x) = H(y)$是计算不可行的. 这保证了即使攻击者知道一个特定的输入和它的哈希值, 他们也不能找到另一个具有相同哈希值的不同输入.

2.3 布谷鸟过滤器

布谷鸟过滤器作为一种新的集合摘要数据结构, 是一种优化的哈希表. 该哈希表由一个桶的数组组成, 其中每个桶包含多个槽位(slot), 每个槽位只存储 1 个元素指纹.

布谷鸟过滤器的核心思想: 每个新插入的元素会有 2 个候选桶, 其候选桶索引分别由哈希函数$h_1(x)$和$h_2(x)$决定. 检查这 2 个候选桶是否存在空槽位, 如果存在, 则将该元素的指纹插入该空槽位中; 否则, 即这 2 个桶不存在空槽位, 则采用“踢出”的思想, 将原有元素指纹移出并让位给新元素指纹. 记原有元素指纹为受害者, 将受害者移动到其另一个候选桶中. 这个踢出和重定位的过程直到所有元素都被成功插入时才结束, 或者达到最大重定位次数, 此时最后被踢出的受害者指纹会被丢弃.

当查询一个元素时, 布谷鸟过滤器仅需通过哈希函数找到该元素的 2 个候选桶, 并检查其中是否存在与该元素指纹匹配的指纹即可.

总的来说, 布谷鸟过滤器是一种高效的数据结构, 用于快速查找元素是否在集合中, 并且在空间利用和删除支持方面具有较大优势, 非常适合于本文中算力受限的端侧设备的本地校验所需要的数据查找比对操作.

2.4 区块链和智能合约

区块链是一种去中心化的分布式数据库技术, 利用密码学和分布式共识确保数据的安全性和一致性, 有效解决了传统中心化系统中的信任问题. 其核心优势在于提升了数据安全性和可靠性, 显著降低了单点故障和数据篡改的风险, 推动了更为公正和透明的信息交换和价值传输方式. 区块链具备透明、安全、不可篡改和可追溯等显著特性.

在区块链的演进过程中, 区块链 2.0 阶段引入了可编程化的应用, 其中智能合约作为一种重要的可编程应用形式, 显得尤为突出. 智能合约是部署和执行于区块链网络中的自治程序, 其独特之处在于无需第三方介入即可确保交易的可靠性和安全性.

区块链的上述性质非常适合本文中完整性校验的审计和存证, 保证了验证的可追溯和公平性, 从而避免了潜在的纠纷和抵赖行为.

3 数据完整性验证方案

本节将详细介绍所提出的可支持多次的数据完

整性验证机制. 首先, 介绍方案中所涉及到的一些参数和符号名称. 其次, 介绍方案的系统架构以及参与者, 对方案进行整体性的概述. 最后, 详细阐述方案涉及到的技术细节, 包括数据和加解密流、数据存储和验证的可信执行环境侧算法设计以及端侧的本地验证算法设计.

3.1 参数设置

本文方案中涉及到的一些参数和符号名称如表 2 所示.

Table 2 Parameter Setting

表 2 参数设置

缩略词	英文全称	中文描述
msk	master secret key	主密钥, 对称密钥, 用于对传输数据进行加密
ssk	session secret key	会话密钥, 对称密钥, 用于对 msk 进行加密
SGX	software guard extensions	所使用的可信硬件
IAS	Intel attestation service	Intel 提供的远程任务服务, 用于验证 SGX 产出的结果是否正确
π	Proof of correct execution of TEE	TEE 正确执行的证明
σ_{tee}	a signature over the enclave code and output	enclave 执行代码和输出的签名
σ_{IAS}	a signature over σ_{tee} and validity of σ_{tee}	对于σ_{tee}和其在 IAS 上验证结果的签名
DO	data owner	数据所有者
CS	cloud server	云服务器
BC	blockchain	区块链

3.2 系统架构

本节将具体介绍方案的系统架构, 包括参与实体和大体的业务流程. 方案主要分为存储和验证 2 个阶段. 存储是指数据从产生到安全私密地发送给云端存储起来的过程, 而验证是指存储后对云端进行数据完整性校验的流程. 如图 1 和图 2 所示, 首先介绍 4 个参与实体:

1)数据所有者(DO)(端侧). 是端云场景中的轻量级端侧设备, 算力和存储能力受限, 具有将本地数据托管出去存储的业务需求.

2)云存储服务器(CS)(云侧). 是端云场景中的云侧服务器, 具有庞大的算力和存储能力, 具有接收端侧数据进行存储的能力和业务需求. 此外, 云服务器中还配备有支持可信执行环境的硬件设备.

3)可信执行环境. 尽管可信执行环境不是一个独立的个体, 但为了彰显其与云服务器之间的隔离性, 我们将可信执行环境从逻辑上与云服务器分离开进行阐述. 可信执行环境被用于执行数据完整性

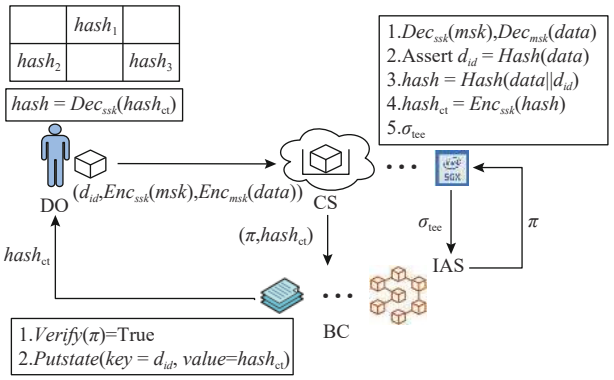

Fig. 1 Illustration of data storage process

图 1 数据存储过程示意图

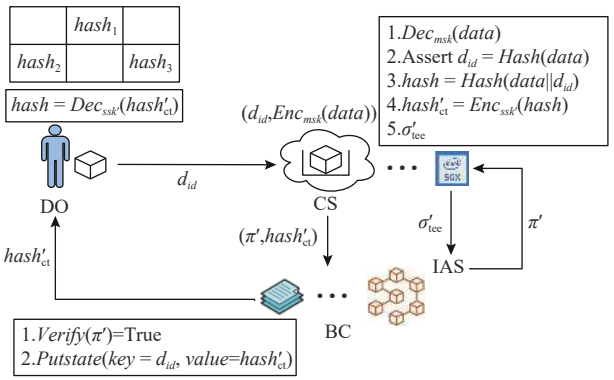

Fig. 2 Illustration of data verification process

图 2 数据验证过程示意图

验证的核心代码, 由于其机密性, 即使是云服务器(可信执行环境的外部主机)也无法访问到其开辟的隔离空间的代码和数据, 仅能得到完整性校验的执行结果.

4)区块链(BC). 区块链作为分布式数据库, 为可信执行环境产生的数据完整性校验结果提供去中心化存储信任根, 同时基于其支持的智能合约的可编程性, 为该结果进行自动校验、审计和仲裁, 从而避免双方潜在的交易纠纷.

下面介绍方案的业务流程. 在双方进行交互之前, 需要进行 3 项准备工作.

首先, 智能合约和可信执行环境的应用程序 enclave 需要被部署和启动起来(为了简略过程, 我们认为在代码的协商和开发过程中参与者已经完成). DO 与 CS 在智能合约上进行注册和存款. 同时, DO 在本地产生主密钥msk.

然后, 进入到存储阶段. 如图 1 所示, 首先, DO 与 enclave 进行远程认证, 并协商出会话密钥 ssk. 然后, DO 利用msk对数据进行加密, 并用ssk对msk进行加密, 然后一并发送给 CS. CS 将这 2 项数据加载进可信执行环境中, 可信执行环境利用ssk对msk进行

解密，并用解密出来的msk再对数据进行解密从而恢复数据. 然后，可信执行环境对该数据进行哈希，并使用ssk对哈希结果进行再加密以进行保护. 可信执行环境会对计算结果产生一个可信证明，与该加密哈希一同输出来，该证明可以被送到 Intel 第三方服务 IAS 上进行验证，以证明计算的正确性. 可信执行环境结果与证明也一并被发送到区块链进行存证. DO 可以从链上获取该加密的哈希，并用ssk进行解密，得到明文哈希，并存储在自己本地的布谷鸟过滤器中，以待后续的验证使用.

最后是数据完整性校验的阶段，与存储阶段的流程基本类似，核心原理是比较数据的哈希值前后是否一致，从而判断数据是否还完好地存储在云端. 如图 2 所示，DO 选择要验证的数据id发送给云，CS 选择对应的数据密文加载进 SGX 中，并利用msk进行解密，执行同样的哈希和加密操作，并发送出来，该哈希密文同样被发送到区块链存证. 用户在本地获取到密文后进行解密，得到明文哈希，并在布谷鸟过滤器中进行检索，看是否存在相同的哈希值. 有则证明验证通过，无则表示数据的完整性已经被破坏，云端的存储出现异常，合约会按照既定的逻辑对该异常进行处理.

3.3 数据和加解密流

本节将着重介绍具体的数据传输细节，包括每个阶段所传输的参数，以及数据加解密的处理. 我们根据不同的参与角色给出了存储和验证 2 个阶段的算法流程. 每个阶段都涉及到端侧和云侧各自的算法执行，我们对其分别进行描述. 首先是存储阶段. 算法 1 描述了云侧的可信执行环境数据完整性验证存储阶段的行为. 算法输入为被主密钥msk加密的数据密文ct_{data}以及主密钥被会话密钥ssk加密后的密文ct_{msk}. 输出为对数据做哈希后的哈希值密文ct_{hash}，以及对 SGX 正确执行产生的证明σ_{tee}. 输入和输出都是密文形式的是由于隐私保护的需求，同时哈希值密文可以防止云服务器得到哈希值明文，从而实现上文所说的仅保留哈希值而不保留数据明文，也能通过验证挑战的恶意行为.

算法 1. 可信执行环境数据完整性验证-存储阶段算法.

输入：密文数据ct_{data}, ct_{msk}，哈希函数$h()$；

输出：密文哈希值ct_{hash}，可信执行环境计算证明σ_{tee}.

① $Store(ct_{data}, ct_{msk}, hash_{id})$;

② $msk \leftarrow Dec(ssk, ct_{msk})$;

③ $data \leftarrow Dec(msk, ct_{data})$;

④ if $hash_{id} = h(data)$ then;

⑤ $hash_{data} \leftarrow h(data||hash_{id})$;

⑥ $ct_{hash} \leftarrow Enc(ssk, hash_{data})$;

⑦ $sealKey \leftarrow SGX.EGETKEY()$; /*enclave 获取封装密钥*/

⑧ $seal_{msk} \leftarrow SGX.SEAL(sealKey, msk)$; /*利用 SGX 的封装机制将 msk 存储起来 */

⑨ $\sigma_{\text{tee}} \leftarrow SGX.genAttestation(ct_{hash})$; /*enclave 产生对执行结果和代码的正确性证明*/

⑩ output $ct_{hash}, \sigma_{\text{tee}}$;

⑪ else

⑫ return failure

⑬ end if

⑭ return failure.

首先，算法先对所输入的密文值ct_{msk}进行解密，得到明文的msk，再用msk对数据密文ct_{data}解密得到明文数据$data$. 而后，开始计算$data$的哈希值$hash$，并用ssk对该哈希值进行加密，得到执行的输出ct_{hash}. 此外，算法利用了 SGX 的封装（sealing）机制，将用于加密数据的主密钥msk封存在执行主机本地，仅有该 enclave 才可以解密，这样做避免了再次进行主密钥的传输，防止了潜在的误操作或恶意行为. 最后，enclave 会产生这段执行的可信证明σ_{tee}，以保证执行的正确性. 该证明可以被用于执行远程认证，由 IAS 验证σ_{tee}的正确性.

当端侧设备获取到哈希值密文ct_{hash}后，需要进行解密再将明文哈希值存入本地的布谷鸟过滤器中. 使用布谷鸟过滤器有以下好处：端侧终端设备，如传感器、无人机等，算力往往较低，其主要算力需要用于接收外部数据上，因此需要尽量压缩本文方案在端侧设备上的计算开销，而布谷鸟过滤器则是一种高效的查找数据结构，可以快速查找本文中的哈希值是否在端侧本地存在，并且可以端侧减少空间利用. 算法 2 描述了端侧设备在存储阶段的算法流程. 我们沿用了文献 [16] 对布谷鸟过滤器的算法设计，将ct_{hash}解密后插入存储结构中. 该文献利用部分密钥布谷鸟哈希的技术，解决了当数据量达到一定规模后再插入新元素导致的位置重复问题，保证了数据插入时基于其指纹能够定位到新备用位置. 具体的算法描述和实现细节可以参考文献 [16].

算法 2. 端侧数据完整性验证-存储阶段算法.

输入：密文哈希值ct_{hash}；

输出：存储结果布尔值b.

① $Insert(ct_{hash}, ssk)$;
② $hash = Dec(ssk, ct_{hash})$;
③ $f = fingerprint(hash)$;
④ $i_1 = h(hash)$;
⑤ $i_2 = i_1 \oplus h(f)$;
⑥ if $bucket[i_1]$ or $bucket[i_2]$ has an empty entry then/*有空的条目*/
⑦ add f to that bucket; /*添加f到 bucket*/
⑧ return done;
⑨ end if
⑩ i = *randomly pick* i_1 or i_2;
⑪ for $n = 0, n < MaxNumKicks, n++$do
⑫ randomly select an entry e from $bucket[i]$; /*从$bucket[i]$中随机选择一个条目 e*/
⑬ swap f and the fingerprint stored in entry e; /*交换f和条目 e 中存储的指纹*/
⑭ $i = i \oplus h(f)$;
⑮ if $bucket[i]$ has an empty entry then /*有空的条目*/
⑯ add f to $bucket[i]$;
⑰ return done;
⑱ end if
⑲ end for
⑳ return failure.

下面是验证阶段,算法 3 则是可信执行环境数据完整性验证算法的验证阶段. 本质上来说,算法 3 与算法 1 的执行原理相同,都是计算数据哈希并加密输出,但仍然存在 2 项区别. 首先,验证阶段主密钥msk不再作为算法输入,需要利用 SGX 的封装机制,调出被封装的msk对数据密文进行解密. 其次,尽管仍然是用会话密钥ssk对哈希值进行解密,但由于每次会话不同, ssk也不同,因此使得ct_{hash}的值也不同,这保证了自私 CS 无法通过保存ct_{hash}的值通过验证挑战的可能. 因此,尽管方案的目的是确认存储和验证时哈希值相同,从而来保证完整性,然而云端得到的执行结果是每次都得到不同的哈希值密文,因此云不得不执行诚实的数据存储.

算法 3. 可信执行环境数据完整性验证-验证阶段算法.

输入: 密文数据ct_{data}, ct_{msk},哈希函数$h()$;

输出: 密文哈希值ct_{hash},可信执行环境计算证明σ_{tee}.

① $Verify(ct_{data}, hash_{id})$;
② $sealKey \leftarrow SGX.EGETKEY()$; /*enclave 获取封装密钥*/
③ $msk \leftarrow SGX.UNSEAL(sealKey, seal_{msk})$;
④ $data \leftarrow Dec(msk, ct_{data})$;
⑤ if $hash_{id} = h(data)$ then;
⑥ $hash_{data} \leftarrow h(data||hash_{id})$;
⑦ $ct_{hash} \leftarrow Enc(ssk, hash_{data}||hash_{id})$;
⑧ $\sigma_{\text{tee}} \leftarrow SGX.genAttestation(ct_{hash})$; /*SGX 产生对执行结果和代码的正确性证明*/
⑨ output $ct_{hash}, \sigma_{\text{tee}}$;
⑩ end if
⑪ else return failure
⑫ end if
⑬ return failure.

算法 4 描述了密文哈希值ct_{hash}在端侧布谷鸟过滤器被验证的过程,其核心原理是通过判断该哈希值是否在过滤器中能被检索到,从而判断该哈希值是否是源数据的哈希值,以确定数据的完整性是否被保证.

算法 4. 端侧数据完整性验证-验证阶段算法.

输入: 密文哈希值ct_{hash};

输出: 校验结果布尔值 b.

① $Lookup(ct_{hash})$;
② $hash = Dec(ssk, ct_{hash})$;
③ $f = fingerprint(hash)$;
③ $i_1 = h(hash)$;
④ $i_2 = i_1 \oplus h(f)$;
⑤ if $bucket[i_1]$ or $bucket[i_2]$ has f then
⑥ return true;
⑦ end if
⑧ return false.

此外,为了避免 SGX 容量不足的问题,我们采取流模式,将大体量数据进行拆分成一个个小块,并依次加载进 SGX 中执行完整性校验处理,从而避免了数据溢出等异常现象.

3.4 合约设计

尽管可信执行环境可以产生其有效范围内的正确结果,并提供相应的正确性证明,这是由其实现机制所保证的. 然而,在本文的场景中,涉及执行结果的带外传输和交易,如何保证该数据交易的正确性和公平性则是可信执行环境无法控制的. 因此,本文还额外引入了区块链,用于记录和存放可信执行环境的执行结果,以提供信任基石,保证端侧和云侧交易的正确性和公平性. 具体来说,我们利用区块链支持的智能合约的可编程性,将可信执行环境提供的

安全证明在链上进行自动校验，从而判断该可信执行环境执行结果的正确性. 在我们的方案中，SGX 正确执行的证明采用签名π的形式. 具体地说，SGX 执行计算得到π的过程如下：首先 SGX 执行计算，产生输出ct_{hash}和一个证明σ_{tee}（为对 enclave 代码和ct_{hash}的签名）. 验证者将σ_{tee}发送给 Intel 认证服务 IAS，由 IAS 验证σ_{tee}，并返回$\pi=(b,\sigma_{\text{tee}},\sigma_{\text{IAS}})$，其中$b\in\{0,1\}$表示对$\sigma_{\text{tee}}$有效性的验证结果，$\sigma_{\text{IAS}}$是 IAS 对 b 和σ_{tee}的签名. 然后将π提交给区块链智能合约，作为ct_{hash}的正确性证明. 由于π只是一个签名，智能合约既不需要可信的硬件，也不需要再次联系 IAS 来验证它. 该部分的实现在图 3 的“*uploadHash*”函数中，当验证过签名π的正确性后，合约才会接收ct_{hash}进行存证.

Contract Wrapper
1: On input (“*create*”, *participant*):
mapping(*uint*=> *string*) public hashStorage;
mapping(*address* => *uint*) public deposit;
Set *deposit*[*participant*] := 0.
2: On input (“deposit”,$deposit):
Assert *msg.sender* = *DO* or *CS*;
deposit [*msg.sender*] :+= *$deposit*.
3: On input (“*uploadHash*”, ct_{hash}, d_{id},$\pi=(b,\sigma_{\text{tee}},\sigma_{\text{IAS}})$):
Assert *msg.sender* = *CS*;
Assert the signature π;
Set *hashStorage*[d_{id}] := ct_{hash}.

Fig. 3 Smart contract wrapper

图 3 智能合约封装

此外，由于区块链自带的经济属性，可以利用其产生的代币进行端云交易的结算和管控手段. 具体来说，双方需要在合约上进行存款以作为押金. 一旦对π验证失败，则可以通过没收押金以进行惩罚，从而使得我们的方案满足经济安全性，有关存款的部分实现在图 3 的“*deposit*”函数中.

总而言之，区块链作为信任锚点，被用于进行数据存证、自动验证和实现奖惩，从而辅助可信执行环境保证执行结果的进一步的安全性.

4 安全模型和安全性分析

在本节中，首先介绍本文方案的安全模型，包括参与方以及所使用的安全技术的信任假设和潜在威胁. 接着，给出方案所旨在满足的安全目标，以及其正式的定义. 最后，给出方案的安全性证明.

4.1 安全模型

1）CS. 假设 CS 是半诚实模型，会遵守程序执行的一般流程，不会恶意破坏用户数据或者阻碍程序执行. 但其是自私的，会有意或者无意破坏数据的完整性，以达到扩大利益的目的. 具体来说，它会诚实地存储用户的初始化存储需求以及查询需求，不会恶意地篡改数据或者数据 *id*，但可能会通过给予一个任意的数据，作为完整性校验的输入进入 SGX 中以企图通过校验. 此外，云还是好奇的，会对用户加密数据产生兴趣，以企图获取明文.

2）端侧设备. 轻量级端侧设备具有较低的算力和存储能力，为了简化模型，我们认为端侧设备是诚实的，它们的目的仅为托管自己的海量数据并得到数据的完整性存储证明，不会恶意攻击云存储服务. 尽管算力不足，但假设它们仍然具有能力运行本文方案所需要的布谷鸟过滤器以进行校验和数据的加解密操作.

3）SGX. 假设 SGX 及其安全协议是安全的，攻击者无法破坏它们来伪造 SGX 的身份或与 SGX 交互的客户端的身份. 用于 SGX 通信的密钥是安全的，在有效时间内不会被破解. 此外，尽管 SGX 本身被认为是安全的，但它的主机是不可信的，并且有可能行为不当. 操作系统和网络堆栈都容易受到攻击. 在受控的 SGX 宿主上，攻击者可以任意地确定流程和消息流的执行. 攻击者可以随意创建和取消进程，延迟或拒绝传入消息，并伪造来自 SGX 的传出消息. 本文系统中的数据传输可以基于传输层安全（transport layer security, TLS）协议等安全通信协议来抵御中间人攻击. 通过将 SGX 远程认证与建立标准 TLS 连接相结合，可以将经过认证的 TLS 端点无缝绑定到 SGX 的 enclave，从而提高安全性.

4）区块链. 假设区块链架构是安全和值得信赖的，它将正确执行特定的计算并始终可用（即是活跃的）. 链上的数据不能被篡改. 矿工是理性的，不会偏离维护系统一致性的意图. 我们不考虑区块链层面的攻击.

5）布谷鸟过滤器. 我们在轻量级端侧应用布谷鸟过滤器以希望达到快速、高效的验证查询，以及降低端侧计算和存储开销的目的，因此假设布谷鸟过滤器算法本身不存在安全漏洞，查询是否在过滤器中的结果与数据完整性校验结果完全对应，即存在表明完整性校验通过，反之则不通过，不考虑误报率等实际应用中出现的问题.

4.2 安全目标

本文所提出的数据完整性验证方案旨在满足以下隐私性和完整性安全目标.

定义 1. 隐私性. 方案满足隐私性，当且仅当如果在选择明文攻击（chosen plaintext attack, CPA）安全性

下, 对于任何可以与方案任意交互的多项式时间对手 A, A 不能在协议执行期间从密文中获取有关明文的信息. 它要求攻击者不能从密文中泄露被封装的密钥. 对于安全参数 λ, 更形式化的表达为

$$\left| Pr\left[\begin{array}{l} b'=b\vee c'=c: \\ (ssk,(m_0,m_1),st)\leftarrow \mathcal{A}_{\text{query}}^{O_{ssk}}\left(1^{\lambda}\right); \\ \left(msk\left(m_0',m_1'\right),st\right)\leftarrow \mathcal{A}_{\text{query}}^{O_{msk}}\left(1^{\lambda}\right); \\ \{b,c\}\leftarrow\{\mathbb{S}\leftarrow 0,1\}; \\ CT_1^*\leftarrow \mathcal{SE}_1.Enc(ssk^*,m_b); \\ CT_2^*\leftarrow \mathcal{SE}_2.Enc\left(msk^*,m_c'\right); \end{array}\right] - \frac{1}{2}\right| \leqslant negl(\lambda).$$

定义 2. 完整性. 方案满足完整性, 当且仅当如果数据一旦出现不一致, 对于任何可以与方案任意交互的多项式时间对手 A, A 可以以忽略的概率通过方案的完整性验证. 更具体地说, A 需要通过 3 个环节的完整性校验: 1)可信执行环境生成的执行结果 ct_{hash}, 以及其执行证明 σ_{tee} 在 IAS 上的校验; 2)IAS 校验生成的证明 σ_{IAS} 在智能合约上的校验; 3) ct_{hash} 解密后在端侧的布谷鸟过滤器上的校验. 只有当这 3 项校验都通过时, 才能够说明完整性验证是成功的. 对于安全参数 λ, 更形式化的表达为

$$Pr\left[\begin{array}{l} \left(pk,pk_{\text{IAS}},\sigma_{\text{sgx}},data,outp,\sigma\right)\leftarrow \mathcal{A}^{\mathcal{F}_{\text{sgx}.cf}}\left(1^{\lambda}\right): \\ \left(\Sigma_{\text{sgx}}.Verify\left(pk,\sigma_{\text{sgx}},encl,outp\right)=1\right)\wedge \\ \left(\Sigma.Verify\left(pk_{\text{IAS}},\sigma_{\text{IAS}},\sigma_{\text{sgx}},b\right)=1\right)\wedge \\ C\mathcal{F}.Lookup(outp)=\text{True}\wedge \\ outp\neq \mathcal{F}_{\text{sgx}}.Resume(data) \end{array}\right] \leqslant negl(\lambda).$$

4.3 安全性证明

本节给出对 4.2 节提出的 2 个安全性质的完整证明, 旨在将方案的安全性规约到所用的安全技术的安全性上. 首先给出 2 个性质的定理, 然后给出其相应的证明.

定理 1. 隐私性. 假设包括 $\mathcal{F}_{\text{sgx}}$ 中的所有对称加密算法 $\mathcal{SE}$ 是 IND-CPA 安全的, 那么方案满足定义 1 下的隐私性.

证明. 假设存在一个概率多项式时间(*PPT*)的敌手 A 能够破解方案的 IND-CPA 安全性, 那么我们可以构建一个模拟器 B, 其能够破解对称加密算法 $\mathcal{SE}$ 的 IND-CPA 安全性.

1)查询阶段 1. 敌手 A 发起 2 个查询: *ssk*查询和 *msk*查询. 每当 A 发起这 2 个查询, 模拟器 B 将其传递给挑战者 C 并将 C 回答的结果返回给 A.

2)挑战. B 调用 A 来获取 2 个长度相同的元组 (m_0,m_1) 和 $\left(m_0',m_1'\right)$ 并发送给挑战者 C. C 生成挑战密文 CT_b^* 和 CT_c^*, 其中 $\{b,c\}\in\{0,1\}$ 且 CT_b^* 和 CT_c^* 分别是 m_b' 和 m_c' 的对称加密算法 $\mathcal{SE}$ 密文. 然后 C 将密文 CT_b^* 和 CT_c^* 发送给 B.

3)查询阶段 2. A 继续像查询阶段 1 那样进行查询.

4)猜测. B 将密文 CT_b^* 和 CT_c^* 发送给 A 并得到响应 b' 和 c'. B 输出 b' 和 c' 作为其猜测.

5)模拟结束.

下面我们分析模拟器 B 破解 $\mathcal{SE}$ 的 IND-CPA 安全性的概率的 2 种情况.

①如果 A 破解了 $\mathcal{SE}_1$ 的 IND-CPA 安全性, 也就是说 A 能够在上述游戏中获胜, 其获胜概率优势非常明显, 记为 ϵ. 那么我们有 $Pr[b'=b]=\frac{1}{2}+\epsilon$.

②如果 A 破解了 $\mathcal{SE}_2$ 的 IND-CPA 安全性, 也就是说 A 能够在上述游戏中获胜, 其获胜概率优势非常明显, 记为 ϵ. 那么我们有 $Pr[c'=c]=\frac{1}{2}+\epsilon$.

在这 2 种情况下, B 破解了 $\mathcal{SE}$ 的 IND-CPA 安全性. 证毕.

定理 2. 完整性. 假设 $\mathcal{F}_{\text{sgx}}$ 是安全的, 其远程认证协议是安全的, 签名算法在选择消息攻击下具有存在性不可伪造性(EU-CMA)且布谷鸟过滤器是安全的, 那么方案满足定义 2 下的完整性.

证明. 在定义 2 中我们给出了敌手 A 需要经历的 3 个验证, 只有当这 3 项校验都通过时才能够通过完整性验证. 下面我们分别对定义 2 中的 3 个环节进行讨论:

1)针对环节 1, SGX 提供了一种验证机制, 保证了 enclave 代码和执行结果可以被 SGX 认证. 如果 A 破解了 $\mathcal{F}_{\text{sgx}}$ 中的 Σ_{sgx}, 从而使得错误的 enclave 代码 *encl* 或者执行结果 *outp* 能够通过 SGX 验证, 那么这破坏了 $\mathcal{F}_{\text{sgx}}$ 的安全性.

2)针对环节 2, SGX 假设存在一个概率多项式时间(*PPT*)的敌手 A 能够破解所提出方案的 EU-CMA 安全性, 那么我们可以构建一个模拟器 B, 其能够破解签名算法 Σ 的 EU-CMA 安全性.

3)查询阶段 1. 敌手 A 发出签名 $Sig(m)$ 查询. 每当 A 对消息 m 发出签名查询时, 模拟器 B 将 m 传递给挑战者 C 并从挑战者 C 处获取签名 σ, 并把 σ 返回给敌手 A.

4)挑战. B 随机生成一条信息 m^*, 然后传递给 A. A 返回一个签名 σ^*, 其中 σ^* 是 m^* 的有效签名. 然后, B 将 (m^*,σ^*) 返回给挑战者 C 作为响应, 完成了 B 的模拟.

针对环节 3, 敌手 A 可以通过布谷鸟过滤器的查询验证, 即提供一个 *outp* 使其在过滤器中存在. 那么可以分为 2 种情况: 敌手 A 发送了正确的 *outp* 使其通过验证; 敌手 A 没有提供正确的 *outp* 也能通过验证.

① 敌手 A 可以提供正确的 *outp*, 这仍然分为 2 种可能, 一种是 *outp* 为正确的 *data* 经过 SGX 计算得到的输出结果, 但由于 A 无法得到正确的 *data* 且 $\mathcal{F}_{sgx}$ 是安全的, 因此该情况不存在. 因此, 此情况得证. 第 2 种是 *outp* 为不正确的 *data* 经过 SGX 计算得到正确的 *outp*, 但由于 $\mathcal{F}_{sgx}$ 是安全的, 因此该情况不成立.

② *outp* 不是正确的 *data*, 但经过 SGX 计算得到的输出结果 *outp* 仍然可以通过验证, 这意味着 *outp* 是其他 *data* 经过 SGX 计算的结果, 即, 用户查询的数据 *id* 为 u_{id}, 而敌手 A 给出了另一个 *id* 为 u'_{id} 的正确数据. 经过 SGX 计算得出的正确结果同样可以通过过滤器的验证, 但方案设计了数据的 *id* 是数据本身的哈希值, 由于哈希的性质, 其 *id* 与数据一一绑定. 并且由于 SGX 中会验证所输入的数据与其 *id* 的对应关系, 因此敌手 A 无法通过 SGX 的校验, 得不到计算结果 *outp*. 因此, 该情况不存在. 由于布谷鸟过滤器是安全的, 因此敌手 A 无法通过错误的 *outp* 通过过滤器验证. 因此, 此情况得证.

综上, 完整性得证. 证毕.

5 实验评估

5.1 实验环境

为了评估方案的可行性和实用性, 我们实现了方案的一个原型系统. 云端我们采用搭载了 Intel® Core™ i7-7700 CPU @ 3.60GHz 的服务器进行模拟. 该服务器配备了 16 GB 内存、4 个物理核心和 8 个逻辑核心, 并支持 SGX 1 运行于 Linux Ubuntu 20.04 操作系统. 我们基于 Intel SGX SDK (版本 2.21.100.1), 使用 C++编程语言进行了 SGX 模块的开发. 数据参与方使用的另一台配备 Intel® Core™ i7-7700 CPU @ 3.60 GHz 的计算机上同样运行于 Linux Ubuntu 20.04 操作系统. 方案中涉及到加解密算法为 128bit 密钥的 AES-GCM 加密算法, 哈希算法为 SHA-256 算法. 使用 Ethereum 和 Fabric Hyperledger v2.4.6 作为底层链, 并分别使用 Solidity 和 Golang 语言进行了智能合约开发. 数据参与方和 CS 采用非阻塞 TCP 协议进行通信, 使用 C++非阻塞 socket 实现.

5.2 计算复杂度对比

在本节中, 将本文方案与其他方案进行比较. 我们比较端侧和云侧在存储和验证 2 个阶段中的计算复杂度. 其中, n 是数据量大小, m 是数据条目数. 引入数据条目数这项指标是为了突出引入布谷鸟过滤器的优越性. 本文方案与其他方案的比较如表 3 所示. 由于不同的方案设计思路、执行过程以及侧重点都有所不同, 我们尽量保持每个方案处于一个可以对比的范围内进行比较, 因此将每个方案根据不同的执行者和执行阶段, 将其划分至本文方案所提出的 4 项复杂度分析中. 可以看到, 本文方案在端侧和云侧的存储和验证 2 个阶段中的计算复杂度均与数据量大小以及数据条目数无关. 由于引入了布谷鸟过滤器, 其插入和查找的时间复杂度均为常数时间复杂度 $O(1)$. 这意味着不论数据规模的大小, 布谷鸟过滤器的插入和查找操作所需的时间都是固定的. 因此相对于其他的方案, 本文方案在数据条目数较大的情况下, 具有非常明显的效率优势.

Table 3 Computation Complexity Comparison of Data Integrity Check Schemes

表 3 数据完整性校验方案计算复杂度对比

方案	端侧		云侧	
	存储阶段	验证阶段	存储阶段	验证阶段
本文方案	$O(1)$	$O(1)$	$O(1)$	$O(1)$
文献 [1] 方案	$O(\log m)$	$O(\log m)$	$O(1)$	$O(1)$
文献 [2] 方案	$O(m)$	$O(m)$	$O(n)$	$O(n)$
文献 [3] 方案	$O(m)$	$O(m)$	$O(1)$	$O(1)$
文献 [4] 方案	$O(m)$	$O(m)$	$O(1)$	$O(1)$

5.3 云侧时间开销

云端配备有可信执行环境以执行本文方案中存储和验证 2 个阶段的可信证明产生过程. 这其中主要包含了初始化阶段的远程认证过程和数据传入 SGX 后的计算过程, 而计算过程也分为存储和验证 2 个步骤. 我们在图 4 和图 5 中给出了不同数据量下这些步骤的平均时间开销. 我们固定每条数据的大小为 1 KB, 考虑不同数据条目数的情况下云侧执行的时间开销. 首先, 端侧和云侧进行远程认证的时间开销

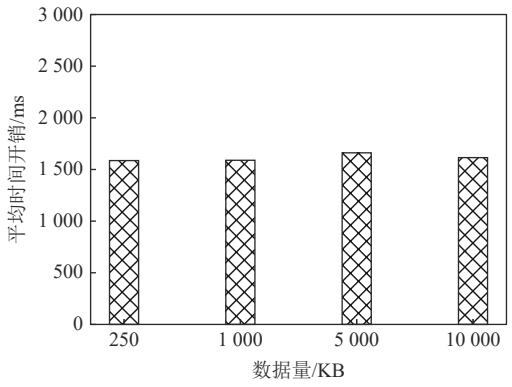

Fig. 4 Average time overhead of remote attestation

图 4 远程认证的平均时间开销

为 1.5 s 左右, 该操作与数据量无关. 其次, 分别展示了存储和验证阶段的 SGX 执行时间, 由于存储时第 1 次需要进行密钥的解码与封装操作, 而我们设置每条数据配有一个*msk*用于加解密, 因此存储阶段的时间开销较大, 但由于数据存储对每条数据来说仅需执行 1 次, 而验证需要多次进行, 因此, 单次验证平均不到 0.3 ms 的时间开销非常高效.

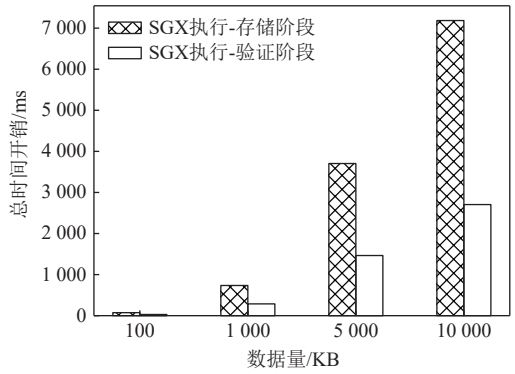

Fig. 5 Total time overhead of each stage at the cloud side

图 5 云侧各个阶段的总时间开销

此外, 我们考虑了由可信硬件带来的额外性能的影响, 因此测量了相同的数据完整性校验算法在 SGX 中执行和在普通环境中执行的时间开销, 如图 6 所示. 可以看到, 由 SGX 执行的时间开销普遍比一般环境中执行慢 3~4 倍, 这是由于 SGX 特殊的设计所导致, 引入了更多的安全操作, 以保证执行的私密性和安全性. 尽管如此, 两者的时间开销仍然处于同一毫秒数量级, 因此我们认为这样的额外开销是可以接受的, 不会过于影响执行效率.

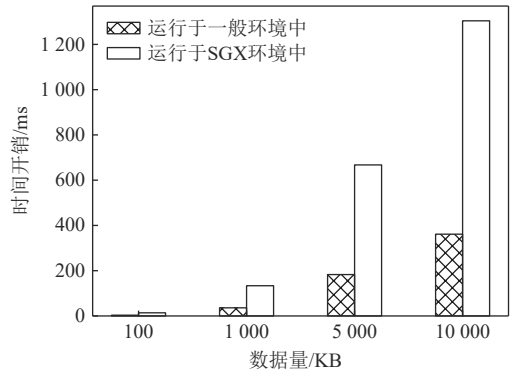

Fig. 6 Time overhead comparison of data integrity verification algorithms running on SGX and general environments

图 6 数据完整性校验算法运行于 SGX 与一般环境中的时间开销对比

5.4 端侧开销

我们考虑到了端侧设备往往可能是资源受限的轻量级移动终端设备, 比如传感器、IoT 设备、无人机等, 这些设备往往缺乏足够的算力和存储能力. 因此, 在端侧引入了一套基于布谷鸟过滤器的验证机制, 帮助端侧设备进一步释放计算和存储负载. 由于在端侧需要对云端可信执行环境产生的证明值进行解密和验证, 因此我们考虑了引入布谷鸟过滤器与不引入 2 种结构下, 在存储、校验阶段进行性能对比, 结果如图 7 和图 8 所示. 我们展示平均每条数据在 2 种结构下的时间开销. 在存储阶段布谷鸟过滤器带来更大的时间开销, 这是由于布谷鸟过滤器复杂的数据结构导致的. 然而, 由于布谷鸟过滤器常数级的查找复杂度, 在验证阶段显著地提高了检索效率, 当数据条目数为 10 000 条时, 时间开销提升了 67 倍左右, 并且当数据量再上升时, 差距更为显著. 由于对每条数据来说数据存储仅需执行 1 次, 而验证需要多次进行. 这表明数据验证上实现的高效可以带来端侧整体执行性能的提升.

除了时间开销之外, 我们还考虑空间开销, 如图 9 所示, 我们展示了不同数据条数下在 2 种结构下的空间开销. 我们需要预估整个方案需要执行的最大

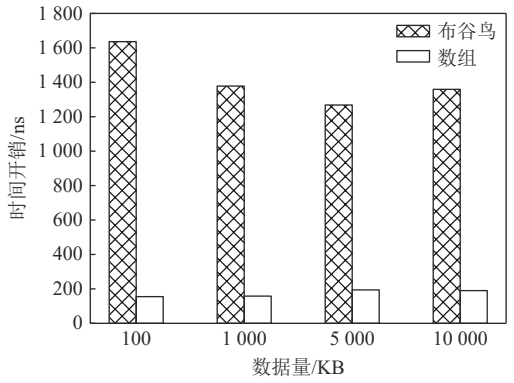

Fig. 7 Time overhead comparison of two different data structures at the client side in the storage phase

图 7 存储阶段端侧 2 种不同数据结构的时间开销对比

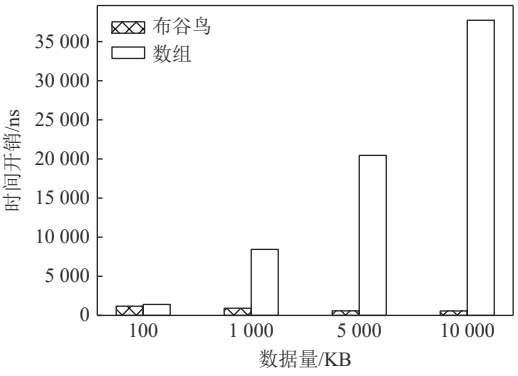

Fig. 8 Time overhead comparison of two different data structures at the client side in the verification phase

图 8 验证阶段端侧 2 种不同数据结构的时间开销对比

数据条数，然后根据最大数据条数为 2 种结构分配空间. 虽然布谷鸟过滤器是一种复杂的数据结构，可能会涉及中间结果、配置参数等空间开销，这种空间开销是很小的，因此我们忽略这部分额外的空间开销. 从实验结果来看，不引入布谷鸟过滤器就需要直接存储每条 256 bit 的数据，这个空间开销相比于布谷鸟过滤器来说大很多.

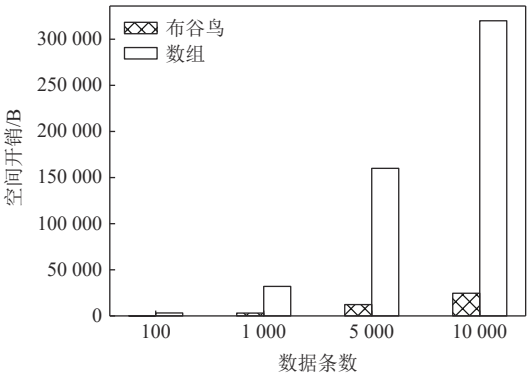

Fig. 9 Memory overhead comparison of two different data structures

图 9 2 种不同数据结构的空间开销对比

5.5 链上开销

区块链的作用是为了存储可信执行环境产生的证明值，并提供信任锚，起到可溯源可验证的作用. 然而，区块链较低的吞吐率可能会带来性能瓶颈. 因此，我们引入了 2 条不同的区块链 Ethereum 和 Fabric 来作为方案的区块链底层，以测试和评估区块链带来的性能影响. 这 2 种区块链都支持智能合约，从而可以实现用户自定义的合约内容开发，以满足各式各样的用户需求. 我们在这 2 个区块链上都实现并部署了图 3 所描述的智能合约，并测试本文方案在区块链上的开销. 由于以太坊是基于一种名为 gas 的统计机制来计算智能合约执行所花费的算力，从而结算成交易费，因此通过该指标可以明确地判断出该智能合约所需要的链上计算开销. 我们在本地搭建了以太坊私链，没有对底层代码进行任何修改，并部署了的智能合约，选择以太坊主网当前（2024 年 5 月 28 日）的均价 16 Gwei 作为交易费. 表 4 展现了该智能合约调用的执行情况，可以看到，除了创建合约的 gas 消耗较高，调用合约的交易开销处于一个较低的 gas 消耗程度.

接着是时间开销，我们同样地搭建了默认配置的 Fabric 私链，并部署了我们的合约. 在表 5 中展现了合约在这 2 条链上的执行结果. 可以看到，Fabric 的执行时间比 Ethereum 要低很多，这是由于 2 个区块链平台的默认区块时间导致的，只有交易被收录到区块上并被确认，交易才算真正上链，因此该时间与区块链底层平台的选择息息相关. 可以根据用户的需求选择相应的区块链底层，比如选择吞吐率更高的区块链以进一步提高效率.

Table 4 gas Cost and Transaction Fees of Different On-Chain Operations

表 4 不同链上操作的 gas 消耗和交易费

操作	gas 消耗	交易费/ETH
ContractCreation	715 945	0.011 455 12
Deposit	26 769	0.000 428 31
UploadHash	241 511	0.003 864 17

Table 5 Time Overhead of Different On-Chain Operations on Etheruem and Fabric

表 5 不同链上操作 Ethereum 和 Fabric 上的时间开销

操作	Ethereum 耗时/s	Fabric 耗时/s
ContractCreation	10.7	2.2
Deposit	7.6	2.1
UploadHash	11.4	2.6

5.6 总体方案评估

上述实验数据都是采用随机生成的数据，并且设定每条数据都是 1 KB. 为了增强数据的多样性，我们采用 MNIST 数据集进行实验，该数据集分为测试数据集和训练数据集，其中测试数据集包含 10 000 个图片样本数据，数据大小为 7 657 KB，我们分为 383 条 20 KB 和 192 条 40 KB 的数据进行实验；训练数据集包含 60 000 个图片样本数据，数据大小为 45 938 KB，被分为 383 条 120 KB 和 192 条 240 KB 的数据进行实验. 表 6 展示了 MNIST 数据集的时间开销情况，可也看到布谷鸟过滤器的插入和查询操作基本不受插入和查询的数据量大小的影响，可信执行环境执行时间会受到数据量大小的影响. 因此，相同的总数据量

Table 6 Time Overhead of Each Stage of Storage and Data Integrity Verification of MNIST Dataset

表 6 对 MNIST 数据集进行存储和数据完整性验证各个阶段的时间开销

不同阶段的时间开销/ns	测试数据集		训练数据集	
	20 KB/条	40 KB/条	120 KB/条	240 KB/条
存储阶段 TEE 执行平均时间	1 278 036	1 760 142	3 729 707	5 396 273
验证阶段 TEE 执行平均时间	830 041	1 312 147	3 281 712	4 948 268
存储阶段平均存储时间	1 207	1 109	1 152	1 064
验证阶段平均查询时间	920	1 002	993	1 029

被分成多条数据进行云存储,单条数据量越大,可信执行环境执行时间会更长,但是需要存储的次数会相对减少.单条数据量越小,可信执行环境执行时间会更短,但是需要存储的次数会相对增加.

文献 [4] 方案存储阶段的计算操作涉及到加解密、签名验签、摘要生成操作;验证阶段涉及到摘要验证操作;而且采用 ECC、聚合签名等操作.本文的存储和验证阶段的计算主要来自于存储阶段和验证阶段可信执行环境中的计算操作,由于可信执行环境技术的支持,我们采用了相对时间复杂度较低的 AES 加密算法和 SHA256 哈希.固定 1 KB 的数据大小对 2 个方案进行时间开销的对比,如表 7 所示,可以发现本文方案更加高效.

Table 7 Time Overhead of Different Schemes in the Storage and Data Integrity Verification Phases

表 7 不同方案在存储和数据完整性验证阶段的时间开销

方案	存储阶段/ms	验证阶段/ms
文献 [4]	15.065	0.320
本文方案	0.719	0.271

6 结 论

本文提出了一种支持即时验证的面向受限端侧设备数据外包存储场景下的数据完整性验证机制.通过引入可信执行环境,在保证数据隐私存储的前提下,保证了数据能够在可信执行环境中隔离地产生完整性校验证明,从而保证云存储服务器只能诚实地存储数据,无法进行作恶.此外,还引入了区块链和布谷鸟过滤器,分别为可信执行环境产生的证明提供可信存证以及提高端侧设备的验证效率.

未来的工作包括 3 个方面.首先,我们会继续研究无需依赖可信硬件的数据完整性校验方案,使其同样能够在隐私保护的前提下,支持多次即时验证.其次,会进一步探究区块链的性能问题,考虑降低区块链带来的端到端延迟.最后,会考虑当数据量增大增多时,端侧和云侧如何承载相应的计算量和存储量.

作者贡献声明:韩冰负责方案的实现和论文撰写;王昊和方敏提出了方案思路并撰写了部分论文;张永超和葛春鹏提出指导意见并修改论文;周璐指导修改论文.

参 考 文 献

[1] Li Jiaxing, Wu Jigang, Jiang Guiyuan, et al. Blockchain-based public auditing for big data in cloud storage[J/OL]. Information Processing & Management, 2020[2024-07-16]. https://doi.org/10.1016/j.ipm.2020.102382

[2] Garg N, Nehra A, Baza M, et al. Secure and efficient data integrity verification scheme for cloud data storage[C]//Proc of the 20th IEEE Consumer Communications & Networking Conf. Piscataway, NJ: IEEE, 2023: 1−6

[3] Fan Yongkai, Lin Xiaodong, Tan Gang, et al. One secure data integrity verification scheme for cloud storage[J]. Future Generation Computer Systems, 2019, 96: 376−385

[4] Zhao Quanyu, Chen Siyi, Liu Zheli, et al. Blockchain-based privacy-preserving remote data integrity checking scheme for IoT information systems[J]. Information Processing & Management, 2020, 57(6): 102355

[5] Babitha M, Babu K. Secure cloud storage using aes encryption[C]//Proc of the Int Conf on Automatic Control and Dynamic Optimization Techniques. Piscataway, NJ: IEEE, 2016: 859−864

[6] Seth B, Dalal S, Le D, et al. Secure cloud data storage system using hybrid Paillier–Blowfish algorithm[J/OL]. Computers, Materials & Continua, 2021[2024-07-16]. https://doi.org/10.32604/cmc.2021.014466

[7] Sarkar M, Kumar S. Ensuring data storage security in cloud computing based on hybrid encryption schemes[C]//Proc of the 4th Int Conf on Parallel Distributed and Grid Computing. Piscataway, NJ: IEEE, 2016: 320−325

[8] Morales-Sandoval M, Cabello M, Marin-Castro H, et al. Attribute-based encryption approach for storage, sharing and retrieval of encrypted data in the cloud[J]. IEEE Access, 2020(8): 170101−170116

[9] Yu Yong, Au M, Ateniese G, et al. Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage[J]. IEEE Transactions on Information Forensics and Security, 2016, 12(4): 767−778

[10] Ping Yuan, Zhan Yu, Lu Ke, et al. Public data integrity verification scheme for secure cloud storage[J]. Information, 2020, 11(9): 409. https://doi.org/10.3390/info11090409

[11] Zhu Hongliang, Yuan Ying, Chen Yuling, et al. A secure and efficient data integrity verification scheme for cloud-IoT based on short signature[J]. IEEE Access, 2019(7): 90036−90044

[12] Garg N, Bawa S, Kumar N. An efficient data integrity auditing protocol for cloud computing[J]. Future Generation Computer Systems, 2020(109): 306−316

[13] Fan Yongkai, Lin Xiaodong, Liang Wei, et al. A secure privacy

preserving deduplication scheme for cloud computing[J]. Future Generation Computer Systems, 2019(101): 127−135

[14] Kurnikov A, Paverd A, Mannan M, et al. Keys in the clouds: Auditable multi-device access to cryptographic credentials[C]//Proc of the 13th Int Conf on Availability, Reliability and Security. New York: ACM , 2018: 1−10

[15] Wang Huifeng, Li Zhanhuai, Zhang Xiao, et cl. A self-adaptive audit method of data integrity in the cloud storage[J]. Journal of Computer Research and Development, 2017, 54(1): 172−183 (in Chinese)
(王惠峰, 李战怀, 张晓, 等. 云存储中数据完整性自适应审计方法[J]. 计算机研究与发展, 2017, 54(1): 172−183)

[16] Fan Bin, Andersen D, Kaminsky M, et al. Cuckoo filter: Practically better than Bloom[C]//Proc of the 10th ACM Int on Conf on emerging Networking Experiments and Technologies. New York: ACM , 2014: 75−88

Han Bing, born in 2000. Master. Her main research interests include trusted execution environment and data security.
韩　冰, 2000 年生. 硕士. 主要研究方向为可信执行环境、数据安全.

Wang Hao, born in 1996. PhD. His main research interests include blockchain and privacy-preserving.
王　昊, 1996 年生. 博士. 主要研究方向为区块链、隐私保护.

Fang Min, born in 1996. PhD. Her main research interests include blockchain data management, trusted hardware, and privacy-preserving computing.
方　敏, 1996 年生. 博士. 主要研究方向为区块链数据管理、可信硬件、隐私保护计算.

Zhang Yongchao, born in 1994. PhD. His main research interests include network traffic measurement, graph stream analysis, and network security.
张永超, 1994 年生. 博士. 主要研究方向为网络流量测量、图流分析、网络安全.

Zhou Lu, born in 1990. PhD, professor. Her main research interests include blockchain, cryptographic and security solutions for the Internet of things.
周　璐, 1990 年生. 博士, 教授. 主要研究方向为区块链、密码学和物联网安全解决方案.

Ge Chunpeng, born in 1987. PhD, professor. His main research interests include information security and privacy-preserving for cloud computing, blockchain, and security and privacy of AI systems.
葛春鹏, 1987 年生. 博士, 教授. 主要研究方向为云计算中的信息安全和隐私保护、区块链、人工智能系统安全和隐私.