基于多分支路径树的云存储大数据完整性证明机制

谢四江[1,2] 贾 倍[1] 王 鹤[2] 许世聪[1]
(西安电子科技大学计算机学院 西安 710071)[1] (北京电子科技学院 北京 100070)[2]

摘 要 随着互联网和移动电子设备的不断普及,网络化存储将成为未来的主要存储方式,而目前的云存储方式也将会是网络化存储的必然趋势,因此如何确保云存储环境下用户数据的完整性成为人们关注的主要问题。针对该问题,提出了一种基于多分支路径树的云存储大数据完整性证明机制,通过引入第三方代理实现公开验证,加入随机掩码实现数据隐私,基于多分支路径树这一动态数据结构实现动态操作,并针对多分支路径树提出一种新的数据完整性检测算法。实验结果表明,所提方案可以高效地完成大量数据的更新,同时支持多用户的数据完整性验证。

关键词 云存储,数据完整性,数据持有性证明,批量审计,多分支路径树

中图法分类号 TP309.2 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.03.028

Cloud Big Data Integrity Verification Scheme Based on Multi-branch Tree

XIE Si-jiang[1,2] JIA Bei[1] WANG He[2] XU Shi-cong[1]
(College of Computer Science, Xidian University, Xi'an 710071, China)[1]
(Beijing Electronic Science and Technology Institution, Beijing 100070, China)[2]

Abstract With the popularization of the Internet and mobile electronic devices, network storage will become the main way of storage in the future, and the cloud storage also will be the inevitable trend of network storage. How to ensure the integrity of users' data on cloud storage environment becomes a major problem people concern. Aiming at the problem, this paper presented a cloud big data integrity verification scheme based on multi-branch tree. It realizes public verification with third party auditor and supports privacy-preserving by adding random masking, as well as, it uses a dynamic data structure which is multi-branch tree to accomplish dynamic operations. This paper also proposed a new algorithm to get information of data integrity verification from multi-branch tree. Test results show that the scheme can be efficiently applied in the cloud environment to verify data integrity with frequent update operations and multi-users.

Keywords Cloud storage, Data integrity, Provable data possession, Batch auditing, Multi-branch tree

1 引言

云计算是指以互联网相关服务的增加、使用和交付模式为基础,通过互联网来提供虚拟化资源,是云计算的主要应用之一。云存储是指通过网络提供虚拟化存储和相关数据的服务。然而,数据的集中存储增大了数据泄漏的风险,众多黑客对互联网运营商进行攻击来窃取用户数据,同时数据服务工作人员也可能存在不当操作,这些都极大程度地威胁着个人信息的安全。一系列的数据安全和隐私泄漏事件也极大地降低了人们对云存储安全的信任度,阻碍了云存储的大力推广与应用,因此确保数据安全是当前云存储所要解决的关键问题之一,数据完整性证明[1-2]是维护云端数据完整性的一种重要方式。

按照是否对需要验证的数据文件采用容错预处理,文献[3]将数据完整性证明机制分为数据可恢复性证明机制(Proofs of Retrievability, POR)[4]和数据持有性证明机制(Provable Data Possession, PDP)[1]。数据可恢复性证明机制可以识别并恢复被损坏的数据;而数据持有性证明机制可以更迅速地识别出其是否被损坏,但不能恢复数据。两种证明机制由于侧重点不同,因此适用于不同的应用需求。POR机制主要被用于重要数据的完整性保证[3];而PDP机制由于在识别损坏数据时具有快速性,多被用于数据文件的完整性检查。

到稿日期:2018-02-08 返修日期:2018-06-14 本文受国家自然科学基金:基于内容的图像光影模板学习与美学质量评价关键技术研究(61402021)资助。

谢四江(1971-),男,硕士,教授,主要研究方向为密码工程、信息安全;贾 倍(1993-),女,硕士,主要研究方向为大数据、云存储安全,E-mail:jiabeibeijia@163.com(通信作者);王 鹤(1992-),女,硕士,主要研究方向为数字签名、可信计算;许世聪(1994-),男,硕士,主要研究方向为隐私保护、机器学习。

PDP 和 POR 机制被提出后，得到了国内外学者的广泛研究。文献[5]针对文献[1]中的不支持动态操作进行改进，但是改进后的方案只支持修改和删除操作，并不能实现数据插入操作。Wang 等[6]设计实现了第一种面向云存储的、支持部分动态操作的 POR 机制。Wang 等[7-8]提出了利用 MHT 的数据完整性证明机制，该机制虽然支持动态操作并且不受证明次数的限制，但是通信和计算开销并没有降低。Erway 等[9]引入认证跳表这一数据结构来实现对数据的全动态操作，但是增加了整体机制的通信开销，影响了整体性能。他们[10]对认证跳表的结构也进行了改进，一定程度上降低了机制的通信和计算开销，提升了机制的整体性能。文献[11-12]以 BLS 签名技术为基础，引入 RSA 结构，实现了在云存储服务中提供隐私保护的公共审计方案，数据拥有者将数据存储到云服务器后即可删除本地的原数据，TPA 仍可以执行完整性审计工作，实现隐私保护。文献[13]利用对称双线性对，提出了一种高效的 PDP 方案，但是该方案易受 3 种攻击。

很多研究者针对现有机制进行改进，添加适用于更多应用场景的新特性[14-21]。文献[14]提出一种新型的 PDP 机制，在 PDP 机制中融入访问控制机制，该方案不仅支持数据完整性验证，还允许数据拥有者撤销其他用户的访问权限。文献[15]提出一种针对跨云服务器存储的完整性证明协议 ID-DPDP，基于身份授权，该协议可以实现私有验证、委托验证和公共验证。文献[16]提出一种安全的云存储系统，该系统支持隐私保护的完整性证明机制，使 TPA 能够同时有效地为多个用户执行审计。文献[17]分析出文献[8]的方案在哈希聚合过程中存在的安全问题，并在 Wang 的方案的基础上进行改进，优化了 MHT 数据结构，解决了安全问题。文献[18]提出一种支持多副本的数据完整性证明机制 FMR-PDP，用向量点积代理取代幂运算，满足了用户可以验证任意数量副本的需求，方案灵活有效。文献[19]提出一种基于身份的支持多副本的数据完整性证明机制 IDPMR-PDP，以提供第三方代理验证多副本数据而不使用公钥机制(PKI)，并针对该方案提出新的安全模型。文献[20]提出一种支持多云环境的基于身份的数据完整性证明机制 ID-PDP，该机制采用身份加密技术和散列消息认证码减少计算代价，可以在分布式环境下使用。文献[21]提出一种支持多云代理的数据完整性证明机制 PDPMT，该方案采用公钥数字签名技术，将多个数据块组织为一个集群并设计一种新颖的签名算法，可以显著降低计算成本。虽然国内外学者在数十年的探索和研究中已提出了许多较好的数据完整性证明机制，但现有机制不仅在处理规模及效率等方面存在着若干缺陷，在面对当今超大规模数据时也面临着严峻的挑战。

2　预备知识

2.1　云存储数据完整性证明特性

与传统的数据完整性证明不同，云存储环境要求数据完整性证明机制拥有新的特性：公开验证、支持数据隐私、支持动态操作、无备份验证和无状态验证。公开验证是指在数据完整性证明机制中，用户可以对自己的文件执行数据完整性证明，也可以将数据完整性证明的工作委托给信任的第三方代理来完成。引入第三方代理加大了数据泄漏的风险，因此用户在采用公开验证的同时，也应该保证与第三方代理间的通信过程不会泄漏相关文件信息。支持动态操作是指服务器可以根据用户要求更新数据文件，包括数据更新、插入和删除操作，并返回执行更新操作的证据。无备份验证是指在进行数据完整性证明时用户在本地不需要存储数据副本。无状态验证是指验证过程中无需保存任何验证状态，保证用户的每一次请求之间相互独立，互不干扰，用户不会记录每一次数据完整性证明的结果。

2.2　系统模型

云存储中数据完整性证明的系统模型如图 1 所示，主要包含 3 个实体：用户、云服务器提供商和第三方代理。

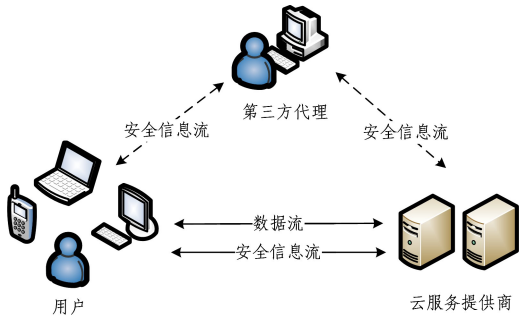


图 1　云存储中数据完整性证明的系统模型

Fig. 1　Model of data integrity verification on cloud storage

用户拥有数据文件，需要将其上传并保存到云存储服务器。云服务器提供商提供、管理并维护云服务器，能够提供存储资源。第三方代理具有更专业的验证知识和更强大的计算能力，能够更准确地完成用户数据的完整性验证任务。在数据完整性证明过程中，用户将数据文件上传到云服务器，云服务器接收到数据文件后将其存储到合适位置。当用户发起数据完整性证明任务时，将验证任务的权限赋给专业的第三方代理，由第三方代理向云存储服务器发送挑战请求；云存储服务器接收到该挑战信息后，计算一个证明文件未被破坏的证据返回给第三方代理；第三方代理对该证据进行验证，并将最后的验证结果发送给用户。云存储中的数据完整性证明机制的主要思想是在不需要下载所有数据文件的情况下，利用标签来验证数据文件的完整性。该思想主要是利用文件分块、同态标签和聚合签名等技术来实现，在资源和通信带宽受限的情况下可以很好地发挥作用。

2.3　安全模型

在第三方代理完全可信的情况下，主要分析不可信的云服务器的安全性。云服务器可能导致的 3 种攻击方式分别为伪造攻击、重放攻击和删除攻击。

1)伪造攻击

伪造攻击是指在云存储数据完整性证明过程中，可能会存在恶意的云服务提供商为了自己的利益或者其他原因伪造用于数据完整性证明的相关标签，并使用伪造的标签欺骗用

户来通过数据完整性证明。

2)重放攻击

重放攻击是指攻击者发送一个目的主机已接收过的包,来达到欺骗系统的目的,主要用于在身份认证过程中破坏认证的正确性。

3)删除攻击

删除攻击是指用户存储在云服务器上的部分数据被损坏或丢失时,云存储服务器的代理提供商使用预先计算的聚合"数据块-标签"来欺骗验证者,以通过完整性证明。

2.4 完整性证明过程

数据完整性证明机制主要由 4 个多项式时间算法组成,分别为密钥生成算法 KeyGen、数据块标签生成算法 SigGen、证据生成算法 GenProof、证据检测算法 VerifyProof。另外,为了实现数据完整性证明的动态操作,引入两个动态操作算法:更新执行算法 ExecUpdate 和更新验证算法 VerifyUpdate。

执行过程分为初始化阶段 Setup 和证明阶段 Audit。1)初始化阶段 Setup:用户对数据文件进行预处理。首先,用户执行密钥生成算法 KeyGen 产生公私钥对,并执行数据块标签生成算法 SigGen 生成标签信息,并将其发送至服务器端。2)证明阶段 Audit:验证者向服务器发起数据完整性验证的挑战,将生成的挑战信息发送给服务器。服务器接收到挑战信息后,根据保存的用户数据运行证据生成算法 GenProof,生成完整性证据返回给验证者。验证者运行证据检测算法 VerifyProof,输入接收到的证据,输出验证结果。

3 具体方案

3.1 MBT 结构

3.1.1 基本介绍

多分支路径树(Multi-branch Tree,MBT)的出度大于或等于 2,即每个节点拥有多个子节点,采用多分支路径结构来确保数据块在位置上的正确性,同时又弥补了二叉树结构所引起的树的深度随数据块数量的增长呈线性增长的不足。本节主要针对该问题引入多分支路径树,随着 MBT 出度的增大,树的深度将呈指数级下降,缩短了数据结构的构造时间,减少了验证过程中所需要的辅助信息,减轻了验证过程中的计算和通信负担,提升了整体效率。另外,在支持动态操作方面,MBT 支持用户在一个位置插入、删除和更新 $n-1$ 个数据块,更具有实际意义。将 MBT 引入到数据完整性验证方案中,树中的每一个叶子节点对应于一个数据块的哈希值 $h(H(m_i))$,非叶子节点是其所有子节点的哈希值的链接形式。如图 2 所示,中间节点 A_1 的值为 $h(A_1)=h(h(H(m_1))\|h(H(m_2))\|\cdots\|h(H(m_n)))$,根节点 R 的值为 $h(R)=h(h(A_1)\|h(A_2)\|\cdots\|h(A_n))$。但是,多分支路径树中的每个节点不仅需要存储与该节点相关的哈希值,还需要存储一个秩值,表示从该节点出发向下搜索可以访问到的叶子节点的个数。如图 3 所示,每一个节点上的值即为该节点的秩值,表示从该节点可访问到的叶子节点数目。

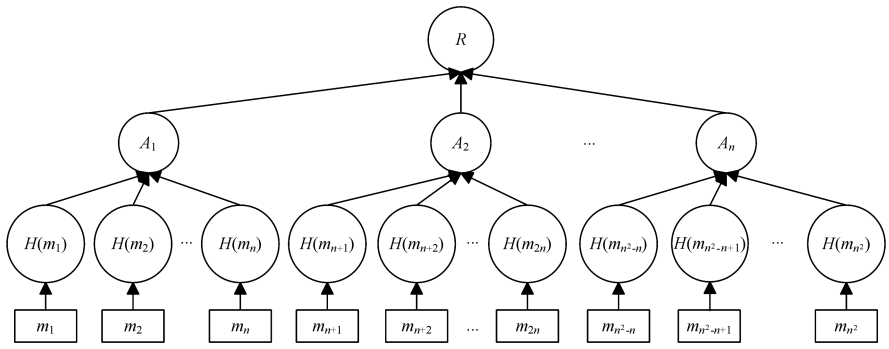


图 2 MBT 结构

Fig. 2 Construction of MBT

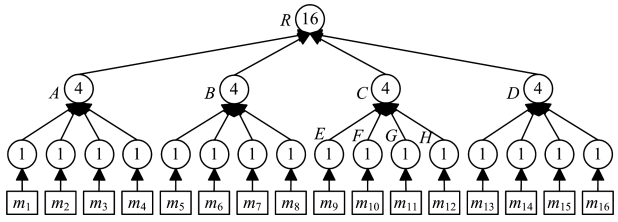


图 3 MBT 完整性检测算法示例图

Fig. 3 Example diagram of MBT integrity detection algorithm

为了验证第三方代理发送的挑战信息中的数据块被保存的位置的正确性,云服务器生成对应的证据信息,即生成该被挑战的数据块 m_i 的认证路径和辅助信息,用于在后续操作中计算 MBT 的根节点值。在 MBT 中,认证路径是指从目标节点到根节点路径中的节点集合,辅助信息则指该认证路径上的所有节点的兄弟节点集合。如图 2 所示,服务器为数据块 m_2 构造的认证路径为 $\{H(m_2),A_2\}$,则验证该数据块的辅助信息为:$\Omega_2=\{H(m_1),H(m_3),\cdots,H(m_n),A_2,A_3,\cdots,A_n\}$。

根据挑战信息中包含的数据块所对应的叶子节点及相应的辅助信息,可计算出 MBT 树的根节点值,如在图 2 中利用 $\{H(m_2),\Omega_2\}$ 即可计算出根节点值。

3.1.2 MBT 完整性检测算法

在 MBT 中需要计算对应的树根 R,并需要根据验证的叶子节点获得对应的遍历路径和认证路径。现有的数据完整性信息算法主要适用于二叉树结构,不适用于类似 MBT 的多分支路径树。本文提出一种新的数据完整性检测算法,其适

用于多分支路径树。

根据每一个节点的秩，可以确定从该节点出发的叶子节点的访问范围，从而确定该节点的孩子节点和右兄弟节点的访问范围。假设一个节点为 v，若该节点的右兄弟节点为 z，第一个孩子节点为 w，$low(v)$ 为从节点 v 向下搜索可以访问到的最左端的叶子节点，$high(v)$ 为从节点 v 向下搜索可以访问到的最右端的叶子节点，$r(v)$ 为节点 v 的秩，则可根据节点 v 的访问范围 $[low(v),high(v)]$ 来确定节点 v 的右兄弟节点 z 的访问范围 $[low(z),high(z)]$ 和节点 v 的第一个孩子节点 w 的访问范围 $[low(w),high(w)]$。多分支路径树中的根节点为 $root$，设置其秩值 $r(root)$ 等于树中的叶子节点数目，$low(root)=1$，$high(root)=n$，叶子节点的秩值为 1。根据节点 v 的访问范围 $[low(v),high(v)]$ 来计算它的右兄弟节点 z 的访问范围 $[low(z),high(z)]$ 和它的第一个孩子节点 w 的访问范围 $[low(w),high(w)]$，具体计算公式为：

$$low(w)=low(v) \tag{1}$$

$$high(w)=low(w)+r(w)-1 \tag{2}$$

$$low(z)=low(v)+1 \tag{3}$$

$$high(z)=low(z)+r(z)-1 \tag{4}$$

在搜索目标节点的过程中，设置一个认证栈，用于保存目标节点的辅助信息中的节点。若要访问叶子节点 i，则具体搜索方法为：若 $i\in[low(w),high(w)]$，则跟随向下的路径并将节点 w 的所有右兄弟节点压入认证栈，否则向右检索并将节点 w 压入认证栈，找到下一节点继续判断 i 的值与该节点的访问范围，直到检索到目标节点并将目标节点的所有右兄弟节点压入认证栈。

然后根据认证栈求解节点的辅助信息。为了根据认证栈得出目的节点的辅助信息，引入过渡栈和结果队列，结果队列存储目的节点的最终辅助信息。

1）过渡栈为空

①若认证栈不为空，则弹出认证栈中的栈顶节点并将其压入过渡栈中。

②若认证栈为空，则直接输出结果队列，即目的节点的认证路径。

2）过渡栈不为空

①若认证栈不为空，则判断认证栈中的栈顶节点与过渡栈中的栈顶节点是否为同一父节点（即两个节点为兄弟节点）。若为同一父节点，则弹出认证栈中的栈顶节点并将其压入过渡栈中；若不为同一父节点，则将过渡栈内的所有节点依次弹出并将其压入结果队列中。

②若认证栈为空，则将过渡栈内的所有节点依次弹出并将其压入结果队列中。

重复上述步骤，直到过渡栈和认证栈都为空，输出结果队列，即目的节点的辅助信息。

3.2　基于 MBT 的数据完整性证明机制

3.2.1　基本操作算法

基于 MBT 的数据完整性证明机制分为初始化阶段 Setup 和证明阶段 Audit。假设文件 F 被分为 n 个数据块 m_1，m_2，…，m_n（$m_i\in Z_p$，p 是一个大素数），$e:G\times G\to G_T$ 是一个双线性映射并且 g 为 G 的生成元，h 是一个同态加密函数，则认证执行过程如下。

1）Setup 阶段

用户采用 KeyGen 算法生成公钥和私钥，用 SigGen 算法对数据文件 F 进行预处理并产生同态认证的元数据。

在 KeyGen 算法中，用户生成随机签名密钥对 (spk,ssk)，选择随机参数 $\alpha\leftarrow Z_p$，计算 $v\leftarrow g^{\alpha}$，则私钥为 $sk=(\alpha,ssk)$，公钥为 $pk=(v,spk)$。

在 SigGen 算法中，给定 $F=(m_1,m_2,\cdots,m_n)$，用户随机选择参数 $u\leftarrow G$。计算 $t=name\parallel n\parallel u\parallel SSig_{ssk}(name\parallel n\parallel u)$，并将其作为 F 的文件标签。用户为每一个文件块 m_i（$i=1,2,\cdots,n$）计算标签 $\sigma_i\leftarrow(H(m_i)\cdot u^{m_i})^{\alpha}$，所有 σ_i 的集合用 Φ 表示，即 $\Phi=\{\sigma_i\}(i=1,2,\cdots,n)$。用户构造 MBT 并生成根 R，MBT 的叶子节点为文件块的哈希值的有序集合 $\{H(m_i)\}$（$i=1,2,\cdots,n$）。用户使用私钥 α 加密根 R，得到 $sig_{sk}(H(R))\leftarrow(H(R))^{\alpha}$。

用户发送信息 $\{F,t,\Phi,sig_{sk}(H(R))\}$ 给服务器，并从本地存储中删除 $\{F,\Phi,sig_{sk}(H(R))\}$。

2）Audit 阶段

Audit 执行过程如图 4 所示。用户或者第三方代理可以通过向云存储服务器发送挑战请求来验证数据的完整性。在挑战前，第三方代理 TPA 首先使用 spk 来验证标签 t，若验证失败，则输出 False，否则恢复 u。为了生成挑战信息，TPA 选择从集合 $[1,n]$ 中选择 c 个元素的随机子集 $I=\{S_1,S_2,\cdots,S_C\}$。对于每一个 $i\in I$，TPA 选择一个随机参数 $\nu_i\leftarrow B\in Z_p$。TPA 发送挑战信息 $chal=\{(i,\nu_i)\}$ 给服务器。

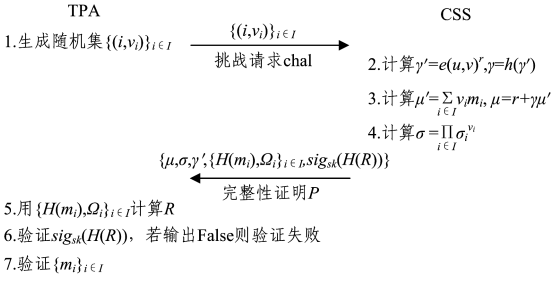


图 4　Audit 的执行过程

Fig. 4　Execution process of Audit

云服务器收到挑战信息 $chal$ 后，运行 GenProof 算法生成证明数据完整的证据。服务器选择随机元素 $r\leftarrow Z_p$，计算 $\gamma'=e(u,v)^r$ 和 $\gamma=h(\gamma')$，计算 μ' 来表示 $chal$ 中信息的线性组合 $\mu'=\sum_{i\in I}\nu_i m_i$。这里使用随机掩码技术来掩饰 μ' 得到 $\mu=r+\gamma\mu'$，并利用聚合标签计算 $\sigma=\prod_{i\in I}\sigma_i^{\nu_i}$。服务器发送 $proof=\{\mu,\sigma,\gamma',\{H(m_i),\Omega_i\}_{i\in I},sig_{sk}(H(R))\}$ 给 TPA 作为响应信息。

TPA 收到响应信息 $Proof$ 后，执行 VerifyProof 算法验证响应信息。TPA 首先用 $\{H(m_i),\Omega_i\}_{i\in I}$ 计算 MHT 的根 R，验证 $e(sig_{sk}(H(R)),g)$ 是否等于 $e(H(R),g^{\alpha})$，若相等则继续进行验证，否则输出 False。验证成功后，TPA 计算 $\gamma=h(\gamma')$ 并验证 $\gamma' e(\sigma^{\gamma},g)$ 是否等于 $e((\prod_{i\in I}H(m_i)^{\nu_i})^{\gamma}\cdot u^{\mu},v)$。

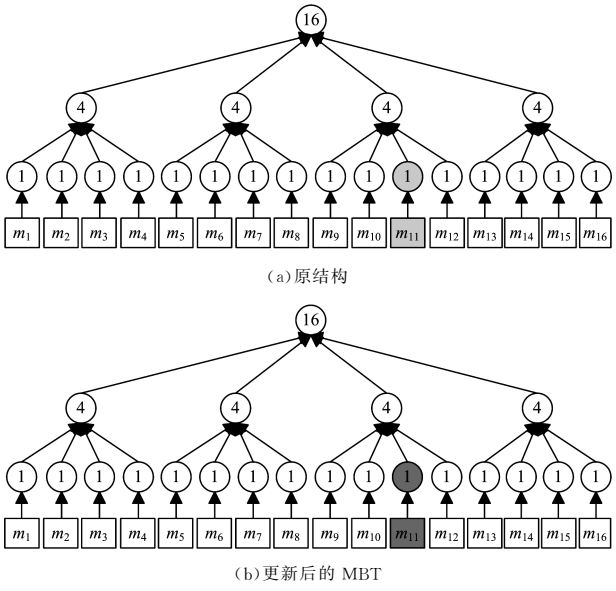
若相等则验证成功，否则输出 False，表示验证失败。整个验证过程分为两次，第一次验证数据块的位置是否正确，第二次验证数据块的内容是否正确。

3.2.2 动态操作

1)数据修改

数据的动态操作主要包括数据的修改、添加和删除。在 MBT 中，数据的修改是指用新的数据块替换原来的指定的数据块。假设用户想要更新第 i 个数据块，则需要用新的数据块 m_i' 替换原来的数据块 m_i。用户生成新的数据块所对应的响应标签 $\sigma_i'=(H(m_i')\cdot u^{m_i'})^{\alpha}$，构造更新请求 $update=(M, i, m_i', \sigma_i')$ 并发送 $update$ 到云服务器，其中 M 表示数据更新操作。云服务器接收到更新请求后，执行 ExecUpdate 算法。服务器用新的数据块 m_i' 替换原来的数据块 m_i 并输出新的文件 F'，用 σ_i' 替换 σ_i 输出 Φ，在 MBT 中用 $H(m_i')$ 替换 $H(m_i)$ 并计算新的 MBT 的根 R'，生成更新响应 $P_{update}=\{\Omega_i, H(m_i), sig_{sk}(H(R)), R'\}$ 并发送给第三方代理。第三方代理收到更新响应信息 P_{update} 后，执行 VerifyUpdate 算法来验证服务器是否正确更新。第三方代理用 $\{\Omega_i, H(m_i)\}$ 生成原来的根 R 并验证 $e(sig_{sk}(H(R)), g)$ 是否等于 $e(H(R), g^{\alpha})$，若相等则继续进行验证，否则输出 False。验证成功后，继续验证服务器是否正确执行数据修改操作，用 $\{\Omega_i, H(m_i')\}$ 生成根 R'' 并与返回的 R' 进行比较，若相等，则说明服务器正确执行了数据修改，用户可删除本地存储的数据文件，否则验证失败。

修改单个数据块时，MBT 的更新过程如图 5 所示，即直接找到需要更新的数据块，用新的数据块替换该数据块，并更新该数据块的遍历路径节点上的节点哈希值。



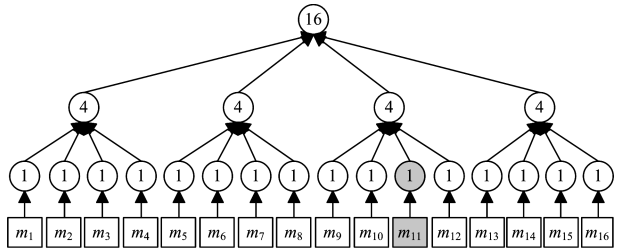
(a)原结构

(b)更新后的 MBT

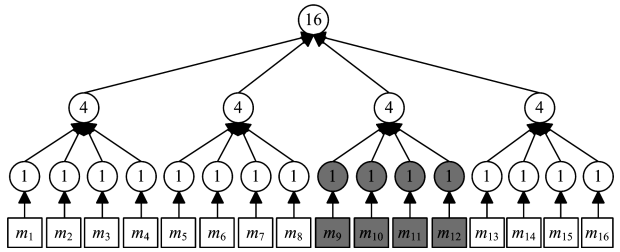
图 5 修改单个数据块时 MBT 的更新过程

Fig. 5 Update process of MBT when a single data block is updated

MBT 在进行数据修改时可以同时修改连续的多个数据块。在 MBT 中，由于每一个非叶子节点可以有多个孩子节点，每个叶子节点都有多个兄弟节点，因此可以一次修改多个连续的数据块，同时也只需要修改一个中间节点的哈希值。修改多个数据块时，MBT 的更新过程如图 6 所示，即直接找到所有需要更新的数据块，用新的数据块替换该数据块，并更新该数据块的公共父节点的遍历路径节点上的节点哈希值以及秩值。



(a)原结构



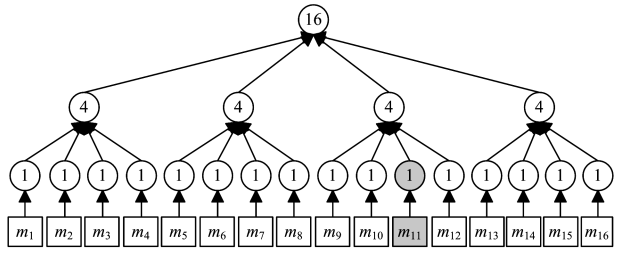
(b)更新后的 MBT

图 6 修改多个数据块时 MBT 的更新过程

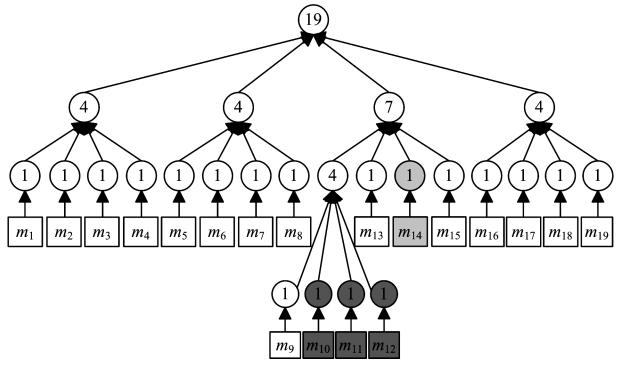
Fig. 6 Update process of MBT when multi-data blocks are updated

2)数据插入

在 MBT 中，数据插入操作会改变数据文件的逻辑结构。当一个数据块插入时，MBT 直接找到需要插入的数据块，并基于该数据块与新的数据块生成一个新的中间节点，用新的中间节点替换该数据块，并更新该数据块和新数据块的遍历路径节点上的所有节点的哈希值以及秩值。



(a)原结构



(b)插入后的 MBT

图 7 插入多个数据块时 MBT 的更新过程

Fig. 7 Update process of MBT when multi-data blocks are inserted

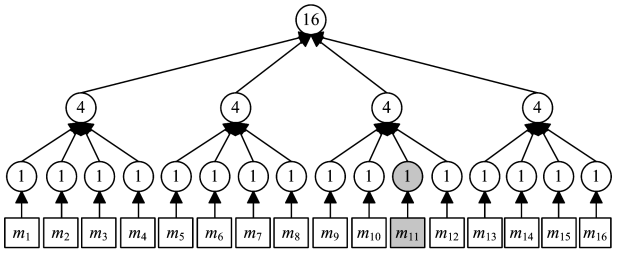
MBT 中可以插入多个连续数据块。如果一个 MBT 的

出度为 q,则它一次最多可以插入 $q-1$ 个数据块,如果插入的文件多于 $q-1$ 个,则可以将其分为多个包含 q 个数据块的集合,依次插入这些数据块集合。MBT 中多个数据块插入的示意图如图 7 所示,即直接找到需要插入的数据块,将该数据块与多个新的数据块生成一个新的公共父节点,用新的中间节点替换该数据块,并更新该数据块和新数据块的遍历路径节点上的所有节点的哈希值以及秩值。

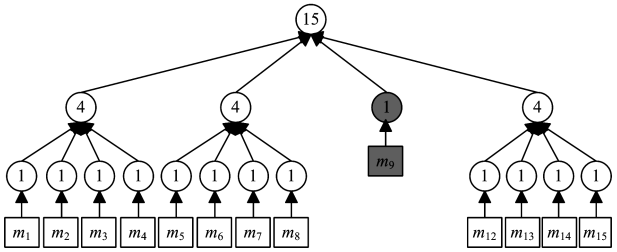
3)数据删除

数据删除操作也改变了数据文件的逻辑结构,因此数据删除操作是数据插入操作的反操作。在删除单个数据块时,在 MBT 中直接找到需要删除的数据块,删除该数据块及叶子节点,并更新该数据节点的遍历路径节点上的节点哈希值和秩值。

由于 MBT 中多个叶子节点共享一个父节点,因此 MBT 可以在一次删除操作中直接删除多个数据块,而不会影响其他节点的更新。如果一个 MBT 的出度为 q,那么它一次最多可以删除 $q-1$ 个数据块。这里的数据删除操作与删除单个数据块的数据删除操作相同。MBT 中删除多个连续数据块的更新过程如图 8 所示,即找到需要删除的数据块,删除该数据块及对应的叶子节点,如果删除的数据块个数大于当前公共父节点的出度,则分多次进行删除;如果删除的数据块个数等于该节点的出度减 1,则直接用剩下的一个叶子节点替换父节点,并更新该数据节点的遍历路径节点上的节点哈希值和秩值。



(a)原结构



(b)删除后的 MBT

图 8　删除多个数据块时 MBT 的更新过程

Fig. 8　Update process of MBT when multi-data blocks are deleted

3.2.3　批量审计

当不同用户在同一时间要求第三方代理认证不同数据文件时,需要第三方代理能够同时完成不同用户的委托。将不同用户的多个签名聚合成一个短标签进行一次性验证可以大大提高整体的验证效率。假设有 K 个用户,每一个用户 k 都有数据文件 $F_k=(m_{k,1},m_{k,2},\cdots,m_{k,n})(k\in\{1,\cdots,K\})$,则数据完整性的验证过程如下:每一个用户 k 用 KeyGen 算法生成公钥和私钥,用 SigGen 算法对数据文件 F 进行预处理并产生同态认证的元数据。

在 KeyGen 算法中,用户 k 生成随机签名密钥对(spk_k,ssk_k),选择随机参数 $\alpha_k\leftarrow Z_p$,计算 $v_k\leftarrow g^{\alpha_k}$,则私钥为 $sk_k=(\alpha_k,ssk_k)$,公钥为 $pk_k=(v_k,spk_k)$。

在 SigGen 算法中,给定 $F_k=(m_{k,1},m_{k,2},\cdots,m_{k,n})(k\in\{1,\cdots,K\})$,用户随机选择参数 $u_k\leftarrow G$,计算 $t_k=name_k\parallel n_k\parallel u_k\parallel SSig_{ssk_k}(name_k\parallel n_k\parallel u_k)$,并将其作为 F 的文件标签。用户为每一个文件块 $m_{k,i}(i=1,2,\cdots,n)$计算标签 $\sigma_{k,i}\leftarrow(H(m_{k,i})\cdot u_k^{m_{k,i}})^{\alpha_k}$,所有 $\sigma_{k,i}$ 的集合用 Φ_k 表示,即 $\Phi_k=\{\sigma_{k,i}\}(i=1,2,\cdots,n)$。用户 k 构造 MBT 并生成根 R_k,MBT 的叶子节点为文件块的哈希值的有序集合$\{H(m_{k,i})\}(i=1,2,\cdots,n)$。用户使用私钥 α_k 加密根 R_k,得到 $sig_{sk_k}(H(R_k))\leftarrow(H(R_k))^{\alpha_k}$。每一个用户 k 发送信息$\{F_k,t_k,\Phi_k,sig_{sk_k}(H(R_k))\}$给服务器,并从本地存储中删除$\{F_k,\Phi_k,sig_{sk_k}(H(R_k))\}$。

在 Audit 阶段,TPA 发送挑战信息 $chal=\{(i,\nu_i)\}$给服务器,对于每一个用户 k,服务器选择随机元素 $r_k\leftarrow Z_p$,计算 $\gamma_k'=e(u_k,v_k)^{r_k}$。计算 $\mu_k'=\sum_{i\in I}\nu_i m_{k,i}$得到$\mu_k=r_k+\gamma_k'$,并利用聚合标签计算 $\sigma_k=\prod_{i\in I}\sigma_{k,i}^{\nu_i}$。服务器计算 $\gamma'=\gamma_1'\cdot\gamma_2'\cdot\cdots\cdot\gamma_K'$,$\xi=v_1\parallel v_2\parallel\cdots\parallel v_k$ 和 $\gamma_k=h(\gamma'\parallel v_k\parallel\xi)$,发送响应信息$Proof=\{\{\sigma_k,\mu_k\}_{1\leqslant k\leqslant K},\gamma',\{H(m_{k_i}),\Omega_{k_i}\}_{1\leqslant k\leqslant K,i\in I},sig_{sk}(H(R_k))\}$到 TPA。

TPA 收到响应信息 $Proof$ 后,执行 VerifyProof 算法验证响应信息。TPA 首先用$\{H(m_{k,i}),\Omega_{k_i}\}_{1\leqslant k\leqslant K,i\in I}$计算每一个用户 k 的 MBT 的根 R_k,验证 $e(sig_{sk_k}(H(R_k)),g)$是否等于 $e((H(R_k))^{\alpha_k},g^{\alpha})$,若相等,则继续进行验证,否则输出 False。验证成功后,TPA 计算 $\gamma=h(\gamma')$并验证 $\gamma' e(\prod_{k=1}^{K}\sigma_k^{\gamma_k},g)$是否等于 $\prod_{k=1}^{K}e((\prod_{i\in I}H(m_{k,i})^{\nu_i})^{\gamma_k}\cdot u_k^{\mu_k},v_k)$。

4　安全性分析

4.1　正确性分析

数据完整性证明机制的正确性是指:在机制实施过程中,如果第三方代理和云存储服务器均为可信的,用户数据未被损坏,那么云存储服务器响应挑战而生成的证据一定能够成功通过数据完整性验证。因此,要证明该机制的正确性,只要证明云存储服务器返回的证据$\{\mu,\sigma,\gamma',\{H(m_i),\Omega_i\}_{i\in I},sig_{sk}(H(R))\}$可以成功通过证明即可。

1)由于 $\gamma'=e(u,v)^r$ 和 $\sigma=\prod_{i\in I}\sigma_i^{\nu_i}$,因此:

$$\gamma' e(\sigma^{\gamma},g)=e(u,v)^r\cdot e((\prod_{i\in I}\sigma_i^{\nu_i})^{\gamma},g) \tag{5}$$

又由于 $\sigma_i=(H(m_i)\cdot u^{m_i})^{\alpha}$,可以计算出:

$$e(u,v)^r\cdot e((\prod_{i\in I}\sigma_i^{\nu_i})^{\gamma},g)=e(u,v)^r\cdot e((\prod_{i\in I}(H(m_i)\cdot u^{m_i})^{\omega_i})^{\gamma},g) \tag{6}$$

根据双线性映射的性质,可得:

$$\begin{aligned}&e(u,v)^r\cdot e((\prod_{i\in I}(H(m_i)\cdot u^{m_i})^{\alpha\nu_i})^{\gamma},g)\\&=e(u^r,v)\cdot e((\prod_{i\in I}(H(m_i)\cdot u^{m_i})^{\nu_i})^{\gamma},g)^{\alpha}\\&=e(u^r,v)\cdot e((\prod_{i\in I}H(m_i)^{\nu_i}\cdot u^{m_i\nu_i})^{\gamma},g)^{\alpha}\end{aligned} \tag{7}$$

由于 $\mu'=\sum_{i\in I}\nu_i m_i$,可推导出:

$$e(u^r,v)\cdot e((\prod_{i\in I}H(m_i)^{\nu_i}\cdot u^{m_i\nu_i})^{\gamma},g)^{\alpha}$$

$$=e(u^{r},v)\cdot e((\prod_{i\in I}H(m_i)^{\nu_i})^{\gamma}\cdot u^{\mu'\gamma},g^{\alpha})$$
$$=e((\prod_{i\in I}H(m_i)^{\nu_i})^{\gamma}\cdot u^{\mu'\gamma+r},v) \tag{8}$$

因为 $\mu=r+\gamma\mu'$，所以：

$$e((\prod_{i\in I}H(m_i)^{\nu_i})^{\gamma}\cdot u^{\mu'\gamma+r},v)$$
$$=e((\prod_{i\in I}H(m_i)^{\nu_i})^{\gamma}\cdot u^{\mu},v) \tag{9}$$

因此，$\gamma' e(\sigma^{\gamma},g)=e((\prod_{i\in I}H(m_i)^{\nu_i})^{\gamma}\cdot u^{\mu},v)$ 成立，从而验证了证据 $\{\mu,\sigma,\gamma'\}$ 的正确性。

2）根据 MBT 结构，可得 $R\leftarrow\{H(m_i),\Omega_i\}_{i\in I}$，由于 $sig_{sk}(H(R))\leftarrow(H(R))^{\alpha}$，可以计算出：

$$e(sig_{sk}(H(R)),g)=e((H(R))^{\alpha},g) \tag{10}$$

根据双线性映射的性质，可得：

$$e(sig_{sk}(H(R)),g)=e(H(R),g^{\alpha}) \tag{11}$$

因此，证据 $\{\{H(m_i),\Omega_i\}_{i\in I},sig_{sk}(H(R))\}$ 的正确性得到验证。

由 1）和 2）的结论可知，云存储服务器返回的证据 $\{\mu,\sigma,\gamma',\{H(m_i),\Omega_i\}_{i\in I},sig_{sk}(H(R))\}$ 是正确的。

4.2 伪造攻击

本文提出的方案使用了 MBT 这一动态数据结构来实现动态操作特性，在执行动态操作时，用户的数据被更新，该数据块对应的标签也随之更新，并且该数据块在 MBT 结构中的认证路径上的节点的哈希值也相应被更新。另外，在本文方案中，数据标签的格式为 $\sigma=\prod_{i\in I}\sigma_i^{\nu_i}$，因此该数据标签只能由云存储服务器根据存储的数据块的对应标签集合来生成。在验证 MBT 结构的正确性之前，需要先验证用户数据标签的正确性，以防云存储服务器的代理提供商使用伪造的数据标签来通过验证。本文提出的验证方案中，第三方验证者已经对数据标签的正确性进行了证明。根据 BLS 标签的特性和哈希函数的单向性可得，伪造的数据标签无法通过数据完整性证明，从而证明了云存储服务器的代理提供商无法使用伪造标签通过数据完整性证明。

4.3 重放攻击

在本文方案的验证过程中，第三方验证者使用云存储服务器返回的响应信息中的辅助信息和随机数据块子集来重新构造 MBT，通过计算根节点 R 的值来证明标签 σ 的有效性。在这个过程中，第三方验证者已经对 MBT 和数据块位置进行了验证。在验证过程中，如果用户在云服务器上存储的数据发生变化，数据块的标签信息也会随之改变。根据第三方验证者对标签的验证过程可得，过期的数据块标签将无法通过第三方验证者的验证，从而证明了本文方案能够抵抗重放攻击。

4.4 删除攻击

为了节约存储资源，提升自身利益，云存储服务器的代理提供商可能会删除用户不常访问的数据文件，并利用预先计算好的数据块标签来欺骗验证者以通过完整性证明。为了抵抗这类删除攻击，不仅要验证数据块的对应标签的正确性，还要排除服务器可以预先聚合出标签块集合的可能性。验证者随机地从 n 个数据块中选取 q 个数据块进行验证，聚合数据块将有 C_n^q 种不同的组合，其存储量远大于 n。云存储服务器的代理提供商若想通过验证者的数据完整性验证，则需要预先计算出所有可能使用的聚合数据块标签，这将需要付出巨大的存储代价，云存储提供商不可能做出这样不获利益的事情，从而证明了本文提出的数据完整性证明机制能够抵抗恶意的云存储提供商的删除攻击。

5 性能分析

本文方案与其他典型方案的性能比较如表 1 所列，其中 k 为 MBT 树的出度，n 为数据块数量，c 为抽样块数量。本文方案在与其他方案具有相同开销的情况下支持公开验证、动态操作、数据隐私等基本功能，还支持批量审计特性。

表 1 方案性能的比较

Table 1 Comparison of scheme performance

方案	文献[1]的方案	文献[4]的方案	文献[8]的方案	文献[17]的方案	文献[18]的方案	本文方案
公开验证	Yes	No	No	No	Yes	Yes
动态更新	No	Yes	Yes	Yes	Yes	Yes
安全性	Yes	Yes	No	Yes	Yes	Yes
批量审计	No	No	No	No	No	Yes
验证者计算开销	$O(c)$	$O(c)$	$O(c)$	$O(c)$	$O(c)$	$O(\log n)$
通信开销	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
服务器计算开销	$O(c)$	$O(c)$	$O(n)+O(\log n)$	$O(\log n)$	$qO(\log n)$	$O(\log n)$
客户端计算开销	—	—	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log_k n)$

为了分析本文方案构建 MBT 和完成数据完整性证明的时间，保证实验结果的随机性，采用随机生成的数据文件作为输入，并取 10 次实验结果的平均值作为最终实验结果。在实验过程中建立 3 台虚拟机搭建 Hadoop 环境，用于模拟实验的云存储服务器，并使用 Java 语言编程实现方案的基本功能。方案中的哈希运算采用 MD5 算法，双线性对运算通过 JPBC 库实现。本文实验中使用的数据文件均为随机生成的数据文件，所有实验结果均取 10 次实验结果的最好结果。

本次实验中计算机的参数规格如表 2 所列。

表 2 实验计算机的参数规格

Table 2 Parameters specifications of experimental computer

参数名称	规格
操作系统	Windows 7
处理器型号	Intel Core i5-4680
内存	24 GB
硬盘	1 TB

本文设计的云存储中，大数据完整性验证机制主要包括客户端和服务器端两个部分，系统架构如图 9 所示。客户端对数据进行预处理，云服务器接收并执行验证请求。

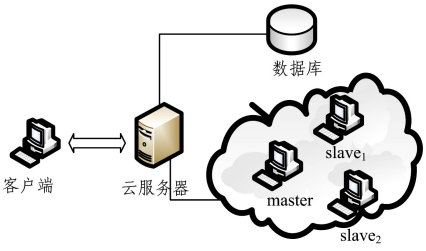


图 9　实验系统架构

Fig. 9　Experimental system architecture

为了验证方案的正确性和有效性，进行以下实验。

实验 1　本实验主要验证本文方案的有效性。分别对 3 个相同的并且大小为 200 M 的数据文件进行预处理，然后将其存储到云服务器上，标记为文件 A、文件 B 和文件 C。对 3 个文件分别进行以下操作：删除文件 A 中 10%的数据块，修改文件 B 中 10%的数据块，文件 C 不做任何处理。最后对这 3 个文件进行完整性检测，检测结果如表 3 所列。实验表明，被破坏的数据文件可以通过该方案被检测出来，完整的数据文件可以成功通过方案检查。

表 3　文件完整性检测结果

Table 3　Results of file integrity verification

文件	验证结果
A	False
B	False
C	True

实验 2　每一次进行数据完整性证明时，云服务器都需要从 MBT 中找到对应的挑战数据块从而生成对应的证据，发出挑战的数据块数量越多，越需要花费更长的时间生成对应的证据。$p=1-(1-t)^c$ ($t=1-\rho$)，其中 ρ 为容错率，t 为损坏率，p 为检测率，c 为抽样块数量。当抽样 460 块时检测率为 99%；当抽样 300 块时检测率为 95%。实验对 200 M 大小的文件按照每个数据块 1 kB 进行分块，每次从中抽取 50～500 不同数量的数据块进行计算，比较不同出度的树对不同数据块数量生成证据的时间，实验结果如图 10 所示。结果显示，MBT 出度越大，云服务器的计算时间越短，抽样数据块的数量越少，计算效率越高。

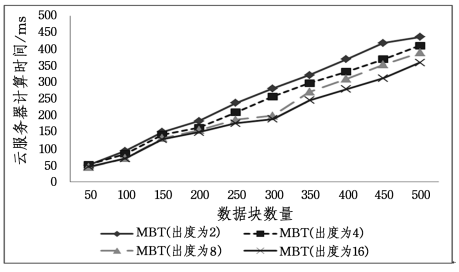


图 10　云服务器计算时间变化图

Fig. 10　Computing time of cloud server

实验 3　在实验过程中，为了分析构造认证树的时间，比较 MBT 与 MHT 的构造时间的差别，对 200～1000 M 大小的文件按照每个数据块 1 kB 进行分块，分别计算不同出度时认证树的构造时间，结果如图 11 所示。结果显示，MBT 的构造时间相比 MHT 的构造时间大大缩短，并且出度越大，构造时间越短，减轻了计算负担，且出度越大的 MBT 在数据块数量越多的情况下表现得越好。

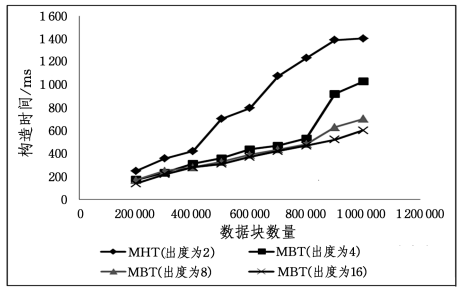


图 11　MBT 构造时间的变化图

Fig. 11　Construction time of MBT

实验 4　第三方代理收到云服务器返回的响应信息后，执行验证算法来验证数据文件的完整性。如果有多个用户同时进行请求，串行处理用户请求的时间和批量处理用户请求的时间如图 12 所示。结果显示，串行处理用户请求的时间随着用户数量呈线性增长，而批量处理用户请求由于使用聚合标签进行一次性验证，因此验证时间基本保持不变，并且不随用户数量的增长而增长，减轻了第三方代理的计算负担。

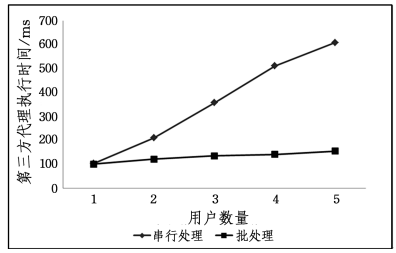


图 12　第三方代理执行时间的变化图

Fig. 12　Execution time of the third party auditor

结束语　云存储的推广使得很多传统的数据完整性证明机制难以直接应用在新的环境下，出现了诸多问题。为了支持动态操作，数据完整性证明机制需要借助动态数据结构，因此本文引入多分支路径树作为数据完整性证明机制中使用的数据结构，并提出了基于 MBT 的支持数据隐私的数据完整性证明机制，同时提出了适用于 MBT 的数据完整性检测算法。本文提出的方案不仅保护了数据对第三方代理的隐私性，而且支持全动态操作与批量审计操作，简化了数据完整性证明信息的获取过程。正确性与安全性分析表明，本文方案切实可行。在未来的研究工作中，将继续改进方案，减少辅助信息的存储，以降低通信与存储代价。

参 考 文 献

[1]　ATENIESE G, BURNS R, CURTMOLA R, et al. Provable data possession at untrusted stores[C]// Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM, 2007: 598-609.

[2]　THANGAVEL M, VARALAKSHMI P, SINDHUJA R, et al. A

survey on provable data possession in cloud storage[C]// 2016 Eighth International Conference on Advanced Computing (ICoAC). IEEE, 2017: 25-31.

[3] TAN S, JIA Y, HAN W H. Research and Development of Provable Data Integrity in Cloud Storage[J]. Chinese Journal of Comptuers, 2015, 38(1): 164-177. (in Chinese)
谭霜, 贾焰, 韩伟红. 云存储中的数据完整性证明研究及进展[J]. 计算机学报, 2015, 38(1): 164-177.

[4] JUELS A, KALISKI JR B S. PORs: Proofs of retrievability for large files[C]// Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM, 2007: 584-597.

[5] ATENIESE G, DI PIETRO R, MANCINI L V, et al. Scalable and efficient provable data possession[C]// Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks. ACM, 2008: 9.

[6] WANG C, WANG Q, REN K, et al. Ensuring Data Storage Security in Cloud Computing[C]// 2009 17th International Workshop on Quality of Service. IEEE, 2009: 1-9.

[7] WANG Q, WANG C, LI J, et al. Enabling public verifiability and data dynamics for storage security in cloud computing[C]// European Symposium on Research in Computer Security. Springer Berlin Heidelberg, 2009: 355-370.

[8] WANG Q, WANG C, REN K, et al. Enabling public auditability and data dynamics for storage security in cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(5): 847-859.

[9] ERWAY C C, KÜPÇÜ A, PAPAMANTHOU C, et al. Dynamic Provable Data Possession[J]. ACM Transactions on Information & System Security, 2015, 17(4): 1-29.

[10] ERWAY C C, KÜPÇÜ A, PAPAMANTHOU C, et al. Dynamic provable data possession[C]// Proceedings of the 16th ACM Conference on Computer and Communications Security. ACM, 2009: 213-222.

[11] WANG C, WANG Q, REN K, et al. Privacy-preserving public auditing for data storage security in cloud computing[C]// INFOCOM, 2010 Proceedings IEEE. IEEE, 2010: 1-9.

[12] WANG C, CHOW S S M, WANG Q, et al. Privacy-preserving public auditing for secure cloud storage[J]. IEEE Transactions on Computers, 2013, 62(2): 362-375.

[13] ZHANG Y, BLANTON M. Efficient dynamic provable possession of remote data via balanced update trees[C]// Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. ACM, 2013: 183-194.

[14] LIN C, LUO F, WANG H, et al. A provable data possession scheme with data hierarchy in cloud[C]// International Conference on Information Security and Cryptology. Springer International Publishing, 2015: 301-321.

[15] WANG H. Identity-based distributed provable data possession in multicloudstorage[J]. IEEE Transactions on Service Computing, 2015, 8(2): 328-340.

[16] ETEMAD M, KÜPÇÜ A. Transparent, distributed, and replicated dynamic provable data possession[C]// International Conference on Applied Cryptography and Network Security. Springer Berlin Heidelberg, 2013: 1-18.

[17] ZOU J, SUN Y, LI S. Dynamic Provable Data Possession Based on Ranked Merkle Hash Tree[C]// 2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI). IEEE, 2016: 4-9.

[18] LI L, YANG Y, WU Z. FMR-PDP: Flexible multiple-replica provable data possession in cloud storage[C]// 2017 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2017: 1115-1121.

[19] PENG S, ZHOU F, WANG Q, et al. Identity-Based Public Multi-Replica Provable Data Possession[J]. IEEE Access, 2017, 5: 26990-27001

[20] RAJENDRAN A, BALASUBRAMANIAN V, MALA T. Integrity verification using Identity based Provable Data Possession in multi storage cloud[C]// 2017 International Conference on Computational Intelligence in Data Science (ICCIDS). IEEE, 2017: 1-4.

[21] HUANG D, WAN C. PDPMT: Provable data possession for multiple cloud tenants[C]// 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). IEEE, 2017: 1-6.