

云计算中数据存储安全的变色龙 Hash 认证树优化审计

李 斌¹, 李启明²

(1 福建师范大学 协和学院, 福建 福州 350117; 2 南京大学 计算机软件新技术国家重点实验室, 江苏 南京 210023)

摘 要: 为提高云计算中数据存储安全性, 提出一种云计算中数据存储安全的变色龙 Hash 认证树优化审计方法。首先, 提出了一种优化的公共审计协议。通过为 TPA 站点上的用户数据进行同态线性验证器存储, 实现对云存储服务器(CSS)响应大小的优化。同时还利用准随机函数优化了对 CSS 的质询请求; 其次, 使用变色龙散列和一个改进的变色龙认证树, 在客户端数据(云计算)上执行高效的动态数据更新, 支持块级更新和细粒度更新; 最后, 通过彻底的安全性和性能分析, 证明了所提方法是安全和高效率的。

关键词: 云计算; 数据存储; 变色龙认证树; 第三方审计; 准随机函数

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2018)06-0001-06

Chameleon Hash Authentication Tree Optimization Audit for Data Storage Security in Cloud Computing

LI Bin¹, Li Qi-ming²

(1 Concord University College Fujian Normal University, Fuzhou 350117, China;

2 Information Engineering College, Shanghai Maritime University, Shanghai 201306, China)

Abstract: In order to improve the security of data storage in cloud computing, a chameleon Hash authentication tree optimized audit method for data storage security in cloud computing is proposed. Firstly, an optimized public audit protocol is proposed. The validator homomorphic linear stored as user data on the TPA site, the cloud storage server (CSS) the size of the optimal response. At the same time, the query request of CSS is optimized by using quasi random function; Secondly, using chameleon hashing and an improved chameleon authentication tree, efficient dynamic data updates are performed on the client data (cloud), supporting block level updates and fine grained updates; Finally, through thorough security and performance analysis, it is proved that the proposed method is secure and efficient.

Key words: cloud computing; data storage; chameleon authentication tree; third party audit; quasi random function

1 引言

云计算是一种基于互联网的计算模型, 是当今计算技术发展的重要方向之一^[1-3]。

然而, 云计算的使用也存在很多挑战, 涉及到整个系统的安全性、保密性和可靠性, 其中一个主要问题是数据安全, 包含完整性验证和动态数据更新, 现有的研究工作可分为静态协议和动态协议两部分^[4]。静态协议处理存储在未更新的云上的归档数

据的完整性验证。动态协议处理客户定期更新的数据(存储在云上)的完整性验证。文献[5]提出的证据提取模型(POR)是一个静态模型, 因为该方案只适用于静态(档案)数据存储。文献[6]提出的可证明的数据占有方案是一种基于数据块的验证方案, 验证器可以通过验证预计算文件标记的组合来验证外包文件的完整性。文献[7]提出支持完整性验证和完整数据动态的方案, 使用基于秩的跳过列表作为验证数据结构以验证更新。但公开验证不通过该方案支

持. 文献[8]提出一个方案提供动态数据更新和公共审计, 这意味着客户端的数据完整性验证可以由第三方审核员完成^[8-9]. 文献[11]提出了一种支持任意大小可变数据块的更新方案. 此外, 还提出了一种客户端和 TPA 之间的认证机制, 以防止未授权的 TPA 带来无限的质询. 文献[12]提出了一种资源受限设备的方案, 但是并未考虑第三方审计问题.

在本文中, 通过在现有协议中提出一些修改来优化数据审计, 还通过修改变色龙认证树, 提出了一种高效的客户端/用户数据(云上)动态更新的构造方法, 最后通过仿真实验对算法有效性进行了验证.

2 问题描述

2.1 系统模型

用于云数据存储的网络架构如图 1 所示. 它有三种不同的网络实体, 可以识别如下.

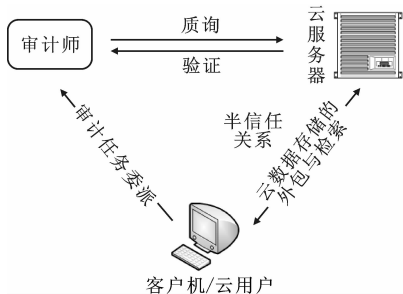


图 1 云数据存储的网络架构

(1) 客户机/云用户 (Client/Cloud-User,): 实体可以是单个消费者或组织, 有大量的数据文件存储在云中, 并且依赖于云进行数据维护和计算.

(2) 云存储服务器 (Cloud Storage Server, CSS): 具有显著的存储空间、计算能力和资源以维护客户数据的实体. 它由云服务提供商 (CSP) 管理. 这里, 我们将不区分 CSS 和 CSP.

(3) 第三方审核员 (TPA) (Third Party Auditor, TPA): 具有访问和公开与云存储服务相关联的风险的能力和专门知识的实体. 根据客户要求, 代表客户执行此任务是可信的.

本文认为审计任务是由授权的非公正的 TPA 完成的. CSS 被认为是半可信的.

2.2 安全威胁

本文中考虑了三种类型的数据/元数据攻击.

(1) 替换攻击 (Replace Attacks): 恶意的 CSS 可能会忽略存在疑问的数据块 (m_i) 或其元数据 (σ_i), 并利用另一对有效的和未损坏的数据块 (m_j) 和元数据 (σ_j) 进行替换, 以便通过审计.

(2) 伪造攻击 (Forge Attacks): 恶意的 CSS 或审计师 (TPA) 不得伪造块元数据 (HLAS) 造成不合理和满意数据审计.

(3) 重播攻击 (Replay Attacks): 恶意 CSS 使用未更新/以前的数据、以及以前的证据或其他信息生成的证据, 而不查询客户端的实际数据.

2.3 设计目标

本文的设计目标如下.

(1) 云数据存储优化公共审计 (Optimized Public Auditability of cloud data storage, OPACDS): 允许授权 TPA 验证存储在云上的客户端数据的正确性, 此过程无需对其进行检索, 不产生任何额外的在线客户/云用户的负担.

(2) 动态数据更新 (Support for dynamic data update operations): 允许云用户/客户端以尽可能高效的方式在其文件上执行块级和细粒度更新.

2.4 变色龙认证树

变色龙认证树是一个广义的 Merkle 哈希树, 每一个节点的右子节点等于其子节点的变色龙哈希值, 每一个节点的左子节点等于子节点的串联拼接的简单哈希值. 图 2 给出本文提出的一种改进版本的变色龙认证树结构 (mCAT).

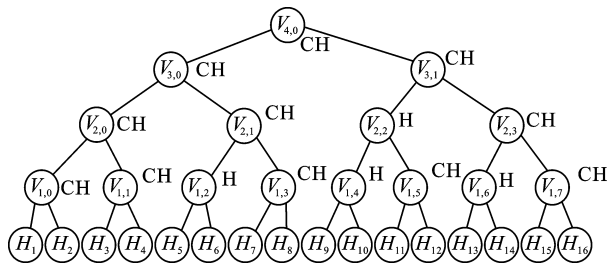


图 2 改进版本的变色龙认证树结构

图 2 所示改进版本的变色龙认证树 (mCAT) 结构的构建方式将在下一节中进行具体描述.

3 基于 mCAT 的公共审计和动态数据更新

为保证数据的完整性, 提出了一种优化的公共审计和动态数据更新方案. 它由三个阶段组成:

(1) 设置阶段: 本阶段包括: 密钥生成、生产块元数据 (HLAS) 的文件预处理、文件的 mCAT 结构生成、授权第三方审核员.

(2) 动态数据更新: 在这个阶段, 客户端在云中使用 MCAT 算法执行块级和细粒度的存储数据更新. 然后, 计算修改块的新 HLA 并将其存储在 TPA 站点上.

(3)第三方审计:在这个阶段,授权的第三方审计员(TPA)向 CSS 发送一个质疑请求. CSS 返回一个完整性证明,与受挑战块的集合相对应,返回到 TPA. 此后,TPA 验证了受挑战的块集的完整性.

3.1 设置阶段

令 G 是一个 p 阶素数集,由 Z_p 构成,其中 p 是大素数, $H: (0,1)^* \rightarrow G$ 是一个抗冲突散列函数, $h()$ 是加密散列函数, $ch()$ 是变色龙散列函数,且有 $e: G \times G \rightarrow G_T$ 是一个双线性映射. 客户机首先与其他方(CSS 和 TPA)协商这些参数,然后运行以下算法:

KeyGen(1^λ):客户端产生数值 $\alpha \in Z_p$, 以及生成器 $g \in G$. α 成为密钥的一部分, (g, g^α) 成为公钥的一部分. 客户端使用 $chGen(1^\lambda)$ 为变色龙哈希函数 $ch()$ 生成密钥对 (csk, cpk) , 其中 csk 是密钥, cpk 是这个变色龙散列函数的公钥. 最后,客户端的密钥是: (α, csk) , 公钥是: (g, g^α, cpk) .

HLA 生成:客户机将文件 F 分成大小相等的 n 个块. 由此可得, $F = (m_1, m_2, m_3, \dots, m_n)$, 其中 $m_i \in Z_p$. 它生成一个随机数 $u \leftarrow G$. 然后,对于每个数据块 $m_i (i = 1, 2, \dots, n)$, 客户端生成一个同态线性认证(HLA): $\sigma_i \leftarrow (H(m_i) \cdot u^{m_i})^\alpha$. 由此可得到 n 组 HLAs, 其集合可表示为 $\varphi = \{\sigma_i\}, 1 \leq i \leq n$. 然后,客户机创建一个 mCAT, 其叶节点是哈希文件块 $(h(m_i), 1 \leq i \leq n)$.

授权 TPA:客户选择一个 TPA 并询问它的 ID. 客户端生成一个授权信息 k_{auth} , 并生成签名 $sig_{auth} = (k_{auth} || ID)^\alpha$. 其发送签名 sig_{auth} , 审计委托对 TPA 进行查询.

最后,客户存储 CSS 中的 $\{F, T, k_{auth}\}$ 和 TPA 中的 $\{\varphi, u\}$. 然后,从其存储记忆中删除 $\{F, T, k_{auth}, \varphi\}$. 这样可保证 mCAT 算法仅使用未被定义使用的节点.

3.2 动态数据更新阶段

在这个阶段,客户机/云用户与 CSS 交互,以执行对其数据的更新. 更新阶段支持以下更新请求: PM(数据块的部分修改)和 R(数据块的替换). 考虑到图 2, PM 的动态数据更新过程如下:

(1) 客户端编写请求: $BlockRequest = \{i\}$, 其中 i 是要修改的块的索引,并将其发送至 CSS.

(2) CSS 接收 $BlockRequest$ 并执行以下操作: CSS 对 m_i 定位,并为 m_i 计算 mCAT 身份验证路径:

$$Response = \{m_i, h(m_{adj}), aPath, sig_c(aPath), v_{h,0}\} \quad (1)$$

式中, m_{adj} 是 m_i 的兄弟节点, $aPath$ 是数据块 m_i 的验证路径, $sig_c(aPath) = (aPath)^\alpha$ 是 $aPath$ 上的客户签名, $v_{h,0}$ 是最左边的节点,可以实现从 $h(m_i)$ 沿着路径移动到根节点. 最后,它向客户机/云用户发送响应.

(3) 在接收到响应后,客户机执行操作 $VerifyAuth$:首先,客户端验证 $aPath$ 签名. 即检查:

$$e(sig_c(aPath), g) == e(aPath, g^\alpha) \quad (2)$$

如果这个测试失败,则客户端中断进程. 否则,客户端计算 $h(m_i)$. 然后客户端利用 $h(m_{adj})$ 、 $aPath$ 和 $h(m_i)$ 计算 $v'_{h,0}$. 最后,客户端检查是否满足下列条件: $v_{h,0} == v'_{h,0}$.

如果条件 $v_{h,0} == v'_{h,0}$ 仍然不满足,则客户端中断进程. 否则,客户端计算 m'_i 和 $h(m'_i)$, 用它们来为第一节点(CAT 中的 $v_{h,i}$) 生成碰撞 R 值(使用 $col()$ 函数),该值可利用变色龙 Hash 函数进行计算,并依赖于从节点 $h(m'_i)$ 到其余节点 $v_{h,0}$ 的路径($h \geq h'$). 然后,客户端使用 r' 更新验证路径 $aPath \rightarrow aPath'$, 并计算路径 $aPath'$ 的签名,即 $sig_c(aPath') = (aPath')^\alpha$. 客户端形成更新请求:

$$UpdateRequest = \{PM, i, o, data, aPath', sig_c(aPath')\} \quad (3)$$

并将此更新请求发送至 CSS. CSS 从 m_i 中的偏移量 o 开始,利用数据将数据块将 m_i 更新为 m'_i . 同时,其利用 $aPath'$ 中新的随机数对 mCAT 中的认证路径进行更新,然后利用 $sig_c(aPath')$ 取代旧的 $sig_c(aPath)$. 然后,它向客户机发送一个“更新成功”的响应.

(4) 在收到 CSS 的“更新成功”响应后,客户端为修改数据块 m'_i 计算新的 HLA: $\sigma'_i \leftarrow (H(m'_i) \cdot u^{m'_i})^\alpha$, 并发送其到 TPA. TPA 利用新的 HLA(σ'_i) 替换旧的 HLA(σ_i). 此后,客户端从内存中删除 m'_i .

3.3 第三方审计阶段

在这个阶段,授权 TPA(sig_{auth} 由客户端发送)与 CSS 共同来确定客户端的数据的完整性. 主要执行三个操作:

(1) 质询请求:此操作是由 TPA 执行的. 在安装阶段其利用客户端对 $\{\varphi, u, sig_{auth}\}$ 进行共享. TPA 取其 ID 并用 CSS 的公钥对其加密: $\{ID\}_{PK_{CSS}}$. 现在, TPA 使用准随机函数 $f_s()$ 生成 c 个不同块指数. 令 $I \subseteq [1, n]$ 是由准随机函数 $f_s()$ 生成的 c 块指数集. 现在, TPA 选择 c 个随机系数表示为 $\{v_i \in Z_p\}_{i \in I}$.

最后,它形成了一个质询请求, $\text{ChallengeReq} = \{\{ID\}_{\text{PKCSS}}, \{i, v_i\}_{i \in I}, \text{sig}_{\text{auth}}\}$, 并利用将其发送至 CSS.

(2) 完整性证明: 在接收到质询信息 $\text{ChallengeReq} = \{u, \{ID\}_{\text{PKCSS}}, \{i, v_i\}_{i \in I}, \text{sig}_{\text{auth}}\}$ 后, CSS 对 sig_{auth} 进行验证. 为此, CSS 首先使用其私钥 SK_{CSS} 以及 $k_{\text{auth}} (i.e. k_{\text{auth}} \parallel ID)$ 解密 ID. 然后, 通过下式进行验证:

$$e(\text{sig}_{\text{auth}}, g) = e(k_{\text{auth}} \parallel ID, g^a) \quad (4)$$

如果 $\text{signature}(\text{sig}_{\text{auth}})$ 验证失败, 然后, 它拒绝 ChallengeReq . 否则, CSS 计算 $\mu = \sum_{i \in I} v_i m_i \in Z_p$ 和 $\xi = \prod_{i \in I} H(m_i)^{v_i}$, 其中 $i \in I$. 最后, 它发送证据 $\text{IntegrityProof} = \{\mu, \xi\}$ 到 TPA.

(3) 完整性验证. 在从 CSS 中接收到 $\text{IntegrityProof} = \{\mu, \xi\}$ 信息后, TPA 执行操作 VerifyIntegrity 其包含: TPA 使用 $\{i, v_i\}_{i \in I}$ 并存储文件块的 $\sigma_i (i \in I)$, 计算 $\sigma = \prod_{i \in I} \sigma_i^{v_i} \in G$, 并验证下式是否成立:

$$e(\sigma, g) = e(\xi, u^a, g^a) \quad (5)$$

如果不是这样, 那么它意味着存储在 CSS 上的客户机数据违反完整性要求, TPA 负责向客户机报告这种情况.

4 安全性分析

在这一节中, 首先阐述各个阶段的正确性. 然后, 将讨论所提方案如何抵御各种存储在云中的数据攻击.

4.1 正确性分析

所提方案有两个验证过程. 首先, 动态数据更新阶段, 客户机请求从 CSS 中更新块 (m_i), 并检查此块中的数据及其认证路径的正确性进行分析. 第二, 在第三方审核阶段, TPA 通过概率证明验证了 CSS 生成数据的完整性. 通过验证下面的定理可以证明这两个验证过程的正确性:

定理 1: 由于 CSS 在这些文件块中存储数据块文件, TPA 这些文件块中存储 HLAS/ 签名, 在第三方审核阶段, TPA 能够正确地核实这些区块的占有情况.

证明: 提供审计方案的正确性等价于证明方程 (5) 中等式 $e(\sigma, g) = e(\xi, u^a, g^a)$ 的正确性. 其正确性可以阐述如下:

令 $I = [s_1, s_2, \dots, s_c]$ 用准随机函数生成一组 c 个不同的指数. 形式为:

$$\begin{aligned} e(\sigma, g) &= e\left(\prod_{i=s_1}^{s_c} \sigma_i^{v_i}, g\right) \\ &= e\left(\prod_{i=s_1}^{s_c} ((H(m_i), u^{m_i})^a)^{v_i}, g\right) \\ &= e(\xi, u^a, g^a) \end{aligned} \quad (6)$$

证毕!

定理 2: 由于 CSS 存储了文件 F 的数据块, 并利用变色龙认证树来验证该文件, 客户机能够正确地验证数据块上的先前更新是否真的发生了.

证明: 在更新同一数据块之前对块上的更新进行正确性验证, 相当于证明式 (1) 中关系式 $e(\text{sig}_c(a\text{Path}), g) = e(a\text{Path}, g^a)$ 的正确性. 式 (1) 的正确性可以阐述如下:

$$\begin{aligned} e(\text{sig}_c(a\text{Path}), g) &= e((a\text{Path})^a, g) \\ &= e(a\text{Path}, g^{a \cdot 1}) \\ &= e(a\text{Path}, g)^{1 \cdot a} \\ &= e(a\text{Path}, g^a) \end{aligned} \quad (7)$$

4.2 抗攻击能力

定理 3: 所提动态审计协议可以抵抗服务器的替换攻击.

证明: 验证式 (5): 如果 $e(\sigma, g) = e(\xi, u^a, g^a)$ 在 CSS 上受到任何质询, 则数据块在未经客户同意的情况下被损坏或修改, 将无法保存. 现在, 为了隐藏一些损坏或修改的数据块 (m_k), CSS 可以用 m_j 对其进行替换. 如果第 k 个数据块受到质询, 则 CSS 将形成以下响应:

$$\xi = (H(m_j))^{v_k} \cdot \prod_{i \in I - \{k\}} (H(m_j))^{v_i} \quad (8)$$

$$\mu = v_k m_j + \sum_{i \in I - \{k\}} v_i m_i \quad (9)$$

然后, 将 (ξ, μ) 发送到 TPA 中, 由 TPA 对式 (5) 进行验证.

令 $I = [s_1, s_2, \dots, s_c]$ 是表示受质询数据块集的索引集.

LHS:

$$\begin{aligned} e(\sigma, g) &= e\left(\prod_{i=s_1}^{s_c} \sigma_i^{v_i}, g\right) \\ &= e((H(m_k), u^{m_k})^{v_k} \cdot \prod_{i \in I - \{k\}} ((H(m_i), u^{m_i})^{v_i}), g) \end{aligned} \quad (10)$$

RHS:

$$\begin{aligned} &e(\xi, u^a, g^a) \\ &= e((H(m_k), u^{m_k})^{v_k} \cdot \left[\prod_{i \in I - \{k\}} ((H(m_i), u^{m_i})^{v_i})\right], g)^a \end{aligned} \quad (11)$$

现在, 由于哈希函数 $H()$ 抗冲突性, $H(m_k) \neq H(m_j)$. 因此, $((H(m_j), u^{m_j})^{v_k}) \neq ((H(m_k), u^{m_k})^{v_k})$,

且有 $LHS \neq RHS$. 因此,如果 CSS 用一个正确的数据块 m_k 替换一个损坏的数据块 m_j , 相应的 HLA(在 TPA) 不能用 CSS 替换. 由于这一点,验证等式(3) 将不能成立,而 CSS 的证明不能通过审核. 因此,我们的动态审计协议可以抵抗 CSS 替换攻击.

5 性能分析

我们建立自己的云测试环境,利用三个 64 位的 Linux 机器,其处理器为英特尔 i7,主频为 3.0 GHz,系统内存为 16 GB. 其中两个充当 OpenStack 快速对象存储节点,第三个充当控制器节点的功能,第四个配置相同的系统作为审计服务器. 利用一台具有 Intel (R) Core(TM) 双核心的 CPU 的 Linux 机器,其主频为 2.40 GHz,内存大小为 4 GB 作为用户^[13-14]. 双线性映射的 GDH 群 G 的 p 阶素数的大小是 160 位长度. G_T 长度为 512 位. 选取参数 s 的大小为 20、40、80 bytes,文件大小选取为 10、20、30、40 kB. 对于不同的文件大小和块大小 MCAT 计算时间见图 3 所示. 动态数据更新过程中,对于不同的文件大小和块指数 MCAT 认证和认证路径更新计算时间,见图 4 所示.

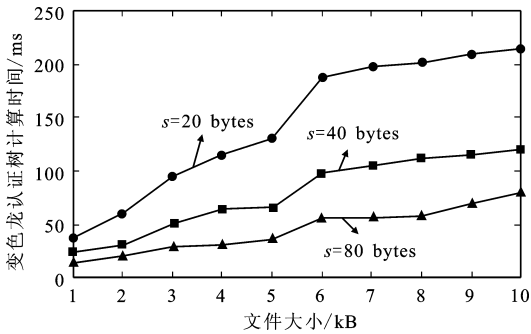


图 3 不同的文件大小和块大小 MCAT 计算时间

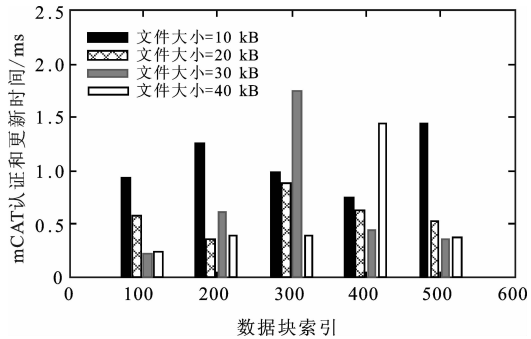


图 4 不同的文件大小和块指数 MCAT 认证和认证路径更新计算时间

图 3 中,对 mMCAT 算法在客户端的计算成本进行了验证,在这里,通过增加文件的大小,对 mCAT 算法增加的计算成本进行分析. 同时,对于一个特定的文件的大小(比如 5 kB),数据块的大小增加(s)会导致计算成本增加. 在图 4 中,展示了对文件块进行身份验证的总体计算时间,然后对不同块索引和不同大小文件的动态数据更新周期的认证路径进行修改和更新.

在图 5 中展示了每一项任务的总体计算时间,用于质询生成+响应生成+完整性验证,增加了现有方案和所提方案的审计任务数量. 每个审计任务都需要将一组块索引发送到 CSS 作为质询请求的一部分,然后响应生成并验证这些挑战块的完整性. 图 6 给出了 mCAT 和 Merkle 哈希树(MHT)的计算结果对比情况,动态更新请求是大小为 2 kB 的 32 字节的文件. 在 MHT 算法中,在树结构的每个级别上进行哈希值计算和更新操作. 如果存在 2^n 个数据块(即 2^n 个叶节点),将计算每一个 m_i ($1 \leq i \leq n$)更新的 $\log_2(n)$ 哈希值. 也就是说,不管更新的数据块 m_i 是否位于文件 F 中,它都保持不变. 可知,本文算法的哈希值计算数量要远低于 MHT 算法的哈希值计算数量.

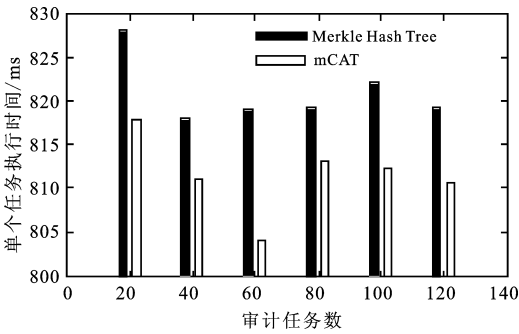


图 5 审计任务数量对计算时间影响

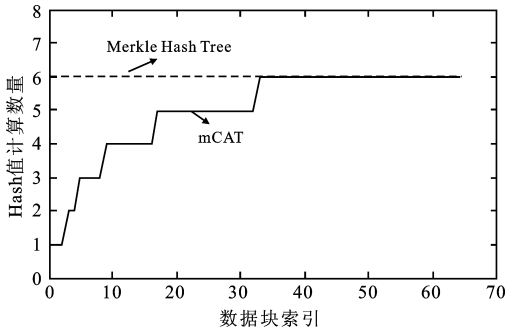


图 6 MCAT 和 MHT 的哈希值计算数对比

6 结束语

本文通过将 HLA 从 CSS 站点迁移到 TPA 站

点,优化了云环境下客户端数据的公开审计^[15].我们还提出了一个采用改进的变色龙认证树(mCAT)的动态数据更新协议.在这里,我们证明了我们的优化审计协议的安全性,显示它可以抵抗重播、替换和伪造攻击.同时,通过理论分析对算法的安全性和完整性验证的有效性进行了分析和证明.

参考文献:

- [1] 王国峰,刘川意,潘鹤中,等.云计算模式内部威胁综述[J].计算机学报,2017,40(2):296-315.
- [2] 雷蕾,蔡权伟,荆继武,等.支持策略隐藏的加密云存储访问控制机制[J].软件学报,2016,27(6):1432-1450.
- [3] 常亚楠,张志勇,韩林茜.一种可信接入云安全协议[J].微电子学与计算机,2016,33(2):34-38.
- [4] 郭保平,程建.基于云物元分析法的军事云计算安全评估研究[J].微电子学与计算机,2016,33(7):49-53.
- [5] Raazia Sosan, Choudhry Fahad Azim. Retracted article: mobile cloud computing: the taxonomy and comparison of mobile cloud computing application models[J]. Wireless Personal Communications, 2016, 89(4): 1435-1442.
- [6] Editorial Note. Editorial Note: multimedia service and applications with mobile cloud computing[J]. Multimedia Tools and Applications, 2017, 76(16): 16873-16881.
- [7] Donald F Ferguson, Víctor Méndez Muñoz. Special issue of cloud computing and services science[J]. Journal of Grid Computing, 2017, 15(2): 139-143.
- [8] Fujun Zhang, Zunsheng Gao, Quanhui Ye. Construction of cloud platform for personalized information services in digital library based on cloud computing data processing technology[J]. Automatic Control and Computer Sciences, 2015, 49(6): 373-379.
- [9] Douglas A Battleeson, Barry C West, Jongwoo Kim, et al. Achieving dynamic capabilities with cloud computing: an empirical investigation[J]. European Journal of Information Systems, 2016, 25(3): 209-230.
- [10] AlAlaa N Tashkandi, Ibrahim M. Al-Jabri. Cloud computing adoption by higher education institutions in saudi arabia: an exploratory study[J]. Cluster Computing, 2015, 18(4): 1527-1537.
- [11] Fawaz Alharbi, Anthony Atkins, Clare Stanier. Understanding the determinants of cloud computing adoption in saudi healthcare organisations[J]. Complex & Intelligent Systems, 2016, 2(3): 155-171.
- [12] Zheng Yan, Xueyun Li, Mingjun Wang, et al. Flexible data access control based on trust and reputation in cloud computing[J]. IEEE Transactions on Cloud Computing, 2017, 5(3): 485-498.
- [13] Jianwei Liao, Francois Trahay, Guoqiang Xiao, et al. Performing initiative data prefetching in distributed file systems for cloud computing[J]. IEEE Transactions on Cloud Computing, 2017, 5(3): 550-562.
- [14] Suyel Namasudra, Pinki Roy. Time saving protocol for data accessing in cloud computing[J]. IET Communications, 2017, 11(10): 1558-1565.
- [15] Hamed Shah-Mansouri, Vincent W S Wong, Robert Schober. Joint optimal pricing and task scheduling in mobile cloud computing systems[J]. IEEE Transactions on Wireless Communications, 2017, 16(8): 5218-5232.

作者简介:

李 斌 男,(1982-),硕士,讲师.研究方向为软件架构与设计、机器学习. E-mail: ljttianjin1980@sina.com.

李启明 男,(1982-),博士,副教授.研究方向为大数据分析、高性能计算.