软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(2):362−380 [doi: 10.13328/j.cnki.jos.005423]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一个支持错误定位的批处理数据拥有性证明方案*

庞晓琼[1], 王田琪[1], 陈文俊[1,2], 任孟琦[1]

[1](中北大学 大数据学院,山西 太原 030051)

[2](中国人民银行 太原中心支行,山西 太原 030001)

通信作者: 庞晓琼, E-mail: xqpang@nuc.edu.cn

摘 要: 数据拥有性证明技术是当前云存储安全领域中的一大重要研究内容,目的是不必下载所有文件,就能安全而高效地远程校验存储在云服务器中的数据是否完整.目前已陆续提出了许多批处理数据拥有性证明方案,但大多数方案都没有考虑用户数据出错后的错误定位问题,仅有的几个批处理校验方案也只能单独定位错误数据所在服务器或其所属用户.提出了利用定位标签辅助第三方审计员快速定位错误的方法,并在 Zhou 等人工作的基础上,利用 Merkle Hash Tree 构造数据定位标签,实现了一个多用户、多服务器环境下支持批处理校验且具备错误数据定位功能的数据拥有性证明方案,可以在批处理校验失败后快速定位错误数据的拥有者和所在服务器.在随机谕言机模型下,该方案是可证明安全的,且性能分析表明,定位错误数据的能力和效率比其他具有单一定位功能的方案更高.

关键词: 错误定位;批处理校验;数据拥有性证明;云存储安全

中图法分类号: TP309

中文引用格式: 庞晓琼,王田琪,陈文俊,任孟琦.一个支持错误定位的批处理数据拥有性证明方案.软件学报,2019,30(2):362−380. http://www.jos.org.cn/1000-9825/5423.htm

英文引用格式: Pang XQ, Wang TQ, Chen WJ, Ren MQ. Batch provable data possession scheme with error locating. Ruan Jian Xue Bao/Journal of Software, 2019,30(2):362−380 (in Chinese). http://www.jos.org.cn/1000-9825/5423.htm

Batch Provable Data Possession Scheme with Error Locating

PANG Xiao-Qiong[1], WANG Tian-Qi[1], CHEN Wen-Jun[1,2], REN Meng-Qi[1]

[1](School of Data Science and Technology, North University of China, Taiyuan 030051, China)

[2](Taiyuan Central Sub-branch, The People's Bank of China, Taiyuan 030001, China)

Abstract: Provable data possession is an important research field in cloud storage security. It allows the data owners remotely checking the integrity of their outsourced data without downloading all files. There have been many batch PDP schemes, but most of them did not consider the error location after the data of users were corrupted. A few batch PDP protocols can identify only the servers in which the corrupted data stored or the clients to which the corrupted data belongs. This study puts forward a method which utilizes location tags to help the third party auditor locating the error data quickly. Based on work by Zhou *et al*., an error locating batch provable data possession scheme is proposed in multi-user and multi-cloud setting by using Merkle Hash tree to create data location tags. The proposed protocol can quickly locate the corrupted data owners and the servers where the error data stored after the batch verification fails. The proposed

* 基金项目: 国家自然科学基金(61379125); 山西省自然科学基金(201601D021075, 201801D121154); 山西省高等学校科技创新项目(2014143); 山西省回国留学人员科研资助项目(2015-083); 山西省研究生教育改革研究项目(2018JG62)

Foundation item: National Natural Science Foundation of China (61379125); Natural Science Foundation of Shanxi Province (201601D021075, 201801D121154); Scientific and Technological Innovation Programs of Higher Education Institutions in Shanxi Province (2014143); Research Project Supported by Shanxi Scholarship Council of China (2015-083); Educational Reform Research Project for Graduate Students of Shanxi Province (2018JG62)

收稿时间: 2017-06-14; 修改时间: 2017-08-07; 采用时间: 2017-09-30; jos 在线出版时间: 2018-04-16

CNKI 网络优先出版: 2018-04-16 10:59:52, http://kns.cnki.net/kcms/detail/11.2560.TP.20180416.1059.007.html

scheme is provably secure in random oracle model, and the performance analysis shows that the scheme has higher error locating ability and efficiency than other schemes that only have single location function.

Key words: error locating; batch verification; provable data possession (PDP); cloud storage security

1 引 言

数据拥有性证明(PDP)方案使得数据拥有者(或校验者)在没有本地备份的情况下,不需要下载数据,就能以很高的概率远程校验用户存储在服务器上的数据是否完整.目前,大多数数据拥有性证明方案是针对单用户存放在单服务器上的数据进行完整性校验[1−11],但是现实的情境中,云存储提供的服务是面向很多用户的,同时,云服务提供商并不是单一的,每个云服务提供商所拥有的也不仅仅是单个服务器.为了更适应现实,近几年,多用户单服务器[12,13]、单用户多服务器[14,15]、多用户多服务器[16−19]情景下的批处理 PDP 方案陆续被提出来.在这些批处理方案中,服务器在计算证明阶段将被挑战数据块的证明按用户或服务器进行聚合计算后,再将聚合证明发送给 TPA(third party auditor)校验,极大地降低了通信开销,且有效地减少了 TPA 校验的计算开销.但是只有当所有用户的数据都是正确存储时,这样的批量处理才能体现出其效率优势,一旦有数据被损毁,批处理校验将不通过,此时定位错误数据所在服务器和所属用户就成为一个亟需解决的问题.最直接的定位错误方法就是对被挑战服务器返回的证明逐一校验,但是这种方法的效率显然不高,并且由于服务器通常返回的是聚合证明,故而目前多用户多服务器环境下的批处理 PDP 方案中,即使采用逐一校验法,TPA 也只能定位错误数据所属用户或所在服务器,无法进行同时定位.因此需要一种有效的方法,在多用户多服务器环境下实现批处理远程数据完整性校验的同时,还能在校验不通过情况下高效地对错误数据定位,即找到错误数据属于哪个用户,且存放在哪个服务器上.

1.1 相关工作

2007 年,Ateniese 等人[1]首次提出了 PDP 的定义,并给出了一个支持公开校验的 PDP 方案.2008 年,Ateniese 等人[2]提出了一个支持动态数据更新的 PDP 方案,但是该方案只支持对文件块的修改、删除和附加操作,并不支持插入,且只能进行有限次校验.2009 年,Wang 等人[3]利用 Merkle 哈希树提出了一个全面支持数据动态更新的方案,同时引入了 TPA 代替用户验证数据的完整性,但该方案没有考虑用户数据隐私会泄露给 TPA 的问题.2010 年,Wang 等人[4]提出一个保护隐私的 PDP 方案,解决用户隐私泄露给 TPA 的问题,但该方案不支持数据动态更新.2011 年,Hao 等人[5]提出了“针对第三方校验者的隐私性”的安全性定义,并构造了一个在此安全性定义下可证明安全的 PDP 方案.2015 年~2016 年,Yu 等人[6−9]重点研究了提高 PDP 方案安全性的问题.2016 年~2017 年,Yu 等人[10,11]提出两种基于身份的 PDP 方案,消除了 PKI 庞大的证书管理开销.以上工作都是针对单用户单服务器环境下的 PDP 方案.

在多用户单服务器环境下,2013 年,Wang 等人[12]利用 BLS 短签名构造同态验证标签,提出了一个保护隐私的批处理 PDP 方案.2014 年,Ren 等人[13]使用椭圆曲线上的 Co-GDH 签名构造同态验证标签,提出一个支持数据动态更新的批处理 PDP 方案.在文献[12,13]中,服务器将被挑战块的证明按用户聚合之后发送给 TPA,TPA 对所有证明聚合计算后进行批量校验.不同的是,文献[12]在批校验不通过时,TPA 会利用二分查找判断哪个用户的数据出错;而文献[13]并未考虑定位错误数据所属用户的问题.

在单用户多服务器环境下,2015 年,Wang[14]提出了一个基于身份的分布式批处理 PDP 方案.2016 年,Mao 等人[15]利用 BLS 短签名提出了一个批处理 PDP 方案.这两个方案中,被挑战服务器将其上所有被挑战数据块的证明聚合计算后发送给 organizer,organizer 将所有被挑战服务器发来的聚合证明再次聚合成一个值,并发送给 TPA 进行校验.文献[14,15]这种交由 organizer 统一聚合证明的方式极大地减轻了 TPA 的计算开销,但也使得 TPA 无法定位错误数据所在服务器.

在多用户多服务器环境下,2014 年,Liu 等人[16]利用双线性对提出一个批处理 PDP 方案,并且使用有序的 Merkle Hash Tree 来抵抗置换攻击.该方案中,被挑战服务器将其上所有被挑战块的证明聚合后发给 organizer,

organizer 将所有被挑战服务器返回的聚合证明再次聚合并发送给 TPA 审计,这种交由 organizer 聚合,再由 TPA 审计的方式在一次挑战响应中无法实现错误数据定位.2016 年,Zhou 等人[17]基于 CDH 假设,利用双线性对提出了一种基于身份的批处理 PDP 方案.该方案中,被挑战服务器将其上所有被挑战块的证明聚合后发送给 TPA,TPA 将所有被挑战服务器返回的证明再次聚合后进行批量校验.虽然文献[17]并未考虑错误数据定位问题,但是在批量审计不通过的情况下,TPA 可以采用最直接的逐一校验方式定位错误数据所在服务器,但无法定位错误数据所属用户.

在多用户多服务器环境下的批处理 PDP 方案中,也曾有学者提出错误数据定位的想法.2013 年,He 等人[18]提出了一个仅能定位出错数据所属用户的批量审计 PDP 方案.该方案中,TPA 批处理校验不通过后,逐一校验 organizer 服务器返回的按用户聚合的证明以实现定位.2015 年,Shin 等人[19]提出了一个仅能定位出错数据所在服务器的批量审计 PDP 方案,并且只能在单个服务器出错的情境下进行定位.该方案中,被挑战服务器为其上属于不同用户的被挑战块计算一个聚合证明,并发送给 TPA 批量审计和错误定位.文献[18,19]不能同时支持定位错误数据所属用户和所在服务器的原因在于:TPA 收到的证明都是经过聚合计算之后得到的聚合值,这种“先聚合再审计”的策略使得 TPA 在审计时无法同时区分这些聚合证明中涉及的数据块所在的服务器和所属的用户,故而在定位错误时只能定位错误数据所属用户或所在服务器.

本文中,我们提出利用定位标签辅助 TPA 快速定位错误的方法,并在 Zhou 等人[17]方案的基础上,在多用户多服务器环境下给出了一个具体实现.方案在支持批处理校验的同时,可以在审计到数据出错后,仅通过比较操作实现错误快速定位,同时找到出错数据的拥有者与其所处的服务器.

1.2 贡 献

(1) 提出了利用定位标签辅助第三方审计员快速定位错误的方法,并给出一个多用户多服务器环境下支持批处理校验和错误数据定位的数据拥有性证明方案框架.云用户对自己的每个文件块计算相应的数据标签,将其和文件块存储在服务器中,同时对存储在不同服务器上的数据计算定位标签并发送给 TPA;TPA 接收到多个云用户的审计请求后,可同时对这些用户存储在多个云服务器上的数据进行挑战;在收到被挑战云服务器返回的证明后,TPA 基于发送的挑战和服务器返回的证明进行有效性批量验证,若验证不通过,则利用定位标签,确定出错数据的所属用户和存储位置.

(2) 给出了一个具体实现.我们的方案是在 Zhou 等人[17]基于身份的批处理 PDP 方案基础上增加了定位标签生成算法,云用户利用 MHT 为其数据块构建定位索引表,并将定位索引表发送给 TPA.在审计阶段,如果批处理校验不通过,TPA 则利用服务器重构的 MHT 树根查找定位索引表以定位错误数据所属用户和所在服务器.我们的方案能够实现仅通过一次比较操作,即可判断出特定用户存放在特定服务器上的数据是否遭到破坏.

(3) 为了防止服务器在某次审计成功后直接存储用户数据的 MHT 树根,并利用其在以后的挑战中构造证明欺骗 TPA,本方案中,每个用户对其存储在不同服务器上的数据分别用不同的参数值构建λ棵 MHT,即对相同数据计算λ个不同的定位标签,在每次挑战时,选用不同的 MHT 参数值,要求服务器计算相应的 MHT 树根.由于服务器无法提前得知每次挑战指定的 MHT 参数,则其成功欺骗 TPA 的概率是可忽略的.

(4) 方案在随机谕言机模型下是可证明安全的.相对于逐一校验定位错误的方式,我们的方案在实现错误数据定位的能力和效率方面均有优势.另一方面,除了可以一次性完成的额外计算外,因定位功能而在审计阶段增加的计算和通信开销都是可以接受的.

1.3 组织结构

本文第 2 节介绍一些相关知识和工具.第 3 节介绍系统模型和本文中所使用的符号,并且提出方案框架和安全性定义.第 4 节是具体方案的构造细节.第 5 节对方案进行安全性分析和性能分析,并提供实验结果.第 6 节进行总结.

2 预备知识

2.1 双线性对

设 q 是一个大素数,群 G_1 和 G_2 是两个阶为 q 的乘法循环群,设 G_1 的生成元为 g,则群 G_1 到 G_2 的一个双线性映射 e:$G_1\times G_1\to G_2$,满足下面的性质.

(1) 双线性:对于任意的 $u,v\in G_1$ 和 $a,b\in Z_q$,有 $e(u^a,v^b)=e(u,v)^{ab}$.

(2) 非退化性:$e(g,g)$的值不等于群 G_2 中的单位元.

(3) 可计算性:对任意的 $u,v\in G_1$,存在一个有效的算法可以计算 $e(u,v)$.

2.2 CDH困难问题

定义 1(CDH 问题). 设 q 是一个大素数,群 G_1 是一个 q 阶乘法循环群,G_1 的生成元为 g,CDH 问题指的是当 a,b 未知,三元组 $(g,g^a,g^b)\in G_1^3$ 已知时,计算 $g^{ab}\in G_1$.设一个概率多项式时间(PPT)算法 A 解决群 G_1 上的 CDH 问题的概率为 $\varepsilon=\Pr[g^{ab}\leftarrow A(g,g^a,g^b)]$.

定义 2. 若对于任意 PPT 算法 A,其解决以上 CDH 问题的概率 ε 是可忽略的,则称群 G_1 上的 CDH 问题是困难的.

2.3 Merkle Hash Tree

Merkle Hash Tree(MHT)是一种二叉树,常被用来进行数据验证.数据值做 Hash 计算后得到的值作为其叶子节点,每个父亲节点的值是由两个子节点的连接值做 Hash 计算后得到的.当一个父亲节点只有 1 个子节点时,父亲节点的值为单个子节点值做 Hash 计算后得到的值.

一般来说,一棵 MHT 的构造方法如图 1 所示.

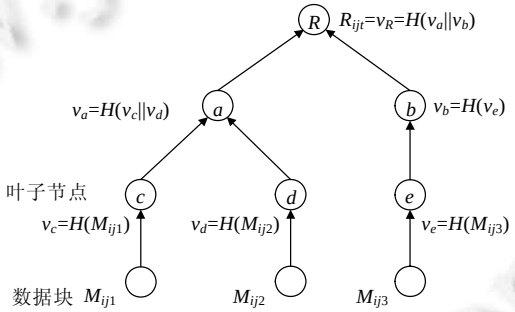


Fig.1 Construction diagram of MHT

图 1 MHT 构造示意图

3 定义与模型

3.1 系统模型与设计目标

一个支持批处理和错误定位的数据完整性验证系统(error locating batch provable data possession,简称 ELBPDP)包含 3 类实体:数据拥有者(即用户,DO)、云服务器(CS)、第三方审计员(即校验者,TPA).

- 数据拥有者:在支持批处理和错误定位的数据完整性验证系统中,有多个数据拥有者.每个数据拥有者拥有各自的文档数据,他们将文档数据预处理后对所有数据块计算数据标签,将数据块和数据标签外包给云服务提供商;对存储在各个服务器上的数据计算定位标签,并发送给第三方审计员.
- 云服务器:在本系统中,云服务器的数量也有多个.每个云服务器存储着数据拥有者们提交的文件块及其数据标签.在接收到第三方审计员发送的校验挑战后,云服务器计算并返回证明.

- 第三方审计员:收到数据拥有者的审计请求后,第三方审计员给各个云服务器发送挑战,并对被挑战服务器返回的证明进行批处理校验.如果校验未通过,则根据定位标签继续确定错误数据块所属的用户和所在服务器,最后将校验结果发送给相应用户.

支持批处理和错误定位的数据完整性验证系统的结构如图 2 所示.

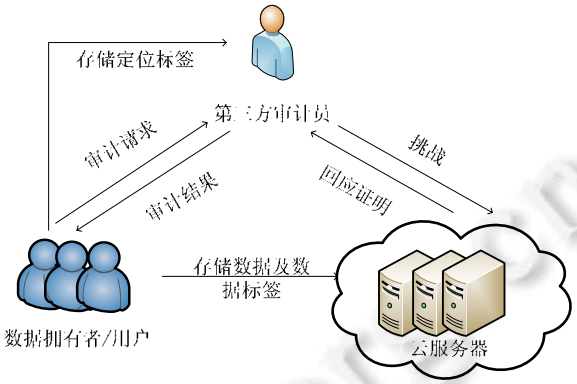


Fig.2 System model diagram

图 2 系统模型图

支持错误定位的批处理 PDP 方案,应满足如下要求.

(1) 支持批处理校验:校验者可以一次性批量校验多个用户存储在多个服务器上的数据是否完整.

(2) 支持错误定位:在批处理校验失败后,校验者可以同时找出错误数据所属用户与所存储的服务器.

(3) 公开审计性:允许第三方审计员验证存储在云服务器上的用户数据的完整性,但是不需要从云服务器上下载全部的数据.

(4) 存储完整性:确保云服务器只有真实存储用户的完整数据才能通过校验者的验证.

3.2 符号定义

方案中涉及到的符号含义由表 1 给出.

Table 1 Table of symbols and notions

表 1 符号表

符号	描述	符号	描述
mpk	PKG 使用的主公钥	msk	PKG 使用的主私钥
DO_i	第 i 个用户	CS_j	第 j 个服务器
ID_i	DO_i 的 ID	sk_i	DO_i 的私钥
pk_i	DO_i 的公钥	F_{ijkl}	DO_i 存放在 CS_j 上的第 k 个块里的第 l 个分区
s	分区数	M_{ijk}	DO_i 存放在 CS_j 上的第 k 个块,由 s 个分区构成
m_{ijk}	$\{F_{ijkl}\}$的线性组合	σ_{ijk}	M_{ijk} 的数据标签
c	TPA 选择的被挑战数据块的块数	O	被校验者选中的用户的索引组成的集合
U	被校验者选中的服务器的索引组成的集合	$chal$	TPA 选取的总挑战
$chal_j$	分发给 CS_j 的挑战	P_j	CS_j 返回的证明
J_i	存储用户 DO_i 数据的服务器索引集合	$N_{i,j}$	DO_i 存放在 CS_j 上的文件块块数
TR_{ijt}	DO_i 存放在 CS_j 上的数据构成的第 t 棵 MHT	R_{ijt}	DO_i 存放在 CS_j 上的数据构成的第 t 棵 MHT 的根节点
a_{it}	DO_i 的第 t 棵 MHT 参数	chr_{ij}	DO_i 存放在 CS_j 上所有数据的定位标签
chr_{ijt}	DO_i 存放在 CS_j 上所有数据的第 t 个定位标签	$Index_i$	用户 DO_i 构建的定位索引表
w_i	每个用户索引 i 的比特长度	w_j	每个服务器索引 j 的比特长度
w_k	每个文件块索引 k 的比特长度	γ	所有用户的数量
η	所有服务器的数量	λ	每个用户对其存放在一个服务器上的数据所构建的 MHT 数量
$x \xleftarrow{R} Z_q$	从集合 Z_q 中随机均匀地选取 x	–	–

3.3 方案框架

定义 3. 一个支持批处理和错误定位的公开可验证的数据拥有性证明方案 *ELBPDP*=(*SetUp*,*Extract*,*TagGen*,*PosTagGen*,*Challenge*,*Prove*,*Verify*)包括以下 7 种算法.

1) *Setup*(1^k)→(*params*,*mpk*,*msk*).以安全参数 *k* 为输入,PKG(private key generator)输出公共参数 *params*,主密钥对(*mpk*,*msk*),*msk* 仅为 PKG 所知.
2) *Extract*(*params*,*msk*,ID_i)→(sk_i,pk_i).以公共参数 *params*、主私钥 *msk* 和用户身份 ID_i 为输入,PKG 输出用户相应的私钥 sk_i 和公钥 pk_i.
3) *TagGen*(*params*,ID_i,sk_i,*mpk*,{M_{ijk}})→({σ_{ijk}}).以公共参数 *params*、用户身份 ID_i 和私钥 sk_i、主公钥 *mpk*、文件数据块的集合{M_{ijk}}为输入,用户 DO_i 对每个块计算数据标签σ_{ijk},并输出数据标签集合{σ_{ijk}},然后将{M_{ijk}}和{σ_{ijk}}存储到相应的服务器端.服务器收到数据块和数据标签后校验其可用性.
4) *PosTagGen*(*params*,*mpk*,{M_{ijk}})→{chr_{ij}}.以公共参数 *params*、主公钥 *mpk*、文件数据块的集合{M_{ijk}}为输入,用户 DO_i 对其存储在服务器 CS_j 上的数据块计算定位标签 chr_{ij},并将所有的定位标签{chr_{ij}}发送给校验者 TPA.
5) *Challenge*({*i*,*j*,*k*})→(*chal*,{$chal_j$}).以所有文件块的索引集为输入,TPA 输出总挑战 *chal*,并将 *chal* 按服务器划分成若干分挑战{$chal_j$},然后将每个 $chal_j$ 发送给相应的服务器 CS_j.
6) *Prove*(*params*,$chal_j$,{ID_i},{σ_{ijk}},{M_{ijk}})→P_j.收到挑战的服务器 CS_j 以公共参数 *params*、挑战 $chal_j$、用户身份集合{ID_i}、数据标签集合{σ_{ijk}}和 CS_j 上所存储的数据块集合{M_{ijk}}为输入,计算证明 P_j,并将 P_j 发送给校验者 TPA.
7) *Verify*(*params*,*chal*,{ID_i},{P_j},*mpk*,{chr_{ijt}})→{1,{(*i*,*j*)}}.以公共参数 *params*、总挑战 *chal*、用户身份集合{ID_i}、所有被校验的服务器返回的证明{P_j}、主公钥 *mpk*、定位标签集{chr_{ijt}}为输入,TPA 批处理校验证明集{P_j}的有效性,以确定各个被校验服务器中的数据保存是否完整.若{P_j}有效,则输出"1";否则,定位所有出错数据所属用户和所在服务器,并输出其索引对{(*i*,*j*)}.

3.4 安全定义

敌手模型:在我们的方案中,考虑校验者即 TPA 是诚实且好奇的,它总会按照协议规定执行,因为作为第三方审计,一旦被用户得知会与服务器勾结,对其声誉有极大的损害,但 TPA 也有可能对用户的数据有一定好奇心.而云服务器为了更高的利益,可能会删除用户不常用的数据,而且在丢失或损毁用户数据后,会伪造数据标签或者证明企图通过校验,欺骗校验者和用户.综上,我们对敌手模型的设定有其合理性.

定义 4. 伪造数据标签游戏 *Dtag-forge*$_A$(*k*).

1. *Setup*:挑战者 *C* 运行 *Setup*(1^k),得到(*params*,*mpk*,*msk*),将(*params*,*mpk*)发送给敌手 *A*,*msk* 秘密保存.
2. *Query*:敌手 *A* 适应性的对 *C* 做 *Extract Query* 和 *TagGen Query* 两种询问:
   (1) *Extract Query*:敌手 *A* 询问 ID_i 的私钥.*C* 运行算法 *Extract*(*params*,*msk*,ID_i),得到私钥 sk_i,并将其发送给 *A*.*C* 设置一个集合 S_1={ID_i},存放被查询过的用户身份.
   (2) *TagGen Query*:敌手 *A* 询问文件块 M_{ijk} 的数据标签,*C* 运行算法 *TagGen*(*params*,ID_i,sk_i,*mpk*,{M_{ijk}})得到数据标签σ_{ijk},并将其发送给 *A*.*C* 设置一个集合 I_1={(*i*,*j*,*k*,M_{ijk})},存放被查询过数据标签的文件块.
3. *Forge*:敌手 *A* 对一个身份为 ID_{i^*} 的用户所拥有的文件块 $M_{i^*j^*k^*}$,伪造出一个数据标签 $\sigma_{i^*j^*k^*}$,且
$$ID_{i^*} \notin S_1,(i^*,j^*,k^*,M_{i^*j^*k^*}) \notin I_1.$$
4. 如果敌手 *A* 输出的数据标签是有效的,则游戏输出为 1;否则输出为 0.如果 *Dtag-forge*$_A$(*k*)=1,则称敌手 *A* 赢得了这次伪造数据标签游戏.

定义 5. 对于任意一个 PPT 敌手 *A*,如果在一个文件块集上存在一个可忽略的函数 *negl*,使得:
$$\Pr[Dtag\text{-}forge_A(k)=1] \leqslant negl(k),$$

则称在这个文件块集上的数据标签是不可伪造的.

定义 6. 伪造数据标签证明游戏 $DtagProof\text{-}forge_A(k)$.

1. *Setup*:与伪造数据标签游戏的 *Setup* 阶段相同.
2. *Query*1:与伪造数据标签游戏的 *Query* 阶段相同.
3. *Challenge*:挑战者 *C* 生成一个包含 c^*组数据 $\{(i_n^*, j_n^*, k_n^*, M_{i_n^* j_n^* k_n^*}) \mid n=1,...,c^*\}$ 的挑战 $chal^*$,其中至少有 1 个 i_n^*, 使得 $ID_{i_n^*} \notin S_1$, 且 $(i_n^*, j_n^*, k_n^*, M_{i_n^* j_n^* k_n^*}) \notin I_1$. 挑战者 *C* 将挑战 $chal^*$发送给 *A*.
4. *Query*2:与伪造数据标签游戏的 *Query* 阶段相似,设置一个集合 $S_2=\{ID_i\}$存放本阶段被查询过的用户身份,设置一个集合 $I_2=\{(i,j,k,M_{ijk})\}$存放本阶段被查询过数据标签的文件块.挑战 $chal^*$中至少存在 1 个 i_n^*,使得 $ID_{i_n^*} \notin (S_1 \cup S_2)$, 且 $(i_n^*, j_n^*, k_n^*, M_{i_n^* j_n^* k_n^*}) \notin (I_1 \cup I_2)$.
5. *Forge*:敌手 *A* 针对挑战 $chal^*$,伪造出一个证明 $P^*=\{P_{j^*}\}$.
6. 如果敌手 *A* 输出的数据标签证明是有效的,则游戏输出为 1;否则输出为 0.如果 $DtagProof\text{-}forge_A(k)=1$, 则称敌手 *A* 赢得了这次伪造数据标签证明游戏.

定义 7. 对于任意一个 PPT 敌手 *A*,如果在一个文件块集上存在一个可忽略的函数 *negl*,使得:

$$\Pr[DtagProof\text{-}forge_A(k)=1] \leqslant negl(k),$$

则称在这个文件块集上的数据标签证明是不可伪造的.

4 具体方案

方案的具体细节如下.

1) $Setup(1^k) \rightarrow (params,mpk,msk)$:输入一个安全参数 *k*,PKG 执行以下操作.

PKG 选择两个阶为 *q* 的乘法循环群 G_1 和 G_2,*q* 是一个大素数且满足 $q>2^k$,取 G_1 的生成元为 *g*,在群 G_1 和 G_2 上选择一个双线性映射 $e:G_1 \times G_1 \rightarrow G_2$.

PKG 选择 4 个密码学 Hash 函数 H_1、H_2、H_3、H_4 和一个伪随机函数 *f*,其中,$H_1:\{0,1\}^* \rightarrow G_1$, $H_2:\{0,1\}^* \rightarrow Z_q$, $H_3:\{0,1\}^* \rightarrow G_1$, $H_4:\{0,1\}^* \rightarrow Z_q$($H_1$ 和 H_3、H_2 和 H_4 分别是不同的 Hash 函数), $f: Z_q \times \{0,1\}^{w_i+w_j+w_k} \rightarrow Z_q$.

PKG 随机选择 $\{v_l \xleftarrow{R} Z_q \mid l=1,...,s\}$ 作为分区系数,并令 λ 作为每个用户对相同数据构建 MHT 的数量.

PKG 随机选择 $x \xleftarrow{R} Z_q^*$ 作为主私钥 *msk*,并令主公钥为 $mpk=g^x$.

将公共参数 $params=(G_1,G_2,q,g,e,H_1,H_2,H_3,H_4,f,\{v_l\},\lambda)$和主公钥 $mpk=g^x$ 公开,将主私钥 $msk=x$ 秘密保存.

2) $Extract(params,msk,ID_i) \rightarrow (sk_i,pk_i)$:PKG 计算用户 DO_i 的公私钥,其中,公钥 $pk_i=H_1(ID_i)$,私钥 $sk_i=H_1(ID_i)^x=pk_i^x$,并用一个安全通道将 sk_i 发送给相应的用户 DO_i.

3) $TagGen(params,ID_i,sk_i,mpk,\{M_{ijk}\}) \rightarrow (\{\sigma_{ijk}\})$:用户 DO_i 执行以下操作.

DO_i 随机选取 $u_i \xleftarrow{R} Z_q$, 对自己的每个文件块 M_{ijk} 计算 $m_{ijk}=\sum_{l=1}^{s} v_l \cdot F_{ijkl}$, $h_i=H_2(ID_i)$, $hpk=H_3(mpk)$, 并计算 $S_{ijk}=g^{u_i}$, $T_{ijk}=sk_i^{m_{ijk}+h_i} \cdot hpk^{u_i}$. 令文件块 M_{ijk} 的数据标签为 $\sigma_{ijk}=(S_{ijk},T_{ijk})$.

DO_i 将其所有的文件块$\{M_{ijk}\}$和相应的数据标签$\{\sigma_{ijk}\}$按服务器索引发送给相应的服务器.每个服务器收到用户发送的数据块和数据标签后,通过校验下面的等式是否成立来确定数据标签是否可用:

$$e(T_{ijk},g) \stackrel{?}{=} e\left(H_1(ID_i)^{\sum_{l=1}^{s} v_l \cdot F_{ijkl}+H_2(ID_i)}, mpk\right) \cdot e(H_3(mpk),S_{ijk}) \tag{1}$$

4) $PosTagGen(params,mpk,\{M_{ijk}\}) \rightarrow \{a_{it},chr_{ijt}\}$.

用户 DO_i 随机选择 $\{a_{it} \xleftarrow{R} Z_q \mid t=1,...,\lambda\}$.

设存储 DO_i 数据的服务器索引集合为 J_i,且 DO_i 在服务器 $CS_j(j \in J_i)$上存储的文件块块数为 N_{ij}.用户 DO_i 对每一个服务器 $CS_j(j \in J_i)$,分别以 $a_{it}(1 \leqslant t \leqslant \lambda)$为 MHT 参数,对其存储在 CS_j 上的 N_{ij} 个数据块构建 λ 棵 MHT.每棵

树用 $TR_{ijt}(1\leqslant t\leqslant\lambda)$表示,$TR_{ijt}$的根节点用 R_{ijt} 表示.

例如,用户 DO_1 在服务器 CS_1 上共存放了 4 个数据块 M_{111}、M_{112}、M_{113}、M_{114},使用 $a_{1t}(1\leqslant t\leqslant\lambda)$作为参数,$TR_{11t}$的构建如图 3 所示,树的根为 R_{11t}.

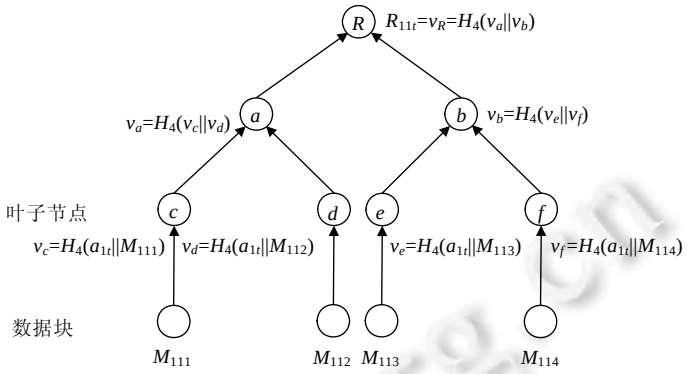


Fig.3 MHT TR_{11t} constructed by DO_1 with the data block stored in CS_1 and parameter a_{1t}

图 3 DO_1 以 a_{1t} 为参数,针对 CS_1 上的数据块构建的 MHT TR_{11t}

DO_i 令 $chr_{ijt}=R_{ijt}(j\in J_i,1\leqslant t\leqslant\lambda)$.

设服务器共有 η 个.DO_i 构建一张定位索引表 $Index_i=\{a_{it},chr_{i1t},chr_{i2t},...,chr_{i\eta t}\}_{t=1}^{\lambda}$,其中,若 chr_{ijt} 不存在,即 $j\notin J_i$,则令 $chr_{ijt}=-1$.DO_i 将定位索引表发送给校验者.定位索引表见表 2.

Table 2 Table of locating indexes constructed by DO_i

表 2 用户 DO_i 构建的定位索引表

a_{i1}	chr_{i11}	chr_{i21}	…	$chr_{i\eta1}$
a_{i2}	chr_{i12}	chr_{i22}	…	$chr_{i\eta2}$
…	…	…	…	…
$a_{i\lambda}$	$chr_{i1\lambda}$	$chr_{i2\lambda}$	…	$chr_{i\eta\lambda}$

5) $Challenge(\{(i,j,k)\},\{a_{it}\},\lambda)\rightarrow(chal,\{chal_j\})$.校验者执行以下操作.

校验者接收用户 DO_i 发来的审计请求,请求为 DO_i 所有数据块的索引集$\{(i,j,k)\}$,包括用户索引 i、存储 DO_i 数据的服务器 CS_j 索引 $j\in J_i$、存放在 CS_j 上的块索引 k.收到多个云用户的审计请求后,TPA 将所有审计请求做并集,得到总的审计请求集合 $Q=\bigcup\{(i,j,k)\}$.

校验者随机选取 MHT 的参数索引 $\tau\xleftarrow{R}[1,\lambda]$.

校验者从总的审计请求集合 Q 中选出 c 个块进行校验,令 $M_{i_nj_nk_n}(1\leqslant n\leqslant c,(i_n,j_n,k_n)\in Q)$ 表示被选中的 c 个块,构建索引集合 $I=\{(i_n,j_n,k_n)|n=1,\ldots,c\}$.

校验者构建映射 $f_1:I\rightarrow Z_q$, $f_1(i_n,j_n,k_n)=\kappa_{i_n}$,满足$\forall s,t\in[1,c]$,当 $i_s=i_t$ 时,有 $\kappa_{i_s}=\kappa_{i_t}$,令集合 $K=\{\kappa_{i_1},\kappa_{i_2},...,\kappa_{i_c}\}$.

校验者构建映射 $f_2:I\times\{\tau\}\rightarrow\{a_{i_1\tau},...,a_{i_c\tau}\}$, $f_2(i_n,j_n,k_n,\tau)=a_{i_n\tau}$,满足$\forall s,t\in[1,c]$,当 $i_s=i_t$ 时,有 $a_{i_s\tau}=a_{i_t\tau}$,令 MHT 参数集合 $\alpha=\{a_{i_1\tau},a_{i_2\tau},...,a_{i_c\tau}\}$.

校验者令总挑战 $chal=(I,K,\alpha)$.

设被挑战的数据块所在服务器的索引集合$\{j\}$用 U 表示,校验者将挑战 $chal$ 按被挑战服务器的不同,划分成 $|U|$个分挑战$\{chal_j\}$,有 $chal=\bigcup_{j\in U}chal_j$. 每个 $chal_j=(I_j,K_j,\alpha_j)$,其中,$I_j=\{(i_n,j_n,k_n)|(i_n,j_n,k_n)\in I$ 并且 $j_n=n\}$, $K_j=\{\kappa_{i_n}=f_1(i_n,j_n,k_n)|(i_n,j_n,k_n)\in I_j\}$, $\alpha_j=\{a_{i_n\tau}=f_2(i_n,j_n,k_n,\tau)|(i_n,j_n,k_n)\in I_j\}$.

校验者将 $chal_j$ 发送给服务器 CS_j.

6) $Prove(params,chal_j,\{ID_i\},\{\sigma_{ijk}\},\{M_{ijk}\})\rightarrow P_j$.

Prove-DataTag:收到挑战 $chal_j$ 的服务器 CS_j 计算 $\{r_n\}=\{f_{\kappa_{i_n}}(i_n\|j_n\|k_n)|(i_n,j_n,k_n)\in I_j\}$(此处$\{r_n\}$表示服务器 CS_j 对其收到挑战 $chal_j$ 中所有块分别计算的伪随机函数值构成的集合),对 $chal_j$ 中涉及的每一个用户,计算 $F'_{ijl}=\sum_{chal_j,i_n=i}F_{i_njk_nl}\cdot r_n$,得到集合$\{F'_{ijl}\,|\,i\in O_j,l=1,...,s\}$,其中,$O_j$表示 I_j 中包含的所有云用户的索引集合.然后,CS_j利用标签$\{\sigma_{i_njk_n}\}$计算 $S'_j=\prod_{chal_j}S^{r_n}_{i_njk_n}$,$T'_j=\prod_{chal_j}T^{r_n}_{i_njk_n}$.

Prove-PositionTag:服务器 CS_j 针对每个被挑战的用户 $DO_i(i\in O_j)$,对存储在其上的所有数据块 M_{ijk},以 α_j 中与 DO_i 的数据块索引对应的 $a_{i\tau}$ 为参数,按照如图 3 所示的方法重构相应的 MHT,表示为 $TR_{ij\tau}$,其树根为 $R_{ij\tau}$,所有 O_j 中云用户的数据块构建的 MHT 树根和与其对应的用户、服务器索引构成集合$\{(i,j,R_{ij\tau})|i\in O_j\}$.

令证明 $P_j=(S'_j,T'_j,\{F'_{ijl}\,|\,i\in O_j,l=1,...,s\},\{(i,j,R_{ij\tau})\,|\,i\in O_j\})$,服务器将证明发送给校验者.

7) *Verify*$(params,chal,\tau,\{ID_i\},\{P_j\},mpk,\{chr_{ijt}\})\rightarrow\{1,\{(i,j)\}\}$.

设校验者生成的总挑战中所涉及的用户 DO_i 的索引集合$\{i\}$用 O 表示.校验者收到所有被挑战服务器发回的证明后,先计算$\{r_n\}=\{f_{\kappa_{i_n}}(i_n\|j_n\|k_n)|(i_n,j_n,k_n)\in I\}$(此处$\{r_n\}$表示 TPA 对总挑战中所有被挑战块分别计算的伪随机函数值构成的集合),然后校验等式(2)是否成立:

$$e\left(\prod_{j\in U}T'_j,g\right)\overset{?}{=}e\left(\prod_{i\in O}H_1(ID_i)^{\sum_{j\in U}\sum_{l=1}^{s}\nu_l\cdot F'_{ijl}+H_2(ID_i)\cdot\sum_{i_n=i}r_n},mpk\right)\cdot e\left(H_3(mpk),\prod_{j\in U}S'_j\right)\tag{2}$$

- 若等式(2)成立,则说明批处理校验通过,即总挑战中涉及的云用户的数据审计结果为验证通过,校验者输出 1;
- 若等式(2)不成立,则对云服务器 $CS_j(j\in U)$返回的集合$\{(i,j,R_{ij\tau})|i\in O_j\}$中的每个元素$(i,j,R_{ij\tau})$,TPA 利用$(i,j)$和$\tau$,查询定位索引表 $Index_i$ 中第τ行、第 j+1 列中的值 $chr_{ij\tau}$,并校验等式(3)是否成立:

$$chr_{ij\tau}\overset{?}{=}R_{ij\tau}\tag{3}$$

若不成立,则输出相应的(i,j).

5 正确性、安全性及性能分析

5.1 正确性证明

定理 1. 若校验者和服务器都是诚实的,那么服务器返回的关于数据标签的证明就可以通过 TPA 的批处理校验.

证明:等式(2)的右边可以进行如下变换:

$$e\left(\prod_{i\in O}H_1(ID_i)^{\sum_{j\in U}\sum_{l=1}^{s}\nu_l\cdot F'_{ijl}+H_2(ID_i)\cdot\sum_{i_n=i}r_n},mpk\right)=e\left(\prod_{j\in U}\prod_{chal_j}H_1(ID_{i_n})^{m_{i_njk_n}\cdot r_n+H_2(ID_{i_n})\cdot r_n},mpk\right)=e\left(\prod_{j\in U}\prod_{chal_j}sk_{i_n}^{m_{i_njk_n}\cdot r_n+h_{i_n}\cdot r_n},g\right),$$

$$e\left(H_3(mpk),\prod_{j\in U}S'_j\right)=e\left(hpk,\prod_{j\in U}\prod_{chal_j}S^{r_n}_{i_njk_n}\right)=e\left(\prod_{j\in U}\prod_{chal_j}hpk^{u_{i_n}\cdot r_n},g\right),$$

则等式(2)的右边就等于:

$$e\left(\prod_{j\in U}\prod_{chal_j}sk_{i_n}^{m_{i_njk_n}\cdot r_n+h_{i_n}\cdot r_n},g\right)\cdot e\left(\prod_{j\in U}\prod_{chal_j}hpk^{u_{i_n}\cdot r_n},g\right)=e\left(\prod_{j\in U}\prod_{chal_j}T^{r_n}_{i_njk_n},g\right)=e\left(\prod_{j\in U}T'_j,g\right).$$

故等式(2)成立.证毕. □

5.2 安全性分析

定理 2[17]. 如果 CDH 问题是困难的,则在随机谕言机模型下,不存在一个 PPT 敌手能够以不可忽略的概率

赢得伪造数据标签游戏.

证明:令 B 的输入为 $(g,g^x,g^y)\in G_1^3$,其中,G_1 为 q 阶乘法循环群,g 为生成元.B 作为挑战者,想要通过与敌手 A 进行交互得到 $g^{xy}\in G_1$,模拟过程如下.

1. *Setup*:B 选择一个 q 阶乘法循环群 G_2,在群 G_1 和 G_2 上选择一个双线性映射 $e:G_1\times G_1\to G_2$,随机选择 $\{v_l\xleftarrow{R}Z_q\,|\,l=1,...,s\}$,设置 $mpk=g^x$,将$(G_1,G_2,q,g,e,mpk,\{v_l\})$发送给敌手 A.

2. H_1-*Query*:A 可以随时查询随机谕言机 H_1,B 需要构建维护一张 H_1 列表 H_1-*list*=$\{(ID_i,b_i,a_i,H_1(ID_i))\}$来存储对 A 的回应.当 A 对 ID_i 查询 H_1,B 回应方法如下:

1) 若 H_1-*list* 中有包含元素 ID_i 的元组,则 B 读取四元组$(ID_i,b_i,a_i,H_1(ID_i))$,并将 $H_1(ID_i)$发送给 A.

2) 若 H_1-*list* 中没有包含元素 ID_i 的元组,则 B 根据 b_i 的值对 A 进行回应,其中,b_i 由 B 通过二项分布 $\Pr[b_i=0]=\delta,\Pr[b_i=1]=1-\delta$确定.

a) 若 $b_i=0$,B 随机选择一个 $a_i\in Z_q^*$,计算 $H_1(ID_i)=g^{a_i}$;

b) 若 $b_i=1$,B 随机选择一个 $a_i\in Z_q^*$,计算 $H_1(ID_i)=g^{ya_i}$.

B 将$(ID_i,b_i,a_i,H_1(ID_i))$加入表 H_1-*list* 中,并将 $H_1(ID_i)$发送给 A.

3. H_2-*Query*:A 可以随时查询随机谕言机 H_2,B 需要构建维护一张 H_2 列表 H_2-*list*=$\{(ID_i,H_2(ID_i))\}$来存储对 A 的回应.当 A 对 ID_i 查询 H_2,B 回应方法如下.

1) 若 H_2-*list* 中有包含元素 ID_i 的元组,则 B 读取二元组$(ID_i,H_2(ID_i))$,并将 $H_2(ID_i)$发送给 A.

2) 若 H_2-*list* 中没有包含元素 ID_i 的元组,则 B 随机选择一个 $H_2(ID_i)$发送给 A,然后将二元组$(ID_i,H_2(ID_i))$加入表 H_2-*list* 中.

4. H_3-*Query*:A 可以随时查询随机谕言机 H_3.B 需要构建维护一张 H_3 列表 H_3-*list*=$\{(mpk_i,z_i,H_3(mpk_i))\}$来存储对 A 的回应.当 A 对 mpk_i 查询 H_3,B 回应方法如下.

1) 若 H_3-*list* 中有包含元素 mpk_i 的元组,则 B 读取三元组$(mpk_i,z_i,H_3(mpk_i))$,并将 $H_3(mpk_i)$发送给 A.

2) 若 H_3-*list* 中没有包含元素 mpk_i 的元组,则 B 随机选择 $z_i\in Z_q^*$,计算 $H_3(mpk_i)=g^{z_i}$,并将 $H_3(mpk_i)$发送给 A.然后,B 将三元组$(mpk_i,z_i,H_3(mpk_i))$加入表 H_3-*list* 中.特别地,当 $mpk_i=mpk$ 时,B 将相应的元组记为$(mpk,z,H_3(mpk)=g^z)$.

5. *Extract Query*:A 可以随时查询随机谕言机 *Extract*.B 需要构建维护一张表 *Extract-list*=$\{(ID_i,sk_i)\}$来存储对 A 的回应.当 A 对 ID_i 查询谕言机 *Extract*,B 回应方法如下.

1) 若 *Extract-list* 中有包含元素 ID_i 的元组,则 B 读取二元组(ID_i,sk_i),并将 sk_i 发送给 A。

2) 若 *Extract-list* 中没有包含元素 ID_i 的元组,则 B 查找 H_1-*list* 中是否有包含元素 ID_i 的元组,若没有,则 B 生成一个 $H_1(ID_i)$的查询,使得四元组$(ID_i,b_i,a_i,H_1(ID_i))$存在.B 根据 b_i 的值对 A 进行回应.

a) 若 $b_i=0$,则 B 计算 $sk_i=(g^x)^{a_i}$,并将 sk_i 发送给 A.然后,B 将(ID_i,sk_i)加入表 *Extract-list* 中;

b) 若 $b_i=1$,则 B 报告失败,并终止模拟.

6. *TagGen Query*:A 可以随时查询随机谕言机 *TagGen*.B 需要构建维护一张表 *TagGen-list*=$\{(i,j,k,M_{ijk},\sigma_{ijk})\}$来存储对 A 的回应.当 A 对(i,j,k,M_{ijk})查询谕言机 *TagGen*,B 回应方法如下.

1) 若 *TagGen-list* 中有包含元素(i,j,k,M_{ijk})的元组,则 B 读取五元组$(i,j,k,M_{ijk},\sigma_{ijk})$,并将$\sigma_{ijk}$ 发给 A.

2) 若 *TagGen-list* 中没有包含元素(i,j,k,M_{ijk})的元组,则 B 查找 H_1-*list* 中是否有包含元素 ID_i 的元组,若没有,则 B 生成一个 $H_1(ID_i)$的查询.然后,B 查找 H_2-*list* 中是否有包含元素 ID_i 的元组,若没有,则 B 生成一个 $H_2(ID_i)$的查询.然后,B 查找 H_3-*list* 中是否有包含元素 *mpk* 的元组,若没有,则 B 生成一个 $H_3(mpk)$的查询.B 根据 b_i 的值对 A 进行回应.

a) 若 $b_i=0$,则 B 随机选择 $S_{ijk}\in G_1$,计算:

$$\sigma_{ijk}=\left(S_{ijk},(g^{x})^{a_i\left(\sum_{l=1}^{s}v_l F_{ijkl}+H_2(ID_i)\right)}S_{ijk}^{z}\right),$$

并将σ_{ijk}发送给*A*.然后,*B*将五元组$(i,j,k,M_{ijk},\sigma_{ijk})$加入表*TagGen-list*中.可以验证,$(i,j,k,M_{ijk},\sigma_{ijk})$满足等式(1),即$\sigma_{ijk}$为数据块$(i,j,k,M_{ijk})$的有效标签.

b) 若b_i=1,则*B*报告失败,并终止模拟.

7. *Forge*:敌手*A*对一个身份为ID_{i^*}的用户所拥有的文件块$M_{i^*j^*k^*}$伪造数据标签$\sigma_{i^*j^*k^*}=(S_{i^*j^*k^*},T_{i^*j^*k^*})$,要求敌手*A*之前并未对$ID_{i^*}$进行过*Extract Query*,且未对$(i^*,j^*,k^*,M_{i^*j^*k^*})$进行过*TagGen Query*.*B*查找$H_1$-*list*中是否有包含元素$ID_{i^*}$的元组,若没有,则*B*生成一个$H_1(ID_{i^*})$的查询.然后,*B*查找$H_2$-*list*中是否有包含元素$ID_{i^*}$的元组,若没有,则*B*生成一个$H_2(ID_{i^*})$的查询.然后,*B*查找$H_3$-*list*中是否有包含元素*mpk*的元组,若没有,则*B*生成一个$H_3(mpk)$的查询.*B*根据b_i的值做如下操作.

1) 若$b_{i^*}=0$,则*B*报告失败,并终止模拟.

2) 若$b_{i^*}=1$,且$(i^*,j^*,k^*,M_{i^*j^*k^*},\sigma_{i^*j^*k^*})$满足等式(1),即伪造标签有效,有:

$$e(T_{i^*j^*k^*},g)=e\left(H_1(ID_{i^*})^{\sum_{l=1}^{s}v_l\cdot F_{i^*j^*k^*l}+H_2(ID_{i^*})},mpk\right)\cdot e(H_3(mpk),S_{i^*j^*k^*}).$$

*B*可得到:

$$e(T_{i^*j^*k^*},g)=e\left((g^{ya_{i^*}})^{\sum_{l=1}^{s}v_l\cdot F_{i^*j^*k^*l}+H_2(ID_{i^*})},g^{x}\right)\cdot e(g^{z},S_{i^*j^*k^*})=e\left((g^{xy})^{a_{i^*}\left(\sum_{l=1}^{s}v_l\cdot F_{i^*j^*k^*l}+H_2(ID_{i^*})\right)}S_{i^*j^*k^*}^{z},g\right).$$

*B*计算$g^{xy}=(T_{i^*j^*k^*}S_{i^*j^*k^*}^{-z})^{a_{i^*}^{-1}\cdot\left(\sum_{l=1}^{s}v_l\cdot F_{i^*j^*k^*l}+H_2(ID_{i^*})\right)^{-1}}$.

令*E*表示事件*B*求解出g^{xy},经过上面分析可知:在*A*进行n_{EQ}次*Extract Query*和n_{TQ}次*TagGen Query*时模拟未终止,且*A*对数据块$M_{i^*j^*k^*}$成功地伪造出有效的数据标签;同时,在对ID_{i^*}进行H_1-*Query*时,若元组中$b_{i^*}=1$,则事件*E*发生.若敌手*A*赢得伪造数据标签游戏的概率ε不可忽略,则$\Pr[E]=\delta^{n_{EQ}+n_{TQ}}\varepsilon(1-\delta)$不可忽略,即*B*解决群$G_1$上的CDH问题的概率不可忽略.当$\delta=(n_{EQ}+n_{TQ})/(n_{EQ}+n_{TQ}+1)$时,$\Pr[E]=\frac{(n_{EQ}+n_{TQ})^{n_{EQ}+n_{TQ}}}{(n_{EQ}+n_{TQ}+1)^{n_{EQ}+n_{TQ}+1}}\varepsilon$取得最大值.

证毕. □

定理 3[17]. 如果CDH问题是困难的,则在随机谕言机模型下,不存在一个PPT敌手能够以不可忽略的概率赢得伪造数据标签证明游戏.

证明:令*B*的输入为$(g,g^{x},g^{y})\in G_1^3$,其中,G_1为*q*阶乘法循环群,*g*为生成元.*B*作为挑战者,想要通过和敌手*A*进行交互得到$g^{xy}\in G_1$,模拟过程如下.

1. *Setup*:*B*选择一个*q*阶乘法循环群G_2,在群G_1和G_2上选择一个双线性映射$e:G_1\times G_1\to G_2$,随机选择$\{v_l\xleftarrow{R}Z_q\mid l=1,...,s\}$,选取伪随机函数$f:Z_q\times\{0,1\}^{w_i+w_j+w_k}\to Z_q$,设置$mpk=g^{x}$.将$(G_1,G_2,q,g,e,f,mpk,\{v_l\})$发送给敌手*A*.

2. *B*对H_1-*Query*、H_2-*Query*、H_3-*Query*、第1阶段*Extract*-1 *Query*、*TagGen*-1 *Query*的回应方式与定理2证明中的回应方式相同.

3. *Challenge*:令S_1表示第1阶段被*Extract*-1 *Query*过的用户身份ID_i的集合,集合I_1存放第1阶段被*TagGen*-1 *Query*的(i,j,k,M_{ijk}).*B*生成挑战$chal^*=(I^*,K^*,\alpha^*)$,其中,$I^*=\{(i_n^*,j_n^*,k_n^*)\mid n=1,...,c^*\}$,$K^*=\{\kappa_{i_n}^*\mid n=1,...,c^*\}$,

$\alpha^*=\{a_{i_n}^* \mid n=1,...,c^*\}$,其中至少有 1 个 $ID_{i_n^*}\notin S_1$,且对于相同的 $ID_{i_n^*}$,至少有 1 个块 $(i_n^*,j_n^*,k_n^*,M_{i_n^*j_n^*k_n^*})\notin I_1$.*B* 将挑战 $chal^*$发送给 *A*.

4. 第 2 阶段 *Extract*-2 *Query*,*TagGen*-2 *Query* 与定理 2 中 *Extract Query*,*TagGen Query* 相同,并令 S_2 表示第 2 阶段被 *Extract*-2 *Query* 过的用户身份 ID_i 的集合,集合 I_2 存放第 2 阶段被 *TagGen*-2 *Query* 过的(i,j,k,M_{ijk}).需保证挑战 $chal^*$中至少有 1 个 $ID_{i_n^*}\notin(S_1\cup S_2)$,且对于相同的 $ID_{i_n^*}$,至少有 1 个块 $(i_n^*,j_n^*,k_n^*,M_{i_n^*j_n^*k_n^*})\notin(I_1\cup I_2)$.

5. *Forge*:敌手 *A* 针对挑战 $chal^*$,伪造证明 $P^*=\{P_{j^*}\}=\{(S'_{j^*},T'_{j^*},\{F'_{i^*j^*l}\mid i^*\in O_j^*\},\{(i^*,j^*,R_{i^*j^*})\mid i^*\in O_j^*\})\}$.*B* 在 H_1-*list* 中查找 $\{ID_{i_n^*}\mid n=1,...,c^*\}$,对于不存在列表中的 $ID_{i_n^*}$,*B* 对其做 H_1-*Query*.*B* 在 H_2-*list* 中查找 $\{ID_{i_n^*}\mid n=1,...,c^*\}$,对于不存在列表中的 $ID_{i_n^*}$,*B* 对其做 H_2-*Query*.*B* 在 H_3-*list* 中查找是否有(mpk,z,g^x),若没有,则 *B* 对 *mpk* 进行 H_3-*Query*.*B* 根据 $\{b_{i_n^*}\mid n=1,...,c^*\}$ 的值做如下操作.

1) 若 $\forall b_{i_n^*}=0,n=1,...,c^*$,则 *B* 报告失败,并终止模拟;

2) 若 $\exists b_{i_n^*}=1,n=1,...,c^*$,则 *B* 计算 $\{r_n^*\}=\{f_{\kappa_n}(i_n^*,j_n^*,k_n^*)\mid n=1,...,c^*\}$,令 O^*表示索引集合$\{i^*\}$,U^*表示索引集合$\{j^*\}$;若 P^*中关于数据标签的证明是有效的,则$\{(S'_{j^*},T'_{j^*},\{F'_{i^*j^*l}\mid i^*\in O_j^*\})\}$满足等式(2),即

$$e\left(\prod_{j^*\in U^*}T'_{j^*},g\right)=e\left(\prod_{i^*\in O^*}H_1(ID_{i^*})^{\sum_{j^*\in U^*}\sum_{l=1}^{s}v_l\cdot F'_{i^*j^*l}+H_2(ID_{i^*})\cdot\sum_{i_n=i^*}r_n^*},mpk\right)\cdot e\left(H_3(mpk),\prod_{j\in U^*}S'_{j^*}\right),$$

则

$$e\left(\prod_{j^*\in U^*}T'_{j^*},g\right)=e\left(\prod_{j^*\in U^*}\prod_{chal_{j^*}^*}H_1(ID_{i_n^*})^{(m_{i_nj^*k_n^*}+H_2(ID_{i_n^*}))r_n^*},mpk\right)\cdot e\left(H_3(mpk),\prod_{j\in U^*}S'_{j^*}\right)$$

$$=e\left(\prod_{n=1}^{c^*}H_1(ID_{i_n^*})^{(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*},mpk\right)\cdot e\left(H_3(mpk),\prod_{j\in U^*}S'_{j^*}\right).$$

令 $N_0=\{n\mid(ID_{i_n^*},b_{i_n^*},a_{i_n^*},H_1(ID_{i_n^*}))\in H_1\text{-}list,b_{i_n^*}=0\}$,$N_1=\{n\mid(ID_{i_n^*},b_{i_n^*},a_{i_n^*},H_1(ID_{i_n^*}))\in H_1\text{-}list,b_{i_n^*}=1\}\neq\varnothing$,则有:

$$e\left(\prod_{j^*\in U^*}T'_{j^*},g\right)=e\left(\prod_{n=1}^{c^*}H_1(ID_{i_n^*})^{(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*},mpk\right)\cdot e\left(H_3(mpk),\prod_{j\in U^*}S'_{j^*}\right)$$

$$=e\left(\prod_{n\in N_0}(g^{a_{i_n^*}})^{(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*}\cdot\prod_{n\in N_1}(g^{ya_{i_n^*}})^{(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*},g^x\right)\cdot e\left(g^z,\prod_{j\in U^*}S'_{j^*}\right)$$

$$=e\left(\prod_{n\in N_0}(g^x)^{a_{i_n^*}(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*}\cdot\prod_{n\in N_1}(g^{xy})^{a_{i_n^*}(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*}\left(\prod_{j\in U^*}S'_{j^*}\right)^z,g\right).$$

B 计算 $g^{xy}=\left[\prod_{j^*\in U^*}T'_{j^*}\cdot\left(\prod_{j\in U^*}S'_{j^*}\right)^{-z}\cdot(g^x)^{-\sum_{n\in N_0}a_{i_n^*}(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*}\right]^{\left(\sum_{n\in N_1}a_{i_n^*}(m_{i_nj_nk_n}^*+H_2(ID_{i_n^*}))r_n^*\right)^{-1}}$.

令 *E* 表示事件 *B* 求解出 g^{xy},经过上面的分析可知,在 *A* 进行 n_{EQ} 次 *Extract Query* 和 n_{TQ} 次 *TagGen Query* 未终止,且 *A* 针对挑战能够成功伪造出可通过校验的数据标签证明;同时,对 $\{ID_{i_n^*}\mid n=1,...,c^*\}$ 进行 H_1-*Query* 时,其中至少有 1 个 $b_{i_n^*}=1$,则事件 *E* 发生.设敌手 *A* 赢得伪造数据标签证明游戏的概率为 ε,则 *B* 解决群 G_1 上的 CDH 问题的概率为 $\Pr[E]=\delta^{n_{EQ}+n_{TQ}}\varepsilon(1-\delta^{c^*})$. 当 $\delta=\left[\frac{(n_{EQ}+n_{TQ})}{n_{EQ}+n_{TQ}+c^*}\right]^{c^{*-1}}$ 时,$\Pr[E]=\left[\frac{(n_{EQ}+n_{TQ})^{n_{EQ}+n_{TQ}}}{(n_{EQ}+n_{TQ}+c^*)^{n_{EQ}+n_{TQ}+c^*}}\right]^{c^{*-1}}c^*\varepsilon$ 取得最大值.证毕. □

定理 4. 如果 Hash 函数是抗碰撞的,则不存在一个 PPT 敌手能够以不可忽略的概率伪造出通过 TPA 校验的定位标签证明.

证明:被挑战服务器在每次挑战时都会收到 TPA 发来的重构 MHT 的参数.由于每次挑战用于定位的 MHT 根标签所使用的参数不同,使得服务器无法提前预知、计算并存储叶子节点,所以收到挑战的服务器都需要根据挑战中 MHT 参数集$\{\alpha_j\}$重新计算相应的 MHT 根值作为证明.在 MHT 中,叶子节点是参数与文件块连接后做 Hash 运算得到的,而根值是由所有叶子节点经过多层计算 Hash 值得到的,所以,若 Hash 函数是抗碰撞的,则要在不知道参数与文件块值的情况下伪造出可通过校验的根值的概率是可以忽略的.证毕. □

此外,要说明的是,为了实现错误定位,且使得用户端不额外存储多余的信息,有些信息,如用户数据块所存储的服务器和所有用户数据的定位索引表,TPA 都是知道的.而定位索引表中的元素是每个用户存储在不同服务器上的数据所构成的 MHT 树根,若 Hash 函数是抗原象攻击的,则 TPA 无法得知用户原始数据块值.

5.3 性能分析

在下面的分析中,n 表示所有云用户的文件块总数,c 表示 TPA 选中的被挑战块的数量,n_1 表示 c 个被挑战块所属用户的数量,n_2 表示 c 个被挑战块所在的服务器的数量,η表示所有云服务器的数量,γ表示所有云用户的数量,s 表示每个数据块的分区数,λ表示每个用户对其存放在一个服务器上的数据所构建的 MHT 数量.c_j 表示被挑战服务器 CS_j 上被挑战的块数,有 $1\leqslant c_j\leqslant c_{\max}=c-n_2+1$.假设云用户 DO_i 总共将 m_i 个文件块存储到多个服务器中,服务器 CS_j 上共存储了 m'_j 个文件块.

方案的通信轮数为 1 轮,在此一轮通信中既实现了批处理校验,又能实现错误数据的定位功能.

1. 关于通信复杂度

在初始阶段,云用户除了将文件块$\{M_{ijk}\}$和相应的数据标签$\{\sigma_{ijk}\}$发送给服务器以外,为了实现对其错误数据进行定位,还需要将定位索引表发送给 TPA,每个用户的定位索引表中包含$\lambda(\eta+1)$个 Z_q 中的元素.

在挑战阶段,TPA 发送总挑战 $chal=(I,K,\alpha)$,其中,I 中包括 $3c$ 个整数,K 中包含 c 个 Z_q 中的元素,α中也包含 c 个 Z_q 中的元素.综上,挑战阶段的通信复杂度为 $O(c)$.

在响应阶段,每个被挑战服务器返回的证明 $P_j=(S'_j,T'_j,\{F'_{ijl}\mid i\in O_j,l=1,...,s\},\{(i,j,R_{ijr})\mid i\in O_j\})$, 其中, S'_j, $T'\in G_1,\{F'_{ijl}\mid i\in O_j,l=1,...,s\}$中至多包含 n_1s 个由数据块分区计算得到的值,$\{(i,j,R_{ijr})|i\in O_j\}$中至多包含 $2n_1$ 个整数和 n_1 个 Z_q 中的元素.因此,每个服务器返回的证明中至多包含 $2+n_1s+2n_1+n_1$ 个元素,所有被挑战服务器共返回至多$(2+n_1s+3n_1)n_2$ 个元素.综上,响应阶段的通信复杂度为 $O(n_1n_2s)$.

2. 关于存储复杂度

服务器存储着用户的数据块和相应的数据标签,与其他方案相似.TPA 需要存储所有用户发来的数据块定位标签,共有γ张定位索引表,包含$\gamma\lambda(\eta+1)$个 Z_q 中的元素,存储复杂度为 $O(\gamma\lambda\eta)$.

3. 关于计算复杂度

云用户:除了为每个数据块计算相应的数据标签以外,为了实现错误定位,额外地,云用户需要对其存储在不同服务器上的数据块分别构建λ棵 MHT.含有 N_{ij} 个叶子节点的 MHT 最多需要做 $2^{\lceil\log_2 N_{ij}\rceil+1}-1$ 次 Hash 运算.拥有 m_i 个数据块的云用户 DO_i 最多计算 $\lambda\cdot(2^{\lceil\log_2 m_i\rceil+1}-1)$ 次 Hash,因此,DO_i 计算定位标签的时间复杂度为 $O(\lambda m_i)$,这项工作是一次性的.

被挑战服务器:计算的证明包含两部分:(1) 用于批处理校验的部分,需要做 c_j 次伪随机函数运算、$2c_j$ 次指数运算、$2(c_j-1)$次群中乘法运算、$s\cdot c_j$ 次普通乘法运算,最多 $s\cdot(c_j-1)$次普通加法运算;(2) 用于定位错误的部分,最多需要进行 $2^{\lceil\log_2 m'_j\rceil+1}-1$次 Hash 运算.

TPA:批处理校验包括 c 次伪随机函数运算、n_1 次指数运算和 3 次双线性对运算,最多 $n_1(s-1)\cdot(n_2-1)+c$ 次普通加法运算、最多 $s\cdot\min(n_1n_2,c)$次普通乘法运算、$2(n_2-1)+(n_1-1)$次群中乘法运算.若批处理校验不通过,则再对服务器返回的树根一一进行对比校验,判断某一用户存放在某一服务器上的数据是否遭到损毁,只需一次比较操作.

下面将我们的方案与其他多用户多服务器环境下支持批处理校验的方案进行比较.从效率和是否支持错误定位方面,我们的方案与 He 等人[18]、Shin 等人[19]和 Zhou 等人[17]的方案进行对比的结果见表 3.

Table 3 Comparison of efficiency and location function

表 3 效率及定位功能比较

方案	我们的方案		He 等人的方案[18]	Shin 等人的方案[19]	Zhou 等人的方案[17]
审计阶段通信复杂度	与数据标签相关的 $2n_2+4c+n_1n_2s$	与定位标签相关的 $3n_1n_2+c$	$2c+n_1n_2+2$	$4n_2+2c+n_1n_2$	$n_1n_2s+2n_2+4c$
用户端计算复杂度	$3m_iC_{exp}+(m_i)C_{mG}+sm_iC_{mZ}+(s+1)m_iC_{aZ}+2C_H$	$2\lambda m_iC_H$	$2m_iC_{exp}+m_iC_{mG}$	$2m_iC_{exp}+m_iC_{mG}+m_iC_H+C_{sig}$	$3m_iC_{exp}+(m_i)C_{mG}+sm_iC_{mZ}+(s+1)m_iC_{aZ}+2C_H$
服务器端计算复杂度	$c_jC_{per}+2c_jC_{exp}+(2c_j-2)C_{mG}+sc_jC_{mZ}+s(c_j-1)C_{aZ}$	$2m'_jC_H$	$n_1c'C_{exp}+n_1c'C_{mG}+n_1c'C_{mZ}+n_1c'C_{aZ}$	$(n_1+c_j)C_{exp}+3n_1C_{par}+(2n_1+c_j)C_{mG}+c_jC_{mZ}+c_jC_{aZ}+2n_1C_H$	$c_jC_{pre}+2c_jC_{exp}+(2c_j-2)C_{mG}+sc_jC_{mZ}+s(c_j-1)C_{aZ}$
TPA 端批处理校验计算复杂度	$cC_{per}+n_1C_{exp}+3C_{par}+(n_1+2n_2-3)C_{mG}+s\cdot\min(n_1n_2,c)C_{mZ}+[n_1\cdot(s-1)(n_2-1)+c]C_{aZ}$		$n_1C_{exp}+3C_{par}+2n_1C_{mG}$	$cC_{exp}+n_1n_2C_{par}+(4n_1+c)C_{mG}+C_{dG}+cC_H$	$cC_{per}+n_1C_{exp}+3C_{par}+(n_1+2n_2-3)C_{mG}+s\cdot\min(n_1n_2,c)C_{mZ}+[n_1\cdot(s-1)(n_2-1)+c]C_{aZ}$
TPA 端存储复杂度	$\gamma\lambda(\eta+1)$		无	无	无
是否支持定位错误数据所在服务器	是		否	是	否
是否支持定位错误数据所属拥有者	是		是	否	否
TPA 定位特定错误数据的时间或计算复杂度	T_{cpr}		$(c'+1)C_{exp}+(c'-1)C_{mG}+3C_{par}+T_{cpr}$	$n_2T_{cpr}+\left(\frac{3}{2}n_2^2+\frac{5}{2}n_2-1\right)C_{mG}$	–

其中,$C_{\exp}$ 表示群 G_1 上单个指数运算的开销,C_H 表示单个 Hash 函数运算的开销,C_{par} 表示一个双线性对运算的开销,C_{per} 表示一个伪随机函数运算的开销,C_{mG} 表示群上单个乘法运算的开销,C_{dG} 表示群上单个除法运算的开销,C_{mZ} 表示单个普通乘法运算的开销,C_{aZ} 表示单个普通加法运算的开销,T_{cpr} 表示一次比较的时间开销.因为文献[18]中每个用户的数据都会被挑战且被挑战的块索引相同,令 c'表示一个被挑战用户被挑战的块数,即 $c=c'\cdot n_1$.

4 个方案中,审计阶段都仅需 1 轮通信,我们的方案在 1 轮通信中不仅能够实现批处理校验,还能在校验不通过情况下定位错误数据所属用户和所属服务器;而文献[18]只支持定位错误数据的拥有者;文献[19]只支持定位错误数据所在服务器,并且仅适用于只有 1 个服务器的数据被损毁的情景.我们的方案是基于查找的方式定位错误数据,判断特定用户存储在特定服务器上的数据是否出错仅需一次比较操作;而文献[18,19]都是利用计算的方式来实现错误数据定位,文献[18]判断特定用户的数据是否损毁需要 $O(c')$次指数运算,文献[19]定位唯一的数据被损毁服务器需要 $O(n_2^2)$ 次乘法运算.

在我们的方案中,为了计算定位标签,用户需要为其数据块构建 MHT,服务器在计算证明时需要重构 MHT,所以相对于其他方案,在用户端和服务器端分别增加了 $2\lambda m_i$ 和 $2m'_j$ 次 Hash 运算.由于 Hash 运算的速度很快,所以增加的计算对总体的计算开销影响并不显著.为了定位错误,相对于文献[17–19],TPA 需要额外存储 γ 张定位索引表,总共 $\gamma\lambda(\eta+1)$ 个 Z_q 中的元素,而 TPA 进行批处理校验的计算量相对于文献[17]来说并没有增加.

5.4 仿真实验

我们首先对云用户、服务器和 TPA 各自计算的时间开销进行了仿真实验,然后对两种定位错误方式的定位效率进行了对比.PC 硬件配置为 Intel Core2Duo 处理器、4G 内存,操作系统为 Ubuntu 16.04 LTS 32 位,利用 PBC 库[20]、GMP 库[21]和 Miracl 库[22],使用 gcc 编译执行.实验中,使用 PBC 库中 a.param 参数设置双线性对,构

造 MHT 时采用 SHA-256.

1. 用户计算数据标签 TagGen 和定位标签 PosTagGen 的计算开销

用户将文件分块后,对每个数据块计算相应的数据标签,然后对存储在不同服务器上的数据块计算定位标签.在实验中,我们设置每个数据块 4KB,每个分区 20B,通过设置不同的文件大小来观察 TagGen 和 PosTagGen 的计算开销.令用户文件大小为 5MB~40MB,相应的数据块数为 1 250~10 000,设置步长 5MB.TagGen 的计算开销与λ=64 和λ=128 时 PosTagGen 的计算开销实验结果如图 4 所示.

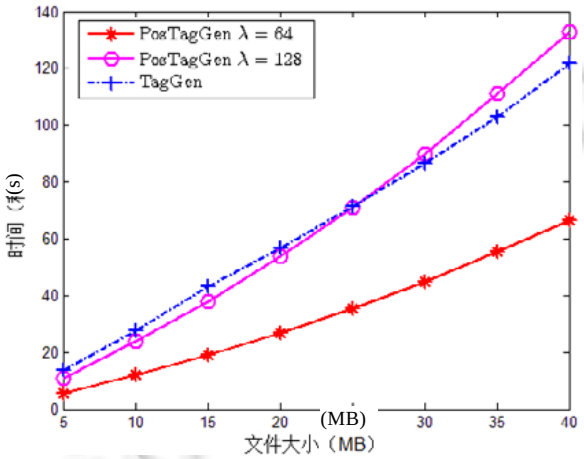


Fig.4 Computational cost of TagGen & PosTagGen for increased size of files

图 4 文件大小增加时 TagGen 和 PosTagGen 的计算开销

从实验结果可以看出,TagGen 的时间随着文件大小的增长(数据块数增长)呈线性增长,与性能分析结果一致.特别地,当文件大小为 5MB(40MB)时,TagGen 的时间为 13.6s(121.7s).PosTagGen 的计算开销与文件大小(数据块数量)和λ值是相关的:当λ值固定时,PosTagGen 的计算开销随着数据块数量增加而增大,基本呈线性增长,实验结果与性能分析相符合.进一步观察,当λ=64,文件大小为 5M(40M)时,PosTagGen 的时间为 5.5s(66.5s).当λ=128,文件大小为 5MB(40MB)时,PosTagGen 的时间为 11.0s(133.0s).TagGen 与 PosTagGen 的计算开销较大,耗费的时间显著高于其他操作耗费的时间,但是对于一个文件来说都仅需计算 1 次.表 4 为当λ=64 和λ=128 时,TPA 进行校验的不同时间间隔所能支持的错误定位有效期.

Table 4 Error locating period of validity for different verification time interval

表 4 不同校验时间间隔下,支持错误定位的有效期

校验时间间隔	支持定位的有效期	
	λ=64	λ=128
7 天	15 个月	30 个月
14 天	30 个月	60 个月
1 个月	64 个月	128 个月
3 个月	192 个月	384 个月

2. 服务器计算证明 Prove-DataTag、Prove-PositionTag 和 TPA 批量校验数据标签证明 Verify 的计算开销

在挑战响应阶段,收到挑战的服务器需要计算数据标签的证明 Prove-DataTag 和定位标签的证明 Prove-PositionTag,而 TPA 需对被挑战服务器返回的数据标签证明进行批量审计.

服务器 Prove-DataTag 的计算开销与 TPA 批量审计 Verify 的计算开销均与 TPA 选取的挑战块数量有关,所以我们在同样的条件下对这两个计算进行了仿真.实验中,我们设置了 10 个用户和 10 个服务器,每个用户拥有 10 000 个数据块,每个数据块 4KB,每个分区 20B,每个用户将其 10 000 个数据块均匀地存储到 10 个服务器

上,这样,每个服务器上存储着 10 个用户的 10 000 个数据块.若云服务器的数据块损毁率为 1%,则 TPA 挑战其上的 300 个(460 个)数据块,就能以 95%(99%)的概率检测出该服务器的损毁数据行为[23].因此,在 TPA 随机均匀选取 10 个服务器上的挑战块的情况下,我们对总挑战块数为 3 000~4 600(相应的每个服务器上被挑战块数为 300~460),以步长为 200,进行了实验,结果如图 5 所示.

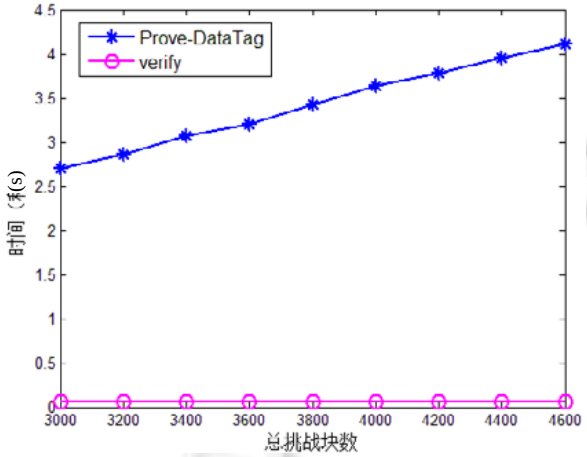


Fig.5 Computational cost of Prove-DataTag by single server & Verify by TPA for increased number of total challenged blocks

图 5 总挑战块数增加时,单个服务器 Prove-DataTag 和 TPA Verify 的计算开销

从实验结果可以看出,服务器 Prove-DataTag 的计算开销随着其上被挑战块数量的增加而增长.这主要是因为当其上被挑战块数增加时,服务器需要为增加的挑战块的数据标签做群上的指数运算.随着总挑战块数的增加,TPA 批量校验数据标签证明 Verify 的计算开销增长并不明显.这是因为增加的操作只是一些伪随机函数和普通加法运算,实验结果与性能分析结果一致.进一步观察,当总挑战块数为 3 000(4 600)个时,服务器 Prove-DataTag 的时间为 2.7s(4.1s),TPA Verify 的时间仅为 53ms(54ms).

被挑战服务器计算定位标签证明 Prove-PositionTag 的计算开销与该服务器上被挑战用户的所有数据块数有关.我们对其中最坏的情况进行了模拟,即数据存放于该服务器上的所有用户均有数据块被 TPA 挑战,所以被挑战服务器需对其上存储的所有数据块进行重构 MHT 的操作.我们令被挑战服务器存储的数据块数为 5 000~12 000,步长为 1 000,对单个被挑战服务器计算定位标签证明 Prove-PositionTag(即重构 MHT)的计算开销进行测试,结果如图 6 所示.

从实验结果可以看出,服务器重构 MHT 的计算开销随着该服务器上存储的数据块数增加而增长,且增长趋势基本呈线性,与性能分析结果一致.由于 Prove-PositionTag 的计算是重构 MHT 树,所做的都是 Hash 操作,因此即使是在最坏情况下,重构 MHT 的时间也是较快的.当服务器上存有 5 000(12 000)个数据块,且所有其上用户都有数据块被挑战时,服务器 Prove-PositionTag 的时间为 0.42s(1.34s).

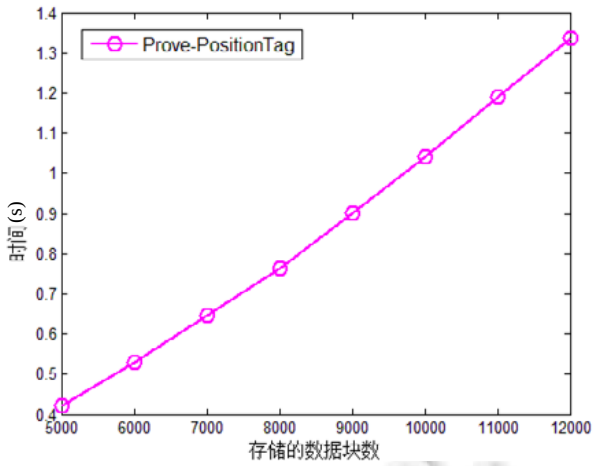


Fig.6 Computational cost of Prove-PositionTag by single server for its increased number of stored blocks

图 6 存储的数据块数增加时,单个服务器 Prove-PositionTag 的计算开销

3. TPA 定位错误时间比较

在本节中,我们对两种定位错误方式的定位效率进行对比:逐一校验方式和利用定位标签辅助定位的方式.为了使对比结果更合理,我们同样是在 Zhou 等人方案[17]的基础上实现逐一校验定位错误,并设置相同的环境参数.但需要说明的是,逐一校验方式使得 Zhou 等人的方案只能定位错误数据所在服务器.

在实验中,我们设置 10 个用户和 100 个服务器,每个用户拥有 100 000 个数据块,每个数据块 4KB,每个分区 20B,每个用户将其 100 000 个数据块均匀存储到 100 个服务器上,这样,每个服务器上就存储着 10 个用户的 10 000 个数据块.在 TPA 随机均匀选取每个被挑战服务器上 10 个用户的 300 个挑战块的情况下,令被挑战服务器个数为 10~100,模拟两种定位错误方法的计算开销,实验结果如图 7 所示.

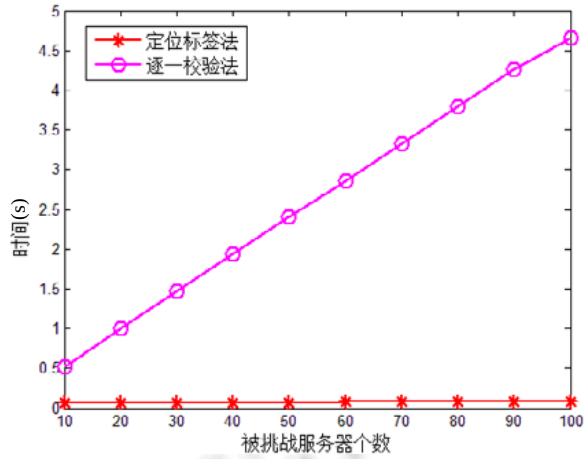


Fig.7 Time comparison of location errors by TPA

图 7 TPA 定位错误时间比较

从实验结果可以看出,随着被挑战服务器数量的增加,逐一校验方式定位错误服务器的时间耗费呈线性增长趋势,而利用定位标签定位错误数据所属用户和所在服务器的计算开销增长并不明显.这是因为逐一校验方式中,TPA 针对每个被挑战服务器返回的证明单独校验,每次校验都需要做多个指数运算和双线性对运算;而利用定位标签的方式只涉及比较操作.在被挑战服务器个数为 10(100)时,定位标签方式仅需 0.059s(0.087s),而逐

一校验方式则需 0.523s(4.652s).显然,随着被挑战服务器个数的增加,使用定位标签辅助错误定位的效率显著优于逐一校验的方式.

6 总 结

本文主要研究了批处理 PDP 方案在批量审计不通过情况下的错误数据定位问题.提出了利用定位标签辅助第三方审计员快速定位错误的方法,并在多用户多服务器环境下给出了一个具体实现,在批处理校验不通过的情况下,仅通过比较操作就能同时定位错误数据所属用户和所在服务器.我们对方案的正确性和安全性进行了证明,并对方案的性能进行了理论分析和仿真实验.性能分析结果表明,我们的方案在定位错误数据的能力和效率方面均高于其他具有单一定位功能的方案.实验结果也表明,利用定位标签辅助错误定位的效率显著优于逐一校验的方式,且实现错误定位功能的额外计算开销是可接受的.

在本文方案中,预先设定的 MHT 数量使得本方案不适合进行无限次的错误定位.为了缓解次数限制问题,有两种可行的解决办法.

(1) 不要求每次校验都返回树的根值.仅当批处理校验发生错误后,校验者给服务器再次发送挑战,要求服务器提供相应树的根值.

(2) 对服务器建立分级制度,校验者在挑战时可设立一个状态标识,若状态为“1”,则要求返回根值;若为“0”则不要求.对信誉好的服务器,可以适当减少其返回根值的次数.

References:

[1] Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson Z, Song D. Provable data possession at untrusted stores. In: Proc. of the 14th ACM Conf. on Computer and Communications Security. ACM Press, 2007. 598−609. [doi: 10.1145/1315245.1315318]

[2] Ateniese G, Pietro RD, Mancini LV, Tsudik G. Scalable and efficient provable data possession. In: Proc. of the 4th Int’l Conf. on Security and Privacy in Communication Networks. ACM Press, 2008. 1−10. [doi: 10.1145/1460877.1460889]

[3] Wang Q, Wang C, Li J, Ren K, Lou WJ. Enabling public verifiability and data dynamics for storage security in cloud computing. In: Backes M, Ning P, eds. Proc. of the Computer Security (ESORICS 2009). LNCS 5789, Berlin: Springer-Verlag, 2009. 355−370. [doi: 10.1007/978-3-642-04444-1_22]

[4] Wang C, Wang Q, Ren K, Lou WJ. Privacy-preserving public auditing for data storage security in cloud computing. In: Proc. of the 2010 IEEE INFOCOM. IEEE, 2010. 1−9. [doi: 10.1109/INFCOM.2010.5462173]

[5] Hao Z, Zhong S, Yu NH. A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. IEEE Trans. on Knowledge & Data Engineering, 2011,23(9):1432−1437. [doi: 10.1109/TKDE.2011.62]

[6] Yu Y, Au MH, Mu Y, Tang SH, Ren J, Susilo W, Dong LJ. Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage. Int’l Journal of Information Security, 2015,14(4):307−318. [doi: 10.1007/s10207-014-0263-8]

[7] Yu Y, Zhang YF, Ni JB, Au MH, Chen LX, Liu HY. Remote data possession checking with enhanced security for cloud storage. Future Generation Computer Systems, 2015,52:77−85. [doi: 10.1016/j.future.2014.10.006]

[8] Yu Y, Ni JB, Au MH, Mu Y, Wang BY, Li H. Comments on a public auditing mechanism for shared cloud data service. IEEE Trans. on Services Computing, 2015,8(6):998−999. [doi: 10.1109/TSC.2014.2355201]

[9] Yu Y, Li YN, Ni JB, Yang GM, Mu Y, Susilo W. Comments on “public integrity auditing for dynamic data sharing with multiuser modification”. IEEE Trans. on Information Forensics & Security, 2016,11(3):658−659. [doi: 10.1109/TIFS.2015.2501728]

[10] Yu Y, Xue L, Au MH, Susilo W, Ni JB, Zhang YF, Vasilakos AV, Shen J. Cloud data integrity checking with an identity-based auditing mechanism from RSA. Future Generation Computer Systems, 2016,62:85−91. [doi: 10.1016/j.future.2016.02.003]

[11] Yu Y, Au MH, Ateniese G, Huang XY, Susilo W, Dai YS, Min GY. Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage. IEEE Trans. on Information Forensics & Security, 2017,12(4):767−778. [doi: 10.1109/TIFS.2016.2615853]

[12] Wang C, Chow SM, Wang Q, Ren K, Lou WJ. Privacy-preserving public auditing for secure cloud storage. IEEE Trans. on Computers, 2013,62(2):362−375. [doi: 10.1109/TC.2011.245]

[13] Ren ZW, Wang LN, Wu QH, Deng RY. Data dynamics enabled privacy-preserving public batch auditing in cloud storage. Chinese Journal of Electronics, 2014,23(2):297−301.

[14] Wang HQ. Identity-based distributed provable data possession in multicloud storage. IEEE Trans. on Services Computing, 2015, 8(2):328−340. [doi: 10.1109/TSC.2014.1]

[15] Mao J, Cui J, Zhang Y, Ma HJ, Zhang JH. Collaborative outsourced data integrity checking in multi-cloud environment. In: Yang Q, Yu W, Challal Y, eds. Proc. of the Wireless Algorithms, Systems, and Applications (WASA 2016). LNCS 9798, Berlin: Springer-Verlag, 2016. 511−523. [doi: 10.1007/978-3-319-42836-9_45]

[16] Liu X, Jiang YJ. Batch auditing for multi-client dynamic data in multi-cloud storage. Int'l Journal of Security & Its Applications, 2014,8(6):197−210. [doi: 10.14257/ijsia.2014.8.6.18]

[17] Zhou FC, Peng S, Xu J, Xu ZF. Identity-based batch provable data possession. In: Chen L, Han J, eds. Proc. of the Provable Security (ProvSec 2016). LNCS 10005, Berlin: Springer-Verlag, 2016. 112−129. [doi: 10.1007/978-3-319-47422-9_7]

[18] He K, Huang CH, Wang JH, Zhou H, Chen X, Lu YL, Zhang LZ, Wang B. An efficient public batch auditing protocol for data security in multi-cloud storage. In: Proc. of the 8th Chinagrid Conf. IEEE, 2013. 51−56. [doi: 10.1109/ChinaGrid.2013.13]

[19] Shin S, Kim S, Kwon T. Identification of corrupted cloud storage in batch auditing for multi-cloud environments. In: Khalil I, Neuhold E, Tjoa A, Xu L, You I, eds. Proc. of the Information and Communication Technology—EurAsia Conf. (ICT-EurAsia 2015). LNCS 9357, Berlin: Springer-Verlag, 2015. 221−225. [doi: 10.1007/978-3-319-24315-3_22]

[20] https://crypto.stanford.edu/pbc/

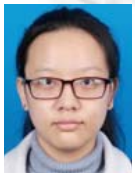
[21] https://gmplib.org/

[22] https://certivox.org/display/EXT/MIRACL

[23] Ateniese G, Burns R, Curtmola R, Herring J, Khan O, Kissner L, Peterson Z, Song D. Remote data checking using provable data possession. ACM Trans. on Information & System Security, 2011,14(1):No.12. [doi: 10.1145/1952982.1952994]

庞晓琼(1982－),女,山西太原人,博士,讲师,CCF 专业会员,主要研究领域为信息安全,密码学.

陈文俊(1980－),男,博士生,高级工程师,主要研究领域为密码学理论及其应用.



王田琪(1993－),女,硕士,主要研究领域为信息安全,密码学.

任孟琦(1994－),女,硕士生,CCF 学生会员,主要研究领域为信息安全,密码学.