第 37 卷　　计 算 机 学 报　　Vol. 37
2014 论文在线出版号 No.32　　CHINESE JOURNA4L O1F COMPUTERS　　Online Publishing No.32

云存储中的数据完整性证明研究及进展

谭霜，贾焰，韩伟红

（国防科学技术大学计算机学院，长沙 410073）

摘　要　随着云存储模式的出现，越来越多的用户选择将应用和数据移植到云中，但他们在本地可能并没有保存任何数据副本，无法确保存储在云中的数据是完整的．如何确保云存储环境下用户的数据是完整的，成为近来学术界研究的一个热点．数据完整性证明(Provable Data Integrity, PDI) 被认为解决这一问题的重要手段，本文对此进行了综述．首先，给出了数据完整性证明机制的协议框架，分析了云存储环境下数据完整性证明所特有的特征；其次，对各种数据完整性证明机制加以分类，在此分类基础上，介绍了各种典型的数据完整性验证机制并进行了对比；最后，指出了云存储中数据完整性验证面临的挑战及发展趋势．

关键词　云存储；数据完整性证明；数据持有性证明；数据可恢复性证明；

Research and Development of Provable Data Integrity in Cloud Storage

TAN Shuang1), JIA Yan1), HAN Wei-hong1)

1) (School of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract With the advent of cloud storage model, more and more users choose to transfer their applications and data into the cloud, and no longer store any data in their local memory, which can not ensure that the data stored in the cloud is still intact. How to protect the data integrity in the cloud has become a hot topic of academic research. The protocol of Provable Data Integrity (PDI) is considered to be the main method to solve this problem, which is studied in this paper. Firstly, we give a framework of provable data integrity in this paper, and analysis the unique features of data integrity under the cloud storage environment. Secondly, classification of the PDI protocol is analyzed, and on the basis of the classification, many typical PDI models are introduced and compared. Lastly, future research trends of PDI protocol for the cloud storage environment are reviewed.

Key words Cloud Storage; Provable Data Integrity; Provable Data Possession; Proofs of Retrievability;

1 前言

云计算是继对等计算、网格计算、效用计算、分布式计算后又一新型的计算模式，其一出现便成为了学术界、工业界关注的焦点[1]．云计算的核心理念是资源租用、应用托管、服务外包，其通过虚拟化技术将分布的计算节点组成一个共享的虚拟化池，为用户提供地服务[2]．通过云计算技术，用户和企业并不需要花费过高的代价在前期硬件的购置和维护上．另外，强大的计算和存储能力也使得用户更愿意依托云来处理各种复杂的任务．当用户选择将大量应用和数据部署到云计算平台中时，云计算系统也相应地变为云存储系统．云存储系统

本课题得到国家高技术研究发展(863计划)项目 (No. 2012AA01A401，2012AA01A402)；国家重点基础研究发展计划(973计划)项目(No.2013CB329601，No.2013CB329602)资助．**谭霜**，男，1984年生，博士，主要研究领域为网络安全、云数据安全，E-mail:tanshuang@nudt.edu.cn．**贾焰**，女，1961年生，博士生导师，职称教授，主要研究领域为分布式数据库、数据挖掘及云安全，E-mail: jiayanjy@vip.sina.com．**韩伟红**，女，1974年生，博士，职称副研究员，主要研究领域为网络安全、云安全，E-mail: weihonghan@nudt.edu.cn．

可以使用户以较低廉的价格获取海量的存储能力，但高度集中的计算资源使云存储面临着严重安全挑战．据Gartner2009 年调查的结果显示，因担心云数据隐私被侵犯，70%受访企业的CEO拒绝大规模的采用云计算的计算模式[1]．而最近几年里，各大云运营商各自暴漏的安全存储问题，引起了人们的广大关注与担忧．如 2011 年 3 月，谷歌Gmail邮箱出现故障，而这一故障造成大约 15 万用户的数据丢失[2]．2012 年 8 月，国内云提供商盛大云因机房一台物理服务器磁盘发生故障，导致客户部分数据丢失[3]．由此可见，云中的数据安全存储已经阻碍了云计算在IT领域得到大规模的使用[3, 4].

当用户采用云存储模式后，其存储在云中的数据可能遭到其他用户或云计算提供商的窥视、修改或损坏．通常，数据的机密性可通过数据加密、匿名机制等机制来确保[5-7]．但当采用云存储后，用户可能并没有在本地保存任何数据副本，从而无法保证云中的数据是完整的．用户将数据存储到云中时，可能面临以下三种损坏数据的行为:①软件失效或硬件损坏导致数据丢失，这种失效属于概率性事件；②存储在云中的数据可能遭到其他用户的恶意损坏，文献[8]以 Amazon EC2 存储服务为例，指出恶意的用户可以对云中同一宿主机上的其它虚拟机发起攻击，损坏其他用户的数据；③云服务提供商可能并没有遵守服务等级协议(SLAs)，其为了经济利益，擅自删除一些用户不常访问的数据，或采取离线存储方式．总而言之，在现有的云体系框架下，云中数据的完整性无法得到有效的确保[9].

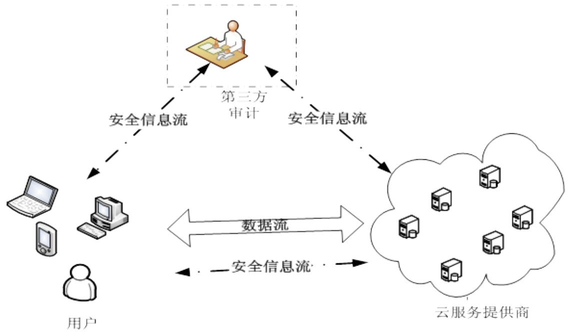


图 1 云环境下存储模型

数据完整性证明机制能及时的识别云中损坏数据的行为[10]．本文通过对现有的多种数据完整性检测机制进行归类分析，探讨了适合云存储的完整性验证机制，并对云存储领域的数据完整性验证机制的发展趋势进行了展望.

2 云存储环境下数据完整性证明

本节将对云存储环境下的数据模型和数据完整性验证机制的基本框架进行了简单介绍.

2.1数据存储模型

云存储环境下数据完整性证明所采用数据存储模型如图 1 所示，由用户(Customer)、云服务提供商(Cloud Server Provider, CSP)及可信第三方审计(Trusted Party Auditor, TPA)组成[9]．其中，用户(Customer)可以自定义地定制云存储服务；云服务提供商(Cloud Server Provider, CSP)为用户提供存储或计算服务，具有强的计算能力和存储能力；第三方审计(Third Party Auditor, TPA)，具有用户所没有的审计经验和能力，可以替代用户对云中存储的数据执行审计任务，减轻用户在验证阶段的计算负担.

由于接入云的设备受计算资源限制，用户不可能将大量的时间和精力花费在对远程节点的数据完整性检测上．通常，云用户将完整性验证任务移交给经验丰富的第三方来完成．采用第三方验证时，验证方只需要掌握少量的公开信息即可完成完整性验证任务.

2.2 构建框架

定义 1. 数据完整性证明机制由$Setup$和$Challenge$两个阶段组成，通过采用抽样的策略对存储在云中的数据文件发起完整性验证．具体实现由6个多项式时间内算法组成，如下所示:

a) 密钥生成算法:$KeyGen(1^k)\to(pk,sk)$．由用户在本地执行．k为安全参数，返回一个匹配的公钥、私钥对(pk,sk).

b) 数据块标签生成算法:$TagBlock(sk,F)\to\Phi$.$TagBlock(\cdot)$算法由用户执行，为每个文件生成同态签名标签集合Φ,作为认证的元数据．该算法输入参数包括私钥sk和数据文件F,返回认证的元数据Φ.

c) 证据生成算法:$GenProof(pk,F,\Phi,chal)\to P$．该算法由服务器运行,生成完整性证据$P$．输入参数包括公钥$pk$，文件$F$，挑战请求$chal$和认证元数据集合$\Phi$．返回该次请求的完整性证据$P$.

d) 证据检测算

---

[1] Gartner Identifies the Top 10 Strategic Technologies for 2009. http://www.gartner.com/newsroom/id/777212, 2009.

[2] http://cio.zol.com.cn/228/2281017.html, 2011.

[3] http://tech.sina.com.cn/it/2012-08-08/01177478277.shtml, 2012.

法 : $CheckProof(pk, chal, P) \rightarrow (“true”, “false”)$. 由用户或可信第三方 TPA 运行，对服务器返回的证据P进行判断. 输入参数为公钥pk，挑战请求$chal$及P返回验证成功或失败.

当存储在云中文件需要支持动态操作时，还需要以下两个算法支持:

e) 更新执行算法 : $ExecUpdate(F, \Phi, Update) \rightarrow (F', \Phi', V_{update})$. 算法由服务器运行，将文件$F$作为输入,相应标签$\Phi$及数据请求操作$Update$，输出一个更新文件$F'$和更新标签集合$\Phi'$，及相对应地更新证据$V_{update}$.

f) 更新验证算法 : $VerifyUpdate(pk, Update, P_{update}) \rightarrow ("true", "false")$. 该算法由用户执行，返回更新操作成功或失败.

数据完整性验证机制在具体实施过程中可以分为个两阶段组成:$Setup$阶段和$Challenge$阶段:

$Setup$阶段:初始化阶段. 用户首先运行$KeyGen(\cdot)$生成密钥对(pk, sk),然后对存储的文件进行划分$F = (m_1, m_2, \cdots, m_n)$运行$TagBlock(\cdot)$为文件中每一个数据块生成同态标签集合$\Phi = \{\sigma_i\}_{1 \leq i \leq n}$,之后. 然后将数据文件$F$和签名集合$\Phi$同时存入云中，删除的本地$\{F, \Phi\}$.

$Challenge$阶段:验证请求阶段. 用户或TPA作为验证者，周期性的发起完整性验证. 从文件F分块索引集合$[1, n]$中随机挑取c个块索引$\{s_1, s_2, \cdots, s_c\}$，并且为每一个索引$s_i$选取一个随机数$\{v_i\}$，将两者组合一起生成挑战请求$chal = \{i, v_i\}_{s_1 \leq i \leq s_c}$发送给服务器.

服务器作为证明者，根据存储在其服务器上的数据文件$\{F, \Phi\}$，调用$GenProof(\cdot)$生成完整性证据P,返回给验证者(用户或 TPA). 验证者接受证据后,执行$CheckProof(\cdot)$验证证据是否正确. 采用第三方审计时, TPA 需要将验证结果发送给用户.

2.3 云存储环境下数据完整性证明所具有的特性

与 P2P 网络、网格计算等分布式网络相比，云存储环境下数据完整性验证机制具有以下几个显著特点:①支持动态操作. 为了满足云中的应用，需要对完整性验证机制支持动态操作，而传统的数据完整性证明机制已预先为每一个数据文件生成不可伪造的数据签名标签集合，当数据进行更新时需要重新生成大部分签名标签，使得计算代价和通信开销较大;②公开认证,为了缓解用户在存储和计算上的压力，云环境下的数据完整性验证机制最好能支持公开认证，允许任意的第三方来替代用户来完成数据完整性验证;③本地无备份认证，用户在本地并没有存储数据副本;④无状态认证，验证过程无需保存任何验证状态;⑤确保用户隐私，用户采用公开验证应能确保用户的数据隐私.

3 数据完整性证明验证机制框架分类

数据完整性验证机制根据是否对数据文件采用了容错预处理分为数据持有性证明 PDP 机制(Provable Data Possession, PDP)和数据可恢复证明 POR 机制(Proofs of Retrievability, POR). 如图 2 所示，本文按不同的关注点进行了分类，PDP 机制能快速判断远程节点上数据是否损坏，更多的注重效率，而 POR 机制不仅能识别数据是否已损坏，且能恢复已损坏的数据. 两种机制有着不同的应用需求，PDP 机制主要用于检测大数据文件的完整性，而 POR 机制则用于重要数据的完整性确保，如压缩文件的压缩表等. 对于这类应用，尽管只损坏很小一部分数据，但却造成了整个数据文件失效.

数据持有性证明 PDP 机制最先运用于网格计算和 P2P 网络中. 用户为了获取更强存储能力，选择将数据备份到远程节点上. Deswarte 等人最先考虑利用 HMAC 哈希函数来实现远程数据的完整性验证，数据存储到远程节点之前，预先计算数据的 MAC 值，并将其保存在本地. 验证时，用户从远程节点上取回数据，并计算此时的 MAC 值，比对证者手中的 MAC 值来判断远程节点上的数据是否是完整性的[11]. 由于需要取回整个数据文件，该机制需要较大计算代价和通信开销,无法满足大规模的应用. 为了支持任意次的完整性检测，Deswarte 等人后来考虑利用 RSA 签名的同态特性来构造 PDP 机制，但该机制需要将整个文件用一个大数表示，导致高昂的计算代价. 之后，Sebé 等人对该算法进行了改进，提出利用分块的方法来减轻计算代价，但采用确定性的验证策略，同样无法验证大的数据文件[12]. 随后，Ateniese 等人提出采用概率性的策略来完成完整性验证，且利用 RSA 签名机制的同态特性,将证据聚集成一个小的值，大大降低了协议的通信开销[10]. 随后, Curtmola 等人考虑了多副本情况下数据完整性的验证，实现了一种满足分布式存储环境的 MR-PDP 机制，但不能支持动态的数据操作[13]. 随着云存储模式的出现, Erway 等人发现，用户可能需要对存储在远程节点上的数据

文件进行动态操作，如插入，修改，删除等，而传统的PDP机制已不能满足这一要求，他们考虑引入动态数据结构来组织数据块集合，实现了支持块级全动态操作[14]. 之后，Wang 等人实现了另外一种支持全动态操作的 PDP 机制，该机制考虑采用 Merkle 哈希树来确保数据块在位置上的正确性，而数据块值则通过 BLS 签名机制来确保[9]. 为了减轻用户的负担，该机制还考虑引入独立的第三方 TPA 来代替用户验证云中数据的完整性. 但采用这种方式存在泄漏隐私的风险. 针对这一缺陷，Wang 等人提出了另一种保护隐私的数据完整性验证机制，该机制通过随机掩码技术，有效地隐藏了云服务器返回证据中的数据信息，使得TPA无法探知数据内容[15].

尽管 PDP 机制能识别远程节点上的数据是否已损坏，但却无法保证数据是否是可用的. Juels 等人最先考虑如何恢复已损坏的数据，提出了一种基于哨兵(sentinel)的 POR 机制，该机制不仅能判断远程节点上的数据是否已损坏，并且可以恢复一定程度的数据失效[16]. 但该机制不支持公开验证，且只能进行有限次验证. 之后，Shacham 等人吸收 Ateniese 等人关于同态验证标签(Homomorphic Verifiable Tags, HVTs)的思想，考虑运用 BLS 短消息签名机制来构造同态验证标签，减少了验证阶段的通信开销[17]. 该方法被证明在任何强威胁模型中是安全的. 由于在初始化阶段引入数据容错预处理，使得动态块级操作在 POR 机制中变得难以实现. Wang 等人提出利用纠错码的线性特征来支持部分动态操作，但无法支持插入操作[18]. Chen 等人对 Wang 的机制进行优化，考虑采用 Cauchy Reed-Solomon 线性编码来进行数据预处理，该方法有效地提高恢复错误的效率，但更新操作需要云服务器重新生成所有的辅助容错信息，导致计算代价较高[19]. 综上所述，设计支持全动态操作的 POR 机制仍是一个开放性的问题.

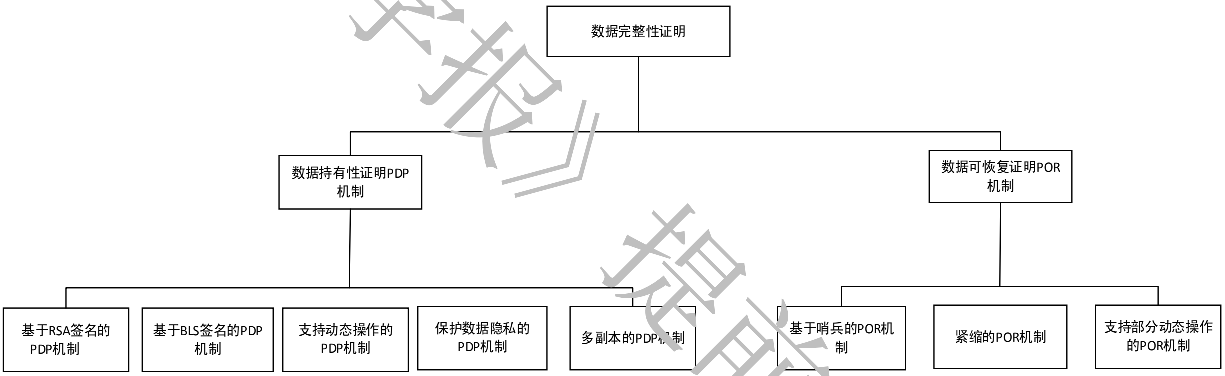


图 2 数据完整性证明机制框架分类

4 数据完整性证明验证机制

根据是否对原数据采用容错预处理技术，数据完整性证明机制可以分为数据持有性证明 PDP 机制和数据可恢复证明 POR 机制，下面将对这两类验证机制进行分析对比.

4.1数据持有性证明PDP机制

现有的 PDP 机制包括：基于 MAC 认证码的 PDP 机制、基于 RSA 签名的 PDP 机制、基于 BLS 签名的 PDP 机制、支持动态操作的 PDP 机制、支持多副本的 PDP 机制及保护隐私的 PDP 机制等.

4.1.1 基于MAC认证码的PDP机制

基于 MAC(Message Authentication Code, MAC)认证码的 PDP 机制(MAC-PDP I)，利用消息认证码 MAC 值作为验证元数据，对远程服务器上存储的数据进行完整性验证[11]. 最为简单的一种方式是，用户首先利用调用函数$MAC_{sk}(\cdot)$为数据文件F生成验证元数据集合$MAC=\{mac_i\}_{1\le i\le n}$，然后将其和文件一起存储到远程节点上，最后将密钥信息sk和文件信息(包括文件名字等)发送给验证者(TPA)；验证时，TPA 将需要验证的数据块的块索引发送给远程服务器，远程服务器返回指定的数据块和与其相对应的MAC值作为响应. TPA 利用私钥计算数据块的MAC值，通过比对返回证据中的MAC值判断文件F是否完整. 每次验证请求时，该方法需要远程服务器返回F中部分数据内容，不但造成了数据文件的隐私泄露，而且需要大的通信开销和计算代价.

*Setup*阶段：

- ➢ 选择两个长度为1024位的素数p和q, 计算RSA模数$N = p \cdot q$
- ➢ 将文件F用整数d表示, 计算验证元数据:$M = a^d \ \ mod \ \ N, a \overset{R}{\leftarrow} Z_N$
- ➢ 将F存储到远程服务器上

*Challenge*阶段：

(1)验证者随机选择$r \overset{R}{\leftarrow} Z_N$，生成验证请求$A = a^r \ \ mod \ \ N$,并发送$A$给远程服务器.

(2)证明者计算证据$B = A^d \bmod N$,发送响应集合B给证明者.

(3)证明者计算$C = M^r \ \ modN$，并判断$C \overset{?}{=} B$.

图 3 Deswarte 等人的基于 RSA 签名 PDP 机制

针对这一问题，Shah 等人提出了另一种基于MAC值的完整性检测机制(MAC-PDP II)[20]. 在用户外包数据之前，用户选取s个随机消息认证码密钥$\{sk_i\}_{1 \le i \le s}$，然后用户对整个文件调用函数$MAC_{sk_i}(F)$求出s个 MAC 值，并将验证元数据$\{sk_i, MAC_i\}_{1 \le i \le s}$发送给 TPA；验证时，TPA 向服务器发出一个挑战请求$\{sk_i\}_{i \overset{R}{\leftarrow} [1,s]}$，远程服务器计算文件的$MAC$值，并将其返回给 TPA. TPA 比对存储在本地的MAC值后，判断数据文件是否是完整的.

基于 MAC 认证码可行的 PDP 机制存在以下固有缺陷：1. 只能进行有限次验证; 2. 验证者必须保存大量辅助验证信息，如密钥信息，验证元数据等; 3. 无法支持动态更新操作，只能用于静态的数据完整性验证.

4.1.2 基于RSA签名的PDP机制

针对基于MAC认证码的PDP机制存在的缺陷，Deswarte 等人考虑利用 RSA 签名机制的同态特性$(a^m)^r = a^{rm} = (a^r)^m (mod \ N)$来构造完整性检测机制，这里$m$为待验证的数据文件，$a$为用户选择的随机数，$r$为每次请求生成的随机数(RSA-PDP I)[11]. 很明显，该机制能进行无限多次完整性验证，同时有效地确保了数据的内容不被验证者察知，但协议需要将整个数据文件F用一个整数d来表示，用于之后的运算. 因此，对较大的数据文件来说，计算代价过高. 该算法的具体实现如图 3 所示.

*Setup*阶段：

- ➢ 选择两个长度为 1024 位的素数p和q, 计算 RSA 模数$N = p \cdot q$
- ➢ 计算欧拉函数$\phi(N) = (p-1)(q-1)$
- ➢ 将文件F用整数d表示，计算验证元数据:$h(d) = d \ mod \ \phi(N)$
- ➢ 将F存储到远程服务器上

*Challenge*阶段：

(1)验证者选择$r \overset{R}{\leftarrow} Z_N$,并发送$r$给证明者.

(2)证明者计算证据$R = r^d \ mod \ N$，并将其返回给验证者.

(3)证明者计算$R' = r^{h(d)} \ mod \ N$，并判断$R' \overset{?}{=} R$

图 4 Filho 等人的基于 RSA 签名 PDP 机制

Filho 等人对 Deswarte 等人提出的协议进行了修改，利用欧拉函数$\phi(N)$的性质来优化验证元数据的生成过程(RSA-PDP II)[21]. 定义一个哈希函数：$h(d) = d \ mod \ \phi(N)$，利用该哈希函数将大的数据文件压缩成小的哈希值后，再用于运算，减轻了计算代价. 为了辅助完成完整性验证，还需定义另一个函数$H: H(d) = r^d \ mod \ N$，该函数具有同态特性，即:$H(d+d') = H(d)H(d')$. 由于$\phi(N)$是群$(Z/nZ)^*$的阶，据此可以构建函数h与函数H的对应关系，即：$H: H(d) = r^d \ mod \ N = r^{d \ mod \ \phi(N)} = r^{h(d)} \ mod \ N$. 利用这一性质可以设计基于 RSA 签名的 PDP 机制，具体实现如图 4 所示. 相比前一机制而言，该 PDP 机制的有效地减少了验证元数据的存储代价和通信开销. 但该机制同样不适合大规模的数据存储. 为了减少认证元数据的大小，Sebé 等人用分块的思想改进 Filho 的工作，从一定程度上减轻了元数据集的大小[12].

Ateniese 等人最先对数据持有性证明 PDP 机制进行了形式化建模，并考虑采用抽样的策略完成完整性验证，有效地减少了计算代价和通信开销(S-PDP III, E-PDP III)[10, 22]. 该机制的创新性主要在于以下三点：第一点，考虑先对文件F进行分块，然后对每个数据块分别计算元认证数据，降低了生成验证元数据的计算代价；第二点，提出了同态认证标签的概念，降低了通信开销. 同态认证标签(Homomorphic Verifiable Tags, HVTs)是一种无法伪造的认证元数据，并且可以将多个数据块的元数据聚集成一个值，解决了基于 RSA 签名 PDP 机制中的证据大小与数据块的数目呈线性增长的问题；

第三，提出了采用抽样的策略对远程节点进行完整性检测．对于存储在远程节点上的数据，若想得到95%以上的损坏识别率，验证者只需从数据分块集合中随机抽取300个以上数据块用于验证[23]．如图 5 所示， Ateniese 实例化了两种基于 RSA 签名的 PDP 机制，S-PDP 机制和 E-PDP 机制．S-PDP 机制在指数知识假定(KEA-r)下是安全的．E-PDP 机制是一种弱化的数据持有性证明机制．但在随后的研究表明，E-PDP 机制并不是安全的[17]．从上可知，Ateniese 等人的研究有效地减轻了用户和服务器端的计算代价，其提出的PDP形式化框架能运用于云存储环境下的数据完整性检测．但是，为了支持外包数据的完整性检测，基于 RSA 的 PDP 机制需要为文件的每个分块都生成一个认证元数据(其大小为 128B)，计算代价和通信开销仍然较大．另外，该机制专门针对静态数据存储，并不支持动态数据块更新，无法满足动态的环境．

4.1.3 基于BLS签名的PDP机制

BLS签名机制是由Boneh等人提出的一种短消息签名机制[24]，相比目前最常用的两种签名机制 RSA 和 DSA 而言，在同等安全条件下(模数的位数为 1024bit)， BLS 签名机制具有更短的签名位数，大约为 160bit(RSA 签名为 1024bit，DSA 签名为 320bit)．另外， BLS 签名机制具有同态特性，可以将多个签名聚集成一个签名．这两点好的特性使得基于 BLS 签名的 PDP 机制可以获得更少的存储代价和更低通信开销来实现．基于 BLS 签名的 PDP 机制是一种公开验证机制，用户可以将繁琐的数据审计任务交由第三方TPA来完成，满足了云存储的轻量级设计要求，具体实现如图 6 所示．另外，相比其他机制而言，该机制具有更小的存储开销和通信开销

S-PDP 验证机制

选取公共参数：f是伪随机函数，π是伪随机置换函数，H为密码学哈希函数

$Setup$阶段：

- 选择两个长度为 1024 位的素数p和q，计算 RSA 模数$N = p \cdot q$
- 选取g作为QR_N生成元，QR_N是模N的二次剩余集
- 生成公私密钥对$pk = (N, g, e), sk = (d, v)$，满足以下关系：
  $e \cdot d \equiv 1 \ mod \ (p-1)(q-1)$
- 选取唯一的文件标识$v \xleftarrow{R} \{0,1\}^{\kappa}$，对文件$F$进行分块：$F = \{b_1, \cdots, b_n\}$，生成认证元数据集合：$\Phi = \{\sigma_i\}_{1 \le i \le n}$，这里$\sigma_i = (h(v||i) \cdot g^{m_i})^d$
- 将F和认证元数据Φ存入远程服务器

$Challenge$阶段：

⑴验证者对c个数据块发出挑战请求$chal$：随机选择二个密钥$k_1 \xleftarrow{R} \{0,1\}^{\kappa}$，$k_2 \xleftarrow{R} \{0,1\}^{\kappa}$，选取随机数$s \xleftarrow{R} (Z/NZ)^*$，计算$g_s = g^s \ mod \ N$．令$chal = (c, k_1, k_2, g_s)$，并将其发送给证明者．

⑵证明者接收到请求$chal$后，首先计算$i_j = \pi_{k_1}(j), a_j = f_{k_2}(j, 1 \le j \le c)$；之后，计算证据$T = \prod_{j=i_1}^{i_c} \sigma_j^{a_j} = \prod_{j=i_1}^{i_c} H(v||j)^{a_j} g^{a_j m_j}$；最后，计算证据$\rho = H(g_s^{a_1 m_{i_1} + \cdots + a_s m_{i_c}})$，将$\{T, \rho\}$发送给证明者．

⑶验证者接收到证据$\{T, \rho\}$后，首先计算$\tau = T^e$；然后对于$1 \le j \le c$，计算$i_j = \pi_{k_1}(j), a_j = f_{k_2}(j), \tau = \frac{\tau}{h(v||i_j)^{a_j}}$；最后，计算$\rho' = H(\tau^s) \ mod \ N$，判断$\rho = \rho'$

E-PDP 机制：

在$Challenge$阶段做以下替换：

第 2 步证据生成部分替换为$T = \prod_{j=i_1}^{i_c} \sigma_j = \prod_{i_1}^{i_c} H(v||j) g^{m_j}$和$\rho = H(g_s^{m_{i_1} + \cdots + m_{i_c}})$

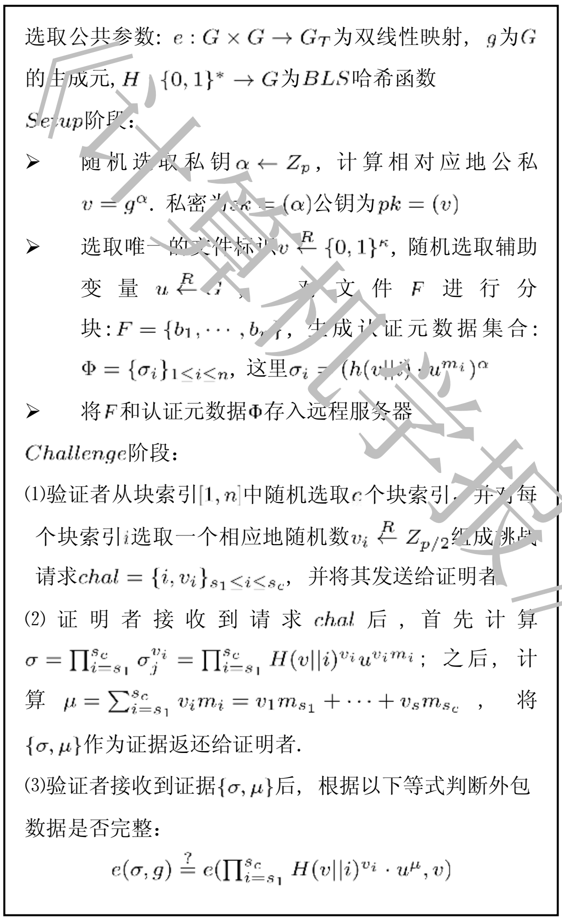
第 3 步$\tau = \frac{\tau}{h(v||i_j)^{a_j}}$替换为$\tau = \frac{\tau}{h(v||i_j)}$

图 5 Ateniese 等人的基于 RSA 签名 PDP 机制

4.1.4 支持动态操作的PDP机制

对于存储在云中的数据，用户可能随时需要更新，而之前提到的 PDP 机制已无法满足这种需求．因为在计算数据块的验证元数据时，数据块的索引作为一部分信息加入了计算，如

$\sigma_i = (H(v|i)u^{m_i})^{sk}$，倘若插入一个新的数据块，将导致该数据块后其它数据块的验证元数据都需重新生成，导致计算代价过高而无法实现．为了解决这一问题，Wang 等人[10]、Erway 等人[14] 考虑采用动态数据结构确保数据块在位置上的正确性，而数据块标签确保数据块在数值上是正确的．前三节已对如何确保数据块值的完整性进行了描述，因此在本小节中，将更多的关注如何支持动态操作．



选取公共参数：$e: G \times G \rightarrow G_T$为双线性映射，$g$为$G$的生成元，$H: \{0,1\}^* \rightarrow G$为$BLS$哈希函数

$Setup$阶段：

- 随机选取私钥$\alpha \leftarrow Z_p$，计算相对应地公私$v = g^{\alpha}$．私密为$sk = (\alpha)$公钥为$pk = (v)$
- 选取唯一的文件标识$v \xleftarrow{R} \{0,1\}^{\kappa}$，随机选取辅助变量$u \xleftarrow{R} G$，对文件$F$进行分块：$F = \{b_1, \cdots, b_n\}$，生成认证元数据集合：$\Phi = \{\sigma_i\}_{1 \le i \le n}$，这里$\sigma_i = (h(v||i) \cdot u^{m_i})^{\alpha}$
- 将F和认证元数据Φ存入远程服务器

$Challenge$阶段：

(1)验证者从块索引$[1, n]$中随机选取c个块索引，并对每个块索引i选取一个相应地随机数$v_i \xleftarrow{R} Z_{p/2}$组成挑战请求$chal = \{i, v_i\}_{s_1 \le i \le s_c}$，并将其发送给证明者

(2) 证明者接收到请求 $chal$ 后，首先计算$\sigma = \prod_{i=s_1}^{s_c} \sigma_j^{v_i} = \prod_{i=s_1}^{s_c} H(v||i)^{v_i} u^{v_i m_i}$；之后，计算 $\mu = \sum_{i=s_1}^{s_c} v_i m_i = v_1 m_{s_1} + \cdots + v_s m_{s_c}$，将$\{\sigma, \mu\}$作为证据返还给证明者．

(3)验证者接收到证据$\{\sigma, \mu\}$后，根据以下等式判断外包数据是否完整：

$$e(\sigma, g) \stackrel{?}{=} e(\prod_{i=s_1}^{s_c} H(v||i)^{v_i} \cdot u^{\mu}, v)$$

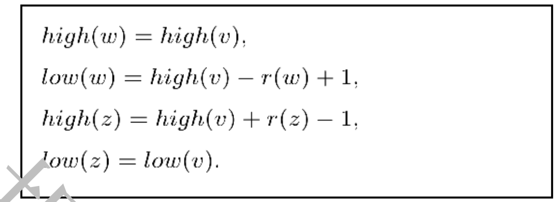
图 6 基于 BLS 签名的 PDP 机制

4.1.4.1 支持部分动态的PDP机制

Ateniese 等人最先考虑如何支持动态操作，通过对文献[10]提出的 PDP 机制做简单的修改，使其支持了部分动态数据操作(DPDP)[25]．但该机制仅能支持数据更新、数据删除、数据追加等操作，而无法支持数据插入．另外，执行删除操作时采用标记$Dblock$来替代之前的数据块位置，造成了一定程度的存储空间浪费．

4.1.4.2 基于跳表的PDP机制

为了解决不能支持插入操作这一问题，Erway 等人考虑引入动态数据结构来支持全动态操作(S-DPDP)[14]．在初始化阶段，该机制首先对数据文件F分块，$F = \{m_1, \cdots, m_n\}$；然后为每个数据块m_i生成数据块标签，$\tau_i = g^{m_i} \ mod \ N$；之后，利用数据块标签生成根节点的哈希值$M_c$，并将其保存在验证者手中．在挑战请求阶段，验证者随机生成挑战请求$chal = \{i, v_i\}_{s_1 \le i \le s_c}$，并将其发送给服务器．服务器接收到挑战请求后，首先计算$M = \sum_{i=s_1}^{s_c} v_i m_i$；之后，返回指定数据块的认证路径$\{\Pi(i)\}_{s_1 \le i \le s_c}$；最后，将$\{M, \{\Pi_i, \tau_i\}_{s_1 \le i \le s_c}\}$作为证据返回给验证者．验证者先利用公式计算$T = \prod_{i=s_1}^{s_c} \tau_i^{v_i}$；之后，判断$T \stackrel{?}{=} g^M \ mod \ N$是否相等，若相等，表示数据内容是否是完整的，否则，认为数据文件已被修改；之后，验证者根据返回认证路径$\{\Pi(i)\}_{s_1 \le i \le s_c}$、标签信息$\{\tau_i\}_{s_1 \le i \le s_c}$及存储在本地认证元数据$M_c$判断数据块在位置上是正确的．这种机制是唯一一种不需要生成私钥的PDP机制，且数据块标签可以在挑战阶段由服务器临时生成．



$$high(w) = high(v),$$
$$low(w) = high(v) - r(w) + 1,$$
$$high(z) = high(v) + r(z) - 1,$$
$$low(z) = low(v).$$

图 8 计算经过当前节点v可访问底层节点的范围．其中，w表示当前节点v右边的节点，z代表当前节点v之下的节点

举例说明：假定认证跳表数据结构如图 7 所示，节点中的数值为节点的$rank$值，左上角的节点为跳表的起始节点w_7，当前节点用v表示，目标节点为v_3，w表示当前节点的右边节点，z表示当前节点的下方节点．根据节点上的$rank$值，可以为每个节点界定一个经过该节点可访问底层节点的范围$[low(v), high(v)]$．例如根节点w_7的$low(w_7)$和$high(w_7)$的值分别为 1, 12．根据当前节点的访问范围，可以计算两个后继节点w、z的访问范围，即:$[low(w), high(w)], [low(z), high(z)]$，计算方法如图 8 所示．如果指定节点的块索引$i \in [low(w), high(w)]$，跟随$rgt(w)$，并设$v = w$，否则跟随$dwn(w)$，即$v = z$，直到找到目标节点$i$．将寻找节点的路径称为访问路径，而与其相反路径称之为认证路径．在图 7 中，目标节点的查找路径是 $(w_7, w_6, w_5, w_3, v_5, v_4, v_3)$，认证路径为$(v_3, v_4, v_5, w_3, w_4, w_5, w_6, w_7)$．

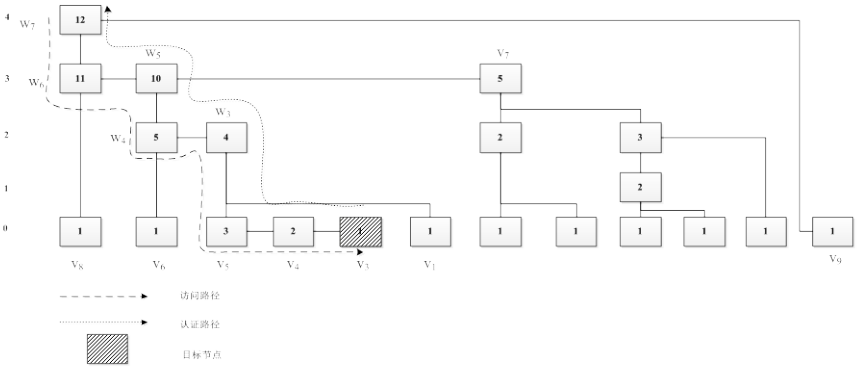


图 7 基于跳表的认证数据结构

动态更新操作时，用户首先发出更新请求 $chal=\{update,m_i',\tau_i'\}$，服务器更新过程分两个阶段进行：第一阶段，解析请求中的更新操作 $update$，倘若是删除操作(D)，则直接删除所在的数据块，倘若是修改操作(M)，更新指定的数据块内容和块签名标签，倘若是插入操作(I)，在指定的位置插入数据块和块标签；第二阶段是，辅助用户更新跳表的根哈希值，服务器返回每个指定节点的认证路径，用户利用认证路径更新根节点的哈希值.

基于跳表的全动态 PDP 机制，是第一种完全支持动态操作的 PDP 机制，但其存在认证路径过长，每次认证过程中需要大量的辅助信息支持，计算代价和通信开销较大.

4.1.4.3 基于Merkle Tree 的PDP机制

Wang 等人提出了另一种支持动态操作的 PDP 机制，即基于 Merkle-Tree 的 PDP 机制，该机制通过 Merkle-tree 结构来确保数据块在位置上的正确性，利用基于 BLS-签名的 PDP 机制来确保数据块内容上完整性(M-DPDP) [10].

与 Eeway 等人提出的基于跳表的 PDP 机制不同的是，数据块标签并没有参与动态结构中根节点哈希值的计算. 初始化阶段分为两步进行，第一步与图 6 中的初始化步骤大致相同，除了块标签计算，为了支持动态操作，需要用 $\sigma_i=(H(m_i)u^{m_i})^{\alpha}$ 替换 $\sigma_i=(H(name||i)u^{m_i})^{\alpha}$；第二步，构造 Merkle 认证哈希树，并计算根哈希值，并将其存储到 TPA 中作为验证元数据. 验证过程与图 6 的验证过程相比，需要做以下修改：1. 在返回证据 $\{\mu,\sigma\}$ 同时，需要返回挑战请求中块索引所对应的辅助认证路径信息 $\{H(m_i),\Omega_i\}_{s_1\le i\le s_c}$；2. 验证证据时，利用辅助认证路径信息计算根节点的哈希值. 只有根节点相同时才进行后续步骤；3. 验证公式需要更改为：$e(\sigma,g)\overset{?}{=}e(\prod_{i=s_1}^{s_c}H(m_i)^{v_i}\cdot u^{\mu},v)$.

Merkle 认证哈希树的数据结构如图 9 所示，Me 叶节点值为数据块的哈希值，对叶节点的访问采用深度优先的方式进行，如图中黑虚线所示. 云服务器为了证明用户的数据是完整的，首先需要构造一条认证路径(如图中蓝色虚线所示)，及其辅助认证信息(Auxiliary Aauthentication Information, AAI)组成证据，返回给 TPA. TPA 根据认证路径和辅助认证信息重新计算根节点的哈希值，比对本地存储的根节点哈希值来判断数据在位置上是否是完整的. 例如，倘若 TPA 发出的挑战请求中选取块索引为 $\{2,7\}$，云服务器需在返回证据 $\{\mu,\sigma\}$ 同时，返回 x_2 认证路径 $\{h(x_2),h_c,h_a\}$、辅助认证路径 $\Omega=\{h(x_1),h_d\}$ 和 x_7 的认证路径 $\{x_7,h_f,h_b\}$、辅助认证路径 $\Omega=\{h(x_8),h_e\}$ 给 TPA，TPA 重新计算 Merkle 哈希树的根节点哈希值，即：根据 $h_c=h(h(x_1)||h(x_2))$，$h_a=h(h_c||h_d)$，$h_f=h(h(x_7)||h(x_8))$，$h_b=h(h_e||h_f)$，最后计算出根节点哈希值 $h_R=h(h_a||h_b)$，对比存储在本地的根哈希值来判断数据文件是否是完整的.

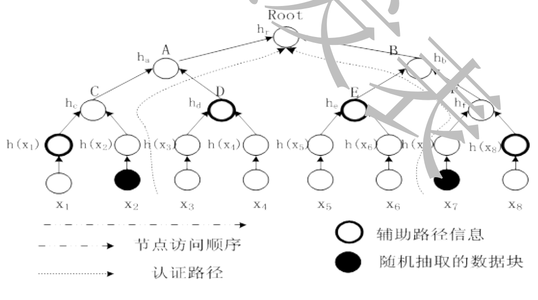


图 9 Merkle 哈希树结构

很明显，采用 Merkle 认证哈希树可以确保数据节点在位置上的完整性. 动态更新操作时，在更新节点同时，需返回辅助认证信息给用户，用

户重新生成根节点的哈希值. 之后, 更新存储在TPA 中的根哈希值. 相比跳表数据结构, Merkle 认证哈希树具有更简单的数据结构.

S-PDP 验证机制

选取公共参数: $e: G \times G \rightarrow G_T$为双线性映射, g为G的生成元, $H:\{0,1\}^* \rightarrow G$ 为BLS 哈希函数, $h: G_T \rightarrow Z_p$将G_T 中的元素均匀的映射到Z_p .

$Setup$阶段:

与图 6 中$Setup$ 阶段一致

$Challenge$阶段:

做以下替换:

第 2 步证据参数的计算替换为

$\mu = r + \gamma \sum_{i=s_1}^{s_c} v_i m_i \bmod p$, 其中

$r \overset{R}{\leftarrow} Z_p, R = e(u,v)^r, \gamma = h(R)$

第 3 步证据检测替换为

验证者计算$R = h(R)$, 判断下式是否成立

$R \cdot e(\sigma^\gamma, g) \overset{?}{=} e(\prod_{i=s_1}^{s_c}(H(v\|i)^{v_i})^\gamma \cdot u^\mu, v)$

图 10 保护隐私的 PDP 机制

4.1.5 保护数据隐私的PDP机制

采用第三方对存储在云中的数据进行完整性验证时, 有可能泄露用户的数据隐私信息. 通过n次挑战请求后, 不怀好意的第三方有可能获取用户存储在云中的文件内容和元验证数据信息. 例如, 假定用户委托第三方对其存储在云中的文件F进行完整性检测, 第三方重复地对$\{i_i, \cdots, i_c\}$位置上的数据进行完整性检测, 每次挑战请求为发送$chal_j = \{i, v_i^j\}, s_1 \le i \le s_c, 1 \le j \le c$给云服务器, 经过$c$次挑战请求后,可以得到以下方程组：

$$\begin{cases} \mu^{(1)} = m_{i_1} v_1^{(1)} + \cdots + m_{i_c} v_c^{(1)}, \\ \vdots \qquad\qquad \cdots \qquad\qquad \vdots \\ \mu^{(c)} = m_{i_1} v_1^{(c)} + \cdots + m_{i_c} v_c^{(c)}. \end{cases}$$

只要上述方程组中的系数行列式不为0, 则可通过高斯消去法计算求得$(m_{i_1}, \cdots, m_{i_c})$的值. 同理, 第三方也可以获得块索引相对应的签名标签$(\sigma_{i_1}, \cdots, \sigma_{i_c})$. 目前大部分基于 BLS 签名的 PDP 机制采用公开验证时, 将有泄露用户数据隐私的风险. Wang 等人建议用随机掩码技术来解决这一问题(PP-PDP)[15, 26]. 该方法的核心思想是, 基于 BLS 签名的 PDP 证据生成过程中, 使用到了线性组合$\mu = \sum_{i=s_1}^{s_c} v_i m_i$来计算证据参数$\mu$, 从而导致了数据隐私的泄露. 因此, 通过引入两个参数r, γ来隐藏μ值, 其中$r \overset{R}{\leftarrow} Z_p$, $\gamma = h(R)^\gamma$, $R = e(u,v)^r$为该次审计特征. 计算μ的公式变为:$\mu' = \gamma \sum_{i=s_1}^{s_c} v_i m_i + r$. 具体实现如图 10 所示.

Wang 等人的保护隐私的PDP机制能防止云存储中采用第三方在审计时泄露隐私的风险, 减轻了用户的审计负担.

4.1.6 支持多副本的PDP机制

采用冗余备份的方式来存储重要的大文件数据, 可以提高数据文件的可靠性(MR-PDP) [13, 27]. 采用冗余备份的方式存储数据文件时, 远程服务器可能并没有按照用户要求的备份数存储数据. 由于存储在远程服务器中数据副本完全一致, 存储服务提供商可能只存储一份或几份数据原文件, 而对外宣称按用户要求存储了多份文件. 因此, 如何确保云中多个数据副本的完整性成为了另一研究方向.

最简单的方式是, 在数据存储到云中之前, 采用多个密钥分别对每一副本进行加密, 然后存储到云服务器上. 进行数据完整性验证时, 每一个副本文件都作为独立的数据文件进行数据完整性验证. 该方法可以确保存放在远程云服务器上的数据是完整的, 但也增加了大量的重复计算和通信开销, 多个内容完全一致的数据文件需要多次验证才能确保其完整性. 为了解决这一问题, Curtmola 等人设计实现了针对多副本的 MR-PDP 机制, 该机制能对所有副本数据的副本进行完整性认证, 而每次验证所带来的开销与对单个文件的进行数据完整性验证所带来的开销大致相同. MR-PDP 机制是在 Ateniese 等人设计的基于 RSA 签名的 PDP 机制上修改而来的. 该机制的具体组成如图 11 所示. MR-PDP 机制包含有五个算法: $KeyGen(\cdot)$, $ReplicaGen(\cdot)$, $TagBlock(\cdot)$, $GenProof(\cdot)$, $CheckProof(\cdot)$. 其中, $KeyGen(\cdot)$和$ReplicaGen(\cdot)$由用户执行, 分别生成需要的密钥对和t个数据副本; $TagBlock(\cdot)$算法为每个数据副本生成元认证数据集合; $GenProof(\cdot)$由服务器执行, 生成完整性证据; $CheckProof(\cdot)$由用户或者 TPA 执行, 通过服务器返回的证据, 验证数据副本的完整性.

MR-PDP 机制同样由两个阶段组成: 初始化

阶段和挑战阶段. 初始化阶段, 用户调用$KeyGen(\cdot)$生成密钥, 利用私钥为数据文件F生成块签名集合$\Phi=\{\sigma_i\}_{1\le i\le n}$. 之后, 调用$ReplicaGen(\cdot)$算法生成$m$个数据副本, 每个数据副本利用随机掩码技术加以区分. 最后将t个数据副本和块签名集合Φ存储到远程服务器中.

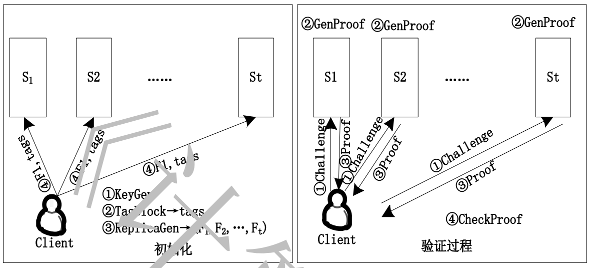


图 11 多副本数据持有性验证机制. 这里, S_i为远程服务器.

在挑战请求阶段, 验证者随机的从m个副本中抽取一个数据副本F_u, 决定对其发起完整性验证. 用户生成挑战请求的过程和 Ateneise 等人设计的 E-PDP[9]一致, 但服务器生成证据参数ρ的过程存在一些差异. 当服务器接到验证请求后, 服务器从数据副本 F_u 中抽取指定数据块$\{b_i+r_{u,i}\}_{s_1\le i\le s_n}$, 然后做计算:$\rho=g_s^{\sum_{s_1\le i\le s_c}(b_i+r_{u,i})}$. 另一个证据参数$T$与之前一致. 在检验证据的过程中, 由于每个副本引入了随机数, 因此需修改图 5 中挑战阶段的第三步, 用$(\frac{T^e}{\prod_{i=s_1}^{s_c}H(v\|i)})^s\overset{?}{=}\rho$替换.

MR-PDP 机制能有效地验证多个副本文件在远程服务器上的完整性, 该机制并不是针对云存储的多副本备份.

4.1.7 其它的PDP机制

Wang 等人利用双线性函数的性质, 设计实现了支持批处理的PDP机制, 该机制能让TPA 同时处理多个审计任务, 优化了审计性能[28, 29]. Wang 等人针对云存储中的共享数据, 利用群签名机制实现了用于验证多用户共享数据的 PDP 验证机制, 该机制能保护用户数据和身份的双重隐私[30].

4.1.8 PDP机制小结

本结对上述的多种 PDP 机制进行简单对比, 主要从以下两方面考虑:

- ■ 计算复杂度: 包括在用户预处理文件的计算代价、在服务器生成证据的计算代价及第三方验证证据的计算代价;
- ■ 通信复杂度:进行完整性时的通信开销

另外, 还考虑了 PDP 机制的其他一些属性包括: 是否支持动态操作、安全模型及文件损坏识别率等.

4.2 数据可恢复证明POR机制

相比 PDP 机制而言, 数据可恢复证明 POR 机制在有效识别文件是否损坏的同时, 能通过容错技术恢复外包数据文件中已出现的错误, 确保文件是可用的. 以下将介绍几种经典的 POR 验证机制.

4.2.1 基于岗哨的POR机制

Juels 等人最先对数据可恢复证明问题进行建模, 提出基于岗哨(sentinel)的 POR 验证机制(SPOR), 该机制主要解决以下两个问题:1. 更有效地识别外包文件中出现的损坏; 2. 能恢复已损坏的数据文件[16]. 针对第一个问题, Juels 等人通过在外包的文件中预先植入一些称之为“岗哨位”的检验数据块, 并在本地存储好这些检验数据块. 对于远程服务器而言, 这些岗哨数据块与数据块是无法区分的. 倘若服务器损坏了数据文件中部分内容, 会相应地损坏到岗哨文件块. 对比存储在本地的检验数据, 能判断远程节点上的数据是否是完整的. 另外, 通过岗哨块损坏的数目可以评估文件中出错的部分在整个文件中所占的概率. 针对第二问题, Juels 等人利用 Reed-Solomon 纠错码对文件进行容错预处理, 使得验证机制可以恢复一部分损坏的数据.

文献[16]表明, 若采用$(223,32)$-Reed-Solomon 纠错码对文件进行分组编码, 文件的大小将增加14%. 若文件损坏识别率高于95%, 文件添加岗哨块后, 大小将增加15%.

基于岗哨的 POR 验证机制存在以下缺点:1. 验证次数是有限的, 取决于岗哨块的数目及每次认证所消耗的数目; 2. 验证机制须在本地存储一定数目的岗哨块数据, 并不是一种轻量级的验证机制.

表 1 多种 PDP 机制对比

Scheme	Server Comp.	Client Comp.	TPA Comp.	Comm.	Dyn.	Model	Probability of detection
MAC-PDP I[11]	$O(1)$	$O(1)$	$O(1)$	$O(1)$	N	-	1
MAC-PDP II[20]	$O(1)$	$O(1)$	$O(1)$	$O(1)$	N	-	1
RSA-PDP I[11]	$O(1)$	$O(1)$	$O(1)$	$O(1)$	N	$standard$	1
RSA-PDP II[21]	$O(1)$	$O(1)$	$O(1)$	$O(1)$	N	$standard$	1
S-PDP[10]	$O(c)$	$O(n)$	$O(c)$	$O(1)$	N	RO	$1-(1-f)^c$
E-PDP10]	$O(c)$	$O(n)$	$O(c)$	$O(1)$	N	RO	$1-(1-f)^c$
BLS-PDP	$O(c)$	$O(n)$	$O(c)$	$O(1)$	N	RO	$1-(1-f)^c$
DPDP[25]	$O(c)$	$O(n)$	$O(c)$	$O(1)$	$Y*$	RO	$1-(1-f)^c$
MR-PDP[13]	$O(c)$	$O(n)$	$O(c)$	$O(1)$	N	RO	$1-(1-f)^c$
M-DPDP[10]	$O(c)+O(log\ n)$	$O(n)+O(log\ n)$	$O(c)$	$O(log\ n)$	Y	RO	$1-(1-f)^c$
PP-PDP[15, 2[illegible]]	$O(c)$	$O(n)$	$O(c)$	$O(1)$	Y	RO	$1-(1-f)^c$
S-DPDP[14]	$O(c)+O(log\ n)$	$O(n)+O(log\ n)$	$O(c)$	$O(log\ n)$	Y	$standard$	$1-(1-f)^c$

c: 随机抽取的数据块数目; n: 文件分块数; Server Comp.: 服务器计算复杂度; Client Comp.: 用户计算复杂度; Comm.: 通信复杂度; Dyn:是否支持动态操作, *表示部分支持; Model:证明安全模型; Probability of detection: 识别概率. f: 数据块损坏的比例.

4.2.2 紧缩的POR机制

为了解决 Juels 等人的不足, Schacham 等人分别提出了针对私有验证(Private Verifiability)和公开验证(Public Verifiability)的数据可恢复 POR 机制(CPOR)[17, 31]. 这两种 POR 机制都具有以下优点: 1. 无状态的验证, 验证者不需要保存验证过程中的验证状态; 2. 任意次验证, 验证者可以对存储在远程节点上的数据发起任意次验证; 3. 通信开销小, 通过借鉴 Ateniese 等人同态验证标签的思想, 有效地将证据缩减为一个较小的值. 无状态和任意次验证需要 POR 机制支持公开验证, 通过公开验证, 用户可以将数据审计任务交由第三方来进行, 减轻了用户的验证负担.

支持私有验证的 POR 机制主要用于企业内部数据完整性验证, 如私有云数据审计. 过程如下: 初始化阶段, 用户先用 Reed-Solomon 纠错码对整个数据文件进行编码; 之后, 用户选择一个随机数$\alpha \xleftarrow{R} Z_p$作为用户的私钥; 然后, 采用公式$\sigma_i = f_k(i) + \alpha m_i \in Z_p$为每个数据块的生成认证元数据集合$\{\sigma_i\}_{1<i<n}$; 最后将数据文件$F$和元数据存入远程服务器上. 挑战阶段与图 6 中的挑战阶段大致相同, 需要修改第 3 步, 用$\sigma \overset{?}{=} \alpha\mu + \sum_{i=s_1}^{s_c} v_i f_k(i)$替换.

公开验证的 POR 机制可以让任意的第三方替代用户来发起对远程节点上数据的完整性检测, 当发现数据的损坏程度小于某一阈值ϵ时, 通过容错机制恢复错误, 大于ϵ则返回给用户数据失效的结论. 相比图 6 中的基于 BLS 签名的 PDP 机制而言, 在进行初始化阶段之前, 需要增加冗余编码数据预处理过程, 使数据文件具有容错能力, 即: 将F分成n个块, 然后对n个块进行分组, 每个组为k个; 之后, 对每组数据块利用 Reed-Solomon 纠错码进行容错编码, 形成新的数据文件$\widetilde{F}$. 而挑战阶段与图 6 中给出的协议完全一致.

文献[17]得出以下结论: 对于 POR 机制而言, 假定ϵ是在允许的错误范围内(比如说, $1,000,000$中出现1次错误, 但通过了 POR 验证), 定义$\omega = 1/\#B + (\rho n)^c/(n-c+1)$, 只要$\epsilon - \omega$是正的可忽略的值, 在$O(n^2 s + (1+\epsilon n^2)(n)/(\epsilon - w))$时间范围内, 通过$O(n/(\varepsilon - w))$次交互, POR 机制的抽取操作能恢复损坏率为$\rho$的数据文件, 这里$B$为挑战请求时随机数$v_i$选取空间, ρ为编码率, c为随机抽取的数据块数目. 举例: 假定 POR 机制参数选定如下, $\rho = 1/2$, $\epsilon = 1/1,000,000$, $B = \{0,1\}^{22}$, $c = 22$, 则 POR 机制能恢复损坏率不超过$1/2$的数据文件. 由此可见, 相比 PDP 机制而言, POR 机制增加了初始化时间, 但也降低了验证代价和通信开销. 另外, 执行抽取恢复操作的人必须是可

信的，因为通过一定次数的验证请求后其将获取部分文件知识.

f 为随机函数，ϕ 为随机置换函数

$Setup$ 阶段：

- ➢ 随机选取一个范德蒙矩阵 A 作为散布矩阵，经过一些列初等变化后，$A=[I|P]$. 生成挑战密钥 k_{chal} 和置换密钥 K_{PRP}
- ➢ 生成编码文件：$G=F\cdot A=\{G^1,\cdots,G^m,G^{m+1},\cdots,G^n\}$，其中 $G^j=(g_1^j,g_2^j,\cdots,g_l^j)^T$，$\{G^1,\cdots,G^m\}$ 为文件 F，$\{G^{m+1},\cdots,G^n\}$ 为冗余信息
- ➢ 生成验证元数据：为每个服务器 $j\in[1,\cdots,n]$，预先生成 t 个验证元数据，每个标签 i 由下式计算得来：
  $v_i^j=\sum_{q=1}^{r}\alpha_i^q * G^(j)[\phi_{k_{prp}^i}(q)],1\le i\le t$
  其中，$\alpha_i=f_{k_{chal}}(i),k_{prp}^i\leftarrow K_{PRP}$
- ➢ 屏蔽冗余信息 $\{G_{m+1},\cdots,G_n\}$：$g_i^j\leftarrow g_i^j+f_{k_j}(s_{ij}),1\le i\le l$，
- ➢ 将 G 存入云中服务器，本地保存和认证元数据 $\{v_i^j\}_1\le j\le n,1\le i\le t$ 和 P

$Challenge$ 阶段：

⑴ 验证者重新生成 $\alpha_i=f_{k_{chal}}(i),k_{prp}^i\leftarrow K_{PRP}$，并将其发送给云服务器

⑵ 服务器计算响应集合：$\{R_i^j=\sum_{q=1}^{r}\alpha_i^q * G^j[\phi_{k_{prp}^i}(q)],1\le j\le n\}$，并将其返回给验证者

⑶ 接收到响应集合后，去除冗余信息的屏蔽值：$R^j\leftarrow R^j-\sum_{q=1}^{r}f_{k_j}(s_{I_q,j})\cdot\alpha_i^q,I_q=\phi_{k_{prp}^i}(q)$，根据本地存储的和 P，判断下式是否成立：

$(R_i^1,\cdots,R_i^m)\cdot P\overset{?}{=}(R_i^{m+1},\cdots,R_i^n)$

⑷不成立，继续比较：$R_i^j\overset{?}{=}v_i^j$，不相等表明服务器 j 上的文件已损坏

动态更新操作：

$F'=(\Delta F_1,\Delta F_2,\cdots,\Delta F_m)$，用 ΔF_i 表示数据内容发生变化的块，当数据没有改变时，$\Delta F_i=0$. 根据初始化阶段的第⑵⑶⑷步，重新生成已发生变化验证元数据 Δv_i^j. 之后更新云中的数据文件. 对于删除操作，令 $\Delta F_i=-F_i$. 该机制不能支持插入操作.

图12 支持动态操作的POR机制

4.2.3 支持动态操作的POR机制

由于POR机制在初始化过程中，由于数据块参与了容错编码,更新数据块的必须同时更新相应地冗余信息，导致计算代价较高. Wang 等人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，该机制可以检测出云存储中已出现的错误，并获取在云服务器上发生错误的位置(DPOR I)[18]. 支持的动态操作包括：修改，删除和追加等三种操作. 具体实现如图12 所示，该机制预先利用 Reed-Solomon 纠错码的生成验证元数据，并将其存储在本地. 验证请求时，服务器利用纠错码的线性特性将多个响应聚集成较小的集合. 验证者通过返回的证据重新生成验证元数据，比对本地存储的验证元数据，判断文件是否正确，不正确时将获得数据出错的服务器. 很明显，该机制是一种私有验证的POR机制，且只能进行有限次的数据完整性检测. Chen 等人考虑采用 Cauchy-Reed-Solomon 纠错码来替换 DPOR I 中的 Reed-Solomon 纠错码，提高抽取阶段的执行效率[19].

4.2.4 POR机制小结

表2 对4. 2 节中对四种POR机制进行了对比，从表中可看出，支持全动态操作仍是POR机制所面临的最大挑战.

5 未来的研究趋势

结合实际应用需求和数据完整性证明机制的研究现状，我们认为未来云存储环境下数据完整性证明机制的研究趋势主要集中以下几点：

(1) 采用多分支路径的方法确保数据完整性

云存储环境下具体应用不再只局限于数据存储和数据备份，更多的是面向动态的服务部署，如何有效地的确保动态环境下数据完整性，将直接影响用户对云存储的体验和云服务提供商的信用. 文献[24]考虑直接在文献[9]提出的静态数据完整性证明机制上做一定程度的修改，使其支持动态数据的完整性证明，但不能支持全部的动态操作，如数据插入等. 之后，文献[10, 14]采用树形的数据结构来重新组织数据，通过增加访存复杂度来获取全动态的支持，但存在认证路径过程和认证路径所需的辅助信息过多等缺陷. 如图 13 所示，可以采用多分支路径代替二叉树，减少节点的认证路径；同时，采用认证路径所特有的信息作为认证路径，减少认证路径的辅助信息，提高了认证效率和减少了通信开销[32].

Tab. 2 多种 POR 机制对比

Scheme	Server Comp.	Client Comp.	TPA Comp.	Comm.	Dyn.	Model	Probability of detection
SPOR[16]	$O(1)$	$O(n^2)$	$O(1)$	$O(1)$	N	RO	$P_m^c \cdot (1-P_f^c)$
CPOR[17,30]	$O(c)$	$O(n^2)$	$O(c)$	$O(1)$	N	RO	$P_m^c \cdot (1-P_f^c)$
DPOR I[18]	$O(1)$	$O(n^2)$	-	$O(1)$	Y^*	$standard$	$P_m^c \cdot (1-P_f^c)$
DPOR II[19]	$O(1)$	$O(n^2)$	-	$O(1)$	Y^*	$standard$	$P_m^c \cdot (1-P_f^c)$

P_m^c:选定的行命中损坏数据概率，P_f^c: 服务器损坏了数据，但通过验证的概率

(2) 保护用户数据隐私的数据完整性验证机制

云存储环境下的数据完整性验证更多的是采用轻量级的数据完整性验证机制，用来支持泛在接入和移动计算. 轻量级的数据完整性验证机制更多的是采用公开的数据完整性证明，用户将数据完整性的验证任务移交给第三方审计方来完成，从而大大的减轻了用户在存储和计算上所需要的开销. 采用文献[15]的方法虽可以确保数据的隐私，但对于 POR 机制而言，采用这种方法将使得验证者无法再通过抽取器去恢复损坏的原文件. 如何设计一种既能保护用户数据隐私又能支持抽取器工作的数据完整性证明机制将成为一个有意义的研究方向.

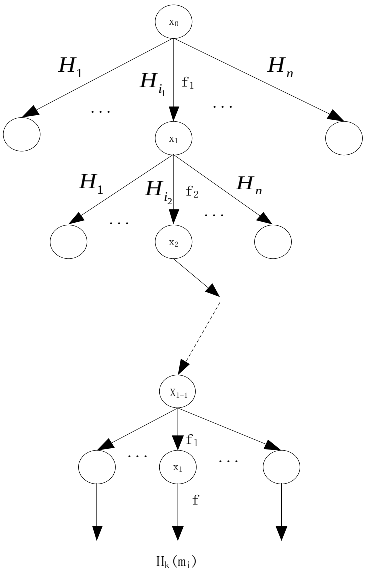


图 13 多分支认证路径

(3) 适合云存储更高效的数据可恢复证明机制

云平台存储的数据通常是大规模数据，为其设计的数据完整性验证机制都是基于抽样的策略，当数据发生位偏转时，用户或可信第三方可能不能及时的发现该小概率事件，如何有效的确保云环境下数据的可恢复性将需要我们需要深入的研究. 文献[16,17]提到的采用经典(n,k,d)Reed-Solomon 纠错码技术[33]在大规模的数据集下并不可行，需要设计更加高效、合理的适合云存储的数据完整性证明机制，而目前在这方面的研究而言还相对较少.

(4) 多副本动态数据完整性验证机制

由于云存储可以为用户提供廉价的存储空间,更多的用户喜欢采用多副本方式存储数据. 如何确保云服务提供商确实按用户所要求的副本数目对原数据进行备份,将是未来需要考虑的一个问题，存储在云中的多个数据副本信息完全一样，采用原有的数据完整性验证机制将无法区分各个副本. 目前最为常用的方法，是采用文献[13]所提到的方法，用户在外包数据之前，预先生成所需要的数据副本，然后为每个数据副本引入随机数加以区分存入云中，但该方法只能并确认云服务提供商自身所提供的存储服务确实符合 SLAs 协议. 另外，在确保数据完整性的情况下，如何同时对多个数据副本进行动态操作也将是值得研究的问题.

(5) 跨云动态数据完整性证明机制

用户可能为了提高数据的可用性和可靠性，可能选择多个云服务提供商来存储数据或部署应用,如何确保多个云服务提供商所存储的数据是完整的将是值得深入研究的问题. 文献[34, 35, 36]对跨云数据完整性证明机制进行建模，但并没有详细的实现步骤，仅仅只是定义了模型，需要设计一种可部署到具体实际应用中的跨云数据完整性证明机制，该机制可以很好支持数据动态操作，以满足更多的应用.

6 结束语

云存储环境下数据完整性验证问题的研究是一个非常活跃的方向．从整体上讲，目前云存储数据完整性验证方面的研究还不成熟，尚未建立起一套完整的理论体系，而且从技术理论的完善到算法的具体应用还有很大的差距．

本文首先回顾了近年来学术界在数据完整性证明研究领域的主要成果，对其进行详细的分类归纳，之后,详细介绍了各种面向不同应用的验证机制的实现原理并加以对比，最后通过分析现有研究指明了未来研究的趋势．

致谢 在此,我们向对本文的工作给予支持和建议的同行,尤其是国防科技大学计算机学院软件所 681 教研室的老师和同学表示感谢．

参考文献

[1] Armbrust M, Fox A, Joseph A D, et al. Above the clouds: A berkeley view of cloud computing. California, USA: EECS Department, University of California, technical report: UCB/EECS-2009-28, 2009.

[2] Mell P and Grance T, The NIST definition of cloud computing. Washington, USA: National Institute of Standards and Technology (NIST), technical report: Special Publication 800-145, 2009.

[3] Julisch K and Hall M, Security and control in the cloud. Information Security Journal: A Global Perspective, 2010, 19(6): 299-309.

[4] Feng Deng-Guo, ZHANG Min, ZHANG Yan, and XU Zhen. Study on Cloud Computing Security. Chinese Journal of Computers, 2011, 22(1): 71-83
(冯登国，et al.，云计算安全研究．软件学报，2011，22(1): 71-83)

[5] Li Ninghui, Li Tiancheng, and Venkatasubramanian S. t-closeness: Privacy beyond k-anonymity and l-diversity. IEEE 23th International Conference on Data Engineering (ICDE2007). Istanbul, Turkey, 2007: 106-115.

[6] Machanavajjhala A, Kifer D, Gehrke J, et al. l-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD), 2007, 1(1): 34-46.

[7] Sweeney L. k-anonymity: A model for protecting privacy. International Journal on Uncertainty Fuzziness and Knowledge based Systems, 2002, 10(5): 557-570.

[8] Ristenpart T, Tromer E, Shacham H, et al. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. Proceedings of the 16th ACM conference on Computer and communications security. Chicago, American, 2009: 199-212.

[9] Wang Qian, Wang Cong, Li Jin, et al. Enabling public verifiability and data dynamics for storage security in cloud computing. Proceedings of 14th European Symposium on Research in Computer Security. Saint Malo, France, 2009: 355-370.

[10] Ateniese G, Burns R, Curtmola R. Provable data possession at untrusted stores. Proceedings of the 14th ACM conference on Computer and communications security (CCS 2007). Alexandria USA, 2007: 598-609.

[11] Deswarte Y, Quisquater J J and Saïdane A. Remote integrity checking. IFIP TC11/WG11.5 Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS). Lausanne, Switzerland, 2004: 1-11.

[12] Sebé F, Domingo F J, Martine B A, et al. Efficient remote data possession checking in critical information infrastructures. Transactions on Knowledge and Data Engineering. 2008, 20(8): 1034-1038.

[13] Curtmola R, Khan O, Burns R, et al. MR-PDP: Multiple-replica provable data possession. Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS'08). Beijing, China, 2008:411－420

[14] Erway C, Kupcu A, Papamanthou C, et al. Dynamic provable data possession. Proceedings of the 16th ACM conference on Computer and communications security (CCS 2009). Chicago, American, 2009: 213-222.

[15] Wang Cong, Wang Qian, Ren Kui, et al. Privacy-preserving public auditing for data storage security in cloud computing. Proceedings of the 29th Conference on Computer Communications (INFOCOM 2010). San Diego, American, 2010: 1-9.

[16] Juels A and Kaliski B S. PORs: Proofs of retrievability for large files. Proceedings of the 14th ACM conference on Computer and communications security. Whistler, Canada, 2007: 584-597.

[17] Shacham H and Waters B. Compact proofs of

retrievability. Proceedings of 14th International Conference on the Theory and Application of Cryptology and Information Security. Melbourne, Australia, 2008: 90-107.

[18] Wang Cong, Wang Qian, Ren Kui, et al. Ensuring data storage security in cloud computing. Proceedings of 17th International Workshop on Quality of Service (IWQOS). Charleston, American, 2009:1-9.

[19] Chen B and Curtmola R. Robust dynamic remote data checking for public clouds. Proceedings of the 2012 ACM conference on Computer and communications security (CCS 2012). New York, USA, 2012:1043-1045.

[20] Shah M A, Swaminathan R, Baker M, et al. Privacy-Preserving audit and extraction of digital contents. California, USA: HP labs, technical reports: HPL-2008-32R1, 2008

[21] Filho D, and Barreto P S, Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, 2006, 1(1):150-159.

[22] Ateniese G, Burns R, Curtmola R, et al., Remote data checking using provable data possession. ACM Transactions on Information and System Security (TISSEC), 2011, 14(1): 1-34.

[23] Curtmola R, Khan O and Burns R. Robust remote data checking. Proceedings of the 4th ACM international workshop on Storage security and survivability. Alexandria, American, 2008: 63-68.

[24] Boneh D, Lynn B and Shacham H. Short signatures from the Weil pairing. Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security. Gold Coast, Australia, 2001: 514-532.

[25] Ateniese G, Pietro R D, Mancini L, et al. Scalable and efficient provable data possession. Proceedings of the 4th international conference on Security and privacy in communication networks. Istanbul, Turkey, 2008: 1-10.

[26] Wang Cong, Wang Qian, Ren Kui, et al. Privacy-preserving public auditing for secure cloud storage. IEEE Transactions on Computers, 2013, 62(2): 362-375.

[27] Barsoum A F and Hasan M A. On verifying dynamic multiple data copies over cloud servers. IACR Cryptology ePrint Archive, 2011, 1(1): 447-455.

[28] Wang Cong, Wang Qian, Ren Kui, et al. Toward secure and dependable storage services in cloud computing. IEEE Transactions on Services Computing, 2012, 5(2):220-232.

[29] Wang Boyang, Li Baochun and Li Hui. Oruta: Privacy-preserving public auditing for shared data in the cloud. IEEE 5th International Conference on Cloud Computing (CLOUD). Hawaii, USA, 2012:295-302.

[30] Wang Qian, Ren Kui, Yu Shucheng, et al. Dependable and secure sensor data storage with dynamic integrity assurance. Transactions on Sensor Networks, 2011, 8(1): 954-962.

[31] Shacham H and Waters B. Compact proofs of retrievability. Journal of cryptology, 2013, 26(3): p. 442-483.

[32] Boneh D, Mironov L and Shoup V. A secure signature scheme from bilinear maps. Proceedings of the Cryptographers’ Track at the RSA Conference. San Francisco, USA, 2003: 98-110.

[33] Wicker S B and Bhargava V K. Reed-Solomon codes and their applications. New York, USA: IEEE Press. 1999.

[34] Zhu Yan, Hu Zexing, Wang Huaixi, et al. A collaborative framework for privacy protection in online social networks. Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom). Chicago, American, 2010: 1-10.

[35] Zhu Yan, Hu Hongxi, Ahn G j, et al. Collaborative integrity verification in hybrid clouds. Proceedings of the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom). Orlando, American, 2011: 191-200.

[36] Zhu Yan, Wang Huaixi, Hu Zexing, et al. Efficient provable data possession for hybrid clouds. Proceedings of the 17th ACM conference on Computer and communications security (CCS 2010). New York, USA, 2010:756-758.

TAN Shuang, born in 1984, Ph.D. His research interests include Cloud Computing and Network Security, etc.

JIA Yan, born in 1961, professor, Ph.D. His research interests include data warehouse and Cloud security, etc.

HAN Wei-hong, born in 1970, Ph.D. His research interests include data analysis and data mining, etc.

Background

With the cloud computing model is widely used, more and more security issues presented in front of people, like Amazon, Google and other international renowned cloud computing service providers have burst safety issues, so the credibility and economic have suffered huge losses. Currently, Cloud Storage has been widespread deployment and implementation in Cloud Computing, but it does not ensure that the user data stored in the cloud in the next period of time is still intact and effective. Provable data integrity models, as one key technology of keeping the integrity of data in cloud storage, have been extensively researched by many papers. Hence, there is an increasing demand to classify and analyze the existing typical model of provable data integrity in cloud storage.

This research is supported by the National 863 Program of China (No. 2012AA01A401, 2012AA01A402) and The National Basic Research Program of China (No.2013CB329601, No.2013CB329602).