基于哈希树的云存储完整性检测算法

颜湘涛 李益发
(信息工程大学 郑州 450002)

摘 要 云存储服务使得用户无需大量软硬件投入即可享受大容量、高规格的存储服务,但是同时也带来了云环境下数据机密性、完整性和可用性等安全问题。针对云存储中的完整性问题,利用哈希树结构和大数模运算,提出了一种新的基于哈希树结构的数据完整性检测算法。分析结果表明,该算法使得用户只需在常量的存储、计算和网络资源下就能高概率地、正确地检测远端服务器数据文件的完整性,且支持文件数据的动态更新。

关键词 云计算,云存储,数据安全,数据完整性
中图法分类号 TP309 **文献标识码** A

Integrity Checking Algorithm Based on Hash Tree for Cloud Storage

YAN Xiang-tao LI Yi-fa
(Information Engineering University, Zhengzhou 450002, China)

Abstract Cloud storage services enable user to enjoy high-capacity and high-quality storage with less overhead, but it also brings many potential threats, for example, data integrality, data availability and so on. This paper proposed a new integrality checking algorithm. Follow the analysis, this new algorithm, based on hash tree and big integer operation, can check mass file's integrality in less storage, compute and network resource. In addition, it also supports some data dynamic update.

Keywords Cloud computing, Cloud storage, Data security, Data integrality

1 引言

信息技术的高速发展对数据存储带了爆炸式的增长,这对存储系统的容量、性能和扩展性等提出了更高的要求。虽然目前存储设备的成本不断下降,但是数据管理的复杂度和维护费用却越来越高,自行运维一个满足正常存储需求的存储系统的代价越来越大。相对于自建的数据存储架构,云计算提供的存储服务在性能和成本上具有诸多优势,它让用户无需投入大量的存储硬件及软件资源,即可享受大容量、高规格的存储服务[1]。但是由于在云环境中用户的数据脱离了自己的安全管控,引发了机密性、完整性和可用性等一系列的安全问题,这其中最基本的就是数据的完整性问题[2]。

为了弥补云环境带来的安全缺陷,人们提出了许多方案来保证数据的完整性,典型的有:Mykletun 等提出的基于 RSA 的压缩签名方法[3]、Ateniese 等提出的 PDP(Proofs of Data Possession)方法[4]和 Juels 等提出的 POR(Proofs of Retrievability)方法[5]。Mykletun 等的方法采用可压缩的 RSA 数字签名解决了数据完整性验证的问题。PDP 方法利用 RSA 同态标签解决了数据的归属验证问题。POR 方法利用抽样检查和纠错码技术,解决了数据的归属验证问题和有效性证明。这 3 个方法在一定程度上解决了云存储中的有效性和完整性问题,但是效率不高,而且不能很好地支持数据动态更新等云存储操作。为了解决这些问题,人们提出了许多改进方案[6-12],典型的有:Wang Cong 等提出的基于分布式环境下的挑战应答协议[8]、Erway 等和 Wang Qian 等各提出的 PDP 改进方案[9,10]。Wang Cong 等的方案能够在一定程度上有效地判定数据完整性,但是只支持部分更新操作。Erway 等和 Wang Qian 等的方案能够判定数据完整性,且支持完全数据更新,但是该方案效率低下。

本文针对云存储存在的完整性检测问题,在一般哈希树结构的基础上,结合大数模乘运算,提出一种新的树形完整性检测结构——IC-树(Integrity Checking tree)。基于 IC-树和 Seny 等提出的云存储框架[13],设计了一种新的云存储数据完整性检测算法。

2 方案描述

2.1 主体框架

本文使用 Seny 等提出的云存储框架[13],其结构如图 1 所示。在该框架中,参与方包括:

(1) 数据拥有者(Data Owner, DO)。DO 创建数据,利用 CSSP(Cloud Storage Service Provider,云存储服务商)提供的接口执行各种数据操作。由于数据脱离了物理控制,数据操

到稿日期:2012-02-15 返修日期:2012-06-02 本文受 863 新概念高效能计算机体系结构及系统研究开发(2009aa012201)项目,现代通信实验室预研项目(9140C1103040902)资助。

颜湘涛 男,硕士,主要研究方向为信息安全,E-mail:taoexcellent@163.com;**李益发** 男,博士后,副教授,主要研究方向为公钥密码和协议安全分析。

作也是由云端代为执行，因此 CSSP 和 TTP(Trusted Third Party，可信第三方)必须提供足够的机制让数据拥有者相信自己的数据和隐私安全。

(2) 数据请求者(Data Requestor，DR)。DR 可以是 DO，也可以不是 DO。当 DR 不是 DO 时，DR 必须先通过 DO 的验证，然后持 DO 颁发的相应证据向云服务商读取 DO 的数据。通常 DR 还能够对下载的数据进行完整性、有效性等检测。

(3) 云存储服务商(Cloud Storage Service Provider，CSSP)。CSSP 构建云存储系统，向 DO 和 DR 提供存储服务。为了达到服务目标，CSSP 必须采取相应的安全措施来保证自己服务的安全性、稳定性和可靠性。

(4) 可信第三方(可选)(Trusted Third Party，TTP)。TTP 在安全领域具备很好的公信力、丰富的经验和完善的技术。在 CSSP 提供服务前，TTP 采用标准化的方法对 CSSP 的服务进行安全评估和认证。在云服务运行过程中，实时监控 CSSP 运维情况。

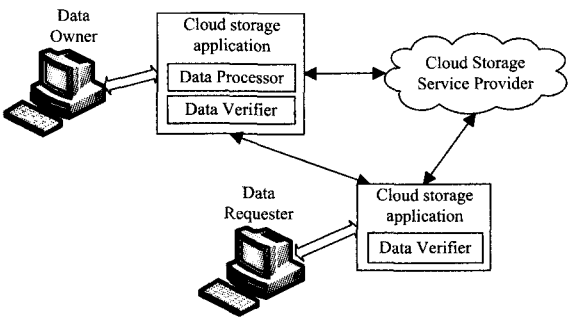

图 1　云存储个人用户框架

本文不讨论 DR、CSSP 和 TTP 的具体实现，也不关注框架中数据加密和索引架构等其它功能，而只关注普通 DO 用户的数据完整性检测功能的实现。对于 DO，完整性检测主要与两个模块相关，DP(Data Processor，数据处理)模块、DV(Data Verifier，数据验证)模块。

(1) DP 模块：DP 模块维护一个 IC-树结构。当用户 Alice 想要上传数据到云中时，DV 模块计算完整性检测需要的验证值，并将该验证值插入树中。当用户进行数据更新时，动态更新 IC-树中的验证值，以保证验证值与数据文件的同步。

(2) DV 模块：对于 DO，DV 模块主要使用主密钥和 IC-树，同云存储服务商交互，检测数据的完整性。而对于 DR，DV 模块利用数据对应的 DO 颁发的证据来实现上述功能。

2.2　敌手模型

在算法中，假设敌手知悉检测方案过程的一切计算细节，可以部分或者全部获取云中存储的用户数据，能够对用户的数据进行修改而不被云服务商察觉。同时，假设云服务商是不可信的。它可能会对用户的数据进行恶意修改、删除等操作，当数据存储出现错误，云服务商会尽可能掩盖并推卸自己的责任。另外，假设用户本地存储的私密信息是只有用户本人知晓，而且在进行验证时，每个用户与云服务商之间的信道是安全可信的。

2.3　设计目标

(1) 正确性。算法必须能够正确地验证数据的完整性。如果 CSSP 没有完整地存储用户的数据或者不按照方案执行方案，那么 CSSP 通过检测的概率应该是可忽略的。

(2) 可靠性。算法必须提供严格的证明，以保证在敌手模型下安全可靠。

(3) 高效。算法在满足完整性检测的同时应该尽可能地减少对存储服务性能的影响，方案需要的计算、存储和网络交互必须尽量少。

(4) 支持数据动态更新。云存储方案服务对象广泛，方案设计应尽可能支持数据的动态修改、删除、追加等操作。

3　IC-树

针对云计算中数据应用的特点，在文献[14]的哈希树结构的基础上，构建了一个用于完整性检测的树形结构——IC-树(Integrity Checking Tree)。

一个 D 层 IC-树的结构和操作方法如下：

结构：对于 $i\in\{1,\cdots,D\}$，用 p_i 表示第 i 个素数(例如：$p_1=2,p_2=3,p_3=5,\cdots$)，IC-树每层节点的子节点数目为 p_i。即第一层每个节点下有 $p_1=2$ 个节点；第二层的每个节点下有 $p_2=3$ 个节点；依此类推。每个节点上存储一组($index$，$value$，$flag$)值。$index$ 为文件的索引号，随机选取、不重复，其值小于前 D 个素数乘积值；$value$ 为此文件的验证值，用来验证数据完整性；$flag$ 为节点有效标志。

查询节点：设 $i=1$，$index$ 是要查询的文件的索引值，N 为节点变量，用 $N\rightarrow j$ 表示 N 节点从左到右第 j 个子节点。首先让 N 等于根节点：

1. 计算 $j_i=(index) \bmod p_i$，如果 $N\rightarrow j_i$ 节点无效(该节点 $flag$=false)，则执行第 3 步操作。

2. 如果 $N\rightarrow j_i$ 有效，则读取该节点的索引值，将它和 $index$ 进行比较。如果相等，则结束查询并返回查找成功和该节点的值；如果不相等，转第 3 步操作。

3. $N=N\rightarrow j_i$，$i=i+1$。如果 $1\leqslant i\leqslant D$，执行第 1 步操作，否则结束查询并返回失败。

插入节点：设 $i=1$，$index$ 是要增加的文件的索引值，N 为节点变量，用 $N\rightarrow j$ 表示 N 节点从左到右第 j 个子节点。首先让 N 等于根节点：

1. 计算 $j_i=(index) \bmod p_i$，如果 $N\rightarrow j_i$ 节点无效(该节点 $flag$=false)，则将值插入到该节点，并返回成功；否则转第 2 步操作。

2. $N=N\rightarrow j_i$，$i=i+1$。如果 $1\leqslant i\leqslant d$，则执行第 1 步操作；否则结束并返回失败。

更新节点：查询到该节点，更新节点 $index$ 或者 $value$ 或者 $flag$ 值。

删除节点：查询到该节点，删除节点 $index$ 和 $value$ 值，将节点的 $flag$ 置为 false。

4　完整性检测算法

算法的目标是使用相对较小的存储和计算，验证存放在云中的大量数据对象的完整性。基本思想是用户端在初次使用时建立一个 IC-树，对于任何要上传到云中的数据文件给予一个随机的索引，并根据索引在 IC-树中的相应节点存储该数据文件的校验值。任何数据文件的改变(例如更新、删除等)都直接导致其在 IC-树上的相应节点的改变(例如节点值的更新，节点删除等)，通过 IC-树和验证协议，一次验证单个文件或者多个文件的完整性。

4.1 表达式定义

$Pad(m,l)\rightarrow m_1\cdots m_n$：$m$ 为数据文件，l 为分块长度。将数据文件 m 分解成 l 比特长的数据块，如果最后一块数据不足 l 比特长，那么在后面填充 0，将其填充至 l 比特长。假设分解完毕后 $m=m_1\cdots m_n$，$i\in\{1,\cdots,n\}$。

$GenKey(1^l)\rightarrow(p,q,N,r_0,r_1,r)$：随机选取 $2l$ 比特的素数 p，l 比特的素数 q,s,t，l 比特的随机数数 r_0，r_1 和 $r_2(r_2\neq 0 \bmod q)$。计算 $N=p^2q^2st$，$r=r_0+r_1pq+r_2p^2$。其中 N 作为验证公钥发送给 CSSP，(p,q,r,r_0,r_1) 作为验证私钥存储在本地。

$GenCheckValue(m,r_0,p,q)\rightarrow(f_0(m),f_1(m))$：输入文件 $m=m_1\cdots m_n$ 和文件验证私钥中的 r_0,p,q，按照如下公式计算：

$$f_0(m)=\sum_{i=1}^{n}r_0^i m_i(\bmod p^2)$$

$$f_1(m)=pq\sum_{i=1}^{n}ir_0^{i-1}m_i(\bmod p^2)$$

得到验证值$(f_0(m),f_1(m))$。

$GenChallenge(k_1,r,p,q)\rightarrow k$：输入随机数 k_1 和密钥中的 (r,p,q)，计算挑战值：$k=k_1pq+r(\bmod N)$。

$GenAnswer(k,m,N)\rightarrow y(m)$：输入挑战值 k 后，服务器按照如下公式计算应答值 y：

$$y(m)=\sum_{i=1}^{n}k^i m_i(\bmod N)$$

$VerifyIntegrity(f_0(m),f_1(m),k_1,r_1,y(m))\rightarrow\{\text{true},\text{false}\}$：$y$ 为服务器端的应答值，用户计算验证值：

$$y_0=y(m)(\bmod p^2)$$

$$y_1=f_0(m)+(k_1+r_1)f_1(m)(\bmod p^2)$$

如果 y_1 等于 y_0，则验证成功，输出 true；如果不等，则验证失败，输出 false。

4.2 详细算法

4.2.1 数据处理

假设 l 为安全参数，当用户端第一次初始化运行的时候，建立一个空的 IC-树，然后调用 $GenKey(1^l)$ 生成验证密钥值 (p,q,N,r_0,r_1,r)，公布 N 给服务器。(p,q,r,r_0,r_1) 由用户本地秘密存储。

当有文件 m 需要上传到服务器时，客户端首先随机选取文件索引值 $index$（$index$ 值不超过 IC-树的索引分辨范围），然后调用 $Pad(m,l)$ 将文件 m 分为 l 长度的块（如果最后一块不足 l 比特则用 0 填充），调用 $GenCheckValue(m,r_0,p,q)$ 对文件内容做预计算，得到验证值 $value=(f_0(m),f_1(m))$，将文件元信息$(index,value,\text{true})$插入到 IC-树中，然后将文件 m 及其索引值一并上传。

4.2.2 单个文件完整性检测协议

当客户端需要验证索引值为 $index$ 的文件的完整性时，首先通过 $index$ 在树中查找到文件 $m=m_1\cdots m_n$，然后按图 2 执行验证协议：

1. 选取随机数 $k_1(k_1\neq 0)$，计算 $k=GenChallenge(k_1,r,p,q)$，将 k 和欲验证文件的索引发送给服务端作为验证挑战值。

2. 服务端计算应答值：$y(m)=GenAnswer(k,m,N)$，将 $y(m)$ 返回给用户。

3. 用户端验证应答值 $y(m)$，计算 $VerifyIntegrity(f_0(m),f_1(m),k_1,r_1,y(m))$，如果验证结果为 true，那么判定文件是完整的，否则判定文件是不完整的。

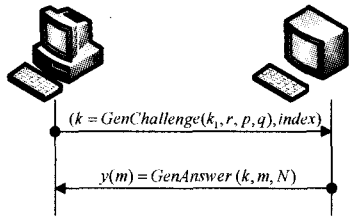

图 2 单个文件验证协议

4.2.3 批量文件完整性检测协议

当用户端需要验证多个文件的完整性时，假设用户端通过某种约定的树遍历顺序（例如前序遍历、后序遍历等），将这些需同时验证的数据文件串联成一个文件。假设总共有 h 个文件，遍历顺序中第 i 个文件有 n_i 块分区，第 i 个文件第 j 块表示为 m_{ij}，全部文件总块数为 n_{all}，则所有的文件可以表示为一个整体文件：

$$M_{all}=m_{11}\parallel\cdots\parallel m_{1n_1}\parallel\cdots\parallel m_{h1}\parallel\cdots\parallel m_{hn_h}$$

对于 $1\leqslant i\leqslant n_{all}$，假设 M_i 为整体文件 $M_{n_{all}}$ 的第 i 块，则 $M_{n_{all}}$ 又可表示为：

$$M_{all}=M_1\parallel\cdots\parallel M_{n_{all}}$$

全部文件的验证过程如下：

1. 随机选取 k_1，计算 $k=GenChallenge(k_1,r,p,q)=k_1pq+r(\bmod N)$，将 k 和欲验证文件的索引序列发送给服务端作为验证挑战值。

2. 服务端对 M_{all} 进行计算：

$$Y(M_{all})=GenAnswer(k,M_{all},N)=\sum_{i=1}^{n_{all}}k^iM_i(\bmod N)$$

将 $Y(M_{all})$ 返回给用户。

3. 假设$(f_{i0}(m_i),f_{i1}(m_i))$为文件 m_i 的验证值，则用户按如下方式计算所有文件的验证值：

$$F_0(M_{n_{all}})=\sum_{i=1}^{h}r_0^{\sum_{j=1}^{i-1}n_j}f_{i0}(m_i)(\bmod p^2)$$

$$F_1(M_{n_{all}})=\sum_{i=1}^{h}\{r_0^{\sum_{j=1}^{i-1}n_j}f_{i1}(m_i)+pq(\sum_{j=1}^{i-1}n_j)r_0^{-1+\sum_{j=1}^{i-1}n_j}\times f_{i0}(m_i)\}(\bmod p^2)$$

然后计算验证结果 $VerifyIntegrity(F_0(M_{n_{all}}),F_1(M_{n_{all}}),k_1,r_1,Y(m))$，如果验证结果为 true，那么判定这批文件现在都是完整的，否则判定此时这些文件不全或不完整。

4.3 动态更新

动态更新是云存储系统的基本操作之一。假设 m 为数据文件，且 $m=m_1\cdots m_n$，则对 m 文件的动态更新后的完整性检测改动如下：

①块修改

假设任意的 $i\in\{1,\cdots,n\}$，假设需将第 i 块数据 m_i 修改为 m_i'，只需对验证值进行如下更新即可：

$$f_0^{update}(m)=f_0(m)-r_0^i m_i+r_0^i m_i'(\bmod p^2)$$

$$f_1^{update}(m)=f_1(m)-pqir_0^{i-1}m_i+pqir_0^{i-1}m_i'(\bmod p^2)$$

1. 追加

假设需将第 $n+1$ 块数据 m_{n+1} 修改为 m'_{n+1}，只需对验证值进行如下更新即可：

$$f_0^{update}(m)=f_0(m)+r_0^{n+1}m_{n+1}(\bmod p^2)$$

$$f_1^{update}(m)=f_1(m)+pq(n+1)r_0^n m_{n+1}(\bmod p^2)$$

2. 块删除

假设需将第 $n+1$ 块数据 m_{n+1} 删除，只需对验证值进行

如下更新即可：

$$f_0^{update}(m)=f_0(m)-r_0^{n+1}m_{n+1}(\text{mod } p^2)$$

$$f_1^{update}(m)=f_1(m)-(n+1)pqr_0^n m_{n+1}(\text{mod } p^2)$$

5 安全分析

在敌手模型中，敌手可能对服务器上的数据进行篡改，服务器端为了自己的利益可能对用户存在欺骗。所以证明方案的完备性和验证的不可伪造性来说明算法的安全性。假设数据的传输信道无错误，由此可得：

定理 1(完备性) 如果 CSSP 是诚实的，那么该完整性检测算法是一个错误概率为$\frac{1}{p^2}$的偏否的 Monte Carlo 完整性判定算法。

证明：

①单个文件完整性验证

当用户端收到服务器端回送的应答值 $y(m)$时，

$$\begin{aligned}y_0&=y(m)(\text{mod } p^2)\\&=km_1+k^2m_2+\cdots+k^nm_n(\text{mod } p^2)\\&=\sum_{i=1}^{n}(k_1pq+r_0+r_1pq+r_2p^2)^im_i(\text{mod } p^2)\\&=\sum_{i=1}^{n}r_0^im_i(\text{mod } p^2)+(k_1+r_1)pq\sum_{i=1}^{n}ir_0^{i-1}m_i(\text{mod } p^2)\\&=f_0(m)+(k_1+r_1)f_2(m)(\text{mod } p^2)\end{aligned}$$

由上述过程可知，如果服务端完整存储了用户的数据并按照方案计算哈希值，那么服务器端可以通过检测方案。如果服务端存储的用户数据被恶意篡改，由 k_1 和 r_1 取值的随机性可知，服务器端依然能够通过检测方案的概率为$\frac{1}{p^2}$。

②所有文件整体完整性验证

当用户端收到服务器端回送的应答值 $Y(m)$时，验证过程如下：

$$\begin{aligned}y_0&=Y(M_{all})(\text{mod } p^2)\\&=kM_1+k^2M_2+\cdots+k^{n_{all}}M_{n_{all}}(\text{mod } p^2)\\&=\sum_{i=1}^{n_{all}}(k_1pq+r_0+r_1pq+r_2p^2)^iM_i(\text{mod } p^2)\\&=\sum_{i=1}^{h}r_0^{n_0+\cdots+n_i}f_{i0}(m_i)(\text{mod } p^2)+(k_1+r_1)\sum_{i=1}^{h}\{r_0^{\sum_{j=1}^{i-1}n_j}f_{i1}(m_i)+pq(\sum_{j=1}^{i-1}n_j)r_0^{-1+\sum_{j=1}^{i-1}n_j}\times f_{i0}(m_i)\}(\text{mod } p^2)\\&=F_0(M_{n_{all}})+(k_1+r_1)F_1(M_{n_{all}})\end{aligned}$$

由上述过程可知，检测方案的错误概率为$\frac{1}{p^2}$。

定理 2(不可伪造性) 假设安全参数为 l，CSSP 在拥有 Q 个正确的挑战/应答值对的情况下伪造一个正确的应答值的概率是$\frac{1}{2^{2l}}$。

证明：假设 CSSP 拥有 Q 个正确的挑战/应答值对，下面考虑 CSSP 伪造正确的应答值的概率。

正确的验证值为：

$$\begin{aligned}y(m)&=f_0(m)+(k_1+r_1)f_2(m)(\text{mod } p^2)\\&=\sum_{i=1}^{n}r_0^im_i(\text{mod } p^2)+(k_1+r_1)pq\sum_{i=1}^{n}ir_0^{i-1}m_i(\text{mod } p^2)\end{aligned}$$

而服务器从公开值和挑战/应答值对中可能计算得到的信息有：

① pq

假设 k 和 k'为两个不同的挑战值：

$$k=k_1pq+r(\text{mod } N)$$

$$k'=k_2pq+r(\text{mod } N)$$

那么 CSSP 可能通过计算 $\gcd(k-k',N)$来得到 pq。

② $r_0+r_2p^2(\text{mod } pq)$

因为 $r=r_0+r_1pq+r_2p^2$，CSSP 在知道 pq 后，能计算出：

$$\begin{aligned}k(\text{mod } pq)&=k_1pq+r_0+r_1pq+r_2p^2(\text{mod } pq)\\&=r_0+r_2p^2(\text{mod } pq)\end{aligned}$$

③ $\sum_{i=1}^{n}r_0^im_i(\text{mod } pq)$

对于正确的验证值 $y(m)$，计算：

$$\begin{aligned}y(m)(\text{mod } pq)&=\sum_{i=0}^{n}r_0^im_i(\text{mod } pq)+(k_1+r_1)pq\sum_{i=1}^{n}ir_0^{i-1}m_i(\text{mod } pq)\\&=\sum_{i=1}^{n}r_0^im_i(\text{mod } pq)\end{aligned}$$

CSSP 可能的伪造方式：

① 重放。因为验证值中 k_1 是随机选取的，所以简单重放以前的正确验证值的伪造方法的成功概率为$\frac{1}{p^2}<\frac{1}{2^{4l}}$。

② 随机选取应答值。因为模数为 p^2，所以随机选取的应答值验证通过的概率为$\frac{1}{p^2}<\frac{1}{2^{4l}}$。

③ 应答值的伪造。

由大数分解的困难性可知，已知 pq 和 $N=p^2q^2$ 分解得到 p 或者 q 的概率是可忽略的，进而认为 CSSP 得到 p^2 的概率也是可忽略的。

从验证值的表达式可知，如果不知道 r_0，那么 CSSP 将无法计算$\sum_{i=1}^{n}r_0^im_i(\text{mod } N)$，也无法计算$\sum_{i=1}^{n}ir_0^{i-1}m_i(\text{mod } N)$，进而不能伪造应答值。如果知道 r_0，CSSP 首次收到数据文件时，计算：$\sum_{i=1}^{n}r_0^im_i(\text{mod } N)$和$\sum_{i=1}^{n}ir_0^{i-1}m_i(\text{mod } N)$。在接收到挑战值 k 时，CSSP 计算 $k^2(\text{mod } N)=(k_1pq+r)^2(\text{mod } N)=r_0^2+2r_0(k_1+r_1)pq(\text{mod } N)$。因为 r_0 已知，所以可能计算得到 $(k_1+r_1)pq(\text{mod } N)$，进而可以计算得到正确的应答值 $y(m)$。因此伪造的关键在于能否计算得到 r_0。

所有含 r_0 的已知值的表达式都是加法和乘法的混合结构，其中最简单的是 $r_0+r_2p^2(\text{mod } pq)$。由 p 和 q 的选取可知 $p>q$，只有当 $r_2\neq q$ 且 $r_0<pq$ 时，$r_0+r_2p^2(\text{mod } pq)=r_0$。所以，只要在 r_0 和 r_2 选取的时候筛去这些值，就可以避免 r_0 的泄露。对于 r_0 的直接分解求解，可知从 $r_0+r_2p^2(\text{mod } pq)$ 分解得到 r_0 的概率最多是 $1/\lfloor\frac{pq}{2}\rfloor$。综上所述，CSSP 能得到 r_0 的概率最多为 $1/\lfloor\frac{pq}{2}\rfloor<\frac{1}{2^{2l}}$。

6 性能分析

假设安全参数为 l，消息块数为 n，在拥有预计算的密钥和验证值的情况下，考虑用户端和服务器端完整性检测所需的计算量、网络通信量、存储占用量。在计算量方面，用户端只需要计算挑战值和验证应答值，挑战 $k=k_1pq+r(\text{mod } p^2q^2)$，只需要一次模乘运算和一次模加法运算，验证时需计算 $y_1=f_0(m)+(k_1+r_1)f_1(m)(\text{mod } p^2)$，只需要一次模乘运算和两次模加运算；服务端只需计算应答 $y(m)=km_0+k^2m_1+\cdots+k^{n+1}m_n(\text{mod } N)$，计算量为 n 次模乘运算和 n 次模加

(下转第 113 页)

Semantics,2003,1(1):98-127

[14] Kalfoglou Y, Schodemmer M. Ontology Mapping: The State of the Art[J]. The Knowledge Engineering Review, 2003, 18(1): 1-31

[15] Borgida A, Serafini L. Distributed description logics: assimilating information from peer sources[J]. Journal of Data Semantics, 2003, 1(1): 153-184

[16] Hu Luo-kai, Ying Shi, Jia Xiang-yang. Towards an Approach of Semantic Access Control for Cloud Computing[C]// Proc. of 1st International Conference on Could Computing. Beijing, China: Springer, 2009: 145-156

[17] Hu Luo-kai, Ying Shi, Jia Xiang-yang. A Semantics Based Approach for Cross Domain Access Control[J]. Journal of Internet Technology, 2010, 11(1): 279-288

[18] Hu Luo-kai, Ying Shi, Chen Rui. A Semantic Web Service Description Language[C]// Proc. of 4th International Conference on Information Engineering. Taiyuan, China: IEEE Computer Society, 2009: 449-452

[19] 吕建,马晓星,陶先平,等. 网构软件的研究与进展[J]. 中国科学E辑:信息科学,2006,36(10):1037-1080

[20] 林闯,封富君,李俊山. 新型网络环境下的访问控制技术[J]. 软件学报,2007,18(4):955-966

(上接第 97 页)

运算。在存储占用量方面,用户端需要为每个文件存储一个预计算的校验值,服务器端只需存储数据文件即可。在网络通信量方面,挑战值和应答值都是固定常数值比特。

在配置为 Intel Pentium(R) Dual-Core CPU 2.7GHz,2G RAM 的机器上,使用 C++语言和 GMP 大数运算库编写了(l=128,n=51200)参数的算法模拟程序。在程序运行中,服务端计算应答值花费的时间为 443165us,用户端验证应答值的时间为 23us。

下面选取云存储完整性检测方案中比较典型的 3 种方案,即 PDP 方法原创者 Ateniese 等的方案[4]、支持完全动态更新的 Erway 等的方案[9]和 Wang Qian 等的方案[10],分别从计算量、网络通信数据量和存储占用量来对比分析算法的性能,具体参数如表 1 所列。从表 1 可以得出,我们的方案在计算量、网络通信量和存储量方面都是常数量级别,具有比较好的实用性。

表 1 部分性能参数对比(n 为数据划分的块数,t 为数据检测时的抽样块数)

方案	计算量		网络通信量	存储占用量
	用户	服务器		
我们算法	O(1)	n	O(1)	O(1)
文献[4]	O(n)	O(n)	O(1)	O(1)
文献[9]	O(tlogn)	O(tlogn)	O(logn)	O(1)
文献[10]	O(tlogn)	O(tlogn)	O(logn)	O(1)

结束语 本文针对云存储数据完整性验证问题,在哈希树结构的基础上,结合大数模运算,提出一种新的树形完整性检测结构——IC-树(Integrity Checking tree)。基于 IC-树和 Seny 等提出的云存储框架[13],设计了一种新的云存储数据完整性检测概率解决算法。分析结果表明,对于远端云中的文件,该算法在同步存储其相应的常量级的校验信息的前提下,用户能够以常量的网络通信量和计算量极大概率地正确检测数据的完整性,且支持文件的数据动态更新。下一步研究工作包括如何减少服务器端的计算量以及如何实施公开验证、提供隐私保护等。

参 考 文 献

[1] Kaufman L M. Data Security in the World of cloud computing [J]. Security & Privacy, 2009, 7: 61-64

[2] 冯登国,张敏,张妍,等. 云计算安全研究[J]. 软件学报,2011,22(1):71-83

[3] Mykletun E, Narasimha M, Tsudik G. Authentication and integrity in outsourced databases [J]. ACM Transactions on Storage, 2006, 2(2): 107-138

[4] Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores[C]// Proceedings of the 2007 ACM Conference on Computer and Communications Security. New York: ACM, 2007: 598-609

[5] Juels A, Kaliski B S. Pors: proofs of retrievability for large files [C]// Proceedings of the 2007 ACM Conference on Computer and Communications Security. New York: ACM, 2007: 584-597

[6] Shacham H, Waters B. Compact proofs of retrievability[C]// Proceedings of Asiacrypt 2008. Berlin: Springer-Verlag, 2008: 90-107

[7] Ateniese G, Pietro R D, Mancini L V, et al. Scalable and efficient provable data possession [C]// Proceedings of the 4th international conference on security and privacy in Communication networks. New York: ACM, 2008: 9: 1-10

[8] Wang C, Wang Q, Ren K, et al. Ensuring data storage security in cloud computing[C]// Proceedings of International Workshop on Quality of Service 2009. New York, USA: IEEE, 2009: 1-9

[9] Erway C C, Kupcu A, Papamanthou C, et al. Dynamic provable data possession [C]// Proceedings of the 16th ACM conference on Computer and communications security. New York: ACM, 2009: 213-222

[10] Wang Q, Wang C, Li J, et al. Enabling public verifiability and data dynamics for storage security in cloud computing[J]. Lecture Notes in Computer Science, 2009, 5789/2009: 355-370

[11] Bowers K D, Juels A, Oprea A. Proofs of retrievability: Theory and implementation[C]// Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW 2009. New York: ACM, 2009: 43-54

[12] Bragantini R, Conti M, Di Pietro, et al. Security in Outsourced Storage: Efficiently Checking Integrity and Service Level Agreement Compliance[C]// Proceedings of CIT. 2010. New York: IEEE, 2010: 1096-1101

[13] Kamara S, Lauter K. Cryptographic Cloud Storage[C]// Financial Cryptography and Data Security. Berlin: Springer, 2010: 136-149

[14] 罗堃,吴朝宏. 哈希树[EB/OL]. http://wenku.Baidu.com/view/16b2c7abd1f34693daef3e58.html, 2011-11-13