文章编号:1671-9352(2014)09-0160-06 DOI:10.6040/j.issn.1671-9352.2.2014.183

# 一种高效的多副本数据持有性证明方案

刘文卓,曹天杰*,黄石
(中国矿业大学计算机科学与技术学院,江苏 徐州 221116)

**摘要**:为了验证云存储中服务提供商是否完整地存储了用户的数据副本,在分析支持动态数据的持有性证明方案(RDPC)的安全缺陷(无法抵抗替换和重放攻击)的基础上,对其进行了改进和扩展,提出一种基于同态哈希技术的多副本持有性证明方案。该方案能够同时对多个副本的持有性进行验证,具有抵抗替换攻击、重放攻击和伪造攻击的能力。通过对Merkel哈希树进行改进,使方案更好地支持动态数据操作,同时利用γ编码技术,使得对数据块的验证和更新等操作过程中通信的数据量更小。

**关键词**:云存储;多副本持有性验证;同态hash;Merkel哈希树;动态数据
**中图分类号**:TP302 **文献标志码**:A

# An efficient multiple-replica data possession checking scheme

LIU Wen-zhuo, CAO Tian-jie*, HUANG Shi
(School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, Jiangsu, China)

**Abstract**: In order to verify whether the users' file replicas were stored by the cloud storage service provider (CSP), a RDPC that supports dynamic data was improved and expanded by analyzing its weakness, and then an efficient multiple-replica data possession scheme based on the homomorphic hash was proposed. Multiple-replica data possession can be verified simultaneously in the scheme, which can resist the replace attack, replay attack and forgery attack. To support data dynamic better, the Merkle hash tree was improved in the scheme, and the γ code was used to decrease the communication bandwith in the process of data blocks checking and update operations.

**Key words**: cloud storage; multiple-replica provable data possession; homomorphic hash; Merkel hash tree; data dynamic

## 0 引言

与传统的存储相比,云存储具有更好的容灾容错能力、易扩展性和弹性付费性、易管理与维护、易增加专业服务以及易迁移等优势。在云存储中,服务提供商(commerce service provider,CSP)是不可信的,为了获益,CSP可能将不常被用户访问的数据转移到线下存储设备上,甚至删除数据来节省存储开销,并对用户隐瞒数据的损坏或丢失。因此,在云存储中用户应当拥有云上元素的可信证据,并能够在公共云上控制他们的外包数据,动态地对数据进行更新。

可恢复性证明(proof of retrievability,POR)是Juels and B. Kaliski提出的[1],其主要思想是对加入了纠错码的数据在使用伪随机序列确定的一系列随机位置上插入一些随机的数据块,用于对数据进行完整性验证。为了防止验证者的欺诈和认证数据的泄露,Giuseppe Ateniese等人提出了数据持有性证明(provable data possession, PDP)[2],PDP采用公开可验证的模式和同态签名算法,使得数据可以被任何人验证且能够使用较少的数据块验证云上的

**收稿日期**:2014-06-24; **网络出版时间**:2014-08-28 16:59
**网络出版地址**:http://www.cnki.net/kcms/doi/10.6040/j.issn.1671.9352.2.2014.183.html
**作者简介**:刘文卓(1990-),女,硕士研究生,研究方向为云计算. E-mail: wzliucumt@gmail.com
*通讯作者:曹天杰(1967- ),男,教授,博士,研究方向为密码学与信息安全. E-mail:tjcao@cumt.edu.cn

海量数据。之后,Chris Erway 等人[3]提出了一个可以支持插入、修改、删除等动态数据的完整性验证方案。为了提高数据存储的容灾能力和可恢复能力,Curtmola 等人[4]实现了一个可证明的安全多副本的 PDP 方案,可以允许客户端验证服务器持有这多个副本。

文献[5]中,Chen 提出的 PDP 方案使用了同态哈希,且没有验证次数的限制,李超零等在文献[6]中对 Chen 的方案进行改进,设计了一种基于同态哈希的多副本持有性证明方案,但不支持动态数据操作。文献[7]中,Chen 对之前的方案进行改进,提出了一个支持动态数据的方案 RDPC,本文通过分析指出此方案无法抵抗替换和重放攻击,并针对这个问题,提出了一个支持动态数据的多副本数据持有性验证方案 EMPC。EMPC 能够同时对多个副本的持有性进行验证,具有抵抗替换攻击、重放攻击和伪造攻击的能力。同时,本文基于文献[8]提出的同态哈希,改进了 Merkel 哈希树[9],使其能更好地支持动态数据操作(如块的更新、插入和删除),同时利用 γ 编码技术,减少对数据块的验证和更新等操作过程中的带宽消耗。

## 1　Chen 方案的安全性问题

Chen 方案[7]的基本思想是通过同态哈希的方法验证远程服务器上数据的持有性。在数据持有性证明过程中可能存在:替换攻击、重放攻击和伪造攻击,而在文献[7]的远程数据持有性证明方案中,CSP 就可以进行替换和重放攻击。攻击过程说明如下。

**替换攻击**　假设用户存储在 CSP 上的某一文件可分成 $n$ 个块,且随机挑战的块为 $\{b_{r_1}, b_{r_2}, b_{r_3}, \cdots, b_{r_c}\}$,CSP 可以在其他 $n-1$ 个块中随机选择一个块 $b_j$ 来替代 $b_{r_t}$,并利用 $b_j$ 对应的标签 $t_j$ 计算 $B = \sum_{i=r_1, i\neq r_t}^{r_c} b_i + b_j$, $T = \prod_{i=r_1, i\neq r_t}^{r_c} t_i \times t_j$,再将 Proof = $\{B, T, \{t_i, \psi_i\}_{r_1\leqslant i\leqslant r_c, i\neq r_t} \cup \{t_j, \psi_j\}, \mathrm{sig}_{\mathrm{sk}}(h(R))\}$ 返回给客户端,导致用户仍能验证 $h(R') = h(R)$, $H_K(B) = T$,实施替换攻击。

**重放攻击**　CSP 可以保存之前成功通过持有性认证的 Proof,当用户再次发起挑战验证时,可以直接将其返回给用户,同样也能通过用户验证,造成重放攻击。

通过对以上两种攻击的分析可以发现由于客户端无法验证服务器端返回的 Proof 与自己所挑战的数据块之间的对应关系,从而导致 CSP 可能用其他数据块进行替换或用已经验证的结果进行重放。

## 2　高效的多副本持有性证明方案

用户把数据外包给 CSP 进行多副本存储,CSP 向用户提供各个副本所在的服务器的逻辑 ID。假设副本文件 $\boldsymbol{F} = (F_1, F_2, \cdots F_u \cdots, F_s)$,每一个副本对应一个服务器逻辑 ID。副本 $\boldsymbol{F}_u$ 可以分割成若干数据块(每个数据块大小为 $\beta = 16$ KB):$F_u = (b_1, b_2, b_3, \cdots, b_n)$,并生成相应的同态哈希标签序列 $\boldsymbol{T}_u = (T_1, T_2, \cdots, T_n)$,生成过程加入了副本 ID 和基于块序号的安全参数 Tsp(Tsp 的计算将在标签生成过程中给出详细说明),这样能具备抗替换攻击和重放攻击的能力。为了使此方案支持数据的动态操作,引入改进的 Merkel 哈希树(AMHT),将块序号与 Tsp 进行绑定。副本表示成数据块的形式如下:

$$\boldsymbol{F}_u = (b_1 b_2 \cdots b_j \cdots b_n) = \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{pmatrix},$$

其中 $F_u$ 是 $m \times n$ 的矩阵,$m$ 是每个数据块中包含的子块数,$m = \lceil \beta/(\lambda_q - 1) \rceil$ (512 bit);$n$ 是副本分块数。

### 2.1　改进的 Merkel 哈希树

图 1 是改进的 Merkle 哈希树结构(AMHT),数据块序号 $i$ 和 $\mathrm{Tsp}_i$ 构成一对叶节点,中间节点 $h_i = h(i \| \mathrm{Tsp}_i)$,例如 $h_1 = h(1 \| \mathrm{Tsp}_1)$,$h$ 为不带密钥的普通哈希函数(Md5、SHA-1 等);根节点 Root 的值等于各个儿子节点同态哈希值的乘积,即 $H_K(R) = H_K(h_1) \times H_K(h_2) \times \cdots \times H_K(h_n) = H_K(h_1 + h_2 + \cdots + h_n)$,其中 $H_K$ 为同态哈希函数,$K$ 为同态哈希密钥。用户用私钥 sk 对 $H_K(R)$ 签名后得到 $\mathrm{sig}_{\mathrm{sk}}(H_K(R))$,并将其发送给 CSP 保存。

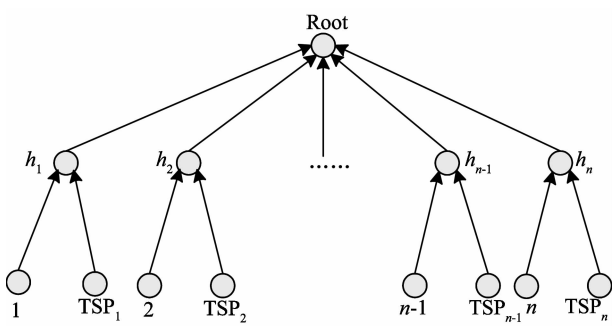


图 1　改进的 Merkle 哈希树

Fig. 1　Adjusted Merkel Hash tree

在验证某一个数据块序号 $i$ 和 $\mathrm{Tsp}_i$ 的绑定关系时，只需要将 $i$、$\mathrm{Tsp}_i$ 和其他中间节点哈希值的和返回给用户，用户根据返回的 $i$ 和 $\mathrm{Tsp}_i$ 计算 $R'=\sum_{j=1,j\neq i}^{n}(h_j+h_i)$，最后验证 $H_K(R')\stackrel{?}{=}H_K(R)$ 即可。验证多个绑定关系的过程也类似，另外使用 $\gamma$ 编码对返回的 Tsp 序列进行压缩来减少传输带宽（一个 20 GB 的文件只有大约 160 kB 的通信流量）。

## 2.2 主要算法

本方案共有以下 6 个算法构成：初始化（KeyGen）、副本生成（ReplicaGen）、标签生成（TagGen）、证据挑战（ProofChal）、证据生成（ProofGen）、证据验证（ProofVerify）。

### 2.2.1 持有性验证过程

（1）$\mathrm{keygen}(\lambda_p,\lambda_q,m,l,\mathrm{seed})\rightarrow(k,K,g_t,\mathrm{sk},\mathrm{pk})$

密钥初始化算法。此算法主要生成整个方案所需的密钥，输入参数 $\lambda_p$（1 024 bit）和 $\lambda_q$（257 bit）分别为随机大素数 $p$ 和 $q$ 的长度，$m$ 是每个数据块的子块数，$l$ 是对称密钥的长度，seed 是随机数种子，用来保证密钥的新鲜性；输出为对称密钥 $k$、同态哈希密钥 $K$、标签生成密钥 $g_t$ 以及 RSA 中的私钥 sk 和公钥 pk。

（2）$\mathrm{ReplicaGen}(k,F,s)\rightarrow(F_1,F_2,\cdots,F_s)$

副本生成算法。数据拥有者使用此算法生成文件 $F$ 的 $s$ 个不同副本（$s$ 是 CSP 服务器数，即文件副本数）。原理是将每个副本服务器的逻辑 ID 和密钥 $k$ 连接，然后进行哈希运算，将哈希值作为每个副本对应的加密密钥，用来对副本加密。用户在进行持有性验证时，使用密钥 $k$ 和服务器返回的逻辑 ID，重新生成加密密钥，解密文件副本。

（3）$\mathrm{TagGen}(F_i,K,g_t,k)\rightarrow(T_i,\mathrm{sig}_{\mathrm{sk}}(H_K(R)),\mathrm{Tsp},E_k(\mathrm{Tsp}_{\max}))$

标签生成算法。每个文件副本 $F_i$ 对应一个公共的加密存储在服务器端的 $E_k(\mathrm{Tsp}_{\max})$（$E_k$ 是对称加密算法，$k$ 是对称密钥）和一个公开的 Tsp 序列，$\mathrm{Tsp}=(\mathrm{Tsp}_1,\mathrm{Tsp}_2,\cdots\mathrm{Tsp}_j,\cdots\mathrm{Tsp}_n)$，$\mathrm{Tsp}_{\max}$ 即为序列中的最大值，$\mathrm{Tsp}_j$ 对应第 $j$ 个数据块，序列中的每个值都不重复。$\mathrm{Tsp}_j$ 生成过程：$\mathrm{Tsp}_j\leftarrow E_k(E_k(\mathrm{Tsp}_{\max}))$，$\mathrm{Tsp}_j\leftarrow\mathrm{Tsp}_{\max}+1$。

标签序列 $T_i=(T_{i1},T_{i2},\cdots,T_{ij},\cdots,T_{in})$，$T_{ij}$ 生成过程：

$r_{ij}=\varphi(h(\mathrm{filename}\,\|\,\mathrm{SeverId}_i\,\|\,k),\mathrm{Tsp}_j)$，（$\varphi(\cdot)$ 是伪随机数生成函数。）

$T_{ij}=g_t^{r_{ij}}\times H_K(b_{ij})\bmod p=g_t^{r_{ij}}\times\prod_{t'=1}^{m}g_{t'}^{b_{it'j}}\bmod p$。

其中，$g_t$ 表示标签生成密钥，从 $Z_p$ 中随机选取，阶为 $q$。

为了防止 CSP 对标签进行替换攻击和重放攻击，需要使用 AMHT 来绑定数据块序号 $j$ 和 $\mathrm{Tsp}_j$ 之间对应关系。最后，客户端将 $F_i$、$T_i$、Tsp、$E_k(\mathrm{Tsp}_{\max})$ 以及 $\mathrm{sig}_{\mathrm{sk}}(H_K(R))$ 发送给服务器保存，用户秘密保留 $g_t$、$K$、$k$、sk、pk。

（4）$\mathrm{ProofChal}(w)\rightarrow(e,c)$

证据挑战算法。输入一个随机数 $w$，生成用于计算挑战块位置的随机密钥 $e$ 和挑战块数 $c$，其中第 $j$ 个挑战块的位置 $u_j=\sigma_e(j)(1\leqslant j\leqslant c)$（$\sigma(\cdot)$ 产生每次随机挑战数据块的序号）。用户将 $(e,c)$ 和要挑战的副本文件名信息发送给 CSP，要求 CSP 提供所有副本中这些块被正确持有的证据。

（5）$\mathrm{ProofGen}(e,c)\rightarrow(B,T,h_c,\mathrm{Tsp}_c=\{\mathrm{Tsp}_{u_j}\mid u_j\in U\},\mathrm{sig}_{\mathrm{sk}}(H_K(R)))$

证据生成算法。CSP 收到副本文件的挑战信息后，计算 $c$ 个挑战块的位置集合 $U=\{u_j\mid u_j=\sigma_e(j)(i\leqslant j\leqslant c)\}$。然后计算挑战块持有性证据：$B=\sum_{i=1}^{s}\sum_{j=1}^{c}b_{iu_j}\bmod q$，$T=\prod_{i=1}^{s}\prod_{j=1}^{c}T_{iu_j}\bmod p$，哈希树中非挑战块位置$(j,\mathrm{Tsp}_j)$的哈希值的和 $h_c=\sum_{j=1,j\notin U}^{n}h_j\bmod q$，并将$(B,T,h_c,\mathrm{Tsp}_c=\{\mathrm{Tsp}_{uj}\mid u_j\in U\},\mathrm{sig}_{\mathrm{sk}}(H_K(R)))$返回给用户。

（6）$\mathrm{ProofVerify}(e,c,B,T,h_c,\mathrm{Tsp}_c=\{\mathrm{Tsp}_{uj}\mid u_j\in U\},\mathrm{sig}_{\mathrm{sk}}(H_K(R)))\rightarrow(\mathrm{True},\mathrm{Flase})$

证据验证算法。用户计算 $c$ 个挑战块的位置集合 $U$，然后利用 $h_c$、挑战块位置信息和 $\mathrm{Tsp}_c$ 重新生成哈希树的哈希值，验证 CSP 返回的标签安全参数 $\mathrm{Tsp}_c$ 是否与挑战的块相对应，若验证通过，继而计算每个副本中各个挑战块位置的随机数参数 $r_{iu_j}=\varphi(h(\mathrm{filename}\,\|\,\mathrm{SeverId}_i\,\|\,k),\mathrm{Tsp}_{u_j})$，验证 $T$ 是否等于 $T'=H_K(B)\times\prod_{i=1}^{s}\prod_{j=1}^{c}g_t^{r_{iu_j}}\bmod p$，若通过验证则认为 CSP 上的文件副本都保存完好。以下为算法的具体实现。

**证据验证算法** $\mathrm{ProofVerify}(e,c,B,T,h_c,\mathrm{Tsp}_c,\mathrm{sig}_{\mathrm{sk}}(H_K(R)))$

```
hc←0
  for j = 1 to c do
    h_c←h_c + h(u_j || Tsp_{u_j}) mod q
done
if (H_K(h_c) == H_K(R)) then
  T'←1
  for i = 1 to s do
    for j = 1 to c do
```

$r_{iu_j} = \varphi(h(\text{filename} \parallel \text{SeverId}_i \parallel k), \text{Tsp}_{u_j})$

$T' \leftarrow T' \times g_t^{r_{iu_j}} \bmod p$

done

done

done

$T' \leftarrow T' \times H_K(B) \bmod p$

If $(T' == T)$ then

return True

done

return False

2.2.2　动态数据操作

为了方便,将针对某个数据块做具体的动态数据操作说明。用户获得副本文件 $F_i$ 后首先对其解密,在数据操作完成后,用户对修改后的数据块重新用 $s$ 个不同密钥加密,然后将加密后的数据块和对应的数据操作请求一起提交给 CSP,对相应的文件副本进行修改。

(1)数据块的插入

此过程分两步:首先更新副本文件及验证标签序列,然后更新 AMHT。

假设插入的数据块为 $b'_I$,用户首先从 CSP 获得 $E_k(\text{Tsp}_{\max})$,计算出该数据块的 $\text{Tsp}'_I$ 和随机参数 $r'_{iI} = \varphi(h(\text{filename}||\text{SeverId}_i||k), \text{Tsp}'_I)$,$(1 \leqslant i \leqslant s)$,对该数据块重新用副本生成过程中的 $s$ 个密钥加密,得到$(b'_{1I}, b'_{2I}, \cdots, b'_{sI})$,计算出该数据块在所有副本中的验证标签 $T'_{iI} = g_t^{r'_{iI}} \times H_K(b'_{iI}) \bmod p = g_t^{r'_{iI}} \times \prod_{t'=1}^{m} g_{t'}^{b'_{i,t'I}} \bmod p (1 \leqslant i \leqslant s)$,客户端将加密后的数据块$(b'_{1I}, b'_{2I}, \cdots, b'_{sI})$和对应的验证标签$(T'_{1I}, T'_{2I}, \cdots, T'_{sI})$发送给 CSP,使其更新 $s$ 个副本文件及其数据块的验证标签集合。

新增的数据块会导致其后的数据块的绑定序号发生递增,即从原来的$(i, \text{Tsp}_i)$变为$(i+1, \text{Tsp}_i)$。因此 CSP 首先需要计算 $h_I = \sum_{i=1}^{I-1} h(i||Tsp_i)$,然后将$(h_I, \{\text{Tsp}_I, \text{Tsp}_{I+1}, \cdots, \text{Tsp}_n\}, \text{sig}_{\text{sk}}(H_K(R)))$返回给用户,验证并重新计算 $\text{sig}_{\text{sk}}(H_K(R'))$。用户计算 $R' = h_I + \sum_{i=I}^{n} h(i||\text{Tsp}_i)$,验证 $H_K(R)$ 是否等于 $H_K(R')$,若相等,则更新块序号的绑定关系,即计算 $R'' = h_I + h(I||\text{Tsp}'_I) + \sum_{i=I+1}^{n+1} h(i||\text{Tsp}_{i-1})$,生成新的根哈希值签名 $\text{sig}_{\text{sk}}(H_K(R))$,将其连同 $\text{Tsp}_I$ 一起返回给 CSP 更新保存。

(2)数据块的删除

与数据块的插入过程类似,首先通知 CSP 从 $s$ 个副本服务器中删除相应的数据块及对应的标签,再更新 AMHT。可参考插入的过程。

(3)数据块的修改

当需要修改某数据块 $b_M$ 时,只需从 CSP 处取回 $\text{sig}_{\text{sk}}(H_K(R))$、$\text{Tsp}_M$ 及 $h_M = \sum_{i=1, i\neq M}^{n} h(i||\text{Tsp}_i)$,计算 $R' = h_M + h(M||\text{Tsp}_M)$,验证 $H_K(R)$ 与 $H_K(R')$ 是否相等,若一致,则生成 $s$ 个随机参数 $r'_{iM} = \varphi(h(\text{filename}||\text{SeverId}_i||k), \text{Tsp}_M)$,$(1 \leqslant i \leqslant s)$,并对修改后的数据块重新用副本生成过程中的 $s$ 个密钥加密,得到$(b'_{1M}, b'_{2M}, \cdots, b'_{sM})$,然后计算该数据块在所有副本中的验证标签 $T'_{iM} = g_t^{r'_{iM}} \times H_K(b'_{iM}) \bmod p = g_t^{r'_{iM}} \times \prod_{t'=1}^{m} g_{t'}^{b'_{i,t'M}} \bmod p (1 \leqslant i \leqslant s)$,将加密后的数据块$(b'_{1M}, b'_{2M}, \cdots, b'_{sM})$及对应的验证标签$(T'_{1M}, T'_{2M}, \cdots, T'_{sM})$发送给 CSP,使其更新 $s$ 个副本文件及其验证标签集合。

## 3　安全性证明

安全性证明包括抗伪造攻击证明、抗替换攻击证明以及抗重放攻击证明。标签的生成过程中加入了密钥 $k$,使 CSP 无法伪造验证标签,因此能抵抗伪造攻击。抗替换攻击和重放攻击证明定义为可以在持有性证明和动态数据操作阶段有效地防止 CSP 用非本次挑战的其他数据块替换或用先前通过验证的结果重放来欺骗用户。

**定理 1**　CSP 无法破坏数据块序号和 Tsp 之间的对应关系,无法使用其他数据块及其对应的 Tsp 和验证标签来实现替换攻击或重放攻击。

**证明**　数据块序号和 Tsp 之间的对应关系通过 AMHT 来绑定,且用户使用私钥对根哈希值签名,当动态数据操作需要更新后续数据块的序号和 Tsp 之间对应关系时,用户利用服务器返回的 Tsp 重新计算根哈希值,与签名的根哈希值比较,结果一致才会更新后续数据块的绑定关系,生成新的根哈希值,用私钥签名后交给 CSP 保存。

进行挑战验证时,用户收到服务器返回的 Tsp 后,首先验证其与挑战数据块序号之间的对应关系,然后验证挑战数据块与标签之间的一致性。若 CSP 使用其他数据块及其对应的标签来替换所挑战的数据块和标签,同时替换对应的 Tsp,则无法通过第一步验证;若未替换对应的 Tsp,则无法通过第二步验证,因此本方案可以抵抗替换攻击。同时,由于用户可验证挑战的数据块序号与返回的持有性证据是否对应,也能抵抗重放攻击,且对于一个数据块总数只有 100 的文件,挑战其 50% 的数据块,所有可能的

组合数达到 $C_{100}^{50}\approx10^{29}$，想要进行穷举重放，其代价比直接存储数据块和验证标签高得多。

## 4 性能分析

在实现 EMPC 时，随机素数 $p$、$q$ 的长度分别是 1 024 bit、257 bit，数据块的大小 $\beta=16$ kB，其对应的 Tsp 为 4 B（最大可以表示到 $2^{32}$），$k$ 的长度为 256 bit，RSA 密钥长度为 2 048 bit。

（1）存储开销

对用户来说，只需要存储密钥信息：$g_t$、$K$、$k$、sk、pk，总的开销约为（512 +（1 024 + 257 + 512）+ 256 + 6 144 + 2 080）/8 = 1.3 KB，占用的空间非常小；而 CSP 需要在每个服务器上保存一个文件副本（16$n$KB）、该副本对应的验证标签集合（128$n$ B，以及一个 Tsp 集合（4$n$ B），一个加密的 $E_k(\mathrm{Tsp}_{\max})$（4 B），一个 $\mathrm{sig}_{\mathrm{sk}}(H_K(R))$（256 B），总的文件膨胀率为 $\frac{128n\times s+4n+300}{16n\times1\,024\times s}\leqslant2.64\%$（当 $n=1$，$s=1$ 时取最大值）。实际情况比这个值要小的多，当文件数据块总数 $n$ 为 1 000，副本数 $s=2$，膨胀率只有 0.79%。

（2）通信开销

在进行完整性验证时，CSP 需要向用户返回：$h_c$（1 024 bit），$\mathrm{Tsp}_c$（4$c$ $B$），$\mathrm{sig}_{\mathrm{sk}}(H_K(R))$（2 048 bit），$B$（257 bit），$T$（1 024 bit），其总的通信流量为（544 +4$c$）$B$。在插入和删除操作时通信开销最大，因为 CSP 需要向用户发送所有与后续数据块序号对应的 Tsp。这里用插入操作举例说明，CSP 向用户发送 $h_l$（1 024 bit），（$\mathrm{Tsp}_l$，$\mathrm{Tsp}_{l+1}$，…，$\mathrm{Tsp}_n$）（4($n-i+1$)B），$\mathrm{sig}_{\mathrm{sk}}(H_K(R))$（2 048 bit），总流量为（384 + 4($n-i+1$)）B，最坏情况下插入位置为 1，此时服务器要向用户返回所有的 Tsp，总流量为（384 +4$n$）B。本方案中使用 $\gamma$ 编码[17]对 Tsp 序列进行压缩，其原理是对 Tsp 的间距进行编码，并在位的粒度上进行编码长度的自适应调整。与文献[7]提出的 RDPC 相比，EMPC 方案产生的通信开销几乎可以忽略不计，而 RDPC 方案需要传回每个挑战块在哈希树中各节点的哈希值。此外，由于 CSP 会将挑战文件的所有副本计算后的一个统一结果返回给用户，即 $h_c$、$\mathrm{Tsp}_c$、$\mathrm{sig}_{\mathrm{sk}}(H_K(R))$、$B$ 和 $T$，因此 EMPC 在挑战验证过程中的通信开销并不会随着副本数的增加而变大。

（3）时间开销

在验证持有性过程中，ProofGen 算法需要进行 $s\times c$ 次模乘运算 $T_{\mathrm{mul}}$，$s\times c+(n-c)$ 次模加运算 $T_{\mathrm{add}}$ 和 $(n-c)$ 次普通哈希运算 $T_h$，总开销为 $(s\times c)T_{\mathrm{mul}}+(s\times c+(n-c))T_{\mathrm{add}}+(n-c)T_h$；ProofVerify 算法需要进行 $c+s\times c$ 次普通哈希运算 $T_h$，$(c-1)$ 次模加运算 $T_{\mathrm{add}}$，1 次同态哈希运算 $T_H$，1 次签名验证运算 $T_{vs}$，$s\times c$ 次随机置换 $T_\varphi$，$s\times c+1$ 次模乘运算 $T_{\mathrm{mul}}$ 和 $s\times c$ 次模幂运算 $T_{\exp}$，总开销为 $(c+s\times c)T_h+(c-1)T_{\mathrm{add}}+T_H+T_{\mathrm{vs}}+(s\times c)T_\varphi+(s\times c+1)T_{\mathrm{mul}}+(s\times c)T_{\exp}$。图 2 是在 2 台 Intel Core i5 3.2 GHz CPU，2 GB RAM 的机器上，使用 C ++ 语言，基于 GNU MP 5.0.5 库实现 EMPC 方案的实验效果。

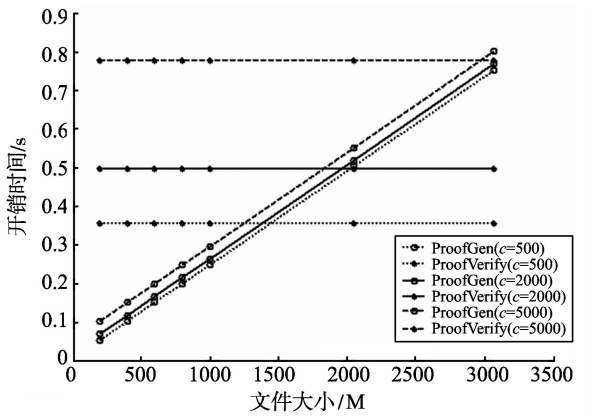


图 2 ProofGen、ProofVerify 时间开销的影响因素（$s=2$）

Fig. 2 Influence factors of time overhead for ProofGen, ProofVerify（$s=2$）

图 2 中，在 $c$ 给定时，随着文件变大，服务器端 ProofGen 的时间呈线性增长的趋势，而对 ProofVerify 几乎没有影响。在文件大小给定时，挑战块数越多，ProofGen、ProofVerify 的耗时均增加，且 ProofVerify 相比 ProofGen 增幅更大，这是由于在 ProofVerify 阶段存在与 $c$ 相关的模幂运算。而且在实际应用中挑战块数可以是稳定的，因为用户只需要挑战 460 块，就能以 99% 的概率检测出 1% 改动的数据块[10]。

图 3 为在单副本挑战时，EMPC 与 RDPC 在挑战中的时间开销比较（1 G 的文件，数据块大小为 16 kB）。随着挑战块数的增加，两种方案的时间开销都有不同程度的增加，RDPC[7]方案的验证过程时间开销增加快，这是因为改进的哈希树深度只有 3，当挑战块数增加时，EMPC 中用户在验证签名的根哈希值时需要的计算次数少于 RDPC。在服务器生成持有性证据时，EMPC 要比 RDPC 稍慢，因为 EMPC 需要额外计算 $h_c$。另外，EMPC 随着挑战副本数的增加，用户 ProofVerify 过程要比服务器 ProofGen 过程的时间开销增加更为明显，原因在于随着副本数的增加，用户需要利用标签安全参数进行更多次的大数模幂运算和哈希运算，而服务器端只增加了模乘和模加运算，相对来说负担较小。

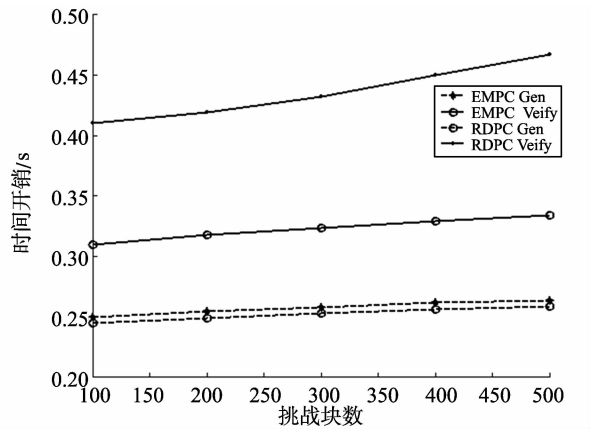


图 3　EMPC 和 RDPC[7] 时间开销比较(单副本挑战)

Fig. 3　The compare between EMPC and RDPCfor time overhead

## 5　结束语

本文提出了一种利用同态哈希技术的多副本持有性证明方案(EMPC),能够同时对多个副本的持有性进行验证。通过改进 Merkel 哈希树,使其支持动态数据操作,且利用 γ 编码技术减少数据块的验证和更新等操作中带宽消耗。安全性分析证明了其具有抵抗替换、重放和伪造攻击的能力。并与文献[7]中的 RDPC 进行比较,结果表明本文的方案无论是在安全性、通信开销还是时间开销方面都是更优的。

**参考文献:**

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

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

[3] ERWAY C, KÜPCÜA, PAPAMANTHOU C, et al. Dynamic provable data possession[C]//Proceedings of the 16th ACM Conference on Computer and Communications Security. New York: ACM Press, 2009: 213-222.

[4] CURTMOLA R, KHAN O, BURNS R, et al. MR-PDP: multiple-replica provable data possession[C]//Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS'08). Los Alamitos: IEEE Computer Society, 2008: 411-420.

[5] CHEN L X. A homomorphic hashing based provable data possession[J]. Journal of Electronics and Information Technology, 2011, 33(9): 2199-2204.

[6] 李超零,陈越,谭鹏许,等. 基于同态 Hash 的数据多副本持有性证明方案[J]. 计算机应用研究,2013,30(1):265-269.
LI Chaoling, CHEN Yue, TAN Pengxu, et al. Multiple-replica provable data possession based on homomorphic hash[J]. Application Research of Computers, 2013, 30(1):265-269.

[7] CHEN Lanxiang, ZHOU Shuming, HUANG Xinyi, et al. Data dynamics for remote data possession checking in cloud storage[J]. Computers & Electrical Engineering, 2013, 39(7): 2413-2424.

[8] KROHN M N, FREEDMAN M J, MAZIERES D. On-the-fly verification of rateless erasure codes for efficient content distribution[C]//IEEE Symposium on Security and Privacy. Los Alamitos: IEEE Computer Society, 2004:226-240.

[9] WANG Qian, WANG Cong, REN Kui, et al. Enabling public auditability and data dynamics for storage security in cloud computing[J]. IEEE Transactions on Parallel DistribSyst, 2011, 22(5):847-859.

[10] ATENIESE G, DI PIETRO R, MANCINI L V, et al. Scalable and efficient provable data possession[C]//Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks. New York: ACM Press, 2008: 1-11.

(编辑:许力琴)

(上接第 153 页)

[3] SOUCY P, MINEAU G W. A simple KNN algorithm for text categorization[C]//Proceedings of IEEE International Conference on Data Mining (CDM 2001). Washington: IEEE Computer Society, 2001: 647-648.

[4] 杨莉莉. 基于数据挖掘的数字取证模型设计[J]. 南京师范大学学报,2006, 29(6):18-21.
YANG Lili. Design of digital forensics model based on data mining[J]. Journal of Nanjing Normal University, 2006, 29(6):18-21.

[5] 鲁婷,王浩,姚宏亮. 一种基于中心文档的 KNN 中文文本分类算法[J]. 计算机工程与应用,2011, 47(2):127-130.
LU Ting, WANG Hao, YAO Hongliang. A KNN Chinese text classification algorithm based on center document [J]. Computer Engineering and Applications, 2011, 47(2):127-130.

[6] 田久乐,赵蔚. 基于同义词词林的词语相似度计算方法[J]. 吉林大学学报:信息科学版,2010, 28(6):602-608.
TIAN Jiule, ZHAO Wei. Words similarity algorithm based on tongyici cilin in semantic web adaptive learning system[J]. Journal of Jilin University: Information Science Edition, 2010, 28(6):602-608.

(编辑:许力琴)