云存储环境下文件完整性验证方案的设计与实现

付艳艳，张敏，冯登国
（中国科学院 软件研究所，北京 100080）

摘 要：在云存储服务中，用户需要准确掌握文件在非可信远程服务器端的存储状况，从而判断是否需要对数据进行恢复或是否可以使用数据进行其他操作。通过提前从文件中随机抽取样本文档进行签名，为用户提供了可信的验证凭据。当用户发起验证时，按照对应的规则，重新生成新的签名。比较签名是否一致，即可以判断文件是否完整。分析表明，这种随机抽样的方法可以以极高的概率发现文件损坏的发生。即使单次抽样的置信度并不高，用户也可以通过多次发起验证来获得更好的验证可信度。并且，这种验证方式的时间复杂度与文件大小无关，仅与验证的可信程度有关。实验结果表明，本方案比其他基于全文签名的验证方式可以更好地适用于云存储模型，并具有极高的可信程度。

关键词：云存储；完整性；随机抽样；存储验证
中图分类号：TP309 **文献标识码**：A **文章编号**：1000-436X(2011)9A-0008-08

Design and implement of integrity checking schema under cloud storage model

FU Yan-yan, ZHANG Min, FENG Deng-guo
(Institue of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: In the cloud storage model, user need to confirm the file status while it's kept in the untrusted remote storage server. Then user could decide to restore the data or use the data for other purpose. By pre-random sampling from a file to form samples and sign them, provide users with a credible certification credentials. When user initiates a verification, the storage server re-generate a new signature in accordance with the same rules. By comparing the signatures, user can verify whether the file is complete. Analysis showed that, with this random sampling method, user can find file corruption with quite high probability. though a single verification may be not quite provable, user can send multiple challenge to get better credibility. Also, time required for verification has nothing to do with file size, but the credibility of the verification. Experimental results show that this schema work better under cloud storage model than other schema based on signatures and has a very high credibility.

Key words: cloud storage; integrity; random sampling; storage verification

1 引言

为了解决用户海量文件存储需求，云存储方案被广泛应用。在云存储服务中，服务提供商根据用户的存储量、存储时间、访问流量等向用户收取一定的费用，并保证用户能够以服务水平协定（SLA）中规定的性能标准对数据进行访问。

但是，云存储平台对用户来说，并不是完全可

收稿日期：2011-02-28
基金项目：中国科学院知识创新工程领域前沿项目（YYYJ-1013）
Foundation Item: The Knowledge Innovation Program of the Chinese Academy of Sciences (YYYJ-1013)

靠的。近期不断爆出的云存储故障表明，即使是大公司也不能保证数据的绝对安全。2009 年微软子公司 Danger 服务器发生了一次巨大故障，导致 Sidekick 手机用户的通讯录、日程表、照片等个人数据丢失。Google 公司则分别在 2008 年和 2010 年发生了 Gmail 和日历服务中断的问题。除去故障原因，黑客攻击、服务器管理员的误操作都可能造成用户数据的遗失或被篡改。用户不能确定云存储服务提供商足够诚实，会向用户报告每一次故障和用户数据受损程度。即使假设服务提供商足够诚实，隐蔽的黑客攻击和管理员无意的误操作带来的数据破坏也可能需要经过相当长的延迟才会被发现。

如果用户不能及时了解所存储数据的安全状态，及时感知其所发生的变化，并予以相应处理，那么就可能导致在需要访问数据时无法找到正确结果，甚至使用出错的数据进行计算，面临无法挽回的灾难。对用户来说，越早发现数据的破坏，就能够越快地进行数据恢复，也在更大程度上降低了因此带来的损失。因此，用户需要通过一种可信赖的手段及时对存储在云服务器上的数据进行完整性检测。

目前常用的文件完整性验证方法可分为 2 类：基于用户和服务器间挑战应答机制的验证和基于第三方审计的验证。基于挑战应答机制的验证通常具有以下缺点：①用户需要额外保存大量的密钥和签名。②服务器端可能利用已生成的签名进行重放攻击。③每次文件签名都需要读取文件。当用户数据量巨大时，会大量消耗服务器端的内存和计算能力。而基于第三方审计的模型则过分依赖第三方的可信程度与安全性，甚至需要将文件内容曝露给第三方。这样即使解决了完整性验证问题，也引入了新的数据泄露隐患，并不具有实用性。

为了解决以上问题，本文提出了一种适用于云存储平台的文件完整性验证方法，通过“挑战—应答”机制抽查文件中的内容进行 hash，并将 hash 结果与预先准备的对应验证标签相比较，从而判断文件是否被破坏。笔者在 hadoop 平台上对方案进行了实现和测试。实验结果表明，本方案能够以较高的概率发现文件损坏，并且在云存储环境下可以获得较好的性能。

2　相关工作

视存储服务的可靠性问题在云服务出现之前就已经存在[1]。常见的解决方案可分为 2 类：基于第三方审计的验证[2]和基于“挑战—应答”协议的验证。由于第三方审计的方案具有以下弱点，本文主要讨论基于“挑战—应答”协议的解决方案。首先，试图审计云存储服务的所有文件操作是不现实的。由于云存储具有海量数据文件和用户的特点，系统的并发连接和操作数很容易超过第三方的审计能力。其次，对用户来说，将所有文件操作都暴露在第三方审计之下，是不可接受的。对用户行为的审计分析，很可能导致用户操作习惯，关注问题等隐私的泄露。再次，审计方案的安全性和可信程度极大地依赖于第三方的安全防护措施和可信程度。用户对于服务提供商和审计第三方的合谋攻击无法进行有效防御。因此认为，基于“挑战—应答”协议的解决方案在解决云存储环境下的文件完整性验证问题具有更大优势。

Deswarte[3]和 Filho[4] 等人提出了一个基于 RSA 算法来检验文件完整性的方案。他们认为，服务器是易受攻击的，因此，在服务器上运行的简单完整性验证结果是不能够采信的。而且，将文件从服务器下载下来再进行验证也是不切实际的。为了解决这个问题，首先引入了“挑战—应答”协议的方式来进行文件的完整性验证。在他们的方案中，每次验证，服务器端都需要重新读取全部文件内容，计算一个签名值。由于 RSA 算法的效率限制，在 3.0GHZ 的处理器上，1 024bit 密钥的 RSA 算法每加密 1MB 数据需要 20s，在需要验证海量文件的云存储场景中，用户很难在可容忍的时间得到所有文件的签名，因此，该方案并不可行。

2007 年，Giuseppe 等人提出了 PDP（provable data possession）模型[5]。该模型对存储在服务器上的文件进行随机采样验证，以一定的概率来判断服务器端整个文件是否完好。该方案需要提前为每一个块生成一个秘密 index 保存在用户本地。用户根据自己持有的秘密 index 和块的内容为每一个块生成 tag，将 tag 和元数据都保存到服务器。客户端随机挑选文件的若干块进行验证。该方案并不需要读取整个文档的内容来生成签名，因此，文件读取过程的 I/O 消耗得到了削减。但是，服务器端的额外存储和验证过程的通信复杂度没有得到控制。在实现方案中，对大小为 4GB 的文件，以 4KB 进行分块，签名密钥为 1 024bit，那么云

存储端的额外存储达到128MB。每次客户端发起challenge需要传递168byte，服务器端传回的proof大小为148byte。

以PDP模型为核心，研究人员提出了多种不同的实现方案[6,7]。这些方案大多都是基于同态函数的验证。基于同态函数的验证方案需要提前为每一个数据块m生成$g^m \pmod N$作为tag，并在验证的时候向服务器提供一个随机的g^s，服务器将g^s作为底数，计算出$g^{sm} \pmod N$。客户端利用幂运算的性质和持有的tag即可验证数据是否完整。Francesc等人[8]设计实现了基于欧拉定理的验证方案。对每一个文件分块m，提前计算出$m \pmod{\Phi(n)}$作为tag秘密保存。通过比较$a^{m \pmod{\Phi(n)}} \bmod(n)$与$a^{m'} \bmod(n)$是否相同判断m与m′是否相同。另外一些方案中，可以基于伪随机函数生成tag。客户端保存一个秘密密钥k，为每一块生成tag，$t_i = f_{k(i)} + \alpha(m_i)$发起验证时，客户端提供抽样的块序号i和对应的参数$V_i$。假设验证总共抽取k块，服务器端返回$\sum_{j=1}^{k} V_{i_j} t_{i_j}$和$\sum_{j=1}^{k} V_{i_j} m_{i_j}$。客户端比较返回的值是否满足关于$\alpha$的验证方程即可。以上2个方案的缺点也很明显。基于同态函数的验证大多需要进行大量的计算，资源消耗较大。而基于伪随机函数同态验证的安全性难以进行严格的证明。

POR(proofs of retrievability)模型随后很快被提出[9~11]。该模型结合检测点和纠错码来验证数据的完整性，并可以在数据出错时利用纠错码进行数据恢复。在文献[12]中讨论了多服务器环境下数据的完整性验证问题，提出一种完整性保护纠错码(IP-ECC, integrity-protected error correcting code)。在此方案中，用户将文件分布到各个不同的云存储服务器上，并分别生成基于单个服务器上所有文件的纠错码和基于单个文件的纠错码。用户可以基于这2种不同的纠错码对单个文件发起验证或者对单个服务器上的所有文件发起验证。

基于哨兵的验证是POR模型的另一类实现[13,14]在这种实现方案中，用户通过计算提前对数据文件进行分块、位置变换等处理，并选定数据文件的若干固定位置，在这些位置插入若干假数据。发起challenge时要求服务器端返回这些假数据的值。通过判断这些数据是否被更改可以判断文件是否完整。因此，也将这些假数据称为哨兵。哨兵数据通常是随机选定若干文件块的mac值，可以用来进行文件恢复。但是，这种方案只适用于加密文件，哨兵的数量也终会用尽。更重要的是，用户每次下载文件也需要将哨兵数据下载，给用户的正常读写操作带来不便，也额外消耗了带宽。

由于代数签名比普通数字签名更加快速方便，Thomas等人将这一技术应用到文件完整性验证方案中[17]。借助于代数签名的以下2个性质，可以对文件的签名进行验证。signature of parity=parity of signatures，sig(X Y)=sig(X) sig(Y)。用户首先利用持有的文件内容计算文件对应的parity数据块。为了减少遭受服务器端暴力攻击的威胁，可以将parity数据块与伪随机串进行异或。当用户发起验证时，只需要随机指定若干的数据块和签名参数。客户端利用代数签名的第一个特点，可验证数据是否正确。本方案的计算效率要远远高于其他散列算法。但是，代数用户每次计算一个签名，都需要访问所有的数据块。虽然需要读取的文件内容总量很少，但是这些内容并不连续，磁盘访问的时间消耗并没有得到优化。签名方案的安全性也并未得到严格证明。

除此之外，基于merkle hash tree验证方案也被应用于文件的完整性验证[16,17]。MHT方案通常用来检索文件树的更新，对于数据改变非常敏感。但是，当其应用于文件完整性验证时，需要花费相对较多的计算进行验证。对大小为4GB的文件，以4KB进行分块，验证一个数据块的完整性需要经过$O(\log(2^{20}))$次散列运算。当需要验证的文件块数增加时，用户的等待时间将不可忽视。

在比较这些方案时，用户主要考虑以下3个方面：①验证效率；②冗余存储的规模；③带宽消耗。考虑到现有方案的特点，提出了一个基于随机抽样选择文件块进行验证的完整性检测方案。该方案能够以较快的速度对大型文件进行完整性验证，并可以满足用户提出的验证可信程度要求。冗余存储的规模仅与用户指定的验证次数有关。

当文件只需要短期存储在非可信存储上时，用户可以依据存储时间自行定制需要验证的次数。需要验证的次数越少，所需要的准备时间越短，本方案的优势越明显。

3　基于随机抽样的完整性验证方案

3.1　定义

在文件完整性验证中，有 2 个角色：用户和云存储服务器。用户将文件上传到云存储服务器时，提前利用文件生成完整性验证的标签，并将标签保存到本地。用户要求进行完整性检测时，传递给云存储服务器某些特定的参数，云存储服务器依照参数重新生成对应的完整性验证标签，并返回给用户。用户通过比较对应的标签判断文件是否完整。

整个完整性验证过程可分为以下几个算法：

setup：该算法以当前系统时间 T，预期文件需要验证的次数 C，单次验证的可信程度 p，SuperBlock 的大小 B 作为输入，输出每次验证需要访问的文件块数 num，以及 C 个随机位置密钥的集合 K 和 SuperBlock 组合密钥 S。

prepare：该算法以文件 F，随机位置密钥集合 K，SuperBlock 组合密钥 S 和每次验证需要访问的文件块数 num 作为输入，为文件生成 C 个完整性验证标签 r。将标签和对应的参数加密保存到云存储端，以备用户进行验证。

challenge：该算法向服务器端发起挑战，挑战中包含：本轮挑战的随机位置密钥 k，本轮挑战的 SuperBlock 组合密钥 s。

prove：该算法以文件 F，本轮挑战的随机位置密钥 k，本轮挑战的 SuperBlock 组合密钥 s 作为输入，输出结果为新的标签 r'。

verify：该算法以本轮挑战对应的提前准备的标签 r 和 prove 算法新生成的 r'作为输入，检查两者是否一致，将检查结果作为输出。

3.2　基本思想

本方案的基本思想是，由用户指定单次验证的可信程度 p（如果数据中有 1% 发生了损坏，能够通过单次验证发现损坏的概率称为单次验证的可信程度 p），根据 p 计算出每次验证需要抽取的文件块数 num，然后通过随机位置密钥计算出 num 个随机位置，并通过 superblock 分配密钥将这 num 个文件块分为若干个 superblock。以 superblock 为单位读取这些文件块的内容作为样本文档。以随机位置密钥为密钥将样本文档进行散列运算。将所有的散列结果进行异或，得到 r，将 r 作为一次完整性验证的标签。其中，随机位置密钥和 superblock 分配密钥可通过当前系统时间生成。为了保证验证时能够重新生成对应的标签，需要将密钥种子，即当前系统时间 T 保存。

如果文件需要进行 C 次验证，则只需要根据当前系统时间 T 为文件生成 C 个随机位置密钥和 superblock 分配密钥。然后根据每一个密钥计算一个完整性验证标签即可。

3.3　算法描述

算法 setup 对完整性标签的生成过程进行初始化，获得用户输入的相关参数，并得到完整性标签计算所需的随机位置密钥集合 K，SuperBlock 组合密钥 S，文件访问所需的文件块数 num。

算法 prepare 是一个循环过程，直到生成全部 C 个验证标签之后结束。每一轮循环中，算法依次执行以下步骤。

1）使用一个新的随机位置密钥 k，为本轮计算随机挑选 num 个文件块。

2）使用一个新的 SuperBlock 分配密钥 s，为本轮计算的 num 个文件块进行分组，每组的文件块数限制在 B 范围内。同一组的文件块组成一个虚拟的 SuperBlock。

3）按照 SuperBlock 中文件块的顺序依次读取文件内容，形成一个 SuperBlock 样本文档。

4）以本轮的随机位置密钥 k 为密钥对样本文档进行散列运算，得到 H[SuperBlock]。

5）重复步骤 3)~步骤 4)，直到获得所有 SuperBlock 的散列值。

6）将所得到的散列值进行异或操作，得到完整性验证标签 r。

7）重复步骤 1)~步骤 6)，直到所有的随机位置密钥用尽。

算法 challenge 负责向服务器发起最新挑战。利用客户端保存的初始密钥 T，以及当前挑战次数，challenge 算法可以得到最新的 k 和 s，并进一步组成挑战，发送到服务器。

算法 prove 根据随机位置密钥 k，superBlock 分配密钥 s，需要访问的文件块数 num，SuperBlock 大小限制 B，重新生成新的完整性标签。

算法 verify 通过比较新的完整性标签和解密得到的完整性标签是否一致，判定文件的完整性是否被破坏。

图 1 是算法的伪代码描述。

```
setup(T,p,C)
p--> num
T--> K(k1;k2;k3;…;kC)
T--> S(s1;s2;s3;…;sC)

prepare(num,K,D,S,B)
for each ki in K si in S
int[num] locations=locationgene(ki,num);
int[] SuperB=SuperBgene(si,B);
countr1; buffer;

for each j in superB
do
buffer+=read(D locations[count])
count++;
while count< j
ri=Hash(bufferki)    ri
output(r1,r2,…,rC)

challenge(T index)
k,s<--  decrypt(T,index)

prove(D ki,num,B)
int[num]locations=generate(ki,num)
int[] SuperB=SuperBgenerate(si,B);
countr buffer;
for each j in superB
do
buffer+=read(D locations[count])
count++;
while count< j
r=Hash(bufferki)     r
output(r)

verify(r,r')
if(equals(r,r'))
output true;
else
output false
```

图1 算法的伪代码描述

4 性能分析

4.1 抗碰撞性分析

在以下情况下，认为攻击者 A 成功地欺骗了文件完整性验证：A 仅持有文件 D 的部分内容，但是 A 能够根据持有的文件内容生成用户 challenge 对应的标签。如果 A 成功生成标签的概率比抛硬币的概率要大，那么，就认为 A 能够欺骗文件完整性验证。例如，在标签大小为 50bit 的情况下，只有当成功欺骗的概率低于 $1/2^{50}$ 时，才可以认为方案是安全的。这个概率是 8.881 784 197 001 252×10^{-16}，也就是说，标签大小越大，相应的安全标准就越高。这对大多数完整性验证方案来说是一个巨大挑战。

在本方案中，每次验证均需要重新生成样本文档进行散列计算。如果服务器端在不持有正确文件的情况下仍能够生成正确的文档散列值，那么，即可认为服务器实施了一次成功的欺骗。考虑到散列函数的强抗碰撞性，只有当 2 次请求的样本文档相同，并且 SuperBlock 分组也完全相同时，服务器才可能利用以前发起请求时计算出的散列值进行重放，从而实现欺骗。

$$r_{success}=r_{collision}=p_{n\_collison}\quad p_{SuperBlock\_collison}$$

忽略 SuperBlock 分组相同的概率，可得：

$$r_{success}<1-\sum_{i=0}^{c-1}N^{n-i}/N^{nc}$$

经过化简，可得：

$$r_{success}<<1-(1-1/N^{n})=1/N^{n}$$

由此可知，样本文档发生碰撞的概率极低，可以忽略。只有当 $1/N^{n}<1/2^{m}$（m 为标签的 bit 数）时，本方案就是安全的。显然，当文件越大，随机选取的文件块数越多，本方案也就越安全。

4.2 验证可信程度分析

在本方案中，每次验证，随机抽取一定数目的数据块形成样本文档，对样本文档的完整性进行检测。通过调整抽取的数据块的数目，本方案能够以极大的概率发现文件中的完整性问题。假设文档中有 1% 的内容发生了损坏，用户希望通过单次验证，即能够以 90% 的概率发现文档发生了损坏。即 s=1%，p=90%。如果不能发现文档损坏，即抽取的 num 个文件块均无损坏，此概率为

$$1-p=C_{N-Ns}^{num}/C_{N}^{num}$$

即为：

$$1-p=(N-Ns)!(N-num)!/N!(N-Ns-num)!$$

当 num << N 时，上述式子可化简为：

$$1-p=(1-s)^{num}$$

即，$num=\log_{1-s}^{1-p}$。由此确定为了满足验证可信度需要抽取的文件块数。

另外，即使通过单次验证发现文件损坏的概率较低，用户可以通过多次验证来获得较高的验证可信度。比如，如果用户单次验证发现文档损坏的概率为 60%，用户连续发起 4 次验证，发现文件损坏的概率即提升为 $p=1-(1-60\%)^{4}=97.4\%$。

4.3 存储、通信复杂度分析

用户需要额外存储用户提前生成的完整性验证标签，以及密钥种子 T。另外，需要计数当前验证次数 i。假设完整性验证标签的长度为 |r| bit。密钥 T 的长度为 |T| bit，i 的长度为 |i|，完整性验证标签总数为 C，因此额外的存储空间消耗为 O(|T|+|i|+C|r|)。

每一次用户发起 challenge，首先需要在本地计算当前验证所需的位置和 superblock 组合密钥。根据 T 和当前验证次数 i 即可求得。用户将得到的 2 个密钥组成 challenge，发送到云存储服务器。发送 challenge 的带宽消耗为 O(|k|)。

prove 阶段，服务器端需要将重新生成的标签返回给用户，这一阶段的带宽消耗为 O(|r|)。

考虑到云存储服务的收费模型，用户下载流量需要进行严格控制。在本方案执行中，下载流量为 O(|r|+|k|)，大小仅为若干字节。

4.4　效率分析

在完整性验证的准备阶段，即 setup 和 prepare 阶段，用户需要提前生成所有的文件完整性验证标签。在这一过程中，用户的等待时间包括：读取每一个文件块消耗的时间 T_{read}，散列计算消耗的时间 T_{hash}。因此，准备阶段的总时间消耗可表示为：$T_{pre}=C\cdot num\cdot T_{read}+C\cdot T_{hash}$。从此公式可以看出，每次读取文件块的时间消耗在标签生成过程中占据了相当大的比例。因此，一次性的将文件读入内存，可以节省相当多的时间。此公式可转变为 $T_{pre}=N\cdot T_{read}+C\cdot T_{hash}$。

在完整性验证发起 challenge 的过程，需要经历一轮密钥生成过程以及发送 challenge 的过程。整个过程中的时间消耗可表示为 $T_{challenge}=T_{communicate}+T_{key\text{-}generate}$。

在完整性验证 prove 阶段，服务器端需要根据用户发起的 challenge 生成对应的完整性验证标签。生成完整性验证标签的过程与准备过程相同，因此，时间消耗可表示为：$T_{prove}=num\cdot T_{read}+T_{hash}$。

在 verify 阶段，主要的时间消耗只有验证标签的传递过程，即 $T_{verify}=T_{communicate}$。

4.5　对文件更新的支持

本方案对于文件更新后的完整性验证也能给予支持。尤其是以块为基本单位的文件更新，包括块的追加、块的删除、块的修改等。

对于块的追加，用户可以选择每次均验证新追加的块，或者随机确定是否进行验证。无论用户采用何种方式进行验证，均需更新响应的验证标签。即根据原始密钥种子生成额外的 superblock 分配密钥，并对新的 superblock 进行相应散列和异或计算来更新验证标签。

对于块的删除，用户需要将涉及到删除块的 superblock 进行更新，从而更新验证标签。同时，服务器需登记删除的文件块的位置，以记录对文件块位置造成的影响。

块的修改也仅需要对相关的 superblock 进行更新。

由此可见，本方案可以完全支持文件的实时更新，并且验证标签更新的代价也相对较低。每更新一个文件标签，可能需要访问的文件块数为 $(1-(N-1)^n/N^n)\cdot 1/B\cdot(B(1+B)/2)<B/2$，这对用户来说，是可以接受的。

4.6　对小型文件验证的支持

本方案不仅支持对单个大文件进行完整性验证，也可以通过定义文件群来对大量的小文件进行整合验证。

假设用户有 2 000 个 40KB 大小的文件保存在服务器，用户期望的验证可信度为 90%。按照原始方案，服务器需要执行 2 000 次“挑战—应答”协议，共访问 2 000×229 块文件才能完成全部文件的验证，这显然是不明智的。对于一个 40KB 大小的文件来说，访问 229 个文件块来达到 90% 的可信度更会导致文件块的重复读取。

为了解决这个问题，提出了文件群的概念。用户可以将适量相关或者无关文件组成一个文件群，将这些文件作为一个整体进行准备和验证。在这种模式下，每一个文件都被看做文件群的组成部分，相当于大文件中的文件块。所有的标签生成和文件块选取都针对大文件进行。在实际计算中，需要将大文件的文件块对应到小文件中的块进行读取和计算。为了实现文件群验证，用户需要记录一个文件群里的文件排列顺序，表 1 表现了文件群验证方案的存储模型。

表 1　文件群验证存储模型

row	值
url	file1，file2，file3……filei……
num	100
current index	i
K seed	T
tags	$r_1r_2r_3\cdots\cdots r_i\cdots\cdots r_c$

与其他方案相比，本方案在准备时间、单轮验证时间、带宽消耗等方面，均有相应的优势。如表 2 所示。

表2 方案分析对比

Roschema	redundancy	block access	$computation_{server}$	bandwidth	$computation_{client}$	provable security
PDP	O(N)	O(n)	O(n)	O(n)	O(n)	√
PRF	O(N)	O(n)	O(n)	O(1)	O(1)	—
BLS	O(N)	O(n)	O(n)	O(n)	O(1)	√
ET	O(N)	O(n)	O(n)	O(1)	O(1)	√
MHT	O(N)	O(nlogN)	O(nlogN)	O(1)	O(n)	√
AS	O(pN)	O(N)	O(n)	O(1)	O(n)	—
RS	O(C)	O(n)	O(1)	O(1)	O(1)	√

5 实验结果及分析

本方案以 hadoop 云平台为基础进行了实现。其中，签名函数采用 HMAC-SHA1，对称加密算法采用 AES，以 hdfs 作为文件存储服务系统，文件验证凭据和相关参数保存在 hbase 中。

本文测试了验证准备阶段，客户端生成 100 个签名所需要的时间。实验表明，生成验证签名所需的时间与文件大小无关。并且，生成单个签名所需的时间，也在用户可以容忍的时间范围内。服务器端生成新标签的步骤与验证准备阶段相同，所需时间可以据此进行推算。

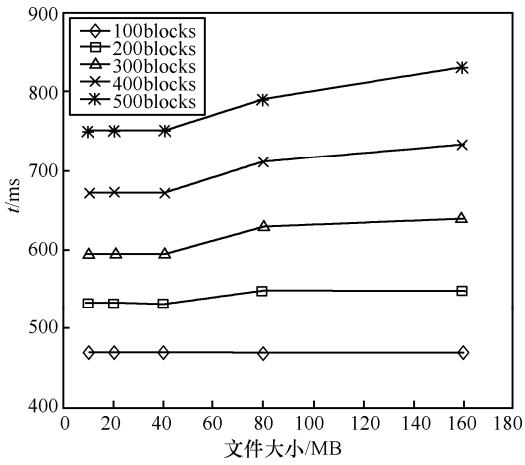

图2 不同大小的文件生成100个tag所需的时间

图2表明，生成标签所需的时间和文件大小基本无关。当文件大小从10MB增加到160MB，生成一个标签所需的时间增加不足1ms。这些微小的增长有可能是由于文件增大，随机挑选的不同块之间的距离增大，由此造成文件读取时间略有变化。这也从另外一个方面表明，挑选的文件块分布是随机的。

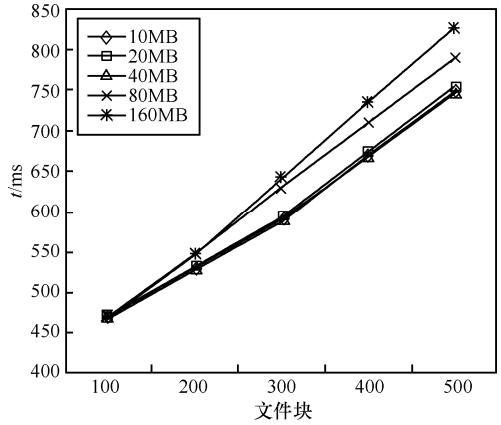

图3 选取不同抽样数目的文件块生成100个tag所需的时间

图3表明，生成标签所需的时间与需要访问的文件块数密切相关，也就是与验证方案的可信程度相关。同理，不同文件花费时间的微小差别可能与文件块间距离增大有关。

与其他方案相比，本方案在准备时间、单轮验证时间、带宽消耗等方面，均有相应的优势。如表2所示。

6 结束语

本文提出了一个适用于云存储模型的文件完整性验证方案。本方案通过挑战应答机制，要求服务器生成随机指定的文件块签名 r'，客户端比较 r 和用户提前生成验证凭据，判断文件是否完整。通过分析表明，本方案在验证可信程度、冗余存储、验证效率和带宽消耗方面均具有良好的性能。实验结果也表明，本方案对于海量文件存储的云服务模型具有很好的适应能力，并且具有极高的验证可信程度。

参考文献：

[1] BAKER M SHAH M ROSENTHAL D et al. A fresh look at the reli-

ability of long-term digital storage[A]. Proceedings of EuroSys[C]. Leuven Belgium, 2006.

[2] SHAH M A MARY B JEFFREY C et al. Auditing to keep online storage services honest[A]. Proceedings of the 11th USENIX Work-Shop on Hot Topics in Operating Systems[C]. USENIX Association Berkeley CA USA, 2007.

[3] DESWARTE Y QUISQUATER J J SAIDANE A. Remote integrity checking[A]. Proceedings of Conference on Integrity and Internal Control in Information Systems[C]. Lausanne Switzerland: IFIP, 2003.

[4] FILHO D L G BARETTO P S L M. Demonstrating data possession and uncheatable data transfer[EB/OL]. http://eprint.iacr.org/2006/150.

[5] ATENIESE G BURNS R CURTMOLA R. Provable data possession at untrusted stores[A]. Proceedings of ACM CCS[C]. Alexandria VA USA, 2007.

[6] ZHU Y WANG H X HU Z X et al. Cooperative Provable Data Possession[R]. Cryptology ePrint Archive Report 2010/234 2010.

[7] ATENIESE G KAMARA S KATZ J. Proofs of storage from homomorphic identification protocols[A]. Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology[C]. Tokyo Japan, 2009.

[8] DOMINGO-FERRER S F MARTINEZ-BALLESTE J DESWARTE A et al. Efficient remote data possession checking in critical information infrastructures[J]. IEEE Transactions on Knowledge and Data Engineering, 2008, 20(8): 1034-1038.

[9] BOWERS K D JUELS A OPREA A. Proofs of retrievability: theory and implementation[A]. Proceedings of the 2009 ACM workshop on Cloud computing security[C]. New York NY USA, 2009.

[10] DODIS Y VADHAN S WICHS D. Proofs of retrievability via hardness amplification[A]. Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography[C]. San Francisco CA, 2009.

[11] SHACHAM H WATERS B. Compact proofs of retrievability[A]. Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology[C]. Melbourne Australia, December 2008.

[12] BOWERS K D JUELS A OPREA A. HAIL: a high-availability and integrity layer for cloud storage[A]. Proceedings of the 16th ACM Conference on Computer and Communications Security[C]. New York NY USA, 2009.

[13] JUELS A KALISKI B. Pors: proofs of retrievability for large files[A]. Proceedings of the 14th ACM Conference on Computer and Communications Security[C]. New York, NY USA, 2007.

[14] CHEN E XU C J. Remote integrity check with dishonest storage server[A]. Proceedings of ESORICS[C]. Malaga Spain, 2008.

[15] SCHWARZ T MILLER E L. Store forget and check: using algebraic signatures to check remotely administered storage[A]. Proceedings of the IEEE International Conference on Distributed Computing Systems[C]. Lisboa Portugal, 2006.

[16] ERWAY C C KUPCU A et al. Dynamic provable data possession[A]. Proceedings of the ACM Int. Conference on Computer and Communications Security[C]. Chicago IL USA, 2009.

[17] YUN A SHIC H KIM Y. On protecting integrity and confidentiality of cryptographic file system for outsourced storage[A]. Proceedings of the 2009 ACM Workshop on Cloud Computing Security[C]. New York NY USA, 2009.

作者简介：

付艳艳（1986-），女，山东烟台人，中国科学院软件所博士生，主要研究方向为数据完整性保护、隐私保护。

张敏（1975-），女，安徽萧县人，中国科学院软件研究所高级工程师，主要研究方向为数据库安全理论与技术。

冯登国（1965-），男，陕西靖边人，中国科学院软件研究所研究员、博士生导师，主要研究方向为信息安全和密码学。