云存储中双备份数据的安全访问

李红卫[1,2],叶飞跃[1,2],古春生[1,2],于志敏[1,2],景征骏[1,2]

(1. 江苏理工学院计算机工程学院,江苏 常州 213001)
(2. 江苏理工学院云计算与智能信息处理常州市重点实验室,江苏 常州 213001)

[**摘要**] 数据的完整性和私密性是客户将数据存储到云存储中时关心的主要问题. 提出了一种新的 ORAM 结构,其访问云存储时间复杂度为 $O(1)$、需要 $O(cN)$ $(0<c<1)$ 客户端存储量和 $O(N)$ 服务器存储量. 将客户数据双倍份到两个服务器中以保证数据的完整性,通过 ORAM 隐藏客户对服务器的访问模式,敌手无法从客户的访问模式中获取有用的信息,从而实现了数据的私密性.

[**关键词**] 云存储,茫然 RAM,访问模式,双服务器
[**中图分类号**]TP309 [**文献标志码**]A [**文章编号**]1001-4616(2014)01-0047-05

Secure Access of Two-Copy Data in Cloud Storage

Li Hongwei[1,2], Ye Feiyue[1,2], Gu Chunsheng[1,2], Yu Zhimin[1,2], Jing Zhengjun[1,2]

(1. School of Computer Engineering, Jiangsu University of Technology, Changzhou 213001, China)
(2. Key Laboratory of Cloud Computing and Intelligent Information Processing of Changzhou City, Jiangsu University of Technology, Changzhou 213001, China)

Abstract: Data integrity and privacy become major problems that the customers concern when the data is stored in the cloud storage. The paper proposes a novel oblivious RAM construction that achieves $O(1)$ time complexity of access cloud storage, while consuming $O(cN)$ $(0<c<1)$ client-side storage and $O(N)$ server-side storage. The paper proposes a technique in which the client data is stored in two servers that each server has a client data copy in order to ensure data integrity. Oblivious RAM allows a client to hide its data access patterns from an untrusted server. The adversary can not obtain useful information from the client access pattern, and the data privacy is guaranteed.

Key words: cloud storage, oblivious RAM, access pattern, two servers

在云存储中,云服务提供商(Cloud Service Provider,CSP)能否保证客户数据的完整性和私密性是每一位客户关心的问题. 如何恢复 CSP 因设备故障或管理不当而丢失的数据,如何防范敌手对客户的攻击、篡改数据或窥视客户访问云存储的地址序列,这些都是客户存储重要且私密数据时须考虑的问题.

1 数据可恢复性证明与访问模式的隐藏

1.1 数据完整性与可恢复性证明

为了验证云存储中数据的完整性,Ateniese 等人[1]提出了可证明数据拥有(Provable Data Possession,PDP)协议,但 PDP 只能验证文件是否遭到破坏,并不能保证文件是可恢复的. Juels 等人[2]提出可恢复性证明(Proof of Retrievability,POR)协议,利用纠错码对存储的数据进行编码,当数据遭到破坏时可利用纠错码恢复数据. 这两种协议有以下共同缺点:第一,不能动态更新数据文件,其应用受到限制;第二,依赖第三方提供证明,这可能会增加 CSP 的代价,并有可能增加信息泄露的机率,给安全带来新的隐患;第三,数据访问模式暴露,敌手可通过跟踪用户对数据访问的模式,从中获取信息. 此后,许多研究者在这两种方案的基础上提出了一些改进方法及一些新的方案. Bowers 等人[3]利用多个存储服务器制作数据副本,当使

收稿日期:2013-08-10.
基金项目:国家自然科学基金(61142007)、江苏省高校自然科学研究项目(13KJB520005)、江苏省普通高校研究生科研创新计划项目(CXZZ13_0493)、常州市云计算与智能信息处理重点实验室建设项目(CM20123004).
通讯联系人:李红卫,副教授,研究方向:嵌入式软件与网络安全. E-mail:jstulhw@ gmail. com.

用POR协议检测到数据遭到破坏时,则利用副本恢复数据. 陈兰香[4]利用同态Hash算法的同态性验证数据的持有性. 朱岩等人[5]提出了一种基于交互式证明系统的POR方案,该方案满足稳固性和零知识性的安全要求,且仅需要固定大小的负载即可使得计算和通信复杂性实现最小化. 曹夕等人[6]提出云存储中数据完整性验证协议(CS-DIV),为用户检查文件的完整性提供了一种有效的方案,提高了云存储系统的可靠性和稳定性.

1.2 数据访问模式的隐藏

Goldreich和Ostrovsky[7]最早提出基于ORAM(Oblivious RAM)的隐藏访问模式模型,其主要思想是将客户的数据与哑元数据打乱,然后按一定规则(除客户外其他人一概不知)存放在外部存储器上,每当客户访问某个数据时,ORAM会访问一序列的存储地址,致使敌手无法知晓哪个地址是客户真正需要访问的. 特别地,所访问的地址序列独立于用户访问的数据,用户的数据在外存中存放的位置是变动的,次次均不相同. 另外,存放在外存中的数据使用语义安全加密模式加密,敌手很难找到用户访问外存储器的规律,从而实现了数据的保护. 在文献[7]提出的金字塔形分层算法中,整个访问过程可看作两个阶段,一是访问阶段,按访问序列对指定存储地址进行访问. 二是洗牌阶段,重新将数据与哑元数据打乱后按一定规则再次存放到外存中. 该算法需要客户存储空间大小为$O(1)$,外存空间大小为$O(N\log N)$,最好的平均时间复杂度为$O(\log^3 N)$. 但该复杂度隐含着巨大的常数项,若在洗牌阶段有客户访问数据时,则客户访问操作会因洗牌而长时间阻塞. 因此,该算法只是在理论上的探讨,离实现还很遥远. 文献[8]在金字塔形分层算法的基础上采用Cuchoo散列函数及随机希尔排序算法提出隐藏存储访问模式模型,当使用客户端$O(1)$存储量和服务器端$O(N)$存储量时,其平均时间复杂度为$O(\log^2 N)$. 文献[9]从ORAM实用角度进行研究,在拥有客户端存储容量$O(cN)(0<c<1)$,服务器存储容量$O(N)$的情况下,由若干小的ORAM构成大的ORAM,并采用背景淘汰算法实现将洗牌操作均摊到每次客户访问操作中,使得客户的一次访问操作仅需访问服务器的次数平均在20~35次之间. 文献[10]在最坏时间复杂度方面做了研究,作者提出一种新的结构,采用二叉树方式组织ORAM存储结构,数据块沿着树的边茫然地移动,当客户存储量为$O(1)$,服务器存储量为$O(N\log N)$时,平均时间复杂度和最坏时间复杂度都为$O(\log^3 N)$. 文献[9]和[10]都采用了一种新的洗牌方式,即将洗牌的负担均摊到每次访问操作中. 文献[11]利用金子塔型分层结构将客户数据分层存储于两台服务器上,高层使用文献[7]的方法进行洗牌,低层利用文献[8]的方法处理. 其性能为需要客户存储量$O(1)$,服务器存储量$O(N)$,平均时间复杂度为$O(\log N)$.

本文在上述文献的基础上提出一种新的ORAM结构,在需要少量客户端存储器和$O(N)$服务器存储容量的情况下,利用双服务器存储双份客户数据,使得客户数据的完整性得到保障,同时将洗牌过程分摊到每次访问操作中,使得对服务器的访问时间保持在常量级.

2 双服务器ORAM模型

为保证数据的完整性分别在两个服务器备份客户数据,当一个服务器上的数据块损坏时可从另一个服务器中恢复. 设两个服务器为S_0和S_1,它们仅与客户进行通信. 若数据块在两个服务器中存储的位序始终保持一致,则称这样的结构为同构ORAM模型,否则为异构ORAM模型.

2.1 同构ORAM模型

2.1.1 服务器中文件的结构

在同构ORAM模型中,两个服务器中存储的数据块的位序是一致的,且对应的存储块存储的数据的变化也是同时进行的. 假设客户文件由N个数据块组成,为了加大敌手对数据攻击的难度,在存储空间中增加N个哑元块,因此需要每个服务器的存储空间为$2N$个数据块. 利用均匀随机函数将N个数据块随机地映射到服务器中$2N$个存储块中,剩余的N个存储块存放哑元块. 从服务器的角度看无法辨别数据块和哑元块,整个结构需要总的服务器存储空间为$4N$个数据块.

2.1.2 客户端数据结构

在客户端设置一个双端口缓存队列Q,两张数据表SerMap0[$2N$]和PosMap[N],从服务器的角度无法看到这些数据结构. 假定缓存队列Q可以存放12个数据块,它用来缓存从服务器读取的数据块. 从服务器随机读取的数据块/哑元块从左端口进入Q,客户指定读/写的数据块从右端口进入Q. 客户读/写指定

的数据块不会被立即写回服务器,而是在 Q 的右端缓存,当 Q 的右端缓冲区变满时,才会将其中一块写入服务器.

数据表 SerMap0 描述存储在服务器上各数据块的属性,它有 2 个域,分别是 FlagD 和 Nonce. FlagD 为数据块标识,其值为 1 表示该块为数据块,否则为哑元块;Nonce 表示服务器存储块的新鲜度,其功能用于数据的加解密,它的初值为随机数,在服务器的每次读操作后都会为其赋一个新的随机数.

数据表 PosMap 表示用户数据块在服务器中存放的位置映射,它有 2 个域,分别是 SaveF 和 Addr0. Addr0 表示数据块在服务器中的位置,也是数据表 SerMap0 的索引. SaveF 表示数据块的位置标识,当其值为 0 时,表示数据块在服务器中存放;当其值为 1 时,表示数据块在 Q 中存放.

数据块由两部分构成,一部分为数据 B,一部分为消息验证码 m,即 $D=(B,m)$,且 $|B|\gg|m|$. 在读数据块时,先解密,然后计算验证码,并验证数据的完整性,用以下语句表示读数据块的过程:

LookUp(read,u,data);D=$\text{Decrypt}_{\text{SK(Nonce)}}$(data);//读数据块,解密,SK 为密钥函数

if(Mac(left(D,|B|)==right(D,|m|))right;else error;

在写数据块时,先计算数据块的消息验证码 m,然后用密钥 SK(Nonce)加密后写入服务器存储位置 u,其过程可用函数 LookUp(write,u,($\text{encrypt}_{\text{SK(Nonce)}}$(B,Mac(B))))表示. 由于数据块在写回服务器时重新加了密,敌手无法基于内容建立块之间的联系.

2.1.3 访问操作

客户的每次访问操作 ORAM 都会访问一序列的存储块,除客户外其他人都不知道客户的操作是读还是写,实际访问的是哪一个存储块. 用 LookUp(op,u,data)表示客户的操作,op 为 read 或 write,分别表示读操作和写操作;u 表示将要读/写的数据块号;data 表示客户端存储器地址. LookUp 操作描述如下:

(1)随机读取一个实际的数据块,并将其从 Q 的左端口存入缓冲队列中;

(2)随机读取一个哑元块,并将其从 Q 的左端口存入缓冲队列中;

(3)如果 u 已在缓冲队列 Q 中,则随机读取一个哑元块,并将其从 Q 的左端口存入缓冲队列中,然后把 u 从缓冲队列 Q 中移出,插入到 Q 的右端口队列中;否则读取指定块,并插入到 Q 的右端口队列中;

(4)如果是写操作,将 data 中的内容复制到 Q 的右端口队列队尾数据块中;

(5)同步骤(1);

(6)同步骤(2);

(7)执行 Evict()操作,进行写服务器操作;

(8)结束.

在该算法中步骤 1、2、5 和 6 随机读取两个实际数据块和两个哑元块,并依次从 Q 的左端口进入缓冲队列中. 当所需要访问的数据块不在 Q 中时,则从服务器读取并放入 Q 的右端口队列,否则随机读一个哑元块并放入 Q 的左端口队列. 在从服务器读写数据块/哑元块时,客户端中的数据结构需要作出相应的调整. 对读取的数据块/哑元块计算其 MAC 值,并与块中的 MAC 值进行匹配,若不相等,则说明数据块遭到破坏,需要报警处理. 并验证另一个备份是否也遭到破坏,若没有则恢复,并通知服务器发生的异常. 在同构 ORAM 模型中,对两个服务器的读写存储块序列是一样的,但只有一个所读数据块/哑元块备份进入 Q 队列.

在每一次执行 LookUp 时都有数据块进入 Q 中,因此需要对 Q 中的数据块/哑元块进行处理,将部分数据块/哑元块写入服务器为下一次执行 LookUp 操作腾出缓冲区空间. 通过分析 LookUp 可知,每执行一次 LookUp,最多有一个数据块从 Q 的右端口进入队列. 当 Q 的右端口队列中保持的数据块数达到 7 块时,将最早进入 Q 右端口队列的那块移向左端口队列,使得每次执行 LookUp 操作后,Q 的右端口缓冲的数据块数最多为 6;每执行一次 LookUp,最多有 5 个数据块/哑元块从 Q 的左端口进入队列,若左端口进入的数据块/哑元块不足 5 块,则用哑元块补足 5 块,并将这 5 个数据块/哑元块写入服务器以腾空左端口队列. 这个过程由 Evict 操作完成. Evict 操作描述如下:

(1)如果 Q 右端口队列有 7 个数据块,则将最久未用的数据块从 Q 的右端口队列移出,并放到 Q 左端口队列;

(2)如果 Q 左端口队列不足 5 块,则填补哑元块补足 5 块;

(3)依次为Q左端口队列中的5个数据块/哑元块在服务器上各分配一个存储哑元块的存储块;

(4)按所分配的存储块地址进行排序,按排序后的顺序依次写入两个服务器中,直到Q左端口队列为空;

(5)结束.

客户指定读/写的数据块不会立即写回服务器,而是存放在Q的右端口队列中,若右端口队列放满数据块时,才将其中一块移到Q的左端口,并写回服务器的一个随机位置.

每执行一次LookUp都会将两个任意数据块和两个哑元块读到Q的左端口队列,且会有5个数据块/哑元块写回服务器.由于写回服务器时,由客户端的随机函数均匀随机产生写回地址,从而实现将洗牌操作分摊到每次LookUp操作中,避免了集中洗牌造成的突发访问.

2.1.4 性能与安全分析

假定一个存储块大小为L字节,则需要服务器的存储容量为$4LN$B,N为客户数据块数,B表示字节.客户端所需存储空间由数据结构和缓冲队列Q决定.SerMap占用存储空间为$33\times 2N/8\text{B}=8.25N$B,PosMap占用存储空间为$(\log(2N)+1)\times N/8$B,Q大约需要$12L$B大小的存储空间.例如,若$N=1024$,$L=32$KB,则客户端所需存储空间大小是所需服务器存储空间的0.30%.因此本文所提方法占用服务器存储容量是$O(N)$,客户端存储容量也是线性级$O(cN)$$(0<c<1)$.访问效率是常量级$O(1)$,即客户每读/写一个数据块,需要读/写20块服务器的存储块,其中10次读10次写.

安全定义[10]:设客户请求访问某一数据块时,ORAM访问服务器存储块的序列为$y=((\text{op}_1,\text{u}_1,\text{data}_1),(\text{op}_2,\text{u}_2,\text{data}_2),\cdots,(\text{op}_M,\text{u}_M,\text{data}_M))$.设用$A(y)=(\text{op}_1,\text{op}_2,\cdots,\text{op}_M)$表示客户访问服务器的操作序列.如果对于任何两个客户访问序列y和y',只要其长度相等,访问服务器的操作序列相等,除客户外对其他人来说$A(y)$和$A(y')$都是在计算上不可区分的,则称该ORAM结构是安全的.

根据该定义可知对两个服务器的访问操作序列是连续5次读操作和连续5次写操作.不论客户的请求是读还是写,其访问服务器的操作序列都是一样的,因此,同构ORAM模型是安全的.但敌手可能观察到刚被写入的数据块又被访问,如果这个数据块是从Q的右端口队列中移到左端口队列然后写入服务器的数据块,那么,敌手可根据这一信息获取一些有价值的信息.因此,在这一点上,同构ORAM模型存在着安全隐患.

2.2 异构ORAM模型

异构ORAM模型与同构ORAM模型的区别是异构ORAM模型中同一个数据块在两台服务器中存储的位序不一致,将每一个数据块都按随机函数分布在两个服务器上不同的位置.

修改2.1中的数据结构以实现异构ORAM模型下的安全访问.将双端口缓存队列Q的容量扩展为可容纳17个存储块,且每个存储块都有一个标识用来说明数据块/哑元块来源于哪一个服务器.在数据表SerMap0中增加一个存储块最近是否访问过标识域FlagA.建立一个新的数据表SerMap1$[2N]$,其结构与SerMap0一样.用SerMap0描述服务器S_0,SerMap1描述服务器S_1.在数据表PosMap增加Addr1,它与Addr0分别表示对应数据块在各自服务器上的位置.当SaveF的值为0时,表示数据块在服务器中存放;当其值为1时,表示数据块在Q中存放且一个备份在服务器S_1上存放(Addr1有效);当其值为2时,表示数据块在Q中存放且一个备份在服务器S_0上存放(Addr0有效);当其值为3时,表示数据块在Q中存放,且服务器上无备份.

2.2.1 访问操作

LookUp算法描述如下:

(1)分别从两个服务器各自随机读取一个实际的数据块,并将它们从Q的左端口存入缓冲队列中;

(2)分别从两个服务器各自随机读取一个哑元块,并将它们从Q的左端口存入缓冲队列中;

(3)如果u已在缓冲队列Q中,则分别从两个服务器各自随机读取一个哑元块,并将其从Q的左端口存入缓冲队列中,然后把u从缓冲队列Q中移除,插入到Q的右端口队列中;否则在任一服务器中读取指定块,并插入到Q的右端口队列中,在另一个服务器读取任一哑元块存入Q的左端口队列中;

(4)如果是写操作,服务器上的备份失效,将data中的内容复制到Q的右端口队列队尾数据块中,并将一个备份插入Q的左端口队列,以便在执行Evict时将一个备份写到一个服务器上;

(5)同步骤(1);

(6)同步骤(2);

(7)执行 Evict()操作,进行写服务器操作;

(8)结束.

Evict 操作描述如下:

(1)如果 Q 左端口队列中分别属于两个服务器的数据块/哑元块不足 5 块,则填补哑元块补足 5 块;

(2)如果 Q 右端口队列有 7 个数据块,则将最久未用的数据块从 Q 的右端口队列移出,并替换 Q 左端口队列中的某个哑元块;

(3)依次为 Q 左端口队列中的 10 个数据块/哑元块各分配一个存储哑元块的服务器地址;

(4)按所分配的服务器地址进行排序,按排序后的顺序依次写入两个服务器中,直到 Q 左端口队列为空;

(5)结束.

2.2.2 性能与安全分析

采用异构 ORAM 模型其性能与同构 ORAM 模型相似,所不同的是当所访问的数据块是刚刚写入某个服务器的数据块时,可在另一个服务器中读取指定的数据块,这样信息就不会泄漏,访问操作更加安全可靠.

3 结语

本文提出双服务器 ORAM 模型双备份客户数据,占用线性级服务器存储容量和少量客户端存储容量,将洗牌过程均摊到每次访问操作中,实现常量级茫然访问服务器.将文件备份到两个服务器使得数据的可靠性得到保障.但该方法需要占用线性级客户端存储空间,因此减少客户端存储空间的使用仍需探讨与研究.

[参考文献]

[1] Ateniese G,Burns R,Curtmolar R,et al. Provable data possession at untrusted stores[C]//Proc of the 14th ACM Conference on Computer and Communications Security. New York:ACM Press,2007:598-609.

[2] Juels A,Kaliski B S. Pors:proofs of retrievability for large files[C]//Proc of ACM-CCS'07. New York:ACM Press,2007:584-597.

[3] Bowers K D,Juels A,Oprea A. HAIL:A high-availability and integrity layer for cloud storage[C]//Proc of CCS'09. New York:ACM Press,2009:187-198.

[4] 陈兰香.一种基于同态 Hash 的数据持有性证明方法[J].电子与信息学报,2011,33(9):2 199-2 204.

[5] 朱岩,王怀习,胡泽行,等.数据可恢复性的零知识证明[J].中国科学:信息科学,2011,41(10):1 227-1 237.

[6] 曹夕,许力,陈兰香.云存储系统中数据完整性验证协议[J].计算机应用,2012,32(1):8-12.

[7] Goldreich O,Ostrovsky R. Software protection and simulation on oblivious RAMs[J]. Journal of the ACM,1996,43(3):431-473.

[8] Pinkas B,Reinman T. Oblivious ram revisited[C]//Proc of CRYPTO. Berlin Heidelberg:Springer,2010:502-519.

[9] Stefanov E,Shi E,Song D. Towards practical oblivious ram[C]//Proc of NDSS' 12. California:the Internet Society,2011:1-19.

[10] Elaine Shi T H,Hubert Chan,Emil Stefanov,et al. Oblivious RAM with $O((\log n)^3)$ worst-case cost[C]//Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security. Berlin Heidelberg:Springer,2011:197-214.

[11] Lu Steve,Ostrovsky R. Distributed oblivious RAM for secure two-party computation[C]//Proccedings of TCC 2013. Berlin Heidelberg:Springer,2013:377-396.

[责任编辑:严海琳]