计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(4):81−90 [doi: 10.15888/j.cnki.csa.008440]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Merkle 树的安全云存储系统①

郑李伟, 王雪平

(复旦大学 计算机科学技术学院, 上海 200082)
通信作者: 郑李伟, E-mail: 382034787@qq.com

摘　要: 随着云存储的普及, 越来越多的文件存储在云存储服务器中而不是用户的计算机中, 这使得用户失去了对数据的绝对控制权, 数据安全性难以保障. 为了解决这一问题, 本文提出了一种新的安全云存储系统. 这套系统在用户态实现, 可以直接架设在计算机的文件系统上, 对计算机硬件、软件要求都很低. 通过使用分组加密算法和Merkle-B+树的设计提供了端到端的数据加密保护、完整性检查和访问权限控制等功能. 本系统使用简单, 对于用户来说是完全透明的, 降低了用户的使用门槛. 对本系统的测试结果显示, 本系统架在 NFS 文件系统上时, 在大文件环境下表现出来的 I/O 性能下降约为 5%, 这说明本系统在保证用户数据安全性、系统易用性的同时, 其性能也是较好的.

关键词: 安全存储系统; 加密文件系统; Merkle-B+树; 分组加密; 完整性检查; 访问控制

引用格式: 郑李伟,王雪平.基于 Merkle 树的安全云存储系统.计算机系统应用,2022,31(4):81–90. http://www.c-s-a.org.cn/1003-3254/8440.html

Secure Cloud Storage System Based on Merkle Tree

ZHENG Li-Wei, WANG Xue-Ping

(School of Computer Science, Fudan University, Shanghai 200082, China)

Abstract: With the popularity of cloud storage, increasingly more files are stored in cloud storage servers rather than in users' computers, which makes users lose absolute control over the data and data security difficult to guarantee. To solve this problem, this study proposes a secure cloud storage system. It is implemented in the user model and can be directly set up on the computer file system, with low requirements for computer hardware and software. Functions such as end-to-end data encryption protection, integrity check, and access control are provided through the block encryption algorithm and the design of Merkle-B+ tree. The system is simple to use and completely transparent to users, with a reduced user threshold. The test results of this system show that when it is mounted on the network file system (NFS), its input/output (I/O) performance in a large file environment is reduced by about 5%, which indicates that the system has good performance in addition to ensured user data security and system ease of use.

Key words: storage system; cryptographic file systems; Merkle-B+ tree; packet encryption algorithm; integrity check; access control

随着数据量越来越大, 越来越多的企业、个人有了上云的需求. 最近几年云存储方面的产业也飞速发展, 各大厂商纷纷推出自己的云存储产品[1]. 而用户们也喜欢使用云存储产品, 因为云存储产品不需要太多的投入就可以解放本地的存储空间, 而且具有良好的扩展性[2].

然而云存储也有坏处: 让用户失去了对文件的控制权[3]. 如果存储方不是绝对安全的, 那么就可能引发安全性方面的问题[4]. 例如, 云存储环境中常见的是以明文方式存储文件, 这样的做法缺少了完整性检查、控制访问

① 收稿时间: 2021-06-14; 修改时间: 2021-07-29, 2021-08-12; 采用时间: 2021-08-17; csa 在线出版时间: 2022-03-22

机制. 而且, 如果把敏感数据存储在他人的云存储环境里, 就可能出现泄密问题, 进而可能引发安全事故.

但是, 大多数的云存储服务商都要求用户以明文存储, 并信任他们的服务器和管理员. 然而, 事实上他们的服务并不那么可靠, 云存储泄密事件屡见不鲜. 过去一段时间, 许多采用亚马逊网络服务公司的 AWS S3 简单云存储服务的企业, 其中包括道琼斯、威瑞森、联邦快递和特斯拉等公司都发生了数据泄露的安全事故. 虽然在大多数情况下并不一定是亚马逊公司云平台的问题. 例如联邦快递和特斯拉公司, 这两家公司的 AWS S3 存储服务器没有设定密码进行保护, 导致其关键数据暴露无遗. 所以, 无论云存储服务的提供商是否值得信赖, 都应该有健全的安全机制来保证用户的数据安全.

为此, 一些存储系统提出了自己的解决方案[5,6]. 这些存储系统保证云存储安全性的主要手段是加密文件[7]. 他们将数据的访问控制权交给数据拥有者, 其他用户想要访问数据, 需要先与数据拥有者联系[8], 这样的做法在一定程度上提高了安全性, 但是这样也引入了新的问题: 第一, 数据拥有者有了更高的数据管理门槛, 甚至可能需要提供在线服务, 这可能让用户流失. 第二, 随着数据拥有者分享的用户增多, 对分享用户的管理也会越来越复杂. 第三, 将数据分享给其他用户后, 其他用户可能会做出超出分享权限的操作.

针对现有的情况来看, 安全云存储系统主要需要解决的问题是在保证数据不会被服务提供者和其他只拥有部分权限的用户窃取的情况下, 让系统的开销尽可能小. 以保证用户能在保证安全的情况下继续低成本的使用云产品服务.

本文基于以下假设: 第一, 数据需要存储在不可信的云存储服务器上, 这个云存储服务器可能会破坏、窃取用户的数据. 第二, 一些恶意用户可能会窃取其他用户的数据或者执行超出自己权限范围的操作[9]. 根据以上假设, 本文提出了一套新的安全云存储系统.

这套安全云存储系统将文件转化为一个基于 Merkle 树设计的 Merkle-B+树来存储. Merkle-B+树的各结点上存储着通过 CTR 模式的 AES 分组加密算法加密形成的文件密文. 基于这样的设计, 当发生对文件的修改时, 不需要对所有文件内容都进行重加密和重哈希, 只需要对其中被修改的部分重加密和重哈希即可, 这样既保证了文件的安全性, 又降低了完整性检查带来的重哈希以及文件重加密的开销. 本系统还通过非对称的读写密钥的分发实现了文件读写权限的分离. 并且本系统用密钥来实现文件的读写分享等功能, 不需要依赖可信第三方也不需要要求客户端长时间在线. 本文设计的安全存储系统有安全性高、加解密速度快、文件额外占用空间少、管理和使用成本低等特点.

本文第 1 节介绍系统设计的目标, 第 2 节介绍系统设计中的关键技术和具体实现, 第 3 节给出系统的性能测试结果并与其他系统进行比较, 第 4 节进行总结和展望.

1 相关工作

如引言所说, 现有的安全存储系统保证安全性的主要手段就是加密文件. 而主要的研究方向在于: (1) 实现现有的存储系统难以实现的功能. 如对加密文件的内容进行检索等功能. (2) 降低加密文件所带来的开销. 本文的研究方向在于第二点, 即提出一种开销更低的安全存储系统.

相关工作介绍如下: CFS[10] 是早期的安全存储系统. 它通过在磁盘上架设一层虚拟磁盘, 在每次往磁盘中写的时候就对文件进行加密, 从磁盘中读的时候就对文件解密的方式来实现安全存储. 而 CFS 的访问控制就是通过分享这个加密文件的密钥来实现的. 这就导致 CFS 只能实现粗粒度的分享而无法实现读写权限的分离.

Cepheus[11] 率先提出了锁盒子的概念. 在用户间进行文件分享时, 需要依赖一个可信的密钥服务器 (可信第三方) 来存储用户信息并依赖此可信第三方进行身份认证和访问控制.

Corslet[12] 在 Cepheus 的基础上进行了进一步的优化创新. 它通过锁盒子和 Merkle 树的使用, 将文件明文的 hash 值和加密文件的密钥统一起来, 而且使得在对文件进行修改时, 只需要对文件中被修改的文件块进行重加密和重哈希即可, 可以实现高效的重加密和重哈希. 但是 Corslet 依然需要引入一个可信第三方来实现访问控制和身份认证.

SiRiUS[13] 使用非对称密钥进行权限控制. 在 SiRiUS 中, 文件是被整体加解密的, 完整性校验也是对整个文件明文去计算哈希来保证的, 当发生权限撤销、文件修改时, 就需要对文件整体进行重新加密和重新哈希, 性能开销较大.

Plutus[8] 提供了组共享、懒惰撤销、随机访问、文件名加密等功能. 但 Plutus 的密钥分享方式门槛很高: 当用户想访问某文件时, 需要向文件拥有者索要密钥, 这时文件拥有者必须在线才能完成密钥的分发.

综上所述, 现有的安全存储系统在高效的加解密方式和不需要可信第三方两者之间只能实现其中一项. 故本文提出了一种新的安全存储系统, 可以既实现高效的加解密又不需要可信第三方.

2 设计目标

2.1 虚拟磁盘

本系统架设在底层文件系统之上, 与底层文件系统相互独立, 当要从底层文件系统中读或者要向底层文件系统中写的时候, 文件经过本系统会自动解密或者加密, 然后从底层文件系统中读出或者写入. 这样就形成了一个虚拟的磁盘, 用户在使用的时候感觉不到加密解密的存在, 仿佛在使用一个安全的虚拟磁盘. 这样的做法提高了系统的易用性, 降低了用户的使用门槛和管理成本.

2.2 文件共享和不依赖服务器的访问控制

本系统使用安全易用的密钥实现文件共享机制. 不需要依赖可信第三方也不需要要求用户长时间在线来保证文件的分享. 文件拥有者可以分享文件给其他用户, 而且可以选择仅分享读权限或分享读写权限. 从服务提供方的角度来看, 不需要可信第三方, 可以大大降低运营成本, 从而可以为用户提供更加便宜的产品. 从用户的角度来看, 不需要用户长时间在线来保证文件的分享, 这样降低了用户的使用门槛和使用成本, 更加简单易用.

2.3 端到端的保密性而且具备完整性保护

本系统保证只有合法的被授权的用户可以获得数据明文, 非法的用户、服务器管理员、底层文件系统管理员都无法获取数据明文. 而且本系统有完整性保护, 对数据的非法篡改可以被用户察觉, 从而保证用户得到的数据都是正确的[12]. 端到端的保密性是为了保证文件的明文不会被服务提供方窃取. 完整性检查用来检测是否有只读用户对文件进行了写操作, 从而保证用户只能做自己权限内的事, 不能做超出权限范围的事. 以上两点保证了存储系统的安全性.

2.4 轻量级的密钥管理

本系统可以让客户端不需要存储密钥, 降低被攻击而泄密的风险, 具有较高的安全性, 也降低了用户管理密钥的成本. 而且密钥的使用和存放对用户是透明的, 对于用户来说更加易用, 使用成本更低. 对于每一个文件只会产生一个对称密钥用来加密, 一方面, 这样加解密更快, 另一方面, 这样文件产生的密钥数较少, 方便密钥管理, 让密钥管理的成本更低, 也更加节约存储空间.

2.5 性能

本系统对文件的加解密都使用的是对称密钥, 这样速度更快. 而且本系统通过将文件分块化处理, 使用分块化处理的方法让文件被修改后只需要重加密其中被修改的文件块, 并只需要对这些块重新进行哈希计算, 而不需要对整个文件进行重加密和重哈希计算. 所以, 表现出较高的性能.

3 系统设计与实现

3.1 总体设计

为下文描述方便, 本文所用术语的缩写及含义如表 1 术语表. 另外, 系统的整体结构如图 1 系统结构图.

表 1 术语表

缩写	含义
SS	存储服务器 (storage server)
CL	客户端 (client)
UEK	用户加密密钥 (user encryption key)
UDK	用户解密密钥 (user decryption key)
FSK	文件密钥 (file secret key)
RHSK	根哈希加密密钥 (root hash encryption key)
RHDK	根哈希解密密钥 (root hash decryption key)
RHV	根哈希值 (root hash value)
FFK	最终文件密钥 (final file key)

存储服务器 SS 负责存放文件, 在用户视角中的一个文件, 在存储服务器中被存储为一棵 Merkle-B+树. Merkle-B+树上不仅存储有文件的密文还有文件明文的哈希值, 其中明文的哈希值用来实现完整性检验. Merkle-B+树的具体实现在后面详细介绍. 每个用户会有一对公私钥——UEK 和 UDK. 其中 UDK 是通过用户的密码直接生成的, 而 UEK 是和 UDK 匹配的公钥, 直接存放在 SS 上. 因此, UDK 只在使用时由用户键入密码后直接生成. 这样 CL 本地就不需要存储 UDK, 提高了系统的安全性. 当需要使用 UEK 时可以直接从 SS 处获得 UEK. FSK 是一个对称密钥, 用来加解密文件, 使用对称密钥来加解密文件速度比非对称密钥更快, 可以达到更高的效率. RHSK 和 RHDK 用来加解密

Merkle-B+树的根结点, 以此来实现读写权限的分离. 这部分内容之后会详细介绍.

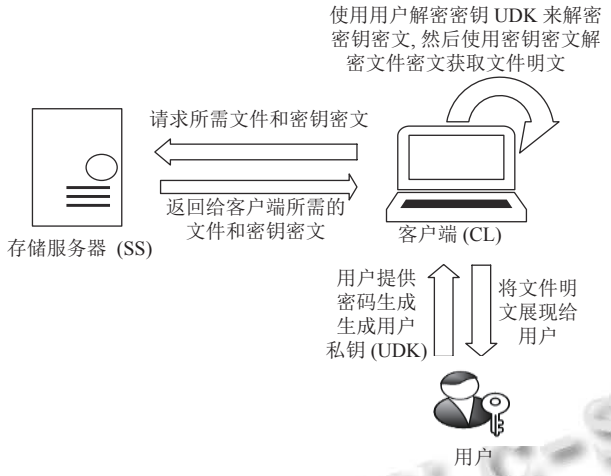

图 1 系统结构图

3.2 Merkle-B+树设计和分组加密算法

本文基于 Merkle 树[14] 设计了 Merkle-B+树来完成存储文件、进行完整性检验等功能. 如图 2, Merkle-B+树分为叶子节点和非叶子节点. 其中, 叶子节点有 6 个部分组成: hash 部分、CTR 部分、文件内容部分、指向父节点的指针、指向下一个叶子节点的指针、指向前一个叶子结点的指针. 其中 hash 部分记录了文件内容明文的哈希值. 系统使用 AES 的 CTR 加密模式, CTR 部分存储 CTR 数. 本系统将一个文件分成多个块, 最小的加解密和哈希计算粒度是文件块, 每个文件块的密文存储在一个结点的文件内容区域上. 非叶子结点由 4 个部分组成: hash 部分、指向父节点的指针、指向下一个非叶子节点的指针、指向上一个非叶子结点的指针. 其中 hash 部分值的计算方法如下:

$$Hash_{n,m}=HASH\left(Hash_{n-1,t-1}\left\|Hash_{n-1,t}\right\|\cdots\right.$$
$$\left.\left\|Hash_{n-1,t+k-1}\right\|Hash_{n-1,t+k}\right)$$

其中, ||表示拼接. 上层结点的 hash 部分是下层节点的 hash 部分的拼接的 hash 值. 这样的 Merkle-B+树设计可以让文件有以下好处:

(1) 轻量级的重加密. 通过将文件分块, 这样在对文件进行修改后, 只需要对其中的一部分进行重加密, 而不需要对整个文件继续重加密. 这样加快了加密速度, 实现了一种轻量级的重加密.

(2) 轻量级的重哈希. 为了实现完整性检验的功能, 原来对文件进行修改后需要对全文进行重哈希, 这样才能通过比对哈希值的方式来确认文件的完整性, 从而发现是否有用户篡改了文件内容. 这样就意味着在对一个文件的部分继续进行修改后需要对全文进行一次哈希计算, 这样开销较大. 将文件组织成 Merkle-B+树后, 可以只对被修改的块进行重哈希, 然后按照树的结构一层层向上重哈希就可以. 这样将 O(n) 的重哈希时间就降低到了 O(log$n$) 的时间. 而且因为 B+树的度较大, 树高较小, 这样在实际进行重加密时所消耗的时间只会更小.

(3) 能够适应多种场景. Merkle-B+树的树高、度等参数都是可以由用户进行自定义调节的, 可以由用户自己调节来适应各种应用场景. 满足各种时间和空间的要求.

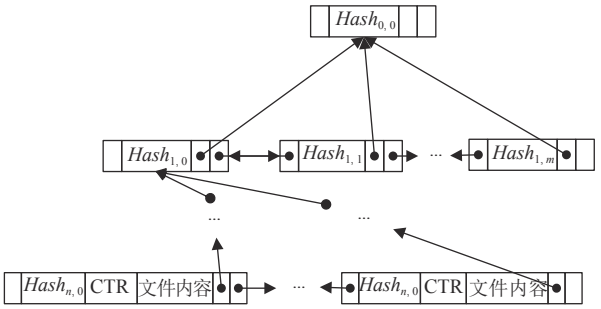

图 2 Merkle-B+树结构图

Merkle-B+树的使用过程在访问协议部分介绍.

本系统为了能让文件能够分块, 采用 AES 分组加密算法的 CTR 加密模式. 采用 CTR 加密模式是因为:

(1) CTR 加密模式不需要对文件进行填充, 方便文件块的处理而且节约空间开销.

(2) 支持并行计算, 可以充分利用 CPU 资源来加速加密、解密.

(3) 相同的明文加密后呈现为不同的密文, 不容易被统计攻击, 具有更好的安全性.

如图 2 所示, 每个叶子节点上不止存储着文件的密文, 还存储着 CTR 数值. 当用户要对文件进行解密的时候, 可以用 FSK 加密 CTR 值后得到用来加解密文件的对称密钥 FFK, 然后用 FFK 对文件进行加解密. 其中, FSK 是在文件上传时随机生成的, 对一个文件的所有文件块进行加解密都使用同一个 FSK 密钥.

至此, 文件被组织成 Merkle-B+树的形式, 文件分块存储在 Merkle-B+树中.

3.3 完整性保护和读写权限分离的实现

完整性检查如上文在 Merkle-B+介绍中所说的那样, 由 Merkle-B+的 hash 部分来完成, 具体使用方法在访问协议部分介绍. 而读写权限分离的实现是通过密钥来实现的, 即, 是否具有某种权限是由是否具有某种密钥来决定的.

文件在上传时会生成一个 FSK、一个 RHSK、一个 RHDK. 其中 FSK 用来进行加密文件. 而 RHSK 和 RHDK 用来加密和解密 Merkle-B+树的根结点, 这样就可以利用完整性检验来实现读写权限分离的.

具体来说, 将文件转化为上文说的 Merkle-B+树后, 用 RHSK 来加密根节点, 然后用 UEK 加密 RHSK 得到 RHSK 的密文. 将 FSK 的密文、RHSK 的密文、RHDK 的明文以及 Merkle-B+树存入 SS. 当用户要读取文件内容时, 用户可以从 SS 处拿到 FSK 密文, 然后用 UDK 解密 FSK 密文. 之后, 用户用 FSK 明文去解密 Merkle-B+树, 得到每一个文件块的明文, 拼接在一起就是文件内容. 按照 Merkle-B+树的结构对每个文件块重新计算 hash 值进行验证, 并对每个结点的 hash 值进行拼接, 一层层沿着路径向上求 hash 值, 得到计算的 RHV. 使用 RHDK 解密 Merkle-B+树的根哈希值, 将该值与计算得到的 RHV 进行比较, 如果一致就代表文件没有被篡改, 如果不一致就代表文件被篡改了. 这样就实现了完整性保护.

而读权限和写权限用是否有将 RHSK 分享给指定用户来区分. 因为如果用户没有 RHSK 那么他只能通过 SS 中存储的 RHDK 来解密得到 RHV, 即, 没有 RHSK 的用户只能进行 RHV 比较, 做到完整性检验来保证文件内容没有被篡改过, 却不能对文件进行写操作. 如果没有 RHSK 的用户篡改 Merkle-B+上的文件内容, 那么这个用户会因为没有 RHSK 无法对 RHV 进行加密, 在之后用户进行读取 RHV 并对 RHV 使用 RHDK 进行解密后, 得到的结果就和计算 RHV 不同, 就可以发现文件被篡改了. 这样就保证了只有读权限的用户无法对文件内容进行篡改. 而有 RHSK 的用户, 即, 具有写权限的用户在修改文件后, 会修改 RHV 并用 RHSK 进行加密. 这样之后用户进行读取的时候, 用 RHDK 对密文解密得到的值就是正确的 RHV, 可以通过完整性检验. 本系统使用每个叶子节点的哈希值保证了每个文件块的完整性, 又用非叶子结点保证了以此结点为根的子树的完整性, 所以, Merkle-B+树的根结点就保证了整棵树的完整性. 又因为只有具有写权限的用户才具有 RHSK 密钥, 这样非法用户一旦篡改了文件内容, 就会立刻因为通不过完整性检查而被发现. 所以, 可以说 RHSK 密钥保证了 Merkle-B+树的完整性, 又因为 Merkle-B+树保存了所有文件块的明文的哈希部分, 所以, 保证了加密文件的完整性.

采用 Merkle-B+树来进行完整性检查和读写权限分离的好处如上面所说. 当合法地修改文件某个或某些块的内容时, 只需要重新计算这些块对应的叶子节点的哈希值, 并沿着这些叶子节点通往根结点的路径上所经过非叶子结点的哈希值. 最后对更新后的根哈希用 RHSK 重新加密. 这样的重加密的时间复杂度是 log$n$ 级的. 如果不使用 Merkle-B+树这样的结构, 就意味着当一个文件的部分被修改时, 就需要对整个文件进行重哈希, 这样的开销较大, 特别对于大文件来说, 开销可能非常巨大. 所以, 使用 Merkle-B+树的方式是更好的.

3.4 文件共享和访问控制的实现

文件的共享和访问控制通过密钥来实现. 即, 是否拥有对一个文件的读权限、写权限的根本区别在于是否拥有某种密钥. 当用户 A 想要分享文件的读权限给用户 B 时, 会从 SS 处得到用自己 UEK 加密的 FSK, 然后用自己的 USK 解密, 得到对应 FSK 的明文, 然后用用户 B 的 UEK 进行加密, 得到 FSK 的密文上传 SS. 这样用户 B 就获得了 FSK, 就可以用 FSK 来解密 Merkle-B+树获得文件内容并进行完整性检验. 当用户 A 想要分享文件的写权限给用户 B 时, 会从 SS 处得到用自己 UEK 加密的 FSK 和 RHSK, 用自己的 USK 解密后, 然后用用户 B 的 UEK 来对 FSK、RHSK 密钥进行加密. 之后将加密后的 FSK、RHSK 上传到 SS 上. 之后用户 B 可以从 SS 得到用自己 UEK 加密的 FSK、RHSK, 用 USK 解密后得到 FSK 和 RHSK, 就可以用 FSK 来解密文件, 得到文件明文. 在修改文件内容, 进行重加密和重哈希后, 用 RHSK 加密新的 RHV, 这样就不会影响后续使用.

而使用密钥去实现文件共享和访问控制必然意味着密钥数量的增加, 这就引入了如何有效的管理密钥的问题[15]. 很多系统都提出了自己的密钥管理方式[10]. 本文采用密钥层级管理的方式来管理密钥. 层级结构如图 3 密钥层级管理示意图所示. 在本系统中, 密钥分为 2 个层级来进行组织和管理: 用户密钥、其他密钥. 其中, 其他密钥包括 FSK、RHDK 等密钥. 这样就将系统所需的密钥按照金字塔形式进行排列, 用上层密钥

来加密、解密下层的密钥, 这样用户只需要管理上层密钥, 其他密钥可以置于不可信的环境中. 这样就在保证系统安全性的前提下, 用户只需要保存少量密钥就可以达到想要的效果, 降低了用户管理密钥的成本. 在本系统的密钥层级中, 用户密钥是密钥体系的第一层, 要想分发或获取其他密钥必须先经过用户密钥. 这样用户就只需要管理用户密钥即可. 又因为用户密钥中的公钥 UEK 是可以直接存放在 SS 上的. 所以, 用户只需要管理用户密钥的私钥 UDK 就可以了. 这样大大降低了用户的管理成本.

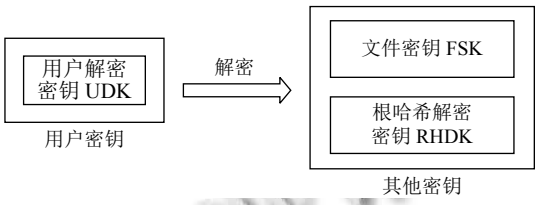

图 3 密钥层级管理示意图

3.5 访问协议

本系统能够在保证安全性的前提下, 实现轻量级的重加密和重哈希不仅和之前的系统架构、加密方法有关, 还和访问协议有关. 下面介绍本系统的访问协议:

(1) 上传文件. 上传文件的流程如下:

① 对文件进行初始分块, 按块分别计算 hash 值. 然后生成一个对称密钥 FSK, 取当前的 CTR 算子, 得到当前的 CTR 数. 用对称密钥 FSK 对 CTR 数进行加密, 用得到的 FFK 与文件明文进行异或来做加密. 至此, 得到了分块的文件密文、分块的明文 hash 值和 CTR 值.

② 将分块的明文 hash 值按照 Merkle-B+树的层次关系沿着树的路径计算上层的非叶子结点的 hash 值, 并完成树的构建. 然后生成一对非对称密钥 RHSK 和 RHDK, 用 RHSK 加密 Merkle-B+树的根哈希值. 形成最终的 Merkle-B+树上传到 SS.

③ 上传用户用自己的 UEK 去加密 FSK、RHSK, 然后将加密后的 FSK、RHSK 的密文以及 RHDK 的明文上传到 SS.

(2) 读取文件. 读取文件的流程如下:

① 从 SS 处获得加密后的 FSK 密文以及 RHDK 明文, 同时获得 Merkle-B+树. 用自己的 UDK 解密 FSK. 用 FSK 解密 Merkle-B+树叶子节点上的分块文件密文, 从而获得文件明文.

② 为了验证文件是否被篡改过. 需要进行完整性检查. 对每个文件块明文求 hash 值, 然后按照树的结点路径层层向上计算出根 hash 值. 用 RHDK 解密根哈希结点, 比较两者是否一致就可以判断出是否被篡改过.

(3) 写文件. 写文件的流程如下:

① 执行读文件的过程. 并从 SS 处获取用自己的 UEK 加密的 RHSK 密文.

② 在用户对文件进行修改后, 将被修改的块用 FSK 密钥加密 CTR 数生成的密钥来重新加密被修改的块, 来完成部分重加密.

③ 然后对这些块的明文进行 hash 计算, 得到新的 hash 值, 按照结点的路径层层向上更新结点的 hash 值. 对根 hash 用 RHSK 加密后形成新的 Merkle-B+树上传到 SS.

(4) 分享读权限. 为了方便叙述, 将分享权限的用户称为 A, 被分享权限的用户称为 B. 分享读权限的流程如下:

① 用户 A 从 SS 处获取用 A 用户的 UEK 加密过的 FSK 密钥和用户 B 的 UEK.

② A 用 UDK 解密 FSK 密文, 得到 FSK 明文, 用用户 B 的 UEK 加密 FSK 明文, 得到用户 B 加密的 FSK 密钥的密文. 然后上传到 SS.

(5) 分享写权限. 为了方便叙述, 将分享权限的用户称为 A, 被分享权限的用户称为 B. 分享写权限的流程如下:

① 用户 A 从 SS 处获取用 A 的 UEK 加密过的 FSK、RHSK 密文以及用户 B 的 UEK.

② A 用 UDK 解密 FSK 密文和 RUSK 密文, 得到 FSK、RHSK 明文, 用 B 的 UEK 加密 FSK、RHSK 明文, 得到用 B 的 UEK 加密的 FSK、RHSK 密钥的密文. 然后上传到 SS.

(6) 权限撤销. 权限撤销的流程如下:

① 拥有文件所有权的用户向 SS 提出撤销某用户的权限. SS 删除所有对应得 FSK、RHSK、RHDK 以及 Merkle-B+树.

② 重新执行上传操作.

③ 重新分享给其他用户权限.

3.6 性能调优

因为考虑到实际使用中会有很多用户并发的访问文件, 所以, 本系统实现了一套锁机制来保证并发安全, 支持多线程并发读同一个文件[16]. 本系统使用惰性重加密、重哈希的方式来提高性能. 比如, 当对 Merkle-

B+树进行修改时, 不会马上进行重加密和重哈希, 而是等到用户确认上传或者确认保存的时候才会进行重加密和重哈希, 这样就减小了开销. 完整性检查也是惰性的, 只有当用户打开文件的时候才会进行完整性检查. 检查完毕之后会修改一个“是否完整”的标记, 从而保证完整性检查是惰性的且只做一次. 另外, 还有缓存机制, 会将读取过的明文缓存起来, 再次读取的时候可以从本地的缓存中直接读取, 省去了解密的过程, 降低了系统开销.

3.7 安全性分析

本系统采用加密的方式来保证系统的安全性. 可以说是将系统的安全性委托给加解密的体系. 与 Corslet 对比来看, 两个系统都是对文件进行切分后使用对称密钥来对文件进行加密. Corslet 在分享文件权限时依赖可信第三方, 本系统通过非对称密钥体系来完成相同的功能, 而这样的非对称加密方式本身就是一种安全性很高、工业界普遍采用的加密方式. 而且两系统在文件服务器上存储的都是文件密文, 不易被攻击. 所以, 总体来看, 本系统的安全性至少不会比 Corslet 差.

4 性能测试实验

在实验部分中, 本系统被架设在网络文件系统 NFS 上, 之所以架设在网络文件系统 NFS 上, 是因为 NFS 可以很好的模拟实际使用场景. 在实际使用中, 当对文件执行写操作时需要通过网络将文件发送到服务器上, 而执行读操作时也需要通过网络从服务器处获取文件. 而在 NFS 客户端上读写的同时, 会把写的内容通过网络发送到 NFS 服务器上, 把需要读的内容通过网络从 NFS 服务器上获取过来. 所以, NFS 服务器能很好的模拟实际情况. 之后对架设了本系统、架设了 Corslet 系统[12]、没有架设任何系统的 NFS 系统进行读写实验, 比较三者的读写耗时差异. 在结果分析部分对多种现有的安全存储系统进行综合比较, 并根据提出各系统的论文中, 给出的各系统与未架设系统的 NFS 文件系统上的读写耗时差异的结果, 来进行定量的分析.

4.1 测试环境介绍

文中用 2 台服务器来对本系统进行性能测试. 其中一台作为 NFS 服务器, 另一台作为 NFS 客户机, 分别作为系统的服务器和客户机. 其中, 服务器的配置为 2 核 CPU、2 GB 内存, 客户机的配置为 4 核 CPU, 4 GB 内存. 两台机器以局域网连接. 在加密算法的选择上, 使用 AES-256 作为文件加密的加密算法, 以 CTR 加密模式作为本系统的 AES 加密模式. 以 RSA-512 算法来生成 RHSK 和 RHDK 用来实现读写权限的区分.

4.2 多系统性能比较测试

在这部分中实现了本系统和 Corslet 系统. 比较架设了本系统的 NFS 客户端、架设了 Corslet 系统的 NFS 客户端、没有架设任何系统的 NFS 客户端三者的读写文件速度.

4.2.1 大文件多系统性能比较测试

这部分中本系统先创建一个 100 MB 的文件[17], 然后以读写模式打开此文件, 将文件平均分成若干块, 来生成对应的 Merkle-B+树. 而 Corslet 也按照相同的分块粒度来对文件进行分块. 而未架设任何系统的 NFS 客户端则不对文件分块. 比较多种系统在不同切分方法下, 在实验中表现出来的从磁盘上读取全部文件内容明文所消耗的时间、将全部文件明文转化为密文并向磁盘写入所消耗的时间、读取文件后顺序修改部分文件后将新文件重新写入磁盘所消耗的时间、读取文件后随机修改部分文件后将新文件重新写入磁盘所消耗的时间. 实验中计量的这 4 个量分别代表了下载文件所消耗的时间、上传新文件所消耗的时间、读取文件并顺序修改部分内容所消耗的时间、读取文件并随机修改部分内容所消耗的时间. 将文件分为 800 块、160 块的实验结果如图 4、图 5 所示.

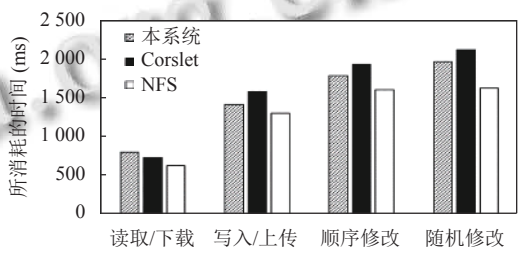

图 4 大文件分 800 块的多系统各操作开销比较图

可以看出与没有架设任何系统的 NFS 相比, 无论是架设了 Corslet 还是架设了本系统, 在读取、写入、修改文件上增加的时间开销都不大. 这证明本系统和 Corslet 系统都是在大文件读写环境下开销较小的系统. 而在写入和修改操作上本系统相比 Corslet 系统所消耗的时间的略少的原因是 Corslet 系统以文件块的 hash 值作为文件块的加密密钥. 所以, 在将文本转化为 Corslet 中的锁盒子结构[4] 时, 为了能够把密钥安全的存储在不可信任的环境下, 需要对所有文件块的

hash 值, 即密钥都进行加密. 而本系统一个文件只生成一个 FSK 密钥, 也只需要对 FSK 进行一次加密. 而求 hash 值的动作两者都有, 这部分开销基本一致. 所以, 在写入上, 因为本系统的密钥加密次数相对较少, 所以速度略快. 而在修改上, 也是因为 Corslet 系统的密钥是由文件块的 hash 值来充当的, 所以, 当一个文件块发生修改就需要修改密钥, 而这样的动作又需要对新 hash 值, 即新密钥值, 进行加密. 因为加密次数较多, 所以开销更大, 所消耗时间稍多.

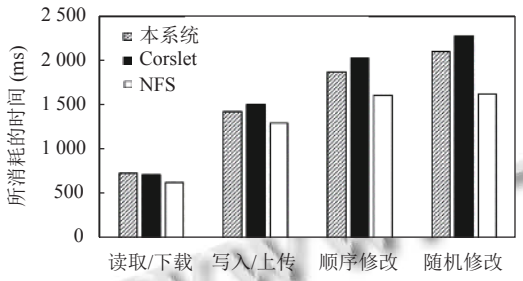

图 5 大文件分 160 块的多系统各操作开销比较图

从产生的密钥数的角度看, Corslet 系统每个文件块都会求 hash 值, 并以这个 hash 值作为密钥, 而且这个密钥需要加密后存储在不可信任环境下, 这样一个文件就会产生多个密钥, 这样多的密钥不仅仅会像上面说的那样, 让读写变慢. 而且管理它们的成本也很高. 而本系统整个文件只会产生一个密钥, 用同一个密钥来对每个文件块来进行加解密. 这样本系统在密钥所需的空间上是占一定优势的, 而且密钥数量更少就意味着更小的管理开销和使用成本.

另外, 可以看到无论是本系统还是 Corslet 系统随机写入所消耗的时间相比顺序写入都较多一些. 这是因为随机写入比顺序写入会涉及更多的文件块, 两个系统都需要对更多的文件块进行重加密、重哈希, 所以所消耗的时间略长. 另外, 从按照 800 块进行划分和按照 160 块进行划分的实验结果可以看出, 划分的块数越多, 在修改时表现的性能越好, 因为每个块更小, 重加密和重哈希的开销也更小. 而且还可以看出分块数量越少, 那么在上传和下载文件上的耗时和未架设安全存储系统的 NFS 文件系统相比差别越小. 这是因为分块数量越多, 那么 Merkle-B+树的结点也越多, 构建和读取这样一棵树的开销也更多. 所以, 对文件的切分越细, 分成越多块, 文件上传和下载所附加的额外开销就越大, 而文件在修改时所需要的额外开销就越小. 所以, 用户可以根据实际情况, 调节分块数量来满足自己的特异性需求.

4.2.2 小文件多系统性能比较测试

这部分中本系统创建一个 20 KB 的文件, 然后以读写模式打开此文件, 按照和大文件一样的实验方法进行实验, 结果如图 6、图 7 所示.

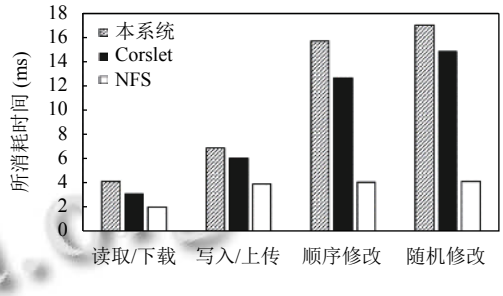

图 6 小文件分 160 块的多系统各操作开销比较图

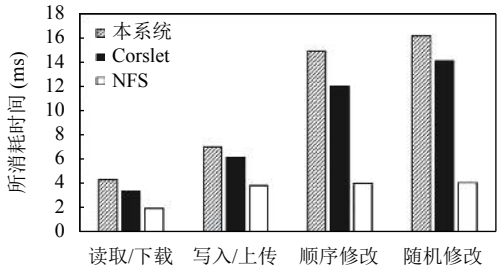

图 7 大文件分 800 块的多系统各操作开销比较图

可以看出在小文件环境下, 无论是本系统还是 Corslet 的开销都是较大的. 这主要是因为本系统在构建 Merkle-B+树时需要添加 CTR 数等信息, 而添加的这些信息所带来的开销与分块个数有关而与文件大小无关的. 所以, 在大文件的情况下这些开销相对文件读写而言很小. 在这样的分块多但是文件小的场景下, 这部分开销相对来说占比较大. 表现为图中的读写所花费时间高于没有架设任何系统的 NFS 系统. 另外, 可以看到在修改操作下, 本系统相对没有架设任何系统的 NFS 客户端所花费的开销多较多. 这主要是因为在修改后需要重新计算 hash 值, 而重新计算 hash 值的开销也是与块数量有关的, 在这样多分块但是文件较小的场景下, 这部分开销占比较大. 再加上本身修改操作包含写操作, 所以额外花费的开销相对单纯写操作来说更大.

针对这种情况, 可以采用添加缓存机制来进行优化. 当一个文件不是第一次被读取时, 可以直接从缓存中读取该文件的内容, 而不需要再从磁盘上读取, 也不需

要再进行密文的解密. 这样可以降低开销, 提高读写效率.

另外, 可以看出在小文件下按照 800 块进行划分和按照 160 块进行划分最终消耗的时间差别不大. 这是因为小文件情况下本身构建 Merkle-B+树和加解密的开销的占比就较大, 虽然划分更细会给修改操作带来了性能上的提升, 会让上传和下载操作的性能降低, 但是相对于本身占比就较大的 Merkle-B+树的构建开销和加解密开销, 这样的性能变化对于整体来说并不明显.

4.3 结果分析

本系统和其他经典安全存储系统的比较如表 2 所示. 同时本文还通过比较安全存储系统架设在 NFS 文件系统前后, 读写性能下降的程度作为量化比较安全存储系统加密开销的依据.

表 2 安全存储系统相关工作对比

系统	密钥类型	加密粒度	完整性检查	懒撤销[18]	密钥分发	可信第三方	性能
SiRiUS[13]	非对称密钥	文件	整个文件求哈希	无	为拥有权限的用户加密文件密钥, 并存放在文件的末尾.	不需要	比NFS性能下降约80%
Plutus[8]	非对称密钥	文件块	Merkle树	有	需要联系在线的拥有者获取密钥.	不需要	没有与底层文件系统进行比较
Corslet	对称密钥	文件块	Merkle树	有	通过可信第三方进行身份验证, 通过的可以获得密钥.	需要	比NFS性能下降约5%
本系统	对称密钥	文件块	Merkle-B+树	无	为拥有权限的用户加密文件密钥, 并将加密后的文件密钥存放在服务器上供用户获取.	不需要	比NFS性能下降约5%

SiRiUS 系统对文件加密使用的密钥是非对称密钥, 这样就导致该系统的加解密速度较慢, 而且该系统的加密粒度是文件级的, 这样对文件的一点局部修改就需要对整个文件全部进行重加密, 开销较大. 所以最终该系统整体读写性能表现得较差. 由提出 SiRIUS 的文献[13] 的实验结果可知, SiRiUS 部署在 NFS 文件系统上之后与未部署前相比性能下降约 80%.

Plutus 系统采用共享密钥的方式来进行密钥分发. 当其他用户想要获得文件的访问权限时, 需要向文件拥有者申请, 此时要求文件拥有者必须在线, 并在线的将文件密钥分享给其他用户. 这样的分享机制比较低效, 它要求用户必须实时在线才能完成密钥分发. 这就导致用户的使用门槛和管理成本增加, 不具有易用性.

Corslet 系统使用对称密钥来加解密文件. 文件按照文件块粒度进行分割来实现部分重加密机制. 这都使得 Corslet 的读写速度较快. 但是 Corslet 也存在需要可信第三方的问题, 这使得 Corslet 的运营和管理成本较高. 另外, Corslet 以文件块的 hash 值作为密钥, 虽然这样的做法可以将完整性检查和密钥统一起来, 但是这就意味着一个文件会有文件块数量个数的密钥, 密钥数量的增大必然导致管理成本的增加, 而且也意味着更大的空间开销. 从提出 Corslet 系统的论文的实验结果可知, 将 Corslet 部署在 NFS 文件系统上之后大约读写性能下降 5%.

Plutus 的创新在于系统实现了一些前人未曾实现的功能, 但在性能上的表现不够出色. 从性能比较上看, 本系统在性能上有明显优势. 因为 Plutus 对文件都是全文加密的, 所以重加密和重哈希的开销很大. 而且 Plutus 的文件分享要求文件拥有者必须长期在线. 所以, 从性能的角度和易用性角度来看, 本系统有明显优势.

从实验结果来看, 本系统在文件的上传、下载、修改上表现出来的性能与 Corslet 相当, 同样是与没有架设系统时相比读写性能下降大约 5%. 而且不需要像 Corslet 那样依赖可信第三方的介入, 具有不依赖可信第三方的简单性和易用性. 而与 SiRiUS 系统相比, SiRiUS 系统部署在 NFS 文件系统后, NFS 文件系统性能下降约 80%, 而本系统仅下降约 5%, 所以本系统在读写性能上有较大优势. 故本系统是一种既实现高效的加解密又不需要依赖可信第三方的安全存储系统. 与其他的安全存储系统相比, 在使用成本、简单性、文件加解密的开销上有较大优势.

相对于其他的安全存储系统来说, 本系统在不需要可信第三方的情况下, 使文件发生修改后不需要对全文进行重加密和重哈希, 而只需要对修改的部分文件块进行重加密和重哈希, 将重加密的开销大大降低, 而重哈希也只需要对修改块进行重哈希计算, 并沿着 Merkle-B+树到根节点的路径重新计算哈希, 将重哈希的复杂度从 O(n) 降低到 O(log$n$). 而且以一个对称密钥

作为文件加解密密钥, 一方面相对非对称密钥来说这样做文件的加解密速度更快. 另一方面这样让一个文件只有一个密钥, 也避免了密钥数量的膨胀. 再者, 也不需要像 Corslet 那样修改文件内容后需要再修改密钥, 使得在一次文件修改后需要随之变化的内容尽可能少. 所以本系统能在表现出速度上的高效的同时, 也不会带来太多的空间开销. 而这些设计都让本系统在大文件场景下取得较好的结果. 特别在大文件频繁修改场景下, 表现出来的性能更好.

而本系统也进行了一些工程上的优化, 包括锁机制、懒加载、缓存机制等, 以此来降低在小文件场景下的开销. 综合来看, 本系统是可以保证用户信息安全, 而且还能够表现出高性能的高效系统.

5 结论与展望

文中提出一种新的安全云存储系统架构, 确保用户可以安全的将数据存储在不信任的环境下, 而且由于额外存储的密钥较少, 所以不会带来太大的空间开销. 而且本系统具有很好的扩展性, 可以直接架设在文件系统之上, 使用简单, 对用户透明, 用户可以很方便的使用该系统. 此外, 本系统不依赖可信第三方, 降低了系统的使用和管理门槛. 另外, 本系统还提供了丰富、高效的安全机制, 包括私密性保护、完整性保护、文件访问控制等. 通过非对称密钥的分发实现了读写权限的分离, 实现了更加精细的访问控制. 通过 Merkle-B+树的设计实现了修改文件时的轻量级重哈希和重加密, 让系统表现出较好的性能. 实验结果表明, 架设了本系统的 NFS 系统相对于没有架设本系统的 NFS 系统读写性能下降约 5%. 总的来说, 本系统是可以保证用户信息安全, 而且还能够表现出高性能的高效系统.

未来将进一步扩充高效实现安全存储系统如文件去重、密文搜索等其他功能的方法.

参考文献

1 Vengala DVK, Kavitha D, Kumar APS. Three factor authentication system with modified ECC based secured data transfer: Untrusted cloud environment. Complex & Intelligent Systems, 2021. [doi: 10.1007/s40747-021-00305-0]

2 李思莉, 杨井荣, 苟强. 轻量级 Web 服务器的高并发技术研究与实现. 计算机技术与发展, 2020, 30(10): 75–78, 85. [doi: 10.3969/j.issn.1673-629X.2020.10.014]

3 Prajapati P, Shah P. A review on secure data deduplication: Cloud storage security issue. Journal of King Saud University—Computer and Information Sciences, 2020. [doi: 10.1016/j.jksuci.2020.10.021]

4 余海波, 陈洁, 张凯. 一种基于区块链的安全云存储方案设计. 计算机应用与软件, 2021, 38(4): 64–68. [doi: 10.3969/j.issn.1000-386x.2021.04.011]

5 Miller EL, Long DDE, Freeman WE, *et al*. Strong security for network-attached storage. Proceedings of Conference on File and Storage Technologies. Monterey: USENIX, 2002. 1–13.

6 O'Shanahan DP. CryptosFS: Fast cryptographic secure NFS [Master's thesis]. Dublin: The University of Dublin, 2000.

7 余宇劲, 凌捷. 基于多云存储的 Android 密钥管理技术. 计算机应用与软件, 2020, 37(9): 286–290. [doi: 10.3969/j.issn.1000-386x.2020.09.047]

8 Kallahalla M, Riedel E, Swaminathan R, *et al*. Plutus: Scalable secure file sharing on untrusted storage. Proceedings of FAST'03 Conference on File and Storage Technologies. San Francisco: USENIX, 2003. 29–42.

9 李晖, 孙文海, 李凤华, 等. 公共云存储服务数据安全及隐私保护技术综述. 计算机研究与发展, 2014, 51(7): 1397–1409.

10 Blaze M. A cryptographic file system for unix. Proceedings of the 1st Conference on Computer and Communications Security. New York: ACM, 1993. 9–16.

11 Fu KE. Group sharing and random access in cryptographic storage file systems [Master's Thesis]. Massachusetts: Massachusetts Institute of Technology, 1999.

12 薛矛, 薛巍, 舒继武, 等. 一种云存储环境下的安全存储系统. 计算机学报, 2015, 38(5): 987–998.

13 Goh EJ, Shacham H, Modadugu N, *et al*. SiRiUS: Securing remote untrusted storage. Proceedings of Network and Distributed System Security Symposium. San Diego: NDSS, 2003. 131–145.

14 Merkle RC. A digital signature based on a conventional encryption function. Proceedings of Conference on the Theory and Application of Cryptographic Techniques. Santa Barbara: Springer, 1987. 369–378.

15 傅颖勋, 罗圣美, 舒继武. 安全云存储系统与关键技术综述. 计算机研究与发展, 2013, 50(1): 136–145.

16 邓彬, 成卫青. 基于改进慢启动算法的大文件快速传输. 计算机应用研究, 2020, 37(3): 860–863.

17 张俊林, 查东辉. 增强云存储系统安全功能的方法探讨. 网络安全技术与应用, 2021, (4): 69–70.

18 王铁滨, 杨晶, 齐秀丽. 基于属性加密的云存储安全技术. 电子技术与软件工程, 2021, (6): 259–260.