一种云存储环境下的安全存储系统

薛 矛[1)] 薛 巍[1),2)] 舒继武[1),2)] 刘 洋[2)]

[1)](清华大学计算机科学与技术系 北京 100084)

[2)](清华大学信息科学与技术国家实验室 北京 100084)

摘 要 如今,数据越来越多地被选择存放在云存储环境,而非个人电脑中.这使得用户失去了对数据的完全控制,从而难以保证数据的安全性.为了解决此问题,文中提出了一种新的安全云存储系统架构.基于这套架构,文中设计并实现了一个安全云存储系统——Corslet. Corslet 可以直接架在已有的云存储系统之上而无需对其进行任何改变,同时提供端到端的数据私密性保护、完整性保护以及访问权限控制等功能. Corslet 使用简单,用户只需在客户端存放他们的身份证书即可.对 Corslet 的测试结果显示,Corslet 架在 NFSv4 集群之上 I/O 性能下降不到 5%,证明 Corslet 在提高用户数据安全性的同时,其性能也是可以接受的.

关键词 安全存储系统;加密文件系统;私密性;完整性;访问控制

中图法分类号 TP309 **DOI 号** 10.3724/SP.J.1016.2015.00987

A Secure Storage System Over Cloud Storage Environment

XUE Mao[1)] XUE Wei[1),2)] SHU Ji-Wu[1),2)] Liu Yang[2)]

[1)](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

[2)](*Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Nowadays, data has been increasingly shared among different users inside the cloud storage systems, instead of being owned by any single private user, which makes an ordinary user usually does not have the control permission over the whole system, thus hard to secure data storage or data sharing of his own files. To solve this problem, this paper proposes a new secure cloud storage system architecture. Based on this architecture, this paper designs and implements a secure cloud storage system called Corslet. Corslet can run directly on deployed underlying cloud storage systems without modification, while bringing end-to-end confidentiality and integrity as well as efficient access control for user data. For individual users, Corslet is easy to use, the only thing to keep locally is their certifications. The experiments and standard benchmark results show that Corslet over NFSv4 cluster brings acceptable I/O throughput reduction which is less than 5%, proving that Corslet can provide enhanced security for user data while maintaining acceptable performance.

Keywords secure storage system; cryptographic file systems; confidentiality; integrity; access control

---

收稿日期:2010-08-01;最终修改稿收到日期:2014-07-15.本课题得到国家自然科学基金(60925006,61232003)、国家"八六三"高技术研究发展计划重大专项课题子课题(2009AA01A403)、国家科技重大专项子课题(2013ZX03002004-003)资助.**薛 矛**,男,1986 年生,硕士研究生,研究兴趣包括大规模安全存储系统. E-mail: lionxuemao@gmail.com. **薛 巍**,男,1974 年生,博士,副研究员,研究兴趣包括并行算法设计和网络存储. **舒继武**,男,1968 年生,教授,博士生导师,研究兴趣包括网络/云存储系统、存储安全与可靠性、并行/分布式处理. **刘 洋**,男,1983 年生,硕士,研究兴趣包括大规模安全存储系统.

1 引 言

在这个数据爆炸性增长的时代,随着云存储的迅猛发展,越来越多的人或服务开始选择使用云存储环境来存放自己的资料[1-2].云存储环境一般采用按使用付费的方式,给使用者带来了不少好处:无前期投入,节省了管理开销,良好的可扩展性和很高的存储资源利用率.

然而,云存储环境让数据的拥有者失去了对数据的完全控制,这使得数据的安全性面临一系列的威胁[3].例如,云存储环境中的数据一般都是以明文方式存储,缺少完整性保护、可靠的用户身份校验和访问控制机制.如果把敏感数据(如经济类文档、个人医疗记录等)存放在由他人控制的云存储环境中,随着云存储的使用者增多,以上问题将变得越发严重.

对数据进行加密是当前保护数据私密性的主流方法[4-5].但是,大多数的云存储服务提供商都要求用户信任他们的存储服务器和系统管理员.部分服务提供商声称自己提供了一套优秀的安全机制来确保用户数据的安全性,然而,Verizon 在 2010 年度数据泄露调查报告中指出,有 49%的数据泄漏是由内部人士造成的,权限滥用占到了数据泄露的很大一部分,48%的数据泄漏是由于用户恶意滥用访问企业信息的权利而造成的①.由此看来,使用云存储的用户实在很难有理由完全地信任服务提供商.

在一些存储系统中,数据的访问控制权交给数据拥有者来完成,其他用户想要访问数据,需要先与数据拥有者联系,这在一定程度上减轻了安全威胁.但是这种方案却引入了新的问题:第一,数据拥有者(用户)需要提供较复杂的数据管理服务,并且可能需要提供在线服务;第二,当用户数量庞大、共享用户数很多时,管理起来并非易事.

针对以上问题,本文主要考虑了一种多用户共享云存储环境.在该环境中,数据拥有者将共享数据存放在不可信的云存储服务提供商处,而其他用户则在不可信的网络环境中可根据自身所被赋予的访问权限对该数据进行操作.在数据存储和访问的过程中,假设云存储服务提供商和网络中的恶意用户有可能对数据采取窥探甚至篡改等行为,而用户也将可能尝试超出其合法权限之外的数据操作.在这个前提假设下,本文提出了一套新的安全云存储系统架构.该架构使得用户在不可信的云存储服务提供商、不可信的网络环境下,依然能够得到对数据安全性与完整性的保护,并使数据的访问控制更加高效可靠,同时保证用户无法进行超出其权限的操作.依据这套安全系统架构,文中实现了一个安全云存储系统原型——Corslet. Corslet 是一个具有栈式加密文件系统的安全云存储系统,它可以架在任何提供 POSIX 标准接口的文件系统之上,而无需对已有的文件系统做任何改变. Corslet 以独立于一切云存储服务提供商的第三方身份存在,为用户提供数据私密性、完整性保护及访问控制服务,让用户消除对数据安全问题的担忧.

本文第 2 节介绍系统设计的原则与预设;第 3 节介绍系统设计中的关键技术与系统实现;第 4 节给出系统的性能测试结果与分析;第 5 节介绍相关工作;第 6 节进行总结.

2 设计原则

与底层文件系统相互独立. Corslet 的设计目的是为已有的云存储系统提供安全机制,因此必须做到与底层文件系统相互独立,保证在使用 Corslet 的时候,不需要对底层文件系统进行任何修改.

文件共享与访问控制. Corslet 必须向用户提供安全易用的文件级共享与访问控制机制.文件拥有者可以指定文件能够被哪些用户进行怎样的访问.

端到端的私密性与完整性保护. Corslet 必须保证只有被合法授权的用户才能获得数据明文,非法的用户以及底层文件系统的管理员均无法获得数据明文.对数据的非法篡改必须能被发现,从而保证用户得到的数据是正确的.

密钥管理. Corslet 的密钥管理机制的设计原则是:用户在使用 Corslet 时,不需要在本地存放任何文件密钥.也就是说,Corslet 的密钥管理机制对用户来说是透明的,增加了易用性和安全性.

密钥分发. Corslet 需要有一套合理高效的密钥分发机制,来保证合法用户能获得他想访问的文件的密钥.

懒惰撤销. 在 Corslet 中引入懒惰撤销机制[6]来减少这部分的性能开销.当权限撤销发生时,Corslet

① Verizon 2010 年度数据泄露调查报告概述. http://netsecurity.51cto.com/art/201008/215676.htm

并不马上对文件重新加密，而是等到文件被修改时才对被修改的内容重新加密.

性能. Corslet 中涉及到的一切加解密(除 PKI 身份认证体系外)，都使用对称加解密. 同时，Corslet 还需要引入缓存机制来避免重复的计算开销和 I/O 开销. 另外 Corslet 必须尽量减少由于安全机制引入所带来的磁盘空间和网络带宽的消耗.

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见表 1. Corslet 安全云存储系统由三部分组成：存储服务器(Storage Server)、客户端(Client)和验证服务器 AS(Authentication Server). 如图 1 所示.

表 1　术语表

缩写	含义
AS	验证服务器(Authentication Server)
AEK	验证服务器加密密钥(AS Encryption Key)
ASK	验证服务器签名密钥(AS Signature Key)
ACL	访问控制列表(Access Control List)
EALG	加密算法(Encryption Algorithm)
EMOD	加密模式(Encryption Mode)
LBK	锁盒子密钥(Lockbox Key)
FSK	文件签名密钥(File Signature Key)
HMAC	ACB 内容验证码(Hash-based Message Authentication Code)
ACB	访问控制块(Access Control Block)
RHi	根哈希(ith Root Hash in the root hash list)

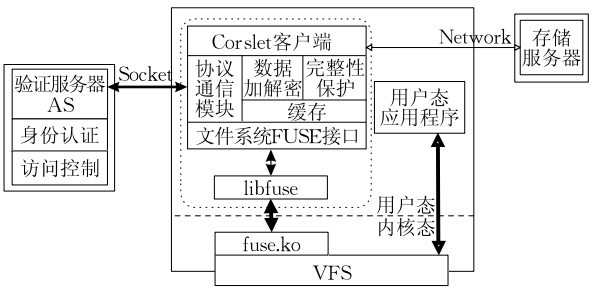

图 1　Corslet 系统结构

存储服务器负责存放文件，用户视图中的一个文件，在存储服务器中被分成两个文件来存放，分别称之为数据文件(data file，或 d-file)和安全元数据文件(secure metadata file，或 smd-file). 数据文件中存放的是文件密文，安全元数据文件中存放了与此文件相关的安全信息，例如访问控制块(Access Control Block，或 ACB)、根哈希链表(Root Hash List，或 RHL)、Merkle Tree 等，具体内容将在后面进行详细叙述.

验证服务器 AS 是整个系统信任的根. AS 的逻辑非常简单，它负责验证用户身份、处理用户的文件访问请求、将相关密钥分发给合法用户等. AS 端只需要储存两个对称密钥 AEK 和 ASK，而不需要储存其他任何信息，所涉及到的操作只是少量的对称密钥加解密、计算 MAC 等. 这种简单的设计能够带来以下好处：

(1) 低开销. 由于计算简单、无磁盘 I/O 开销，AS 可以轻松地同时响应多个请求.

(2) 让系统更可靠，可用性更强. 由于 AS 除了两个对称密钥外，无需储存其他信息，因此如果一台 AS 崩溃了，另外一台具有相同 AEK 和 ASK 的服务器可以立即接替它的工作，无需进行复杂的数据恢复和状态恢复以保持一致性.

(3) 可扩展性强. 简单的逻辑使得 AS 服务器可以轻松地扩展为验证服务器集群，消除性能瓶颈.

客户端负责处理用户的请求，执行文件的各种操作. 同时，文件数据加解密、完整性检验也是在客户端完成的. 必要的时候，客户端还需要与验证服务器通信来获取密钥. 而这一切对用户来说都是透明的. 另外 Corslet 客户端只需要存放用户的身份证书，除此之外无需储存其他信息，这也增加了系统的易用性和安全性. 在技术选型上，Corslet 是一个基于 FUSE 实现的用户态文件系统，因此它可以架设在任意一个提供标准 POSIX 接口的文件系统之上，为用户提供安全功能. 这也使得 Corslet 与底层文件系统完全相互独立，适用范围广泛.

3.2　对称密钥层级管理

在安全云存储系统中，密钥管理有两个关键问题：如何减少需要维护的密钥数量以及权限撤销发生时如何处理密钥的更新. 如图 2 所示，在 Corslet 中，密钥分为 3 个层级来组织：文件密钥、安全元数据文件和验证服务器.

文件密钥. 密钥层级的第 1 层是文件密钥. 为了更高效安全地处理大文件，Corslet 以分块加密的形式来加密文件数据. 每一个文件块 FB_i 采用单独的对称密钥 FBK_i 来进行加解密，FBK_i 的计算方法如下：

$$FBK_i = \mathrm{HASH}(FB_i) \| offset_i,$$

其中符号“‖”表示拼接(concatenate)，HASH 表示计算文件块的哈希值(例如使用 SHA-1 算法)，$offset_i$ 是块 i 在文件中的偏移量. 也就是说，文件块的加密密钥，是由它明文的哈希值与它的偏移量拼接而成的，这种密钥的选择方式能够带来以下 3 点好处：(1) 由于要保护文件的完整性，就免不了要计

算文件内容的哈希,而采用明文的哈希作为密钥,就能使这部分信息得到重用,很大程度上节省了密钥存储空间;(2)由于相同内容的文件块会得出相同的哈希值,因此在哈希值后拼接上偏移量来作为密钥,就能够使相同内容的文件块产生不同的密文,增强了保密性;(3)以这种方式来选择密钥,将使得文件块内容发生改变时,密钥也随之发生改变,这对系统的安全性和权限撤销(后面会详细叙述)都有好处.

安全元数据文件. 密钥层级的第 2 层是安全元数据文件. 如图 2 所示,在 smd-file 的访问控制块 ACB 中,有一个锁盒子密钥 LBK. 从图中可以看出,所有的文件块密钥 FBK_i 都用 LBK 进行了加密,并以 Merkle Tree[7] 的形式存放在安全元数据文件(smd-file)中. 只有拿到了 LBK 的用户,才可以解密得到文件块密钥,进而对数据文件进行解密,得到明文内容.

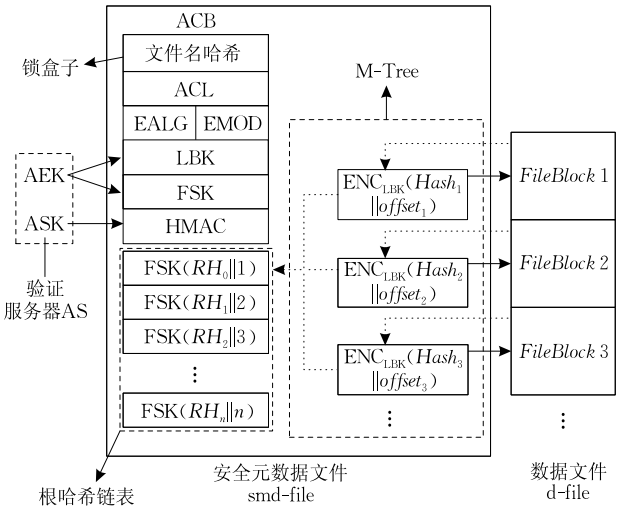

图 2 密钥层级管理

ACB 中还有一个文件签名密钥 FSK,只有获取了 FSK 的用户才具有对文件进行合法修改的能力,即具有写操作权限,这部分内容在 3.3 节将进行详细叙述.

验证服务器. 密钥层级的第 3 层是验证服务器. 如图 2 所示,验证服务器 AS 中存放了两个对称密钥:AEK 和 ASK. 前者是验证服务器加密密钥(AS Encryption Key),后者是验证服务器签名密钥(AS Signature Key). 这两个密钥只有 AS 自己知道,是对其他的任何实体保密的. 从图 2 可以看出,在 smd-file 的 ACB 中,锁盒子密钥(LBK)和文件签名密钥(FSK)都被 AS 用 AEK 加密过. 也就是说,用户想要获得 LBK 或 FSK,只能通过与 AS 进行通信来获取. 签名密钥 ASK 则用来计算访问控制块 ACB 的 HMAC 值. 利用 smd-file 中的 HMAC 值,AS 就能够判断 ACB 的完整性是否遭到破坏,而其他实体由于无法得到 ASK,所以不具备合法修改 ACB 内容的能力.

通过这样的 3 层密钥管理结构,数目众多的密钥可以被高效地组织起来,在保证数据私密性与完整性的同时,提高了密钥管理的效率,并且对用户的身份认证、访问授权以及权限撤销都是很有好处的,这将在 3.5 节详细叙述.

3.3 完整性保护

Corslet 通过为文件的每一个块计算明文哈希值来保证它的完整性(如前文所叙述的,该哈希值也是其加解密密钥 FBK 的一部分),进一步地把这些哈希值组成一棵或多棵 Merkle Tree.

传统的 Merkle Tree 的叶子结点存放文件块哈希,非叶子结点用来保证儿子的完整性,存放的是所有儿子拼接后的哈希值. 这样的方式对于不变文件来说比较适合,但是如果文件被频繁修改,那么 Merkle Tree 的维护和存储就比较麻烦了. 在 Corslet 中,使用改进的 Merkle Tree,非叶子结点也参与存放文件块哈希,其结构如图 3 所示.

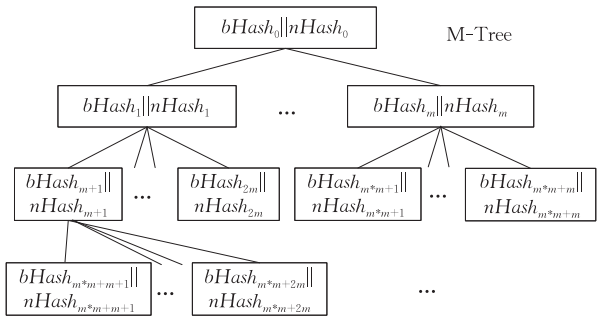

图 3 Corslet 中的 M-Tree

其中,图 3 中 $bHash_i$ 的内容为

HASH(FBi) 20 Byte	offset 8 Byte	空白 2 Byte	块存在标记 1 Byte	结点存在标记 1 Byte

$nHash_i$ 的计算方法:

$$nHash_i = \mathrm{HASH}(bHash_{m*i+1} \| nHash_{m*i+1} \| \cdots \| bHash_{m*i+m} \| nHash_{m*i+m}).$$

Corslet 中的 Merkle Tree(以下简称为 M-Tree)是一棵 m 叉树,除叶子结点外,每个结点有 m 个子结点. 结点 i 中的 $bHash_i$ 其实就是文件块 i 的密钥 FBK_i. $nHash_i$ 是结点 i 的所有子节点内容拼接后,再计算出来的哈希值. 另外还使用了"块存在标记"来标识文件块 i 是否存在(文件有可能存在空洞),以及用"结点存在标记"来标识 M-Tree 中结点 i 是

否存在(如果 $bHash_i$ 存在或者 $nHash_i$ 存在，就认为结点 i 存在)，这两个标记可以帮助识别文件空洞，以及提高完整性校验的效率(不存在的 M-Tree 结点可以不参与计算).

最后对 M-Tree 的根用文件签名密钥 FSK 加密并存放在 smd-file 的根哈希链表中，为了提供并发写的可能，用多棵 M-Tree 和一个将这些 M-Tree 的根哈希串起来的链表来保证一个文件的完整性.

在 Corslet 中，用 $bHash_i$ 保证了文件块 i 的完整性，用 $nHash_i$ 保证了以结点 i 为根的子树的完整性，所以 M-Tree 的根就保证了整棵 M-Tree 的完整性. 由于只有具有文件写权限的合法用户才能获取到文件签名密钥 FSK(这将在 3.5 节中详细叙述)，因此在 smd-file 中存放的根哈希保证了非法用户一旦篡改了文件内容就能被发现. 总的来说，用 FSK 加密的根哈希来保证了 M-Tree 的完整性，而 M-Tree 中存放了所有文件块的明文哈希，进而保护了整个文件数据的完整性.

采用 M-Tree 来保护文件完整性的好处是显而易见的. 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的 $bHash$，以及这些块通往根结点的路径上所经过结点的 $nHash$. 最后对更新后的根哈希用 FSK 重新加密存放在 smd-file 中. 这样的修改复杂度是 log 级的. 如果不使用 M-Tree，而是把所有的 FBK 拼接在一起计算出一个哈希值，进而保证所有 FBK 的完整性，那么哪怕只是修改一个块，也需要重新对所有 FBK 进行拼接并计算哈希值，这样的开销对大文件来说是难以接受的.

另外，采用文件块明文的哈希值而非密文的哈希值作为完整性校验的方式，一来可以重用明文哈希值作为密钥，节省了空间；二来可以确保用户获得的信息的的确确是他想要的(因为仅仅保证密文的完整性是不够的，如果完整性保护机制不够健全的话，密钥有可能被篡改，解密出来的明文就不对了).

在 Corslet 中，对文件的每次读写访问都会先检查访问内容的完整性. 先检查 M-Tree 根哈希的完整性，然后检查访问所涉及到的 M-Tree 结点以及这些结点通往根结点的路径上所经过的结点的完整性，这样可以确保这些结点的 $bHash$，也就文件块密钥 FBK 的完整性. 检查 M-Tree 结点完整性的方法是根据它的子结点重新计算它的 $nHash$ 值，并与 smd-file 存放的 $nHash$ 值进行比较，最后检查每一个文件块的完整性(对解密出来的明文计算哈希值，与 FBK 中的明文哈希进行比较).

3.4　懒惰权限撤销

在用户数较多的云存储系统中，权限撤销经常发生[4]. 大多数云存储系统主要依靠存储服务器来管理文件的访问权限，权限撤销的开销很小，但是这要求用户完全信任存储服务器，而这对用户的数据是不安全的. 相对的，在加密存储系统里，权限撤销所带来的性能开销要大得多，因为要避免权限被撤销的用户继续对文件进行访问，需要重新生成文件的相关密钥，重新加密，并将新的密钥分发给依然具有访问权限的用户. 其中对文件的重新加密将会较严重地影响系统性能，并导致在此过程中文件无法被访问.

为了减少权限撤销所带来的额外开销，Corslet 采用了懒惰权限撤销技术[6]. 对于每一个文件块而言，在权限撤销后，只有当它的内容发生改变时，它才被重新加密，这在很大程度上降低了权限撤销对性能的影响. 而对于 smd-file 而言则不然，一旦权限撤销发生，就需要为该文件重新生成锁盒子密钥 LBK 和文件签名密钥 FSK，然后用新的 LBK 对所有的文件块密钥 FBK_i 重新加密，用新的 FSK 对所有的根哈希 RH_i 重新加密. 由于 Corslet 只对所有的 FBK 和 RH 进行重新加密，并没有对文件块进行重新加密，相比之下，数据量减少了非常多，性能开销也大大降低，由此也可体现出密钥层次管理机制的优越性. 另外，当文件块的内容被修改后，它的哈希值也会随之改变，于是文件块密钥 FBK 也就自动的发生了改变，因此 Corslet 无需像已有的懒惰权限撤销机制那样，用一套复杂的方法和额外的空间来记录文件的历史密钥和历史状态等信息[4,8]，这又进一步地节省了时间和空间开销. 另外，已有的一些权限撤销机制需要对整个文件重新加密，而 Corslet 是精确到文件块级别的，这也在一定程度上提高了性能.

3.5　访问协议

Corslet 的文件访问协议与已有加密文件系统中的访问协议比起来，具有更安全、更简单、更快等特点. 在这套协议里，客户端与认证服务器间的通信均采用了 SSL 加密，能有效解决网络不可信的问题. 另外，验证服务器 AS 只需要维护两个对称密钥，而客户端只需要维护自己的身份证书，不需要维护任何密钥，这不仅简单高效，而且还减少了密钥泄露的风险，更为安全. Corslet 能够做到这一点，与它独特的文件访问协议是密不可分的，接下来将详细叙述这套协议.

身份认证. 在 Corslet 中采用 X509 标准来实现身份认证. 每个用户都需要向 CA 申请一个证书,用以唯一标识自己的身份. 当客户端与验证服务器 AS 通信时,会尝试与 AS 建立一个 SSL 连接①. 连接建立的时候,客户端会将用户的身份证书发送给 AS,该证书通过 AS 的验证后,AS 也会将自己的证书发送给客户端. 在双方都确认对方的身份信息无误后,SSL 连接正式建立. 也就是说,在验证服务器 AS 端,每一个连接都是与用户证书绑定的,即所有的通信都是身份可识别的通信.

创建文件. 创建文件的流程如下:

(1) 客户端初始化一个创建文件请求,其中包括文件名、加密算法、加密模式、访问控制列表等信息,然后将此请求发送给验证服务器 AS.

(2) AS 收到请求后为文件生成锁盒子密钥 LBK 和文件签名密钥 FSK,并对这两个密钥用验证服务器加密密钥 AEK 进行加密,创建出 ACB(参考图 2),接着用验证服务器签名密钥 ASK 计算出 ACB 的 HMAC 值. 最后初始化根哈希链表,并将生成好的 ACB 返回给客户端.

(3) 客户端根据 AS 返回的 ACB 在存储服务器上创建两个文件:数据文件 d-file 和安全元数据文件 smd-file.

读文件. 读文件的流程如下:

(1) 客户端从存储服务器中找到安全元数据文件 smd-file,从中读出 ACB 的内容,并将读请求与 ACB 发送给验证服务器 AS.

(2) AS 首先检查 ACB 的完整性,然后通过 ACL 访问控制列表判断用户是否有读权限,接着用 AEK 将 LBK 和 FSK 解密出来,并用 FSK 解密根哈希链表(用于验证 M-Tree 的完整性),最后把 LBK 和根哈希链表返回给客户端.

(3) 客户端用 LBK 把需要访问到的文件块的块密钥 FBK 解密出来,验证 M-Tree 及其根哈希的完整性. 接着从 d-file 中读出相关的文件块并用对应的 FBK 解密得到明文,并对明文计算哈希来检验文件块的完整性.

写文件. 写文件的流程如下:

(1) 客户端从存储服务器中找到安全元数据文件 smd-file,从中读出 ACB 的内容,并将写请求与 ACB 发送给验证服务器 AS.

(2) AS 首先检查 ACB 的完整性,然后通过 ACL 访问控制列表判断用户是否有写权限,接着用 AEK 将 LBK 和 FSK 解密出来,并用 FSK 解密根哈希链表,最后把 LBK、FSK 和根哈希链表返回给客户端.

(3) 客户端用 LBK 把需要访问到的文件块的块密钥 FBK 解密出来,验证 M-Tree 及其根哈希的完整性. 然后对要写入的数据以文件块为粒度计算哈希和新的 FBK,并用新的 FBK 对数据进行加密,写入 d-file 中. 同时,还需要更新 M-Tree,重新计算根哈希并用 FSK 加密,写入 smd-file 中.

共享文件. 共享文件的流程如下:

(1) 文件拥有者从存储服务器中找到 smd-file,读出 ACB. 把文件共享请求和 ACB 发送给 AS. 文件共享请求中包括要在访问控制列表 ACL 中加入哪些用户、每个用户具有哪些权限等信息.

(2) AS 首先检查 ACB 的完整性,并检查该用户是否为文件拥有者. 接着将客户端请求中的访问控制项插入到 ACL 中,并用验证服务器签名密钥 ASK 重新计算 ACB 的 HMAC 值. 最后将更新后的 ACB 返回给客户端.

(3) 客户端将新的 ACB 写入到 smd-file 中.

权限撤销. 权限撤销的流程如下:

(1) 客户端从存储服务器中找到 smd-file,读出 ACB. 把权限撤销请求和 ACB 发送给 AS. 权限撤销请求中包括要对哪些用户进行权限撤销、每个用户被权限撤销后所应该拥有的权限(例如从读写权限降级为只读权限)等信息.

(2) AS 首先检查 ACB 的完整性,并检查该用户是否为文件拥有者. 接着根据客户端的请求更新访问控制列表 ACL,然后重新生成锁盒子密钥 LBK 和文件签名密钥 FSK,并用验证服务器加密密钥 AEK 对新的 LBK 和 FSK 进行加密,写入 ACB 中. 最后用验证服务器签名密钥 ASK 重新计算 ACB 的 HMAC 值,将更新后的 ACB、旧的 LBK 和 FSK、新的 LBK 和 FSK 返回给客户端.

(3) 客户端用旧的 LBK 将所有的文件块密钥 FBK 解密,并用新的 LBK 对它们进行加密,写入 smd-file 中. 再用旧的 FSK 对所有的根哈希进行解密,并用新的 FSK 对根哈希进行加密,写入 smd-file 中. 最后将更新后的 ACB 写入 smd-file 中.

3.6 正确性保证与性能调优

本研究在 Corslet 中实现了一套独立的锁机制

① SSL/TLS. http://tools.ietf.org/html/rfc5246

(包括文件读写锁和线程互斥锁)来实现读写互斥，保证文件数据的一致性. Corslet 支持多线程并发读同一个文件.

为了提高 Corslet 的性能，本研究使用了缓存机制来降低加解密、完整性校验的开销. 例如将访问到的 M-Tree 的前三层结点的明文缓存在内存中，直到需要的时候(例如文件关闭时)才把它们重新加密写回 smd-file，这样可以减少在完整性校验过程中对 M-Tree 前三层结点多次 I/O 访问和加解密的开销. 相应地，可以为被缓存的 M-Tree 结点设立一个“是否已检查完整性”标记，这样可以避免对缓存的 M-Tree 结点重复校验完整性.

如果用户对某一段数据进行重复读写，每次读操作都需要从加密的数据文件中读出对应内容，进行完整性校验，然后从安全元数据中读入相关的文件数据加密密钥(密文形式)，然后用文件根密钥 FBK 对这些密钥进行解压，再利用这些密钥解密数据文件得到明文数据，写操作也是类似的. 为了提高性能，本研究在 Corslet 中实现了一套独立的缓存系统，对最近访问的文件块明文进行缓存，这使得以上的步骤都能被省略，减少了不必要的 I/O 操作、完整性校验以及加解密开销.

在具体实现中，本研究使用了 Radix Tree[9] 来组织文件块缓存，这可以高效地对缓存块进行查找、插入和删除，同时使用了 LRU 链表来管理缓存池，提高缓存的命中率. 另外还使用了完善的锁机制来保证缓存系统的正确性.

4　功能及性能测试

文中对 Corslet 的功能和性能进行了一系列的测试，包括验证 Corslet 在不可信的网络及存储环境下所能提供的安全功能，测出加解密、完整性检验、文件共享与权限撤销等部分的开销，以及使用 Bonnie++[①] 和 IOzone[②] 来测试 Corslet 的整体性能.

4.1　功能测试

文中用 3 台服务器对 Corslet 进行了功能测试. 其中一台服务器作为 Corslet 的验证服务器 AS，同时也是 NFSv4 的服务器端；另外两台服务器作为 NFSv4 的客户端以及 Corslet 的客户端，分别以用户 A 和用户 B 的身份将 Corslet 挂载在 NFSv4 之上. 测试内容如表 2 所示.

表 2　Corslet 功能测试

测试内容	测试结果
数据私密性保护：绕过 Corslet 查看用户文件内容	文件内容为加密后的乱码，无法得知文件 fa 明文
数据完整性保护：绕过 Corslet 篡改数据文件或者元数据文件	Corslet 报出文件完整性遭到破坏的警告
权限管理：测试文件的访问者在其被授予读/写权限之前，被授予读/写权限后，以及被撤销权限后对文件进行读/写操作的结果	被授予读/写权限之前以及被撤销权限后对文件的读/写操作均失败；被授予读/写权限后对文件的读/写操作成功

4.2　性能测试环境与参数选择

性能测试的硬件环境是两台配置相同服务器，型号为 Sun SunFire™ V20z，1.8GHz 的 AMD 双核 CPU，4 GB 内存，两台服务器间以千兆局域网连接，一台做验证服务器，一台做客户端. 软件环境是 Debian Linux 2.6.30 内核，fuse 2.8.1，openssl 0.9.8k.

在安全机制的选择上，使用了 SHA-1 函数来计算哈希，用基于 SHA-1 的 MAC 算法 HMAC 来计算 MAC，AES-256 系列函数为默认的加解密函数，cfb 为默认的加密模式(用户也可以通过配置文件来选择加密算法和加密模式)，用 X509 系列函数来实现身份验证.

设置 Corslet 文件块大小为 64 KB，这是由于测试场景以大文件应用为主. 如果需要将 Corslet 应用在小文件的云存储环境中，也可以很方便地对文件块大小进行调整.

在 M-Tree 的结构上，选择了 M-Tree 的叉数为 64，并规定一棵 M-Tree 高度不超过 4 层. 这样的选择是为了在 M-Tree 的完整性校验时减少磁盘 I/O. M-Tree 的层数越少需要进行的 I/O 操作就越少. 实际上在具体实现中采用了缓存机制来保证每次 M-Tree 完整性校验最多只会出现 1 次 I/O 操作. 另外单个文件最大支持 512 棵 M-Tree，所以 Corslet 最大可以支持大小为 8TB 的文件.

4.3　加解密开销

先创建一个文件，随后以读写模式打开此文件，以每次 64 KB 的粒度写入 500 MB 的内容，再将这 500 MB 的内容读取出来，最后关闭文件. 表 3 列出了测试过程中各种文件操作中的加解密开销.

从表 3 可以看出，加解密的开销绝大部分都在文件数据的加解密中，另外写操作中的“计算 M-Tree 结点值”与读操作中的“验证文件块哈希”也消耗了

① Bonnie++. http://www.coker.com.au/bonnie++/
② IOzone. http://www.iozone.org/

表 3 文件操作的加解密开销

文件操作	加解密操作	总开销/ms	执行者	操作频率
create	生成 FSK,LBK	0.013	AS	文件级
	加密 FSK,LBK	0.009	AS	文件级
	计算 ACB HMAC	0.048	AS	文件级
Open	验证 ACB HMAC	0.049	AS	文件级
	解密 FSK, LBK	0.013	AS	文件级
	解密根哈希链表	0.013	AS	文件级
Close	加密 M-Tree	2.453	写者	文件级
	加密根哈希链表	0.005	写者	文件级
Write	验证 M-Tree 结点值	0.314	写者	文件块级
	验证 M-Tree 根哈希	0.661	写者	M-Tree 级
	计算 M-Tree 结点值	2556.483	写者	文件块级
	计算 M-Tree 根哈希	16.595	写者	M-Tree 级
	加密文件块	9996.914	写者	文件块级
Read	验证 M-Tree 结点值	1.926	读者	文件块级
	验证 M-Tree 根哈希	2.301	读者	M-Tree 级
	解密文件块	8693.968	读者	文件块级
	验证文件块哈希	2053.295	读者	文件块级

部分时间,这是两部分属于完整性校验的开销.以上开销跟读写内容的大小是成正比的.另外还可以看出验证服务器 AS 所参与的操作由于逻辑简单,耗时非常短.

为了测试文件共享与权限撤销方面的开销,进行了如下测试:文件 A 的拥有者先赋予 500 个不同的用户对文件 A 的只读权限,接着将这 500 个用户的权限升级为读写权限,最后将他们的权限全部撤销.记录了每一步的时间开销(从用户开始执行操作到操作完成的用时),如表 4 所示.

表 4 Corslet 权限操作时间开销

操作描述	时间开销/ms
文件只读共享	3.553
权限升级	3.578
权限撤销	5.240

从表 4 可以看出,由于文件的共享或者权限升级只需简单地修改文件的访问控制列表 ACL 并重新计算 ACB 的 HMAC 即可,因此速度很快.而权限撤销需要重新生成 LBK 和 FSK,并对已有的文件块密钥以及根哈希链表重新加密,因此相对来说耗时较长,但由于没有立刻对文件块内容进行重新加密,速度还是非常快的.

4.4 文件读写测试

4.4.1 大文件读写测试

在与 4.2 节相同的测试环境下,使用 Bonnie++ 1.03e 对 Ext3 和 Corslet 进行了测试.先在客户端的本地文件系统 Ext3 上运行 Bonnie++测试性能,然后将 Corslet 架在 Ext3 上,再运行 Bonnie++测试 Corslet 的性能.测试结果如图 4 所示.

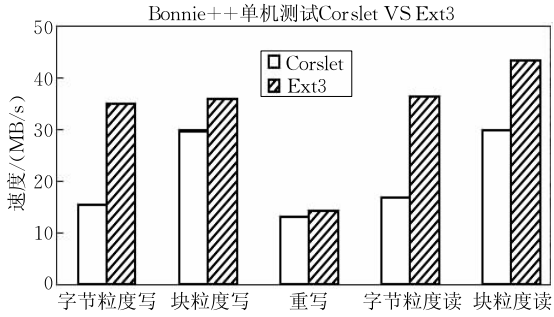

图 4 Bonnie++单机测试 Corslet vs Ext3

由图 4 可以看出,Corslet 与 Ext3 相比,文件块粒度的读写操作性能分别下降了 31.3%和 17.4%,主要原因是文件内容加解密引入的开销.而字节粒度的读写性能分别下降了 53.6%和 55.6%.字节粒度的读写性能下降更严重的原因是每次的读写操作,Corslet 都需要检验完整性,包括访问涉及到的文件块的完整性,以及 M-Tree 的完整性.又由于在实现中 Corslet 选择了 64KB 作为文件块大小,哪怕是访问文件块中的一个字节,也需要对整个文件块的内容计算哈希来检验完整性,因此字节粒度的读写会引入更多的完整性校验开销.

接着在 NFSv4 环境下对 Corslet 进行了测试,仍然采用了相同的测试环境,使用 Bonnie++来测试.两台服务器,其中一台做验证服务器 AS,同时也作为 NFSv4 服务器提供存储服务.另一台服务器既是 NFSv4 客户端,也是 Corslet 客户端(将 Corslet 挂载在 NFSv4 之上).先后在 Corslet 挂载点和 NFSv4 挂载点上运行 Bonnie++进行测试,测试结果如图 5 所示.

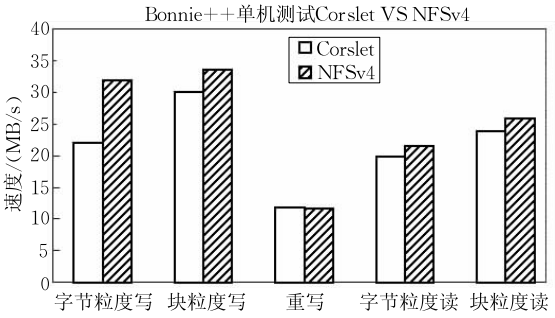

图 5 Bonnie++单机测试 Corslet vs NFSv4

对比图 4 和图 5 可以发现,NFSv4 和 Ext3 相比,写性能略降了 10%左右,但读性能下降了 40%以上,这主要是由 NFSv4 引入的网络开销和协议开销导致的.而 Corslet 的性能变化很小,只有略微下降,这是因为 Corslet 引入的文件内容加解密和完整性校验等开销,是性能下降的主要原因.由于这些

操作是计算密集型的，CPU 的性能会成为瓶颈，因此 NFSv4 虽然削弱了底层存储的 I/O 能力，但并没有削弱到使其成为瓶颈的程度，也就是说 CPU 的性能依然是瓶颈. 所以 Corslet 的性能只有略微的下降. 由此推测，在云存储环境下，随着用户的增多，也就意味着客户端的增多，计算能力的增强，计算部分将不再成为瓶颈，Corslet 的性能开销会越来越不明显. 为了验证该想法，对 Corslet 进行了集群测试.

在集群测试中，硬件环境采用了 5 台配置相同的服务器，型号是 Dell PowerEdge™ M605，服务器拥有两个 2.8 GHz 的 AMD 四核 CPU，8 GB 内存. 软件环境与配置和之前的测试相同. 测试文件的大小选择为 16 GB. 这 5 台服务器中，有 1 台作为 NFSv4 的服务器，将本地磁盘的一个 Ext3 分区导出，对外提供存储服务，1 台作为验证服务器，其余 3 台作为客户端，均挂载了 NFSv4，并在其上挂载 Corslet.

这次采用了 IOzone 3.347 进行测试，因为 IOzone 具有集群测试功能. 先让 3 台客户端同时在本机的 NFSv4 挂载点上进行 IOzone 集群测试，然后类似地改为在 Corslet 挂载点上，得到的结果如图 6 所示.

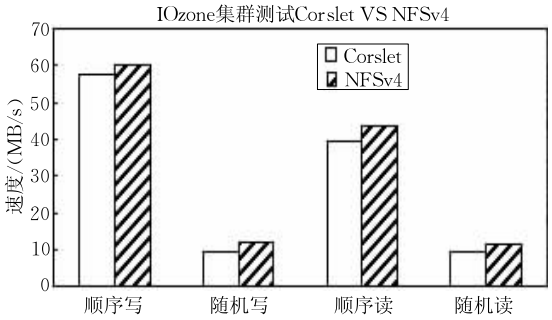

图 6　IOzone 集群测试 Corslet vs NFSv4

由图 6 可以看出，Corslet 架在 NFSv4 之上的聚合访问速度达到了 NFSv4 的 95% 以上，且达到了单块磁盘的性能极限. 由此可见当客户端增多时，存储服务成为了瓶颈，Corslet 引入的计算开销就很不明显了. 实验结果表明，在云存储环境中，Corslet 将能得到很好的应用.

4.4.2　大批量小文件读写测试

针对实际环境中所存在对大规模小文件操作需求，分别测试了 Corslet 和 NFS 在网络环境下对大规模小文件操作的性能对比. 该测试选用了 3 台服务器分别作为云存储的文件服务器、认证服务器和客户端，其硬件环境和软件环境与配置和之前所测试 IOzone 的服务器相同. 在对 NFS 和 Corslet 测试的过程中，客户端分别在相应的挂载点下对 1000 个小文件进行创建、写操作和读操作，其中每个文件的大小设置为 512 KB. 测试的结果如图 7 所示.

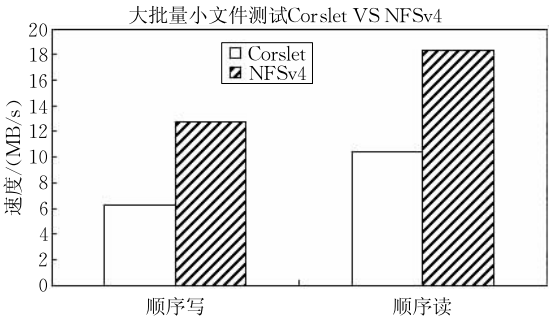

图 7　大批量小文件测试 Corslet vs NFSv4

由图 7 可以看出，Corslet 系统和 NFSv4 相比，小文件的读写操作性能分别下降了 50.7% 和 43.1%，这是因为 Corslet 在每次访问数据文件的同时，还需要访问安全元数据文件. 在处理小文件时，处理安全元数据所占的时间比例比大文件所占的比例要大，从而导致 Corslet 的小文件处理性能不如大文件.

在实际访问中，为了优化性能，Corslet 提供了缓存机制，能够在很大程度上改善小文件读写的性能. 例如，当 Corslet 系统再次访问某个文件时，由于其内容已在内存中，因此访问速度会大大的提升.

5　相关工作

本文工作与相关研究工作的对比如表 5. 其中：CFS[10] 是最早的加密文件系统之一. CFS 是一个虚拟的加密文件系统，在把数据写到磁盘之前，对文件名和文件数据进行加密. CFS 用一个密钥来加解密整个目录中的文件. 访问控制则是通过把密钥共享给其他用户，这决定了 CFS 只允许在一台机器上进行粗粒度的共享，并且不区分读权限和读写权限.

Cryptfs[11]、ECFS①、Cepheus[6] 和 TCFS[12] 都是 CFS 的著名变种. Cryptfs 赋予文件组对称密钥，允许组文件共享. Cepheus 则引入了锁盒子来实现用户组间的共享管理，依赖一个可信的密钥服务器来存储用户组成员信息来进行身份认证，同时依赖存储服务器实现访问控制，它是第一个提出懒惰权

① Bindel D, Chew M, Wells C. Extended cryptographic file system. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.4339&rep=rep1&type=pdf

表 5 加密文件系统相关工作对比

	加密粒度	密钥生成	加密机制	完整性保护	访问控制与密钥分发	懒惰撤销	文件共享的扩展性	性能
FARSITE[14]	文件块	明文哈希	公私钥	Merkle Tree	为所有具有文件访问权限的用户加密文件密钥,存放在文件末尾	无	一般	比 NTFS 慢接近 3 倍
RTP-NFSv4[15]	文件块	随机	公私钥	无	文件密钥用用户主密钥加密后存在扩展属性中	无	一般	比 RPCSEC_GSS 的 NFSv4 快 10%
CRUST[8]	文件块	随机	对称密钥	Merkle Tree	为所有具有文件访问权限的用户加密文件密钥,存放在文件末尾	有	较差	比 NFS 慢 10%以内
Tahoe[16]	文件	特殊计算	公私钥	Merkle Tree	将文件的 capabilities 分发给合法用户使之能算出文件密钥	无	较差	未知
SiRiUS[17]	文件	特殊计算	公私钥	整个文件哈希	为所有具有文件访问权限的用户加密文件密钥,存放在文件末尾	无	较差	比 NFS 慢 80%以上
Plutus[4]	文件块	随机	公私钥	Merkle Tree	直接联系文件所有者获取相关密钥	有	一般	没有与底层文件系统比较
Corslet	文件块	明文哈希	对称密钥	Merkle Tree	身份验证+ACL,由管理结点控制	有	较好	比 NFS集群慢 5%以内

限撤销的安全文件系统. ECFS 和 Cepheus 都实现了完整性保护. TCFS 为每个用户颁发一个主密钥来保护该用户的文件的密钥. 这 4 个文件系统都不能区分读共享和读写共享.

NCryptfs[13]是一个在内核态实现的安全文件系统,可以支持一台机器上的多用户文件共享,但是无法支持大规模的文件共享.

Tahoe[14]是一个分布式安全文件系统,包括访问控制、加密、完整性检查等功能,以及采用了纠删码来实现容错. 它被部署在一个商业的备份服务中. Round-Trip Privacy with NFSv4[15]是在 NFSv4 上的一个改进,修改了 NFSv4 中的 RPCSEC_GSS 协议,使得文件服务器上的文件以密文方式存放. 但 RTP-NFSv4 并没有对文件进行完整性保护,密钥机制过于简单. Farsite[16]是一个安全文件系统,提供一个集中式的文件服务器功能,但实际上是由多台分布式的不可信计算机组成. Farsite 通过多副本机制提供文件可用性和可靠性,通过加密来保证文件内容保密性,通过一个能防止拜占庭错误的协议来保证文件和目录数据的完整性. Tahoe、RTP-NFSv4、Farsite 三者的访问权限控制机制对大量用户的频繁文件共享与权限撤销支持得不好.

CryptosFS[17]和 SNAD[18]都使用了公私钥加密体系来实现访问控制,并且通过文件服务器来验证用户的访问权限,因此需要完全信任文件服务器. CryptosFS 的用户需要使用非对称密钥从安全元数据文件中解密出相应的对称密钥,再用这些对称密钥去解密数据文件,因此它的密钥管理机制是带外的.

SiRiUS[19]是一个栈式文件系统,为已有的文件系统提供安全机制. SiRiUS 使用了大量的非对称密钥进行权限控制,还需要一个专门的公私钥服务器. 在 SiRiUS 中,文件是整个被加解密的,完整性校验也是对整个文件计算哈希的,权限撤销时马上重新加密,性能开销较大. Plutus[4]同样使用了公私钥加密体系,提供了组共享和懒惰撤销、随机访问、文件名加密等功能. Plutus 采用了用户间共享文件密钥的密钥管理机制,当其他用户想访问文件时,需要向文件拥有者索要密钥,这种机制要求文件拥有者实时在线. CRUST[8]和 SiRiUS 一样,也是一个栈式文件系统,但它没有使用公私钥加密体系,所有的加解密都使用了对称密钥. CRUST 依赖一些公共的数据结构来实现分布式密钥管理、密钥回滚和权限撤销等等,这会导致随着用户数增多,需要维护的信息将以平方级增长,不适合在用户数较多的环境下使用. CRUST 和 Plutus 都使用了 Merkle Tree[7]来保证文件完整性.

与以上工作相比,Corslet 具有较明显的优势. Corslet 是一个具有栈式加密文件系统的安全云存储系统,可以架在已有文件系统之上. Corslet 有一套自己的权限控制机制,不依赖于云存储服务器. 除 X509 身份认证外,Corslet 中涉及到的所有加解密均使用了对称密钥,与公私钥加密体系比起来开销更小. Corslet 对文件分块加解密,使用了层级密钥管理机制和带内的密钥分发机制,用户无需维护任何密钥信息,使得密钥管理更加安全高效. Corslet 采用了比已有实现更高效的懒惰权限撤销机制,无需存储密钥历史信息,节省了时间和空间上的开销.

Corslet采用了改进后的 Merkle Hash Tree 和缓存机制，使完整性检查更加快速，并为并发写提供可能.

6 总　结

文中介绍了一种新的安全云存储系统架构，确保用户在不可信的网络环境和云存储环境下得到数据的安全.通过引入可信的验证服务器，这套系统架构消除了对云存储服务器的依赖，非常适合越来越流行的云存储应用场景，同时，验证服务器因为逻辑简单而具备较高的可扩展性.

本文提出的安全云存储系统 Corslet，为已有的云存储环境下的用户提供安全保护，包括私密性保护、完整性保护以及文件访问控制等.同时，Corslet 较好地支持了随机访问与并发访问.对 Corslet 的性能测试结果表明 Corslet 挂载在 NFSv4 集群之上带来的性能下降不足 5%，由此可见，Corslet 在提供强大的安全性保护的同时，所带来的额外性能开销是可以接受的.

参　考　文　献

[1] Sandberg R, Goldberg D, Kleiman S, et al. Design and implementation of the SUN network filesystem//Proceedings of the Summer USENIX Conference. Portland, USA, 1985: 119-130

[2] Weil S, Brandt S, Miller E, et al. Ceph: A scalable, high-performance distributed file system//Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Seattle, USA, 2006: 307-320

[3] Hasan R, Myagmar S, Lee A J, Yurcik W. Toward a threat model for storage systems//Proceedings of the 2005 ACM Workshop on Storage Security and Survivability. Fairfax, USA, 2005: 94-102

[4] Kallahalla M, Riedel E, Swaminathan R, et al. Plutus: Scalable secure file sharing on untrusted storage//Proceedings of the 2nd USENIX File and Storage Technologies. San Francisco, USA, 2003: 29-42

[5] Riedel E, Kallahalla M, Swaminathan R. A framework for evaluating storage system security//Proceedings of the 1st USENIX File and Storage Technologies. Monterey, USA, 2002: 15-30

[6] Fu K. Group Sharing and Random Access in Cryptographic Storage File Systems[M. S. dissertation]. Massachusetts Institute of Technology, Boston, USA, 1999

[7] Merkle R. A digital signature based on a conventional encryption function//Proceedings of the Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. Santa Barbara, USA, 1987: 369-378

[8] Geron E, Wool A. CRUST: Cryptographic remote untrusted storage without public keys//Poceedings of the 4th International IEEE Security in Storage Workshop. San Diego, USA, 2007: 3-14

[9] Bovet A, Cesati M. Understanding the Linux Kernel. 3rd Edition. Translated by Chen Li-Jun, Zhang Qiong-Sheng, Zhang Hong-Wei. Beijing: China Electric Power Press, 2007 (in Chinese)
(Bovet A, Cesati M. 深入理解 Linux 内核. 第 3 版. 陈莉君，张琼声，张宏伟，译. 北京：中国电力出版社，2007)

[10] Blaze M. A cryptographic file system for Unix//Proceedings of the ACM Conference on Computer and Communications Security. Scottsdale, AZ, USA, 1993: 9-16

[11] Zadok E, Badulescu I, Shender A. Cryptfs: A stackable vnode level encryption file system. Columbia University, New York: Technical Report CUCS-021-98, 1998

[12] Cattaneo G, Catuogno L, Sorbo A D, Persiano P. The design and implementation of a transparent cryptographic file system for Unix//Proceedings of the USENIX Annual Technical Conference. Boston, USA, 2001: 199-212

[13] Wright C, Martino M, Zadok E. NCryptfs: A secure and convenient cryptographic file system//Proceedings of the USENIX Annual Technical Conference. San Antonio, USA, 2003: 197-210

[14] Wilcox-O'Hearn Z, Warner B. Tahoe: The least-authority filesystem//Proceedings of the 4th ACM International Workshop on Storage Security and Survivability. Alexandria, USA, 2008: 21-26

[15] Traeger A, Thangavelu K, Zadok E. Round-trip privacy with NFSv4//Proceedings of the 2007 ACM Workshop on Storage Security and Survivability. Alexandria, USA, 2007: 1-6

[16] Adya A, Bolosky W, Castro M, Cermak G, et al. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment//Proceedings of the 5th Symposium on Operating Systems Design and Implementation. San Francisco, USA, 2002: 1-14

[17] O'Shanahan P. CryptosFS: Fast cryptographic secure NFS [M. S. dissertation]. The University of Dublin, Ireland, 2000

[18] Miller E, Freeman W, Long D, Reed B. Strong security for network-attached storage//Proceedings of the 1st USENIX File and Storage Technologies. Monterey, USA, 2002: 1-13

[19] Goh E, Shacham H, Modadugu N, Boneh D. SiRiUS: Securing remote untrusted storage//Proceedings of the 10th Network and Distributed Systems Security Symposium. San Diego, USA, 2003: 131-145

XUE Mao, born in 1986, M. S. candidate. His research interest is in the area of secure cloud storage system.

XUE Wei, born in 1974, Ph. D., associate professor. His research interests include parallel/disturbed computing, network storage system.

SHU Ji-Wu, born in 1968, professor, Ph. D. supervisor. His research interests include cloud storage system, security and reliability of storage, and parallel/disturbed computing.

LIU Yang, born in 1983, M. S. His research interest is in the area of secure cloud storage system.

Background

This research is supported by the National Natural Science Foundation of China (Grant No. 60925006) and the National High Technology Research and Development Program (863 Program) of China (Grant No. 2009AA01A403).

This paper focuses on the problem of data security under shared storage environment. Amount these years, cloud storage is becoming more and more popular, users' data is moved from local disk to shared distributed storage, which makes the users can hardly guarantee the safety of their data. In view of the shared storage provider may use the users' data in improper ways, we can't trust them. So we eagerly need third-party authorities, who is totally irrelevant with the storage providers, to offer secure services.

This paper proposes a secure storage system called Corslet, which can be deployed by the third-party authorities to protect users' data on the shared storage. There are several key problems in secure storage system over shared storage environment, such as how to isolate the secure storage system from the underlying file system, how to authenticate users, how to protect data's confidentiality and integrity, how to manage and distribute huge number of keys, and how to handle authorization and permission revocation, etc.

Corslet is a secure and efficient stackable file system, which solves the problems mentioned above well. Corslet can guarantee the confidentiality and integrity of user data with the support of access randomness and concurrency, while saving users from complexity of key management. For users' convenience, Corslet allows users to choose which files are stored in encrypted form and which are kept in plaintext. By measuring its performance on several tests and benchmarks, we show that Corslet has acceptable overhead while providing strong security. In addition, Corslet has good scalability.

Our research group has been conducting research on several areas related to storage QoS, which include data distribution, storage virtualization and storage management, storage security, etc. Our group has proposed techniques and developed systems to enhance various aspects of QoS for storage, including performance, reliability, flexibility, manageability and security. We have published a number of high-quality papers in these research areas.