武漢理工大學

（申请工学硕士学位论文）

基于 FUSE 的安全网络文件系统的研究与实现

培养单位：信息工程学院

学科专业：信息与通信工程

研 究 生：方敏

指导教师：龙毅宏 教授

2016 年 5 月

分类号_____ 密　　级

UDC _____ 学校代码____10497

武漢理工大學

学　位　论　文

题　目 基于 FUSE 的安全网络文件系统的研究与实现

英　文 Research and Implementation of Secure Network File
题　目 System Based on FUSE

研究生姓名 方敏

指导教师 姓名 龙毅宏 职称 教授 学位 博　士

单位名称 信息工程学院 邮编 430070

申请学位级别 硕　士 学科专业名称 信息与通信工程

论文提交日期 2016 年 5 月 论文答辩日期 2016 年 5 月

学位授予单位 武汉理工大学 学位授予日期

答辩委员会主席 刘泉 评阅人

年　月　日

独 创 性 声 明

本人声明，所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得武汉理工大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：_____日 期：

学位论文使用授权书

本人完全了解武汉理工大学有关保留、使用学位论文的规定，即学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权武汉理工大学可以将本学位论文的全部内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存或汇编本学位论文。同时授权经武汉理工大学认可的国家有关机构或论文数据库使用或收录本学位论文，并向社会公众提供信息服务。

（保密的论文在解密后应遵守此规定）

研究生（签名）： 导师（签名）： 日期

摘　要

随着互联网的发展和信息化时代的来临，文件云存储技术进入了高速发展时期，越来越多的个人用户和企业用户通过云存储系统存储、共享和传递文件数据，云存储系统通常需要使用专用客户端访问、操作文件，这与用户通过操作系统资源管理器访问、操作文件的习惯不相符，同时云存储系统的广泛使用也增加了用户信息泄漏的可能性，特别是云存储系统的运维者对文件的非授权访问，是用户非常担心的问题，也是云存储业务发展的重要障碍。

本文描述的是一种基于 FUSE(Filesystem in Userspace)的安全网络文件系统的实现方案，该系统能够让用户按常规访问和操作文件系统中的文件的方式访问和操作存放在网络文件服务器中的文件，实现授权用户文件共享，并能防止系统的运维者对文件的非授权访问。本系统利用 Dokan 文件系统驱动把文件服务器上的共享文件目录映射为用户本地计算机的一个虚拟文件盘，用户对虚拟文件盘内的文件及文件目录的操作（包括文件创建、读写、删除，文件目录的创建、删除等）将转化为对文件服务器上对应文件及文件目录的操作，贴合了用户的使用习惯。文件在上传时进行加密处理并设置解密控制策略，加密采用混合加密方式，文件数据使用一个随机生成的密钥采用 AES 分块加密，并根据解密控制策略对该密钥使用 IBE(Identity Based Encryption)加密，保障文件信息的安全。

本系统中解密控制策略分为个人解密策略和群组解密策略两种。个人解密策略针对某些特定的个人用户，具有较高的操作权限；群组策略则针对某个用户组、某种用户角色、某个用户部门等群体性用户，可设置某项或全部操作权限，两种策略的配合设置能够更好的满足各种情况下的文件共享。

本论文的创新点在于利用 FUSE 文件系统技术解决了传统文件云存储系统需要通过专用客户端操作网络文件的问题，让用户操作网络文件更加方便。同时，对文件采用对称加密和 IBE 加密相结合并在客户端设置解密控制策略的方案解决了传统云存储客户端普遍存在的文件共享不够安全的问题，特别是防止了系统的运维者对文件的非授权访问。最后进行了测试，测试结果表明本系统在不改变用户操作习惯的前提下实现了文件的安全存储和授权共享。

关键字：文件云存储，FUSE，信息安全，文件共享

Abstract

With the development of the Internet and the advent of the information age, file cloud storage technology has entered a rapid development period. More and more individuals and enterprises are storing, sharing and transfering data files through cloud storage systems. Cloud storage systems usually use their own client to access and manipulate files, which does not match the user’s habbit that accessing and operating the files through the operating system file explorer. Meanwhile, extensive use of cloud storage system also increases the possibility of user information leakage, especially the unauthorized access to the file from the maintainers of cloud storage system. This not only the main issue that user is concerned about, but also a major obstacle of further development of cloud storage service.

This thesis describes an implementation scheme of security network file system based on FUSE(Filesystem in Userspace).The system allows user to access and manipulate files in file system in a conventional method, achieving file sharing among authorized users, and preventing unauthorized access from the maintainers of the network file system.The system make use of Dokan file system driver to virtualize the shared directory in the network file servers to a local virtual disk. Users’ operations on files and directories within the virtual disk (including file creation, write, delete, create a file directory, delete, etc.) will be converted to the operations on the file and file directory within the sever’s sharing file directory, which conforms to the user's usage habit. Files have been encrypted in a hybrid way during the upload process and the decryption control strategy has been set up. To ensure the security of files, the file data is encrypted using AES block encryption algorithm with a randomly generated key, which is encrypted using IBE (Identity Based Encryption) according to the decryption control strategy.

In this system the decryption control strategy includes individual and group decryption strategies. Individual decryption strategies are special for some certain individuals with higher operating authority, Group decryption strategies are for a group of users, some user roles or a user department, which be set some or all operation authorities.Combining two strategies will meet the various file-sharing case

much better.

The innovation of this thesis is using FUSE technology to solve the problem that traditional file cloud storage system requires users to operate network file through a special client, and allow users to operate the network files more conveniently. At the same time, the combining use of symmetric encryption and IBE encryption and setting the decryption control strategy by end users solved the insecurity of traditional cloud storage's file sharing, in particular, prevented the unauthorized access to the documents from the maintainers of cloud storage system. Finally, the test results demonstrate that the system can achieve a secure file storage and sharing authorization without changing user habits.

Keyword: file cloud storage, FUSE, information security, file sharing

目　录

第 1 章　绪论

1.1 课题来源

本论文课题来源国家科技支撑计划课题（课题编号：2014BAH26F03）：安全可信的公众保险服务交互系统与作业支撑环境开发。

1.2 研究背景及意义

人类进入 21 世纪以来，互联网技术、通信技术不断创新和发展，计算机和移动终端迅速普及，改变了人们的日常生活。人们对网络的依赖也日益加深，在互联网上进行信息交流、数据传递与文件共享的现象更加普遍，数字化作业和网络化作业越来越流行，大大提高了人们日常的工作效率和生产效率，但与此同时也产生了大量的网络数据，为了满足人们对于数据存储以及更好的数据共享体验的需求，文件云存储业务应运而生，文件云存储使得用户能够在不同位置、不同用户终端上使用文件，给用户带来了极大方便，日益受到人们的欢迎[1]。

文件云存储系统是一种通过局域网或广域网的方式向注册用户提供网络文件存储和操作服务的系统[2]。只要能够连接网络，用户都能访问存储在云存储服务器上的共享文件，完成阅读、修改等相关操作，随着网络基础设施的逐步完善，网络覆盖的范围越来越广，用户几乎能在任何地方使用云存储系统，让用户不再受到工作环境的制约，大大提高效率。文件云存储系统通常由云存储服务系统和云存储客户端两部分组成，云存储服务系统用于存放文件，云存储客户端则用于用户进行网络文件操作，比如向云存储服务器上传文件以及从云存储服务器下载文件、删除文件等[3]。云存储突破了通过 U 盘，移动硬盘等方式转发文件的时间、距离限制，作为一种新型的存储方式，正在逐渐取代其他传统的存储方式。

文件云存储系统固然给人们带来了极大的便利和好处，但也增加了用户信息泄漏的可能性[4]。一直以来，人们对于信息安全问题都极为重视，对于涉及商业机密、个人私密信息的文件的存储更是慎之又慎。云存储系统的广泛使用，

使得用户存放在云存储服务器的文件可能因数据被他人非法截取而造成信息泄漏，同时目前大多数云存储系统都具有文件共享功能，即由一个用户将需要共享的文件上传到云存储系统，并指定能够访问和操作该文件的用户或用户群组，而文件共享如果没有采用合理的文件共享策略加以规范，导致非共享用户非法窃取了共享文件信息，这将极大的损害用户的个人利益。针对上述两个安全问题，大多数的云存储系统通常会对存放在云存储服务器中的文件采用一定的加密算法进行加密，当用户需要访问或操作文件时再对加密文件进行解密，而在确保文件共享安全方面，云存储系统在服务端通常会采用一种文件共享策略，当用户试图解密和操作加密文件时首先对用户的身份进行判断，如果是共享用户则可以解密共享文件，如果是非共享用户则不能解密共享文件，这在一定程度上防止了共享文件被他人进行非法访问[5,6]。但是这种由云存储系统的服务端对文件进行加密并制定文件共享策略的方案并不能防止云存储系统的的运维者对文件的非授权访问，系统运维者可以通过在服务端修改文件共享策略使得自己能够访问实际未被共享的机密文件，从而可能造成重要数据或机密的外泄，这是广大用户目前非常担心的问题[7]。

云存储业务的迅猛发展，让许许多多的公司参与到云存储系统的开发与研究中来[8]。但是目前的云存储系统通常是通过专门的的云存储客户端或者网页客户端向用户提供文件存储和共享功能，不同的云存储系统具有不同的客户端，使用不同的云存储系统需要用户了解使用不同的客户端，给用户增加了学习成本，并且当本地应用程序或系统程序要使用保存在云存储服务器中的文件时，必须先通过云存储客户端将文件下载到本地计算机，然后再进行相应的访问操作，应用程序不能按照通常访问和操作文件系统中文件的方式直接访问和操作保存在云存储服务器上的的文件，也给用户带来了不便[9]。

为了进一步提高云存储系统的安全性，特别是解决云存储系统的运维者对文件的非授权访问的问题，保护用户权益和隐私，同时为了改善用户的操作体验，让用户能用操作本地文件的习惯方式操作网络文件，研究和开发一种基于用户模式的安全网络文件系统具有十分重要的意义。

1.3 国内外研究现状

文件云存储系统是一种通过网络提供文件存储和访问服务的系统[10]。云存储系统的用户可以在任何时间、任何地方，透过任何可连网的设备连接到云存

储系统上方便地上传、下载、共享数据。近年来，随着互联网技术的迅猛发展，网络用户特别是企业用户需要的存储的数据越来越多，而本地存储需要高昂的软硬件维护费用，而云存储利用网络以及系统的高性能服务器使得用户可以通过低廉的价格享受到高效的存储服务，已经成为存储发展的新方向。

目前国内外知名的互联网公司纷纷推出了自己的云存储产品，国外如苹果公司推出的 iCloud，iCloud 能够将用户在苹果系列产品中的文档、照片、购买信息等数据进行实时存储并提供分享功能[11]，再如微软公司推出的 OneDrive（原名为 SkyDrive），它需要通过微软账户登录，已经被集成到了 Windows 8 以后的操作系统中，用户能够方便的存储和分享文档、照片和视频等，再如谷歌推出的 Google Drive，它在拥有传统的文件存储和共享功能的同时还具备强大的搜索功能，能够让用户方便的搜索他人共享的资源文件，并且开放了 API 供其他应用程序使用，用户能够通过任何使用了该 API 的应用程序上传、下载文件[12]；国内如百度公司推出的百度云管家，它提供给用户免费的超大的存储空间，能够实现文件的存储、同步和共享，并且能在绝大多数的移动设备上使用，再如金山软件推出的金山快盘，金山快盘通过文件实时同步技术，把网络文件同步到用户本地计算机的一个文件夹中，方便用户访问和操作网络文件，再如腾讯推出的腾讯微云，同样提供巨大的免费存储空间，并且将其集成在 QQ 中，用户通过 QQ 就能享受腾讯微云提供的文件存储及共享服务。

云存储和传统的本地存储相比有三大优势：第一，节约存储成本，云存储无须用户自己购买硬件和软件，并支付高昂的维护费用，用户仅需支付少量使用费给云存储运营者便能享受云存储服务；第二，更好的备份本地数据，将数据备份到云存储系统不会因为存储设备的遗失、损坏而丢失，具有更好的可靠性；第三，异地处理和共享数据，存放在云存储系统上的文件、数据只要在有网络的地方便可以访问和使用，并能共享给他人，大大提高了效率[13]。云存储的灵活性和便捷性也使得传统的通过软盘，移动硬盘，U 盘等方式进行的移动存储逐渐被取代，云存储已经成为目前人们最常使用的存储方式。

从云存储业务诞生起，安全性始终是企业提供云存储服务的首要考虑的问题之一，对于云存储的用户来说，安全性也是他们使用何种云存储系统的首要考虑因素[14,15]。对于大多数企业用户来说，他们对云存储安全性的要求远超过对云存储系统的业务要求[16]。面对企业用户对于网络存储数据安全性如此之高的要求，针对云存储系统的存在的各种安全问题，业内专家和开发人员已经进行

了相关探讨和研究，并提出了一些改进建议和解决方案，主要的方案有：对存储文件进行加密，对用户进行访问控制，利用安全的传输通道传输文件数据等。

目前传统的文件云存储系统一般都会对存放在服务器上的文件进行加密并采用一定的访问控制机制和文件共享策略，防止第三方非法窃取用户的机密文件、隐私信息而给用户带来损失[17]。它们通常采用的访问控制方案是对用户进行授权，只允许被授权的用户进行访问和操作，云存储的服务器针对某个文件目录对特定用户或者具有某种共同角色身份的用户群设置相关的操作权限（如上传、下载、修改、删除等），同时用户在上传文件时也可以对相应的用户或用户群设置访问权限，没有访问权限的用户则无法访问相应文件，也就不能获取文件的信息，但是这种常用的访问控制方案有一个巨大的缺陷，它的访问控制是在服务端实现的，因此不能防止云存储系统的运维者对文件的非法访问，云存储系统的维护者或者管理员可以通过临时修改访问控制权限的方式使得自己能够访问未经授权的用户文件，可能导致用户信息泄露，这是广大用户目前非常担心的问题。

众所周知，要保护文件信息，最好的办法是对文件数据进行加密，被加密的文件即使被窃取，没有密钥的人也得不到文件信息，能够很大程度上提升安全性，但是这里也存在一个问题，就是如果文件加密是在云存储系统的服务端进行的话，在文件上传到服务器的过程中就容易被他人截获，同时如果上传到服务器再进行加密，云存储系统的运维者同样能够在文件加密之前就窃取文件数据，因此我们需要提出一种方案，在文件上传之前就将文件进行加密，从而防止文件在传输过程中被他人窃取和被云存储系统的运维者非法窃取取。

文件加密可以分为对称加密和非对称加密。对称加密采用的是对称加密算法，对称加密算法到目前为止已经比较完善和成熟，应用也比较广泛，对称加密中，加密时使用一个随机生成对称密钥对明文数据进行加密得到密文数据，在解密时同样使用同一个随机对称密钥对密文数据进行解密得到明文数据，这个过程是对称的，DES、3DES、RC5、AES 等是目前比较主流而且可靠的对称加密算法[18,19]。

在这些主流的对称加密算法中属 AES 加密算法最为常用。AES 是美国联邦政府采用的一种以块为加密单位的标准加密算法，它是一种基于排列和置换的算法[20]，排列指对需要加密的数据进行重排，置换指对需要加密的数据单元进行替换[21]，在密钥的使用上，它可以使用 128,192 或 256 位的随机字符串作为加

解密密钥，分块数据大小则为 16 字节，即 128 位，加密高效安全，十分适合用于文件加密。

非对称加密采用的是非对称加密算法，它的加密解密是不对称的，使用的不是同一个密钥而是一个密钥对，一个公钥，一个私钥，两者必须对应才能进行解密[22]。非对称加密的基本原理是：A 用户生成一对用于加解密的密钥对，并把公钥给加密用户 B，B 用户用 A 的公钥对数据进行加密，再将加密后的密文数据通过网络、移动硬盘等方式交给 A 用户，A 用户再用自己保存的私钥对密文数据进行解密得到明文数据[23]。目前比较成熟可靠的非对称加密算法主要有 RSA、ECC、IBE 等[24,25]。

IBE 算法是目前比较完善和先进的非对称算法。IBE 原型系统是由 Boneh 和 Franklin 在 2001 年提出的[26]。IBE 算法是一种基于身份标识的非对称加密算法，它能够通过用户的身份标识生成加密公钥用于加密，因此与其他非对称加密算法需要其他用户公开加密公钥不同，在进行加密时，加密方只需要知道解密方用户的身份标识就可以完成加密操作，而在进行解密时，解密方只需要使用自己的身份标识申请属于自己的私钥进行解密，其中用户的身份标识可以是用户的身份证号、邮箱地址、域名、手机号等等，这对加密双方来说无疑更加方便和高效，并且具有很高的安全性。

目前在 PC 端，云存储系统通常是通过一个专门的的云存储客户端向用户提供文件存储和共享功能[27]。不同的云存储系统具有不同的客户端，比如百度云管家和苹果 iCloud，两者的客户端界面有很大区别，操作文件的方式同样有很大不同[28]。使用不同的云存储系统需要用户了解使用不同的客户端，给用户增加了学习成本，而且几乎所有的客户端都不能让用户像操作本地文件一样去操作存放在云存储系统中的文件，比如打开存放在服务器上的网络文件时，一般的云存储系统都只提供在线预览功能不能直接进行修改，或者是先把文件下载到本地临时文件目录，然后再打开，当用户修改后再重新上传，如果用户进行反复修改保存操作，系统效率很低，影响体验。

文件系统是计算机操作系统磁盘上组织文件的方法，它简化了系统对文件、文件目录的查找[29]。一台用户计算机，可以有多个具有不同的功能不同特性的文件系统[30]。正是由于文件系统的复杂性和多样性，开发一个特定功能的文件系统的是十分困难的，在用户空间文件系统出现之前，文件系统都是在内核层开发的，需要文件系统的开发者熟练的掌握专业的 C/C++ 编程和调试技能以及

与系统内核开发相关的一些专业知识，同时又需要其他开发人员熟练地操作和测试文件系统以添加个性功能并对其加以改进，因此，在内核态开发文件系统十分不方便而且效率低下[31]。

用户空间文件系统是操作系统的一个概念，区别于在内核层开发的文件系统，它是在用户层实现的[32]。在 Linux 系统下，FUSE 提供了一个 API 库，它是一组实现文件操作的接口[33]，开发者使用这些接口以及 FUSE 文件系统框架可以在用户态开发属于自己的具有完备功能的文件系统，它突破了以往只能在内核层开发文件系统的限制，降低了开发难度，提高了开发效率，因此，FUSE 特别适合于各种虚拟文件系统和网络文件系统的开发。

目前，基于 FUSE 开发的文件系统非常多，功能特性丰富多样、各具特色。如 ExpanDrive，它是 Unix 系统下的一个网络文件系统，定义了文件传输协议，将远程网络文件服务本地化, 极大的方便了用户的操作和管理[34]。ZFS（Zettabyte File System），从最初开发至今已经有 10 多年历史，它是世界上第一个 128 位文件系统，能够存储约 1800 亿亿倍于当前 64 位文件系统的数据，被称为终极文件系统，，性能十分稳定[35]。TFS，阿里针对淘宝大量的小文件数据存储需求而开发的一个分布式文件系统，具有高性能、高扩展性的特点，是我国较为领先的用户空间文件系统[36]。其他基于 FUSE 开发的文件系统还有 glusterfs[37]和 lustre[38]等等，FUSE 文件系统技术目前已经相当成熟。

Dokan 是运行在 Windows 系统上的虚拟文件系统驱动, 它是 Windows 上的用户空间文件系统驱动[39]。对于开发者来说，在 Windows 内核态下开发文件系统驱动程序比在 Linux 下更加复杂和困难，但是利用 Dokan 库开发者就可以非常容易的在用户应用层开发自己的文件系统，节省学习和了解内核原理的时间，同时易于调试和测试，降低开发成本，提高开发效率[40]。利用 Dokan 文件系统驱动所创建的文件系统将被挂载到本地计算机，用户能够用操作本地文件的方式操作和访问新创建文件系统中的文件，非常方便。近年来，国内外的 Windows 开发者不再只局限于在内核层开发文件系统[41]，利用用户空间文件系统驱动在应用层扩展和开发文件系统的开发者也越来越多。

基于以上云存储技术、文件加密技术、文件系统技术的研究现状，本文提出的基于 FUSE 的安全网络文件系统针对目前云存储系统普遍存在的问题, 利用 FUSE 文件系统技术以一种简单高效的方式在用户空间开发一个网络文件系统，采用在客户端对文件进行 AES 对称加密和 IBE 非对称加密相结合的方案，能够

有效保护文件信息，同时采用客户端用户授权的方案更是防止了系统的运维者对文件的非授权访问，最后，区别于其他云存储系统通过专用客户端提供网络文件服务的方式，本系统将网络文件服务器上的文件目录映射到用户本地的一个虚拟文件盘，更是方便了用户操作和使用网络文件，这是一项非常有意义并且值得研究的工作。

1.4 本论文的主要研究工作

本论文要求研究和设计一种基于 FUSE 的安全网络文件系统, 该系统能够让认证用户按常规访问和操作文件系统中的文件的方式访问和操作存放在网络文件服务器中的文件，实现授权用户文件共享，并能防止网络文件服务器的运维者对文件的非授权访问。本论文围绕这个目标，所做的主要工作如下：

（1）详细分析了现有文件云存储系统的优缺点，并在此基础上对研究和开发的安全网络文件系统进行了需求分析和系统设计。

（2）利用 FUSE 文件系统技术将文件服务器上的文件目录映射到用户本地计算的一个虚拟文件盘。

（3）完成了文件客户端程序的开发，该程序将用户对虚拟文件盘中文件和文件目录的操作转化为对文件服务器上对应文件和文件目录的操作，并实现了文件读写的自动加解密。

（4）完成了网络服务程序的开发，该程序采用 Socket 通信原理，定义了通信协议，实现了客户端和服务器的数据通信。

（5）设计了文件解密控制策略，实现了文件的个人共享，群组共享以及文件目录策略保存和继承机制。

（6）采用 MFC 技术设计和实现了策略管理工具，用于策略的更新和管理。

（7）最后，对系统进行了整体测试，并且对完成工作做总结和展望。

本论文的创新点在于利用 FUSE 文件系统技术解决了传统文件云存储系统需要通过专用客户端操作网络文件的问题，让用户操作网络文件更加方便。同时，对文件采用对称加密和 IBE 加密相结合并在客户端设置解密控制策略的方案解决了传统云存储客户端普遍存在的文件共享不够安全的问题，特别是防止了系统的运维者对文件的非授权访问。

1.5 本论文的组织结构

本文总共分为六章，各章节安排如下：

第一章介绍课题的来源以及研究内容的背景和意义，先介绍了文件云存储系统的国内外研究现状和发展水平，并对文件加密技术的发展状况进行了分析，然后介绍了文件系统的开发方式及国内外研究状况，并提出了本文的创新点，最后对本论文的主要工作、工作以及论文的组织结构作了介绍。

第二章首先对目前云存储系统普遍存在的问题进行了分析，对 FUSE 文件系统进行了工作原理的研究，并对系统进行了设计需求分析，设计了系统的系统整体架构，对系统各个模块的功能作了简要介绍。

第三章首先设计了文件加密方案并说明了文件加密和解密的过程和原理，并说明了加密文件的存储结构，然后设计与实现了文件系统客户端程序，包括文件打开、文件写入、文件读取、文件信息获取、文件关闭等各个文件操作函数的具体设计与实现，最后对网络服务程序的通信原理、通信协议、断线检测机制作了介绍。

第四章详细介绍了个人解密控制策略和群组解密控制策略的结构以及个人解密和群组解密的原理，然后介绍了目录策略的保存和继承机制，最后对策略管理工具进行设计与实现，说明了如何通过策略管理工具添加、删除和更新解密控制策略。

第五章首先对系统的整体功能进行了测试与分析，包括用户的登录测试，文件加密、解密测试，通过策略管理工具对解密控制策略的管理测试以及文件目录策略的继承测试，并对系统安全性进行了分析。

第六章对本文所做的工作和系统的实现的功能进行了总结，并对系统的不足之处提出了改进方向，对未来的工作进行了展望。

第 2 章　需求分析与系统整体架构

2.1 系统需求分析

随着网络技术特别是云计算和云存储技术的迅猛发展，传统的存储方式正逐渐被淘汰，文件云存储已经成为人们越来越来常用的存储方式，通过网络我们可以随时访问、操作和共享存储在云存储服务器上的文件，很大程度上提高了工作效率。但在另一方面，云存储带来的安全问题也不容忽视，由于用户都把文件存放在云存储服务器上，因此面临着被非法窃取的可能性，特别是云存储系统的运维者对用户文件的非法访问，某些重要的机密文件或者用户隐私一旦泄露将会对用户造成巨大的损失，同时传统的云存储系统也无法满足用户不通过额外操作就能够用操作本地文件的习惯方式操作网络文件的要求。

本文设计和实现的是一个基于 FUSE 的安全网络文件系统, 不仅需要保证文件上传、下载、共享时的安全，而且需要不改变用户平时的操作习惯，让用户能够像操作计算机本地文件那样操作存放在网络文件服务器上的文件。系统通过文件加密来保证文件存储的安全，通过访问权限的设置和管理，防止系统的运维者对文件的非授权访问, 同时, 利用 FUSE 文件系统技术将网络文件服务器上的文件目录映射到用户计算机上的一个虚拟文件盘，方便用户的操作。根据上述分析，本系统的设计需求如下：

（1）虚拟文件盘的映射，将网络文件服务器上的文件目录虚拟为用户本地计算机的一个虚拟文件盘，将用户对虚拟文件盘内的文件或文件目录的操作转化为对网络文件服务器上文件目录内对应文件或文件目录的操作。

（2）文件的加密，当用户进行上传文件或修改保存文件操作时，文件客户端程序需要自动对明文文件数据进行加密然后上传到网络文件服务器进行保存，确保文件数据的安全。

（3）文件的解密，当文件共享用户进行文件下载、读取操作时，文件客户端程序需要从网络文件服务器获取密文文件数据并自动完成解密，无需用户额外进行解密操作。

（4）文件列表的显示和文件操作的实现，在虚拟文件盘内显示文件列表并

能实现文件的创建、修改、移动、删除等操作，文件列表对应于文件服务器上对应文件目录的文件列表，文件操作对应于文件服务器上对应文件目录内的文件操作。

（5）策略的设置和管理，用户在创建或者上传文件时对文件设置解密控制策略，并且拥有策略管理权限的用户可以通过策略管理工具对文件的解密控制策略进行管理和更新。

（6）文件的个人共享，针对个人用户，满足个人解密控制策略的用户可以访问和操作共享文件。

（7）文件的群组共享，针对群组用户，满足群组解密控制策略的用户可以访问和操作共享文件。

2.2 系统整体架构

本论文所述的基于 FUSE 的安全网络文件系统主要包括文件服务器、文件客户端程序、虚拟文件系统驱动、解密服务器以及身份管理系统五个部分，系统总体架构如图 2-1 所示。

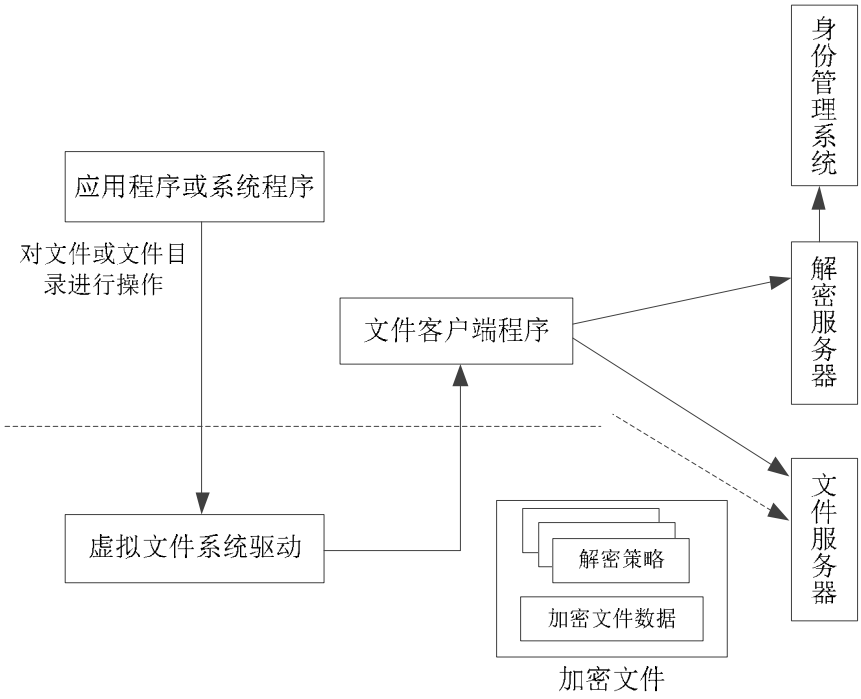

图 2-1 系统整体架构

下面对系统的各个模块进行简要的介绍：

文件服务器：文件服务器是一个用于存放用户网络文件的系统，在文件服务器上存放的文件是加密文件，文件服务器对加密文件实施访问控制策略。同时每一个加密文件都使用一个随机对称密钥进行加密，并且每个加密文件都有一条文件解密控制策略，解密控制策略分为个人解密控制策略和群组解密控制策略。个人解密控制策略指定个人共享用户，每一个共享用户都有一条解密策略与之对应，它分为管理用户策略和普通用户策略两种。群组解密控制策略指定群组共享用户及其操作权限，一个群组对应一条群组解密策略，其中群组共享用户是具有某种角色、属于某个用户群或组、属于某个部门的用户或其他具有某种身份特性的用户。群组解密控制策略有明文、密文两部分，密文部分由明文部分使用对应加密文件的随机对称密钥通过 AES 加密得到，因此明文部分和密文部分指定的群组共享用户是一致的，明文部分用于管理用户查看群组共享用户及其权限，密文部分则作为对群组共享用户的文件操作权限判断的依据。一个加密文件的解密控制策略包含在加密文件中，它是由上传或新建加密文件的用户设置的，拥有管理权限的个人用户或群组用户可以对文件解密控制策略进行管理。文件解密控制策略的具体实现原理将在第 4 章中详细说明。

虚拟文件系统驱动：本文采用的虚拟文件系统驱动是 Windows 系统下的 Dokan 文件系统驱动，它被加载在用户计算机操作系统中，虚拟文件系统驱动借助文件客户端程序将文件服务器上的文件和文件目录映射到用户本地计算机的文件系统的一个虚拟文件盘中，虚拟文件盘是用户计算机的文件系统中的一个显示为文件盘且具有独立文件盘符的文件组织结构，但是虚拟文件盘中的文件和文件目录不是对应于用户计算机的一个真实存在的磁盘驱动器或磁盘卷中的文件和文件目录，而是对应于文件服务器上的加密文件和文件目录，通过映射，虚拟文件盘中的文件和文件目录与文件服务器上的加密文件和文件目录一一对应，虚拟文件系统驱动将系统程序或应用程序针对虚拟文件盘中文件或文件目录的操作请求发送给文件客户端程序，再由文件客户端程序转化为对文件服务器上对应文件或文件目录的操作，关于虚拟文件系统的具体实现原理将在第 3 章中详细说明。

文件客户端程序：文件客户端程序是用户模式文件系统的核心模块，它是一个运行在用户计算机的操作系统用户模式下的用于对存放在文件服务器上的加密文件和文件目录进行操作的应用程序，它将虚拟文件系统驱动发送的用户对虚拟文件盘中文件或文件目录的操作请求（文件的创建、读写、删除等操作，

文件目录的创建、删除等操作）转化为对文件服务器上对应文件或文件目录的操作。

解密服务器：解密服务器是一个根据用户的身份信息和加密文件的群组解密策略判断用户是否具有对应文件操作权限（如文件的读写、删除操作）的系统，解密服务器首先使用群组共享私钥解密由群组共享公钥加密的随机对称密钥，再用解密获得的随机对称密钥解密群组解密策略的密文部分并判断当前用户是否具有该文件的操作权限，若用户具有操作权限，则将随机对称密钥返回给用户，否则则通知用户没有该文件的操作权限。

身份管理系统：它是一个对用户的身份信息进行管理的系统，该系统存储和管理的用户信息包括用户身份标识（用户帐号、手机号、身份证号等)、用户所属角色、用户所属群、用户所属组以及其他证明或描述用户特性的信息，在用户登录或者用户对文件进行操作时都需要访问身份管理系统，对用户进行身份认证。

2.3 本章小结

本章主要介绍了基于 FUSE 的安全网络文件系统的设计需求, 根据设计需求设计系统的整体架构，并对系统各个模块的功能进行了介绍。

第 3 章　基于 FUSE 的网络文件系统的设计与实现

基于 FUSE 的安全网络文件系统由虚拟文件系统驱动、文件客户端程序和网络服务程序 3 个模块构成。虚拟文件系统驱动接收用户或应用程序针对虚拟文件盘中文件或文件目录的操作请求并将请求发送给文件客户端程序，文件系统客户端程序通过网络服务程序将操作请求转化为针对文件服务器上的文件或文件目录的操作，文件客户端程序自动对文件的读写进行加密和解密处理。

3.1 文件加密的设计与实现

3.1.1 文件加密方案设计

为了提高网络文件存储的安全性，对存储在文件服务器上的文件进行加密是一个十分有效的办法。文件加密通常有服务端加密和客户端加密两种方式，但是由服务端来对文件进行加密的方式不能防止网络文件系统的运维者对文件的非授权访问，因此本文采用在客户端进行加密并进行授权访问控制的方式，即在文件上传到文件服务器前由文件客户端程序对文件进行加密并设置解密控制策略，在文件从文件服务器下载到用户本地计算机之前根据解密控制策略对文件进行解密的方式，这种方式下，文件在被上传到服务器之前已经是密文并实行了访问控制，因此系统的运维者不能直接获取用户上传的文件信息也不能通过修改访问控制控制权限的方式间接获取文件信息，从而保证了网络文件存储的安全性。

在加密方案的设计上，若采用单一对称密钥加密的方案十分不合适，因为对于共享文件来说在不同用户之间分发对称密钥是十分麻烦的一件事情，如果对称密钥由共享口令产生，共享口令过于简单则不安全，过于复杂则不方便用户记忆，而且面对大量的共享文件如果使用同一个共享口令，一旦口令泄露，所有文件都将面临泄露的风险，如果使用不同口令让用户记忆则更加不现实；若采用通常的非对称密钥加密方案，即用一个随机对称密钥加密文件，再用每个共享用户的公钥加密该随机对称密钥的方案，这种方案在共享用户较少时是可行的，但是在共享用户很多时，每个共享用户都要用自己的公钥加密随机对

称密钥，会产生大量加密后的密钥数据，这显然是不合理的。因此本系统采用的加密方案在通常的非对称加密方案的基础上做了改进的方案，具体实现为用一个随机生成的对称密钥加密文件，在随机对称密钥的加密上，除了使用少数的几个特定用户的公钥加密随机对称密钥之外对于其他大量的共享用户统一使用一个群组共享公钥加密随机对称密钥，一个共享用户解密共享文件的流程为用自己的私钥解密用自己的公钥加密的随机对称密钥或者用群组共享私钥解密由群组共享公钥加密的随机对称密钥，再用解密获得的随机对称密钥解密加密文件。

对于上述随机对称密钥的生成，常用的方式为利用 srand(time(NULL))函数生成一个随机数种子，再使用随机函数 rand()产生一个固定长度的随机序列，最后再通过简单的哈希算法把生成的随机序列摘要成指定长度，这种方式使用较为普遍，但不够安全，因为这种方式并不是真随机，一旦别人知道了对称密钥生成的时间，就可以利用相同的过程得出加密所用的随机对称密钥进而解密文件窃取文件信息。本文使用 Windows 下的 CryptGenRandom 真随机函数生成随机对称密钥，具体过程为：（1）声明一个密钥容器句柄；（2）使用 CryptAcquireContext 获取密钥容器句柄；（3）使用 CryptGenRandom 函数获得指定长度的随机对称密钥。

在文件加密算法的选择上，本文采用 AES 分块加密算法对文件数据进行加密。所谓分块加密就是把文件分成一块块固定长度的数据块逐一进行加密，本文设置的每个数据块的长度为和随机对称密钥相同长度的 16 个字节，利用 AES_Set_Key(&AesCtx,EncryptKey,128), AES_Encrypt(&AesCtx,in,output)函数完成对 16 字节数据块的加密，但在将文件数据进行分块时，文件末尾的最后一块数据往往不满 16 个字节，此时不能使用 AES 分块加密，本文使用的解决办法是将随机对称密钥进行哈希，再对最后一块不满 16 字节的数据采用按位异或的方式进行加密，文件加密的过程如图 3-1 所示。

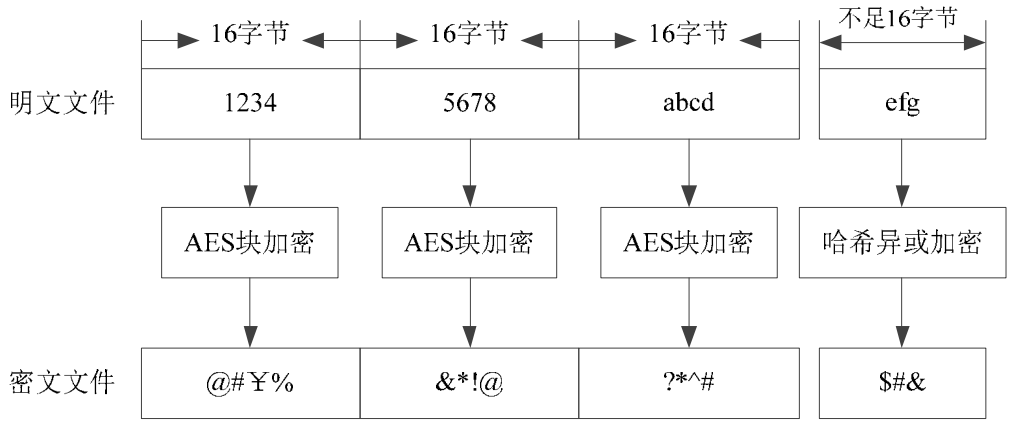

图 3-1 文件加密过程

对称加密不改变文件数据大小，明文数据和密文数据是一一对应的，因此加密数据的解密同样是通过分块来处理，将密文数据分成一个个数据块，利用AES_Set_Key(&AesCtx,StreamKey,128)，AES_Decrypt(&AesCtx,in,output)函数对满 16 字节的数据块进行解密，对于最后一块不满 16 字节的密文数据，用哈希的随机对称密钥再次进行按位异或即可得到明文数据，文件解密的过程如图 3-2 所示。

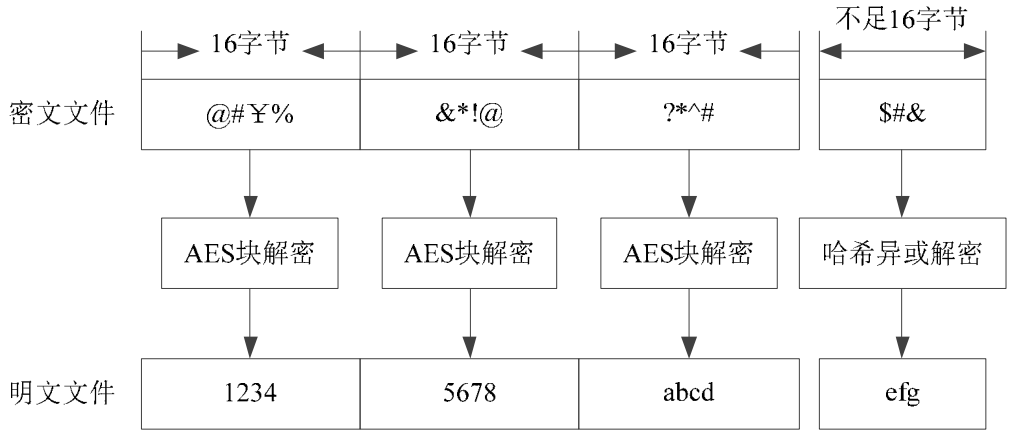

图 3-2 文件解密过程

对于随机对称密钥的加密，本文采用的是基于身份标识的 IBE 算法,IBE 算法能够通过用户的身份标识生成加密公钥，并用此公钥对随机对称密钥进行加密，关于 IBE 密码模块的研究与开发由项目组的其他同学完成，本文不作详细说明，本文调用 IBE 加密和解密的两个接口完成对随机对称密钥的加密和解密，这两个接口的定义如下：

加密接口：BOOL IBE_ENCRYPT(

__In	CONST CHAR*	Identity,
__In	DWORD	Identitylen,
__In	BYTE	*DataIn,
__In	DWORD	* pdwDataIn,
__Out	BYTE**	DataOut,
__Out	DWORD*	pdwDataOut

);

解密接口：BOOL IBE_DECRYPT(

__In	CONST CHAR*	Identity,
__In	DWORD	Identitylen,
__In	BYTE	*DataIn,
__In	DWORD	* pdwDataIn,
__Out	BYTE**	DataOut,
__Out	DWORD*	pdwDataOut

);

其中 Identity 为用户身份标识，Identitylen 为标识长度，DataIn 为需要加密或解密的数据，pdwDataIn 为加密或解密数据的长度，DataOut、pdwDataOut 分别为加密或解密后数据和数据长度，需要注意的是 16 字节的随机对称密钥经过 IBE 加密后将生成 128 字节的密钥数据，对于密钥数据的存放将在下文中介绍。

3.1.2 加密文件结构设计

每一个加密文件都与一个加密信息相对应，加密信息具有两个作用：第一，当加密文件解密时，需要根据加密信息判断用户是否具有解密权限，具有权限的用户能从加密信息中获取得到加密文件所用随机对称密钥，因此包含加密文件的密钥数据、个人解密策略、群组解密策略；第二，根据加密信息需要能够判断文件是否是加密文件，用户上传文件时需要判断文件是否已经加密，以防文件被多次加密，因此加密信息还包含文件加密标识以及加密信息的哈希值。

本文设计的文件加密信息大小为 4KB，用 FILE_ENCRYPT_INFO 结构体来存储，该结构体的定义如下所示：

typedef struct _FILE_ENCRYPT_INFO

```
{
    UCHAR    FileGuid[16];                  //16 字节加密标识
    UCHAR    PersonalStrategy[2032];        //个人解密策略
    UCHAR    GroupStrategy[2032];           //群组解密策略
    UCHAR    HashNum[16];                   //加密信息哈希值
}FILE_ENCRYPT_INFO, *PFILE_ENCRYPT_INFO;
```

该结构的前 16 个字节 FileGuid 为加密标识，它是固定的 16 个字符‘ A ’，接下来的 2032 个字节为个人解密策略，再接下来的 2032 个字节为群组解密策略，最后 16 个字节 HashNum 为加密信息的前 4080 个字节的哈希值，个人解密策略和群组解密策略的格式和结构将在下章中重点说明，这里不做详细介绍。

文件加密信息的保存通常有两种方式：一是单独保存在一个配置文件中，二是直接保存在加密文件中。第一种方式需要含有加密信息的配置文件和加密文件绑定，当下载、读取、移动加密文件时需要同时下载、读取、移动含有加密信息的配置文件，这无疑十分麻烦，增加额外操作，而且降低了系统效率。第二种方式在操作加密文件时则无需额外操作其他配置文件，加密信息可以直接从加密文件中得到，本文正是采用第二种方式保存文件加密信息的。

将文件加密信息保存在加密文件中也一个不足之处，那就是会改变加密文件的结构，加密文件会增加 4KB 大小，因此，文件加密信息该存放在文件的哪个位置是我们需要考虑的问题。一般来说，文件加密信息的存放具有两种方案，

第一种方案是存放在加密文件的头部，其结构如图 3-3 所示。

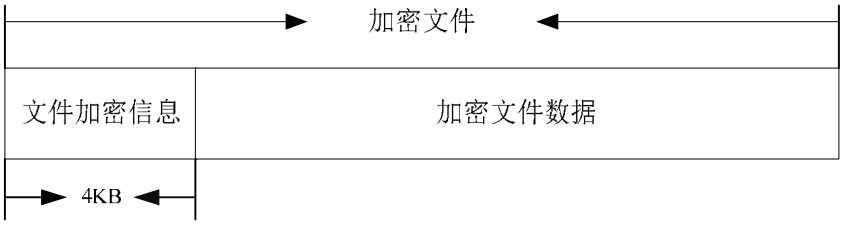

图 3-3 文件加密信息存放在头部的加密文件结构

这种方案加密文件的前 4KB 用于保存文件加密信息，同时所有加密文件数据都依次向后移动 4KB 的长度，这种方案文件加密信息的位置是固定的不会因为加密文件大小的变化而改变，但是在读取加密文件数据时需要格外小心。

第二种方案是存放在加密文件的尾部，其结构如图 3-4 所示。

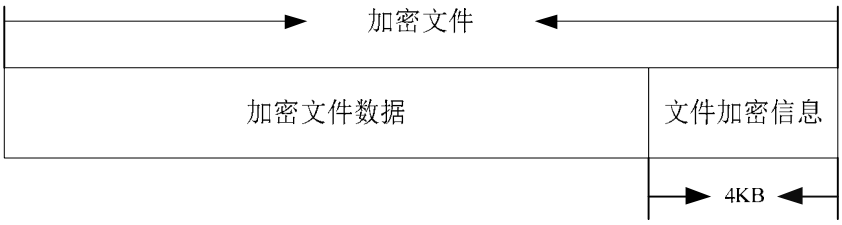

图 3-4 文件加密信息存放在尾部的加密文件结构

这种方案加密文件的尾部存放 4KB 的文件加密信息，实现起来比第一种简单，加密文件数据无需移动，读取加密文件数据时只需去掉尾部的文件加密信息即可，但是文件加密信息的位置不固定，它会随着加密文件大小的改变而变化，如果加密文件的大小经常变化或遇到加密文件非正常关闭的情况，很容易造成文件加密信息的丢失，导致加密文件无法解密，进而造成用户文件数据丢失，这是一个十分严重的问题，而且这个问题很难避免，因此，本文采用的方案是实现稍微复杂但更为可靠的第一种方案。

3.2 文件系统客户端的设计与实现

3.2.1 Dokan 文件系统技术原理

在用户空间文件系统出现之前，文件系统的开发一直是内核开发人员的工作，在内核态下开发文件系统是一件十分复杂和困难的事情，不仅需要专业的内核编程技术而且不易于调试和测试，效率十分低下。本论文所述的基于 FUSE 的安全网络文件系统是基于 Windows 系统下的 Dokan 文件系统驱动设计与开发的。Dokan 是 Windows 系统下的用户态文件系统驱动，可以称之为 FUSE For Windows，利用 Dokan Library 我们可以轻松建立用户空间文件系统，并将所创建的文件系统挂载到本地计算机，即在用户本地计算机资源管理器映射出一个虚拟文件盘，用户可以按常规方式访问和操作虚拟文件盘中的文件。

Dokan Library 主要包含：

- user-mode library (dokan.dll)　Dokan 用户模式库
- driver (dokan.sys)　Dokan 文件系统驱动
- control program (dokanctl.exe)　Dokan 控制程序
- mount service (mouter.exe)　Dokan 挂载程序

dokan.sys 是内核态文件系统驱动，在 Windows 系统上安装此文件系统驱动

以后，我们就可以在 Windows 用户空间创建和普通文件系统一样的文件系统了。我们创建的文件系统的应用程序称之为文件系统客户端程序，用户程序或系统程序发出的文件操作请求（比如：CreateFile，ReadFile，WriteFile 等）将被发送到 Windows I/O Subsystem（运行在内核中），后者会将请求继续发送到 dokan.sys。文件系统客户端程序使用 dokan.dll（Dokan 用户模式库)提供的函数将我们实现的操作函数（回调函数）注册到 dokan.sys 中，当 dokan.sys 收到文件操作请求后调用操作实现函数并将结果返回给用户程序或系统程序。用户空间文件系统原理如图 3-5 所示。

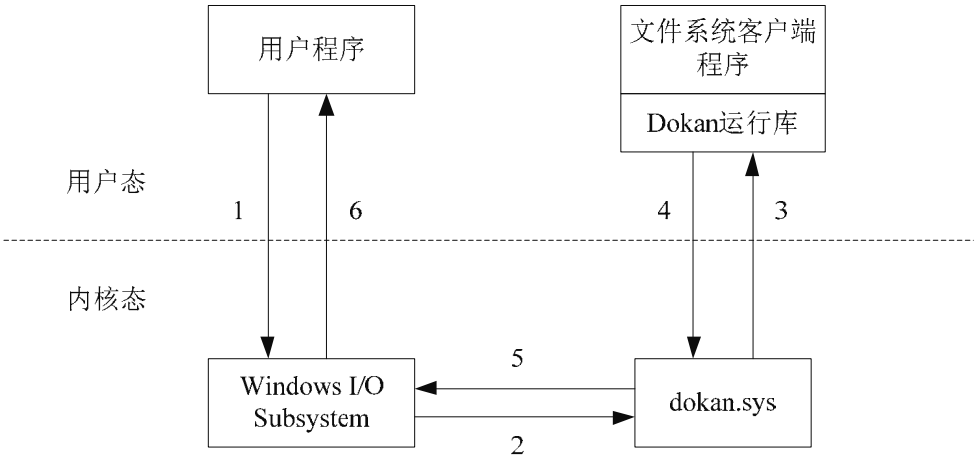

图 3-5 用户空间文件系统原理

以读文件为例来具体说明整个执行过程：

1.用户程序向 Windows I/O Subsystem 发送读文件（ReadFile）请求；

2.Windows I/O Subsystem 将读文件请求发送给 Dokan 文件系统驱动；

3.Dokan 文件系统驱动调用文件系统客户端程序的 ReadFile 回调函数；

4.文件系统客户端程序将获得的数据返回给 Dokan 文件系统驱动；

5.Dokan 文件系统驱动将文件数据返回给 Windows I/O Subsystem；

6.Windows I/O Subsystem 最后将文件数据返回给用户程序。

由上述过程可知，Dokan 文件系统驱动相当于一个运行在内核态的代理，作为发送请求的用户程序和我们实现各种操作的文件系统客户端程序的中间桥梁，能够实现在用户态开发和创建文件系统，提高开发效率。

与 Linux 下的 FUSE 类似，利用 Dokan 创建一个用户空间文件系统，文件系统客户端程序必须要实现 DOKAN_OPERATIONS 结构体中各个操作的回调函数。这些回调函数的的参数与相应的 Windows API 一致，同时为了能够被多个

线程调用，回调函数必须做到线程安全。每个回调函数的最后一个参数都是DOKAN_FILE_INFO 结构体，该结构体的定义如下：

```
typedef struct _DOKAN_FILE_INFO
{
    ULONG64 Context;             //文件句柄
    ULONG64 DokanContext;        //系统保留
    ULONG     ProcessId;         //进程 ID
    BOOL      IsDirectory;       //目录标识位
    BOOL      DeleteOnClose      //删除标识位
} DOKAN_FILE_INFO, *PDOKAN_FILE_INFO;
```

每一个用户程序的文件句柄都与一个 DOKAN_FILE_INFO 结构体相对应，该结构体在文件被 CreateFile 系统调用的时候创建，在文件被 CloseFile 系统调用的时候释放回收。回调函数的典型调用顺序如图 3-6 所示。

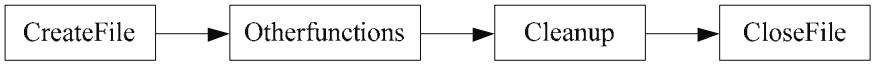

图 3-6 回调函数的调用顺序

3.2.2 文件打开的设计与实现

当应用程序进程对虚拟文件盘中的文件进行操作时，首先进行的是文件打开操作，打开操作在用户空间文件系统中是在 CreateFile 函数中实现的，在DOKAN_FILE_INFO 结构体中，CreateFile 函数的定义如下：

```
static int CreateFile(
    __In        LPCWSTR              FileName,
    __In        DWORD                AccessMode,
    __In        DWORD                ShareMode,
    __In        DWORD                CreationDisposition,
    __In        DWORD                FlagsAndAttributes,
    __In        PDOKAN_FILE_INFO     DokanFileInfo
);
```

其中 FileName 是被操作的文件名， AccessMode 、 ShareMode 、

CreationDisposition 和 FlagsAndAttributes 分别是文件的访问模式、共享模式、打开方式和打开属性，它们都在执行文件操作时由 dokan.sys 传入，DokanFileInfo 是本次操作的文件信息。

本文对文件打开操作的处理步骤如下：

第一步：CreateFile 函数将文件名和文件参数传给文件服务器，服务器需要判断被操作的文件是文件目录还是文件，若是文件目录，则直接返回，将 DokanFileInfo->IsDirector 设置为 TRUE，转到 CreateDirectory 函数中处理，并结束操作，若是文件，则转到下一步；

第二步：文件服务器根据传递的文件参数打开或创建文件，并获取文件信息，根据文件信息判断文件大小是否为 0，将判断结果与文件句柄一同返回给 CreateFile 函数，若文件大小为 0，则转到第三步，否则转到第四步；

第三步：生成针对当前文件的 4KB 大小的文件加密信息，并将文件加密信息发送给服务器写入到对应文件中，完成对空文件的加密，转到下一步；

第四步：将之前获得的文件句柄保存到 DokanFileInfo->Context 中，供其他回调函数调用和访问。

为了保证对文件服务器上的每一个文件都被加密，这里为文件大小为 0 的文件添加了 4KB 大小的文件加密信息，因为对于文件大小为 0 的文件不论是在新创建还是上传的过程中都不会调用 WriteFile 函数，而本文的文件加密处理都是在 WriteFile 函数中完成的，因此需要在这里进行特殊处理，生成文件加密信息的具体过程为：

（1）设置文件加密信息的加密标识；

（2）设置文件个人共享用户并用个人共享用户标识对加密文件数据的随机对称密钥进行 IBE 加密生成个人解密控制策略；

（3）设置文件群组共享用户并用群组共享用户标识对加密文件数据的随机对称密钥进行 IBE 加密生成群组解密控制策略；

（4）对加密标识、个人加密策略、群组解密策略进行哈希获得文件哈希值；

（5）由加密标识、个人加密策略、群组解密策略、文件哈希值组合生成文件加密信。

3.2.3 文件写入的设计与实现

文件的写入和读取操作是本安全网络文件系统的核心，应用程序或系统程

序对虚拟文件盘内文件执行写入操作时，文件系统客户端程序需要对文件数据进行加密并设置解密控制策略将文件保存到文件服务器，写入操作在用户空间文件系统中是由 WriteFile 这个回调函数实现的, 在 DOKAN_FILE_INFO 结构体中，WriteFile 函数的定义如下：

```
static int WriteFile(
    __In        LPCWSTR             FileName,
    __In        LPCVOID             Buffer,
    __In        DWORD               NumberOfBytesToWrite,
    __Out       LPDWORD             NumberOfBytesWritten,
    __In        LONGLONG            Offset,
    __In        PDOKAN_FILE_INFO    DokanFileInfo
);
```

其中参数 FileName 为被操作的文件名，Buffer 为将要写入的文件数据，NumberOfBytesToWrite 为将要写入的文件数据字节数，NumberOfBytesWritten 为实际写入的文件数据字节数，是一个返回值，Offset 是文件偏移量，DokanFileInfo 中则包含了本次文件操作的信息。

本文中将文件的写入操作分为受信进程写入和非受信进程写入，DokanFileInfo 中保存有操作进程 ID，通过遍历系统进程，对比进程 ID 可以得到本次操作的系统进程名，将得到的进程名在与本地配置文件中保存的进程名作比较即可判断出操作进程是受信进程还是非受信进程。

对于受信进程来说，将要写入的文件数据一定是明文，因此需要对文件数据进行加密。文件数据的加密采用上节所述的加密方案，首先生成一个随机对称密钥，再用随机对称密钥对文件数据进行 AES 分块加密。

当文件偏移量 Offset 为 0 时，写入数据从文件的起始位置开始，此时还需要为文件添加文件加密信息，写入文件数据的字节数相应增加 4KB，生成文件加密信息的过程与上文中打开大小为 0 的文件时生成文件加密信息的过程一致。

当文件偏移量 Offset 不为 0 时，写入的是文件的中间数据，写入文件数据的字节数不用增加，但是由于加密文件在头部增加了一个 4KB 的文件加密信息，因此此时的文件数据需要向后移动 4KB，即文件偏移量 Offset 需要增加 4KB。

WriteFile 函数完成文件数据的加密之后，将加密后的文件数据（包括 4KB 的文件加密信息）上传到文件服务器，由服务器完成相应的写入保存操作并返

回实际写入的字节数保存到 NumberOfBytesWritten 中，由 WriteFile 函数返回给 dokan.sys 完成本次写入操作，受信进程写入文件数据的整个流程如图 3-7 所示。

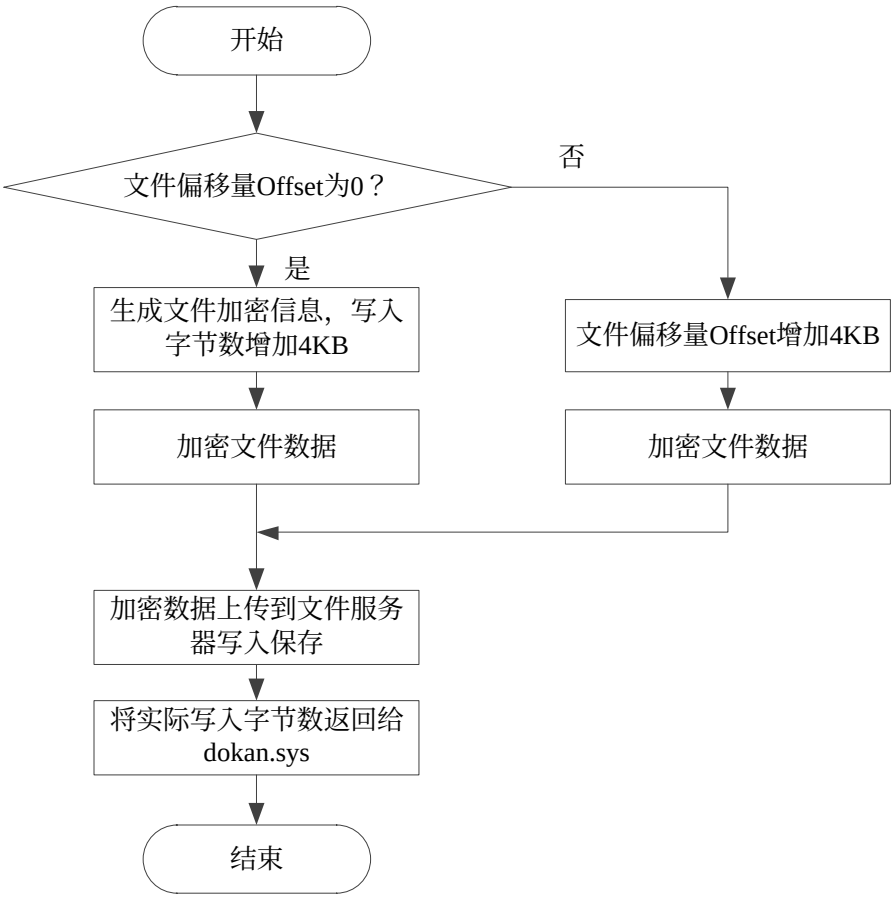

图 3-7 受信进程文件写入流程

对于非受信进程来说，将要写入的文件数据可能是明文也可能是密文，本文中用布尔类型的加密标志 IsEncrypt 来判断文件数据是否已加密，IsEncrypt 为 TRUE 则表示文件数据已加密，为 FALSE 则表示文件数据未加密，该标志在文件偏移量 Offset 为 0 时进行重置判断和保存，由于加密文件都有一个 4KB 的加密头部，所以如果写入数据小于 4KB 则一定是明文，对于大于 4KB 的写入数据取前面的 4KB 数据判断其是否为文件加密信息，若是则为明文，若否则为密文，整个判断流程如图 3-8 所示。

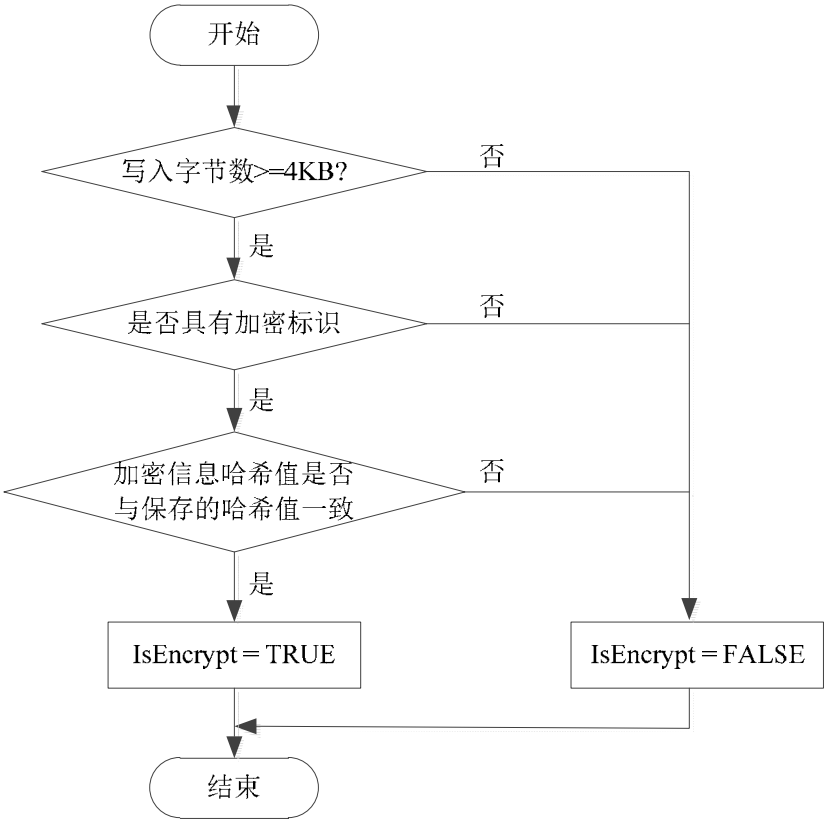

图 3-8 文件是否加密判断流程

如果写入的文件数据是明文则同样需要对文件数据进行加密，其处理过程与受信进程一致，如果是密文则不对文件数据进行处理，直接将密文文件数据上传到文件服务器写入保存，并返回实际写入的数据长度，非受信进程写入文件数据的整个流程如图 3-9 所示。

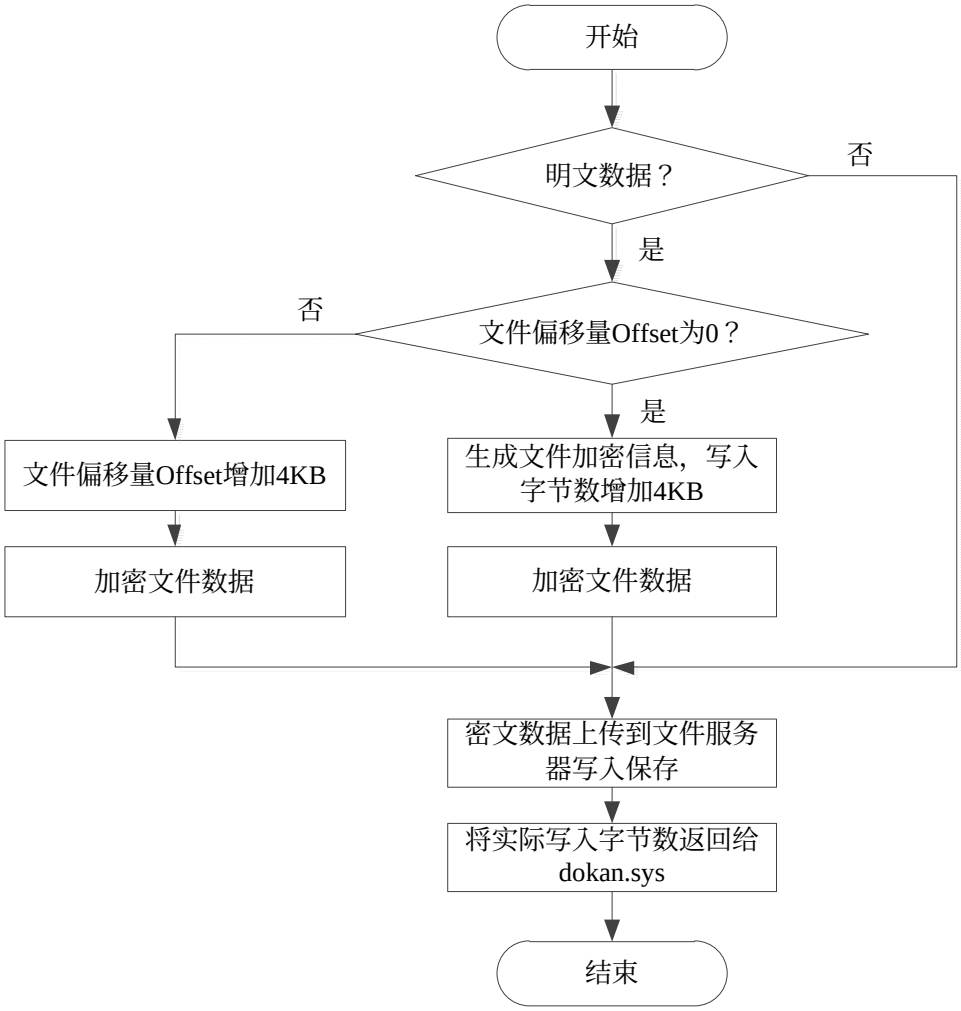

图 3-9 非受信进程文件写入流程

3.2.4 文件读取的设计与实现

与文件的写入相对应, 文件的读取操作在用户空间文件系统中是由 ReadFile 这个回调函数实现的，在 DOKAN_FILE_INFO 结构体中，ReadFile 函数的定义如下：

```
static int ReadFile(
    __In      LPCWSTR          FileName,
    __Out     LPVOID           Buffer,
    __In      DWORD            BufferLength,
    __Out     PDWORD           ReadLength,
    __In      LONGLONG         Offset,
```

```
    __In        PDOKAN_FILE_INFODokanFileInfo
);
```

其中参数 FileName 为操作文件名，Buffer 为返回给 dokan.sys 的文件数据，BufferLength 为需要获取的数据字节数, ReadLength 为实际读取的字节数, Offset 为读取文件时的偏移量，DokanFileInfo 中则包含了本次文件操作信息。

本文中，文件数据的读取同样分为两类，受信进程读取和非受信进程读取。非受信进程的读取相对较为简单，从服务器获得的密文数据无需进行解密处理直接存放到 Buffer 中返回给 dokan.sys 进而返回给应用程序。下面重点介绍受信进程文件数据的读取。

当偏移量 Offset 为 0 时，读取数据从文件的起始位置开始，由于加密文件在头部包含 4KB 大小的文件加密信息，解密文件数据需要从文件加密信息中获取加密文件所用的随机对称密钥，因此此时 BufferLength 需要增加 4KB，向服务器多请求 4KB 大小的文件数据来获得所需文件数据和文件件加密信息。根据文件加密信息得到随机对称密钥的过程为：

第一步：获取文件加密信息中的个人解密控制策略，通过当前用户标识判断是否有当前用户对应的一条个人解密控制策略，若有则转到第二步，否则转到第三步；

第二步：用当前用户的私钥解密个人解密控制策略中保存的对称密钥加密数据得到加密文件的随机对称密钥，结束本次操作；

第三步：获取文件加密信息中的群组解密控制策略，并与用户帐号、用户密码一起发送到解密服务器，由解密服务器判断用户是否具有解密权限，若用户具有解密权限，则用群组共享私钥解密保存在群组解密策略中的由群组共享公钥加密的随机对称密钥，并将解密得到的随机对称密钥返回，若用户无解密权限，则直接返回，结束本次操作。

经过上述过程，若得到了随机对称密钥，则将该随机对称密钥保存并用得到的随机对称密钥对文件数据进行 AES 分块解密，将解密后的明文数据放到 Buffer 中返回，并将解密权限标志 IsPermit 置为 TRUE，若没有得到随机对称密钥，则提示用户没有解密权限，将密文数据放入到 Buffer 中返回，并将解密权限标志 IsPermit 置为 FALSE。同时在返回实际读取的数据长度 ReadLength 时，由于多读了 4KB 的文件加密信息，因此需要将这 4KB 长度减去，否则 docan.sys 会认为本次多读了数据而在之后的读取中少读数据，造成数据的丢失。

当偏移量 Offset 不为 0 时，读取的是文件的中间数据，由于加密文件的头部是 4KB 的文件加密信息，因此实际读取的数据应向后偏移 4KB，即 Offset 应当增加 4KB。对于获取的加密文件数据，根据 IsPermit 判断用户是否具有解密权限，若有解密权限则用保存的随机对称密钥解密文件数据将明文数据放到 Buffer 中返回，若无解密权限，则将密文数据放到 Buffer 中返回。此时读取的文件数据只是做了偏移，实际读取的文件数据长度并未增加，因此将 ReadLength 直接返回。

3.2.5 文件缓存的设计与实现

在读取文件数据时，如果反复连接服务器获取数据是一件效率十分低下的事情，特别是有的应用进程在读取文件时每次只读几个字节，而每次只从服务器获得几个字节的数据也是十分不科学的，因此本文在文件读取时设置一个了 64K 的缓存，用于存放文件缓存数据。

本文定义的缓存结构如下：

```
typedef struct _FileCache
{
    WCHAR FileName[256];
    ULONG   Offset;
    ULONG   Length;
    UCHAR   CacheBuffer[65536];
}FileCache;
```

该结构中 FileCache.FileName 为缓存文件的文件名，FileCache.Offset 为缓存文件的起始偏移位置，FileCache.Length 为缓存文件数据大小，FileCache.CacheBuffer 为缓存的文件数据。

当 ReadFile 函数读取文件数据时首先需要判断缓存是否为空，待读取文件名和缓存文件名是否一致，当缓存为空，或者文件名不一致时，首先需要向服务器获取 64KB 数据刷新缓存。当缓存不为空且读取文件的文件名和缓存文件的文件名一致时，根据读取数据的偏移量 Offset、读取文件的长度 BufferLength 和缓存数据的偏移量 FileCache.Offset、缓存数据的长度 FileCache.Length 之间的关系，共有以下 5 种情况：

第一种情况：Offset+BufferLength 小于 FileCache.Offset，即待读取数据全部

在缓存数据之前，这种情况如图 3-10 所示。

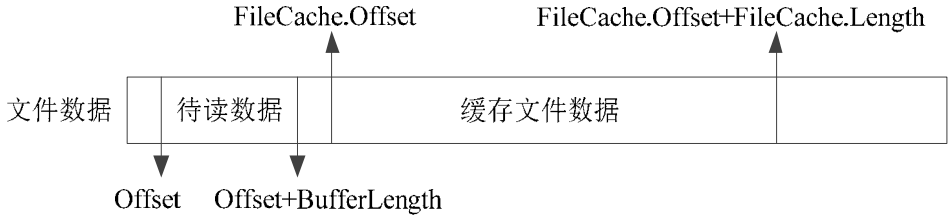

图 3-10 待读数据全部在缓存数据之前

这种情况下缓存中没有可读数据，因此需要重新向服务器获取数据，请求数据的文件偏移量为 Offset，数据长度应为缓存最大长度 64KB，将获得的文件数据放到 FileCache.CacheBuffer 中刷新缓存，将此时的文件偏移量 Offset 保存为新的 FileCache.Offset，将实际获得的数据长度保存为新的 FileCache.Length(因为剩余文件数据长度可能没有 64KB)，最后将待读的文件数据返回处理。

第二种情况：FileCache.Offset 大于 Offset 并且小于等于 Offset+BufferLength，即待读数据部分在缓存文件数据之前，部分在缓存文件数据之内，这种情况如图 3-11 所示。

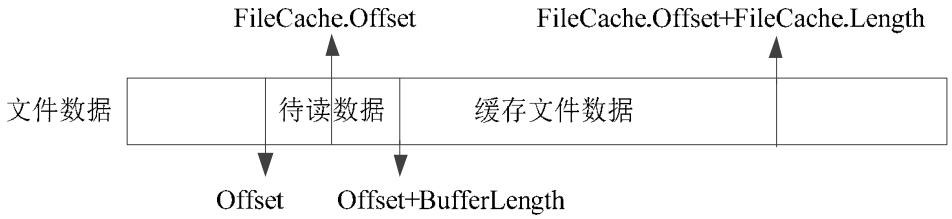

图 3-11 待读数据部分在缓存数据之前部分在缓存数据之内

这种情况下，缓存文件中有部分文件数据可用，但是由于没有前面的数据，因此也要向服务器获取新的数据，dokan.sys 在通过 ReadFile 函数请求数据时一次读取的数据长度（BufferLength）不会超过 64KB，因此向服务器获取的新的文件数据必然包含之前可用的数据，这种情况下刷新缓存和第一种情况下相同，刷新缓存后将待读数据返回处理即可。

第三种情况：Offset 小于等于 FileCache.Offset 并且 Offset+BufferLength 小于等于 FileCache.Offset+FileCache.Length，即待读数据全在缓存数据之内，这种

情况如图 3-12 所示。

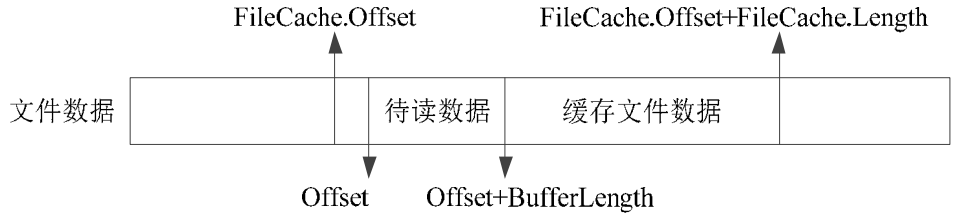

图 3-12 待读数据全在缓存数据内

这种情况下，无需再向服务器请求数据，直接从缓存文件数据中取出待读文件数据返回处理即可，并且不需要刷新缓存。

第四种情况：FileCache.Offset+FileCache.Length 大于等于 Offset 并且小于 Offset+BufferLength，待读数据部分在缓存数据之内部分在缓存文件数据之后，这种情况如图 3-13 所示。

图 3-13 待读数据部分在缓存数据之内部分在缓存数据之后

这种情况下，取出可用的缓存文件数据保存，并以文件偏移量 FileCache.Length、请求数据长度 64kB 向服务器获取新的数据刷新缓存，将 FileCache.Length 保存为新的 FileCache.Offset，将实际获得的数据长度保存为新的 FileCache.Length,并将缺少的待读数据补充完整返回处理。

第五种情况：Offset 大于 FileCache.Offset+FileCache.Length，即全部待读数据在缓存文件数据之后，这种情况如图 3-14 所示。

图 3-14 全部待读数据在缓存数据之后

这种情况下，缓存中同样没有可用数据，因此需要向服务器获取数据刷新缓存，并将待读数据返回处理，刷新缓存过程与第一种情况相同。

本系统设计的缓存的目的在于提高打开文件的速度，在实际应用中，文件大小一般不是很大，并且很多应用程序进程在读取文件时都是少量多次读取，因此大多数情况满足第三种情况，从而减少了连接服务器获取数据的次数，提高了效率，达到了设计目的。文件缓存基于文件读取而设计，因此当文件被打开进行写入保存或者进行文件更新并重新上传到服务器时，进行清缓存处理，以避免再次读取时获取到未更新时的数据。

3.2.6 文件信息查询的设计与实现

应用程序在对文件进行读写操作时，会反复查询文件信息，特别是文件的大小信息，如果给出的文件信息不恰当或者有误，读写操作就会出错，因此必须确保给出的文件信息的准确性，在用户空间文件系统中文件信息查询是在 GetFileInformation 回调函数中实现的，在 DOKAN_FILE_INFO 结构体中，该函数的定义为：

```
static int GetFileInformation(
  __In    LPCWSTR                     FileName,
  __Out   LPBY_HANDLE_FILE_INFORMATION HandleFileInformation,
  __In    PDOKAN_FILE_INFO            DokanFileInfo
);
```

HandleFileInformation 是一个存储文件信息的结构体，通过它我们将需要返回的文件信息返回给 dokan.sys。GetFileInformation 回调函数在处理时，首先将文件名发送给文件服务器，文件服务器找到对应文件，并调用 GetFileInformationByHandle(handle,&HandleFileInformation)函数得到文件信息并

返回，对于接收到的文件信息，我们需要注意的是文件的大小信息，因为在用户空间文件系统中 ReadFile 函数读取文件数据的总大小是根据该函数查询得到的文件大小决定的，由于服务器上存放的是加密文件，包含 4KB 的文件加密信息，对于受信进程来说需要获取的文件数据不应包括这 4KB 的文件加密信息，因此在得到服务器传递过来的文件大小后需要减去 4KB，再返回给 dokan.sys，如果不减去这 4KB 则会导致多读数据而出错，对与非受信进程来说，需要读取的是包含文件加密信息在内的所有文件数据，因此不需要减去 4KB，直接将文件信息返回。

3.2.7 文件关闭的设计与实现

当文件操作完成后，需要关闭文件解除文件占用，虽然在文件关闭时，Cleanup 函数和 CloseFile 函数会先后被调用, 但是在本文中关闭文件操作实际是在 Cleanup 函数中完成的，Cleanup 回调函数在 DOKAN_FILE_INFO 结构体中，该函数的定义为：

```
static int Cleanup(
    __In        LPCWSTR                 FileName,
    __In        PDOKAN_FILE_INFO        DokanFileInfo
);
```

在该函数中将文件名 FileName 和 DokanFileInfo 发送给服务器, 服务器根据 DokanFileInfo->Context 保存的文件句柄关闭对应文件，并且如果 DokanFileInfo->DeleteOnClose 为 TRUE，则在关闭文件后删除文件。

3.2.8 文件其他操作的设计与实现

除了上文说明的文件操作外，对于一个完整的网络文件系统来说其他的文件操作同样不可缺少，各个操作函数的相互配合才能实现整个系统的功能，下面简要介绍其他几个重要操作函数的设计与实现。

（1）FindFiles 回调函数，该函数实现虚拟文件和目录的构造，在 DOKAN_FILE_INFO 结构体中，该函数的定义为：

```
static int FindFiles(
    __In        LPCWSTR                 FileName,
    __In        PFillFindData           FillFindData,
    __In        PDOKAN_FILE_INFO    DokanFileInfo);
```

当打开虚拟文件盘或虚拟文件盘中文件目录时，FindFiles 函数将会被调用，服务器根据发送过来的 FileName 找到对应的文件目录，查找该目录下的所有文件和子目录并将查找到文件信息和子目录信息返回，该函数利用作为参数传递过来的函数指针 FillFindData(&win32FindDataw, DokanFileInfo)将文件信息保存到 DokanFileInfo 中返回给 dokan.sys，需要特别处理的是每个文件目录下都有一个名为“ _____@@@@@_____.ini ”的策略文件，该文件对于用户来说是隐藏的，因此需要过滤掉。

（2）SetEndOfFile 函数，设置文件真实大小的函数，在 DOKAN_FILE_INFO 结构体中，该函数的定义为：

```
static int SetEndOfFile(
        __In    LPCWSTR             FileName,
        __In    LONGLONG            ByteOffset,
        __In    PDOKAN_FILE_INFO    DokanFileInfo);
```

其中 ByteOffset 参数为设置的 EndOfFile，表示文件的实际大小，在受信进程进程操作时，由于此前在 GetFileInformation 函数中减去了 4KB 大小的文件加密信息，对于文件的真实大小来说比原来减少了 4KB，因此在这里 ByteOffset 还需要加上这 4KB，在非受信进程操作时无需处理。

（3）SetAllocationSize 函数，该函数用于设置文件实际占用的空间大小，在 DOKAN_FILE_INFO 结构体中，该函数的定义为：

```
static int SetAllocationSize(
        __In    LPCWSTR             FileName,
        __In    LONGLONG            AllocSize,
        __In    PDOKAN_FILE_INFO    DokanFileInfo);
```

其中 AllocSize 参数表示设置的 AllocationSize，即文件的占用空间大小，与 SetEndOfFile 函数中的 ByteOffset 一样，它也表示文件真实占用空间值，因此在受信进程进行操作时同样需要加上 4KB 大小，将文件加密信息包含进去，非受信进程操作同样不做处理。

实现了 DOKAN_OPERATIONS 中的各个回调函数以后，可以说已经完成了用户空间文件系统的创建，但是挂载文件系统还需要为 Dokan 库填写 DOKANOPTIONS（包含 Dokan 版本信息、使用线程数、挂载点等信息），填写了 DOKANOPTIONS 之后，就可以调用 DokanMain 函数来挂载文件系统，该

函数会阻塞到文件系统被卸载。用户空间文件系统被创建和挂载之后，就可以操作虚拟磁盘内的文件和目录了。

3.3 网络服务程序的设计与实现

3.3.1 Socket 通信原理

客户端和服务器端之间的网络通信是实现一个安全网络文件系统的前提，本文采用 Socket 技术进行客户端和服务器端数据传输和交换，实现文件的上传、下载等功能。Socket 又称套接字，是一种网络编程的底层接口，通过 Socket 能够访问通信协议从而实现不同计算机或不同虚拟机之间的网络通信。Socket 通信方式可以分为面向连接的字节流类型（流式套接字）和面向无连接的数据报类型（数据报套接字）两种。网络文件系统的客户端和服务器之间需要进行大量的数据传输，并且需要确保数据在各种网络环境下的准确性，因此本文采用面向连接的字节流类型的 Socket 实现客户端和服务器端的网络通信。

面向连接的字节流类型的 Socket 采用 TCP 协议，提供可靠的、双向的、有序无重复的字节流服务，通信时需要经过建立连接、使用连接、关闭连接 3 个过程，并且具有数据校验和超时重发机制，能够确保数据传输的准确性。流式套接字的工作流程如图 3-15 所示。

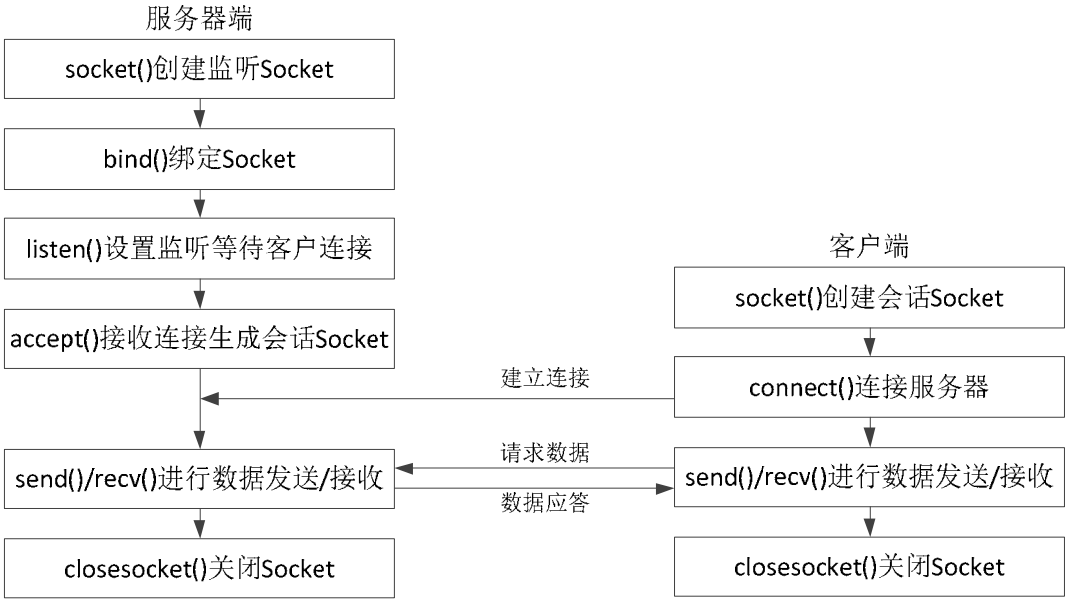

图 3-15 面向连接的 Socket 工作流程

服务器先于客户端启动，服务器使用 socket()函数创建监听套接字，将所创建的监听套接字通过 bind()函数绑定到本地地址和端口，然后调用 listen()函数进入监听状态等待客户端连接，并通过 accept()函数接收客户端连接请求，客户端在用 socket()函数创建一个会话套接字后调用 connect()函数连接服务器，当服务器和客户端的连接建立以后，双方就可以使用 send()函数和 recv()函数进行通信，通信完成后双方调用 closesocket()函数关闭会话套接字。

3.3.2 通信协议的设计

客户端程序与服务器建立 Socket 连接后用户首先要完成登录操作，本文在 TCP 协议的基础上，在应用层设计了用户登录协议，登录协议包含一个登录码以及用户账号和用户密码，其结构如图 3-16 所示。

图 3-16 用户登录协议结构

客户端在请求登录时向服务器发送一个包含登录码的数据包，服务器收到数据包根据登录码得知用户请求登录后，获取数据包中的用户账号和密码访问身份管理系统对用户进行身份认证，当用户登录通过身份认证完成登录，服务器会创建一个子线程来实现当前用户与服务器之间的数据通信。

本文中采用的基于 TCP 协议的 Socket 通信方式是通过字节流发送数据的，如果不设计合理的通信协议，在数据收发过程就可能会出现粘包甚至数据接收不完整的现象，因此在发送数据时需要对数据进行封包处理，为数据添加一个固定长度的数据包头，本文中数据通信协议的设计如图 3-17 所示。

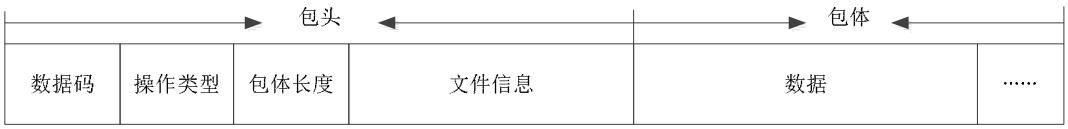

图 3-17 数据通信协议结构

包头中的文件信息用如下结构体存储：

```
typedef struct _FILE_INFO
{
```

```
    HANDLE Handle;
    WCHAR FilePath;
    ULONG Offset;
    DWORD BufferLength;
}FILE_INFO,*PFILE_INFO;
```

Handle 为操作文件的句柄，FilePath 为操作文件的路径，Offset 为文件偏移量，BufferLength 为请求文件数据大小，该变量在读文件时起作用。

使用该通信协议，在文件客户端程序与服务器进行数据通信时，服务器根据数据码可知客户端发起数据通信请求，接着读取一个包头长度的数据，获得包体长度然后完整的接收剩余数据，最后根据包头中的文件操作类型标识、文件信息以及文件数据完成相应的文件操作，完成相应文件操作后如有数据需要返回（如文件读取操作）则按相同的通信协议将数据发送给客户端，由客户端完成相应处理。

3.3.3 网络连接断线检测机制设计

本系统中用户登录成功以后，客户端和服务器之间的连接就建立了，在理想状态下，该连接会一直保持，直到用户退出连接关闭。在实际使用中,客户端和服务器之间的通信通常需要穿过多个中间节点，如路由器、防火墙、网关等，其中一个节点出了问题就会导致 Socket 连接断连，除此之外，如果客户端出现断电、系统崩溃、异常重启等事故而导致连接未能正常关闭，也会导致 Socket 连接断连。每一个连接都会占用服务器的系统资源，而维护大量断连的连接就会造成服务器资源的浪费，并且本系统中客户端的文件操作与服务器端是相对应的，如果客户端打开了文件，在服务器端相应的文件也处于打开状态，一旦客户端出现非正常关闭，就会导致服务器打开的文件永久被占用。因此，服务器需要定时检测客户端是否掉线，在检测到客户端断线时及时释放系统资源、关闭文件、解除文件占用。

检测 Socket 连接是否断线主要有三种方法：SO_KEEPALIVE 机制、SIO_KEEPALIVE_VALS 机制和心跳线程机制。

（1）SO_KEEPALIVE 机制

SO_KEEPALIVE 机制是 Socket 库提供的功能，具体设置为：

```
BOOL SetSocket = TRUE;
```

setsockopt(msocket,SOL_SOCKET,SO_KEEPALIVE,(const char*)&SetSocket, sizeof(BOOL));

当为 socket 设置了 KEEPALIVE 选项后, 如果 2 小时内客户端和服务器双方之间没有数据交换，TCP 就会自动给对方发送一个 keepalive probe（保持存活探测分节）请求对方响应，发送探测分节后有以下三种情况：

1.对方成功接收，并以期望的 ACK 响应，则说明对方未掉线，2 小时后再发送另一个探测分节。

2.对方已崩溃且已经重新启动以 RST 响应，此时 socket 的待处理错误被置为 ECONNRESET，并关闭该 socket。

3.对方无任何响应，此时每隔 75 秒 TCP 会再次发送一个 keepalive probe，总共发送 8 次，如果 8 次之后对方仍然无响应则放弃,socket 的待处理错误被置为 ETIMEOUT，并关闭 socket。

这种机制实现起来十分简单，但是空闲 2 小时才发送一个保活保持存活探测分节不能保证实时检测，当然这种机制也可以通过修改时间参数来减小探测间隔，但是这会影响到所有设置了 KEEPALIVE 选项的 socket。

（2）SIO_KEEPALIVE_VALS 机制

该机制是 Windows 提供的功能，通过 WSAIoctl 这个 API 实现，使用时需要添加 MSTCPiP.h 头文件，该头文件中定义了 tcp_keepalive 结构体，该结构如下：

```
struct tcp_keepalive
{
ULONG onoff;
ULONG keepalivetime;
ULONG keepaliveinterval;
};
```

其中 onoff 表示是否启用,为 1 时启用，为 0 时不启用，keepalivetime 为连接空闲多长时间发送探测包，单位为毫秒，keepaliveinterval 为探测无响应后的重发间隔，单位也为毫秒，具体实现为：

```
DWORD dwBytes;
tcp_keepalive m_Input = {0}, m_Output= {0} ;
m_Input.onoff = 1 ;
m_Input.keepalivetime = 60000 ;
```

m_Input.keepaliveinterval = 3000 ;

WSAIoctl(msocket,SIO_KEEPALIVE_VALS, &m_In,sizeof(tcp_keepalive), &m_Output,sizeof(tcp_keepalive), &dwBytes,NULL,NULL) ;

SIO_KEEPALIVE_VALS 机制的实现原理和 SO_KEEPALIVE 机制的实现原理基本相同，上述设置为当连接空闲 1 分钟后向对方发送探测包，如无响应则每隔 3 秒再次发送探测包, 默认探测次数为 5 次, 如无响应则放弃并关闭 Socket, 与 SO_KEEPALIVE 机制不同是，该设置只针对当前使用的 socket,不会影响其他 socket,不足是只能在 Windows 下使用，不具有通用性。

（3）心跳线程机制

心跳线程机制是指在客户端和服务器建立连接以后，服务器启用一个后台线程每隔固定时间发送一个特定心跳数据包信息给客户端，客户端收到心跳数据包后向服务器发送一个响应包，如果服务器在指定时间内没有收到客户端发送的响应包则认为客户端已经掉线，服务器将连接关闭。

心跳线程机制是在应用层实现的，它的的优点使用灵活，通用性强，不依赖于传输协议，探测间隔探测次数都可根据实际情况设计，缺点是应用层的数据包传递到协议层时会被加上额外的包头包尾，因此需要消耗更多的流量，同时每个连接都需要维护一个后台线程，当有大量客户端在线时，服务器需要消耗大量的系统资源维护这些后台线程。

通过对比三种掉线检测机制的优缺点，本文采用的是第二种掉线检测机制，在实际应用中能够有效检测客户端是否掉线，并且消耗的系统资源很少，不影响客户端和服务器的数据通信。

3.4 本章小结

本章主要围绕基于 FUSE 的安全网络文件系统的实现展开, 针对保护文件信息的问题设计了文件加密方案，包括加密方式的选择、加密算法的选择以及加密文件的结构设计，接着着重完成文件系统客户端程序的设计，包括文件写入加密、文件读取解密的实现，文件读取缓存的设计及其它文件操作函数的实现，最后设计与实现了网络服务程序，在 Socket 通信原理的基础上完成了客户端程序与服务器的通信协议及网络连接掉线检测设计。

第 4 章　策略管理的设计与实现

解密控制策略的设计也是本安全网络文件系统需要实现的重点之一，本章主要介绍个人解密控制策略和群组解密控制策略的实现原理，解密控制策略在加密文件中的存储结构，并使用 MFC 设计和实现了策略管理工具。

4.1 个人共享的原理与实现

所谓文件的个人共享是指针对某个特定的个人授予解密加密文件的权限，在本安全网络文件系统中文件的个人共享是通过设置个人解密控制策略实现的，每一个拥有解密加密文件权限的个人共享用户都有一条个人解密控制策略与之对应，因此添加和删除一个个人共享用户只需添加和删除一条对应的个人解密控制策略即可，下面将详细介绍个人共享解密控制策略的结构以及个人共享的实现原理。

4.1.1 个人解密控制策略结构

本文中个人解密控制策略可分为管理用户解密控制策略和普通用户解密控制策略，管理用户不仅可以解密和操作加密文件还可以添加和删除普通用户，普通用户则只有解密和操作加密文件的权限，一条普通用户的解密控制策略由用户身份标识和用身份标识对加密文件的随机对称密钥进行 IBE 加密后的密钥数据组成，结构如图 4-1 所示。

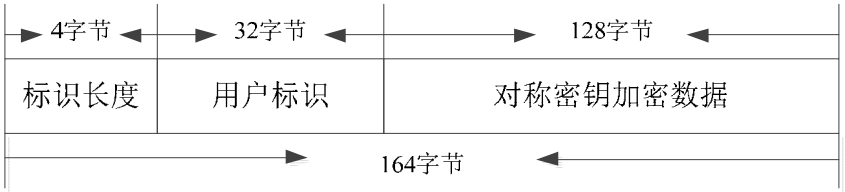

图 4-1 普通用户解密控制策略存储结构

管理用户解密控制策略在普通用户解密控制策略的基础上增加了管理信

息，该信息同样使用管理用户标识进行了 IBE 加密，由于个人共享用户较少，管理用户一般只有一个，即文件的创建者或上传者，其余的管理用户可在群组解密控制策略中设置，管理用户解密控制策略的结构如图 4-2 所示。

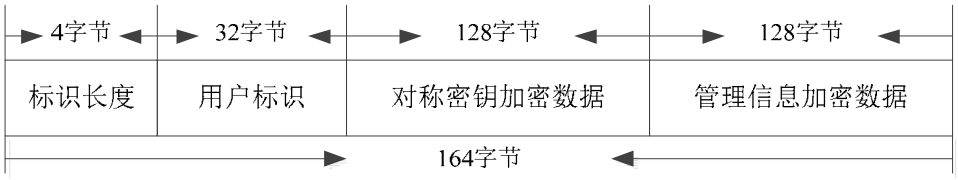

图 4-2 管理用户解密控制策略结构

在上章中提到个人解密控制策略存储的位置为文件加密信息的加密标识之后的 2032 个字节，因此整个个人解密控制策略在文件加密信息中的存储结构如图 4-3 所示。

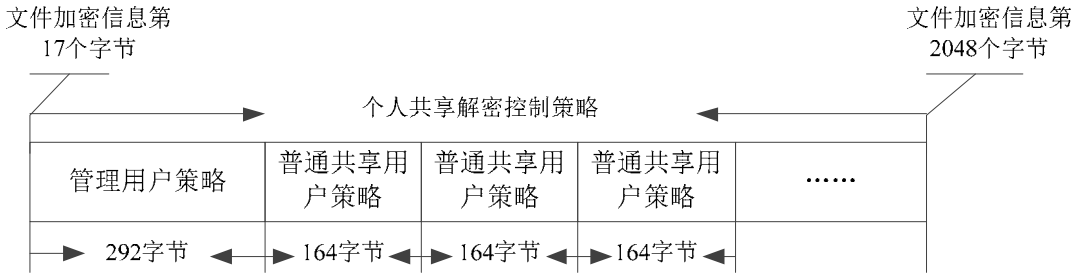

图 4-3 个人解密控制策略存储结构

4.1.2 个人共享解密原理

用户要解密加密文件必须得到加密文件所用的随机对称密钥，对于个人共享用户来说，随机对称密钥需要从个人解密控制策略中获得，因此找到属于当前用户的个人解密控制策略（管理员策略或普通用户策略）是关键，在个人解密控制策略中可能存放多条个人解密策略，如果使用用户私钥尝试对每一条解密控制策略中的对称密钥加密数据进行解密，将多次调用解密函数并判断解密是否成功，效率十分低下，因此我们采用的方式是先通过用户标识来寻找属于当前用户的个人解密控制策略，然后再进行解密获得随机对称密钥，个人共享用户获得随机对称密钥的过程如图 4-4 所示。

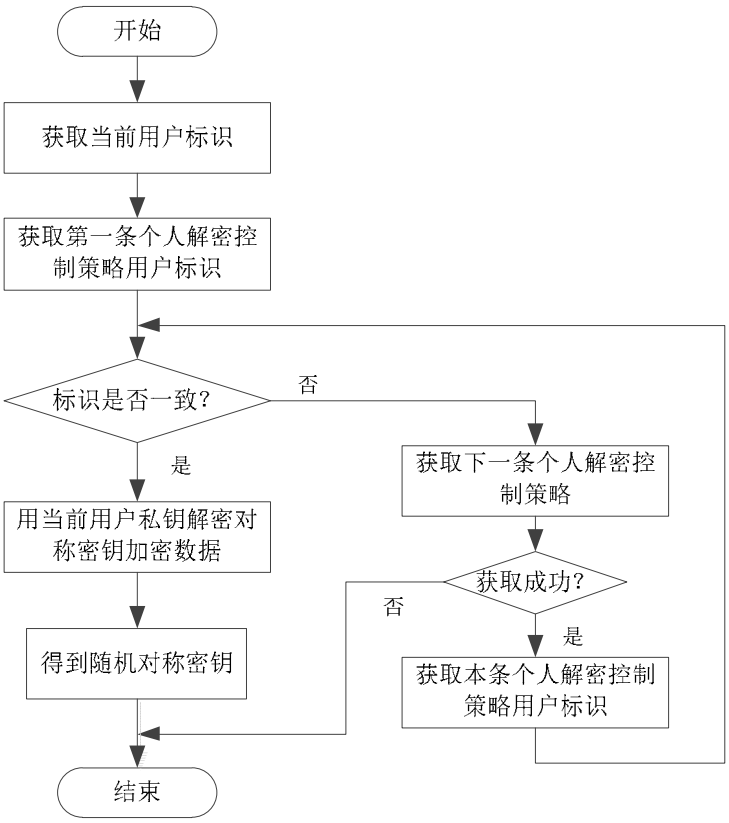

图 4-4 个人共享用户获取随机对称密钥过程

由上述过程可以看到，解密对称密钥加密数据的操作最多只执行了一次，通过标识先找到属于当前用户的个人解密策略再用私钥解密去对称密钥加密数据的方式比用私钥去逐个解密每条个人解密控制策略中的对称密钥加密数据的方式有效很多。如果用户通过上述过程得到了随机对称密钥之后，就能对加密的文件数据进行解密，解密文件的过程已在上章中讲到，这里不再赘述，如果用户通过上述过程没有获得随机对称密钥，则他不是个人共享用户，这时用户可以转到群组共享的方式解密，群组共享的实现原理将在下节中说明。

4.1.3 个人共享用户的管理

对于一个已经被加密并且设置了文件解密控制策略的文件，如果需要对其他个人用户进行授权共享，则需要为该个人用户增加一条新的个人解密控制策略，用户添加个人解密控制策略的步骤如下：

第一步：连接服务器获取文件加密信息，并从中取出个人解密控制策略；

第二步：获取当前用户标识，判断当前用户是否与管理用户标识一致，如

果一致转到第三步，否则提示用户没有添加策略权限，结束本次操作；

第三步：用当前用户标识解密管理用户策略中的管理信息加密数据，通过解密得到管理信息进一步判断当前用户是否为管理用户，若是管理用户转到第四步，否则提示用户没有添加策略权限，结束本次操作；

第四步：用当前用户标识解密管理策略中的对称密钥加密数据得到随机对称密钥，用需要添加的个人用户标识对随机对称密钥进行 IBE 加密得到一份新的对称密钥加密数据，并与新用户标识一起生成一条普通用户解密控制策略；

第五步：将新生成的普通用户解密控制策略添加到个人加密控制策略中并存放到文件加密信息中；

第六步：对文件加密信息的前 4080 个字节重新进行哈希，将得到的新的哈希值存放到文件加密信息的最后 16 字节，最后将新的文件加密信息覆盖写入到加密文件的头部，结束本次添加操作。

在上述过程中，第三步中需要对管理用户身份进行再次确认是为了防止管理用户策略被篡改，如用普通用户策略对管理用户策略进行了替换，这也是对管理用户设置管理信息并加密的意义所在。如果需要添加多个个人共享用户只需在第四步中用每个需要添加的个人共享用户标识分别对随机对称密钥加密得到多份新的对称密钥数据，并生成多条新的普通用户解密控制策略，一起添加到个人解密控制策略中即可。

删除个人共享用户的原理与添加类似，首先需要判断当前用户是否是管理用户，判断是管理用户后找到与要删除用户标识对应的普通用户解密控制策略，将其从个人解密控制策略中删除即可，同时删除个人共享用户后同样需要对文件加密信的前 4080 个字节进行重新哈希。这里还需要时说明的是，除了个人共享的管理用户，群组共享中具有管理策略权限的用户组的用户同样可以添加和删除个人共享用户，管理权限在个人共享和群组共享上是统一的，群组共享的管理权限判断将在下节中说明。

4.2 群组共享的原理与实现

文件的个人共享只适合共享人数比较少的情况，因为在个人共享中，每一个个人共享用户都需要与一条个人解密控制策略相对应，共享人数越多，生成的个人解密控制策略就越多，把它们全部存放在文件加密信息中是不现实的，因此本文针对大量共享用户的情况设计了文件的群组共享，群组共享是通过设

置群组解密控制策略实现的，下面介绍群组解密控制策略的结构和群组解密的实现原理。

4.2.1 群组解密控制策略结构

实现文件的群组共享，首先，需要对用户进行分组，而用户分组的依据多种多样，可以根据用户的级别分组，可以根据用户所属的部门分组，可以根据用户的角色分组等等；其次，与个人共享指定特定的用户不同，群组共享用户多而广，让所用群组共享用户都有文件的所有操作权限是不科学的，因此，对于不同的用户组应设置不同的用户组权限。

针对上述分析，一条群组解密策略应包括群组名和群组权限，其结构本文中用一个 8 字节长的字符串来表示群组名，用 4 个字节的数据位来表示权限信息，4 个字节总共 32 位最多可以表示 32 种不同权限的组合，本文中几种群组权限的宏定义如下：

```
#define    FILE_DOWNLOAD_POWER          0x00000001 //文件下载权限
#define    FILE_DECRYPT_POWER           0x00000002 //文件解密权限
#define    FILE_MODIFY_POWER            0x00000004 //文件修改权限
#define    FILE_DELETE_POWER            0x00000008 //文件删除权限
#define    FILE_RENAME_POWER            0x00000010 //文件重命名权限
#define    STRATEGY_MANAGE_POWER        0x00000020 //策略管理权限
```

一条或多条群组解密策略构成了群组解密策略的明文部分，其结构如图 4-5 所示。

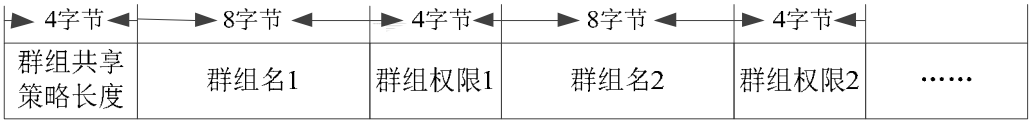

图 4-5 群组解密策略的明文结构

直接将群组解密策略的明文放在文件加密信息中是很不安全的，因为明文策略很容易被篡改使得没有权限的用户组非法获得文件的操作权限，因此就需要对群组解密策略进行加密，加密方式是用加密文件所用的随机对称密钥对群组解密策略明文数据使用 AES 加密，加密后就得到了群组解密策略的密文部分。在群组共享中，对随机对称密钥加密使用的是群组共享公钥，它由一个群组共享标识生成，用户只有公钥没有私钥，因此随机对称密钥只能通过解密服务器

解密获得。

群组解密策略的明文部分、群组解密策略的密文部分以及由群组共享公钥加密的对称密钥数据共同构成了完整的群组解密控制策略，之所以保留群组解密策略的明文部分是为了方便用户查看，对群组权限的判断仍以群组解密策略的密文部分为依据，群组解密控制策略存放的位置为从文件加密信息的第 2049 个字节开始的 2032 个字节，整个群组解密控制策略在文件加密信息中的存储结构如图 4-6 所示。

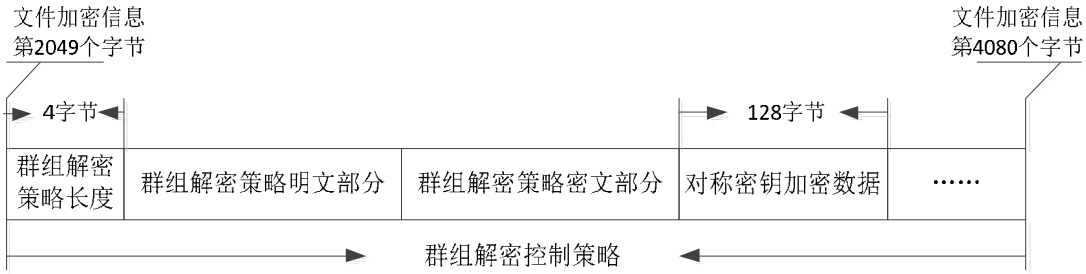

图 4-6 群组解密控制策略存储结构

4.2.2 群组共享解密原理

本文中用户所属的群组信息与用户帐号、用户密码一同存放在身份管理系统中，用户通过群组解密时需要连接解密服务器，由解密服务器访问身份管理系统确定用户所属的群组，并根据群组解密控制策略判断用户是否具有解密权限，用户使用群组解密的过程如下：

第一步：获取文件加密信息中的群组解密控制策略，将群组解密控制策略与用户的账号信息一同发送到解密服务器；

第二步：解密服务器访问用户身份管理系统并根据用户的账号信息确定用户所属的群组；

第三步：解密服务器用群组共享私钥解密群组解密控制策略中的对称密钥加密数据，得到随机对称密钥，并用随机对称密钥解密群组解密策略的密文部分，得到作为判断依据的群组解密策略明文部分；

第四步：根据用户所属的群组在群组解密策略的明文部分中查找有无属于当前用户的一条群组解密策略，若有则转到第五步，否则通知用户没有解密权限，结束本次操作；

第五步：根据查找到的这条群组解密策略中保存的权限信息判断用户是否

具有解密权限，若有则转到第六步，否则通知用户没有解密权限，结束本次操作；

第六步：解密服务器将随机对称密钥用当前用户的公钥（由用户标识生成）进行加密，并把加密后的密钥数据发送给当前用户；

第七步：用户用自己的私钥解密由解密服务器发送过来的密钥数据得到随机对称密钥；

第八步：用户使用随机对称密钥对加密的文件数据进行解密。

在上述过程中，查找有无属于当前用户的一条群组解密策略时依据的是群组解密策略的密文部分，文件加密信息中存储的群组解密策略的明文部分在群组解密时不起作用，当解密服务器判定当前用户具有解密权限后，再次对随机对称密钥进行加密的原因是为了数据传输的安全，如果随机对称密钥以明文数据发送，一旦在传输过程中被他人截获，就有可能造成文件信息的泄露。

个人解密与群组解密的本质都是从文件加密信息中获得随机对称密钥，从而对加密的文件数据进行解密得到明文数据，在实际应用中，个人解密和群组解密是相结合的，判断一个用户是否具有解密权限首先判断其是否为个人共享用户，如果不是再判断其是否是具有解密权限的群组共享用户，提高了效率也防止了权限冲突。

4.2.3 群组共享用户的管理

群组共享用户的管理包括群组用户的添加和删除、原有群组用户权限的修改和更新，管理群组共享用户同样需要用户拥有管理权限，上节中讲过管理权限在个人共享和群组共享中是统一的，个人共享管理用户同样具有管理群组共享用户的权限，关于个人共享管理用户的判断已经在上文中讲述，这里不再赘述，群组解密管理用户的判断过程与群组解密的过程类似，同样需要通过解密服务器实现，具体步骤如下：

第一步：获取文件加密信息的中的群组解密策略，将群组解密策略与用户的账号信息一同发送到解密服务器；

第二步：解密服务器访问用户身份管理系统并根据用户的账号信息确定用户所属的群组；

第三步：解密服务器用群组共享公钥解密群组解密控制策略中的对称密钥加密数据，得到随机对称密钥，并用随机对称密钥解密群组共享策略的密文部

分，得到作为判断依据的群组解密策略明文部分；

第四步：根据用户所属的群组在群组解密策略的明文部分中查找有无属于当前用户的一条群组解密策略，若有则转到第五步，否则用户不是群组管理用户，结束本次操作；

第五步：根据查找到的这条群组共享策略中保存的权限信息判断用户是否具有管理权限，若有则当前用户为群组解密管理用户，并将随机对称密钥加密后发送给用户，否则用户不是群组管理用户，结束本次操作。

当确定用户为管理用户后就能管理群组共享用户了，用户添加一个共享群过程如下：

（1）选择一个需要添加的用户群组，并为需要添加的新用户群组设置群组权限；

（2）由新添加的用户群组名和群组权限生成一条新的群组解密策略；

（3）将新生成的群组共享策略添加到原来的群组解密策略中形成新的群组共享策略的明文部分；

（4）用随机对称密钥对新的群组共享策略的明文部分进行 AES 加密得到密文部分；

（5）用群组共享公钥对随机对称密钥加密并与群组解密策略的明文部分和密文部分生成新的群组解密控制策略。

删除和修改群组共享用户的过程类似，首先需要在群组共享策略中找到对应的那条解密策略，如果是删除群组用户，则将该条解密策略删除，并将之后的每条策略向前移动一条策略的长度（12 字节长），如果是修改权限，则为该用户组设置新的权限并对原策略作替换，完成删除或修改后生成新的解密控制策略，生成方法与添加操作相同。

不管是群组用户的添加、删除还是对权限进行修改，都会改变原来的解密控制策略，因此完成修改操作后都需要对文件加密信息的前 4080 个字节重新进行哈希并替换原来的哈希值，最后将新的文件加密信息覆盖写入到加密文件的头部。

4.3 解密控制策略的保存与继承

4.3.1 解密控制策略的保存

用户在上传文件时如果每次都要弹出用户交互界面然后设置解密控制策略是一件非常繁琐的事情，尤其是当上传多个文件并且需要设置相同的解密控制策略时，逐个文件设置十分耗时，影响用户体验，因此本文采取的办法是针对文件目录设置解密控制策略，将个人共享用户和群组共享及其权限信息保存起来，用户在上传文件时自动去获取当前目录下的共享信息，再与本次使用的随机对称密钥一起生成解密控制策略保存到文件加密信息中，从而避免了在上传文件时反复设置的麻烦，对于个别需要设置不同解密控制策略的文件可通过右键菜单单独修改。

文件目录下的共享用户信息使用一个名为“ _____@@@@@@_____.ini ”的配置文件来保存，将配置文件命名为这个特殊名字是为了防止与目录下的其他普通加密文件重名，对于个人共享来说，只需要保存每个个人共享用户的用户标识即可，对于群组共享来说，则需要保存共享群组名和群组权限，文件共享信息的存储结构如图 4-7 所示。

个人共享信息标志	个人共享信息长度	个人用户标识1	个人用户标识2	……	群组共享信息标志	群组共享信息长度	群组信息1	群组信息2	……

图 4-7 文件共享信息存储结构

对于普通用户来说，保存共享信息的配置文件是隐藏的，网络文件系统在显示文件列表时会将此文件过滤掉，用户无法操作这个配置文件，但是对于文件服务器的管理人员来说这个配置文件是可见的，为了防止文件服务器管理人员非法篡改共享信息，需要对共享信息进行加密，加密方式是用一个公共标识对共享信息采用 IBE 加密，用户在获取共享信息时需要经过解密服务器的身份认证，通过身份认证后由解密服务器对加密的共享信息进行解密，并返回给用户。

4.3.2 解密控制策略的继承

用户上传文件到一个已经设置了解密控制策略的文件目录时可获取得到文件目录的共享信息从而对文件设置解密控制策略，但是如果用户上传文件到没

有设置解密控制策略的文件目录或者是新创建的文件目录如何设置文件的解密控制策略就是一个问题，这个问题在本文中是通过文件目录的策略继承来解决的。

用户在上传文件时如果当前文件目录下没有找到针对该文件目录的文件共享信息则在该文件目录的上级目录中寻找，如果上级目录中依旧没有共享信息，则再往它的上级目录中寻找，依次类推，直到找到文件共享信息或者到达文件根目录为止，如果最终找到了文件共享信息，则将保存文件共享信息的配置文件复制到每个查找路径上的文件目录下；如果到达文件的根目录依旧没有找到文件共享信息，则弹出用户交互界面，让用户设置个人共享用户和群组共享用户，将共享信息保存到配置文件中，并将其复制到每个查找路径上的文件目录下。上述过程说明了本文的策略继承原理，即文件目录的解密控制策略继承自它的上级目录。

一个文件目录的解密控制策略继承自的它的上级文件目录，但它们的解密控制策略不一定是完全一致的，文件子目录的策略可以在继承上级目录策略的基础上添加，即能够向下扩展，但是不能够向下缩减，从用户解密的角度来说就是能够解密上级文件目录中文件的用户一定能够解密它的子目录中的文件，但是能够解密子目录中的文件的用户不一定能够解密上级目录中的文件。文件目录解密控制策略的继承关系如图 4-8 所示。

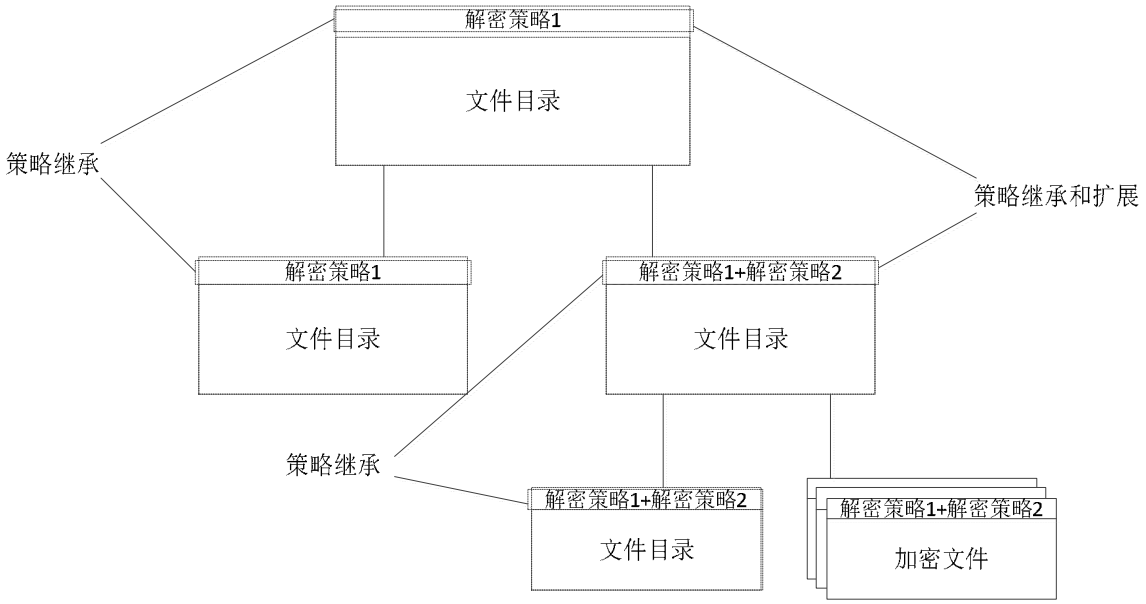

图 4-8 文件目录的策略继承关系

4.4 管理工具的设计与实现

为了方便管理用户对文件个人共享用户和群组共享用户的管理，本文还设计和实现了策略管理工具，管理用户可通过右键虚拟文件盘内的文件或文件目录在弹出菜单中选择“管理文件策略”或“管理目录策略”选项进入策略管理界面，在管理界面可以查看和管理当前文件或文件目录的共享用户。

4.4.1 文件策略的管理

本文的策略管理工具是基于 MFC 框架设计与实现的，分为文件的策略管理和文件目录的策略管理，文件策略管理的主界面如图 4-9 所示。

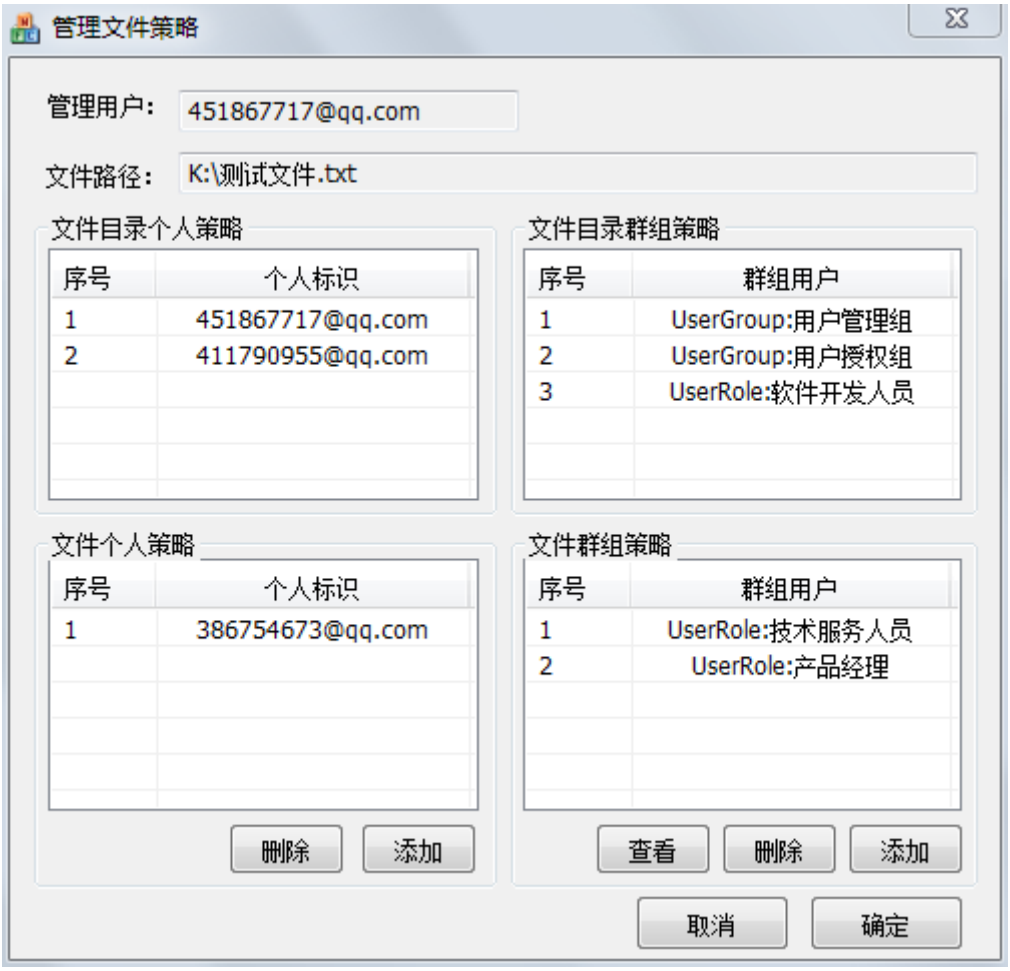

图 4-9 文件策略管理主界面

当用户请求管理文件解密控制策略时文件管理界面的上方显示有当前管理

用户的身份标识和管理文件的所在路径，对于文件策略管理来说，它包括两部分，当前文件所在目录的策略和文件的扩展策略，为了保证策略的继承性和扩展性，此时管理用户只能修改文件的扩展策略，而不能修改目录策略。添加个人共享用户只需添加用户标识即可，添加操作如图 4-10 所示。

图 4-10 添加个人共享用户标识

添加群组共享用户需要选定新增用户组，并设置群组权限，添加操作如图 4-11 所示。

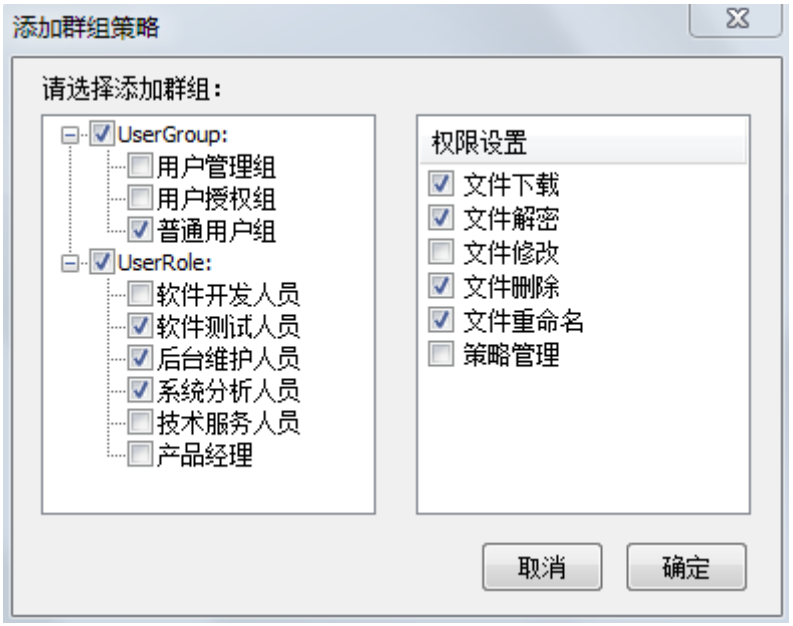

图 4-11 添加群组共享用户及权限

对于共享用户的删除，只需选择需要删除的用户标识或需要删除的用户群组点击对应的“ 删除 ”按钮即可实现，完成策略管理操作后点击主界面的“ 确定 ”按钮，管理工具会生成新的解密控制策略存储到文件加密信息中并覆盖写入到加密文件头部，新增加的共享用户就可以对该文件进行相关的文件操作，而被删除的共享用户则失去了对该文件的操作权限。

4.4.2 文件目录策略的管理

文件目录策略管理的主界面如图 4-12 所示。

图 4-12 文件目录策略管理主界面

文件目录管理界面的上方同样显示有当前管理用户的身份标识和当前文件目录路径，与文件策略管理界面不同的是它只有文件目录策略，管理用户此时可以修改目录策略，添加、删除策略的操作与添加、删除文件策略的操作相同，需要注意的是，如果为目录策略添加了共享用户或者添加了共享用户权限，则需要对该目录下的所有文件（包括所有子目录及子目录下的文件）添加该共享用户或者共享用户权限，以保证目录策略的继承性，删除共享用户操作则不影响该目录下的其他文件和目录。

4.4.3 右键菜单的实现

本文中策略管理工具是用户通过右键菜单弹出的，在右键菜单中添加管理文件策略选项和管理目录策略选项是通过添加注册表实现的，添加管理策略选

项具体是在注册表 HKEY_CLASSES_ROOT*\shell 分支下新建项“管理文件策略”，接着在“管理文件策略”下新建 command 项，并将 command 主键的键值修改为“C:\FileStrategyControl\Debug\FileStrategyControl.exe "%1" ”，添加管理目录策略选项具体是在 HKEY_CLASSES_ROOT\Folder\shell 分支下新建项“管理目录策略”，接着在“管理目录策略”下新建 command 项，并将 command 主键的键值修改为“C:\FolderStrategyControl\Debug\FolderStrategyControl.exe "%1" ”。

当用户对虚拟文件盘的文件通过右键菜单点击管理文件策略选项时，启动策略管理程序 FileStrategyControl.exe，管理程序首先读取用户个人身份信息，并通过 CommandLineToArgv 函数获得文件路径，连接文件服务器获取对应文件的文件加密信息，用户身份信息和文件加密信息判断用户是否为管理用户，判断原理上文中已经讲述，如果当前用户是管理用户则弹出文件策略管理界面，否则提示用户没有管理权限。同样的，当用户对虚拟文件盘的文件通过右键菜单点击管理目录策略选项时，启动策略管理程序 FolderStrategyControl.exe，接下来的处理过程与文件策略管理程序一致，如果当前用户为管理用户则弹出目录策略管理界面，否则提示用户没有管理权限。

4.5 本章小结

本章围绕文件个人共享和群组共享的实现展开，首先设计了个人解密控制策略和群组解密控制策略的结构，在此基础上说明了文件的个人共享和群组共享的实现原理，为了用户设置方便，将文件目录解密控制策略保存在目录下被加密的配置文件中，并设计了目录策略的继承机制，最后设计了策略管理工具并说明了如何通过策略管理工具修改和更新解密控制策略。

第 5 章　应用测试与分析

本章将完成对本系统实现功能的测试与分析，主要包括用户登录测试，文件上传写入时的加密测试与分析，文件下载和读取时的个人解密测试与群组解密测试，策略管理测试，包括添加和删除个人共享用户、群组共享用户测试，以及文件目录策略继承测试，最后对系统安全性进行分析。

5.1 系统测试环境

本文所述的基于 FUSE 的安全网络文件系统使用两台用户计算机分别作为客户机和服务器进行系统测试，由于本系统是针对 Windows 系统研究和设计的，因此操作系统选择的是目前使用最为广泛的 Windows 7 系统，在服务器上安装网络通信程序，文件服务器，解密服务器以及身份管理系统，客户机除了主机以外还安装了 3 个同样安装了 Windows 7 系统的虚拟机作为另外 3 个客户机，每个客户机都安装了 dokan 虚拟文件系统驱动，文件客户端应用程序，策略管理程序以及 IBE 密码模块，完成以上配置以后，整个系统的测试环境就搭建完成了。

5.2 用户登录测试

本系统中，用户对网络文件进行操作时，首先需要进行用户登录操作，将网络文件服务器上的文件目录挂载到本地计算机，用户登录需要输入用户账号和密码，之后用户对共享文件所有操作的权限，都将根据登录账户的用户信息进行判断，用户登录界面如图 5-1 所示。

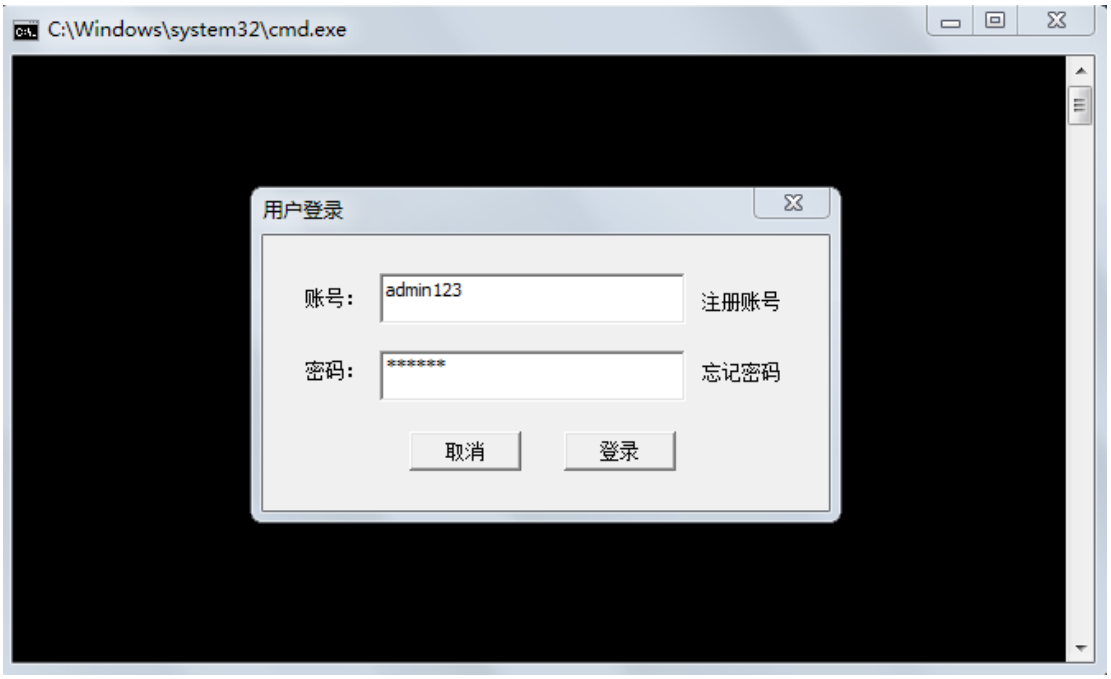

图 5-1 用户登录界面

用户通过服务器的身份认证后，便可成功登录，登录以后网络文件服务器的共享目录会在用户计算机中映射出一个虚拟文件盘，同时网络文件共享目录下的所有文件及文件目录都会映射为虚拟文件盘内的文件和文件目录，但它们不是真实存在的，只是通过文件客户端程序将它们显示出来，实际不会占用本地磁盘空间，虚拟文件盘和虚拟文件盘的属性如图 5-2 所示。

图 5-2 虚拟文件盘及属性

5.3 文件上传及加密测试

文件的上传是一个网络文件系统最基本的功能，在本系统中，用户将本地文件拖入或复制到虚拟文件盘或虚拟文件盘中的文件目录中，文件就会被自动上传到文件服务器上的对应文件目录中，同时上传前未加密的文件，会通过文件系统客户端程序添加该文件的解密控制策略，并对文件数据进行加密，在文件服务器中存放的是加密后的密文文件，这里通过上传一个 txt 文件测试系统的上传功能及文件自动加密功能，文件上传测试过程如图 5-3 所示。

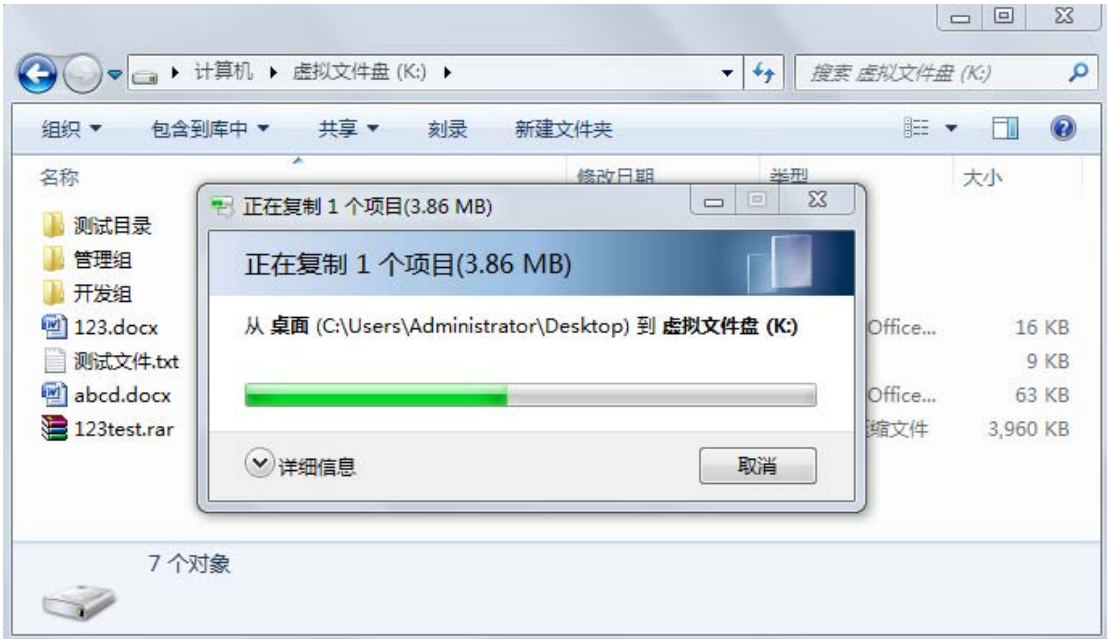

图 5-3 文件上传测试

文件上传前文件数据和文件上传后文件服务器中该文件的文件数据对比如图 5-4 和 5-5 所示。

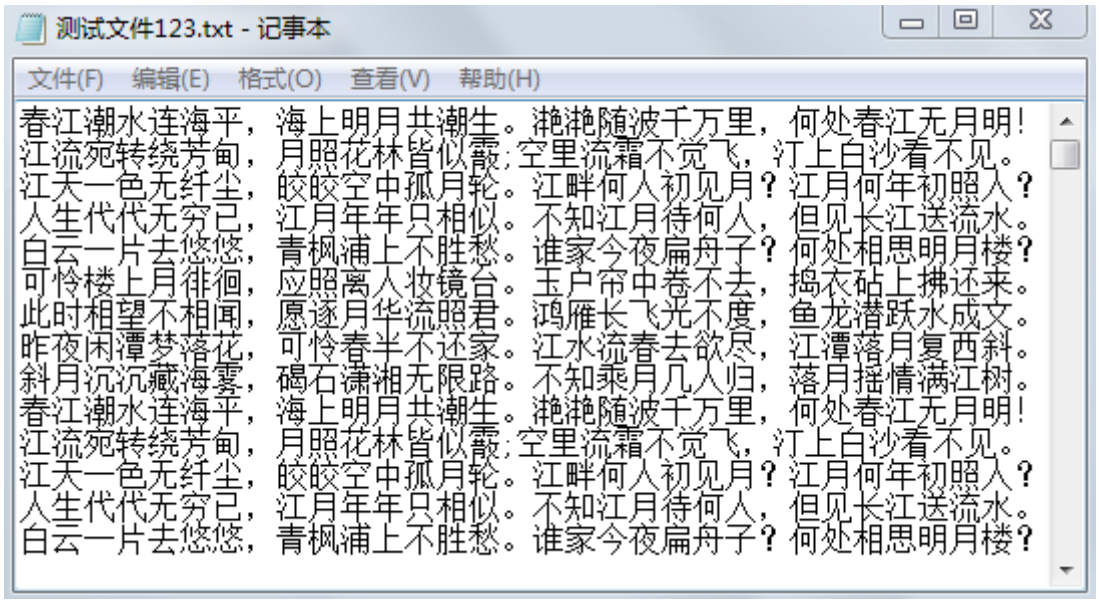

图 5-4 加密前文件数据

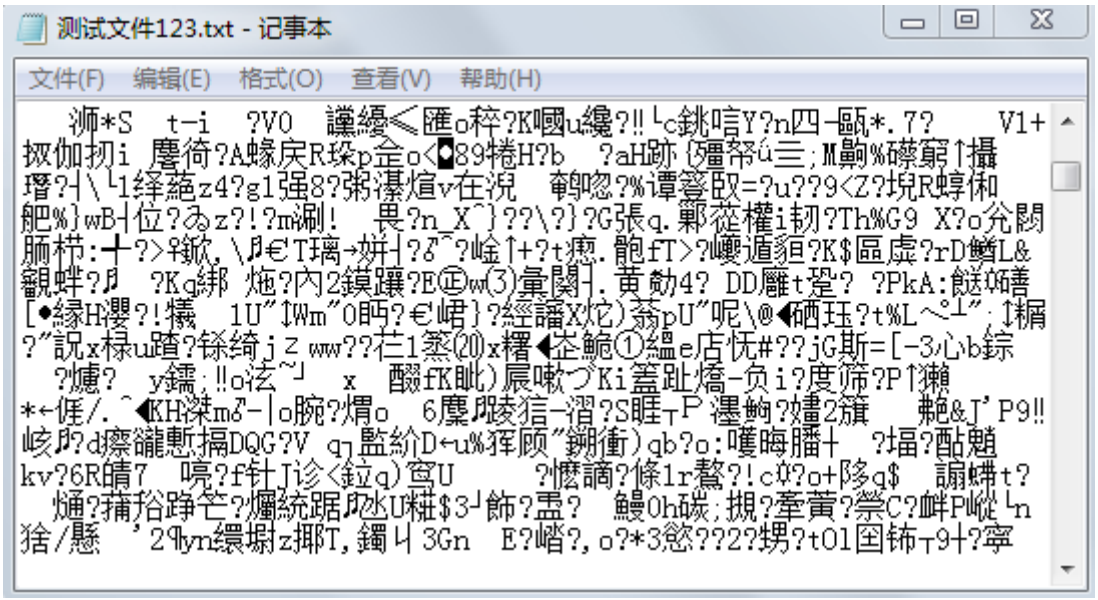

图 5-5 加密后文件数据

通过测试，可以发现，本系统实现了文件的上传及文件加密功能。

5.4 文件下载及解密测试

文件的下载在本系统中只需将文件从虚拟文件盘或虚拟文件盘中的文件目录中复制或拖拽到用户本地计算机的文件目录下即可实现，这里需要重点测试的是文件的解密功能，使用上文所述的测试环境，本文用主机登录一个 word 文件的管理用户，三个虚拟机同时登录三个账户分别模拟三种用户，虚拟机 A 上登录的为个人共享用户，虚拟机 B 上登录的为群组共享用户，虚拟机 C 上登录的为非共享用户，三个用户分别对该 word 文件进行下载打开操作和双击文件打开操作，测试结果如表 5-1 所示。

表 5-1 文件的个人解密与群组解密测试

测试功能	测试用户	测试结果
加密文件的个人解密与群组解密	虚拟机 A 上的个人共享用户	下载正常打开，双击正常打开
	虚拟机 B 上的群组共享用户	下载正常打开，双击正常打开
	虚拟机 C 上的非共享用户	下载、双击均无法正常打开

通过测试可以发现，虚拟机 A 上的个人共享用户和虚拟机 B 上的个人共享用户 B 均可以正常打开文件，如图 5-6 所示，虚拟机 B 上的非共享用户打开时提示用户没有解密权限，无法正常打开文件，如图 5-7 所示。

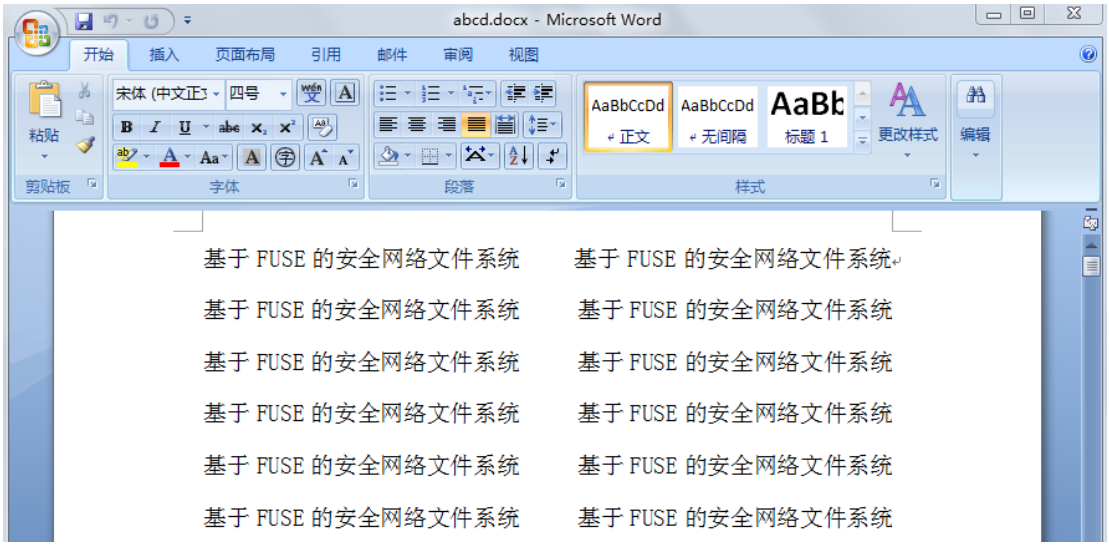

图 5-6 共享用户正常解密加密文件

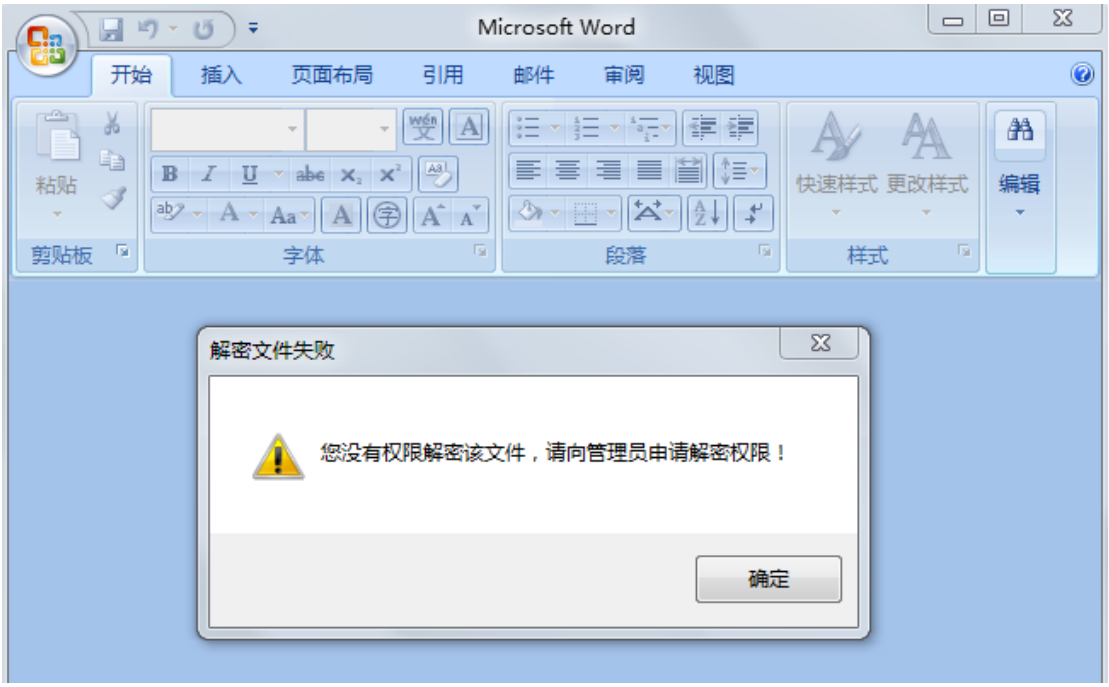

图 5-7 非共享用户无法解密加密文件

5.5 策略管理测试

对于已经设置文件解密控制策略的文件，管理用户可以通过右键菜单对文件解密控制策略进行管理，我们沿用上述的测试条件，使用主机上的管理用户

对上述 word 文件进行策略管理，并对三个虚拟机上的用户重新进行文件双击打开操作来测试策略管理功能，测试结果如表 5-2 所示。

表 5-2 策略管理功能测试

测试功能	测试操作	测试结果
文件解密控制策略管理	将虚拟机 A 上的用户修改为群组共享用户	文件正常打开
	将虚拟机 A 上的用户修改为非共享用户	文件无法正常打开
	将虚拟机 B 上的用户修改为个人共享用户	文件正常打开
	将虚拟机 B 上的用户修改为非共享用户	文件无法正常打开
	将虚拟机 C 上的用户修改为个人共享用户	文件正常打开
	将虚拟机 C 上的用户修改为群组共享用户	文件正常打开

通过测试可以发现，我们实现了文件的策略管理功能，当文件策略被修改后，用户按照新设置的解密控制策略对文件进行解密操作。

5.6 文件目录策略的继承测试

一个文件或一个文件的解密控制策略继承自它的上级文件目录，为了测试文件目录的继承功能，本文首先对虚拟文件盘内的一个测试目录设置解密控制策略，如图 5-8 所示。

图 5-8 测试目录策略

接着将包含一个测试文件的测试子目录上传到该测试目录下，然后查看测试子目录和测试子目录下的测试文件的解密控制策略，如图 5-9 和 5-10 所示。

图 5-9 测试子目录策略

图 5-10 测试子目录下的测试文件策略

通过对比解密控制策略可以发现，测试目录的解密控制策略得到了继承，同时子目录及子文件的策略可以在上级目录的基础上进行扩展。

5.7 系统安全性分析

在本系统中，文件的加密采用对称加密和非对称加密相结合的方式，并且在上传到网络文件服务器之前完成加密，从而提高了文件存储的安全性，同时由用户设置解密控制策略对文件实行访问控制还防止了系统的管理员对文件的非授权访问，有效控制了机密信息、隐私文件的泄露，解除了用户长期以来的担忧。

在个人解密控制策略的设计上，管理用户策略相比普通用户策略增加了一个用管理用户标识加密的管理信息密钥数据，有效防止了管理用户策略被篡改替换的情况，避免了管理权限被滥用的情况。在群组解密控制策略的设计上，对群组策略的明文部分进行了加密，用户的权限判断以策略的密文部分为依据，同样防止了群组解密控制策略被篡改的情况。同时，对于拥有群组解密权限的用户，解密服务器在返回对称密钥时先用用户标识对其进行加密再返回，当用户接收到加密的随机对称密钥后再解密，有效避免了密钥在传输过程中被他人非法截获并解密文件造成文件信息泄露的问题。

5.8 本章小结

本章对本文设计的基于 FUSE 的安全网络文件系统所实现的各个功能进行了测试，对测试的结果进行分析并得出结论，并且说明和分析了系统整体的安最后的试结果表明系统总体实现的功能和完成情况符合预期设计目标。

第 6 章　总结与展望

6.1 全文工作总结

本文提出了一种基于 FUSE 的安全网络文件系统的技术方案, 针对目前云存储系统普遍存在的安全问题，提出了一种用户端设置解密控制策略和文件加密相结合的解决方案，能够防止来自外部和内部对共享文件的非授权访问，特别是防止系统的运维者对共享文件的非授权访问，同时利用 FUSE 文件系统技术，将网络文件服务器上的共享文件目录虚拟为用户本地计算机的一个虚拟文件盘，方便了用户对网络文件的操作。本文完成的主要工作如下：

（1）设计与实现了基于 FUSE 的网络文件系统，完成了服务器共享文件目录到本地虚拟文件盘的映射，将用户对虚拟文件盘内的文件或文件目录的操作转化为对网络文件服务器上共享文件目录内对应文件或文件目录的操作。

（2）实现了文件客户端程序和网络服务程序的设计，通过通信协议的设计，完成了文件客户端程序各个操作函数与文件服务器的数据通信，实现了用户登录、退出，文件的创建、上传、下载、删除等操作，并采用了断线检测机制，确保客户端与服务器的有效连接。

（3）实现了文件的加密和解密，当用户上传文件或写入文件时，客户端程序生成一个随机对称密钥对明文文件数据进行加密并上传到网络文件服务器进行保存，当拥有权限的用户下载文件或读取文件时，客户端程序从网络文件服务器获取密文文件数据并进行解密，将明文文件数据返回给用户。

（4）实现了文件的个人共享和群组共享，用户在创建或者上传文件时对文件设置解密控制策略，针对个人用户，满足个人解密控制策略的用户可以访问和操作共享文件，针对群组用户，满足群组解密控制策略的用户可以根据群组权限对共享文件执行相应操作。

（5）设计与实现了策略管理工具，管理用户可以通过右键菜单选项对共享文件或共享文件目录的解密控制策略进行查看、添加、删除和修改，实现对个人共享用户和群组共享用户的管理。

本论文的创新点在于利用 FUSE 文件系统技术解决了传统文件云存储系统

需要通过专用客户端操作网络文件的问题，让用户操作网络文件更加方便。同时，对文件采用对称加密和 IBE 加密相结合并在客户端设置解密控制策略的方案解决了传统云存储客户端普遍存在的文件共享不够安全的问题，特别是防止了系统的运维者对文件的非授权访问。

6.2 未来工作展望

本文设计与实现了基于 FUSE 的安全网络文件系统, 经过功能和性能测试实现了设计目标。由于时间和个人研究水平限制，本系统依然存在未完善之处，需要在以下几点作进一步研究：

（1）在对文件读取缓存的处理上做的比较简单并且没有设计写入缓存，当进行不规则读取和大量数据读取时缓存的作用很小，需要设置分块缓存并且实现缓存的读写同步。

（2）用户每次上传文件从文件目录获取共享信息都需要通过解密服务器解密，效率较为低下，需采取一种更为高效且安全的方式。

（3）策略管理工具目前只能对单个文件或单个目录进行操作，还需设计一个后台管理工具，对多文件、多目录策略同时进行管理以方便管理用户操作。

致谢

光阴似箭，岁月如梭，一转眼，三年的研究生生涯也即将结束了，在这三年中我成长了许多，也收获了许多，在此我要对所有关心过我、帮助过我的人表示衷心的感谢。

首先，我要感谢我的父母和家人，是他们一直在我的背后默默付出，支持着我，鼓励着我。

其次，我要感谢我的导师龙毅宏教授，龙老师在学术上兢兢业业，一丝不苟，为我们树立了良好的榜样。三年来，我参与了很多项目的研究，在遇到困难时龙老师总是耐心的跟我讲解原理、探讨方案，帮助我解决问题，在龙老师的悉心指导下，我的科研能力能力有了很大提升。同时本篇论文的完成也离不开龙老师的帮助，在论文的选题、方案的设计以及论文的修改上他都花费了大量的时间精力给予指导，在此，向龙老师致以深深的谢意。

最后，我要感谢实验室的小伙伴和寝室的室友，在日常学习和生活中他们给予了我很大的帮助，他们是郑李恒、李超、周旭、陈瑛、李凤利、陈慧、卢衍军、孙蒙、何翔、吴静琳，在此表示真挚的感谢。

与此同时，也由衷的感谢评审本文的各位专家教授，谢谢您们对本论文提出的宝贵意见。

参考文献

[1] 傅颖勋, 罗圣美, 舒继武. 安全云存储系统与关键技术综述[J]. 计算机研究与发展,2013,01:136-145.

[2] 朱学君等. 关于影响计算机网络安全因素及防止措施的探讨[J]. 信息通信, 2013,(1)：123-125.

[3] 龙毅宏, 唐志红. 一种安全文件共享系统[P]. 中国专利: CN103561034A, 2014.02.05.

[4] Ren S Q, Aung K. PPDS: Privacy Preserved Data Sharing Scheme for Cloud Storage[J]. International Journal of Advancements in Computing Technology, 2012, 4(16):493-499.

[5] Blanke W. Data loss prevention using an ephemeral key: IEEE, US 8499359 B1[P]. 2013.

[6] 李凤华, 苏锦, 史国振等. 访问控制模型研究进展及发展趋势[J]. 电子学报，2012, 40(4): 805-813.

[7] 张兵. 安全网络文件存储系统的客户端的研究与开发[D]. 武汉理工大学, 2014.

[8] 胡伟明. 云存储系统的设计与实现[D]. 中国地质大学（北京）,2013.

[9] 龙毅宏, 唐志红. 一种基于用户模式文件系统的安全网络文件系统[P]. 中国专利: CN103841113A, 2014.06.04.

[10] 王博，吴健. 一种网络文件安全存储系统的设计与实现[J]. 微型电脑应用,2009,08:36-38.

[11] Tribbey C. Apple Launches Its iCloud Movie Service[J]. Home Media Magazine, 2012.

[12] Kubaszewski L,Kaczmarczyk J,Nowakowski A. Management of scientific information with Google Drive.[J]. Polish orthopedics & traumatology,2013,78:213-217.

[13] Zhen Zhou,Shuyu Chen,Tao Ren,Tianshu Wu. File heat-based Self-adaptive Replica Consistency Strategy for Cloud Storage[J]. Journal of Computers,2014,9(8).

[14] 刘建毅,王枞,薛向东. 云存储安全分析[J]. 中兴通讯技术,2012,06:30-33.

[15] 张富政. 云存储安全技术的研究[D]. 长春理工大学,2014.

[16] 边根庆,高松,邵必林. 面向分散式存储的云存储安全架构[J]. 西安交通大学学报,2011,04:41-45.

[17] 涂山山. 云计算环境中访问控制的机制和关键技术研究[D]. 北京邮电大学,2014.

[18] Minaam D S A, Abdualkader H M, Hadhoud M M. Evaluating the Effects of Symmetric Cryptography Algorithms on Power Consumption for Different Data Types[J]. International Journal of Network Security, 2010, 11(2):78-87.

[19] Diaa Salama Abd Elminaam,Hatem Mohamed Abdual Kader,Mohiy Mohamed Hadhoud. Evaluating The Performance of Symmetric Encryption Algorithms[J]. International Journal of Network Security,2010,10(3).

[20] 张金辉, 郭晓彪, 符鑫. AES 加密算法分析及其在信息安全中的应用[J]. 信息网络安全,2011,05:31-33.

[21] 袁巍. AES 算法的设计原则与其密钥扩展算法的改进[D]. 吉林大学,2010.

[22] 虞淑瑶. 在线公钥系统及相关安全技术研究[D]. 中国科学院研究生院（计算技术研究所）,2005.

[23] 潘家超. 面向云存储共享文件的文件加密系统的研究与开发[D]. 武汉理工大学,2015.

[24] Yvo Desmedt, Jean-Jacques Quisquater. Public-key Systems Based on the Difficulty of Tampering(Is there a difference between DES and RSA?)[J]. Advances in Cryptology-CRYPTO'86 Lecture Notes in Computer Science Volume 263, 1987, 111-117.

[25] H. Tanaka. A realization scheme for the identity-based cryptosystem[C]//Proe of Advances in Cryptology-Crypto'87. [S.1.]: Springer-Verlag, 1987:341-349.

[26] 王宣丹. IBE 关键技术的研究及在安全电子邮件中应用[D]. 哈尔滨理工大学,2009.

[27] 刘光亚. 实时同步云存储客户端的设计与实现[D]. 大连理工大学,2014.

[28] 王妙娅. 国内外网盘搜索引擎分析与比较[J]. 大学图书馆学报, 2011, 29(3):71-75.

[29] Mendon A A, Schmidt A G, Sass R. A Hardware Filesystem Implementation with Multidisk Support[J]. International Journal of Reconfigurable Computing,2009,2009:13.

[30] 彭舰, 谢纲. 可扩展固件接口的 NTFS 文件系统驱动[J]. 计算机工程,2008,09:83-85.

[31] 张荣亮,余敏,余文斌. Linux 文件系统内核机制分析与研究[J]. 计算机与现代化,2007,12:14-17+21.

[32] Khashan O A, Zin A M, Sundararajan E A. Img FS: a transparent cryptography for stored images using a filesystem in userspace[J]. Frontiers of Information Technology & Electronic Engineering, 2015, 16(1):28-42.

[33] 魏东林,卢正鼎,董俊,聂岚. 在用户空间扩展 Linux 操作系统功能方法研究[J]. 华中科技大学学报(自然科学版),2002,07:42-44.

[34] 黄永胜. 基于 FUSE 的用户态文件系统的设计与实现[D]. 东北大学,2012.

[35] Beebe N L, Stacy S D, Stuckey D. Digital forensic implications of ZFS[J]. Digital Investigation the International Journal of Digital Forensics & Incident Response, 2009, 6:S99-S107.

[36] 段翰聪,王勇涛,李林. EDFUSE:一个基于异步事件驱动的 FUSE 用户级文件系统框架[J]. 计算机科学,2012,S1:389-391.

[37] Noronha R, Panda D K. IMCa: A High Performance Caching Front-End for GlusterFS on

InfiniBand[C]// International Conference on Parallel Processing. IEEE, 2008:462-469.

[38] Yuan Wang,Yongquan Lu,Chu Qiu,Pengdong Gao,Jintao Wang. Performance Evaluation of A Infiniband-based Lustre Parallel File System[J]. Procedia Environmental Sciences,2011,11(1):316-321.

[39] 吴宗坤. 基于 Fuse 的资源搜索文件系统设计与实现[D]. 华南理工大学,2011.

[40] 吴一民,刘伟安. 基于 Fuse 的用户态文件系统的设计[J]. 微计算机信息,2010,06:159-160+168.

[41] 李凡,刘学照,卢安,谢四江. WindowsNT 内核下文件系统过滤驱动程序开发[J]. 华中科技大学学报(自然科学版),2003,01:19-21.