武漢理工大學

（申请工学硕士学位论文）

基于 FUSE 的透明文件加密系统研究与实现

培 养 单 位： 信息工程学院

学 科 专 业： 信息与通信工程

研 究 生： 郑李恒

指 导 教 师： 龙毅宏 教授

2016 年 4 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 的透明文件加密系统研究与实现

英　文 Research and Implementation of Transparent File
题　目 Encryption System Based on FUSE

研究生姓名 郑李恒

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 4 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 刘泉

刘岚

2016 年 4 月

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，论文中不包含其它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它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。本人承诺所提交的学位论文（含电子学位论文）为答辩后经修改的最终定稿学位论文，并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、缩印或其它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

在保证企业数据安全方面，企业防止敏感信息泄露的要求是，工作人员可以在企业内自由的使用加密的数据文件，但无法在公司外使用。透明文件加密技术就是解决这个问题的一种重要的技术手段，所谓透明文件加密技术，就是在不影响用户使用习惯的基础上自动实现文件的加密与解密的技术。本文研究的主要是基于 FUSE(File System in User Space)的透明文件加密系统。

对于传统的基于文件过滤器驱动的透明文件加密系统，通常将访问数据的程序进程分为受信进程和非受信进程。当受信进程和非受信进程对一个已经加密的文件不断交替的访问时，文件加密系统需要不断的清理文件缓存中的缓存数据，以保证文件缓存中缓存的是正确数据（明文或密文），这种暴力清理缓存的方法大大降低了文件操作的效率，并且也存在一些兼容性问题。

本文针对传统透明文件加密系统在缓存处理方面的不足，提出了一种基于 FUSE 的透明文件加密系统。该系统将应用程序进程对安全目录中文件的操作重定向到 FUSE 文件系统所映射的虚拟磁盘中，将针对安全目录中文件的操作转化为 FUSE 用户空间进程对安全目录中文件操作的实现。

本文基于 FUSE 的透明文件加密系统分成三大模块进行开发，文件重定向过滤器模块，加密桥文件系统模块，系统安全加固模块。文件重定向过滤器模块，设计并实现了应用层安全目录的管理，内核层针对安全目录的文件重定向。加密桥文件系统模块，实现了进程的识别，实现了文件操作的处理，同时采用三种加密模式实现了文件的加解密，并针对文件读写效率问题添加了缓存机制。系统安全加固模块，利用进程监控和剪贴板监控实现针对安全目录文件的防复制防截屏，同时对进程进行了真实性验证，以及对配置文件进行了安全保护。

本论文的创新点在于以一种全新的思路借助一个起着桥梁作用的 FUSE 文件系统并采用 IBE(Identity Based Encryption)加密算法来实现文件的透明加解密，在确保文件系统缓存文件数据始终为密文的同时，将对安全目录中文件的 I/O 操作通过文件重定向转移到 FUSE 用户空间中进行实现，使得实现更加容易而且易于维护，而且通过对剪贴板的监控，防止了由于截屏、复制等操作而导致的数据泄露，同时对受信进程进行了真实性验证，实现了系统的安全加固。

关键词：数据安全，透明文件加密，FUSE，IBE

Abstract

To ensure enterprise data security, the requirement of preventing sensitive enterprise information disclosure is that staff can use the encrypted data file in the enterprise freely, but cannot be used outside the company. Transparent file encryption technology is a important technology to solve this problem, the so-called transparent file encryption technology, is a technology that implemented the encryption and decryption of files automatically without changing the user's habit . This thesis mainly research transparent file encryption system based on FUSE(File System in User Space).

Traditional transparent file encryption system based on file filter driver, usually divide the program processes that accessing the data into trusted processes and non-trusted processes. When trusted processes and non-trusted processes alternate access an encrypted file, in order to ensure the correct data (plaintext or cipher text) in the file cache, file encryption system requires to clean up the file cache data constantly. The method of cache cleanup is such violence and reduces the efficiency of file operations greatly, and there are some compatibility issues.

For the shortcomings of traditional transparent file encryption system in the cache processing, this thesis proposed a transparent file encryption system based on FUSE. The system redirected file operations of application processes in security directory to the virtual disk mapped by FUSE file system, transferring file operations in security directory to the realization of file operations in security directory by FUSE user space process.

The transparent file encryption system based on FUSE of this thesis is divided into three modules for development, file redirection filter module, bridge file system encryption module, system security reinforcement module. File redirection filter module, designed and implemented the management of security directory at application layer, file redirection for secure directory at kernel layer. Bridge file system encryption module, implemented the identification of process and files operations, while implemented the encryption and decryption of files using three

encryption modes, and adds caching mechanism for the efficiency of document literacy. System security reinforcement module, it implemented anti-copy and anti-screenshots for files in security directory using the process monitoring and clipboard monitoring, while the process has been verified authenticity, and protected the security of the configuration file.

The innovation of this thesis is to present a new idea that by means of FUSE file system that plays a role as a bridge and the use of IBE(Identity Based Encryption) encryption algorithm to achieve transparent file encryption and decryption, to ensure that file data in the file system cache is always cipher text. At the same time, transferring I/O operations of files in security directory to the FUSE user space, making it easier to implement and easier to maintain, but also using the clipboard monitoring, prevent data loss due screenshots, copy and other operations, and the authenticity verification of trusted processes, implemented the security reinforcement of the system.

Key words: Data security, Transparent file encryption, FUSE, IBE

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 课题研究背景及意义

随着互联网时代的到来，企业数据经历了爆炸性的增长，推动企业信息化管理逐步进入数据技术（DT）时代，数据资源已然成为企业的核心竞争力，企业的大多数机密数据都是以电子数据的形式存在的。然而保存机密数据的电子文档在创建、保存、使用、传输等环节中均存在着被泄漏出去的风险。计算机中电子文档的安全问题不仅是来自病毒、木马、网络入侵、系统攻击等方面的安全威胁，据研究数据显示，目前信息安全泄密事件中，有 78.9％的损失都是来自于企业内部人员的主动泄密，实际上，企业内部的安全威胁更为严重[1]。

2015 年仅在国内就发生了很多因企业内部员工泄密而导致的信息安全事故。比如，一名就职于中兴公司从事软件研发的员工，负责公司某主流产品业务的研发，2015 年初离职后与合伙人创业开设自己的公司，同样从事与原公司类似业务的研发，为加快自己公司平台的上线速度，该名离职员工违反与中兴公司签订的有关技术保密的协议，将公司内部与研发业务相关的技术文档和程序通过电子邮件等方式发送给合伙人。之后他们创业公司的业务平台上线，给中兴公司造成了 100 多万的损失。2015 年 4 月，因看到公司产品背后巨大的销售利润，南京某公司从事销售工作的职员陈某辞职后创建与原公司同类型企业，并伙同保管公司机密设计数据的同事李某窃取公司机密数据，并通过互联网传输、U 盘拷贝等多种方式，将公司即将推出的新产品的全部设计数据交给陈某。陈某通过生产和销售与原公司同类型的产品，从中非法获利近 300 万元[2]。由于企业内部员工能够接触到企业的最新敏感数据信息，所以一旦内部员工泄密敏感信息，内部攻击的威力要甚于外部。可见，对于信息安全防护，有时防内甚至更重于防外[3]。

为了防止类似上述企业内部员工泄密的发生，有些企业封闭 USB 接口来防止核心机密被 U 盘拷贝出去而导致的泄露，还有些企业禁止企业办公电脑访问互联网[4]。虽然在一定程度上防止了机密数据的泄露，但是这种做法不仅降低了员工的工作效率，而且给日常的办公带来了极大的不方便。从源头开始保障企业机密电子文档的安全，才能更好的确保无论是企业内部员工的泄密，还是企业外部黑客的恶意攻击，都无法获得可以正常使用的机密电子文档[5]。在这种思路下，透明文件加密技术应运而生[6]。

所谓透明加解密，是指在不影响用户使用习惯的情况下自动实现文件的加密和解密，是保证数据安全的重要技术手段[7,8]。在基于文件过滤器驱动的透明文件加密系统中，通常将程序进程分为受信进程和非受信进程[9]；一个受信进程被允许获得加密文件的明文数据，而一个非受信进程则不被允许，只能获得文件的密文数据[10]，比如 Word 程序的程序进程是加密后的 Word 文档的受信进程，能够获得 Word 文档加密文件中的明文数据，而浏览器进程作为 Word 文档的非受信进程而不被允许获得 Word 文档加密文件中的明文数据。

由于文件系统的缓存机制，当受信进程对一个已经加密的文件进行访问时，内存中缓存的是明文数据（在 Windows 环境下，由于存在以内存映射方式打开的文件，因此，透明文件加密系统通常是在文件 Page 读时进行数据解密，从而使得内存中缓存的是明文数据），而当非受信进程对一个已经加密的文件进行访问时，内存中缓存的是密文数据[11,12,13]。当一个受信进程和一个非受信进程对一个已经加密的文件进行不断交替访问时，文件加密系统必须不断地清空系统缓存中缓存的数据[14]，以确保内存中缓存有正确的数据，这称为暴力刷缓存或清缓存问题，暴力刷缓存或清缓存降低了文件操作的效率[15]；进一步地，当一个受信进程和一个非受信进程对一个已经加密的文件同时进行访问时，文件加密系统就很难处理了，这时内存中缓存的数据有可能同时存在明文和密文数据，这既可能造成敏感数据泄漏，又使得受信进程和非受信进程获得错误的数据。

针对透明文件加密中的数据缓存问题，人们提出一些解决方案，如双缓存方案，多 FCB 方案等。双缓存方案在内核层改变文件 I/O 的操作处理，设法使得受信进程访问存有明文的一个缓存，而非受信进程访问存有密文的另一个缓存，但技术方案复杂[16,17]。而多 FCB 方案实际上是引入了一个新的文件系统，它使得每个进程有自己的 FCB 和缓存，开发一个这样的新的文件系统当然是件很复杂的事情，而且内存利用率低、存在重复读取数据的问题，这种方案的最

大问题是用户和程序不能在原有文件系统上直接操作，而是在新文引入的件系统上操作。

本文研究的基于 FUSE 的透明文件加密系统，首先，无论是受信进程还是非受信进程访问文件数据，保证文件系统缓存中的数据始终为密文，解决了暴力清缓存导致的机密文件损坏，效率低下等问题。其次，将文件透明加解密功能通过应用层进行实现，技术方案相对简单，实现起来相对容易。同时，在应用层使用进程监控技术对安全目录中文件打开操作进行监控，防止恶意用户通过复制，截屏泄露机密数据，具有一定的研究意义。

1.3 国内外研究现状

透明文件加密技术是近几年以来针对企业机密电子文档保护而提出来的一种电子文档保密技术。最先提出的透明文件加密技术是随着微软 Windows2000 系统发布而出现的加密文件系统（EFS，Encrypting File System），是一种针对 NTFS 磁盘格式（采用 NTFS 文件系统的磁盘）的文件加密技术[18]。虽然 EFS 加密系统具有简单易用，对用户来说完全透明，与操作系统结合紧密等特点，但是在实际应用过程中还是体现出它的不足之处，比如，只能对 NTFS 磁盘格式分区进行加密，对 FAT 系列磁盘格式分区不具有兼容性，当一个用户在重载操作系统时，没有及时备份与文件加解密有关的加密证书，被 EFS 加密系统所加密的加密文件将无法解密[19]。

在 EFS 加密系统出现之后，微软提出了分层文件系统的概念，这使得实现透明文件加密系统有了更多的选择。在业界，按照实现方式的不同，将透明加解密技术分为四代，分别是基于 API HOOK 实现的第一代技术，基于文件过滤器驱动实现的第二代技术，基于 LayerFSD 的双缓存实现的第三代技术，基于 Minifilter 框架的双缓存实现的第四代技术。

第一代是基于 API HOOK 的在应用层实现的透明文件加密技术，俗称钩子透明加密技术。HOOK 技术是一种基于 Windows 消息处理机制而产生的一种技术。API HOOK 的实现过程就是在一个应用程序进程调用系统的 API 函数前先执行一些特殊的功能[20]。因此利用 API HOOK 技术的这种实现机制，在应用程序对文件进行读操作和写操作过程中，对应用程序进行监控，当读取文件数据的时候，先将加密文件的密文数据解密成明文数据，再调用系统的读 API 函数

将这些明文数据读到内存中，这样当应用程序对文件进行读取操作时显示的就是明文，当写文件数据的时候，先将即将被写入的明文数据进行加密处理，再调用系统的写 API 函数将加密的密文数据写入磁盘，通过这种方式来实现对文件的透明加解密。然而，API HOOK 技术与应用程序联系紧密，是通过监视应用程序对文件的操作来完成文件的加解密的，但是对于不同的应用程序来说，其对文件的操作实现都不尽相同，因此，对于开发的基于 API HOOK 的透明文件加密系统很难兼容所有的应用程序[21]。除此之外，由于 API HOOK 技术经常被黑客应用于各种病毒中，很多应用程序都设置了反 HOOK 保护，一旦发现应用程序中的函数被 HOOK 立即关闭自身应用程序，同时，杀毒软件检测到使用 API HOOK 技术，会将程序当做病毒杀掉。由于以上原因，API HOOK 技术的局限性逐渐的暴露出来，基于 API HOOK 的透明文件加密技术也逐渐被淘汰，后来又开始研究基于内核层的透明文件加密技术。

第二代是基于文件过滤器驱动实现的透明加密技术，是使用 Windows 的文件系统过滤器驱动技术实现的[22,23]，其工作在 Windows 的内核模式下，在 I/O 管理器的下面，磁盘设备过滤驱动的上面[24]。目前的 Windows 文件系统是带缓存的文件系统，即，当一个程序进程读取一个文件的数据时，文件系统先看内存中是否缓存有程序进程要读取的数据，若有，则直接将内存中的缓存的数据返回，否则，文件系统从存储介质中读取一定数量的文件数据（通常是几个页的数据，如通过文件 I/O 的 page 读操作），然后一方面将读取的文件数据缓存在内存中，另一方面将程序进程所需数据返回[25]；当多个进程同时访问一个文件时，它们共享文件系统缓存数据（在 Windows 文件系统中，一个文件只有一个文件控制块 FCB，一个 FCB 对应一个文件缓存，因此多个进程对同一个文件同时进行访问时，它们公用一个 FCB 和文件缓存）[26]。基于以上的原理，为了防止非受信进程获取文件系统缓存中的明文数据，每次在进行受信进程和非受信进程切换时都要清理文件系统缓存。这种暴力清理缓存的方法大大降低了文件操作的效率，并且也存在一些兼容性问题[27]。

针对第二代透明加密技术存在的单缓存缺陷，后来又发展为基于 LayerFSD 的双缓存的第三代透明加密技术。双缓存方案在内核层改变文件 I/O 的操作处理，设法使得受信进程和非受信进程访问不同的缓存区域，当受信进程对一个加密文件进行访问时，使用的是专门保存明文数据的明文缓存区，当非受信进程对一个加密文件进行访问时，则使用保存密文数据的密文缓存区，两个缓存

区同时存在而且互相不干扰，但技术方案实现起来极度复杂[28]。因此，开发基于双缓存的透明文件加密系统存在一定的技术难度。

由于 Minifilter 框架的出现，后来又提出了基于 Minifilter 的双缓存透明文件加密的第四代技术， Minifilter 框架是近年来微软引入的一个轻量级的微过滤器驱动框架，可以不用关注文件过滤器驱动的内部具体实现细节，比如 IRP 的构造以及传递等问题，使得采用 Minifilter 框架进行驱动程序开发可以忽略一些与业务逻辑关系不大的细节性的问题，将开发驱动程序的注意力集中在需求逻辑的实现上，这相对采用原始版本的驱动框架的 Sfilter 框架来说在一定程度上实现更加简单，有一定的实用性优势。但第四代技术仍然使用的是 LayerFSD 的双缓存技术，只是使用了 Minifilter 框架来实现，仍然具有双缓存技术开发难度大，不易于维护等问题。

从国外的研究状况来看，对于透明文件加密技术的研究起步比较早，至今也有二十多年的发展历程，从事该项技术研究的人员和企业众多，所以技术能力比较成熟，具有很强的市场导向。下面对国外几款主流的文件加密系统进行列表分析，如表 1-1 所示。

表 1-1 国外主流文件加密产品

商家	产品名称	采用技术
Microsoft	Windows BitLocker	文件过滤器驱动
Symantec	PGP Encryption Desktop	文件和文件夹加密
未知	TrueCrypt	虚拟磁盘技术

上述国外主流文件加密产品中，Windows BitLocker 是微软继 EFS 之后随着 Windows 7 系统而出现的文件加密工具，它采用文件系统过滤器驱动技术，只能针对整个驱动器进行加密，无法根据用户的实际需求设定安全目录和非安全目录，同时必须在 TPM 模式或 U 盘模式下工作，对计算机硬件有一定的要求，因此在实际使用过程中具有一定的局限性。

PGP Encryption Desktop 是 Symantec 公司研发的一款针对文件和文件夹的加密软件，是一款在全世界范围内比较流行的文件加密软件，它采用逻辑分区来保护文件，针对文件和文件夹进行加密，需要访问加密的文件和文件夹时，必

须要输入加密时设置的密码，这影响了用户的正常使用习惯，无法实现真正意义上的透明，同时需要用户具备比较高的安全意识，如果不小心忘记密码，就永远无法解密。

TrueCrypt 是一款开源的免费的磁盘文件加密软件，由于其操作简单，使用方便，免费，同时能兼容 Windows、Linux、Mac OS 等主流的操作系统的优点而被企业以及个人所广泛使用，TrueCrypt 采用的是虚拟磁盘技术在计算机中建立一个虚拟磁盘，对虚拟磁盘中的文件进行自动加密，在访问加密文件时同样需要输入加密时设置的密码，只有载入后才能实现真正意义上的透明。

从国内的发展状况来看，由于国内透明加密技术起步比较晚，所以从事这方面研究的企业和人员比较少，技术还不是很成熟，同时，与国外相比，国内所提供的有关透明文件加密技术的资料相对比较少，使得透明加密系统的研发比较困难，因此，透明加密技术的发展也相对缓慢。国内使用的透明加密系统主要是采用第二代透明文件加密技术开发的，能够很好的防止企业内部人员通过 U 盘拷贝或者互联网传输导致的敏感数据泄密。下面对国内几款主流的文件加密系统进行列表分析，如表 1-2 所示。

表 1-2 国内主流文件加密产品

商家	产品名称	采用技术
上海索远科技 有限公司	索远 Docsecurity	文件 格式转换
北京亿赛通科技 发展有限公司	亿赛通 CDG 文档 安全管理系统	文件 格式转换
上海山丽信息安全 有限公司	山丽防水墙 系统	文件和文件夹 加密
深圳市巨石安全科技 有限公司	HS-Key 巨石文档 安全管理系统	文件系统过滤器 驱动
宁波海曙绿盾 网络技术有限公司	绿盾-安全信息 管理软件	文件系统过滤器 驱动

上述国内主流透明加密产品，相对国外的同类产品，技术方案相对落后，从采用的技术实现方案上来看，可以分为文件格式转化，文件和文件夹加密，文件系统过滤器驱动三类。

其中以索远 DocSecurity 和亿赛通 CDG 文档安全管理系统为代表的通过文件格式转换技术实现文件加解密产品，需要将机密文件的格式转化为一种特殊的文件格式来实现文件的加密，这种技术方案无法防止企业内部泄密，并且一定程度上影响了用户正常的使用习惯，使用非常不方便。

以山丽防水墙系统为代表的通过文件和文件夹加密技术实现的文件加密产品，需要在加密文件和文件夹时手动输入加密密码，在解密文件和文件夹时也要输入加密密码，这种技术方案也在一定程度上影响了用户的正常使用习惯，无法做到真正意义上的透明。

以 HS-Key 巨石文档安全管理系统和绿盾-安全信息管理软件为代表的通过文件系统过滤器驱动技术实现的文件加密产品，采用的是第二代透明文件加密技术，在系统内核层实现文件的透明加解密，这种单缓存方式的透明加密产品，当机密进程和非机密进程交替访问加密文件时，存在着暴力清缓存的问题，因此系统兼容性和稳定性一直不高。

基于以上的透明加密技术的研究现状，本文借助起着桥梁作用的 FUSE 文件系统[29,30]，并采用 IBE 加密算法以及 AES 加密算法对安全目录下的文件实现透明加解密，已克服原有技术方案在缓存处理方面的缺陷，并实现了针对安全目录文件的反截屏[31]、反复制，在不操作安全目录下的文件的情况下，可正常对文件进行截屏、复制操作，不影响用户的正常使用，同时对操作进程进行了防伪处理，并对保存系统信息的配置文件进行了隐藏以及签名验证处理，更加全面的保证了机密文件数据的安全，是很有意义的一项研究工作。

1.4 本文的主要研究内容

本论文主要使用了一种起着桥梁作用的 FUSE 文件系统，将针对安全目录的文件操作通过文件重定向过滤器驱动重定向到加密桥文件系统所映射的虚拟磁盘中，最终通过 FUSE 用户空间进程来实现文件的透明加解密，克服了现有技术方案在缓存处理方面的不足，同时，针对系统可能存在的安全隐患，本文提供了一种解决方案来实现系统的安全加固，更加全面的保证了机密文件的安全。

本文主要研究工作如下：

（1）文件重定向过滤器部分，实现安全目录的管理和保存，通过鼠标右键设置和取消安全目录，将安全目录信息转换成驱动层可以识别的设备信息保存在配置文件中。文件重定向过滤器驱动将应用程序进程对安全目录中的文件操作和重命名重定向到加密桥文件系统所映射或对应的虚拟文件盘或虚拟文件目录中，将应用程序进程对安全目录中文件的操作转化为加密桥文件系统对安全目录中文件的操作实现。

（2）加密桥文件系统部分，通过将进程名与文件后缀名相结合的方式实现受信进程与非受信进程的识别，在 FUSE 用户空间进程中对应用程序进程针对安全目录中的文件打开、创建以及读写等操作的处理进行了实现，并通过 AES-ECB 加密模式，AES-CTR 加密模式，HASH-CTR 加密模式这三种块加密工作模式对文件的加解密进行了实现，同时在读写文件时添加了缓存机制，提高了文件的读写效率。本文中的透明文件加密系统的透明加解密功能就是在加密桥文件系统模块中实现的。

（3）系统安全加固部分，通过剪贴板监控和进程监控技术，在应用层对打开的安全目录下的文件实现反截屏反复制，打开非安全目录中的文件时可进行正常的复制截屏操作，不影响用户的正常使用，并在进程的判断过程中，同时对进程的 HASH 值进行验证，防止恶意进程伪造成与受信进程同名的进程读取敏感数据信息，同时对配置文件在驱动层进行隐藏和在应用层进行签名验证处理，防止因配置文件被恶意修改而导致的泄密，实现了对机密文件数据的全面防护。

（4）测试部分，首先测试了系统的透明加解密功能，然后分别测试了受信进程和非受信进程对安全目录下加密文件的操作，即，受信进程获取明文数据，非受信进程获取密文数据。在改变进程策略的情况下，对安全目录下文件读写的测试，并测试了如何防止进程欺骗。对使用不同加密模式进行的文件加解密性能进行了测试与对比分析，并对缓存方式和非缓存方式实现的文件读写性能进行了测试与对比分析，并根据测试结果选择合适的加密模式应用于系统之中，最后，针对系统可能存在的安全风险，对系统的安全性进行了分析。

本论文的创新点在于以一种全新的思路借助一个起着桥梁作用的 FUSE 文件系统并采用 IBE 加密算法来实现文件的透明加解密，在确保文件系统缓存文件数据始终为密文的同时，将对安全目录中文件的 I/O 操作的实现通过文件重定

向过滤器驱动转移到 FUSE 文件系统所对应的虚拟磁盘中，使得实现更加容易而且易于维护，并且通过对剪贴板的监控，防止了由于截屏、复制等操作而导致的数据泄露，同时对受信进程进行了真实性验证，并对系统配置文件进行了安全保护，实现了系统的安全加固。

1.5 本文的组织结构

本文总共分为七章，各章节的组织安排如下：

第一章首先分析了本课题的研究背景和研究意义，然后介绍了该研究课题现阶段国内外的发展水平以及研究动态，最后介绍了本论文主要的研究工作和创新点，以及论文的组织结构安排。

第二章首先分析了透明文件加密系统的需求，然后根据系统的需求对系统的整体结构进行了设计并描述了组成系统的模块功能，最后对系统所加密文件的文件结构进行了设计。

第三章主要是对文件重定向过滤器模块进行开发，当应用程序进程对安全目录下的文件实行 I/O 操作时，该模块将安全目录下的文件操作和重命名重定向到加密桥文件系统所映射或对应的虚拟磁盘或者虚拟目录中。同时实现了通过鼠标右键设置与取消安全目录，并通过配置文件管理安全目录信息。

第四章主要是对加密桥文件系统模块进行开发，首先分析了 FUSE 文件系统的实现原理，然后实现了受信进程与非受信进程的识别。在 FUSE 用户空间程序中对文件的各种操作进行了实现，并通过 AES-ECB 加密模式，AES-CTR 加密模式，HASH-CTR 加密模式这三种块加密工作模式实现了文件的加解密，同时在进行文件读写操作时添加了文件数据缓存机制，通过对文件的“预读”和“延迟写”来提高文件读写效率。

第五章主要是对系统安全加固模块进行开发，使用剪贴板监控技术，实现针对安全目录的文件的防复制和防截屏，对受信进程所对应的可执行文件的真实性进行了验证，防止因进程欺骗而导致的泄密，同时对配置文件进行了安全保护，防止因配置文件被恶意修改而导致的泄密。

第六章主要对系统的功能进行了测试，包括透明加解密功能的测试，进程控制策略功能的测试，防止进程欺骗功能的测试，防复制防截屏功能的测试。最后对系统的性能进行了测试与对比分析，包括使用的三种加解密模式进行读

写时性能的测试与比较分析，以及不带缓存机制和带缓存机制进行读写时性能的测试与对比分析，并对系统的安全性进行了分析。

第七章对本论文所完成的工作进行了总结，对其中还需改进的地方做出说明，并且提出了具体的改进方向，对未来的工作进行了展望。

第 2 章 需求分析与整体系统构架设计

2.1 系统需求分析

本文研究的基于 FUSE 的透明文件加密系统，不仅要保证文件存储时的安全，同时也要保证文件使用时的安全。系统通过对保存在磁盘中的文件进行加密来保证存储的安全，防止了文件通过 U 盘或者互联网传输而导致的泄密，另一方面，系统通过对剪切板的监控来保证使用时的安全，防止了对文件内容进行复制，截屏等操作导致的泄密，本系统所实现的基本功能如图 2-1 所示。本系统的设计需求如下：

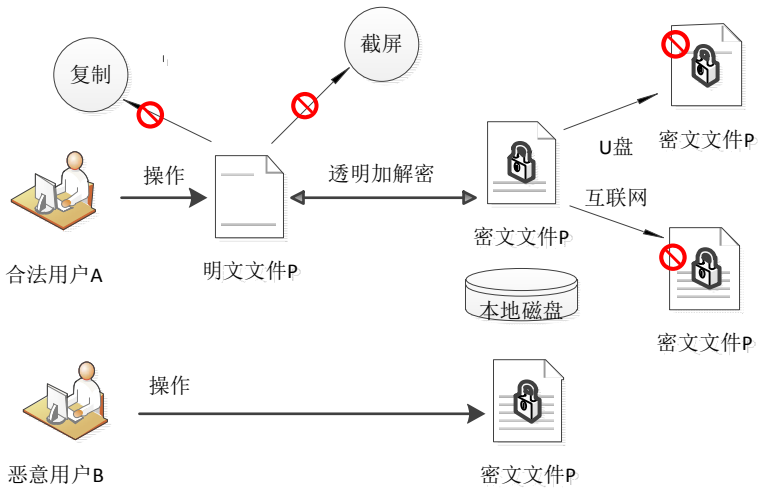

图 2-1 系统功能

（1）文件打开时自动加密，对于安全目录中的文件，任意进程对文件只要有打开动作就对文件进行自动加密，当一个未加密文件从安全目录外复制或者拖入到安全目录中时，文件也进行自动加密处理。

（2）设置进程访问策略，对指定文件的指定进程授予访问权限，可以获取文件的明文数据，未授权的进程只能获取文件密文数据，同时对进程进行防伪处理，防止恶意进程仿冒授权进程而导致的泄密。

（3）读文件时自动进行解密，对于受信进程，读文件时文件进行自动解密，

得到文件明文，非受信进程，读文件时不进行解密操作，得到文件密文。

（4）写文件时自动进行加密，无论是受信进程还是非受信进程对文件进行写操作时，对即将写入的文件数据都要进行自动加密处理。

（5）防止加密文件的恶意传播，通过 U 盘或者互联网等方式传播加密文件时，文件仍然保持密文状态。

（6）防复制防截屏，当打开的是安全目录中的文件时，开始对剪贴板进行监控，只要有复制、截屏操作，无法对文件进行粘贴操作，防止文件通过复制、截屏操作导致文件泄密，当没有安全目录中的文件打开时，可以进行正常的复制、截屏操作。

（7）性能需求，在使用本系统后，对文件的读写操作速度下降的百分比控制在 20%以内。

2.2 系统整体结构设计

本文中透明文件加密系统主要包括三大模块的设计：文件重定向过滤器模块、加密桥文件系统模块、系统安全加固模块。

文件重定向过滤器模块：插入到计算机文件系统中的一个过滤器类型的驱动，当一个应用程序进程打开或创建一个加密文件（在许多文件系统中打开和创建是同一种文件 I/O 操作），或者打开或创建安全文件目录或安全文件盘中的一个文件（加密或未加密文件）时，文件重定向过滤器驱动将文件打开或创建操作重定向到加密桥文件系统所映射或对应的虚拟文件盘中的一个文件（虚拟文件），即在重定向后的文件打开或创建操作中，应用程序进程要打开或创建的原文件的文件路径被转换成（加密桥文件系统所映射或对应的）虚拟文件盘中的一个文件路径（一个虚拟文件路径）。

加密桥文件系统模块：所谓加密桥文件系统，就是一个在应用程序进程对安全文件目录或安全文件盘中的文件进行文件 I/O 操作过程中起着桥梁作用并实现文件的透明加解密的用户空间文件系统（FUSE）。该模块首先要实现通过文件后缀名和进程名相结合的方式来判断进程是受信进程还是非受信进程。加密桥文件系统（的 FUSE 文件驱动）被映射或对应到计算机文件系统的一个虚拟文件盘，其中虚拟文件盘不对应计算机文件系统的存储介质上的一个磁盘分区或文件目录结构，但在用户和程序进行文件 I/O 操作时表现为一个文件盘或文件

目录。加密桥文件系统的 FUSE 用户空间程序的程序进程对所有加密文件而言都是非受信进程，当文件重定向过滤器驱动将一个应用程序进程打开或创建安全文件目录或安全文件盘中的一个文件的操作，重定向到加密桥文件系统所映射或对应的虚拟文件盘后，加密桥文件系统通过 FUSE 用户空间程序将应用程序进程针对重定向后的文件的所有操作转化为针对重定向前的原文件的操作。因此，一个应用程序进程对安全目录中文件的透明加解密最终转化为加密桥文件系统中 FUSE 用户空间进程对文件读写操作的实现。

系统安全加固模块：本模块通过剪贴板监控技术，实现针对读取文件的明文数据的防复制和文件窗口的防截屏。通过对操作进程的进程文件进行防伪处理，防止因进程名篡改而导致的明文数据泄露。同时，对保存系统信息的配置文件进行安全保护，防止因恶意修改配置文件而导致的泄密。

由三大模块组成的透明文件加密系统的整体执行流程如图 2-2 所示。

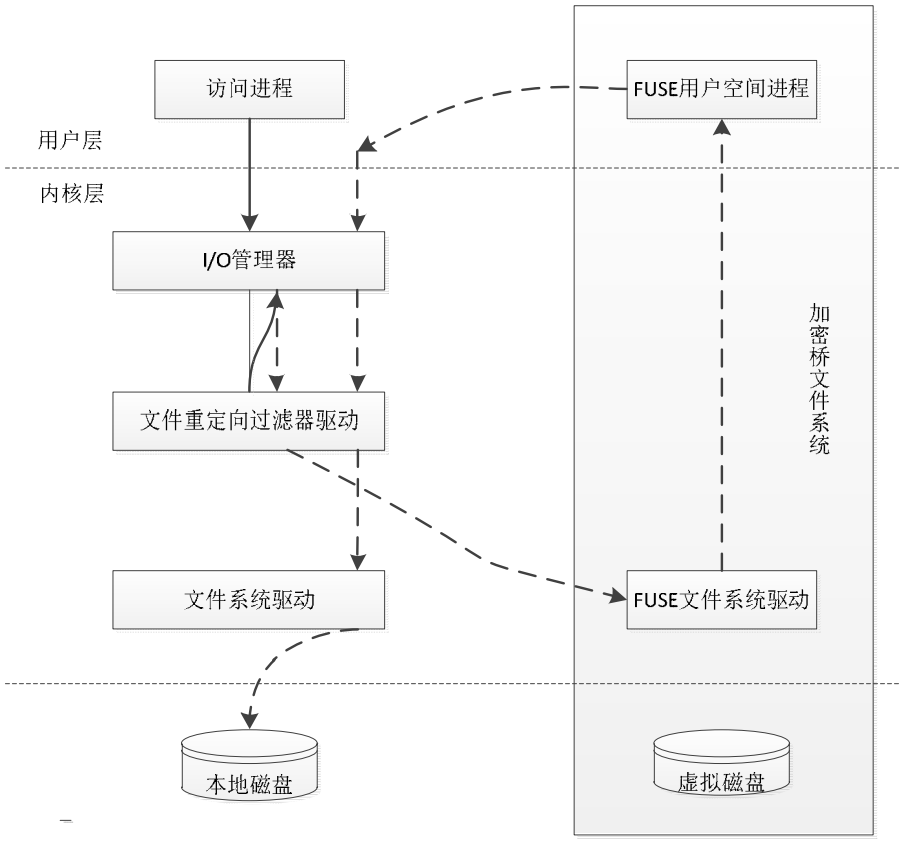

图 2-2 系统整体执行流程

当一个应用程序进程对本地磁盘安全目录中的文件进行操作时，文件重定

向过滤器模块通过获得配置文件中的安全目录信息将针对指定安全目录中的文件操作重定向到虚拟磁盘中，加密桥文件系统模块将这个本地磁盘安全目录映射成一个虚拟磁盘，并对重定向过来的虚拟磁盘中的文件操作进行了实现，当一个安全目录中的文件被打开时，系统安全加固模块对文件提供了更加全面安全的防护，这个过程就组成了本文中的透明文件加密系统。其中文件重定向过滤器模块又由信息配置管理模块，文件重定向模块这两个模块组成，加密桥文件系统模块则由进程识别模块，文件操作实现模块，文件加解密模块，读写缓存机制模块这四个模块组成，系统安全加固模块由防复制防截屏模块，防止进程欺骗模块，配置文件保护模块这三个模块组成。系统整体构架如图 2-3 所示。

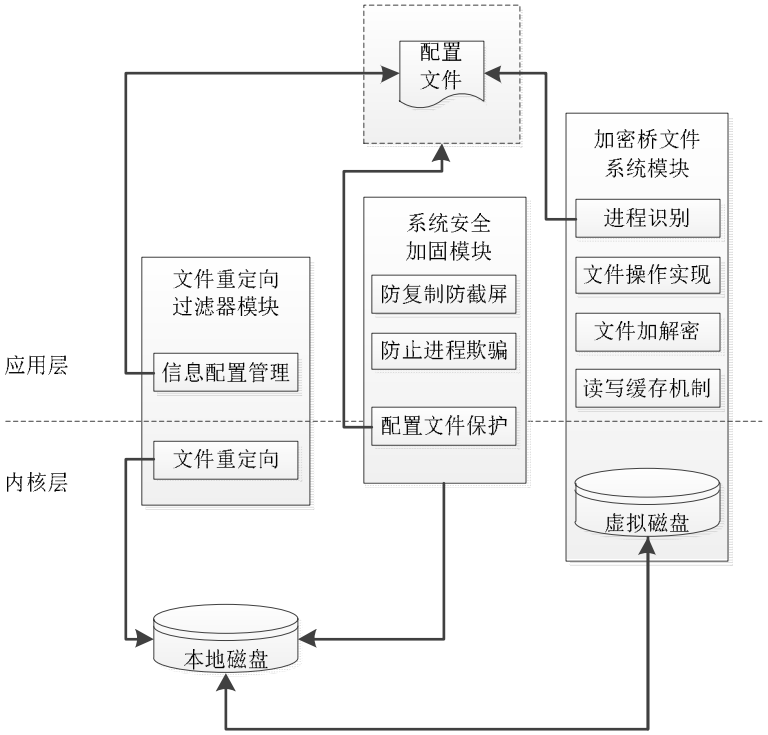

图 2-3 系统整体构架

信息配置管理模块：实现通过鼠标右键菜单对安全目录的设置和取消，并将安全目录信息通过配置文件来管理，将加解密需要的用户身份标识也使用配置文件来进行统一管理。

文件重定向模块：在驱动层实现将应用程序进程对安全目录中的文件操作及文件重命名操作重定向到加密桥文件系统所映射或对应的虚拟文件盘或者虚拟目录中。

进程识别模块：该模块实现受信进程与非受信进程的设置，是通过文件后缀名与进程名相结合的方式来设置的，并将设置的信息保存在配置文件中。通过配置文件中的设置信息来完成受信进程与非受信进程的识别。

文件操作实现模块：该模块涉及到应用程序进程对安全目录中文件的主要操作，其中包括对安全目录中文件打开操作，对文件的读操作，对文件的写操作，文件信息查询操作，文件信息设置操作，文件关闭操作。文件操作实现模块中所涉及到的部分如图 2-4 所示。

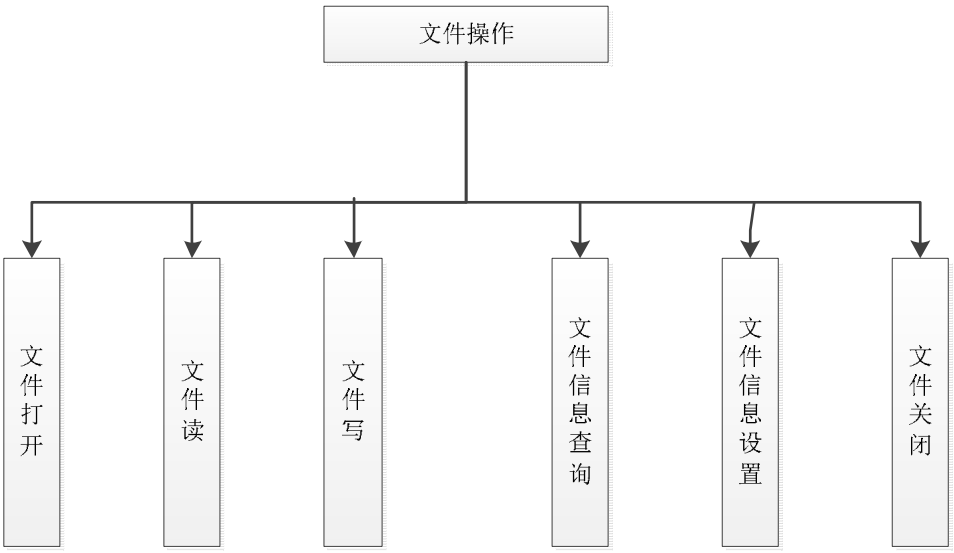

图 2-4 文件操作

文件加解密模块：该模块采用 AES-ECB 模式，AES-CTR 模式，HASH-CTR 模式这三种加密模式实现对文件的加解密处理。

读写缓存机制模块：在读写文件时，添加文件数据“预读”和“延迟写”的文件缓存机制，提高文件读写时的效率。

防复制防截屏模块：监控剪贴板中内容的变化，发现剪贴板中有复制的文本类型文件或者截屏的图片类型文件则清空剪贴板中的内容。通过主动侦测进程所打开的文件，只针对安全目录中所打开的文件才启动剪贴板监控实现防复制防截屏，而不影响正常的文件复制和截屏操作。

防止进程欺骗模块：该模块通过对操作进程的进程文件进行 HASH 验证，防止通过将恶意进程名修改为受信进程名而导致的机密信息泄露。

配置文件保护模块：为了防止配置文件中的信息被恶意用户所更改而导致

机密信息泄漏，本文通过在驱动层将配置文件进行隐藏处理和在应用层对配置文件进行签名验证来保护配置文件。

除了读文件时进行自动解密处理，写文件时进行自动加密处理之外，必须要注意到另外一个问题：受信进程或非受信进程所操作的文件不一定是已经加密的文件[32]，如何判断打开的这个文件是加密的还是未加密的。本文采取在加密文件的文件内容中保存文件加密标识。那么问题出现了：文件加密标识应该保存在加密文件的文件内容中的哪个位置。一般有两种方案，一种是保存在加密文件的文件头部，另一种是保存在加密文件的文件尾部[33]。

将文件加密标识保存在加密文件的文件尾部实现起来是比较简单的，但是存在极大的安全隐患，这是因为加密文件的文件尾部绝对不是一个维持不变的部位，这个位置是随着文件的增大或者变小而随时变化的，一旦在这个改变过程中出现一些意外情况，比如断电，则这个尾部的加密标识容易丢失 [27]。将文件加密标识保存在加密文件的文件头部的好处是稳定可靠，位置是固定不变的，文件尾部的内容不管怎样操作都不对文件加密标识产生干扰[34]。因此，本文采取将文件加密标识保存在文件头部的处理方案，带有加密标识的加密文件结构图如图 2-5 所示。

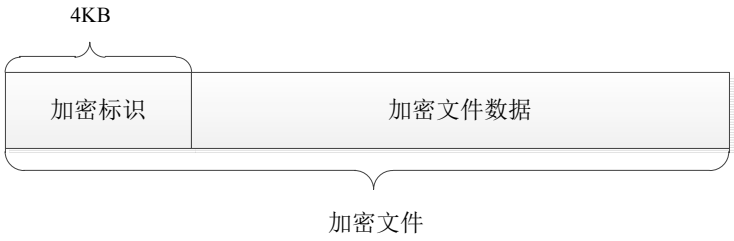

图 2-5 加密文件结构

文件加密标识不仅可以用于判断文件是否加密，还可以用来存储跟加解密有关的一些信息，比如身份标识信息、密钥信息等。由于本文中对安全目录中的文件内容采取 AES 对称加密算法加密的方案[35,36]，所以需要将对称加密的密钥存储在文件加密标识中，但是这个密钥不能直接暴露出来，因此本文采取的策略是将这个用于加密文件内容的对称密钥经过 IBE 非对称加密算法[37]加密后存放在文件加密标识中，这样在读取文件进行解密时，可以直接从加密标识中

解析出加密时的对称密钥。本文对加密标识的结构设计如图 2-6 所示。

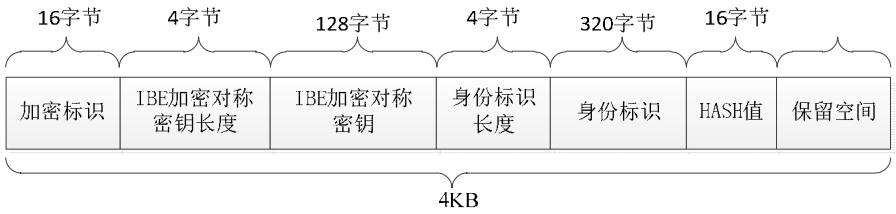

图 2-6 加密标识结构

其中前 16 个字节的内容表示加密标识，本文使用 16 个字母“A”来进行填充，以此来标识文件是由本系统所加密的文件。随后存储了经过 IBE 加密后的对称密钥长度和对称密钥信息，便于在对文件进行解密处理时获得解密对称密钥。由于对称密钥是采用 IBE 加密算法进行加密处理的，因此后面又存储了用于 IBE 加密的身份标识长度和身份标识信息[38]，用于解密文件时解密使用 IBE 加密过的对称密钥。若简单地使用前面 16 个字节的加密标识来判断文件是否加密是存在安全隐患的，倘若安全目录中的某文件前 16 个字节存储的刚好是 16 个“A”，那么该文件就会被当作是已经加密的文件进行处理，这样该文件就永远不会被加密。为了避免这种问题的发生，本文在加密标识中保存了一个 16 字节长度的 HASH 值，该 HASH 值是对加密标识的前 472 个字节取 MD5 值[39]。因此，在判断文件是否加密时，不仅需要比较前 16 个字节的加密标识，还需要获取加密标识前 472 个字节的 MD5 值，然后比较该 MD5 值与加密标识中所保存的 MD5 值是否一致，这样就可以准确无误的判断一个文件是否加密。在加密标识的最后保留了一块预留空间，便于对文件加密标识内容的扩展。

2.3 本章小结

本章主要针对本文中基于 FUSE 的透明文件加密系统设计的需求，完成了系统功能分析和整体结构的设计，明确了由各个模块组成的系统的执行流程，最后对加密文件的结构进行了设计，并根据系统的实际使用需求对文件加密标识结构进行了设计。

第 3 章 文件重定向过滤器模块实现

文件重定向过滤器部分主要是实现将安全目录中的文件重定向到加密桥文件系统所映射或对应的虚拟文件盘或者虚拟文件目录中的一个文件。其中文件重定向过滤器信息配置管理是在应用层实现的，对文件的重定向是在内核层实现的。

3.1 文件重定向的驱动层实现

实现本文中透明文件加密系统的重点是将安全目录中的文件重定向到加密桥文件系统所映射或对应的虚拟磁盘中，然后通过加密桥文件系统对安全目录中的文件进行操作实现，其中安全目录相关知识会在后面章节中介绍。本节重点介绍文件重定向在驱动层的实现以及涉及到的相关技术的实现。

3.1.1 避免重入

在文件系统过滤器驱动中进行文件操作时，重入是极其常见的，而且给系统带来了很多不稳定的因素。所谓重入，就是在调用一个函数时，在函数的执行过程中，需要重新调用该函数，这种情况如果处理不当很可能造成死循环而导致程序崩溃，递归就是典型的重入。本文中的透明文件加密系统也涉及到重入问题，如图 3-1 所示。

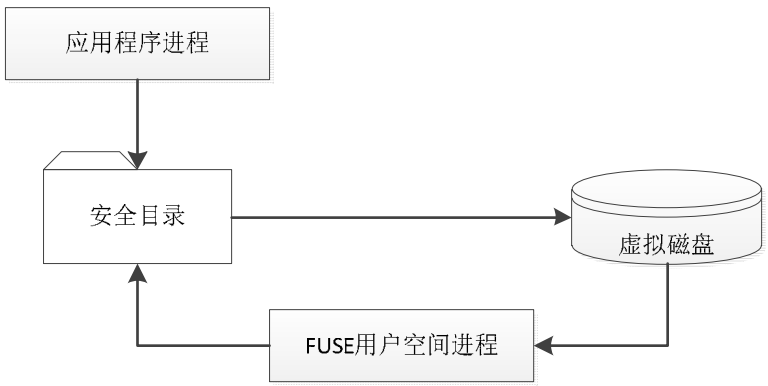

图 3-1 重入问题

当应用程序进程操作一个安全目录中的文件时，系统将应用程序进程对安全目录中文件的操作重定向到对加密桥文件系统所映射或对应的虚拟磁盘中的一个同名文件的操作，实际上虚拟磁盘上的文件是不存在的，是通过加密桥文件系统的 FUSE 用户空间进程来操作虚拟磁盘对应的本地磁盘目录的，而这个本地磁盘目录就是应用程序进程操作磁盘的安全目录，这就转化为 FUSE 用户空间进程对安全目录中文件的操作，此时系统又将 FUSE 用户空间进程对安全目录中文件的操作重定向到对加密桥文件系统所映射或对应的虚拟磁盘中文件的操作，如此反复就导致了死循环，造成了重入。

为了避免上述重入现象的发生，本系统采取对 FUSE 用户空间进程不进行重定向的方式实现的，即当应用程序进程对安全目录中文件的操作转化为加密桥文件系统 FUSE 用户空间进程对安全目录中文件的操作时，在重定向驱动程序中判断进程是否是 FUSE 用户空间进程，如果是 FUSE 用户空间进程则返回 FLT_PREOP_SUCCESS_WITH_CALLBACK，将 FUSE 用户空间进程对安全目录文件操作的请求向下层发送，此时 FUSE 用户空间进程对文件的操作不会被重定向，这样就避免了因死循环而导致的重入问题。

3.1.2 驱动层进程名获取方案

为了避免上述介绍的重入问题，需要在驱动层获得操作安全目录中文件进程的进程名，得到进程名后需要判断操作进程是否是 FUSE 用户空间进程，如果是 FUSE 用户空间进程则不进行重定向操作。因此，解决重定向问题的重点是如何在驱动层获得操作进程的进程名，本文提供了两种在驱动层获得进程名的方案。

方案一

在 Windows 内核中，为每一个进程都维护了一个名为 EPROCESS 的数据结构，在这个结构中保存有进程的进程名。通过 Windbg 调试工具的 Kernel Debug 功能，使用 dt 命令调试 EPROCESS 结构，发现 EPROCESS 结构中有一个 ImageFileName 域，这个域保存的就是进程的进程名。例如：使用记事本进程 notepad.exe 进程访问一个安全目录中的文件，可以看到 ImageFileName 域如图 3-2 所示。

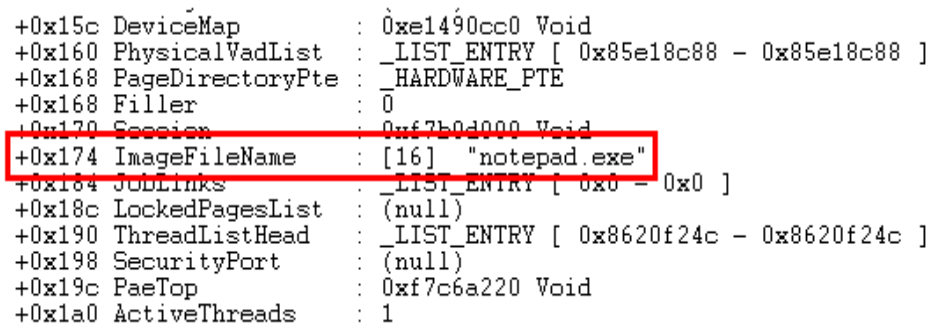

图 3-2 EPROCESS 中的 ImageFileName 域

对于不同版本的操作系统而言，ImageFileName 域在 EPROCESS 结构体中的偏移量是不同的，所以不能简单的根据一个偏移量来获取 ImageFileName 域中的内容。由于内核模块的入口函数 DriverEntry 总是在一个系统进程（system 进程）中被执行，那么在 DriverEntry 入口函数函数中得到的当前进程就是 system 进程。因此，可以在 DriverEntry 函数中查找 EPROCESS 结构中的 system 字符串，一旦查找到这个字符串，保存进程在 EPROCESS 结构中的相对位置。有了这个相对位置之后，就可以从 EPROCESS 结构中获得其他进程的进程名了。

方案二

EPROCESS 结构中有一个 SeAuditProcessCreationInfo 域，这个域是一个名为_SE_AUDIT_PROCESS_CREATION_INFO 的结构体，如图 3-3 所示，这个结构体下面同样有一个名为 ImageFileName 的域，这个域保存的是进程的全路径名。

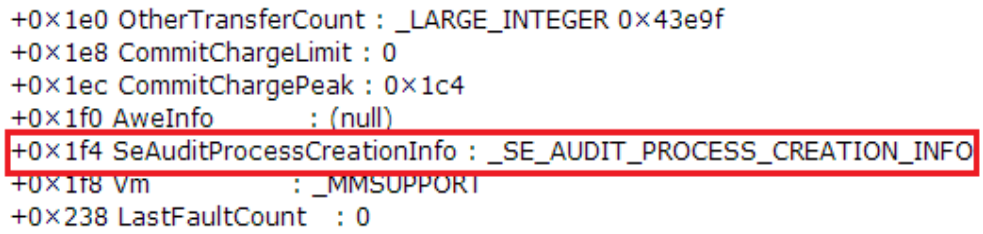

图 3-3 EPROCESS 中的 SeAuditProcessCreationInfo 域

要获得 EPROCESS.SeAuditProcessCreationInfo.ImageFileName 中的内容，就需要使用 ZwQueryInformationProcess 函数，此时得到的 ImageFileName 并不是进程名，而是进程的全路径名，根据进程的全路径名解析出进程名即可。

对比上述两种方案，方案一采取的是获取 EPROCESS.ImageFileName 的方法，该方案虽然实现起来简单，但是由于 ImageFileName 域存在 16 个字节的长度限制，当进程名超过 16 个字节时会被截断，无法获取正常的进程名。方案二采取的是获取 EPROCESS.SeAuditProcessCreationInfo.ImageFileName 的方法，该方案能够获取进程的全路径，而且不受操作系统类型和进程名长度的限制。因此，本系统采用方案二来获取驱动层进程名。

3.1.3 文件重定向实现

所谓文件重定向，就是把对一个文件 F1 的创建、打开或者读写等操作请求转化为对另一个文件 F2 的对应操作[40]。在对一个文件进行读写等操作之前，首先必须要打开这个文件，因此，对一个文件进行重定向就是把对这个文件的打开或创建操作重定向到另一个文件[41]。

因此，系统对文件操作进行重定向，实质上是对 IRP_MJ_CREATE 这个 IRP 请求包进行重定向。文件系统过滤器驱动可以拦截到这个 IRP 请求包，并返回 STATUS_REPARSE 状态码以实现文件的重定向。本文的透明文件加密系统所涉及的文件重定向是将应用程序进程对安全目录中文件的操作重定向到加密桥文件系统所映射或者对应的虚拟磁盘中，实现原理如图 3-4 所示。

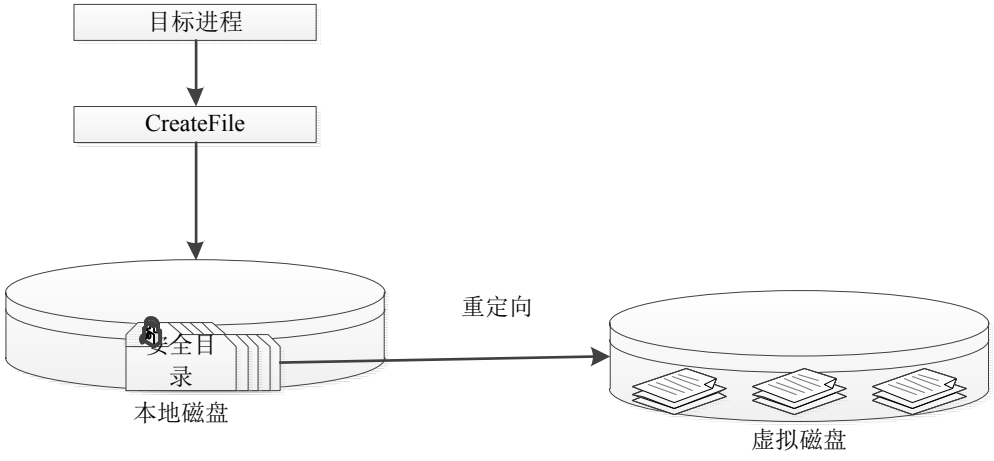

图 3-4 文件重定向原理

对于 IRP_MJ_CREATE 这个 IRP 请求包，在其当前栈空间中，存在着一个 FileObject 域，用来表示 IRP 请求操作的文件的文件对象，其中 FileObject 域中

的 FileName 域表示请求操作的文件所对应的文件全路径名。因此，可以通过FileObject 域得到操作安全目录中文件的文件路径，然后用加密桥文件系统所映射或对应的虚拟磁盘中文件的文件路径替换掉 FileObject 域中 FileName 域的内容，然后设置 IRP 请求的参数信息，完成这个 IRP 请求并返回 STATUS_REPARSE 状态码。

当 I/O 管理器检测到 IRP 对应的返回值为 STATUS_REPARSE 并且 IRP 中的 Information 域被设置为 IO_REPARSE 时，开始重新检查 FileObject 域中的 FileName 域，发现 FileName 这个域被更改后，会重新构造一个新的 IRP 请求。此时在 IRP 的当前栈空间中，FileObject 域已经是替换后的加密桥文件系统所映射或对应的虚拟磁盘中的文件所对应的文件对象。I/O 管理器然后会调用对象管理器解析新的替换后的文件名，并根据新的替换后的文件对象所对应的设备对象确定将新的 IRP 请求发送到加密桥文件系统所映射的虚拟磁盘所对应的卷设备对象。因此，不管重定向后的目标文件与原始请求操作的文件是否在同一个磁盘分区上，新构建的 IRP 请求都能够被传递到目标文件所对应的卷设备对象上。因此完成了针对安全目录中文件的打开操作的重定向，也完成了对文件其他操作的重定向。

因此，实现将应用程序进程对安全目录中的文件操作重定向到加密桥文件系统所映射或对应的虚拟磁盘中文件的操作，具体的实现流程如图 3-5 所示，可以将实现过程分为如下几个步骤：

1）获得当前操作文件的进程名，并判断是否是 FUSE 用户空间进程，如果是则直接返回 FLT_PREOP_SUCCESS_WITH_CALLBACK;，下发 IRP 请求。

2）从 FileObject 文件对象的 FileName 域中获得操作文件的文件全路径名。

3）通过获得的文件全路径名，判断应用程序进程所操作的文件是否是安全目录中的一个文件，如果不是安全目录中的文件则下发 IRP 请求。

4）用加密桥文件系统所映射或对应的虚拟磁盘中的文件全路径名替换 FileObject 中 FileName 域的内容。其中虚拟磁盘中的文件全路径名包含卷设备对象名。由于 FileObject 中的 FileName 是一个 UNICODE_STRING 类型的字符串，因此需要分配一块自己的缓冲区，该缓冲区中存储虚拟磁盘中文件的全路径名，并用该缓冲区替换掉 FileObject 中 FileName 的缓冲区。

5）设置 IRP 中的 IoStatus.Status 域为 STATUS_REPARSE，并设置 IoStatus.Information 域为 IO_REPARSE。

6）完成 IRP 请求，并返回 STATUS_REPARSE。

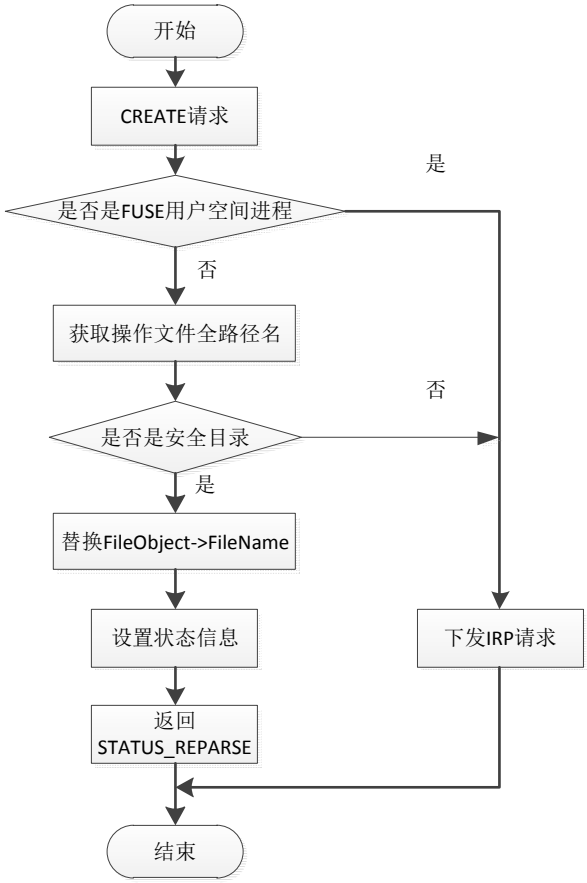

图 3-5 文件重定向程序流程

3.1.4 文件重命名重定向实现

对于某些应用程序进程，当修改安全目录中的文件时，会产生一些临时文件，进程对文件的修改实际上是对产生的临时文件的修改，修改完成后临时文件会重命名回原来的文件，此时，如果不对文件的重命名操作进行重定向，文件修改后将无法保存。因此，除了对打开或创建文件操作（IRP_MJ_CREATE）中的文件路径进行重定向处理外，对于文件重命名操作中的重命名文件路径也需要进行重定向处理，以确保重命名后的文件在加密桥文件系统所映射或对应的虚拟磁盘中。

当安全目录中的文件发生重命名操作时，I/O 管理器会调用对象管理器产生一个主功能代码为 IRP_MJ_SET_INFORMATION 的 IRP 请求包，文件系统过滤

器驱动可以过滤到这个 IRP 请求，从而将安全目录中文件的重命名操作转化为对文件系统过滤器驱动程序中 IRP_MJ_SET_INFORMATION 所对应的分发例程的调用。

对于 IRP_MJ_SET_INFORMATION 这个 IRP 请求包，在 IRP 的当前栈空间中有一个 FileInformationClass 域，该域表示的是应用程序进程对安全目录中文件的一些设置操作，当安全目录中的文件有重命名操作时，FileInformationClass 域中的内容将会变成 FileRenameInformation。因此，可以通过判断 FileInformationClass 中的内容是否为 FileRenameInformation 来判断是否是重命名操作。

在加密桥文件系统中有一个 renameContext 结构体，该结构体存放与重命名有关的信息，其中包括重命名后的文件名 FileName，当文件系统过滤器驱动过滤到安全目录中文件的重命名操作之后，将 renameContext 结构体中的 FileName 域填充为安全目录中文件重命名后的文件名，并将 eventContext->SetFile.FileName 填充为安全目录中将要被重命名的文件名，此时，将对安全目录中文件的重命名操作重定向到加密桥文件系统映射或对应的虚拟磁盘中，因此，重命名后的文件便可出现在桥文件系统所映射会对应的虚拟磁盘中。

3.2 文件重定向过滤器信息配置管理

为了满足不同用户的实际使用需求，本文的透明文件加解密系统提供了可根据用户自身使用需求设置计算机本地磁盘中的任何目录为安全目录，只有安全目录中的文件才可以进行透明加解密并使用系统所提供的所有功能。同时，用户也可以根据自身的使用需求，取消设置好的安全目录，取消后的目录中的文件不能进行透明加密及使用其他功能。

本文的透明文件加解密系统是通过针对目录点击鼠标右键来设置安全目录和取消安全目录的，添加设置安全目录和取消安全目录的鼠标右键菜单是在注册表 HKEY_CLASSES_ROOT\Folder\shell 分支下创建主键“设置安全目录”和“取消安全目录”，并在“设置安全目录”和“取消安全目录”下创建 Command 主键，在“设置安全目录”下的 Command 主键的键值编辑框中输入“C:\Program Files\mini_scrypt\SelectSafeDir.exe /p %1”，在“取消安全目录”下的 Command

主键的键值编辑框中输入“C:\Program Files\mini_scrypt\RemoveSafeDir.exe /p %1”。

当针对目录通过鼠标右键点击设置安全目录时，设置安全目录进程SelectSafeDir.exe 启动，在 SelectSafeDir..exe 进程中通过 CommandLineToArgv 函数获得命令行字符串，得到的命令行参数即为安全目录的全路径。同样的当针对安全目录通过鼠标右键点击取消安全目录时，取消安全目录进程RemoveSafeDir.exe 启动，在 RemoveSafeDir.exe 进程中同样通过CommandLineToArgv 函数获得即将要取消的安全目录的全路径。系统设置安全目录的执行流程如图 3-6 所示，取消安全目录的执行流程与此相似。

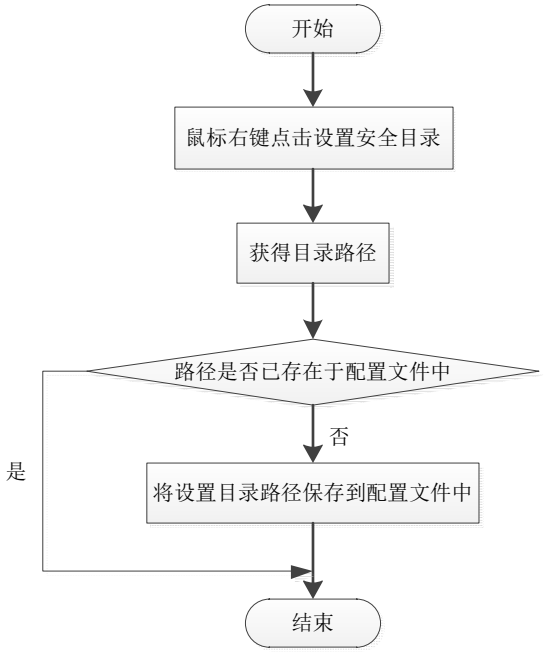

图 3-6 设置安全目录

3.3 本章小结

本章主要针对文件重定向过滤器模块进行了设计和实现，其中在系统内核层实现了将安全目录中的文件操作及文件重命名重定向到加密桥文件系统所映射的虚拟磁盘中，在应用层实现文件重定向过滤器信息配置管理，通过鼠标右键实现安全目录的设置和取消。

第 4 章 加密桥文件系统模块实现

加密桥文件系统模块主要实现受信进程与非受信进程的识别，文件打开、创建以及读写等操作的处理，并通过三种加密模式对文件的加解密进行了实现，同时在读写文件时添加了缓存机制，提高了文件的读写效率。本文中的透明文件加密系统的透明加解密功能就是在加密桥文件系统模块中实现的。

4.1 FUSE 文件系统技术分析

本文中的 FUSE 文件系统就是基于 Dokan 库来开发的，Dokan 文件系统会在计算机中虚拟出一个磁盘，称作虚拟磁盘，本节主要针对 FUSE 文件系统实现原理和回调函数进行了分析。

Dokan 库包含一个用户空间的 DLL 文件（dokan.dll），一个用户空间的进程文件（mirrorfs.exe）以及一个处于内核层的文件系统驱动（dokan.sys）[42]。当一个应用程序进程操作虚拟磁盘中的文件时，首先会将这个操作请求发送给 FUSE 文件系统驱动（dokan.sys）。然后文件系统驱动会通知 FUSE 用户空间 DLL（dokan.dll）完成这个操作请求，最后 FUSE 用户空间 DLL 会通过 FUSE 用户空间进程（mirrorfs.exe）的操作回调函数对操作请求进行实现，一个应用程序对 FUSE 文件系统所对应虚拟磁盘中文件进行操作执行流程如图 4-1 所示。

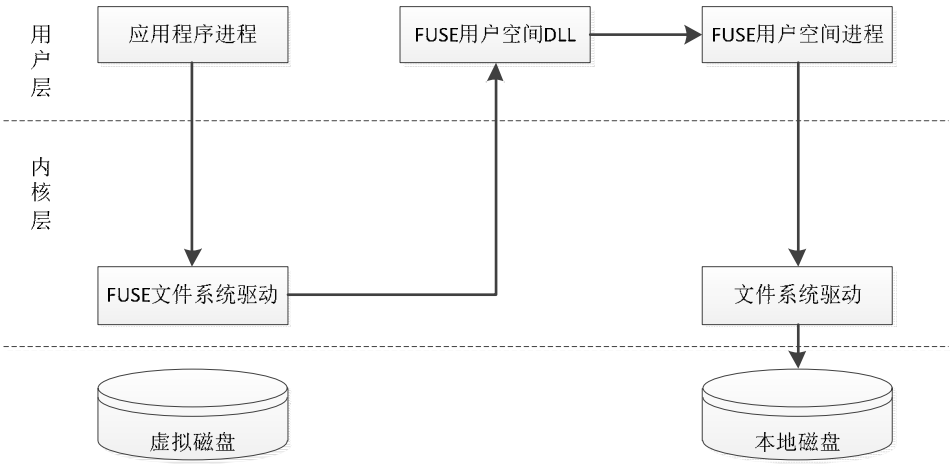

图 4-1 FUSE 执行原理

为了创建一个 FUSE 文件系统，FUSE 用户空间程序必须要实现 DOKAN_OPERATIONS 结构体中的回调函数。实现了这些回调函数以后，FUSE 用户空间进程就可以将 DOKAN_OPERATIONS 结构体作为参数调用 DokanMain 函数来挂载用户空间文件系统。DOKAN_OPERATIONS 结构体中的回调函数的相应的语义功能与 Windows API 中相应的函数是非常相似的，因此这些回调函数的参数也与 Windows API 中相应的函数是相似的。由于是多线程的调用这些回调函数，因此这些回调函数都是线程安全的，否则可能会出现很多问题。

表 4-1 几种典型的回调函数

回调函数	FUSE 用户空间实现
dokanOperations->CreateFile	MirrorCreateFile
dokanOperations->ReadFile	MirrorReadFile
dokanOperations->WriteFile	MirrorWriteFile
dokanOperations->GetFileInformation	MirrorGetFileInformation
dokanOperations->SetEndOfFile	MirrorSetEndOfFile
dokanOperations->SetAllocationSize	MirrorSetAllocationSize
dokanOperations->Cleanup	MirrorCleanup
dokanOperations->CloseFile	MirrorCloseFile

表 4-1 中列出了 DOKAN_OPERATIONS 结构体中几种典型的回调函数。在访问进程对文件进行操作（读文件操作，写文件操作，查询文件大小操作等）之前，文件的创建函数如 CreateFile 会被首先调用。另一方面，当文件被关闭时，Cleanup 回调函数会被 Dokan 文件系统驱动调用。当文件操作成功时，每个回调函数都应该返回 STATUS_SUCCESS，否则应该返回 NtStatues（Win32 错误码）。例如，当 CreateFile 函数打开一个文件失败时，应该返回 STATUES_OBJECT_NAME_NOT_FOUND。

4.2 受信与非受信进程识别技术实现

在透明文件加密中，所有的操作文件的进程被分为两类：受信进程与非受信进程。所谓受信进程，是指用户在工作时使用的创建工作文档的进程以及被授予获取明文数据权限的进程。这些文档在某个范围内是机密的，是不允许外

泄的。此外，受信进程在打开或创建文档时自动加密，也可以正常读取加密后的机密文件（自动解密）。其他的进程则是非受信进程，但是读取文件时是不进行解密的，数据内容等同无效。

根据上述的原则来区分进程，人们容易产生一个常见的误解，认为既然受信进程所生成的文件是加密文件，具有保密安全性，那么可以将所有的操作进程都设置为受信进程，这样所有的操作进程所创建的文件都具有保密安全性，而且还省去了区分进程的麻烦。但是这种想法是有问题的，因为所有的进程都被设置为了受信进程，也就是说，所有的进程读取加密后的机密文件时都会自动解密。例如：如果一个公司职员通过 E_mail 把加密后的公司机密文件传送给公司之外的竞争对手，由于发送 E_mail 的进程是一个受信进程，因此该进程在读取机密文件时会自动解密。这样所导致的后果是，虽然机密文件在公司磁盘上已经加密了，但是竞争对手接收到的文件是经过解密后的明文数据，机密信息就被泄露了，会给公司造成不必要的损失。

想要识别受信进程和非受信进程，首先必须获得操作进程的进程名，由于进程的判断是在 FUSE 用户空间程序中进行的，所以可以通过 DOKAN_FILE_INFO 结构体中的 ProcessId 域来获得操作进程的进程名。

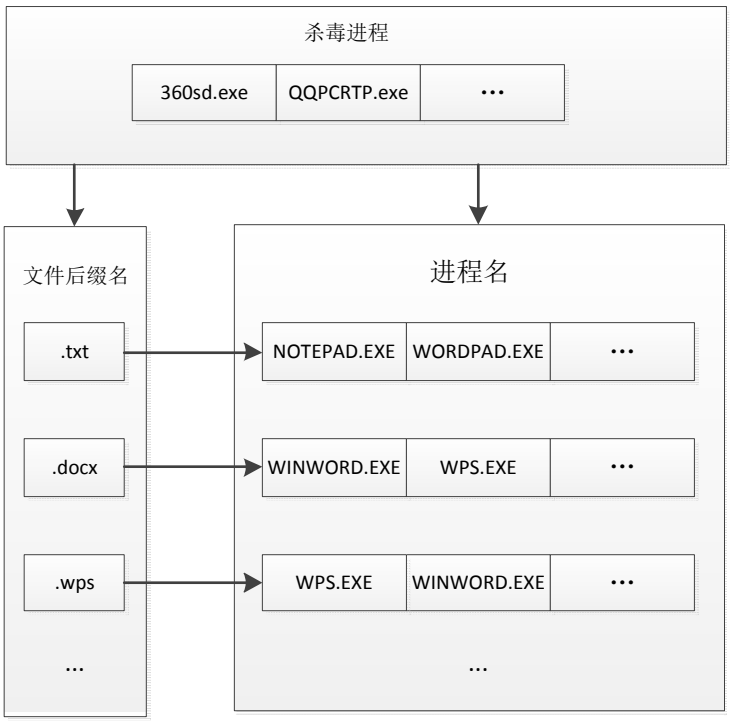

图 4-2 受信进程表

本文中对受信进程表的设置如图 4-2 所示。本文并不是简单地根据进程名来设置进程策略的，而是通过进程所操作的文件的文件后缀名和进程名相结合的方式来设置的，有利于提高进程与文件类型的关联性。所采取的具体方案是：创建一个专门保存受信进程表的配置文件，收集常用办公文档的文件后缀名，针对不同的文件后缀名，设置相应的进程为受信进程，并根据这种文件后缀名与进程名的对应关系建立一个受信进程表。

假如操作系统中安装了杀毒软件，杀毒软件会一直扫描磁盘中的文件来检查病毒文件，此时当进程操作安全目录中的文件时，正好杀毒软件也在扫描安全目录中的文件时，就会对进程的操作产生干扰甚至会拦截安全目录中的文件，所以，不管是针对何种类型的文件，统一将杀毒软件对应的杀毒进程当做是受信进程处理，让杀毒进程读取明文数据，此时杀毒软件就不会对安全目录中的文件操作产生干扰。本文中受信进程与非受信进程的识别过程如图 4-3 所示。

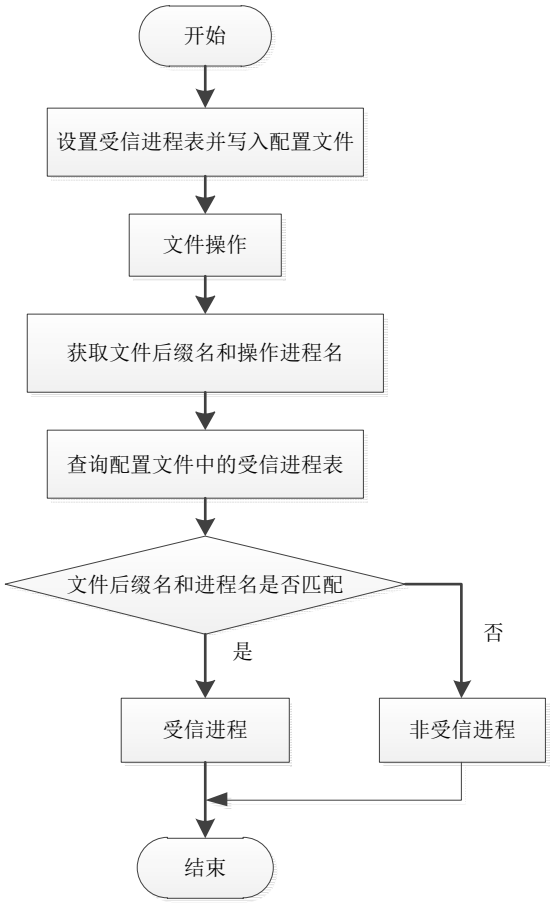

图 4-3 进程识别

4.3 文件操作实现

4.3.1 文件打开操作处理

当一个应用程序进程打开安全目录中的一个文件时，会触发 I/O 管理器向安全目录所在的磁盘设备发送一个 IRP_MJ_CREATE 的请求，文件过滤器模块会拦截这个 IRP 请求，将应用程序进程对安全目录中文件的打开操作重定向到加密桥文件系统所映射的虚拟磁盘中，这个虚拟盘是由安全目录映射过来的，因此，根据 FUSE 文件系统原理，FUSE 用户空间进程会实现对应的安全目录中的文件的打开操作。从而将应用程序对安全目录中文件的打开操作转化为 FUSE 用户空间进程对安全目录中文件的打开实现。

在 FUSE 用户空间进程中，回调函数 dokanOperations->CreateFile 对应着文件的打开操作，当对安全目录中的文件有打开操作时，会相应的调用 FUSE 用户空间实现函数 MirrorCreateFile。FUSE 文件系统对 MirrorCreateFile 函数的定义如下。

```
NTSTATUS MirrorCreateFile(
    LPCWSTR             FileName,
    DWORD               DesiredAccess,
    DWORD               ShareMode,
    DWORD               CreationDisposition,
    DWORD               FlagsAndAttributes,
    PDOKAN_FILE_INFO    DokanFileInfo);
```

由于要确保安全目录中文件的保密性，所以需要在文件的打开阶段对未加密文件进行加密处理。应用程序进程对于安全目录文件的打开操作实现流程如图 4-4 所示，可以将这个过程分为如下几个步骤：

1）将应用程序进程对安全目录中文件的打开操作重定向到加密桥文件系统对应的虚拟磁盘中，从而转化为 FUSE 用户空间进程对安全目录中文件打开操作的实现。

2）获取打开的文件的属性，根据属性判断打开的是一个文件还是一个文件目录，如果打开的是一个文件目录，更新 flag，并根据 flag 创建文件目录，然后返回 STATUS_SUCCESS 完成打开操作，打开的如果是文件则进入下一步。

3）创建打开的文件，并根据文件大小信息，判断文件大小是否为 0，如果

文件大小为 0，则在配置文件中获取身份标识信息并准备好该文件的加密标识头，然后将加密标识头写入到文件开始位置，文件大小不为 0 则进入下一步。

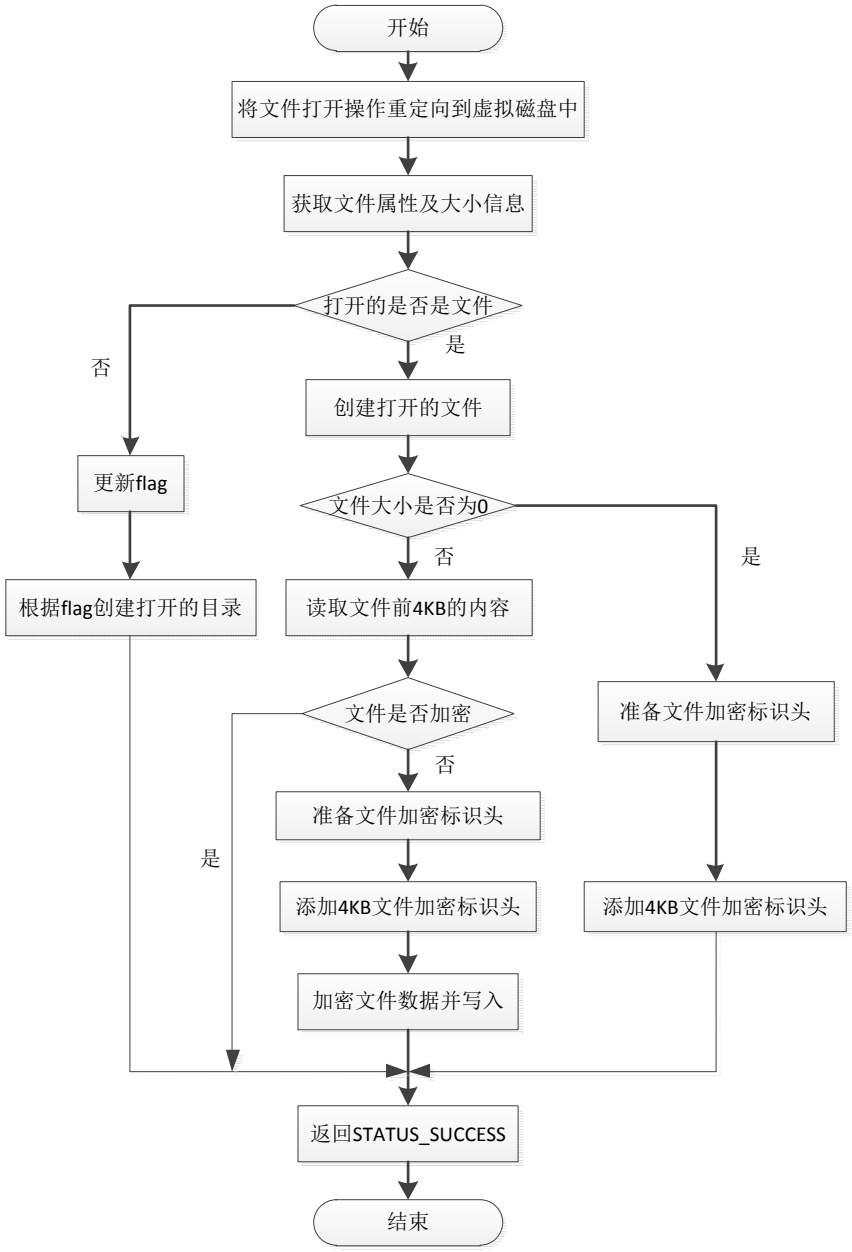

图 4-4 文件打开操作流程

4）读取文件前 4KB 的内容，根据加密标识和保存的 HASH 值来判断文件是否加密，判断过程如图 4-5 所示。如果文件已经加密则直接返回 STATUS_SUCCESS 完成打开操作，文件未加密则进入下一步。

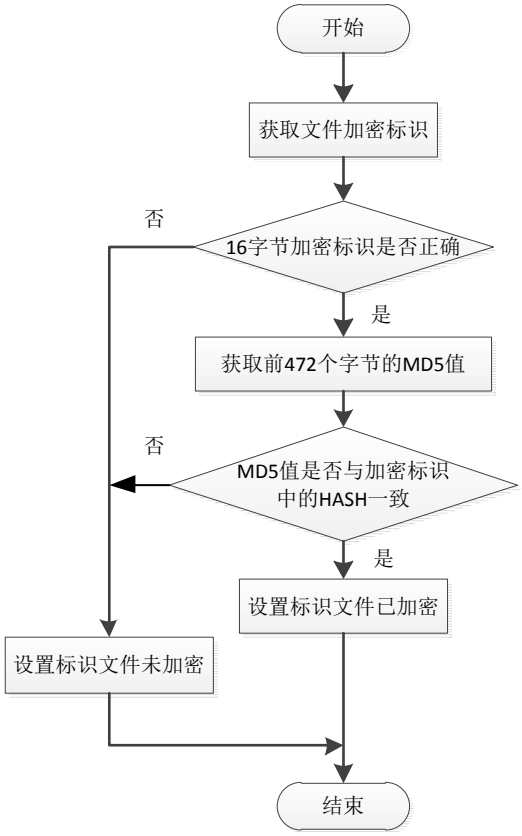

图 4-5 判断加密文件

5）对于未加密文件，首先要读取文件内容，然后在配置文件中获取身份标识信息并准备好相应的加密标识头，然后将文件加密标识写入到文件头部，根据加密标识中保存的对称密钥对文件内容进行加密处理，并将加密后的文件内容写入，最后返回 STATUS_SUCCESS 完成打开操作。

4.3.2 文件读操作处理

当一个应用程序进程对安全目录中的一个文件进行读操作时，会触发 I/O 管理器向安全目录所在的磁盘设备发送一个 IRP_MJ_READ 的请求，文件重定向过滤器模块会拦截这个 IRP 请求触发前所对应的 IRP_MJ_CREATE，将应用程序进程对安全目录中文件的读操作重定向到加密桥文件系统所映射的虚拟磁盘中，从而将应用程序对安全目录中文件的读操作转化为 FUSE 用户空间进程对安全目录中文件的读操作实现。

在 FUSE 用户空间进程中，回调函数 dokanOperations->ReadFile 对应着文件

的读操作，当对安全目录中的文件有读操作时，会相应的调用 FUSE 用户空间实现函数 MirrorReadFile，因此需要在 MirrorReadFile 函数中实现文件的透明解密，FUSE 文件系统对 MirrorReadFile 函数的定义如下。

```
NTSTATUS MirrorReadFile(
    LPCWSTR             FileName,
    LPVOID              Buffer,
    DWORD               BufferLength,
    LPDWORD             ReadLength,
    LONGLONG            Offset,
    PDOKAN_FILE_INFO    DokanFileInfo);
```

其中参数 Buffer 表示返回给应用程序进程的读取的数据，Buffer 中存储的为解密后的明文数据，参数 BufferLength 表示应用程序进程读取的数据长度，参数 ReadLength 表示实际读取的返回给应用程序进程的数据的长度，参数 Offset 表示读取文件时的文件偏移。

应用程序进程对文件读操作的实现流程如图 4-6 所示，可以将这个实现流程分为如下几个步骤：

1）将应用程序进程对安全目录中文件的读操作重定向到加密桥文件系统对应的虚拟磁盘中，从而转化为 FUSE 用户空间进程对安全目录中文件读操作的实现。

2）根据配置文件中的受信进程表判断读文件进程是否是受信进程，如果是非受信进程则直接读取密文数据，将数据保存到 Buffer 中并返回实际读取数据长度，如果是受信进程进入下一步。

3）判断此时的读文件的偏移 Offset 是否为 0，如果偏移量不为 0，则根据标识变量 IsEncrypt 判断文件是否加密，如果文件未加密则读取明文数据并将明文数据保存到 Buffer 中并返回数据长度，如果文件已经加密则将偏移 Offset 值加上文件加密标识头大小 4KB，读取文件数据并进行解密处理，将解密后的数据保存到 Buffer 中并返回数据长度。如果偏移量为 0 则进入下一步。

4）读取数据并取出前 4KB 的数据，根据这 4KB 的数据判断文件是否加密，如果文件未加密则将明文数据保存到 Buffer 中并返回数据长度，如果文件已经加密则进入下一步。

5）首先设置用于判断文件是否加密的标识变量 IsEncrypt 为 TRUE，并对读

取的数据进行解密，将解密后的数据保存到 Buffer 中并返回数据长度。

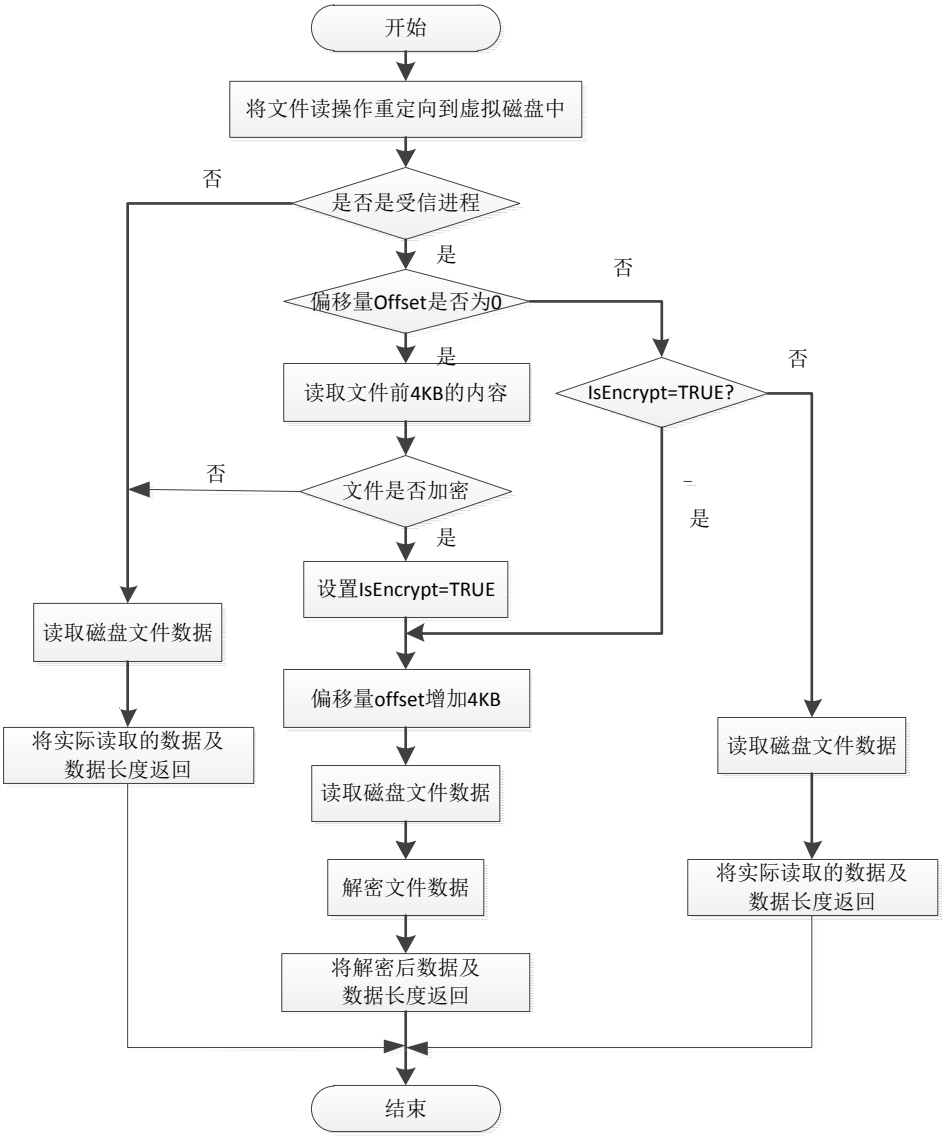

图 4-6 文件读操作流程

4.3.3 文件写操作处理

当一个应用程序进程对安全目录中的一个文件进行写操作时，会触发 I/O 管理器向安全目录所在的磁盘设备发送一个 IRP_MJ_WRITE 的请求，文件重定向过滤器模块会拦截这个 IRP 请求触发前所对应的 IRP_MJ_CREATE，将应用程序进程对安全目录中文件的写操作重定向到加密桥文件系统所映射的虚拟磁盘中，从而将应用程序对安全目录中文件的写操作转化为 FUSE 用户空间进程对安

全目录中文件的写操作实现。

在 FUSE 用户空间进程中，回调函数 dokanOperations->WriteFile 对应着文件的写操作，当对安全目录中的文件有写操作时，会相应的调用 FUSE 用户空间实现函数 MirrorWriteFile，因此需要在 MirrorWriteFile 函数中实现文件的透明加密，FUSE 文件系统对 MirrorWriteFile 函数的定义如下。

```
NTSTATUS MirrorWriteFile(
    LPCWSTR             FileName,
    LPCVOID             Buffer,
    DWORD               NumberOfBytesToWrite,
    LPDWORD             pNumberOfBytesWritten,
    LONGLONG            Offset,
    PDOKAN_FILE_INFO    DokanFileInfo);
```

其中参数 Buffer 表示即将要写入的数据，对于受信进程而言一般写入的数据为明文数据，而对于非受信进程而已写入的可能是明文数据也可能是密文数据，参数 NumberOfBytesToWrite 表示即将要写入数据的长度，参数 pNumberOfBytesWritten 表示实际写入磁盘的数据的长度，参数 Offset 表示写入文件的文件偏移。

应用程序进程对文件写操作的实现流程如图 4-7 所示，可以将这个实现流程分为如下几个步骤：

1）将应用程序进程对安全目录中文件的写操作重定向到加密桥文件系统对应的虚拟磁盘中，从而转化为 FUSE 用户空间进程对安全目录中文件写操作的实现。

2）根据配置文件中的受信进程表判断写文件进程是否是受信进程，由于非受信进程有可能写入的为密文数据，为了防止密文数据被二次加密，因此当写文件进程是非受信进程时需要判断写入的是密文还是明文，如果写入的是密文数据，可将该密文数据直接写入并返回相应的长度信息，写入的是明文数据则进入下一步。

3）对于写入的明文数据，首先要获取配置文件中的身份标识信息，准备文件加密标识头，并对明文数据进行加密，将加密后的密文数据写入磁盘并返回相应的长度信息。

4）对于受信进程而言，写入的数据一定是明文数据，根据第三步进行处理。

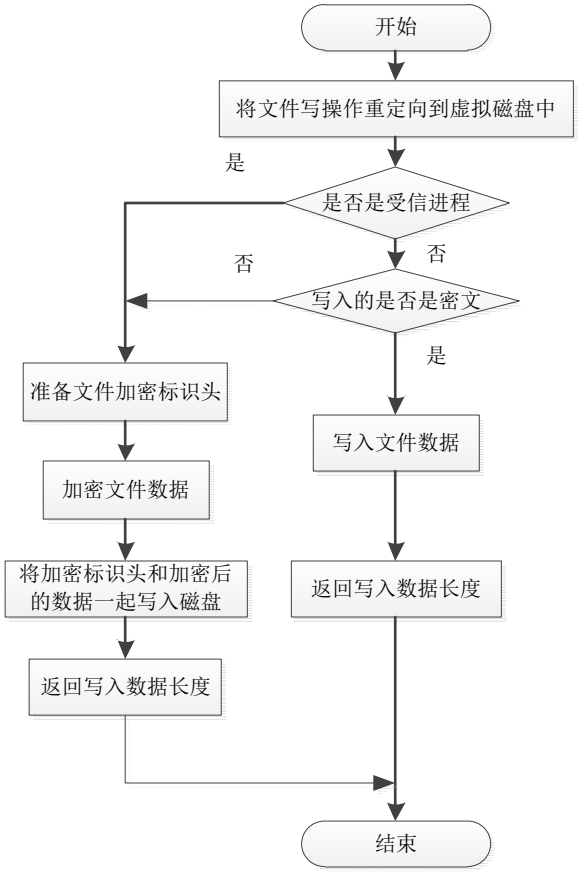

图 4-7 文件写操作流程

4.3.4 文件查询信息操作处理

在进程对安全目录中的文件进行读操作时，会首先查询文件的大小信息。当一个应用程序进程试图查询安全目录中的一个文件的信息时，会触发 I/O 管理器向安全目录所在的磁盘设备发送一个 IRP_MJ_QUERY_INFORMATION 的请求，文件重定向过滤器模块会拦截这个 IRP 请求触发前所对应的 IRP_MJ_CREATE，将应用程序进程对安全目录中文件的查询信息操作重定向到加密桥文件系统所映射的虚拟磁盘中，从而将应用程序对安全目录中文件的查询信息操作转化为 FUSE 用户空间进程对安全目录中文件的查询信息操作实现。

由于安全目录中的文件都是添加了文件加密标识头的加密文件，因此在受信进程查询文件大小信息时需要隐藏文件加密标识头的大小。其中查询文件大小信息对应的是 FUSE 用户空间程序中的 MirrorGetFileInformation 函数实现，该函数的定义如下。

```
NTSTATUS MirrorGetFileInformation(
    LPCWSTR                     FileName,
    LPBY_HANDLE_FILE_INFORMATION HandleFileInformation,
    PDOKAN_FILE_INFO            DokanFileInfo);
```

其中参数 HandleFileInformation 是一个用于存储文件信息的结构体，通过函数查询得到的文件信息需要存入该结构体中返回。本系统主要关注的是文件的大小信息，其他信息直接返回。由于受信进程要读取得到解密后的明文数据，因此，当受信进程进行查询文件信息操作时，需要修改文件大小信息，减去文件加密标识头的大小。这样对于受信进程看来，文件的大小就不包含文件加密标识头了，达到隐藏文件加密标识头的目的。非受信进程直接读取包含文件加密标识头的密文数据，所以当非受信进程进行查询文件信息操作时，无需修改文件大小信息。

4.3.5 文件设置信息操作处理

在进程对安全目录中的文件进行写操作完成后，会设置文件的大小信息。当一个应用程序进程试图设置安全目录中的一个文件的大小信息时，会触发 I/O 管理器向安全目录所在的磁盘设备发送一个 IRP_MJ_SET_INFORMATION 的请求，文件重定向过滤器模块会拦截这个 IRP 请求触发前所对应的 IRP_MJ_CREATE，将应用程序进程对安全目录中文件的设置信息操作重定向到加密桥文件系统所映射的虚拟磁盘中，从而将应用程序对安全目录中文件的设置信息操作转化为 FUSE 用户空间进程对安全目录中文件的设置信息操作实现。

与查询文件信息相对应，当受信进程试图重新设置加密文件的大小，也必须有针对性地对文件的大小做出一些特殊改变。设置文件信息对应的是 FUSE 用户空间程序中的 MirrorSetEndOfFile 函数实现和 MirrorSetAllocationSize 函数实现，用于设置文件的 EndOfFile 和 AllocationSize，这两个函数的定义如下。

```
NTSTATUS MirrorSetEndOfFile(
    LPCWSTR             FileName,
    LONGLONG            ByteOffset,
    PDOKAN_FILE_INFO    DokanFileInfo);


NTSTATUS MirrorSetAllocationSize(
```

```
    LPCWSTR             FileName,
    LONGLONG            AllocSize,
    PDOKAN_FILE_INFO    DokanFileInfo);
```

其中 MirrorSetEndOfFile 函数中的 ByteOffset 参数表示设置的 EndOfFile，即文件的实际大小，MirrorSetAllocationSize 函数中的 AllocSize 参数表示设置的 AllocationSize，即文件的占用空间。当受信进程对文件进行设置文件信息操作时，需要将 EndOfFile 和 AllocationSize 的大小加上文件加密标识头的大小，这样就受信进程看来，文件的大小就包含了文件加密标识头。非受信进程在设置文件信息时，无需加上文件加密标识头的大小。

4.4 文件加解密实现

在对安全目录中的文件进行打开操作，读写操作等操作时需要对文件进行加解密处理，对此，本节将采用 AES-ECB 模式、AES-CTR 模式、HASH-CTR 模式这三种块加密工作模式完成对安全目录中文件的加解密。

4.4.1 采用 AES-ECB 模式的文件加解密

本文中的透明文件加密系统对文件的加解密都是采用分块加密算法的，所谓分块加密算法，是一种对称密钥算法。它将明文数据分成很多个等长的数据块（block），使用确定的文件加密算法和对称密钥对每个数据块分别进行加密和解密操作。本文所使用的是 AES 分块加密算法。

本小节采用的是 ECB 块加密工作模式对文件进行加解密处理，所谓 ECB 块加密工作模式，即为电子密码本（Electronic Codebook，ECB）模式。需要将加密的数据按照块加密长度分为多个数据块，并对每一个数据块进行单独加密[43]。本文中使用的 AES 块加密是以 16 个字节为块加密的长度的，因此对文件最后一块不满 16 个字节的数据不能使用 AES 块加密，本文采取的解决方案是，将对称密钥进行 HASH 之后与最后一块不满 16 字节的文件内容进行按位异或来进行加密。AES-ECB 模式的加密流程如图 4-8 所示。

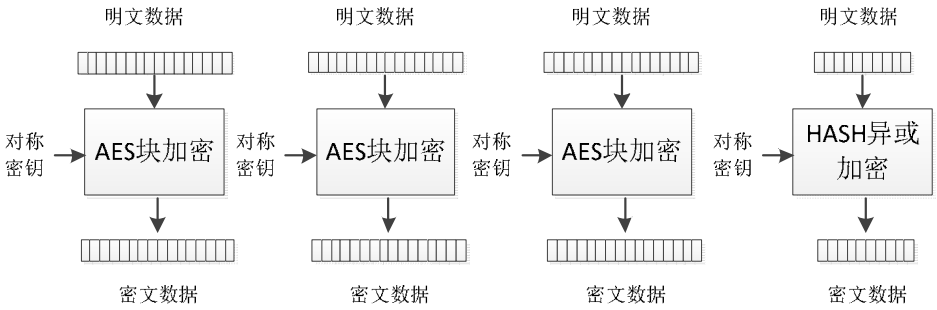

图 4-8 AES-ECB 模式加密

需要特别注意的是：即将要写入的需要加密的明文数据 Buffer 的长度 NumberOfBytesToWrite 可能也不是块加密长度的整数倍，也无法使用 AES-ECB 模式对数据进行加密处理，这就需要将数据 Buffer 进行截断处理，截断后的数据刚好是块加密长度的整数倍，截断的剩余碎片信息就保存在一个 map 表中,这些碎片相邻的一部分的长度和刚好是块加密长度的整数倍，在文件进行 Flush 或者 Close 时将这些碎片拼接成块加密长度的整数倍通过 AES-ECB 加密后再写入文件。对数据进行截断处理的示例过程如图 4-9 所示。

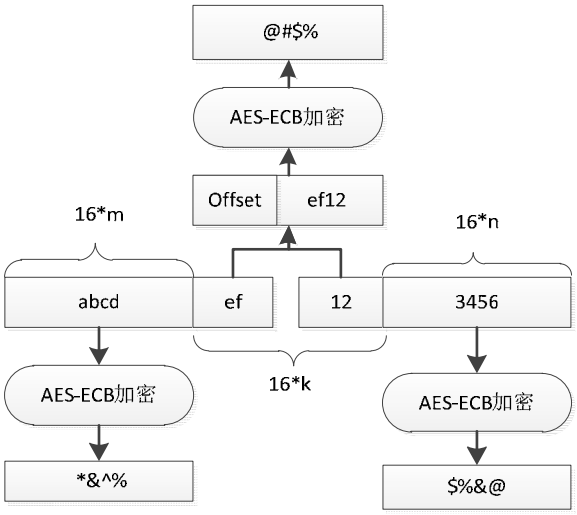

图 4-9 数据截断处理过程

采用 AES-ECB 模式对文件进行解密时，同样是以 16 字节为块解密长度来进行处理的，对最后一块不满 16 字节的文件数据同样采用 HASH 异或方式进行解密处理。AES-ECB 模式的解密流程如图 4-10 所示。

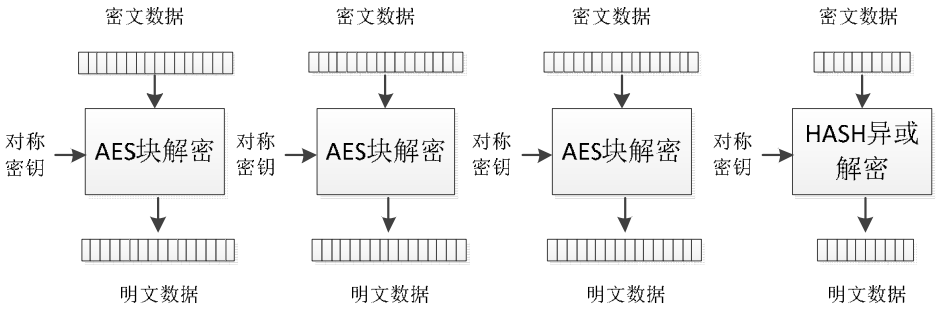

图 4-10 AES-ECB 模式解密

需要特别注意的是：需要解密的密文数据长度 BufferLength 可能不是块加密长度的整数倍，读取文件时偏移量 Offset 也可能不在块加密长度整数倍边界上，这样就不能直接使用 AES-ECB 模式进行解密处理，这就需要首先将 Offset 和 BufferLength 进行扩展处理，扩展后的数据是块加密长度的整数倍，此时就可以将整块数据通过 AES-ECB 模式来进行解密了，最后再截取需要返回的解密后的明文数据和数据长度即可。数据扩展处理示例过程如图 4-11 所示。

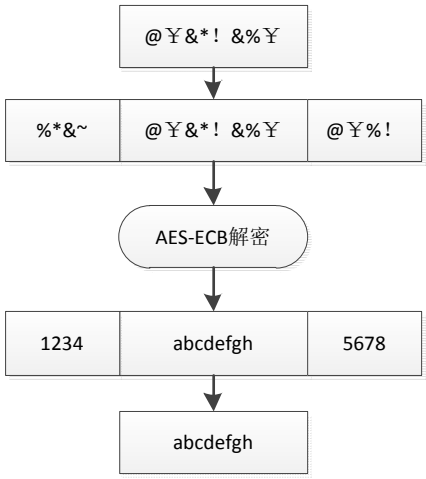

图 4-11 数据扩展处理过程

AES-ECB 加密模式的优点是实现简单，具有良好的差错控制机制。AES-ECB 加密模式需要对最后一块不满块加密长度的数据进行异或加密处理，这种处理方式实际上是不安全的。AES-ECB 加密模式的需要进行填充的机制，使得它对任意随机长度的文件的加解密无能为力，这是因为，明文数据只要有一个字节的不同，加密后的密文数据会完全不同，在进行随机存取加解密的情况下，很难恢复出原始的明文数据。

4.4.2 采用 AES-CTR 模式的文件加解密

在采用 AES-ECB 加密模式对文件进行加解密时，在数据大小不是块加密长度整数倍时需要对文件数据进行特殊处理，这样处理起来比较麻烦，这是由于 AES-ECB 加密模式不支持数据的随机存取所导致的。本小节将采用 AES-CTR 加密模式对文件进行加解密处理，省去了对文件数据进行扩展或者截断的麻烦。

本小节采用的是 CTR 块加密工作模式对文件进行加解密处理，所谓 CTR 块加密工作模式，即计数器模式（Counter Mode）[43]。CTR 块加密工作模式将块密码变为流密码。它通过一个随着文件偏移量不断递增的加密计数器以产生连续的密钥流。

采取 AES-CTR 模式进行加密时，首先需要产生一个 16 字节的初始化向量，然后根据文件长度将文件分成以块加密长度为基本单位的很多块，从 0 开始对每一块进行计数，然后将初始化向量与这个计数值进行异或操作得到一个仍然为 16 字节的数据，将异或后的数据进行 AES 加密，将加密后的数据与明文数据进行异或即可得到加密的密文数据。AES-CTR 模式加密流程如图 4-12 所示。因此，AES-CTR 加密模式支持随机存取的文件加解密。

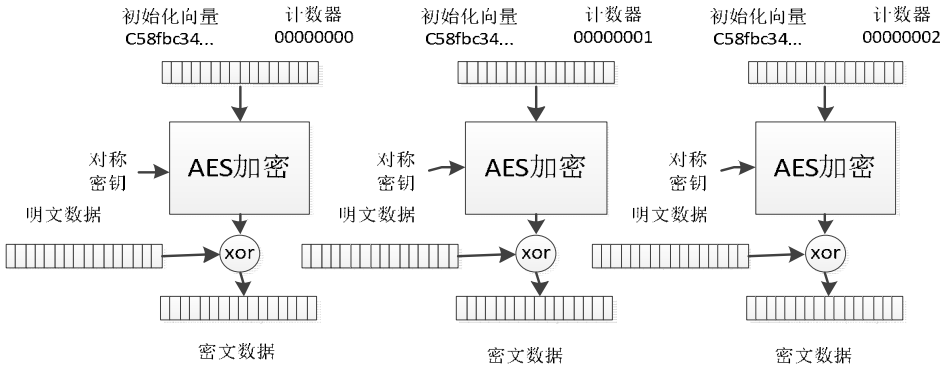

图 4-12 AES-CTR 模式加密

采取 AES-CTR 模式进行解密时，需要与加密文件数据时相同的 16 字节的初始化向量，将初始化向量与数据块计数值进行异或操作得到一个仍然为 16 字节的数据，将异或后的数据进行 AES 加密，将加密后的数据与密文数据进行异或即可得到解密的明文数据。AES-CTR 模式解密流程如图 4-13 所示。

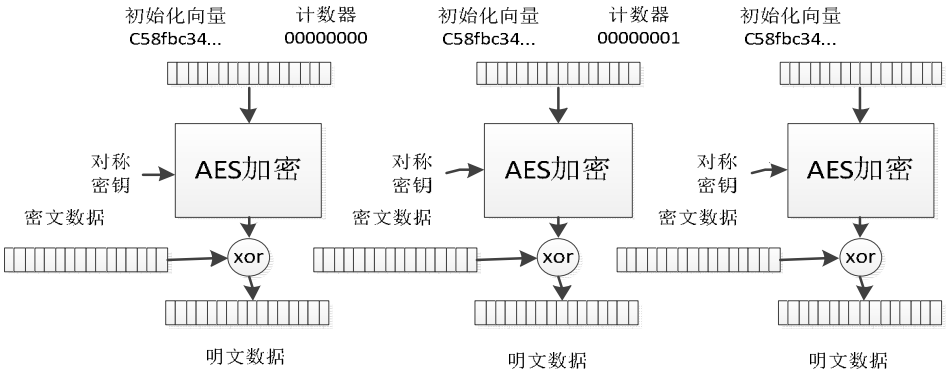

图 4-13 AES-CTR 模式解密

4.4.3 采用 HASH-CTR 模式的文件加解密

本小节在 AES-CTR 加密模式的基础上进行了一些改变，将 AES 加密算法替换成 HASH 算法，但仍然采取 CTR 块加密模式，即为 HASH-CTR 模式。本小节所采取的 HASH 算法为 SHA1（Security Hash Algorithm）算法，它具有不可以从消息摘要中复原信息，两个不同的信息不会产生同样的消息摘要的特别。采取 HASH-CTR 模式对文件进行加解密与 AES-CTR 模式类似，同样省去了对文件数据进行扩展或者截取的麻烦。

采取 HASH-CTR 模式进行加密时，无需产生一个额外的初始化向量，只需将对称密钥与文件块计数值进行字符串的拼接处理，然后取其 SHA1 值，将得到的 SHA1 值与明文数据进行异或处理，就能得到加密后的密文数据。HASH-CTR 模式的加密流程如图 4-14 所示。

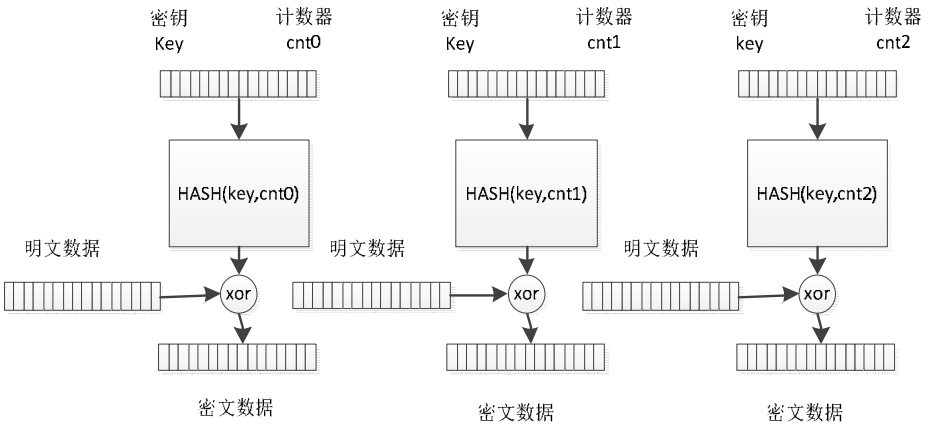

图 4-14 HASH-CTR 模式加密

采取 HASH-CTR 模式进行解密时，同样无需产生一个额外的初始化向量，只需将对称密钥与文件块计数值进行字符串的拼接处理，然后取其 SHA1 值，将得到的 SHA1 值与密文数据进行异或处理，就能得到解密后的明文数据。HASH-CTR 模式的解密流程如图 4-15 所示。

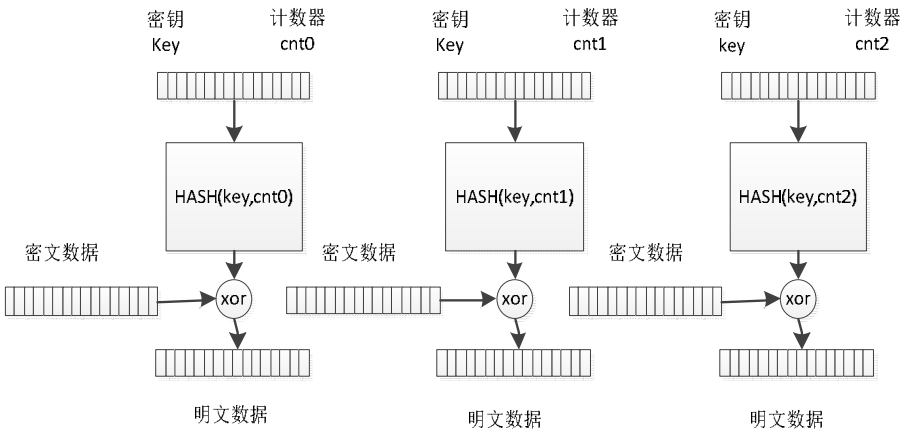

图 4-15 HASH-CTR 模式解密

采取 HASH-CTR 模式省去了产生初始化向量的麻烦，而且不需要将密钥与文件块计数值处理后的数据大小控制在 16 个字节，实现起来比较简单。

4.5 采用缓存方式的文件读写操作实现

采用上述几种块加密工作模式对文件的读写操作进行处理时，需要针对文件的反复多次数的读写进行文件数据处理以及加解密处理，这种需要多次与磁盘进行数据交互的方式会严重影响文件读写效率。因此，本节提出了一种带缓存的文件读写操作处理方案，大大提高了文件读写效率。本节采用 AES-CTR 加密模式对带缓存的文件读写操作进行了实现。

4.5.1 缓存结构设计

在文件进行读写操作处理时添加缓存机制，目的在于提高文件读写的效率。由于进程对文件进行读写操作时不一定是按照顺序进行读写的，也不一定每次读写的数据大小都是固定的，因此，在设计缓存结构时，不仅要从提高文件读写效率这方面去考虑，而且要考虑文件读写数据的随机性。

基于以上的设计原则，本文将缓存结构设计为一个 map 存储表，再定义一个宏定义 CACHE_BLOCK_NUM，用于限定 map 表中存储的缓存块个数。map 表中存储的是一个保存有缓存信息的结构体 DataInfo，其中 DataInfo 结构体的定义为：

```
typedef struct DataInfo
{
    LONGLONG offset;
    int len;
    WCHAR filePath[256];
    UCHAR readCache[CACHE_LENGTH];
    BOOL dirty;
}DataInfo,*pDataInfo;
```

其中 offset 表示缓存块在文件中的偏移量，len 表示实际缓存的缓存数据长度，filePath 表示缓存文件的文件全路径，readCache 用于存储缓存数据，其中 CACHE_LENGTH 是一个宏定义，用于限定一个缓存块的数据长度，并可以根据实际使用情况进行调整，dirty 用于标识缓存是否被修改，如果缓存被修改，dirty 被设置为 TRUE。综上所示，本文缓存的结构设计如图 4-16 所示。

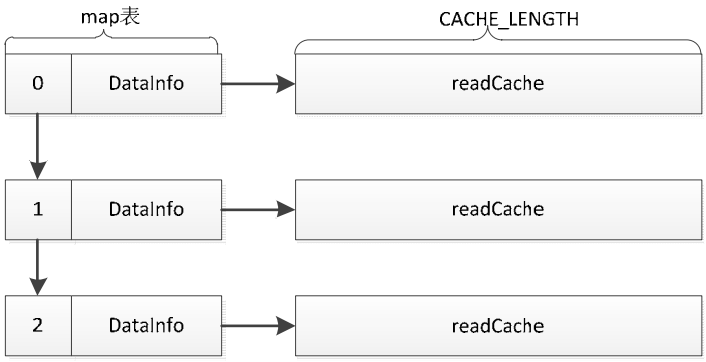

图 4-16 缓存结构

4.5.2 文件读操作的缓存机制

当进程对文件进行读操作时，会发起很多次读文件请求，每次读文件请求都会从磁盘读取文件数据然后再进行解密返回，这样就会导致读文件效率低下，

当读取大文件时尤为明显。本文提出的缓存机制，就是在读文件时，首先将文件数据解密后存入缓存 map 表中，之后再发起读文件请求时，如果读取的数据在缓存中，则直接返回，这样不仅省去了从磁盘重新读取数据的操作，而且也省去了对数据的解密操作，通过这种“预读”的方式，大大提高了读文件效率。

本文将缓存块数设置为 CACHE_BLOCK_NUM ，每个缓存块都对应着一个编号，当缓存的个数超过这个数字时，就从编号为 0 的那块缓存开始重新缓存，通过这种方式管理缓存可以提高缓存管理效率。那么怎么判断需要读取的文件数据是否在缓存中呢，读取的文件数据与某一块缓存的位置关系如图 4-17 所示。

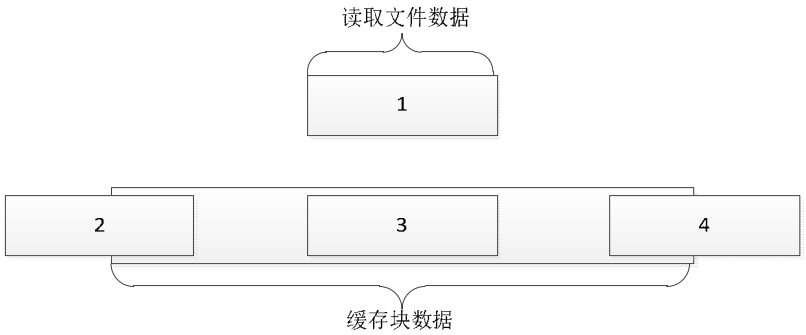

图 4-17 文件数据与缓存块位置关系

图 4-14 中描述了文件数据与缓存块的四种位置对应关系，文件数据处于缓存块的不同位置就需要进行不同的处理。当文件数据处于图示位置 1 时，表示文件数据不在缓存中，此时需要从磁盘读取包括位置 1 数据的长度为 CACHE_LENGTH 的数据，将该数据进行解密并存入缓存块中，然后返回位置 1 处的明文数据。当文件数据处于图示位置 2 和 4 时，表示文件数据只有一部分在缓存中，此时需要从磁盘中读取包括位置 2 和 4 不在缓存中的那部分数据的长度为 CACHE_LENGTH 的数据，同样将数据进行解密并存入缓存块中，然后将在缓存中的那部分和不在缓存中的那部分进行拼接后返回。当文件数据处于图示位置 3 时，说明文件数据全部在缓存中，此时直接从缓存中返回相应的数据即可。文件读时的缓存处理流程如图 4-18 所示。

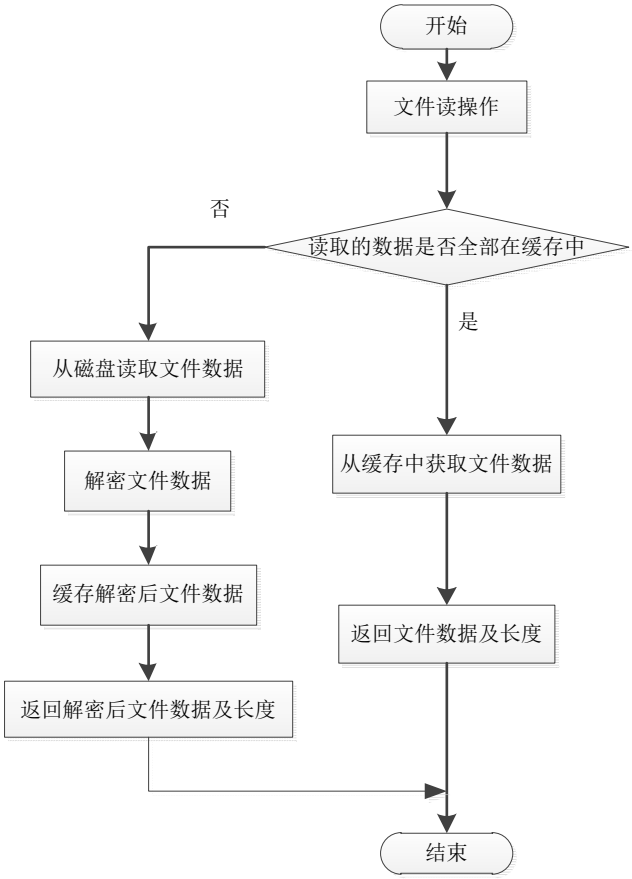

图 4-18 文件读缓存机制

4.5.3 文件写操作的缓存同步

需要特别注意的是：当一个受信进程在读文件时，可能存在一个非受信进程同时对文件进行写操作。由于我们使用了缓存机制，出现上述这两种可能情况时，缓存中保存的依然是写之前的数据，此时缓存中数据得不到更新，受信进程最终读取的就是原始的未更新的数据，这就导致读写数据错误。

为了避免上述错误的发生，本文采取了文件读写时进行缓存同步的方案。对于文件读操作时缓存的文件数据，如果写入的文件数据范围刚好在缓存块中，此时就需要将写入的数据更新到缓存的相应块中，保持读写数据的缓存同步，并将 dirty 标志位置为 TRUE，表示进行了缓存同步，原有缓存数据被修改，需要写入的数据并不是直接加密后写入磁盘的，本文采取一种“延迟写”的方案，当文件进行 Flush 或 Close 操作或者缓存内容即将被重新缓存时，将标志位 dirty 被置为 TRUE 的缓存块加密后写入磁盘，这样做的好处是减少了文件数据写入

磁盘的次数，提高了写文件时的效率。如果写入的文件数据不在缓存中，直接将文件数据加密后写入磁盘。文件写操作的缓存同步过程如图 4-19 所示。

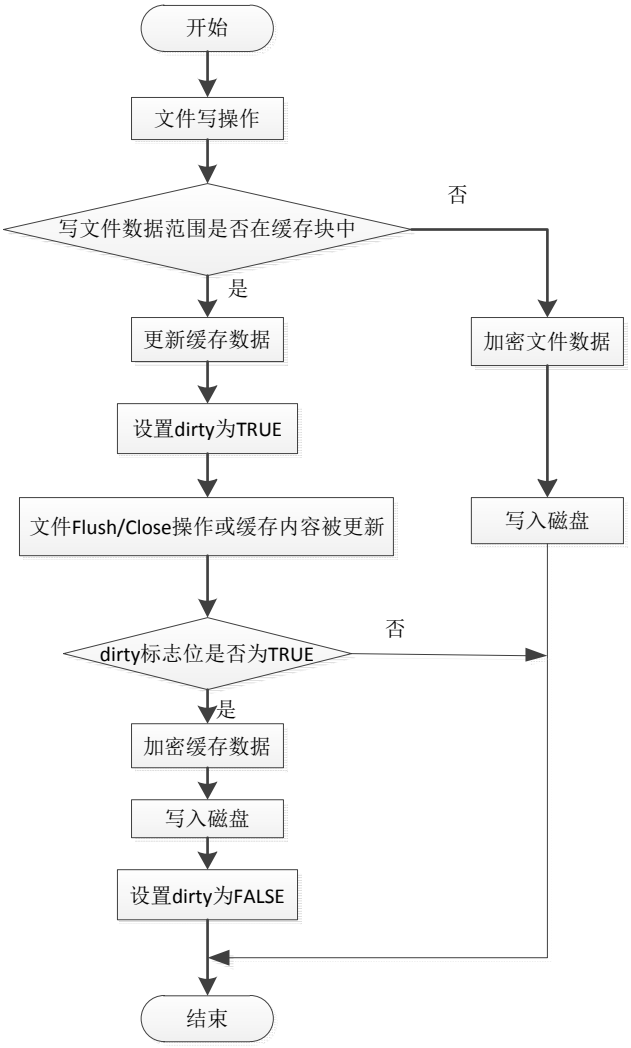

图 4-19 文件写缓存同步

4.6 本章小结

本章主要设计与实现了加密桥文件系统模块，首先对加密桥文件系统的核心 FUSE 文件系统进行了技术分析，然后对受信进程与非受信进程的设别技术进行了详细的方案设计与实现，最后在 FUSE 用户空间程序中对几种文件操作进行了处理，实现了文件的透明加解密，同时还设计与实现了缓存机制，提高文件读写效率。

第 5 章 系统安全加固模块的实现

系统安全加固模块主要是针对本系统可能存在的安全问题，提出相应的解决方案。本章实现了针对安全目录文件的防复制防截屏，并对受信进程进行了防伪处理，同时对配置文件进行了安全保护，对系统提供了安全加固。

5.1 防复制防截屏实现

本文中透明文件加密系统的防复制防截屏模块，是在不影响用户正常使用的前提下，对安全目录中受信进程打开的文件实现防复制防截屏，其中涉及到剪贴板监控技术、进程技术、桌面窗口监控技术。

5.1.1 剪贴板监控

本文采用操作剪贴板监听链的方式来实现剪贴板监控。通过监控进程即时监控剪贴板中的内容变化，一旦发现剪贴板中出现了非法内容随时清空剪贴板中的相应数据，系统过滤的基本原则是：不允许剪贴板中出现文本格式或图片格式的文件，一旦发现立即清空剪贴板。剪贴板的监控流程如图 5-1 所示。

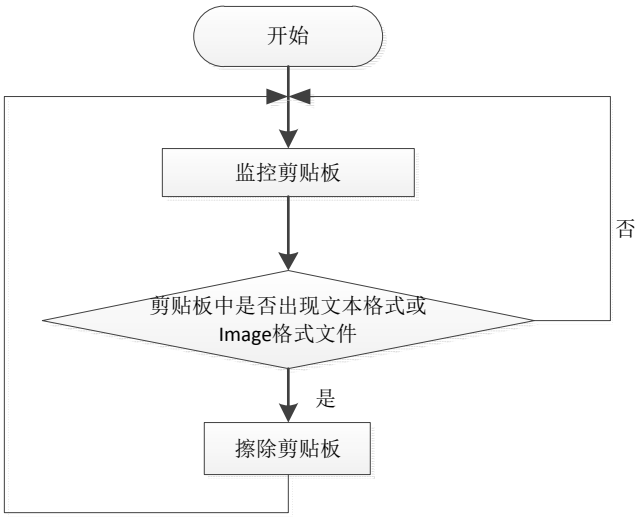

图 5-1 剪贴板监控

5.1.2 防复制防截屏控制

本文的防复制防截屏模块，是将剪贴板监控技术和进程技术相结合，对于受信进程打开的安全目录中的文件窗口实现防复制防截屏，保护机密文件数据不被窃取。本模块主动侦测受信进程在安全目录下对文件的打开操作，只要有安全目录中的文件处于打开状态，剪贴板监控进程保持启动，此时用户无法进行复制截屏操作，当安全目录中的文件全部关闭时，剪贴板监控进程自动关闭，此时用户可以正常使用复制截屏功能。该模块在不影响用户的正常使用的前提下，对安全目录中的文件提供了保护。

防复制防截屏模块的执行流程如图 5-2 所示，可以将该模块的实现分为如下几个步骤：

1）主动侦测进程的打开操作，并通过获得进程命令行参数的方式解析出进程所打开的文件的路径。

2）判断打开的文件是否是安全目录中的文件，如果是则进入下一步，如果不是则不进行任何操作，用户可以进行正常的复制截屏操作。

3）判断打开文件的进程是否是受信进程，如果是则进入下一步，如果不是则不进行任何操作，用户可以对非受信进程打开的文件进行正常的复制截屏操作。

4）获得并保存进程打开的文件窗口的句柄，同时判断进程所对应的文件窗口是否处于打开状态，如果是则开启剪贴板监控进程，此时用户不能对打开的文件进行复制截屏操作，如果不是则不进行任何操作，允许用户进行正常的复制截屏操作。

5）根据上一步中保存的打开的文件窗口的句柄，判断这些打开的文件窗口是否全部被关闭，如果是则关闭剪贴板监控进程，此时没有安全目录中的文件被打开，用户可以正常的进行复制截屏操作，如果不是则剪贴板监控进程始终保持开启状态，用户无法进行复制截屏操作。

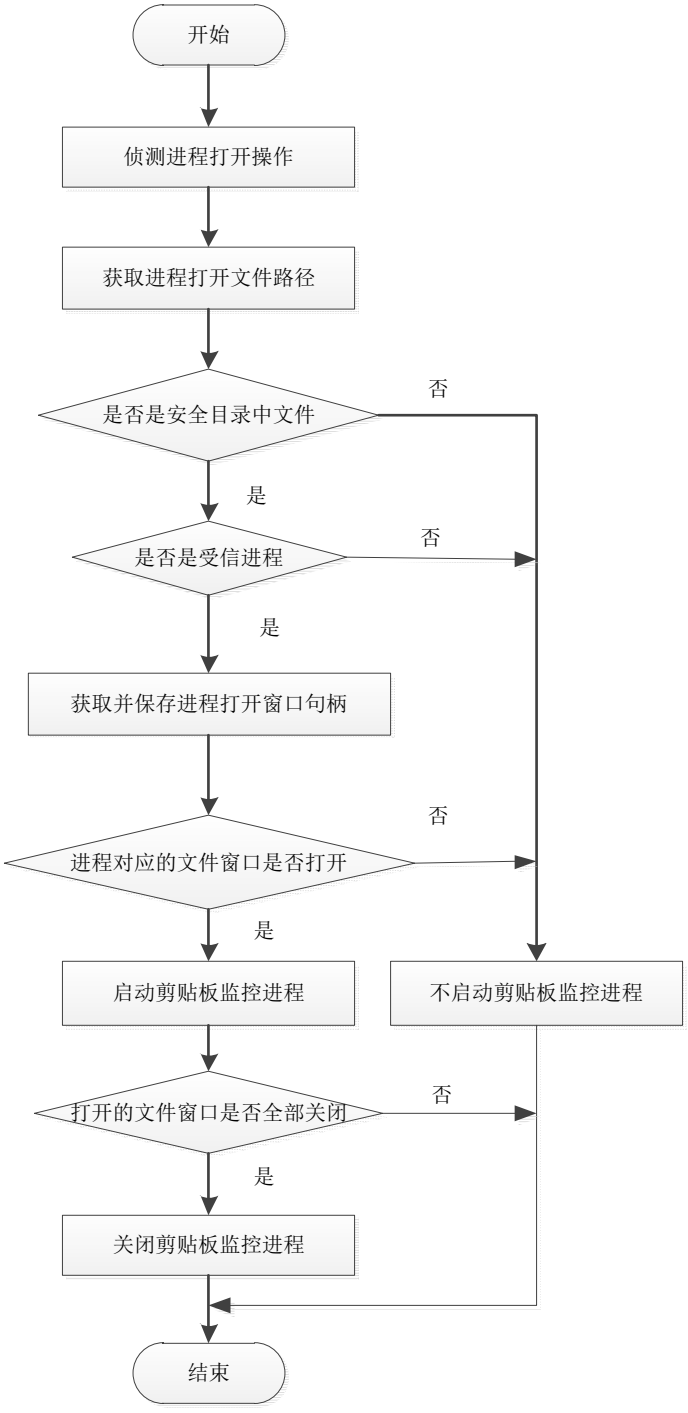

图 5-2 防复制防截屏程序流程

5.2 防止进程欺骗实现

对受信进程和非受信进程是通过操作的文件的文件后缀名和操作进程的进

程名来进行识别的，而操作进程的进程名是可以被任意更改的。但是如果有用户将恶意进程的进程名字改为受信进程名，系统就无法鉴别而会把这个改了进程名的恶意进程当做是受信进程来进行处理，最终可能导致机密数据泄漏。例如：对于文件后缀名为 docx 的文件，WINWORD.EXE 进程是对应的受信进程，如果将腾讯 QQ 对应的进程 QQ.exe 的进程名修改为 WINWORD.EXE，腾讯 QQ 会被当做是受信进程来进行处理，此时若将安全目录中的机密文件通过 QQ 发送出去，腾讯 QQ 的进程读取安全目录文件时会自动解密，接收方得到的就是解密后的明文，此时机密文件被泄漏，这就是由于进程欺骗导致的泄密的发生。

因此，在判断受信进程和非受信进程时需要验证进程的真实性，一旦发现有恶意进程冒充受信进程窃取机密数据时，系统自动弹出相应警告提示存在仿冒进程并且关闭系统。本文根据受信进程所对应的可执行文件的 HASH 验证来判断可执行文件是否对应的是真实的受信进程。

本文是通过对进程所对应的可执行文件获取 MD5 值来验证进程的真实性的。MD5（Message-Digest Algorithem 5）即信息摘要算法，由于其具有抗修改性和强抗碰撞性，因此被广泛运用于加解密技术中，它可以被认为是文件的数字标识。对于任何一个文件，无论是文本文件还是可执行文件或者是其他任何类型的文件，都有一个唯一的 MD5 值，这个 MD5 值就像是这个文件的身份证号一样，假如这个文件被修改，哪怕只修改一个字节，得到的 MD5 值都会有很大的区别。因此，可以通过对比同一文件的所产生的 MD5 值是否相同，来判断这个文件是否被篡改。下表 5-1 列出了几种常见进程的 MD5 值。

表 5-1 几种常见进程的 MD5 值

进程名	MD5 值
notepad.exe	b32189bdff6e577a92baa61ad49264e6
wordpad.exe	715bff236158f61c042928a53c0d5aa8
WINWORD.EXE	ceaa5817a65e914aa178b28f12359a46
wps.exe	d4cc94d4e1ea6dd94ce4b575f8e2cca5

本文根据指定的受信进程的可执行文件生成对应的 MD5 值，并将指定的受信进程与 MD5 值建立一个 MD5 验证表，然后存入到一个配置文件中，通过配置文件来进行统一的管理。为了防止恶意用户将恶意进程及其对应的 MD5 值替换原有受信进程及其 MD5 值，本文中的配置文件在驱动层进行了隐藏，恶意用

户无法得知配置文件路径，因而无法对配置文件进行恶意的更改，防止因配置文件被更改而导致的泄密。受信进程的真实性验证流程如图 5-3 所示，将这个过程分为如下几个步骤：

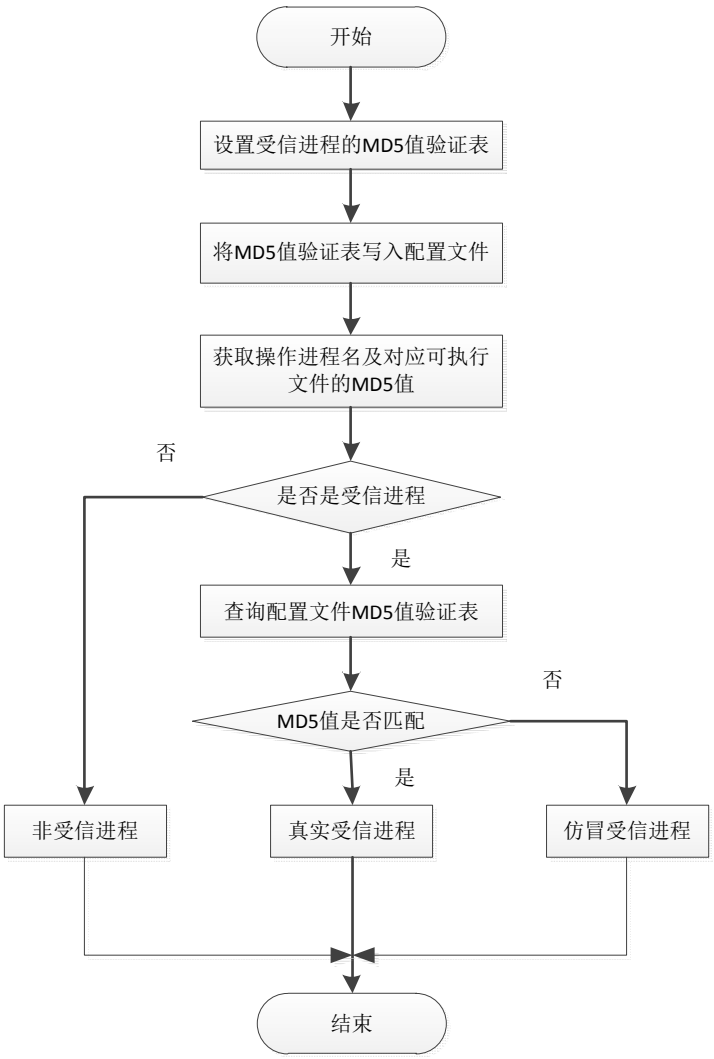

图 5-3 进程防伪处理过程

5.3 配置文件安全保护

配置文件中记录了受信进程表信息和进程 MD5 验证表信息，这两个信息决定了一个进程能否获得加密文件的明文数据，由此可见，需要对本文中的配置文件进行安全保护，以防止配置文件中的重要信息被非法篡改而导致泄密。系统对配置文件的保护是通过两个方面来实现的，第一，在内核层对配置文件进

行隐藏处理；第二，在应用层对配置文件进行签名验证。

传统的文件+S +H 隐藏或者是通过鼠标右键操作隐藏与驱动隐藏相比，无法实现对文件真正的隐藏，是可以通过系统设置将隐藏的文件显示出来的，如果通过驱动层来实现文件隐藏，一旦系统加载了隐藏文件驱动程序，需要隐藏的文件会在文件系统中被过滤掉，被隐藏的文件通过任何方式都无法被遍历出来。在驱动层实现文件的隐藏，主要是在 IRP_MJ_DIRECTORY_CONTROL 这个 IRP 请求中进行处理的。

在驱动的入口函数 DriverEntry 中，MINIFILTER 驱动注册了与 IRP_MJ_DIRECTORY_CONTROL 相关的后操作回调函数，注册回调函数使用的是 FltRegisterFilter 函数。FLT_OPERATION_REGISTRATION 结构体中记录了相应的 IRP 和对应的回调函数信息，并作为函数的一个参数传入到 FltRegisterFilter 函数中，其结构如下：

```
CONST FLT_OPERATION_REGISTRATION Callbacks[] =
{
    { IRP_MJ_DIRECTORY_CONTROL,
      0,
      NULL,
      HideFilePostDirCtrl },
    { IRP_MJ_OPERATION_END }
};
```

当应用层发起 IRP_MJ_DIRECTORY_CONTROL 这个 IRP 请求时，驱动程序就会调用注册的回调函数，其中不同的操作系统返回的 FileInformationClass 结构是不一样的。文件隐藏的重点就是获取 FLT_CALLBACK_DATA 结构中的缓冲地址 Data->Iopb->Parameters.DirectoryControl.QueryDirectory.MdlAddress 或者 Data->Iopb->Parameters.DirectoryControl.QueryDirectory.DirectoryBuffer 中的内容，需要隐藏的文件信息就在其中，找到需要隐藏的文件信息，更改与文件相关的前驱结点中指向下一个结点的偏移量，直接跳过需要隐藏的文件的文件结点，达到隐藏目的。

如果恶意用户获取了配置文件的路径，即使配置文件在驱动层被隐藏，恶意用户仍然可以通过修改配置文件获取机密信息。本文系统在驱动层隐藏配置文件的同时，对配置文件进行 RSA 签名验证处理，只要配置文件被恶意修改，

签名验证便会不成功，此时无论是受信进程还是非受信进程都只能获取安全目录中文件的密文数据，有效的防止了因恶意修改配置文件而导致的泄密。系统对配置文件的签名验证过程如图 5-4 所示。

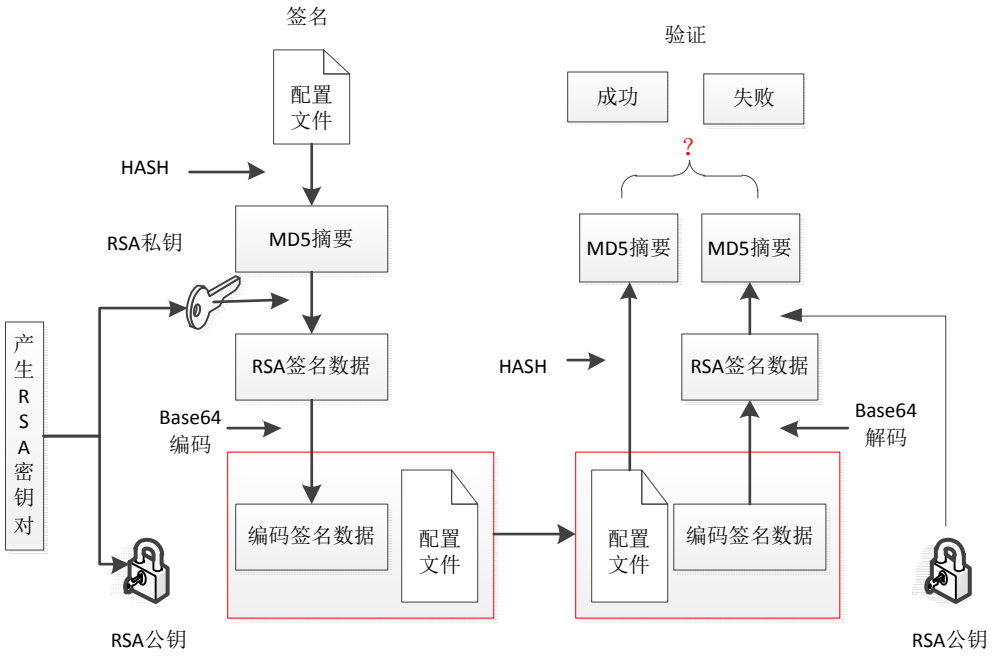

图 5-4 配置文件签名验证

在对配置文件进行签名时，首先针对配置文件的信息进行哈希处理，得到配置文件信息的 MD5 摘要，然后产生 RSA 密钥对，利用 RSA 私钥对 MD5 摘要进行数字签名，由于经过 RSA 签名后的数据是不可识别的乱码，所以需要将签名数据进行 Base64 编码处理。将编码签名数据，配置文件封装成经过 RSA 数字签名的签名文件。对配置文件进行验证时，首先将签名文件中的签名数据进行 Base64 解码，还原出原始的签名数据，然后利用 RSA 公钥验证签名是否成功。

5.4 本章小结

本章主要设计与实现了系统安全加固模块，通过剪贴板监控技术，实现针对安全目录中打开文件的内容的防复制与文件窗口的防截屏，并对受信进程进行了 HASH 验证，防止因恶意进程冒充受信进程而导致的泄密。在内核层和应用层实现了配置文件的双重安全保护，防止因配置文件被修改而导致的泄密。

第 6 章 系统测试与分析

本章将完成本文中透明文件加密系统的功能测试和性能测试分析。对系统的功能性测试主要包括透明加解密功能测试，进程控制策略功能测试，防止进程欺骗功能测试以及防复制防截屏功能测试。对系统性能测试主要包括三种加密模式性能测试与分析，缓存方式与非缓存方式读写性能测试与分析。

6.1 系统测试环境

由于本文开发的是基于 Windows 平台的透明文件加密系统，因此本文选定目前广泛使用的 Windows 7 操作系统作为系统测试环境，由于本文中系统对计算机硬件配置要求不高，本文在虚拟机中安装 Windows 7 操作系统进行系统的功能测试和性能测试。由于文件类型繁多，为了避免重复和累赘，本文选定 Office 文档和记事本对应的 txt 文档作为代表对系统的功能进行测试。

除了操作系统环境和相应的测试软件外，还需要配置系统的运行环境。本文指定本地磁盘的一个目录作为安全目录，基于此安全目录对系统功能及性能进行测试分析。本文首先通过鼠标右键菜单设置目录“C:\x\y”作为测试的安全目录，然后将该目录设置为加密桥文件系统虚拟磁盘的映射目录，这样系统的运行环境就基本配置完成了。

6.2 系统功能测试与分析

6.2.1 透明加解密测试

本小节主要测试本文中的透明文件加密系统是否能够在本地完成正常的透明加解密工作，将加密文件通过 QQ，E-mail 等网络方式传输出去时能否导致文件泄密。本小节选定 txt 文档作为代表来测试系统的透明加解密功能，对此设计了下表 6-1 的测试用例。

表 6-1 透明加解密功能测试用例

测试项目	测试步骤	测试结果
透明加解密功能	1 将记事本进程设置为受信进程，将文件大小为 197KB 的 a.txt 文档复制到安全目录中	文件大小变成 201KB
	2 在虚拟机中打开文件 a.txt	可以正常打开
	3 关闭文件重定向过滤器驱动，然后打开文件 a.txt	无法正常打开
	4 将文档 a.txt 通过 QQ、E-mail 或 U 盘传输到未安装本系统的主机中	无法正常打开

将文件 a.txt 复制到安全目录时，文件被自动加密，添加了一个 4KB 的文件加密标识头，因此文件大小会增加 4KB。当在虚拟机中打开文件 a.txt 时，进行了正常的透明解密，文件正常打开，如图 6-1 所示。当关闭文件重定向过滤器驱动时，无法将文件读操作重定向到加密桥文件系统所映射的虚拟磁盘中，也就无法对文件进行透明解密，文件 a.txt 无法正常打开，当文档通过互联网或外设传输到未安装本系统的主机中时，文件同样无法正常打开，如图 6-2 所示。

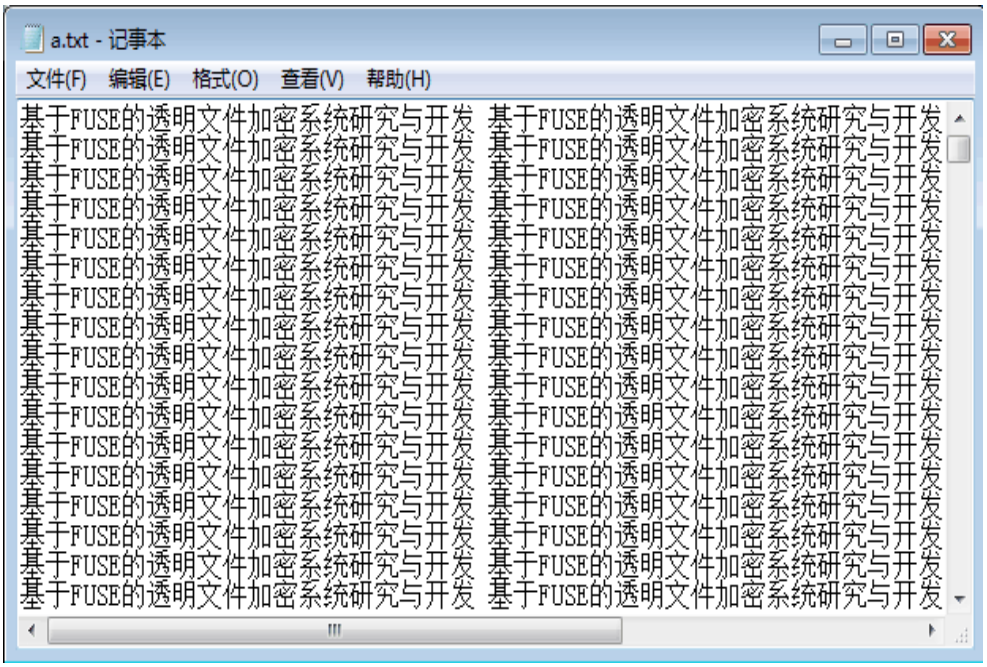

图 6-1 记事本进程正常打开文件

图 6-2 记事本进程打开文件错误

对比图 6-1 和图 6-2 可以明显看出，本文的透明文件加密系统达到了透明加解密的效果，本系统可以有效防止因互联网或者外部设备传输而导致的机密文件泄露。

6.2.2 进程控制策略测试

本小节主要测试系统的进程控制策略功能，选定 WORD 文档作为代表来进行测试，对此设计了下表 6-2 的测试用例。

表 6-2 进程控制策略功能测试用例

测试项目	测试步骤	测试结果
进程控制策略功能	1 将 WORD 进程设置为受信进程， a.docx 文档为加密文件，打开文档 a.docx	可以正常打开
	2 将 QQ 进程设置为受信进程，将 a.docx 文档通过 QQ 发送到主机，然后打开文档 a.docx	可以正常打开
	3 将写字板进程设置为非受信进程，然后打开文档 a.docx	无法正常打开

当 WORD 进程作为文件后缀名为 docx 的文档的受信进程时，可以正常打开 a.docx 文档，将 QQ 进程同样作为 docx 文档的受信进程，通过 QQ 将文档发送到未安装本系统的主机中，同样可以正常打开 a.docx 文档，如图 6-3 所示。将写字板的进程作为 docx 文档的非受信进程，此时无法正常打开 a.docx 文档，如图 6-4 所示。

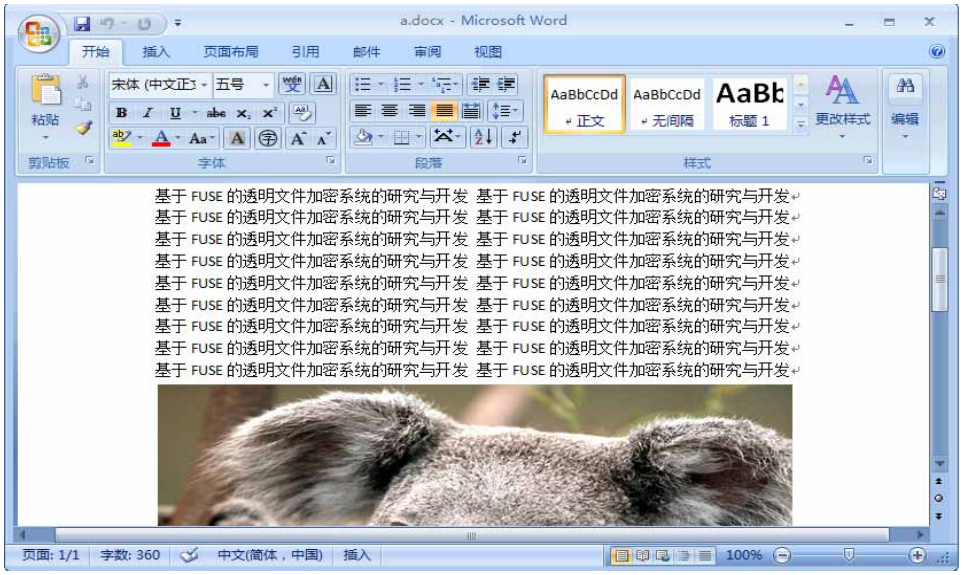

图 6-3 WORD 进程正常打开文件

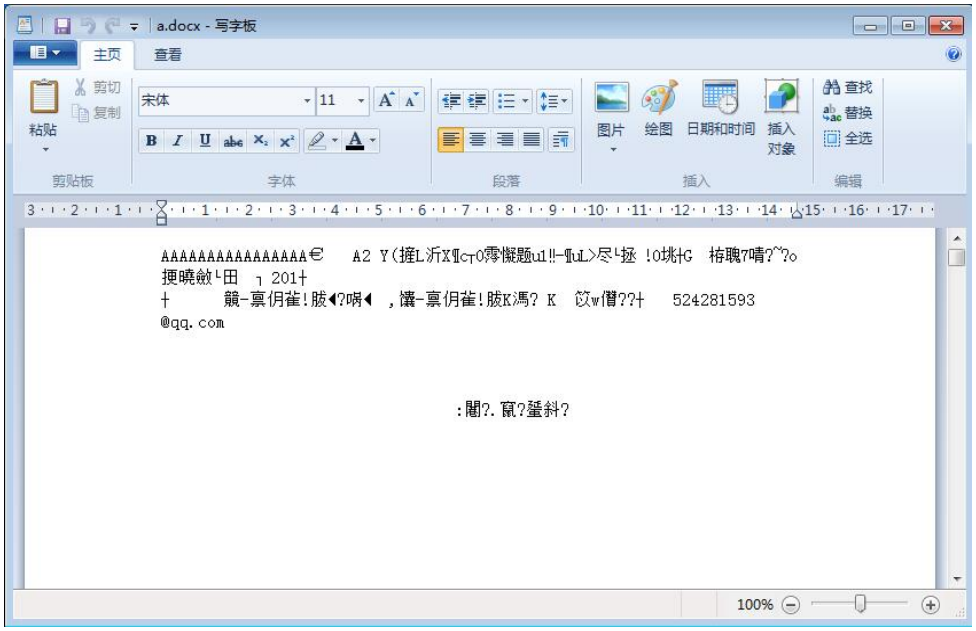

图 6-4 写字板进程打开文件错误

由图 6-3 和图 6-4 对比可知，对于一个文件后缀名来说，如果将进程设置为后缀名对应的受信进程，则可以通过透明解密得到明文数据，如果是非受信进程，则只能获得密文数据，保证了机密文件的安全。

6.2.3 防止进程欺骗测试

本小节主要测试系统的防止进程欺骗功能，选定 PPT 文档作为代表来进行测试，对此设计了下表 6-3 的测试用例。

表 6-3 防止进程欺骗功能测试用例

测试项目	测试步骤	测试结果
防止进程欺骗功能	1 将 PPT 进程设置为受信进程，a.pptx 文档为加密文件，打开文档 a.pptx	可以正常打开
	2 修改配置文件中 PPT 进程的 MD5 值来模拟伪造进程，然后打开文档 a.pptx	弹出警告无法正常打开
	3 将 PPT 进程的 MD5 值恢复原来正确值，将 QQ 进程名 QQ.exe 修改为 POWERPNT.EXE，然后将文档 a.pptx 通过 QQ 发送到主机，然后打开主机中的文档 a.pptx	无法正常打开

当 PPT 进程作为文件后缀名为 pptx 文档的受信进程时，可以正常打开 a.pptx 文档，如图 6-5 所示。一旦修改了 PPT 进程的 MD5 值，这就相当于一个进程冒充了 PPT 进程，此时无法正常打开 a.pptx 文档，同时会弹出仿冒进程的警告窗口，如图 6-6 所示。如果将 QQ 的进程名修改为 PPT 进程的进程名时，此时 QQ 进程也会被当做是仿冒进程，通过 QQ 进程发送出去的 a.pptx 文档也将无法正常打开，如图 6-7 所示。

对比图 6-5，图 6-6 和图 6-7，根据上面的测试结果可知，对于一个进程而言，其进程名是可以冒充的，但是其 MD5 值却是无法改变的，恶意进程是无法通过将进程名修改为对应文件的受信进程名而冒充受信进程来获取明文数据的，防止了因进程名仿冒而导致的泄密。

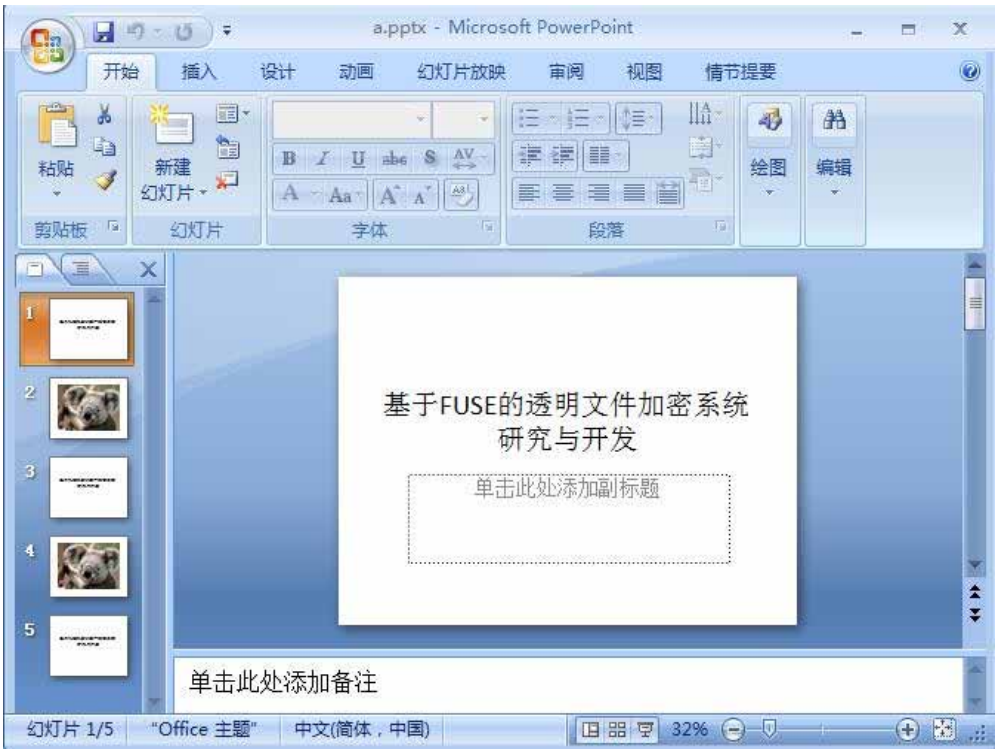

图 6-5 PPT 进程正常打开文件

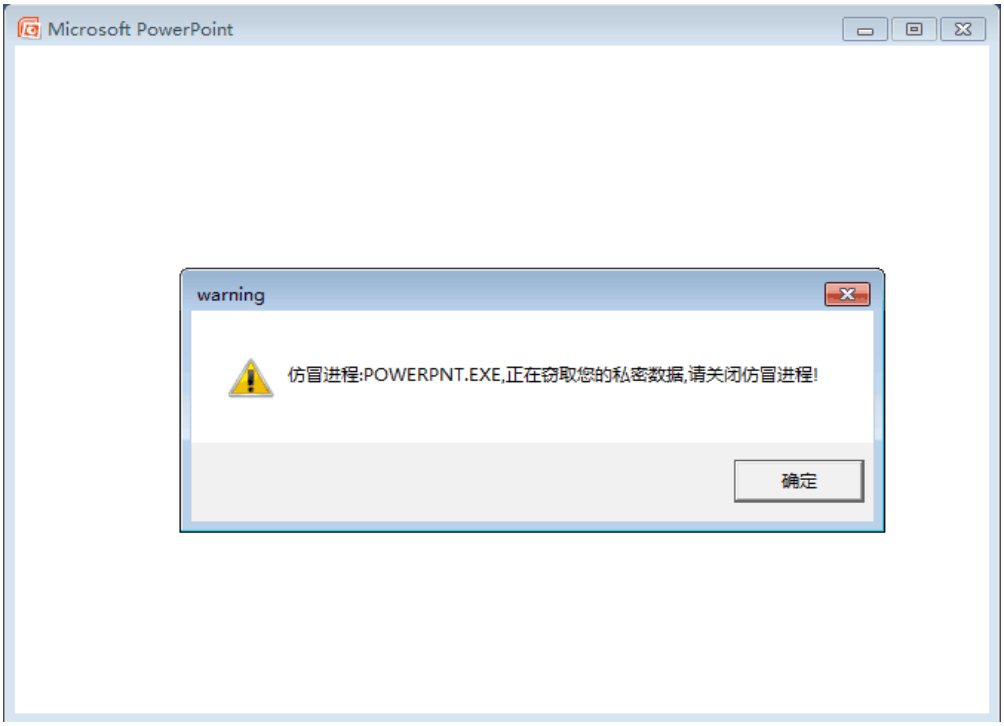

图 6-6 PPT 进程弹出警告

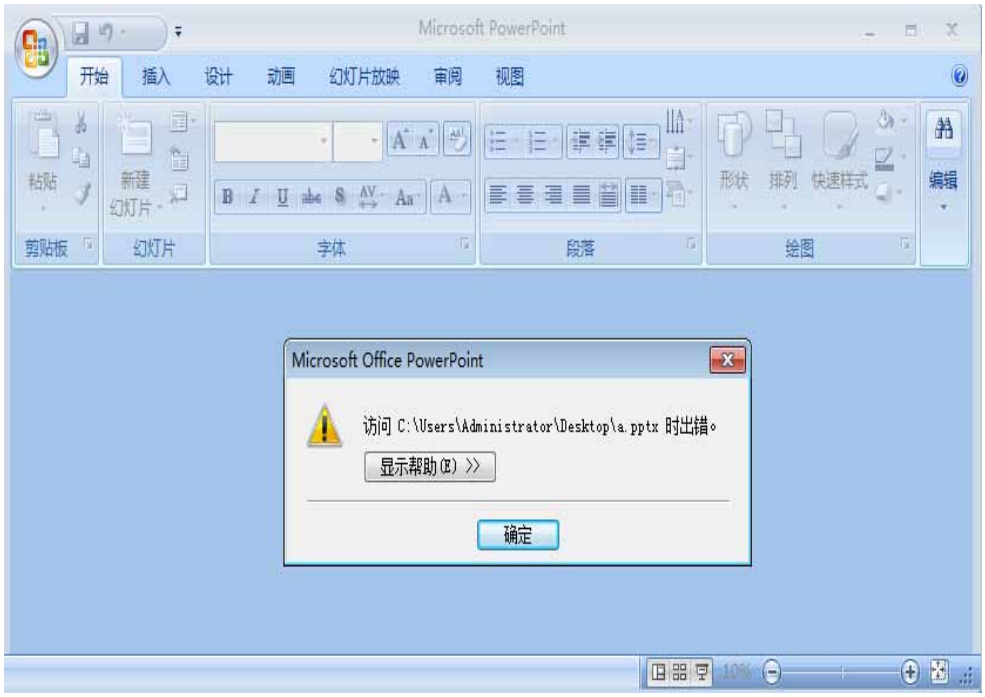

图 6-7 PPT 进程打开文件错误

6.2.4 防复制防截屏测试

本小节主要测试系统的防复制防截屏功能，选定 WORD 文档作为代表来进行测试，对此设计了下表 6-4 的测试用例。

表 6-4 防复制防截屏功能测试用例

测试项目	测试步骤	测试结果
防复制防截屏功能	1 将 WORD 进程设置为受信进程， a.docx 文档为加密文件，打开文档 a.docx，复制文档中的内容，并粘贴到 b.docx 文档中	无法正常粘贴
	2 通过 QQ 截屏功能对 a.docx 文档窗口进行截屏，将截屏图片粘贴到 b.docx 文档中	无法正常粘贴
	3 将记事本进程设置为非受信进程，打开 a.docx，复制文档中的内容并对窗口进行截屏	可以正常粘贴
	4 打开非安全目录中的 c.docx 文档，复制 c.docx 文档中的内容并对窗口进行截屏并粘贴。	可以正常粘贴

当受信进程打开安全目录中的加密文件 a.docx 时，防复制防截屏模块监测到此操作，剪贴板监控进程启动，当复制 a.docx 文档中的内容或者对文档窗口进行截屏时，剪贴板中出现文本格式或者图片格式的文件，剪贴板被清空，此时无法将复制的内容和截屏的图片粘贴至安全目录外的文件 b.docx 中。当非受信进程打开文件 a.docx 时，剪贴板监控进程不会启动，可以正常的对复制和截屏的内容粘贴到文件 b.docx 中。打开非安全目录中的文件 c.docx 时，同样也可以进行正常的复制和截屏。由上述测试结果可知，当受信进程打开安全目录中的文件时，无法对文件进行复制和截屏，在不影响用户正常使用的前提下，防止了因复制明文和截屏而导致的泄密。

6.3 系统性能测试与分析

本文中的透明文件加密系统采取了三种加密模式并选取其中一种加密模式使用两种读写机制进行实现，本节将针对这三种加密模式以及两种读写机制进行性能的测试与对比分析。

6.3.1 三种加密模式性能测试

本小节对本文中采用的 AES-ECB、AES-CTR、HASH-CTR 这三种加密模式实现的透明加密系统进行性能测试与分析。同时为了对比，选取了安全目录外同大小的未加密文件进行实验。对此设计了下表 6-5 的测试用例。

表 6-5 三种加密模式性能测试用例

测试项目	测试步骤
三种加密模式性能	1 编写读写文件的测试程序，从开始读写文件到读写结束进行计时，时间精确到毫秒，并由程序输出时间
	2 将测试程序设置为相应文件的受信进程，使用测试程序读写未加密的大小分别为 2M,8M,16M,32M 文件，并分别记录时间，测试 30 组取平均值
	3 使用测试程序分别读写 AES-ECB 模式、AES-CTR 模式、HASH-CTR 模式加密的大小分别为 2M,8M,16M,32M 文件，并分别记录时间，测试 30 组取平均值

根据上述的测试用例，得出的测试结果如表 6-6 所示。

表 6-6 三种加密模式性能测试结果（M/S）

	2M	8M	16M	32M
未加密	45.4	44.7	43.5	44.6
AES-ECB 加密	25.9	26.8	24.5	25.7
AES-CTR 加密	30.4	29.8	29.6	30.6
HASH-CTR 加密	30.5	29.9	29.8	30.3

由于外界因素影响，以及操作系统读写文件机制的复杂性，测试的数据可能存在一定的不稳定性以及不准确性，但是能明显的区分几种加密模式的性能。由表 6-6 可知，未加密文件的读写性能最佳，采用 AES-ECB 加密模式加密的文件在读写时由于 AES 算法的引进以及存在对数据的扩展和截断处理，所以导致读写的延迟较大，性能也最差。由于 CTR 加密模式支持文件的随机存取，无需对数据进行扩展和截断处理，因此，AES-CTR 加密模式和 HASH-CTR 加密模式的性能要好于 AES-ECB 加密模式。AES 算法和 HASH 算法在进行加解密效率上差别不大，因此采用 AES-CTR 加密模式和 HASH-CTR 加密模式读写性能基本一致。

6.3.2 缓存方式与非缓存方式读写性能测试

本小节选取 AES-CTR 加密模式作为代表，对以缓存方式和非缓存方式这两种文件读写机制实现的系统进行性能测试分析，对此设计下表 6-7 的测试用例。

表 6-7 两种读写机制性能测试用例

测试项目	测试步骤
两种读写机制性能	1 编写读写文件的测试程序，从开始读写文件到读写结束进行计时，时间精确到毫秒，并由程序输出时间
	2 将测试程序设置为相应文件的受信进程，使用测试程序读写非缓存方式的 AES-CTR 模式加密的大小分别为 2M,8M,16M,32M 文件，并分别记录时间，测试 30 组取平均值
	3 使用测试程序读写缓存方式的 AES-CTR 模式加密的大小分别为 2M,8M,16M,32M 文件，并分别记录时间，测试 30 组取平均值

根据上述的测试用例，得出的测试结果如表 6-8 所示。

表 6-8 两种读写机制性能测试结果（M/S）

	2M	8M	16M	32M
非缓存方式	30.2	29.7	30.7	29.8
缓存方式	40.5	41.3	40.1	40.8

表 6-8 中的测试结果显示，采用缓存方式的文件读写性能要比非缓存方式提高了 30%左右，这是由于缓存方式减少了进程读写磁盘的次数以及加解密的次数，因此读写效率会有大幅提高。

从系统安全性方面考虑，AES 加密算法的安全级别显然要高于 HASH 算法。结合 AES 加密算法的安全性和 CTR 块加密工作模式的高效性，本系统采用 AES-CTR 加密模式，不仅保证了系统的安全性，同时保证了文件读写的效率。对比表 6-6 中的测试结果，采用缓存方式和 AES-CTR 加密模式的文件读写性能与未加密的文件读写性能相比，只延迟了 10%左右，达到预期的系统设计要求。

6.4 系统安全性分析

本文基于 FUSE 的透明文件加密系统，通过对安全目录中文件的透明加密，使得安全目录中的文件始终是以密文的形式保存在磁盘中，并根据文件后缀名来设置进程访问策略，对于不受信任的进程只能获取无法正常阅读的密文，即使数据通过 U 盘拷贝，网络传输等方式恶意传播，得到的也只是密文数据，从而保证了机密数据的安全性。

在用户恶意操作的安全性保证方面，本文通过剪贴板监控和针对安全目录文件操作的监控，防止了恶意用户通过复制，截屏操作将明文数据传播出去，对受信进程进行了真实性验证，防止计算机木马程序通过进程名伪造非法获取系统明文数据，同时，对系统配置文件提供了内核层隐藏和应用层签名验证双重的安全保护，防止黑客程序恶意篡改配置文件而导致的泄密。

6.5 本章小结

本章对本文所开发的基于 FUSE 的透明文件加密系统进行了功能测试和性能测试，介绍了测试所需的环境，并设计了测试用例。对测试结果进行了对比与分析，达到了预期的设计效果，最后对系统的安全性进行了分析。

第 7 章 总结与展望

7.1 总结

本文通过将文件重定向技术和 FUSE 文件系统相结合，提出了一种基于 FUSE 的透明文件加密系统方案。该方案以安全目录为基本单位，将应用程序进程对安全目录中文件的操作重定向到加密桥文件系统的虚拟磁盘中，实际转化为 FUSE 用户空间进程对安全目录中文件的操作实现。同时与剪贴板监控技术相结合，对安全目录中的文件实现防复制防截屏，对机密文件提供了全面的保护。本文完成的主要工作如下：

（1）分析了企业机密数据加密的重要性，研究了透明加密技术的国内外现状，并针对现有的技术方案提出了本文基于 FUSE 的透明文件加密系统模型，分析了系统需求，对系统整体方案进行了设计，同时完成了加密文件结构的设计。

（2）开发了文件重定向过滤器驱动模块，其中包括跟文件重定向相关的安全目录信息的策略信息管理，同时在驱动层对配置文件进行了隐藏保护，实现将应用程序进程对安全目录中文件的文件操作以及重命名重定向到加密桥文件系统映射的虚拟磁盘中。

（3）开发了加密桥文件系统模块，其中包括实现受信进程与非受信进程的识别，受信进程的防伪处理，实现了打开文件时对文件进行自动加密的处理，同时采取了 AES-ECB 模式、AES-CTR 模式、HASH-CTR 模式这三种块加密工作模式实现了文件读写时的透明加解密。针对文件读写时的效率问题，设计了缓存结构，实现了带有缓存机制的透明加解密。

（4）开发了系统安全加固模块，通过进程对安全目录中文件操作的监控以及剪贴板监控，实现了针对安全目录中打开的文件内容的防复制和打开的文件窗口的防截屏，同时针对受信进程进行了防伪处理，并对保存系统信息的配置文件进行了安全保护，实现了系统的安全加固。

（5）编写了测试程序，对系统各模块的功能进行了测试，并对以不同实现方式实现的系统性能进行了测试和分析。

本文的创新点在于借助起着桥梁作用的 FUSE 文件系统，针对传统透明文件

加密技术在缓存处理方面的不足，通过文件重定向过滤器驱动将应用程序进程针对安全目录中文件的 I/O 操作转移到 FUSE 用户空间进程来实现，无需在内核层对文件进行加解密处理，使得系统更加稳定且易于维护。同时实现了针对安全目录中文件的防复制防截屏，并且对受信进程的 HASH 值进行了有效性验证，防止因进程名篡改而导致的泄密，对机密文件数据提供了全面的保护，具有一定的实用价值。

7.2 展望

本文设计与实现了基于 FUSE 的透明文件加密系统，该系统各方面功能以及性能达到预期目标。但由于研究水平和时间限制，并随着企业对透明文件加密系统的安全性与稳定性要求越来越高，本系统仍然存在不足之处需要从以下几点作进一步研究：

（1）针对非受信进程维护一个密文缓存，实现一种双向的透明文件加密系统。

（2）将防打印技术与本文中的防复制防截屏技术相结合，使得安全目录中的文件得到更全面的保护。

（3）结合机密文档的自动备份以及灾难自动恢复等相关技术形成保证文档安全的整体解决方案。

致谢

一转眼，论文的撰写工作已经接近尾声，这也就意味着三年的研究生求学之路即将结束，纵有万般不舍，我也将从校园这座象牙塔步入社会。在此，我要感谢这三年里陪伴我成才和给予我帮助的人。

首先，我要感谢我的父母，他们虽然文化水平不高，但是深知教育的重要性，感谢他们一直以来对我默默的支持和无私的付出。

其次，我要由衷的感谢我的导师龙毅宏教授，在平时的科研生活中，龙老师以严谨的科研态度告诉我什么是科研精神，当遇到科研难题时，龙老师总是第一时间帮助我设计方案解决问题并耐心讲解其中的原理，不仅使我的科研技术得到很大的提高，而且使我培养了一种严谨的思维。本论文的顺利完成是与龙老师的帮助分不开的，无论是论文的选题、方案的设计还是最后的修改定稿，龙老师都倾注了大量心血。

最后，我要感谢伴随我成长的所有实验室同门，他们丰富了我的生活，在日常生活和学习中给予了我很大的帮助，他们是方敏、周旭、李超、陈瑛、李凤利、孙蒙、卢衍军、陈慧、何翔、吴静琳。在这里，再此对他们表示感谢。

求学生涯即将完成，我将带着家人、导师、朋友们的期望和鼓励，迈出校园步入社会，进入人生崭新的阶段。在此祝愿所有的亲人朋友、老师同学们身体健康，开心快乐，幸福美满。

参考文献

[1] 王世伟,赵付春. 企业用户感知视角下的云计算信息安全策略研究[J]. 图书情报工作,2012,06:29-32.

[2] 2015 上半年企业泄密事件整理[EB/OL]. https://www.douban.com/note/507712430/,2015.

[3] 金彪,熊金波,姚志强,等. 基于身份的受控文档透明加解密方案[J]. 计算机应用,2013,11:3235-3238.

[4] 李世富. 基于透明加密的文件保密技术研究[D].武汉理工大学，2012.

[5] 张小川,陈最,涂飞. 基于过滤驱动的透明加密文件系统研究与实现[J]. 计算机应用与软件,2013,04:44-47.

[6] 周道明,钱鲁锋,王路路. 透明加密技术研究[J]. 信息网络安全,2011,12:54-56.

[7] 高汉军,寇鹏,王丽娜. 面向虚拟化平台的透明加密系统设计与实现[J]. 武汉大学学报(理学版)，2010，(02)：223-226.

[8] 赵铭伟,毛锐,江荣安. 基于过滤驱动的透明加密文件系统模型[J]. 计算机工程,2009,01:150-152.

[9] Zhou Zhen-liu, Tian Ling, Li Zhuo-ling. Transparent Encrypting Office Document Based on Plug-in Software[J]. Procedia Engineering,2011,23:.

[10] Wu Y, Chen X C. File Transparent Encryption System Design Based on Security Directory[J]. Applied Mechanics & Materials, 2013, 433-435:1742-1746.

[11] Xiaosong Zhang, Fei Liu, Ting Chen, Hua Li. Research and application of the transparent data encryption in intranet data leakage prevention[C].Proceedings of the 2009 International Conference on Computational Intelligence and Security,2009: (vol.2) 376-9.

[12] Hu J (Hu, Ji), Klein A (Klein, Andreas).A Benchmark of Transparent Data Encryption for Migration of Web Applications in the Cloud[C]. EIGHTH IEEE INTERNATIONAL CONFERENCE ON DEPENDABLE, AUTONOMIC AND SECURE COMPUTING, PROCEEDINGS,2009:735-740.

[13] Jin Qian Liang，Xiao hong Guan．A virtual disk environment for providing file system recovery[C]．Computers&Security,2006,25(8):589~599.

[14] 孙成浩. 安全文件系统的设计与实现[D].山东大学 2009.

[15] 张佩,马勇,董鉴源．竹林蹊径:深入浅出 windows 驱动开发[M]．北京：电子工业出版社，2011.

[16] 刘晗. 基于双缓冲过滤驱动的透明加密系统研究与实现[D].重庆大学,2010.

[17] 陈炜,曹斌. 基于双缓冲文件系统过滤驱动技术研究[J]. 信息安全与技术，2013,11:44-46+52.

[18] 顾正义,黄皓. 新加密文件系统的研究与实现[J]. 计算机工程与设计, 2009, 30(14) 3272-3277.

[19] 钱镜洁,林艺滨,陈江勇. EFS 离线解密方法及其取证应用[J]. 信息网络安全,2013,08:67-70.

[20] Osama A. Khashan, Abdullah M. Zin,Elankovan A. Sundararajan. ImgFS: a transparent cryptography for stored images using a filesystem in userspace[J]. Frontiers of Information Technology & Electronic Engineering,2015,161:.

[21] 石京民,陈道敏. Hook 技术及其应用[J]. 计算机应用, 2007, 21(4):83-84.

[22] Anonymous. Data Encryption; Vormetric Announces Transparent Encryption for SAP[J]. Information Technology Newsweekly,2011,:.

[23] L. Xie. Research and Application of the Transparent Document Encryption Gateway in Information Leakage Prevention[J]. Internet Technology and Applications, 2010.

[24] Y. Li, D. Hu, C. Miao, Y. Yuan. A Traceable File Protection System with Transparent Encryption Technology[J]. Internet Technology and Applications (iTAP), 2011.

[25] Solomon D A, Russinovich M E. Windows NT File System Internals [M]. 4th ed. [S.l]: Microsoft Press, 2005.

[26] Solomon D A, Russinovich M E. Inside Microsoft Windows 2000 [M]. NewYork: Microsoft Press, 2001.

[27] 谭文, 杨潇, 邵坚磊, 等. 寒江独钓：Windows 内核安全编程[M]. 北京：电子工业出版社，2009, 06：163-280.

[28] 金华锋. NTFS 下基于双缓冲机制的透明加解密系统的研究与实现[D]. 北京工业大学, 2011.

[29] W. A. Bhat, S. M. K. Quadri. suvfs: A virtual file system in userspace that supports large files[J]. Big Data, 2013.

[30] Junsup Song, Dongkun Shin. Performance improvement with zero copy technique on FUSE-based consumer devices[C]. Consumer Electronics (ICCE), 2014.

[31] S. Li, S. Lv, X. Jia, Z. Shao. Application of Clipboard Monitoring Technology in Graphic and Document Information Security Protection System[J]. Intelligent Information Technology and Security Informatics (IITSI), 2010

[32] 王煦,朱震,苗启广. 文件头中存储加密标识技术的研究与实现[J]. 计算机工程与设计,2012,10:3746-3750.

[33] 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.

[34] 尹浩然,张伟,蔡满春. 基于系统内核的全方位立体文件保护系统[J]. 信息网络安全,2012,09:41-42.

[35] 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.

[36] HU Zhihua, XIONG Kuanjiang. A Novel Key Scheduling Scheme for AES Algorithm[J]. Wuhan University Journal of Natural Sciences,2016,02:110-114.

[37] 曾梦岐,卿昱,谭平璋,等. 基于身份的加密体制研究综述[J]. 计算机应用研究,2010,01:27-31.

[38] 胡亮,初剑峰,林海群,等. IBE 体系的密钥管理机制[J]. 计算机学报，2009,03:543-551.

[39] 罗江华. 基于 MD5 与 Base64 的混合加密算法[J]. 计算机应用,2012,S1:47-49.

[40] 陈玲,唐祯敏. 重定向在文件系统过滤驱动中的实现及其应用[J]. 中国科技信息，2007,22:92-93.

[41] 马俊,王志英,任江春,等. DIFS:基于临时文件隔离实现数据泄漏防护[J]. 计算机研究与发展,2011,S1:17-23.

[42] Dokan API[EB/OL]. https://github.com/dokan-dev/dokany/wiki/API,2015.

[43] Block cipher mode of operation[EB/OL]. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation,2015.