

分 类 号 \_\_\_\_\_

学号 \_\_\_\_\_ M201572702 \_\_\_\_\_

学校代码 10487

密级 \_\_\_\_\_

华中科技大学  
硕 士 学 位 论 文

基于全闪存盘阵的数据安全删除研究

学位申请人： 徐德传

学 科 专 业： 计算机技术

指 导 教 师： 曾令仿 副教授

答 辩 日 期： 2017 年 5 月 27 日

A Thesis Submitted in Partial Fulfillment of the Requirements  
for the Degree of Master

**Research of Secure Deletion based on Solid-State Disk  
Array**

Student : Dechuan Xu

Major : Computer Technology

Supervisor : A/Prof. Lingfang Zeng

**Huazhong University of Science & Technology**

**Wuhan 430074, P. R. China**

**May 27, 2017**

## 独创性声明

本人声明所呈交的学位论文是我个人在导师的指导下进行的研究工作及取得的研究成果。尽我所知,除文中已标明引用的内容外,本论文不包含任何其他人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体,均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名:

日期: 年 月 日

## 学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、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。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索,可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于  保密口,在 \_\_\_\_ 年解密后适用本授权书。  
 不保密口。  
(请在以上方框内打“√”)

学位论文作者签名:

日期: 年 月 日

指导教师签名:

日期: 年 月 日

## 摘 要

全闪存盘阵是一种全新的存储架构,由于其读写性能优越,I/O时延小,是用于取代目前阵列使用的机械硬盘的首选,将会是未来数据中心的主流物理存储设备。但由于闪存盘硬件的特性,其上存储的数据在被删除后,数据依然存在其物理页上,有可能造成数据泄露。基于全闪存盘阵的数据安全删除是存储研究领域积极研究的课题之一,但目前尚无很好的解决方案。

针对固态盘的异地更新导致数据可能泄露的问题,本研究在全闪存盘阵列的基础上,提出一种通用性的方案,其根据原始数据及其哈希信息经过非对称加密、RS纠删码冗余编码等步骤,产生冗余加密数据单元。在擦除单个盘上的元数据,或者是擦除一定数量的加密数据后,其对应的原始数据再也无法恢复出来,从而保证数据的安全。

在依据研究方案实现的仿真实验上,通过对仿真系统的安全删除等功能的验证,以及对系统的数据处理带宽、I/O时延、系统负载、数据冗余率等方面的详细测试,结果表明,在擦除了哈希信息等元数据后,原始数据无法恢复,达到了安全删除的目的;系统的数据处理等性能表现正常,满足可靠性和可用性的要求。

**关键词：** 固态盘, 阵列, 安全删除

## Abstract

All-flash disk array is a new storage architecture. Because of its superior read and write performance, smaller I/O latency, is preferred to be used to replace the current array that use the mechanical hard drive. SSD will be the mainstream physical storage device used in data center in the future. But due to the characteristics of flash disk hardware, after the data stored on it is deleted, the data is still existing on its physical page. There maybe caused data leakage. Secure Deletion based on Solid-State Disk Array is one of the most active research topics in the storage field, but there hadn't good solution yet.

For the out-of-place update of SSD may lead to data leakage problems, this study proposes a general solution, which raw data and its hash information participate in asymmetric encryption, RS erasure code redundancy encoding, and so on, generate redundant encrypted data units, after erasing the hash information on the key disk or erasing a certain amount of encrypted data, the original data can no longer be recovered. Therefore, the proposal ensures the security of data.

On the basis of simulation experiments that be achieved by the research, through verifying the secure deletion and other functions. In addition, testing detailly data processing bandwidth, I/O latency, system load, data redundant rate and other aspect, the result shows that, besides the simulation achieved the purpose of secure deletion that the original data can not be recovered after deleting the hash information of the raw data, the system data processing performance is normal and it meets the reliability and availability requirements.

**Key words:** SSD, Disk Array, Secure Deletion

## 目 录

<b>摘要</b>	<b>I</b>
<b>插图索引</b>	<b>V</b>
<b>表格索引</b>	<b>VII</b>
<b>1 绪论</b>	<b>1</b>
1.1 研究背景 . . . . .	1
1.2 研究目的和意义 . . . . .	2
1.3 国内外研究现状 . . . . .	3
1.4 本文的研究内容 . . . . .	5
<b>2 方案系统设计</b>	<b>6</b>
2.1 方案前提 . . . . .	6
2.2 技术关键 . . . . .	6
2.3 方案详细设计 . . . . .	8
2.4 方案整体流程 . . . . .	15
2.5 本章小结 . . . . .	22
<b>3 仿真系统实验设计</b>	<b>23</b>
3.1 实验目的 . . . . .	23
3.2 实验环境 . . . . .	23
3.3 实验方案 . . . . .	24
3.4 仿真系统实现 . . . . .	25
3.5 本章小结 . . . . .	33
<b>4 测试与分析</b>	<b>34</b>
4.1 测试平台及环境说明 . . . . .	34
4.2 功能测试 . . . . .	34
4.3 I/O 带宽测试 . . . . .	35
4.4 I/O 时延测试 . . . . .	38

# 华 中 科 技 大 学 硕 士 学 位 论 文

---

4.5 系统负载测试 . . . . .	39
4.6 数据冗余率测试 . . . . .	43
4.7 本章小结 . . . . .	45
<b>5 总结与展望</b>	<b>46</b>
致谢	47
参考文献	48

## 插图索引

图 1-1	固态盘异地更新 . . . . .	2
图 2-1	支持 TRIM 的固态盘删除、更新数据操作 . . . . .	7
图 2-2	Ming the Merciless . . . . .	8
图 2-3	模块功能架构 . . . . .	9
图 2-4	编码运算 . . . . .	10
图 2-5	vandermonde 矩阵 . . . . .	11
图 2-6	加密算法流程 . . . . .	11
图 2-7	新编码矩阵运算式 . . . . .	12
图 2-8	同乘 $B'^{-1}$ 矩阵运算式 . . . . .	13
图 2-9	RS 恢复数据编码 . . . . .	13
图 2-10	删除数据块的方法 . . . . .	14
图 2-11	删除存储的元数据 . . . . .	14
图 2-12	系统整体架构 . . . . .	15
图 2-13	数据写入架构 . . . . .	16
图 2-14	数据写入流程 . . . . .	17
图 2-15	数据读取架构 . . . . .	18
图 2-16	数据读取流程 . . . . .	19
图 2-17	擦除哈希信息验证读取架构 . . . . .	20
图 2-18	擦除哈希信息验证读取流程 . . . . .	21
图 3-1	测试流程 . . . . .	24
图 3-2	仿真环境系统图 . . . . .	25
图 3-3	输入缓存数据结构设计 . . . . .	26
图 3-4	输出缓存数据结构设计 . . . . .	27
图 3-5	输出缓存架构设计 . . . . .	29
图 3-6	密钥存储结构 . . . . .	30
图 4-1	数据读取带宽 . . . . .	36
图 4-2	数据写入带宽 . . . . .	37
图 4-3	系统读取延迟 . . . . .	38

图 4-4	系统写入延迟 . . . . .	39
图 4-5	系统前端 CPU 占用 . . . . .	40
图 4-6	系统后端 CPU 占用 . . . . .	41
图 4-7	系统前端内存占用 . . . . .	42
图 4-8	系统后端内存占用 . . . . .	43
图 4-9	数据冗余率测试结果 . . . . .	44

## 表格索引

表 1.1	SLC、MLC、TLC、QLC 四种闪存芯片的特性 . . . . .	3
表 3.1	实验环境硬件配置 . . . . .	24
表 4.1	系统各项功能测试结果 . . . . .	35

## — 緒論

### 1.1 研究背景

大容量、低成本、高性能的存储系统设计一直是存储领域研究的热点,而且随着大数据云存储时代的到来,人们对这样的存储系统的要求更加迫切。而传统的磁盘虽然容量大价格低,但磁盘的机械特性导致了它的性能在很大程度上已经到达极限。因此人们将希望寄托于新型的存储介质上,比如闪存存储器和相变存储器等。近年来固态硬盘已经成为固态硬盘(SSD)已经成为固态技术中的领先技术,最常见的SSD都是基于NAND Flash芯片设计的。SSD以闪存作为存储介质,具有随机读取速度快,功耗低,抗震性好等优点。因此目前广泛运用于家用和企业市场,在发展中的云存储系统中渐渐取代了传统的普通机械硬盘。

但是,目前依然存在一些问题,比如在对数据安全性要求高的场合,普通的SSD组成的阵列无法保证数据的安全。另外,由于SSD固有的特点,被删除的数据依旧可能被非法窃取,导致数据泄露等安全事故。针对这些问题,本研究旨在固态硬盘阵列存储系统之上设计一套冗余存储方案,存储的数据经过与其哈希信息复杂运算后,生成冗余的加密数据,只要在物理意义上完全删除哈希信息或者一定量的数据块,原始数据即无法恢复,保证数据的安全。

固态盘的特性决定了,普通删除方式是根本不可能“真正”删除数据的,数据依然存在物理介质上<sup>[1]</sup>。在对闪存页(page)写操作时,需要首先进行块擦除操作,因此,不能对闪存进行就地更新(in-place-update)的操作,只能采用异地更新(out-of-place update)的写方式,这就导致了旧的数据在某个时间窗口内依然存留在闪存上,删除操作只是系统反馈的一个“假的”删除成功。数据删除整个过程如图1-1所示。用户在文件系统上首先创建了FileA、FileB、FileC、FileD四个文件数据,反映到固态盘的逻辑块存储视图上的数据,以及在固态盘物理介质上的数据块分别如图1-1所示。当用户在系统中删除了文件FileC,但是在固态盘的逻辑存储地址和物理介质上,文件C的两个数据块C1、C2却没有删除,也就是说,在用户删除文件数据的时候,数据依然存在。系统只是简单的返回一个“删除成功”的假象给用户。

第三步用户写入新的文件FileE时,虽然在固态盘逻辑块存储地址上,数据块C1、C2的位置被重新写入了新文件FileE的存储地址,但是,在固态盘物理介质上,数据块E1、E2是写入到新的位置,而旧的数据块C1、C2只是被固态盘的垃圾回收标记为“可回收”状态,真正被擦除的时机是不可控的。

	1.用户写入四个新文件	2.用户删除FileC	3.用户写入FileE																																																				
系统存储视图	<table border="1"> <tr><td>FileA</td><td>FileB</td></tr> <tr><td>FileC</td><td>FileD</td></tr> <tr><td>Free</td><td></td></tr> </table>	FileA	FileB	FileC	FileD	Free		<table border="1"> <tr><td>FileA</td><td>FileB</td></tr> <tr><td></td><td>FileD</td></tr> <tr><td>Free</td><td></td></tr> </table>	FileA	FileB		FileD	Free		<table border="1"> <tr><td>FileA</td><td>FileB</td></tr> <tr><td></td><td>FileE</td></tr> <tr><td>Free</td><td></td></tr> </table>	FileA	FileB		FileE	Free																																			
FileA	FileB																																																						
FileC	FileD																																																						
Free																																																							
FileA	FileB																																																						
	FileD																																																						
Free																																																							
FileA	FileB																																																						
	FileE																																																						
Free																																																							
SSD逻辑存储视图(LBA)	<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>C1</td><td>C2</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	C1	C2	D1					<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td></td><td>D1</td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6		D1						<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>E1</td><td>E2</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	E1	E2	D1								
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	C1	C2	D1																																																				
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6		D1																																																					
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	E1	E2	D1																																																				
SSD物理存储视图	<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>C1</td><td>C2</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	C1	C2	D1					<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>C1</td><td>C2</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	C1	C2	D1					<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>无效</td><td>无效</td><td>D1</td></tr> <tr><td>E1</td><td>E2</td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	无效	无效	D1	E1	E2						
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	C1	C2	D1																																																				
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	C1	C2	D1																																																				
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	无效	无效	D1																																																				
E1	E2																																																						

图 1-1 固态盘异地更新

从图 1-1 的三个简单操作可以发现,当用户删除了文件 FileC 之后,在物理介质上它的数据 C1、C2 在回收之前,一直都没有真正地被删除掉,这就是固态盘“异地更新”的特性,利用固态盘的这个特性,攻击者使用一些安全工具依然可以将被删掉的文件 FileC 恢复出来,这就造成了数据被窃取的风险。

怎样设计一套针对全闪存盘阵的安全删除方案,使得用户在物理介质上删除了密钥数据之后,存储的数据就能够保证安全,这就是我们课题要研究的重点。

## 1.2 研究目的和意义

在信息化时代,经济、军事领域对于数据的安全性要求是比较高的。由于关键信息的泄露,可能导致国家或者个人非常重大的损失。目前,数据泄露主要存在以下三个问题:

(1) 在数据处理、传输、存储等中间过程中,数据可能被非法复制窃取。例如网络传输的邮件等可能会被第三方服务商缓存下来,这种不可预知的数据复制操作在互联网时代更加普遍。导致一些企业用户纷纷兴建私有的数据中心,而不是接受公有云的服务,就是为了规避这类问题的风险。

(2) 由于不同存储介质的特性不同,导致用户很难真正地删除数据。虽然针对磁盘已存在很多安全工具,如 O&O SafeErase、DataWipe、BCwipe、HDDerase、SDelete、

表 1.1 SLC、MLC、TLC、QLC 四种闪存芯片的特性

闪存特性	SLC (Single-Level Cell)	MLC (Multi-Level Cell)	TLC (Triple-Level Cell)	QLC (Quad-Level Cell)
存储单元模型	每单元一位数据	每单元两位数据	每单元三位数据	每单元四位数据
速度	特别快	快	慢	很慢
擦除寿命	10 万次 (5×nm)	1 万次 (5×nm)	2500 次 (5×nm)	约 500 次
	10 万次 (3×nm)	5000 次 (3×nm)	1250 次 (3×nm)	N/A
	N/A (2×nm)	3000 次 (2×nm)	750 次 (2×nm)	N/A

Secure Eraser 等<sup>[2-4]</sup>，这些工具基本是使用特定数据对目标文件逻辑页多次写入。对于固态盘这样的新型介质，这些安全删除工具就不起作用。不同介质的存储设备在提供存储服务时，用户很难针对不同的设备使用不同的安全删除手段清除数据，很有可能导致数据泄露。

**(3) 数据安全删除手段可能降低存储设备或系统的可靠性以及性能。**由于工艺制程的提高，固态盘的线宽发展得更小，每单元存储的数据也越来越多，擦除寿命也变得更短<sup>[5,6]</sup>。表 1.1 显示了闪存芯片从 SLC 发展到 QLC 的线宽、擦除寿命等数据。从表中可以看出，随着闪存芯片的发展，数据安全删除的次数是不断下降的，因此，对于数据安全的删除操作，会减少闪存芯片的剩余擦除次数，使其擦除寿命降低，可能会导致坏块的增多，降低其性能，擦除操作过多更有可能导致不可预期的问题出现，影响存储设备的可靠性。

上述的三个问题说明了，一方面，数据的安全性需要做到安全删除，另一方面，研究基于固态盘阵的数据安全删除是一项有意义的工作，国内外的研究工作都处于起步阶段，积极投身这个领域将会为我国的存储安全研究打开新的局面。

### 1.3 国内外研究现状

近几年来，针对固态盘的安全删除技术受到了广泛关注。在国内，包括华中科技大学、中南大学、武汉大学等高等院校和科研机构都开展了相关课题和理论的研究。在国外，美国的华盛顿大学、哥伦比亚大学以及 IBM、HP 等实验室也积极地投身在安全删除领域的研究上。

### 1.3.1 基于密码学意义上的安全删除

数据在被加密存储之后,如何保证删除密钥数据,原来的数据就无法再被还原,成为新的难题。Perlman 等人首次提出了基于时间的数据安全删除方法,针对加密文件,如果密钥过期后数据即被删除,而且文件无法恢复<sup>[7]</sup>。受 Perlman 等人的启发,Tang 等人设计出了 FADE 系统<sup>[8]</sup>,该系统使用公钥密码并使用简单的布尔操作来调整安全删除策略。但是 FADE 只支持一到两层的布尔表达,而且需要使用复杂的公钥系统。Perlman 等人还设计了 Ephemerizer 系统<sup>[9,10]</sup>,该系统需要一个可信服务器保存并管理由数据拥有者指定过期时间的解密密钥。Geambasu 等人给出了一个基于时间的数据可信删除方法的原型 Vanish<sup>[11]</sup>,但 Vanish 容易受到跳跃攻击(hopping attack)和嗅探攻击(sniffing attack)<sup>[12]</sup>。华中科技大学武汉光电国家实验室曾令仿等人提出了一种 SafeVanish<sup>[13]</sup> 方案,通过扩展密钥等份长度的方法提高跳跃攻击的成本,并且引入公钥密码体制抵御嗅探攻击,显著改善了 Vanish 方案。Reimann 等人<sup>[14]</sup>针对 Vanish 系统中当文件在 8 小时后需要被访问时,文件拥有者需要更新节点缓存中的密钥部分问题,提出了将密钥分量分发到随机的网页中保存的改进方案,随着时间的改变,网页会改变存储内容或是被删除。针对以 Vanish 为代表的加密方案存在的单个密钥加密全部数据不能对数据进行细粒度的管理等问题,武汉大学王丽娜等人<sup>[15]</sup>提出了利用秘钥生成树,门限秘密共享(threshold secret sharing)来组织和管理秘钥,同时利用分布式散列表(distributed hash table, DHT)周期性地删除相关秘钥来实现安全删除。中南大学王国军等人改进了 Vanish 方案,除了将密钥分发到 DHT 网络中,还提取部分密文并分发到 DHT 网络中,从而更有效抵抗传统密码分析和暴力攻击<sup>[16]</sup>。西安电子科技大学马建峰和中国科学院信息工程研究所李凤华所在课题组针对网络内容生命周期隐私保护问题,提出了面向网络内容隐私的基于身份加密的安全自毁方案<sup>[17-19]</sup>,这种方案基于身份的加密和分布式 Hash 表(DHT)网络。这些方案更多的是偏向上层应用,在原型系统方面的实现和方法验证方面显得不足。针对密钥从物理介质上“真正地”删除,这些方案涉及的研究都不是很深。

### 1.3.2 物理介质上数据安全删除

面对这一问题,研究者有的提出了从物理损坏或者化学破坏的方式,这种方法不本方案不涉及相关研究。M.Wei 的一个研究<sup>[20]</sup>表明:

1. 存储介质的内建命令通常能提供有效的数据擦除功能,但是并不能保证这些命令总是被制造商正确地实现
2. 将 SSD 的所有数据全部覆盖两次(在大部分情况下)能够有效地清除数据

### 3. 对于单个数据块删除而言,当前的 SSD 都存在不足之处

因此,这一过程的实现并不一直是正确的,在有些情况,文件系统显示已被删除,而数据仍在设备中存在。Peterson 等人<sup>[21]</sup> 在数据块层使用全有或全无的转换技术(AONT)来实施安全删除,该方法通过 AONT 存储每一个数据块,然后覆盖其中的一部分,这使得整个数据块不可用。2012 年,Diesburg 提出 TrueErase<sup>[22]</sup>,一种类似于 TRIM 指令的功能,但是只针对属于某敏感文件的数据块。他们在文件系统和设备驱动之间增加了一个新的传输通道。被修改的设备驱动一旦接受到被删除块的信息,使用其下层的接口就可以实现快速安全删除。TrueErase 比 TRIM 更有效,因为它可以只是安全擦除部分敏感块。TrueErase 改造了存储管理层,全方位监控敏感文件的编辑和更新,并可彻底删除敏感文件,前提是假设文件系统可以直接访问和操作物理闪存,所以其缺点是无法应用于普通固态盘。

总体来说,针对全闪存盘阵应用场景,目前研究都处在初始阶段,需要采用新的架构来保证数据的安全删除<sup>[23]</sup>。现有研究偏向上层应用,并且越是接近上层系统越抽象,数据确定性删除就越困难<sup>[24]</sup>,也难以在存储性能、计算开销和数据通信延迟方面达到很好的平衡<sup>[25]</sup>。另外,现有研究缺乏从数据按需删除的角度,只针对敏感数据实施安全删除,从而并不改变普通用户的使用习惯,或根据不同的安全级别需求采取“合理”的数据安全删除策略。

## 1.4 本文的研究内容

本研究旨在提供一种基于闪存盘组成的阵列的数据安全删除方案。通过对数据进行冗余加密,保障数据安全存储,即高可靠性和高安全性。一方面,采用了非对称加密算法、群域运算保障数据可靠性;另一方面,在编码前引入加密的方式增强数据隐私保护,利用冗余编码的特性结合弱密钥思想,对数据的删除不再需要对整个数据进行覆盖写,而是删除部分数据块,破坏数据完整性或者密钥完整性,使数据无法正常读取。即使攻击者得到数据,也不能获取明文,达到数据安全删除的目的。本研究在保证数据安全存储的同时,降低了对固态盘阵列的读写访问,相当于延长了固态盘的使用寿命。

## 二 方案系统设计

### 2.1 方案前提

方案是将文件的元数据和冗余加密后的数据分别存放在单块固态盘和固态盘阵，因此安全删除数据的关键是要完全擦除单块固态盘上的密钥数据，所以，本方案是构建在单块固态盘支持 TRIM 命令，并且可以安全擦除据的基础上的。

### 2.2 技术关键

随着原始数据量的增长，加密数据单元的大小选取将对整体的数据存储时间损耗产生显著影响。同时，加密算法的设计也需要重点关注，分散加密算法的三个关键参数  $(n, k, r)$ ， $n$  表示最终生成的数据单元， $k$  表示恢复原始数据所需要的最低数据单元数量， $r$  表示如果有效的加密数据单元不大于  $r$  值，则数据无法恢复。如何设计这样一个算法，完成数据的加密，同时占用的存取时间尽可能少。完全删除存储的密钥数据，可以利用固态盘的 TRIM 指令，在需要频繁安全删除数据的场合，可以显著的提升固态盘阵列的性能。传统的机械硬盘在删除数据的时候，简单的在逻辑数据表内把存储要删除的数据的位置标记为可用而已，使用机械硬盘的系统根本就不需要向存储设备发送任何有关文件删除的消息，因为在将来，系统可以随时把新数据直接覆盖到无用的数据上。使用固态盘就不同，当系统准备把新数据要写入那个位置的时候，固态硬盘才意识到原来这写数据已经被删除了。

由于固态盘支持 TRIM 指令，系统在删除逻辑表中删除逻辑扇区地址的同时通知固态硬盘某些数据已经无用了。TRIM 的先进性在于它可以让固态硬盘在进行垃圾回收的时候跳过移动无用数据的过程，从而不再用重新写入这些无用的数据，达到节省时间的目的。同时也减少了闪存删除数据的次数，从而在写入过程中实现高性能。固态硬盘也不需要立即删除或者“垃圾回收”这些 TRIM 指令告知的位置了，它只是先标记这些位置的数据为“无用”即可。

支持 TRIM 的系统，在用户写入数据时并没有不同。但是当用户删除文件的时候，因为系统支持了 TRIM 指令，固态硬盘立刻就把数据标记为“无用”，从而为接下来的垃圾回收做准备。原来存放该文件的空间，固态硬盘把其看做是可用空间，这意味着固态硬盘在执行垃圾回收的过程中拥有更多的可用空间，从而整体提高性能。支持 TRIM 命令的固态盘更新文件如图 2-1 所示。

	1. 用户写入四个新文件	2. 用户删除 FileC	3. 用户写入 FileE																																																				
系统存储视图	<table border="1"> <tr><td>FileA</td><td>FileB</td></tr> <tr><td>FileC</td><td>FileD</td></tr> <tr><td>Free</td><td></td></tr> </table>	FileA	FileB	FileC	FileD	Free		<table border="1"> <tr><td>FileA</td><td>FileB</td></tr> <tr><td></td><td>FileD</td></tr> <tr><td>Free</td><td></td></tr> </table>	FileA	FileB		FileD	Free		<table border="1"> <tr><td>FileA</td><td>FileB</td></tr> <tr><td></td><td>FileE</td></tr> <tr><td>Free</td><td></td></tr> </table>	FileA	FileB		FileE	Free																																			
FileA	FileB																																																						
FileC	FileD																																																						
Free																																																							
FileA	FileB																																																						
	FileD																																																						
Free																																																							
FileA	FileB																																																						
	FileE																																																						
Free																																																							
SSD逻辑存储视图 (LBA)	<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>C1</td><td>C2</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	C1	C2	D1					<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td></td><td></td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6			D1					<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>E1</td><td>E2</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	E1	E2	D1								
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	C1	C2	D1																																																				
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6			D1																																																				
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	E1	E2	D1																																																				
SSD物理存储视图	<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>C1</td><td>C2</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	C1	C2	D1					<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>无效</td><td>无效</td><td>D1</td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	无效	无效	D1					<table border="1"> <tr><td>A1</td><td>A2</td><td>A3</td><td>B1</td></tr> <tr><td>B2</td><td>B3</td><td>B4</td><td>B5</td></tr> <tr><td>B6</td><td>无效</td><td>无效</td><td>D1</td></tr> <tr><td>E1</td><td>E2</td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td></tr> </table>	A1	A2	A3	B1	B2	B3	B4	B5	B6	无效	无效	D1	E1	E2						
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	C1	C2	D1																																																				
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	无效	无效	D1																																																				
A1	A2	A3	B1																																																				
B2	B3	B4	B5																																																				
B6	无效	无效	D1																																																				
E1	E2																																																						

图 2-1 支持 TRIM 的固态盘删除、更新数据操作

在图 2-1 中的更新文件过程中, 删除文件 C 时, 删除指令立即调用内建的 TRIM 命令准备回收文件 C 占用的 C1, C2 两个数据块, 物理介质上, 这两个数据块已经被标记被“回收”状态, 这样每次 GC 的时候就不用再移动 C1、C2 两个数据块, 减少了数据泄露窗口和数据写入量。

如何确保原始数据不被非法地逆向解密读取出来, 涉及到这套方案的冗余加密模块的两个关键参数:(1)直接参与运算的密钥数据;(2)加密后相互独立的数据块单元。只要将这两个条件中任意一个的数据“真正”地破坏掉, 原始数据就再也无法恢复。

安全删除密钥数据, 需要从物理意义上真正地将密钥擦除。目前, 针对固态盘数据主流的物理擦除方法主要有以下三种:

- 使用固态盘驱动器的 ATA 安全擦除。SSD 固件有一个嵌入命令, 可以将驱动器的每一个数据位覆写为 0, 从而达到安全删除数据的目的。这种擦除手段使用引导环境下的 USB 闪存盘, 运行相应的清除工具即可以进行。
- 使用自加密驱动器的加密擦除。在一个自加密的固态盘驱动器(SED)上, 加密密钥存储在驱动器的一块小型存储区域, 由内部硬件加密待存储的数据和解密预备读取的数据, 并且使用安全管理软件管理。我们可以将文件的加密密钥放置在该类 SSD 上, 需要删除数据时。进入 SED 的管理软件, 删除并重新生成固态盘的密钥。这将导致先前写入的密钥数据完全不可读。因此, 冗余加密方案生成的数据也就无法还原成原始数据。

- 损坏密钥存储区的物理介质、这种擦除手段是将 SSD 介质通过标准的高温或者细化手段破坏,以至于不可再读。适用于安全要求非常高的应用场合。

为了适应大多数的固态盘的特性,本方案采用的是 ATA 安全擦除。一方面,带有自加密的固态盘厂商不一定完全设计实现,不能针对所有的固态盘硬件,并且物理或者化学方式的损坏不在方案的讨论范围;另一方面,ATA 安全擦除方法操作相对简单,而且即使系统被恶意破坏,我们也可以通过引导方式下的安全擦除(Secure Erase)功能擦除密钥,保证数据的安全性。

ATA 安全擦除方法根据具体固态盘厂商的实现,内建的擦除命令使用的基本是覆盖数据的方法,即对固态盘的物理页覆盖特定的数据多次,达到无法恢复原始数据的目的,这种覆盖手段通常最高能达到 95.9-99.9% 的数据完全擦除<sup>[20]</sup>。

在 M.Wei 的论文<sup>[20]</sup>中指出,验证数据安全删除需要绕过文件系统,直接读取闪存芯片的数据。需要使用特殊的读取设备 Ming the Merciless 如图 2-2 所示。直接连接闪存芯片的针脚,读取其中的数据,从而避免了闪存转换层(FTL)的干扰。

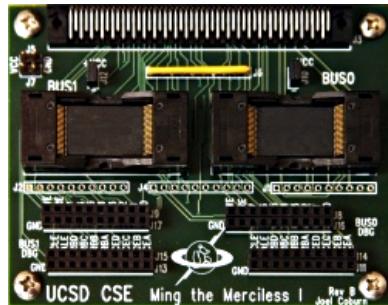


图 2-2 Ming the Merciless

针对本方案涉及到的两个关键问题:(1)数据安全删除;(2)系统可用性或者是可靠性。技术关键就是为了解决怎样安全删除的问题,系统可靠性应该依赖于在解决安全删除的基础之上进行的改进。

## 2.3 方案详细设计

### 2.3.1 模块架构

将多个 SSD 组成一个存储阵列。对于上层组件,如文件系统,将 SSD 看做一个简单的块设备,而不关心其内部的负载结构。用户可以在上面创建分区和文件系统,和普通硬盘一样。这样很容易整合到现有系统,因为对整个系统的改动很小。整体模块架构如图 2-3 所示。

其中,有一个单独的元数据盘用来存储系统初始化时产生的一段随机数据,和原始数据的哈希信息这些元数据,元数据的总量很小,但却是整个系统在进行读取还原数据或者是安全删除时,这部分数据都是关键的信息。攻击者在无法获取这部分元数据时,是无法窃取到原始数据的。

用于存储元数据的固态盘容量也相应的非常小,比如一个 16GB 的闪存盘就可以存储这些数据。这样,安全删除数据时,只要擦除元数据盘上的全部数据,就可以达到数据安全的目的。而且,针对小容量的固态盘的全盘删除,因为数据量小,对固态盘的寿命影响也比较小。

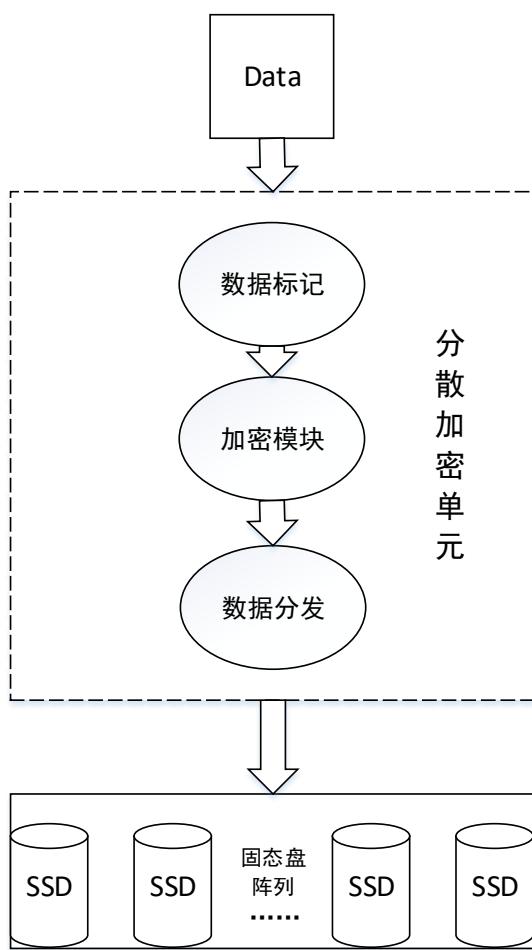


图 2-3 模块功能架构

原始数据经过分散加密系统,首先是数据分成一个个的加密单元数据块,在系统中目前选取的平均数据块大小是 8k,然后由数据标记中心记录原始数据块的元数据,并将元数据与数据块进行多次的哈希和异或、非对称加密运算,得到初始加密数据块,

最后通过冗余编码,得到编码数据块,将这些数据块写入阵列的相应独立区域中。

### 2.3.2 编码原理

纠删码是存储领域常用的数据冗余技术,相比多副本存储而言,纠删码能够以更小的数据冗余度获得更高的数据可靠性。本方案采用的是 Reed Solomon 编码<sup>[26,27]</sup>。它的基本原理如下:给定  $n$  个数据块  $d_1, d_2, \dots, d_n$ ,  $n$  和一个正整数  $m$ , RS 根据  $n$  个数据块生成  $m$  个校验块  $c_1, c_2, \dots, c_m$ 。对于任意的  $n$  和  $m$ ,从  $n$  个原始数据块和  $m$  个校验块中任取  $n$  块就能解码出原始数据。

RS 编码最多容忍  $m$  个数据块或者校验块同时丢失。当存储的数据单元丢失或者被破坏到大于  $m$  个数据单元时,原始数据将无法被恢复,从而保证了数据的安全。具体的 RS 编码步骤如图 2-4 所示。

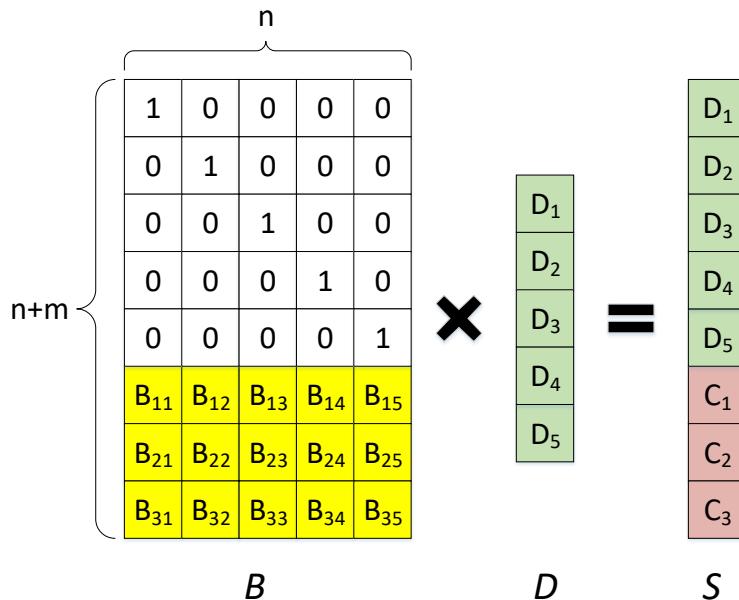


图 2-4 编码运算

输入数据视为向量  $D=D_1, D_2, \dots, D_n$ , 编码后数据视为向量  $S=D_1, D_2, \dots, D_n, C_1, C_2, \dots, C_m$ 。图 2-4 左边是编码矩阵,矩阵上部是单位矩阵( $n$  行  $n$  列),下部是 vandermonde 矩阵矩阵  $B$ ( $m$  行  $m$  列), vandermonde 矩阵如图 2-5 所示。第  $i$  行第  $j$  列的元素值为  $j^{(i-1)}$ )。

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \cdots & n^{m-1} \end{bmatrix}$$

图 2-5 vandermonde 矩阵

因为 RS 数据恢复算法要求编码矩阵的任意  $n*n$  子矩阵可逆, 所以这里采用了 vandermonde 矩阵。数据 D 经过编码后得到了数据 S, 其中包含原始的数据 S 和校验数据 C, 这就是我们最终存储的加密数据单元。

### 2.3.3 冗余加密算法

数据编码算法主要借鉴了 M.Li<sup>[28]</sup> 的冗余加密思想, 将数据和元数据, 以及系统随机的一段初始化数据混合在一起经过多次的非对称加密、冗余编码得到最终的加密数据单元。这种算法在删除了元数据这些关键数据后, 原始数据将无法恢复。整体加密流程如图 2-6 所示。

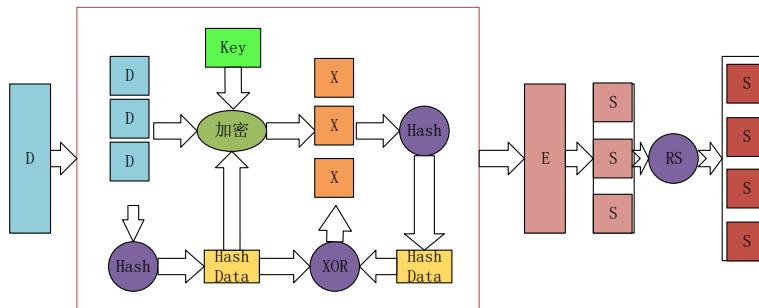


图 2-6 加密算法流程

加密模块在输入原始数据块后, 首先分为大小一定的数据块, 进行一系列的哈希运算和加密操作后, 得到初始的加密数据块 X, 再由 X 的哈希信息数据和之前原始数据的哈希信息数据异或操作得到初步的加密数据块。这些数据块组合成大小相同的加密数据单元 S, 最后由冗余运算模块操作得到最终的加密存储数据。详细的加密流程与算法如公式 2.1 所示。

$$c_i = d_0 \bigoplus E(h_{key}, i), i = 0, 1, \dots, s - 1 \quad (2.1)$$

$$c_s = h_{key} \bigoplus H(c_0, c_1, \dots, c_{s-1})$$

1. 将原始数据  $D$  分割成  $d_0, d_1, \dots, d_{s-1}$  共  $s$  个数据块, 按照公式 2.1 的运算得到初步加密后的数据块  $c_1, c_2, \dots, c_s$
2. 加密后的数据块经过群运算得到最终相互独立的加密数据单元

### 2.3.4 数据恢复原理

在数据被多次加密运算, 并且经过 RS 编码后, 得到了最终的冗余加密数据。在编码原原理一节中提到, RS 最多容忍  $m$  个数据丢失, 当有  $m$  个数据没有参与解码运算时, 数据恢复过程如下。

1. 从编码矩阵  $B$  中删除丢失数据块和丢失编码块对应行。假设  $D_1, D_2$  丢失, 根据图图 2-4 所示的 RS 编码运算等式, 我们得到新的编码矩阵  $B'$  以及等式如图图 2-7 所示。

$$\begin{matrix}
 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 \\
 B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\
 B_{31} & B_{32} & B_{33} & B_{34} & B_{35}
 \end{matrix}
 \times
 \begin{matrix}
 D_1 \\
 D_2 \\
 D_3 \\
 D_4 \\
 D_5
 \end{matrix}
 =
 \begin{matrix}
 D_2 \\
 D_3 \\
 D_5 \\
 C_1 \\
 C_3
 \end{matrix}$$

$$B' \quad D \quad S'$$

图 2-7 新编码矩阵运算式

2. 由于  $B'$  是可逆的, 等式两边乘上  $B'$  的逆矩阵  $B'^{-1}$ , 得到如下等式<sup>1</sup>。

<sup>1</sup>  $\oplus$  表示异或操作

$h_{key}$  表示对数据块取哈希操作, 安全领域使用的主流哈希算法有 SHA-128、SHA-256 等  
 $E$  表示非对称加密算法, 常用的算法有 AES-128、AES-256

$$\begin{array}{c}
 \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 \\ \hline & & & & \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \\ \hline \text{B}_{21} & \text{B}_{22} & \text{B}_{23} & \text{B}_{24} & \text{B}_{25} \\ \hline \text{B}_{31} & \text{B}_{32} & \text{B}_{33} & \text{B}_{34} & \text{B}_{35} \\ \hline \end{array} \times \begin{array}{|c|c|c|c|c|} \hline \text{D}_1 & & & & \\ \hline \text{D}_2 & & & & \\ \hline \text{D}_3 & & & & \\ \hline \text{D}_4 & & & & \\ \hline \text{D}_5 & & & & \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 \\ \hline & & & & \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|c|} \hline \text{D}_2 & & & & \\ \hline \text{D}_3 & & & & \\ \hline \text{D}_5 & & & & \\ \hline \text{C}_1 & & & & \\ \hline \text{C}_3 & & & & \\ \hline \end{array} \\
 \\ \text{B}'^{-1} \quad \text{B}' \quad \text{D} \quad \text{B}'^{-1} \quad \text{S}' \\
 \end{array}$$

 图 2-8 同乘  $B'^{-1}$  矩阵运算式

3. 将等式图 2-8 化简, 就得到了原始数据 D 的恢复算法公式图 2-9。

$$\begin{array}{c}
 \begin{array}{|c|c|c|c|c|} \hline \text{D}_1 & & & & \\ \hline \text{D}_2 & & & & \\ \hline \text{D}_3 & & & & \\ \hline \text{D}_4 & & & & \\ \hline \text{D}_5 & & & & \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 \\ \hline & & & & \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline \end{array} \times \begin{array}{|c|c|c|c|c|} \hline \text{D}_2 & & & & \\ \hline \text{D}_3 & & & & \\ \hline \text{D}_5 & & & & \\ \hline \text{C}_1 & & & & \\ \hline \text{C}_3 & & & & \\ \hline \end{array} \\
 \\ \text{D} \quad \text{B}'^{-1} \quad \text{S}' \\
 \end{array}$$

图 2-9 RS 恢复数据编码

4. 对 D 重新编码, 得到丢失的校验数据块  $C_2$ , 至此所有的数据均被恢复。

在矩阵求逆的过程中, 使用的高斯消元法无法作用于定字长 w 的二进制数据。RS 编码采用伽罗瓦群  $GF(2^w)$  中定义的四则运算法则,  $GF(2^w)$  域有  $2^w$  个值, 每个值都对应一个低于 w 次的多项式, 通过这种方式, 域上的四则运算转化成多项式空间的运算。 $GF(2^w)$  域中的加法就是异或 XOR 操作, 乘法需要维护两个大小为  $2^w - 1$  的表格:log 表 gfllog 和反 log 表 gfilog。乘法公式如公式公式 2.2 所示。

$$a * b = gfilog(gfllog(a) + gfllog(b)) \% (2^w - 1) \quad (2.2)$$

### 2.3.5 破坏恢复条件

前面已经提到, 为了使原始数据再不可恢复, 针对加密后的数据, 可以有两种解决办法: 一是至少删除 n-r 份数据单元的加密数据, 使得 RS 解码数据块的条件被破坏; 二是完全删除元数据, 没有元数据参与解密运算, 此时就无法解密出原始数据。两种删除方法分别对应图 2-10 和图 2-11 所示。

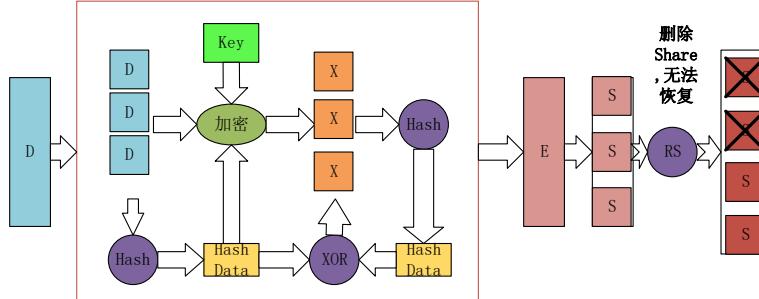


图 2-10 删 除数据块的方法

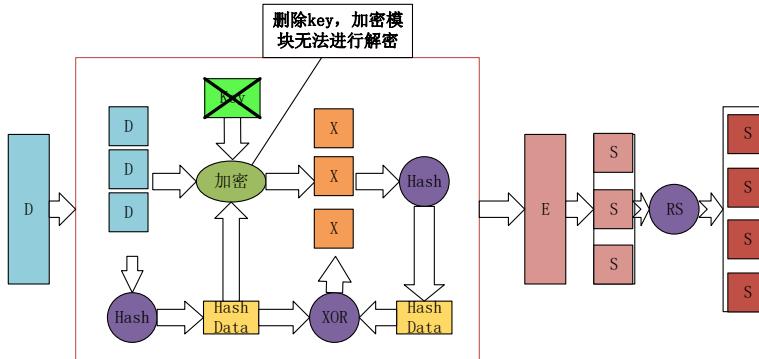


图 2-11 删 除存储的元数据

这里我们选择的是删除哈希信息数据,有如下三个原因:

- 元据通常都比较少,而且大小固定。可以利用固态盘的特殊性,安全的删除。而删除 Share 数据量太大,对固态盘阵列来说,势必会降低其使用寿命。
- 固态盘使用 TRIM 指令可以很快速地完全删除,真正地达到清除的地步。而完全删除相互独立的加密数据单元是一件非常耗时的操作,会显著影响系统的可用性,在一些频繁操作的场合,这种方案的劣势会更加突出。
- 可以将文件及相关的元数据存放在一个独立的固态盘上,需要删除时,直接使用擦除软件安全删除密钥,管理比较方便。而删除加密数据单元的手段比较费时费力,无法集中管理。

### 2.3.6 从物理介质上删除敏感信息数据

方案采用的是将哈希信息等元数据单独存储在一个固态盘上,需要删除对应的数据时。使用 ATA 低级擦除方法擦除该固态盘数据,即多次写入方法,首先数据位全部覆盖写 0,然后写入随机数据,最后再次擦除数据,达到安全删除的效果<sup>[29,30]</sup>。

## 2.4 方案整体流程

本方案采用的方案是在 SSD 组成的混合存储结构上, 分出一个单独的密钥盘存放密钥数据, 使用多个 SSD 组成盘阵, 用来存放加密后的数据单元。整体结构如图 2-12 所示。

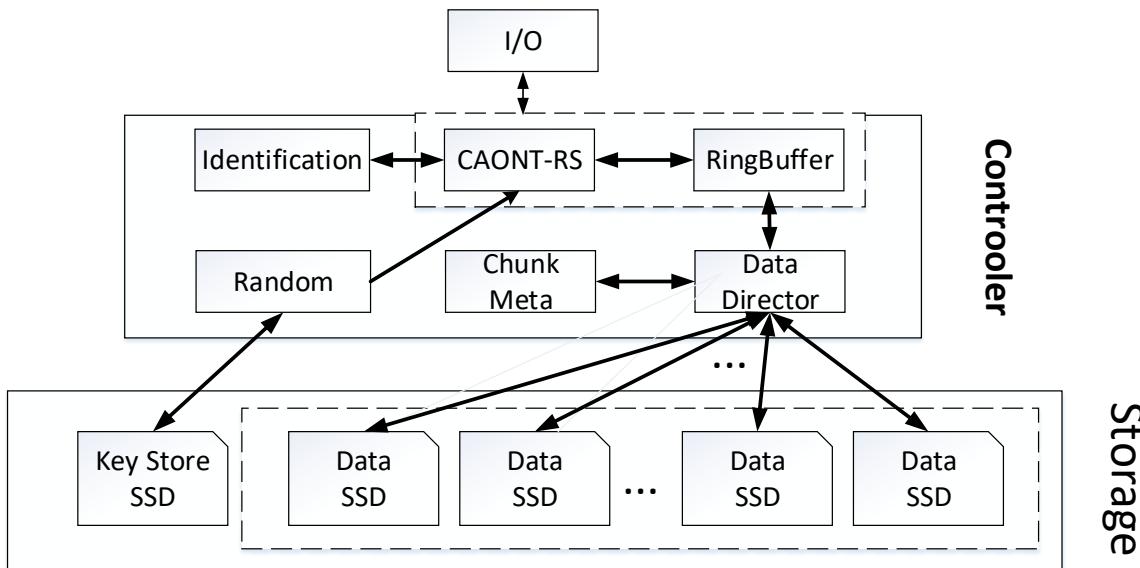


图 2-12 系统整体架构

当有 I/O 请求到来时, 逻辑控制层首先将数据分成固定大小的数据块标识每一个数据块的序号。同时, 密钥生成块产生随机 32 位数据作为密钥与数据块起输入冗余加密模块加密, 生成冗余数据块, 输出到循环队列中缓存等待存储。数据块信息注册模块录下加密后的数据块的编号, 由数据分发模块将相互独立的数据单元有序分发到数据盘。与此同时, 密钥数据存储在密钥盘。在安全删除部分, 我们只需要将密钥盘安全擦除数据即可。

下面结合方案的具体功能详细描述系统读、写、删除的流程。

### 2.4.1 系统写入数据

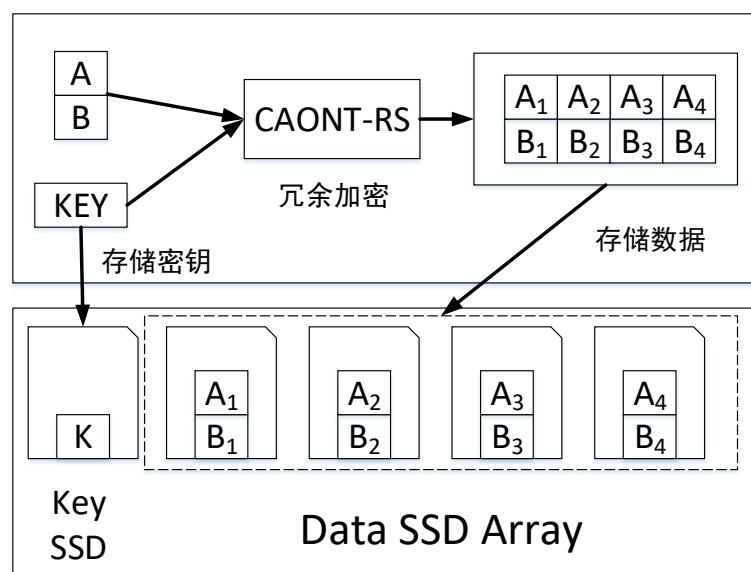


图 2-13 数据写入架构

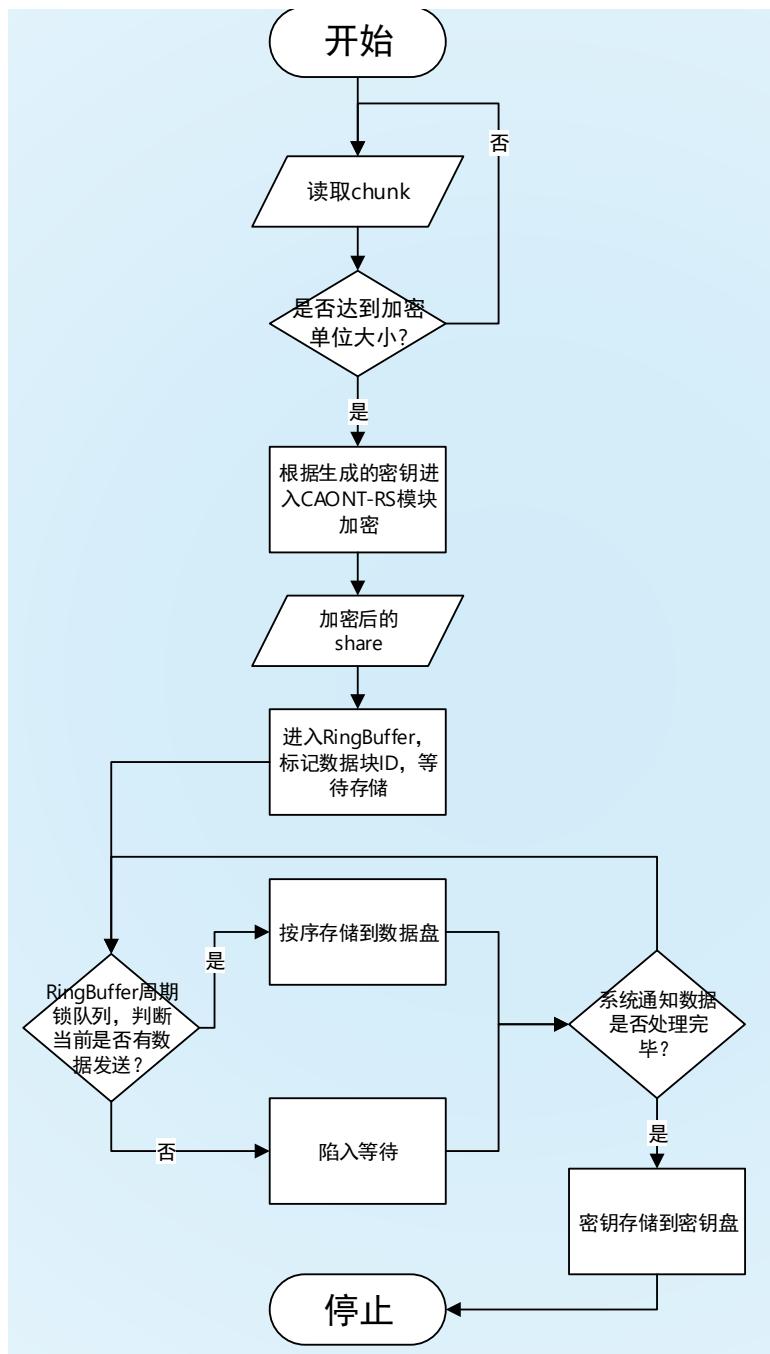


图 2-14 数据写入流程

在图 2-13的数据写入存储结构以及图 2-14数据写入流程中,当系统写入文件时,加密模块首先判断数据是否达到加密单元的大小,并且这个数据粒度可以调整。与此同时,密钥模块随机产生密钥数据,一起输入到加密模块,产生冗余的加密数据单元,并且按照顺序缓存到循环队列中等待存储,循环队列周期性地检查是否有数据需要转

发存储,将数据依据数据盘的数量依次轮询发送存储请求。数据处理完毕,密钥模块将密钥数据存放到密钥盘,整个系统的加密写入流程完成。

#### 2.4.2 系统读取数据

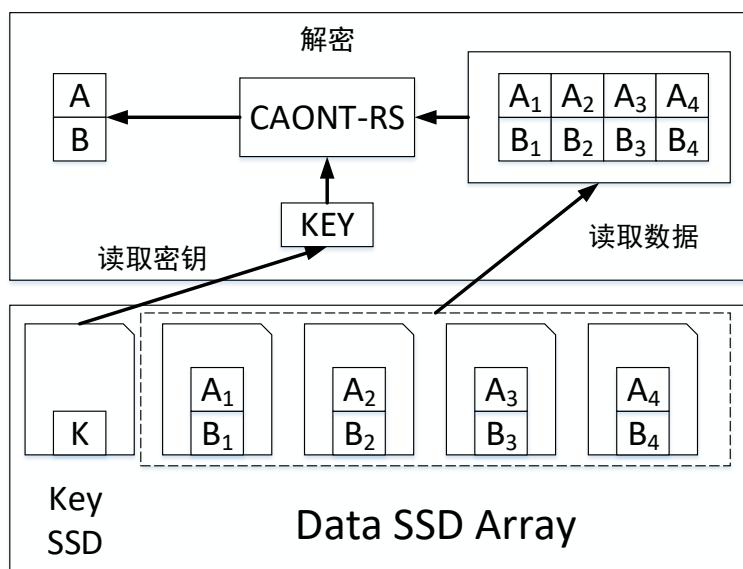


图 2-15 数据读取架构

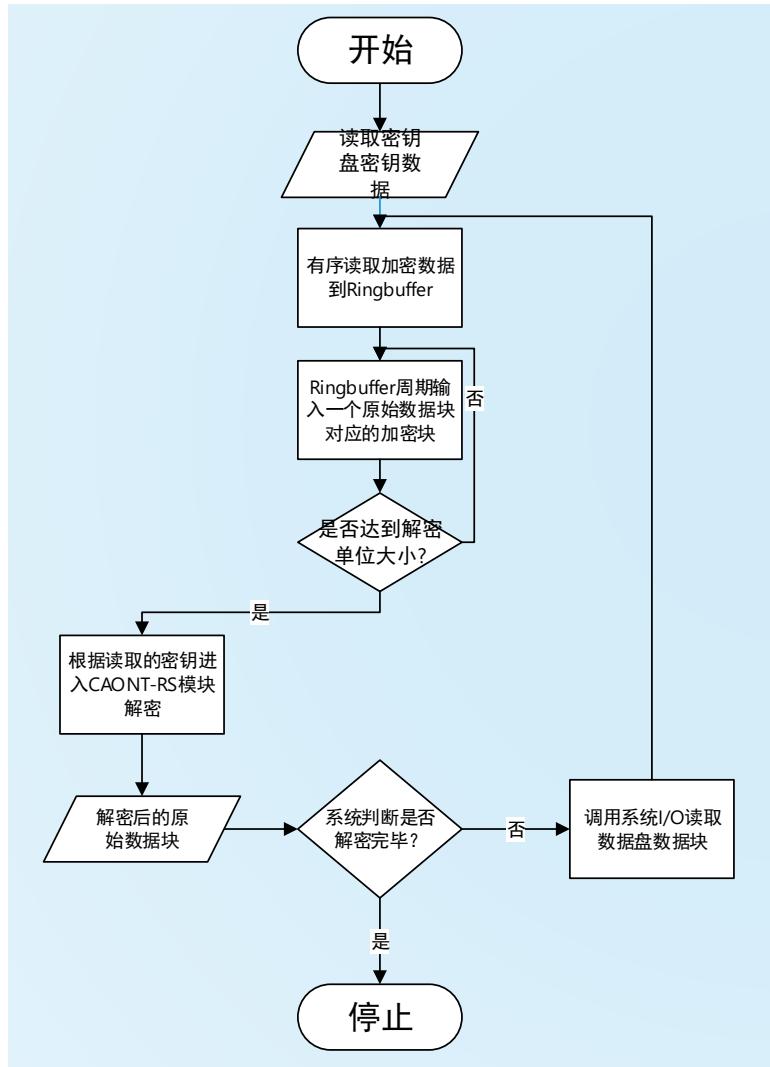


图 2-16 数据读取流程

在图 2-15 所示的数据读取结构和图 2-16 的数据读取流程中, 当需要读取原始数据时, 密钥模块首先读取密钥盘上的数据, 初始化密钥。然后数据分发模块按照次序读取各个数据盘上的加密单元数据, 并缓存在循环队列里。当数据符合解密单元大小时, 加密模块利用数据和密钥解密出原始数据块。系统处理完加密数据后, 原始数据已被全部解密还原, 数据读取流程结束。

### 2.4.3 安全删除密钥数据

由于每个单位大小的原始数据块的哈希信息很小, 因此相比加密后的数据块来说, 原始数据块的哈希信息等这些密钥相关的数据量很小, 在密钥盘上存储的区域范

围也很小,针对密钥盘的全盘擦除所花费的时间很少,而且,擦除数据时,真正有用的部分也只是针对这一块区域的数据进行覆写,从密钥盘的闪存使用寿命角度来说,这部分的擦除数据对于整体的固态盘擦除寿命影响很小。

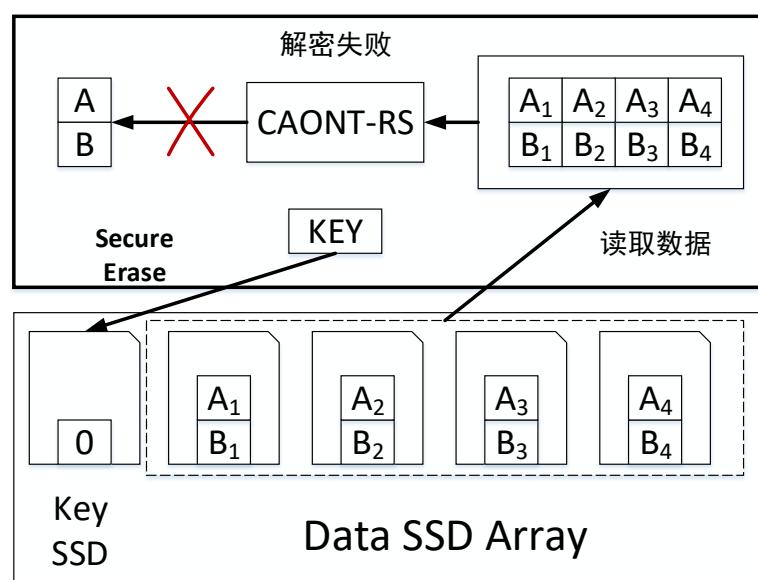


图 2-17 擦除哈希信息验证读取架构

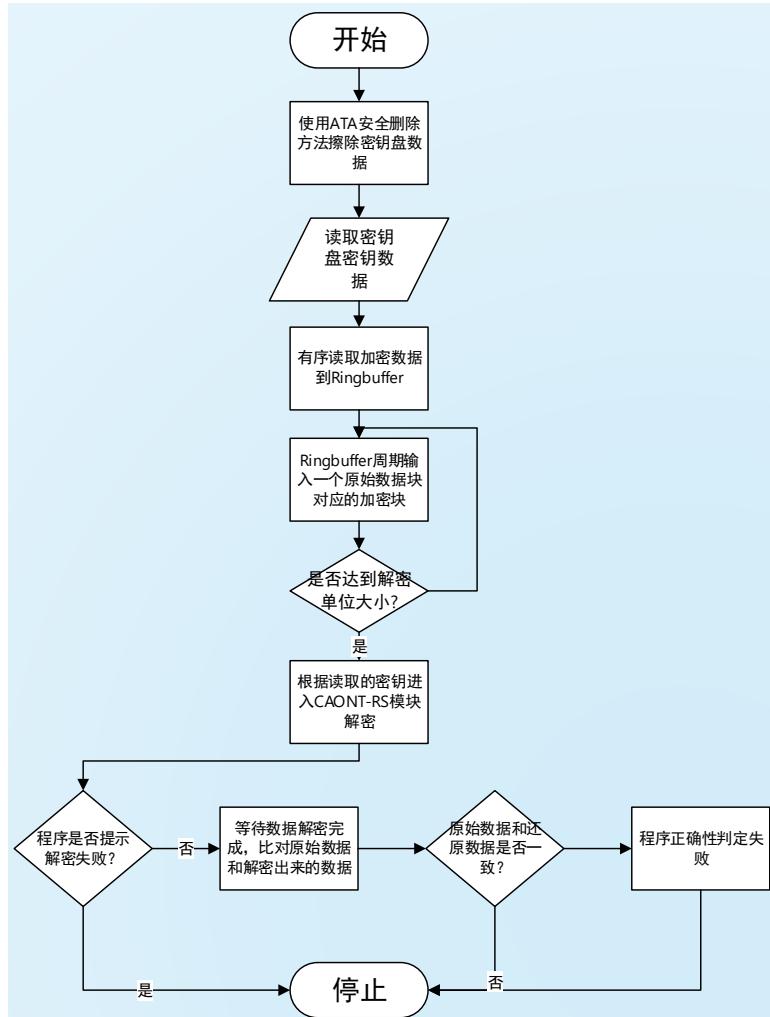


图 2-18 擦除哈希信息验证读取流程

在图 2-17所示的删除密钥信息以及图 2-18的删除后验证读取流程中, 对密钥盘进行数据覆写时, 使用了固态盘物理页复写的技术。即美国加利福尼亚大学的研究人员通过实验证实<sup>[20]</sup>, 目前市面上的 SLC 闪存基本都支持覆盖写入(只能将 1 改成 0), 即可以在擦除前对物理页进行复写(即第二次写操作), 而对闪存的错误率并不产生明显的影响。因此整个擦除手段如算法算法 2.1 进行, 在上层系统激活了删除功能时, 系统立即调用密钥盘的 TRIM 命令标记密钥盘无效数据操作, 然后针对密钥盘采取多次全盘覆写操作, 具体操作如图 2-18所示的擦除实现: 首先使用安全擦除工具擦除密钥盘上的全部数据, 这种擦除手段是先向每一个物理页覆写数据 0, 然后再次向物理页覆写随机数据, 最后覆写一次数据 0, 达到安全删除的目的。

---

**算法 2.1:** Secure Erase data Hash informathons.(Key data)

---

**Require:**  $addr$ : address of the NAND flash page;  $i$ : position of page  $data$ : page content

```
// 1. write zero to each page
data ← 0
for  $i = 0$  to SSD page total size do
    write data to page  $addr_i$ 
end for

// 2. write random bit to each page
for  $i = 0$  to SSD page total size do
    write random bit to page  $addr_i$ 
end for

// 3. write zero to each page again.
data ← 0
for  $i = 0$  to SSD page total size do
    write data to page  $addr_i$ ;
end for
```

---

此时密钥盘上的数据位全部恢复了初始值,然后密钥模块读取密钥数据,同时数据分发模块获取每个数据盘上的加密数据,放入循环队列中缓存,加密模块尝试对数据进行解密操作,如果出错则说明密钥数据已经被更改,有效地阻止了数据被还原出来。如果解密流程正常进行,则比对最后被还原的数据和初始的数据内容,两者不一致则说明系统安全删除功能正常,否则判定失败。

## 2.5 本章小结

本文是全文的重点之一,旨在阐述全闪存盘阵的数据安全删除方案的设计,首先介绍了方案依赖的前提和安全删除的技术关键,然后按照方案的各个模块定义介绍了数据加密方法,破坏数据恢复的条件以达到安全删除的目的,最后依次详细介绍了写入、读取、安全删除数据这些功能的架构和流程。

### 三 仿真系统实验设计

本方案是将原始数据块的哈希信息等密钥相关的数据存放在单独的固态盘上, 要求这种固态盘內建的 TRIM 和安全删除命令已经被厂商正确地实现, 而加密后的数据是存放在以固态盘组成的阵列上的, 每个加密数据单元存储在阵列的一个条带上, 写入数据的时候按照方案设计的缓存模块, 依次顺序写到每个条带上, 读取的时候, 由于输入缓存支持并行传输, 利用了阵列并发 IO 的特点, 快速读取到缓存中并且按序存放。要使得原始数据无法再被恢复, 擦除掉密钥盘上的哈希信息, 或者破坏方法预设的条带数量临界值的数据, 就可以达到数据安全的结果。

采用本方案可以实现多种原型系统, 例如, 基于 iSCSI 平台, 原始数据经过方案实现的系统加密, 产生的加密数据分发到以 TCP/IP 相互连通组成的阵列上, 解密数据时只要通过网络读取到各个存储点上的加密数据以及存储在单个盘上的哈希信息, 完成解密操作即可。

为了方便验证方案, 实验选择了环境相近的仿真实现。仿真实验环境也要求有一个单独的闪存盘用以存储加密的哈希信息参数数据, 并且这个固态盘能够做到数据的安全删除, 以及 TRIM 命令的內建实现, 为了容易实现, 是在以固态盘组成的阵列上划分分区, 成为一个个独立的存储单元来存放加密之后的数据单元这种设计, 来完成仿真实验。

#### 3.1 实验目的

采用仿真实验主要有以下三个方面的目的:

- 确认方案能够达到盘阵数据安全删除的要求。具有可行性。
- 仿真测试不同安全级别下, 方案在带宽、I/O 时延、系统负载、数据冗余率等维度的性能表现, 验证方案是否可靠。
- 通过较容易实现的仿真环境来进行测试, 确保方案针对所有固态盘阵都具有可用性。

#### 3.2 实验环境

由于密钥盘存储的数据很少, 本应该使用如方案所说明的那种容量很小的闪存盘, 比如 SDSASFK-016GB 这种产品, 但是因为环境的限制, 最终使用的密钥盘的型

号与数据盘一致。仿真实验环境硬件配置如表 3.1 所示。

表 3.1 实验环境硬件配置

类别	型号	数量
CPU	Intel(R) Xeon(R) L5520 @ 2.27GHz	16
内存	16GB	1
密钥盘	SV300S37A SATA3 120G	1
数据盘	SV300S37A SATA3 120G	4

### 3.3 实验方案

本次仿真实验使用了 4 块 SSD 组成的阵列，然后分为 4 个独立存储区域作为冗余加密数据的存放点。一块单 SSD 盘作为密钥存储盘。数据块信息虽加密块信息分散存储在数据盘阵列上。整体测试流程如图 3-1 所示。

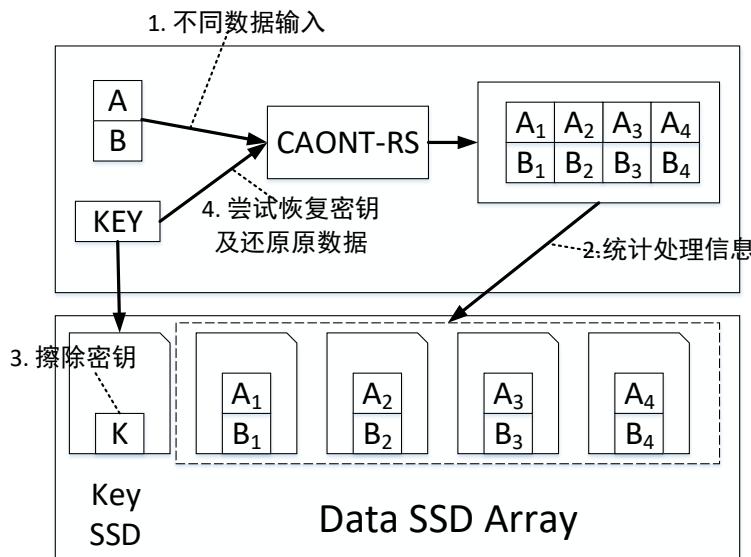


图 3-1 测试流程

- 从 1MB 到 1024MB 大小的原始数据依次输入仿真系统中
- 记录数据从读取到第一块数据被冗余加密, 分发到阵列的过程中每个环节所消耗的时间, 生成数据的大小以及系统运行状态等相关数据
- 记录加密数据读取到被加密模块解密, 最终还原为原始数据的各个环节的时间、状态等数据

4. 使用经过验证的安全删除软件将存储的密钥盘的哈希信息等数据擦除, 尝试还原数据, 验证安全删除功能
5. 统计仿真系统测试数据, 得到功能测试和性能测试等结果

仿真环境结构如图 3-2 所示。

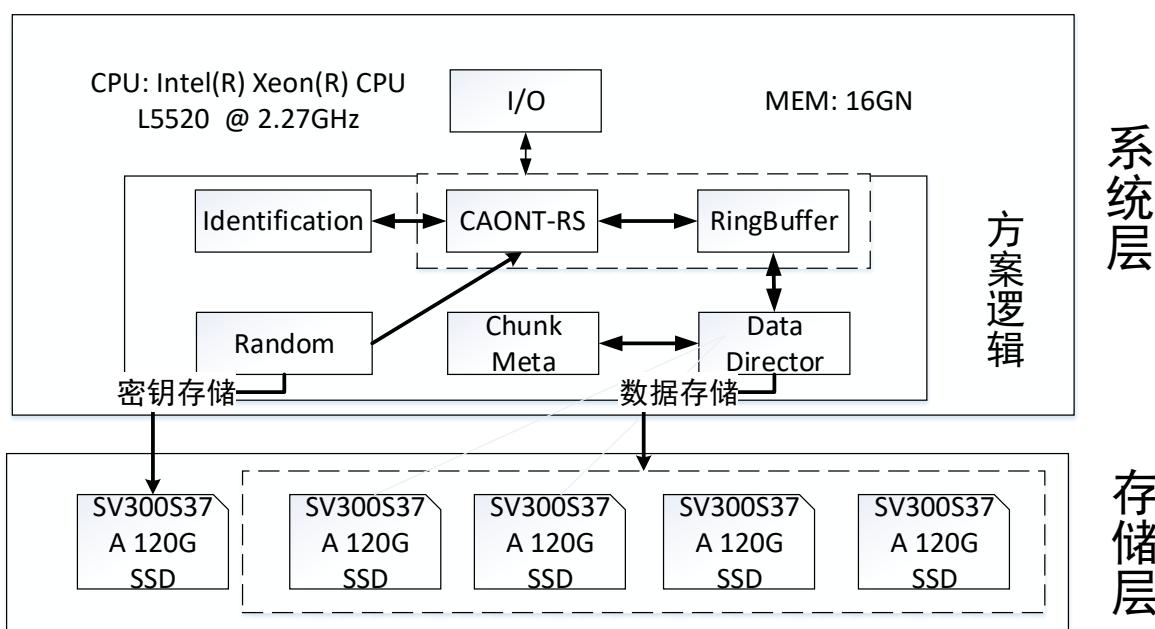


图 3-2 仿真环境系统图

### 3.4 仿真系统实现

#### 3.4.1 数据结构设计

数据结果部分, 编码和解码的逻辑处理单元都有两个分别处理输入和输出的数据缓存中心, 而这两个缓存中心得数据结构都是以线程安全的循环队列为中心, 因此, 设计一个多线程高效的循环队列成为了提高数据缓存效率的关键。图 3-3 和图 3-4 分别代表了编码输入缓存和输出缓存。

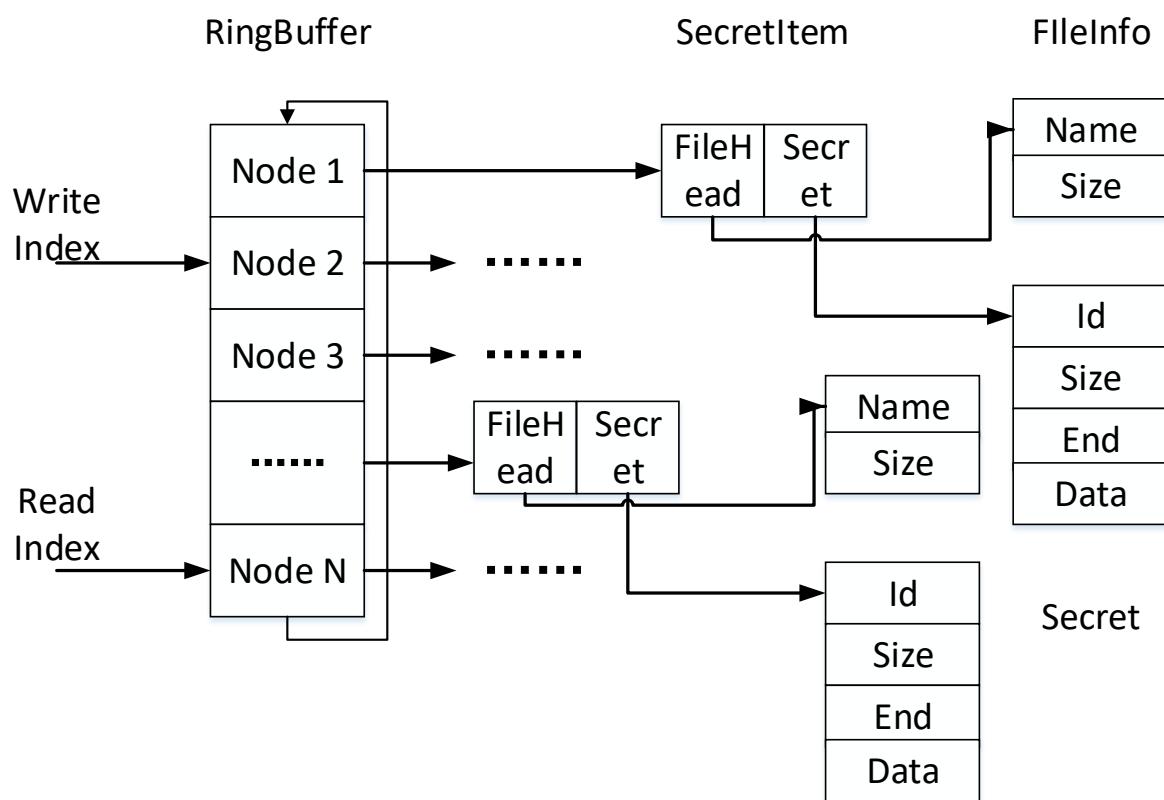


图 3-3 输入缓存数据结构设计

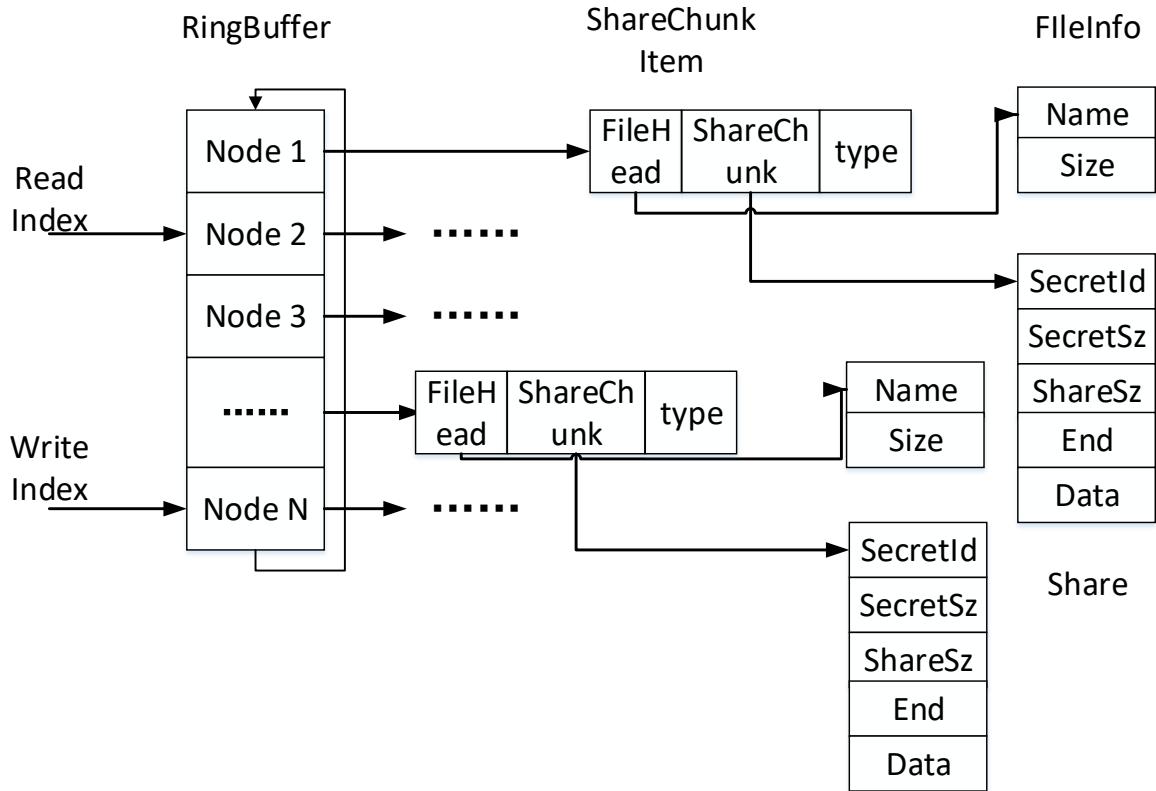


图 3-4 输出缓存数据结构设计

在数据进行编码时, 使用如图 3-3所示的缓存数据结构, 当一个数据块被加密处理完, 生成加密数据块后, 循环队列在当前写结点开辟一个新的内存, 存放原始数据块的文件名、文件大小信息, 加密子结点指向一个加密块的序号 Id, 大小 Size, 是否是最后一个加密块的标志 End, 以及最终的加密数据块 Data, 同时, 写索引向后移动一个结点位置, 这样一个加密块的缓存过程就结束了。如果写索引后面是读结点, 说明队列的缓存已满, 队列将在接下来的一段时间拒绝数据缓存请求。

当缓存需要写入到数据盘时, 循环队列的读索引获取加密数据块的 Id, 按照数据盘的个数取模, 决定存储在哪个数据盘, 队列将当前的读结点的数据传递给数据存储线程, 读索引向后移动一个结点位置, 缓存读取过程结束, 如果读索引后面位置是写结点, 则表明当前循环队列已空, 接下来的一段时间内数据存储请求将被队列忽略, 相关线程也会陷入休眠, 知道有数据到达。

加密数据解码时, 使用的是如图 3-4所示的缓存结构, 相关的读写操作刚好和编码相反, 只是输出缓存的相关结点内容发生了变化, 首先是队列的结点指向的是一个原始文件信息和加密数据块联合的结构体, 由类型标明这是一个数据块还是信息块, 其中数据块部分相比编码前是多了一个表示最终的冗余加密数据块的大小。数据块

依然是由标记 End 标明是否是最后一个数据块，在数据解密流程起到通知模块解密结束的作用。

### 3.4.2 关键流程实现

#### 3.4.2.1 将数据读写模式进行了改进优化

为了提高系统处理数据的整体性能，除了设计一种比较高效的循环队列以外，方案还对数据存取的具体环节进行了优化设计。以实验系统中的四个数据盘来说，在编码过程中，编码器会将编码完成的数据按照其编号顺序依次存入各个盘对应的循环队列中，各个队列之间的数据缓存以及写入是相互独立的，这样带来的好处就是，相比一个循环队列，数据缓存和存储的数据同步冲突更小，线程切换的次数减少，从而可以并行地存储数据到各个数据盘。

同时在解码时，由于编码算法的特性，只需要读取  $k$  份数据即可将原始数据还原出来，具体到本方案中，相比编码需要传输四份数据单元，解码的时候只要读取其中任意三份数据，有效的节省了数据传输的带宽，也减少了等待数据缓存的时间，系统处理性能自然就提升上去了。整体流程架构如图 3-5 所示。另外一个改进是，每次传输数据时，系统都会记录下当前各个数据盘的状态，依据这个状态在那些综合性能比较好的数据盘上优先读取数据，这样带来的一个好处是，系统的容错性大大提升。以测试的 4 个盘为例，假使有 1 个盘损坏，也不影响数据读取还原，在比测试规模大很多的环境下，因为数据盘的损坏可能性比较明显，这种容错性带来的优势更加明显，数据的完整性也得到了保证。同时，系统选择那些性能较好的数据盘读取数据，也有效地避免了因为单盘影响其他所有数据盘读取性能的问题。

#### 3.4.2.2 实现了密钥数据存储到密钥盘的策略

由于环境、操作系统等问题造成的不可抗力，都可能使得这套系统崩溃，由于初始密钥向量是随机化生成的，此时仍在内存中，就会因为系统崩溃而丢失，原始的数据无法再通过系统解密再恢复出来，造成了数据丢失。现有的设计对这一问题的解决方法是，在每次处理完数据加密后，密钥模块负责将本次生成的密钥数据存储到单独的密钥盘，设计的密钥存储结构如图 3-6 所示。

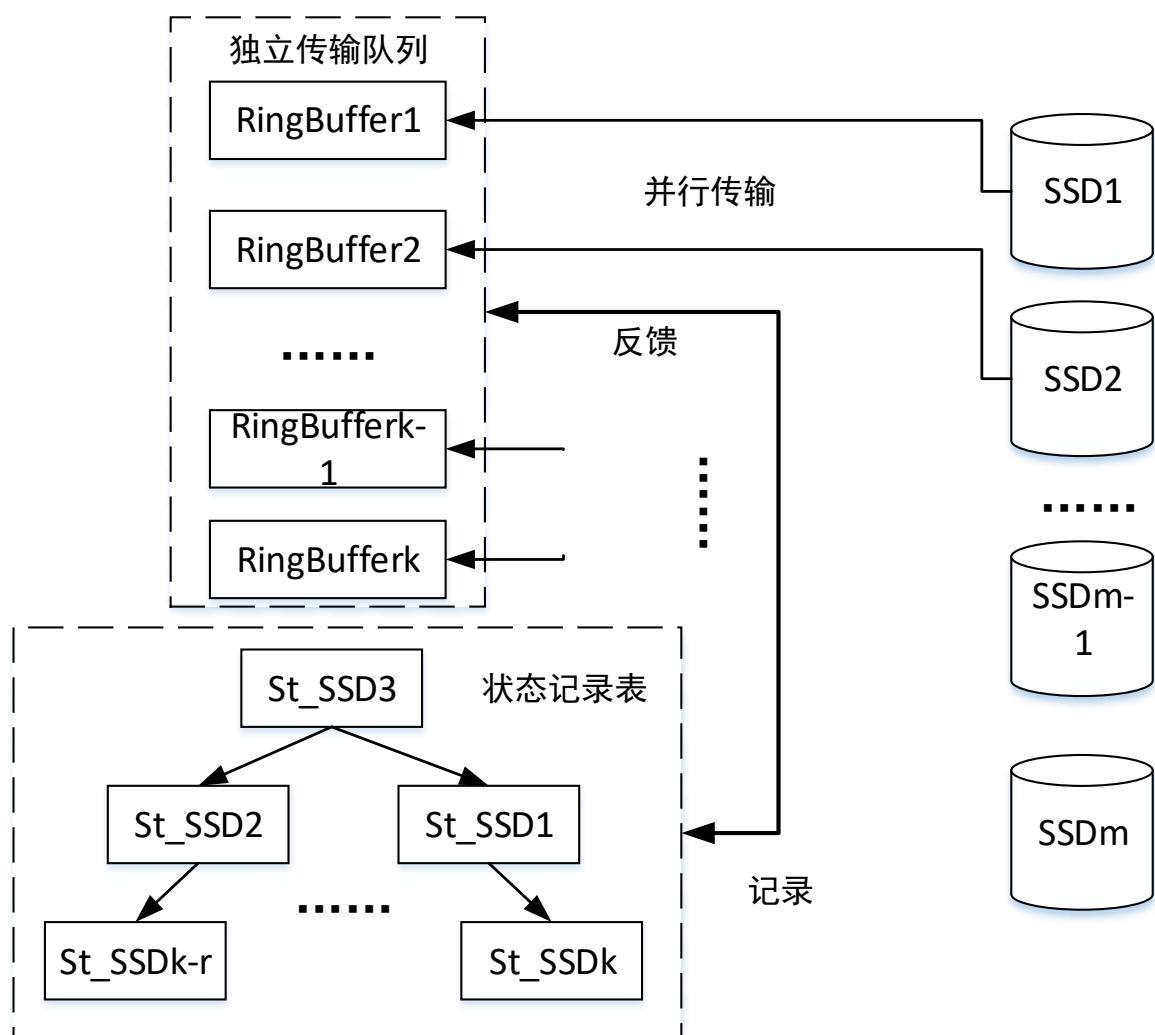


图 3-5 输出缓存架构设计

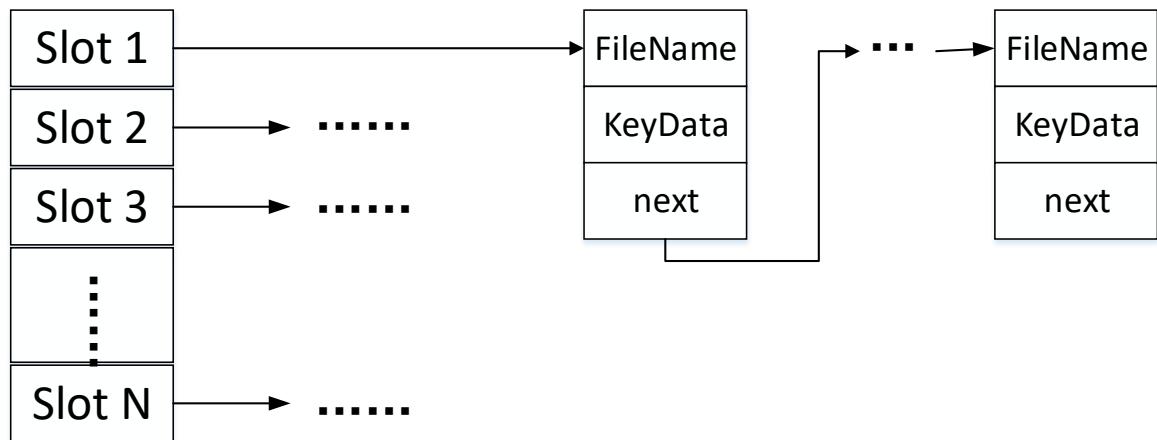


图 3-6 密钥存储结构

这样,在系统重启后,对于存储过的文件信息,系统会优先在密钥盘中查找密钥信息,如果是读取还原数据,直接使用密钥信息即可,对于重新存储请求,只需要覆盖掉原来得记录即可。

### 3.4.2.3 加密算法实现

---

**算法 3.1:** Encode origin data with hash info.

---

**Require:**  $secretbuffer, m \geq k$

$B \leftarrow bytes of per secret word$

$S \leftarrow secret buffer size$

$A \leftarrow aligned secret buffer size$

**Ensure:**  $A \leq \min\{(S + B) \% (B + K), B * (k - 1) / (B * k + 1)\}$

$shareSize \leftarrow (A/B + 1)/k * B$

$alignedbuffer \leftarrow secretbuffer$

$Key \leftarrow GenerateHash with alignedbuffer.$

**for** i = 0 to A **do**

$alignedsizeconstantArray_i \leftarrow i \& 0xff;$

**end for**

    erasure coding data  $\leftarrow$  Encrypt with aligned size constant Array and hash info.

    erasure coding data  $\leftarrow$  Galois Group multiply region hash info with value 1 and, step by B, adding by 1.

    share buffer  $\leftarrow$  erasure coding data

**for** i=0 to m **do**

**for** j=0 to k **do**

$value \leftarrow distribution_{k^2 + k*i + j};$

            share buffer  $chunk_{k+i} \leftarrow$  Galois Group multiply region, share buffer  $chunk_j$  with value , step by shareSize, adding by 1 if  $j \neq 0$  else 0

**end for**

**end for**

**return** share buffer

---

加密算法算法 3.1 中, 关键的操作主要有, 首先根据对齐的原始数据块生成它的哈希值, 作为该数据块加密的密钥, 使用常向量 **constant array** 加密并且群运算之后得到初始的加密数据块, 之后就按照每个冗余数据单元的大小从前往后依次群运算得到每个加密的数据块, 后面的加密单元依赖前面已生成的加密单元。

### 3.4.2.4 解密算法实现

---

**算法 3.2:** Decode encrypted data with Key(hash info)

---

**Require:** share buffer,kShareID,k

B  $\leftarrow$  bytesof per secret word

S  $\leftarrow$  secret buffer size

A  $\leftarrow$  aligned secret buffer size

**Ensure:** shareSize % B = 0  $\vee$  erasure codeing data size  $\geq shareSize * k$

**for** i=0 to k **do**

**for** j=0 to k **do**

*squaredmatrix*<sub>k\*i+j</sub>  $\leftarrow$  *distribution*<sub>k\*kShareID<sub>i+j</sub></sub>

**end for**

**end for**

    invert squared matrix

**for** i=0 to k **do**

**for** j=0 to k **do**

            value  $\leftarrow$  *squaredmatrix*<sub>k<sup>2</sup>+k\*i+j</sub>;

            erasure coding data *chunk<sub>i</sub>*  $\leftarrow$  Galois Group multiply region share buffer *chunk<sub>j</sub>*

            with value , step by shareSize,adding by 1 if  $j \neq 0$  else 0.

**end for**

**end for**

    Key  $\leftarrow$  Generate Hash with erasure coding data.

    Key  $\leftarrow$  Galois Group multiply region erasure coding data with 1 ,step by B,adding by 1.

    erasure coding data  $\leftarrow$  Encrypt with aligned size constant Array and hash info.

    aligned sercet buffer  $\leftarrow$  Galois Group multiply region erasure coding data with 1 ,step by aligned secret size,adding by 1.

    erasure coding data  $\leftarrow$  Generate Hash with aligned sercet buffer.

    secret buffer  $\leftarrow$  aligned sercet buffer

**return** secret buffer

---

解密算法算法 3.2 中，首先根据分散向量和当前的冗余数据单元运算得到解密矩阵 *squad matrix*，然后逆置，并作为接下来的群运算的乘积因子，从冗余数据单元的每个数据块从前往后依次得到相应的初始加密数据，初始加密数据的末尾信息部分在群运算和解密操作后得到原始数据块的哈希值，使用该哈希值与初始加密数据再次群运算和取哈希操作，就得到了最终的原始数据块。

### 3.5 本章小结

本章阐述了依据全闪存盘阵数据安全删除的原型系统实现的仿真系统。主要分为四个部分，首先介绍了采用仿真实验的目的和仿真环境的系统结构，其次列出了仿真实验的大体步骤，最后详细描述了仿真系统中的数据结构设计和关键流程实现，总结出系统的存储哈希信息等数据到单独的密钥盘，以及加解密算法设计的实现。

## 四 测试与分析

本章介绍测试的环境、使用的工具和测试的方法。测试结果表明，本安全删除方案能够保证在删除原始哈希信息后，原始数据无法恢复；对仿真的带宽测试，对比固态盘的硬件性能，能够充分利用固态盘的 I/O 读写能力；系统 I/O 整体时延比较小；系统负载属于正常水平，占用的计算资源和内存资源较为正常；加密后的数据冗余稳定在一个固定的范围内。验证了方案的可用性和可靠性。

### 4.1 测试平台及环境说明

测试环境使用的硬件配置同仿真实验平台一致。

### 4.2 功能测试

系统各项子功能测试结果如表 4.1 所示。

表 4.1 系统各项功能测试结果

操作	实验状态	结果	结论
写入测试文件	密钥盘、数据盘均写入数据, 加密数据与原始数据内容完全不同, 且是不可读文本	加密功能正常, 数据写入成功	写入子功能正常
读取测试文件	密钥盘、数据盘有数据传输, 解密还原的文件内容与原始文件一致	解密功能正常, 数据读取成功	读取子功能正常
安全删除 密钥盘数据, 尝试还原测试 文件	系统还原获取的 文件内容与原始文件 不一致	密钥数据 已经改变	安全删除子功能正常
擦除其中 两个数据盘上的 数据, 再次 还原测试文件	系统解密过程 出错, 提示 缺少文件数据	加密数据已 被部分删除	安全删除子功能正常

### 4.3 I/O 带宽测试

仿真方案处理 IO 读取时, 首先需要将盘阵中将一个数据单元对应的全部加密数据单元读取到内存中, 同时在密钥盘中读取到这个数据块的原始哈希信息, 才能进行解密操作得到原始数据单元。测试读取带宽就是衡量这个整体过程中数据处理的效率。读取带宽结果如图 4-1所示。

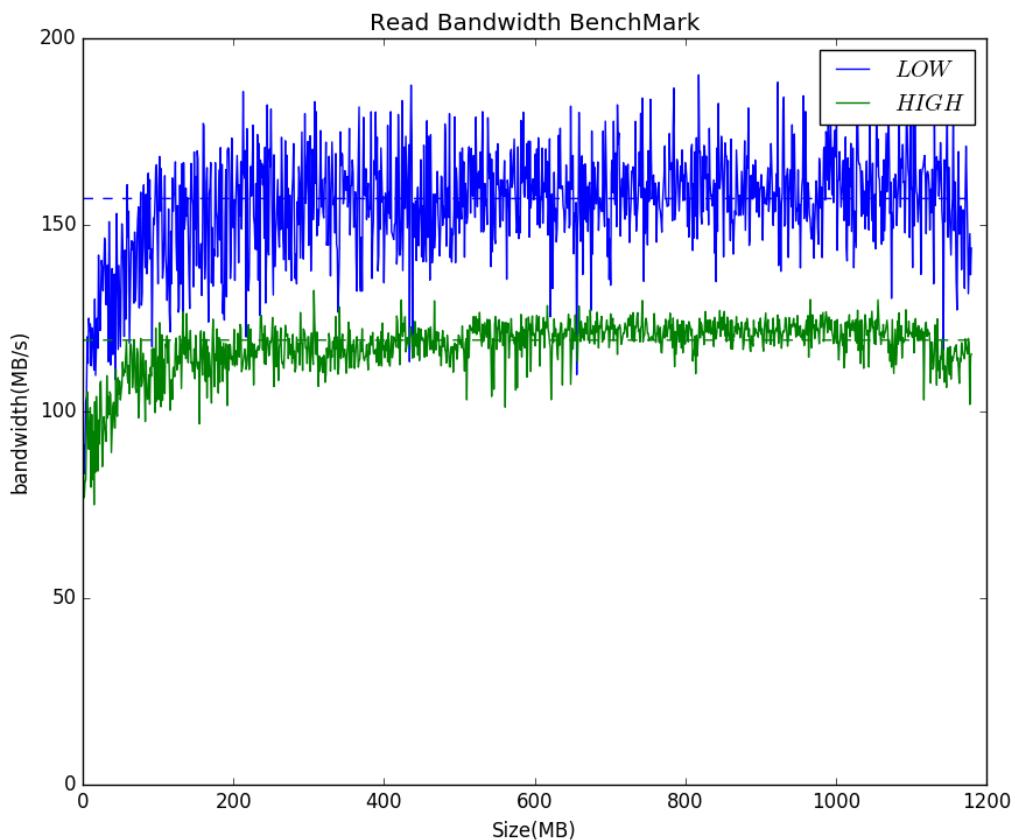


图 4-1 数据读取带宽

仿真系统处理数据写入时,首先将原始数据块按照预定义的数据大小分成一个个数据单元,根据系统的缓存参数和目标文件的大小来决定缓存的数据单元数量,并开始针对每个数据单元完成冗余加密操作,后端将缓存的加密数据单元按照次序写入阵列中存储,写入带宽衡量的是这整个过程中原始文件完成冗余加密存储的处理带宽。写入带宽测试结果如图 4-2所示。

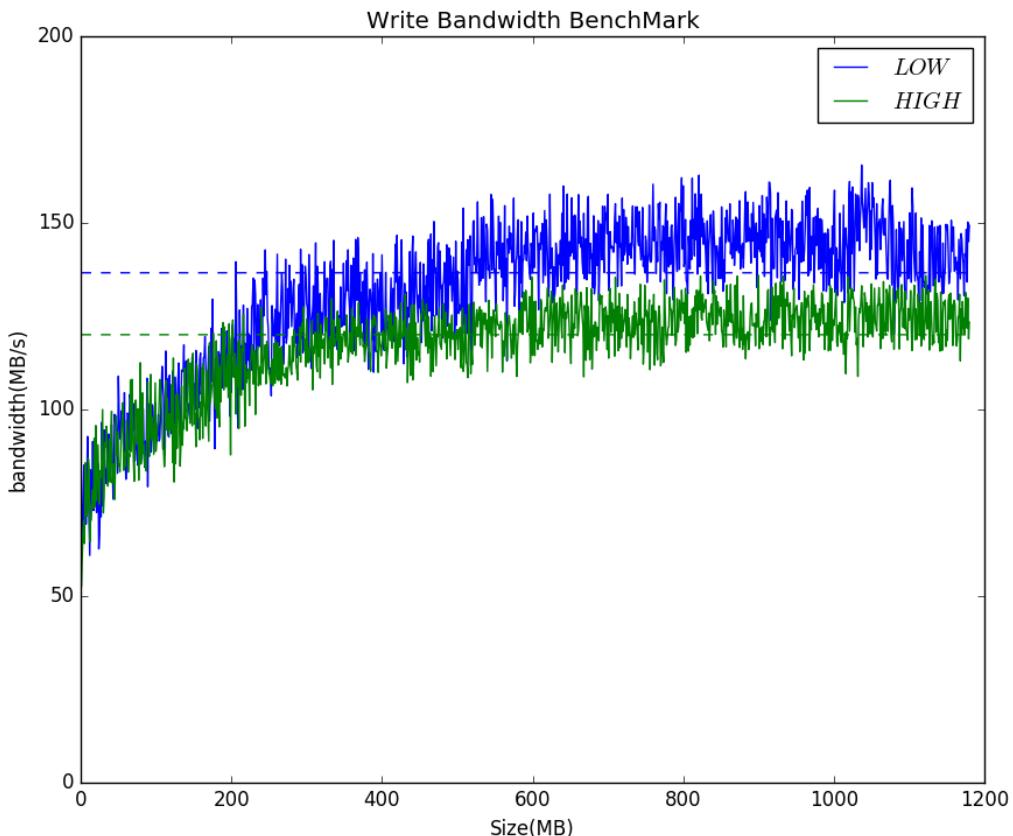


图 4-2 数据写入带宽

从读写带宽的测试结果可以看出,在一般加密级别下,数据读取带宽平均水平在 158MB/s 左右,而数据写入带宽大约为 130MB/s。在高强度加密级别下,数据读写带宽整体都比一般加密级别要低一些,其中读取带宽大约是 129MB/s,而写入带宽也是在 120MB/s 左右。

整体上,高强度加密级别的数据处理带宽比一般级别的要低,是因为数据单元在经过非对称加密的过程所耗费的时间要长,但是这种情况是在原始数据量大的情况下,此时加密模块的操作耗费时间成为仿真系统的主要瓶颈。在数据量较小时,二者的差别主要在数据预取和数据缓存的不同上,差别不是很大。

用于存储的 SSD 盘规格是读取速度为 180MB/s,写入速度 133MB/s。与原始盘的读写速度相比,仿真系统在写入状态下基本已经达到峰值,而读取状态的损失大约分别在 12.2% 和 33.3%,考虑到日常使用对固态盘的损耗,方案的读取性能是可以接受的。

另外,仿真系统的读写带宽最终都稳定在平均水平附近,波动很小,可以得出,系统在处理数据带宽方面,具有一定的稳定性。

#### 4.4 I/O 时延测试

I/O 延迟测试,在数据被读取时,是计量从发出请求,到缓存模块将第一个数据单元对应的加密数据单元全部取出,推送到解密模块处理的时间。读取时间延迟测试结果如图 4-3 所示。

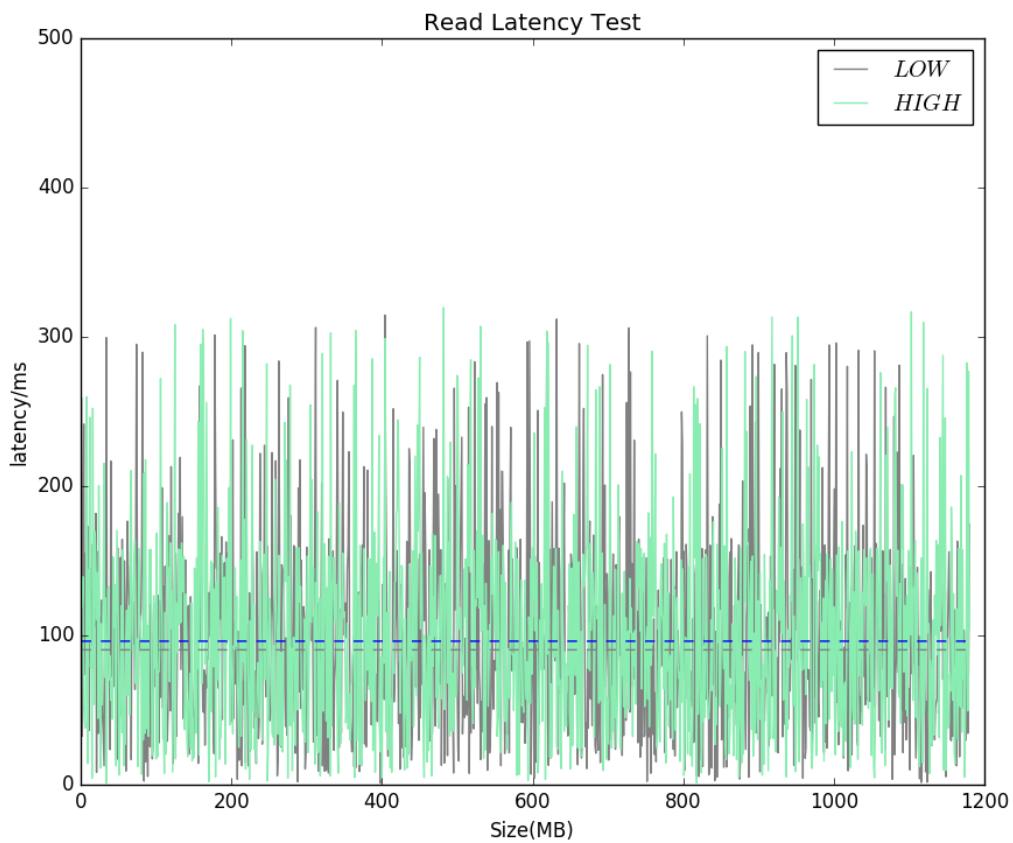


图 4-3 系统读取延迟

在写入数据时,是记录第一个数据单元推送到加密模块的时间。写入时间延迟测试结果如图 4-4 所示。

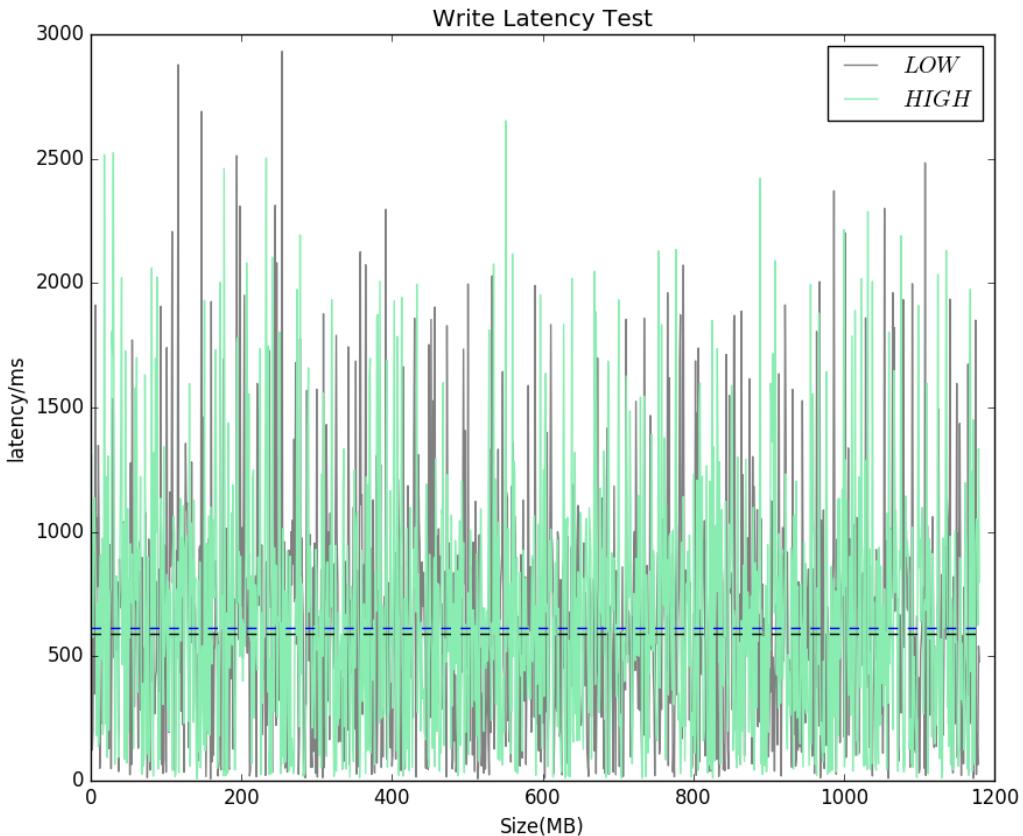


图 4-4 系统写入延迟

从时间延迟测试结果来看,仿真系统的读取延迟和写入延迟在不同加密级别的差别很小,其中读取延迟大约为 90ms 左右,读取延迟最差不超过 300ms。写入延迟均在 600ms 附近波动,最差情况不超过 3000ms。两种测试情形下,系统时延波动都比较显著,然而大部分延迟数据都集中在平均水平,从这点来看,系统的时延也具有一定稳定性。

## 4.5 系统负载测试

资源占用测试分为代表仿真系统需要的计算资源和内存资源,系统在对数据进行冗余加密时需要耗费更多的 CPU 时间片,在缓存读取数据和加密后数据时需要额外的内存空间,因此,从进程的 CPU 占用和内存占用就可以反映整体系统运行需要的环境。其中前端负载表示的主要 IO 请求、加密这些即时性的负载,后端表示的是数据存储、读取这些长时间的负载。系统不同安全级别下的 CPU 负载如图 4-5 和图 4-6 所

示。

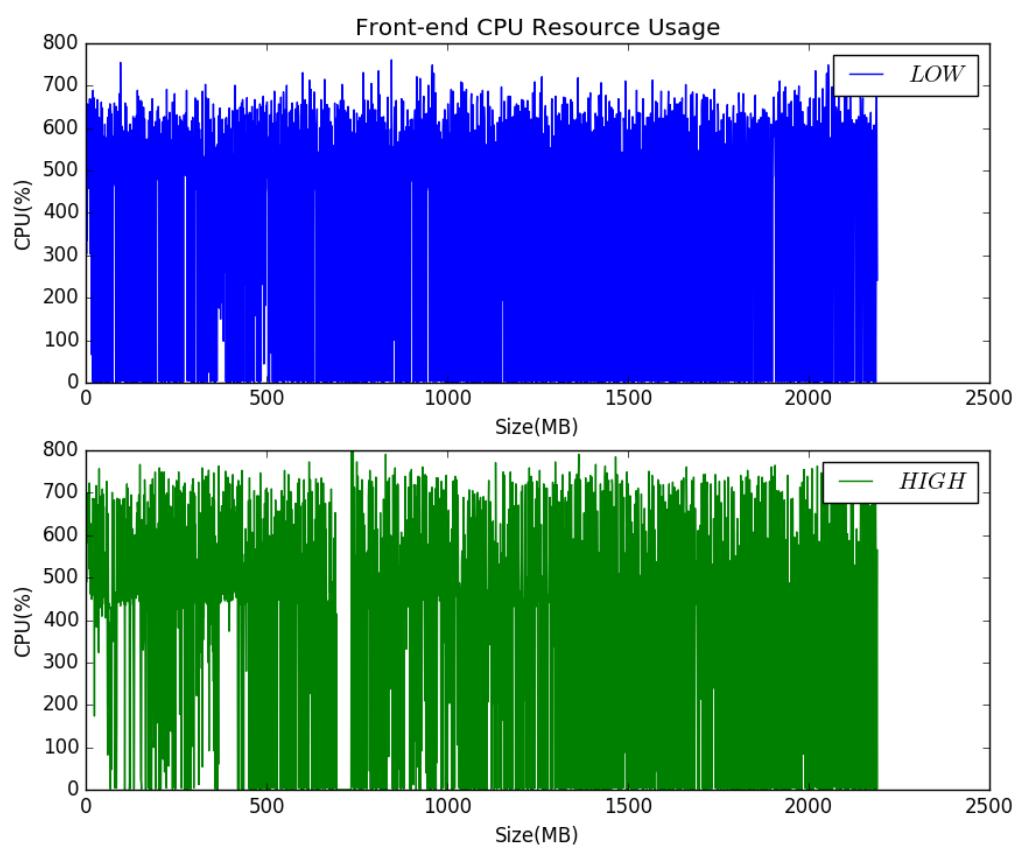


图 4-5 系统前端 CPU 占用

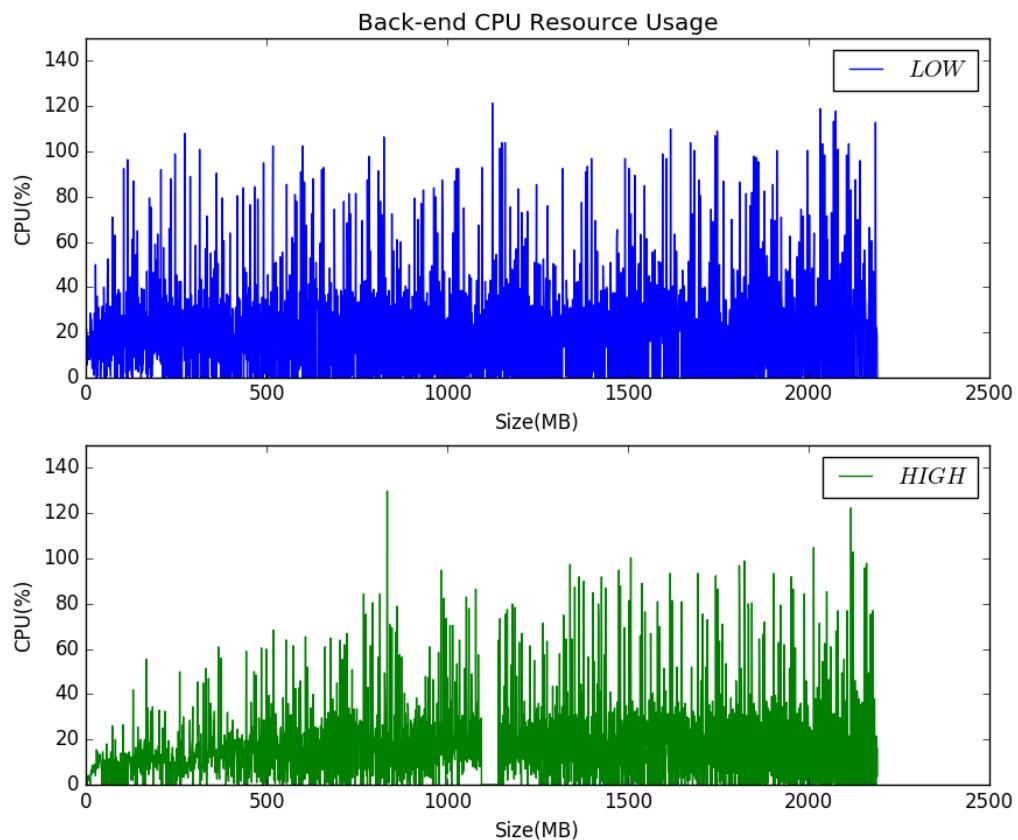


图 4-6 系统后端 CPU 占用

前后端内存负载分别如图 4-7和图 4-8所示。

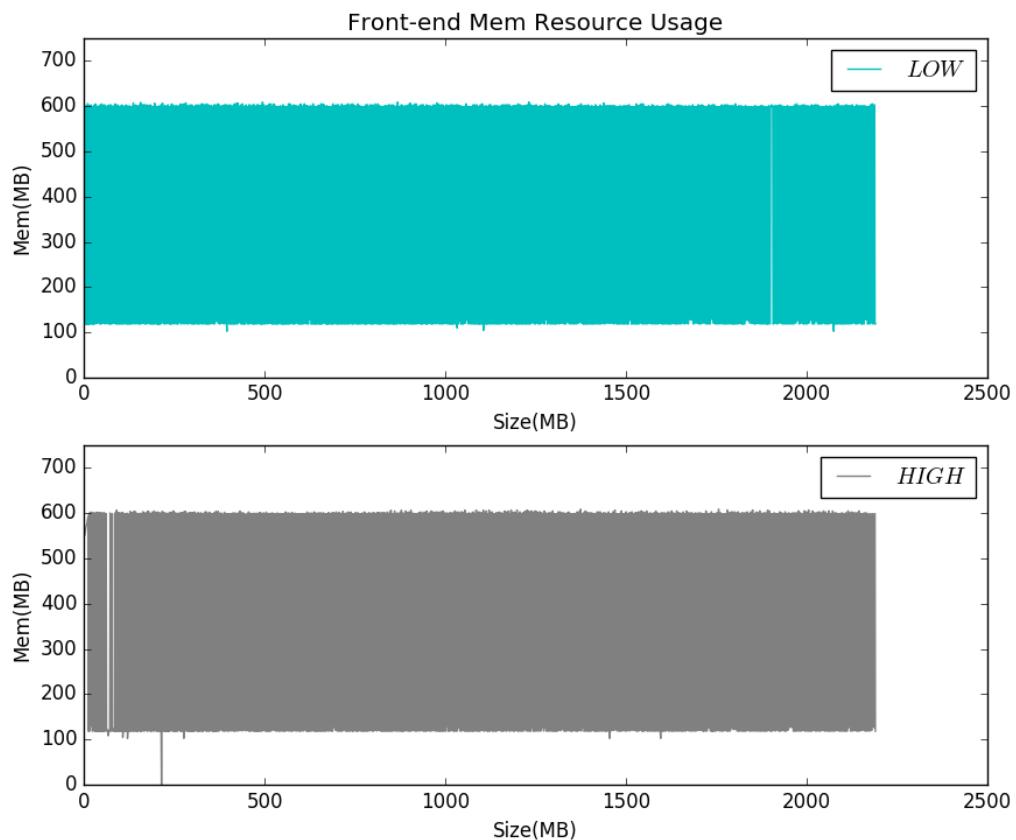


图 4-7 系统前端内存占用

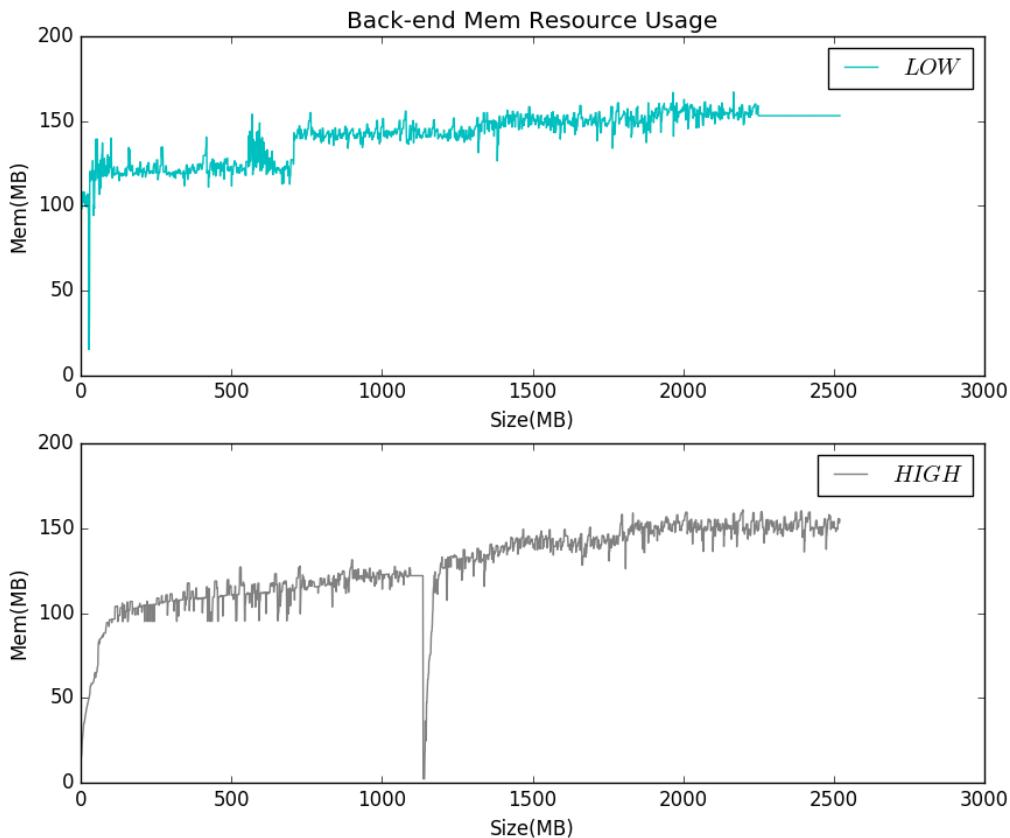


图 4-8 系统后端内存占用

从负载测试结果来看,由于前端处理的任务具有瞬时性,加密操作耗费计算资源,所以两种加密级别的 CPU 占用都在 0 到 700% 之间变化,内存负载也在 100MB 到 600MB 左右变动,可以推断,仿真系统空闲时,前端几乎不占用计算资源,内存占用也在 100MB 左右。前端总体负载差异较大。

后端处理的任务主要是数据的缓存,读写这些请求,具有长期性的特点,不同的 IO 请求对其影响比较平稳,因此测试结果显示的数据趋势比较平缓,其中 CPU 负载保持在 20%-40%,内存负载保持在 100MB-150MB 范围内。后端总体负载比较小且稳定。

## 4.6 数据冗余率测试

数据冗余表示加密后的数据总量相比原始数据大小的差值,与原始数据的比值,由于系统设计方案是一种冗余存储系统,一个原始数据单元经过处理后,会产生几份

冗余的加密数据,因此通过这个比值来判断方案在数据存储层面的可用性,冗余率越低,加密数据占用的存储单元越少,方案的实用性就更好。不同大小的数据冗余率如图 4-9 所示。

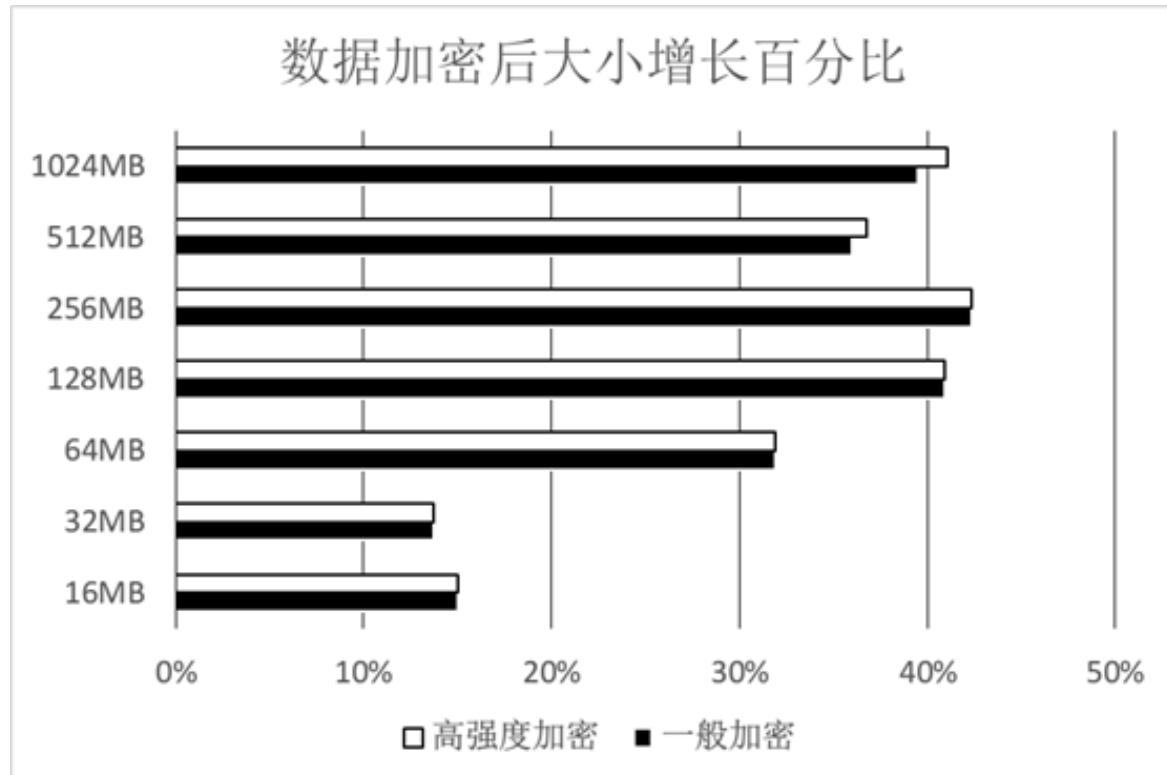


图 4-9 数据冗余率测试结果

两种强度加密系统,加密后的数据增长率在小文件(<16M)时,基本为0,而在中型文件(16M 64M)时大约为16% 20%,在大文件(>64M)时大约在38% 41%,不同大小的存储需求,系统的冗余率分为几个区间,在这些区间中变化。

结合上面的一些结论,可以发现,我们的安全删除方案不同安全级别的性能表现确实有差异;而且,整个系统基本可用。由于在物理上删除了 key,在测试中原始数据无法恢复,系统提示错误。由此得到结论,如果彻底破坏密钥存储数据的话,数据是完全无法恢复的。

功能测试部分确保了方案的写入、读取、加密、删除功能是可以正常工作并且产生预期效果的,也就是证明了我们方案的可行性。

系统测试分别从带宽、时延、负载、冗余量这四个维度,详细测试了系统的相关性能,根据测试结果的表现并加以分析,初步证明了方案的可靠性。

## 4.7 本章小结

本章对全闪存盘阵数据安全删除的仿真系统进行了功能测试和性能测试,测试平台采用的是仿真实验的实现平台,功能测试部分主要测试了仿真系统的写入、读取、安全删除这些子功能,验证了系统在安全删除关键信息后,数据无法恢复的方案要求。性能测试部分通过对系统处理带宽、I/O 时延、系统负载和冗余率的测试,验证了系统的可靠性。

## 五 总结与展望

相比目前大量被使用的机械硬盘，固态硬盘由于其读写性能和 I/O 延迟等优点，必将成为未来数据存储的主流硬件产品。然而，在阵列领域，由于固态盘的异地更新可能导致被删除数据泄露的问题，其安全性得不到保障。针对这一问题，本文在全闪存盘阵的基础上，设计了一种安全删除方案，将原始数据经过冗余加密后，如果擦除存储的密钥数据，或者擦除一定数量的加密数据，原始数据将再也无法被还原出来，从而实现了数据的安全存储。本文重点介绍了方案的详细实现，以及加密流程是如何设计，怎样擦除关键数据保证数据无法恢复，然后，按照方案的设计思想，实现了一套仿真系统，着重描述了仿真实现中的数据结构设计和相关算法设计，并结合一系列的测试结果来论证方案的可行性和可靠性。

设计方案和实现仿真实验的过程中，遇到了很多问题。结合这些问题，完成的主要工作如下。

1. 调研了固态盘的相关工作机制，由于固态盘的异地更新，原始被删除的数据，可能发生泄露问题。以及在 TRIM 命令在安全删除过程中的作用，TRIM 命令和数据多次覆写方法是固态盘安全删除的基础。并了解了相关磁盘阵列的存储方式。
2. 在前人相关的研究基础上，实现并改进了冗余加解密算法，设计了仿真系统中的输入输出缓存数据结构，为了实现安全删除功能，将密钥数据单独存储在一个固态盘上。优化了数据处理的流程之后，使得系统整体的性能相比原生的读写没有明显的损失，达到了数据安全性和性能的平衡。
3. 方案针对的环境，不仅针对阵列，而且在 iSCSI 这些平台上，也能够发挥其作用。只要满足密钥数据存储的固态盘支持 TRIM 命令这一条件，方案就可以在该环境下正常工作，发挥安全删除的作用，具有一定的平台通用性。

目前方案依然存在一些不足之处，由于固态盘阵列的复杂性，没有实现方案对应的原型系统。另外，系统中的密钥数据是存储在单独的密钥盘上，密钥盘容量通常很小，全盘擦除的代价比较小，在数据安全性和硬件代价上能够得到比较好的平衡。如果能够将密钥数据存储在盘阵的固定位置，结合阵列的运行机制，实行定点删除，将会使方案的安全性得到极大提升。

## 致 谢

能完成本研究，首先要感谢我的导师曾令仿老师，是您在忙碌中还对我谆谆教导，细心检查科研过程中的错误。您的每一次指导，都让我学习到了很多东西。感谢谭支鹏老师，在这两年的时间里对我的关心和帮助，感谢实验室团队里的其他老师和学生，你们为实验室团队辛苦付出着，我们才有这么好的科研环境。感谢刘文国师兄，在我遇到困惑时能够给予我帮助和启发。

本文受国家自然科学基金项目“面向非易失存储器的一站式存储访问模型及技术研究”(61472153)，湖北省自然科学基金项目“基于固态盘和瓦记录磁盘的冷存储系统关键技术研究”(2015CFB192)资助。

## 参考文献

- [1] 固态硬盘技术解析之垃圾回收和 TRIM 指令. <https://dirtysalt.github.io/ssd-gc-and-trim.html>. Accessed Dec 8, 2016.
- [2] O&O Software GmbH. O&O SafeErase 7. <http://www.oo-software.com/en/products/oosafeerase>. Accessed Dec 8, 2016.
- [3] Secure Eraser. <http://www.secure-eraser.com/>. Accessed Dec 8, 2016.
- [4] BCWipe. <http://www.secure-eraser.com://www.jetico.com/products/personal-privacy/>. Accessed Dec 8, 2016.
- [5] Yaakobi E, Grupp L, Siegel P H, et al. Characterization and error-correcting codes for TLC flash memories. in: Proceedings of International Conference on Computing, NETWORKING and Communications, 2012, 486-491.
- [6] Shibata N, Kanda K, Hisada T, et al. A 19nm 112.8mm<sup>2</sup> 64Gb multi-level flash memory with 400Mb/s/pin 1.8V Toggle Mode interface. in: Proceedings of Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International, 2012, 422 - 424.
- [7] Perlman R. File System Design with Assured Delete. in: Proceedings of IEEE International Security in Storage Workshop, 2005, 6 pp. - 88.
- [8] Tang Y, Lee P P C, Lui J C S, et al. Secure Overlay Cloud Storage with Access Control and Assured Deletion. Dependable & Secure Computing IEEE Transactions on, 2012, 9(9):903–916.
- [9] Perlman R. The ephemeralizer: Making data disappear. 2005, 1:51–68.
- [10] Tang Q. Timed-ephemeralizer: make assured data appear and disappear. in: Proceedings of Public Key Infrastructures, Services and Applications - European Workshop, Europki 2009, Pisa, Italy, September 10-11, 2009, Revised Selected Papers, 2009, 195-208.
- [11] Geambasu R, Kohno T, Levy A A, et al. Vanish: increasing data privacy with self-destructing data. in: Proceedings of Usenix Security Symposium, Montreal, Canada, August 10-14, 2009, Proceedings, 2009, 299-316.
- [12] Wolchok S, Hofmann O S, Heninger N, et al. Defeating Vanish with Low-Cost Sybil Attacks Against Large DHTs. in: Proceedings of Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, Usa, February - March, 2010.
- [13] Zeng L, Shi Z, Xu S, et al. SafeVanish: An Improved Data Self-Destruction for Protecting Data Privacy. in: Proceedings of Cloud Computing, Second International Conference, CloudCom 2010, November 30 - December 3, 2010, Indianapolis, Indiana, USA, Proceedings, 2010, 521-528.
- [14] Reimann S, Dürmuth M. Timed revocation of user data: long expiration times from existing infrastructure. in: Proceedings of ACM Workshop on Privacy in the Electronic Society, 2012, 65-74.
- [15] 王丽娜, 任正伟, 余荣威, et al. 一种适于云存储的数据确定性删除方法. 电子学报, 2012, 40(2):266–272.
- [16] Wang G, Yue F, Liu Q. A secure self-destructing scheme for electronic data. Journal of Computer & System Sciences, 2010, 79(2):279–290.
- [17] 熊金波, 姚志强, 马建峰, et al. 云计算环境中的组合文档模型及其访问控制方案. 西安交通大学学报, 2014, 48(2):25–31.

- [18] 熊金波, 姚志强, 马建峰, et al. 基于属性加密的组合文档安全自毁方案. 电子学报, 2014, 42(2):366–376.
- [19] 熊金波, 姚志强, 马建峰, et al. 面向网络内容隐私的基于身份加密的安全自毁方案. 计算机学报, 2014, 37(1):139–150.
- [20] Wei M, Grupp L M, Spada F E, et al. Reliably Erasing Data From Flash-Based Solid State Drives. in: Proceedings of Usenix Conference on File and Storage Technologies, San Jose, Ca, Usa, February, 2011, 8-8.
- [21] Peterson Z N J, Burns R, Herring J, et al. Secure Deletion for a Versioning File System. in: Proceedings of FAST '05 Conference on File and Storage Technologies, December 13-16, 2005, San Francisco, California, Usa, 2005, 143–154.
- [22] Diesburg S, Meyers C, Stanovich M, et al. TrueErase: Per-file secure deletion for the storage data path. in: Proceedings of Computer Security Applications Conference, 2012, 439-448.
- [23] 傅颖勋, 罗圣美, 舒继武. 安全云存储系统与关键技术综述. 计算机研究与发展, 2013, 50(1):136–145.
- [24] Reardon J, Basin D, Capkun S. SoK: Secure Data Deletion. in: Proceedings of Security and Privacy, 2013, 301-315.
- [25] 李晖, 孙文海, 李凤华, et al. 公共云存储服务数据安全及隐私保护技术综述. 计算机研究与发展, 2014, 51(7):223–223.
- [26] Plank J S. A tutorial on Reed–Solomon coding for fault-tolerance in RAID-like systems. Software Practice & Experience, 1996, 27(9):995–1012.
- [27] Plank J S. Erasure codes for storage systems : a brief primer. ;login:: the magazine of USENIX SAGE, 2013, 38:págs. 44–50.
- [28] Li M, Qin C, Lee P P C, et al. Convergent dispersal: toward storage-efficient security in a cloud-of-clouds. in: Proceedings of Usenix Conference on Hot Topics in Storage and File Systems, 2014, 1-1.
- [29] Lee J, Heo J, Cho Y, et al. Secure deletion for NAND flash file system. in: Proceedings of ACM Symposium on Applied Computing, 2008, 1710-1714.
- [30] Swanson S, Wei M. Safe: fast, verifiable sanitization for SSDs. Knowledge-Based Systems, 2010, 75(C):224–238.