# 可信固态硬盘：大数据安全的新基础

田洪亮, 张勇, 许信辉, 李超 , 邢春晓

清华大学计算机科学与技术系，清华大学信息技术研究院，清华信息科学与技术国家实验室 北京 中国 100084

**摘　要**　大数据平台，因其数据多、价值高和存储集中的特点，已经成为对攻击者非常有吸引力的目标。因此，大数据安全是一个非常重要的研究课题。然而，当前两种保障大数据平台（如 Hadoop）数据安全的常见方法各有不足：一种是访问控制，通常由系统软件实现，存在被外部黑客攻破或内部管理员绕过的风险；另一种是数据加密，虽然密码学方法的安全性较高，但加密解密海量数据，增加了额外开销。综上所述，现有方法难以在保护海量数据时既提供较高的安全保证，又只带来可忽略不计的额外开销。

本论文提出可信固态硬盘（TrustedSSD），它提供安全增强的存储设备接口和协议，使得用户可以对存储中的数据施以细粒度的访问控制，保障存储中数据的安全性。访问控制层是可信固态硬盘的安全引擎，与通常固态硬盘的存储引擎闪存转换层紧密结合，实现高效的用户认证和操作授权。结合用户库函数和操作系统支持，可信固态硬盘可以为数据密集型应用提供安全的数据存储服务。我们在商用成功的固态硬盘控制器上实现了原型系统。实验结果表明，无论是在合成的、还是真实的工作负载上，可信固态硬盘的运行开销只有不到 3%。因此，我们认为可信固态硬盘有望成为大数据安全的新基础。

关键词　大数据；安全；数据机密性；固态硬盘

**中图法分类号**　TP319

## TrustedSSD：New Foundation for Big Data Security

TIAN Hongliang,　ZHANG Yong,　XU Xinhui,　LI Chao,　XING Chunxiao

Research Institute of Information Technology, Tsinghua National Laboratory for Information Science and Technology,

Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China

**Abstract**　Big data, known for its characteristics of high volume, high value and centralized storage, is an attractive target for attackers. Thus, big data security is an important issue. However, access control and data encryption, which are the two most common data security measures used in big data platforms (e.g. Hadoop), are not satisfactory. Access control mechanisms, which are usually enforced by system software, are prone to bugs and vulnerabilities. Data encryption is provably secure, but incurs considerable overheads during data processing.

In this paper, we present TrustedSSD, a secure Solid State Drive (SSD) that enforces a fine-grained access control to data at rest, providing practically the same confidentiality guarantee as the encryption-at-rest approach.

———————————————

本课题得到国家973计划（编号：2011CB302302），新闻出版重大科技工程项目数字版权保护技术研发工程多硬件环境版权保护技术研发（编号GXTC-CZ-1015004/02），国家支撑计划（编号：2012AA09A408）资助。田洪亮，男，1987年生，博士研究生，主要研究领域为云数据安全，基于新硬件的数据处理，E-mail: tatetian@gmail.com. 张勇，男，1973年生，博士，副研究员，主要研究领域为数据管理和数据分析，E-mail: zhangyong05@tsinghua.edu.cn. 许信辉，男，1990年生，硕士研究生，主要研究领域为数据管理，E-mail: xuxh13@mails.tsinghua.edu.cn. 李超，女，1978年生，博士，副教授，主要研究兴趣包括存储系统和数据管理，E-mail：li-chao@tsinghua.edu.cn。邢春晓，男，1967年生，博士，研究员，博士生导师，主要研究领域有数据库和数据仓库，海量数字媒体管理，网络存储，数字图书馆、档案馆研究等， E-mail: xingcx@tsinghua.edu.cn。

We built a prototype of TrustedSSD on a commercially successful SSD controller. Experimental results on both synthetic and real-world workloads showed that TrustedSSD incurs less than 3% overhead. We believe TrustedSSD is a promising approach to big data security.

**Key words** big data; security; data confidentiality; solid-state drive (SSD)

# 1 引言

我们正处于大数据时代。企业和组织都在收集、存储和管理快速增长的信息，并试图最终把这些海量数据转换成新的知识和见解。为了发掘大数据中蕴藏的价值，企业内各种不同类型的数据，包括个人、财务、医疗等敏感信息，正被整合到集中化的存储系统中。大数据平台集中地存储了大量敏感的信息，无疑将成为攻击者的理想目标。因此，保护大数据安全是一个重要的研究课题。

在典型的大数据平台（比如 Apache Hadoop）中，海量数据是存储在廉价服务器集群中各个节点的本地硬盘中的。为了保护存储介质中的数据，防止敏感数据泄露，目前主要使用两种手段：访问控制和数据加密。但这两种方法都各有不足。访问控制，通常由系统软件实施，其安全性依赖于代码的安全性和健壮性。随着系统越来越庞大复杂，很难保证不存在软件错误或安全缺陷，即使像 Google 这样技术先进的公司也不能幸免[1]。除了软件可能存在问题外，企业内部的管理员也可能利用特权绕开访问控制，窃取隐私数据，这从瑞士银行员工泄露用户资料事件可见一斑[2]。为了弥补系统软件访问控制的不足，一个常见做法是数据加密。基于密码学的方法虽然具有更高的安全性，但对海量数据加密解密不可避免地带来了显著的额外开销。以 Hadoop 为例，其在设计安全机制时的一大考虑因素是对性能的影响（不超过 7%）[3]。到目前为止最新的 2.4 版本，Hadoop 仍然没有提供默认的加密解密框架，部分原因是顾忌加密解密的开销。综上所述，现有方法难以在保护海量数据时既提供较高的安全保证，又只产生可忽略不计的额外开销。

通常来讲，服务器的本地存储介质（通常是硬盘）可以被主机任意读写，因此是不可信的，需要系统软件的访问控制或数据加密对其保护；但我们认为，硬盘是应该且可以增加安全机制的。近些年，固态硬盘，因其延迟低、吞吐量大、能耗低、噪音小，在各种应用场景正逐渐取代传统的机械硬盘，包括企业级服务器市场[5]。而且，最近的一些针对固态硬盘的存储内计算（In-Storage Processing）的一系列研究[7, 8, 9,10]表明，固态硬盘内有充足的资源（如 CPU 和内存等）支持额外的功能。

基于以上观察，本文提出一种解决大数据平台数据存储安全的新思路——存储内安全（In-Storage Security）——把对数据的访问控制从主机上的系统软件下放到底层存储内部。更具体地说，我们设计并开发了可信固态硬盘（TrustedSSD）。它提供安全增强的存储设备接口和协议，使得用户可以对存储中的数据施以细粒度的访问控制，保护存储中数据的安全性。在固态硬盘的固件中，我们增加了访问控制层（Access Control Layer, ACL），它是可信固态硬盘的安全引擎，与普通固态硬盘具有存储引擎闪存转换层（Flash Translation Layer，FTL）紧密结合，实现高效的用户认证和操作授权。结合用户库函数和操作系统支持，可信固态硬盘可以为数据密集型应用（比如 Apache Hadoop）提供安全的数据存储服务。

本研究的主要贡献有：

1. 我们为可信固态硬盘设计了安全增强的存储设备接口和协议，详细分析和论证了其安全性，展望了其在大数据平台中的应用前景（见第 3 节“存储内安全”）；

2. 我们在可信固态硬盘的设计与实现过程中解决了数项技术难点，包括提高硬件平台的基准性能，设计高效的访问控制机制，以及修改复杂的操作系统 I/O 栈（见第 4 节“设计与实现”）；

3. 我们在商用成功的固态硬盘控制器上实现了可信固态硬盘的原型系统。在合成的和实际的工作负载上测试了可信固态硬盘的性能，实验结果表明可信固态硬盘的安全机制带来的运行开销小于 3%（见第 5 节“实验结果”）。

传统的观点认为，存储是不可信的，大数据的安全保障应该由系统软件负责。但我们在可信固态硬盘上的工作表明，存储不仅可以信任，而且可以有效地保障数据安全，成为大数据安全的新基础。

## 2 相关工作

此前也有一些在存储层面的安全解决方案的工作，但它们主要是针对传统硬盘的，而且在目的、手段和适用范围上都与本研究不同。自安全存储（Self-Securing Storage）[22] 在一段时间内保留数据的历史版本，以应对入侵者在未发觉的情况下篡改或删除数据。网络安全硬盘（Network-Attached Secure Disks）[23] 通过把访问控制下放到各个硬盘，从而消除了在传统网络文件系统中访问控制完全集中在文件服务器这一系统瓶颈。可存活存储系统（Survivable Storage System）[24] 把一个文件的每个片段打散分布到多个存储节点，这样即使单个存储节点被入侵，也可以保证数据的隐私性、可靠性和可用性。

最近几年，有不少研究工作探索如何为固态硬盘增加额外的功能。文献[7]在固态硬盘内增加硬件逻辑以更高效地处理数据扫描操作。文献[8]研究把关系型数据库的部分运算下推到固态硬盘内部完成。文献[9]扩展了 Hadoop 了，使得部分 Map 任务可以在固态硬盘内部完成。文献[10]提出了事务型闪存转换层（Transaction FTL），把 SQLite 保证事务原子性（Transaction Atomicity）的功能放到了固态硬盘中。与上述针对固态盘的存储内处理（In-Storage Processing）的研究工作不同的是，可信固态盘是一种存储内安全（In-Storage Security）的尝试。

据我们所知，本研究是目前唯一探索为固态硬盘添加的安全和隐私保护机制的研究工作。

## 3 存储内安全

存储内安全（In-storage Security）是指一种在存储设备内部实现安全机制的做法。安全界的一项共识是，没有一种安全措施能够防范所有安全威胁，多层次的安全机制才能最大限度地保证系统安全。传统上，普通存储介质（如硬盘）被认为是不可信任的；而我们认为，存储介质的不可信正是大数据安全的缺失环节。存储内安全可以有效地增强存储中数据的安全性。

大数据系统的安全性，最重要的就是存储数据的机密性。数据安全的需求，从数据状态来看，可分为：存储中数据、使用中数据（即内存中数据）和传输中数据等；从威胁角度来看，又可分为：机密性、完整性和可用性等。一方面，正在传输中或使用中的数据，相比存储中数据，毕竟是小部分；另一方面，攻击者最常见的动机是窃取有价值的、敏感的数据，最难发觉的行为是窃取数据，而非删改数据。因此，我们认为在众多数据安全需求中，保障存储数据的机密性是最迫切的。

本文主要关注于利用可信固态硬盘保护存储数据的机密性。本节中，我们首先介绍可信固态硬盘应对的安全威胁，然后讨论如何设计可信固态硬盘的接口和协议以有效地保证数据机密性，最后我们以 Hadoop 为例说明可信固态硬盘在大数据平台中的应用。

### 3.1 安全威胁

我们假设攻击者：

1．可以获得了主机的特权。他们既可能是恶意的外部入侵者，也可能是好奇的内部管理员。他们可能在服务器上运行恶意程序直接读取数据文件，也可能修改操作系统或数据库管理系统的权限。无论手段如何，结果是他们可以成功得绕开所有系统软件对存储中数据的访问控制。

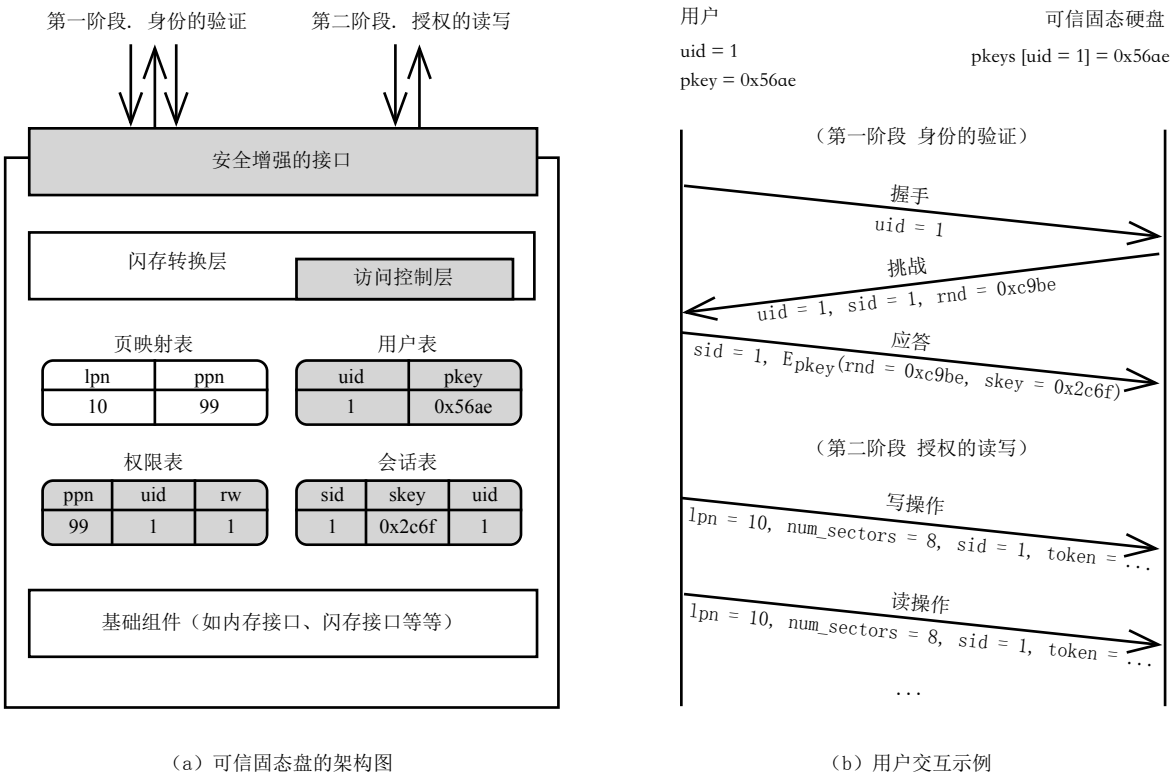
2. 企图获取存储中的敏感数据，但对篡改或销毁数据并不感兴趣。我们认为这个假设是比较符合实际的，因为读取数据很难被检测到，而篡改或删除数据相对容易被发现，作案风险大、利益小。

3. 通过网络远程访问目标主机，而不能物理接触。对于管理良好的数据中心，这是一个合理的假定。可信固态硬盘主要是用作集群中数据节点上的直连存储设备，通过常见的存储接口（SCSI、SATA 和 SAS 等）与主机连接。这个定义良好的存储接口是存储与主机交互的唯一途径，也是对存储设备的唯一攻击面（Attack surface）。

4. 攻击者无法篡改可信固态硬盘的固件程序（Firmware），因为固件程序是受到保护的。一个比较理想的保护方式是，可信固态硬盘的新固件必须在能被验证是合法的（比如有生产厂家数字签名）之后才能替换原来的固件。另一个可能的方式是，由可信固态硬盘上的一个物理开关决定固件是否处于可更新的状态。我们的原型系统采用了第二种方式。由于攻击者不能物理接触可信固态硬盘，也就无法篡改固件。

### 3.2 安全协议

本小节讨论如何设计一个安全增强的存储接口和协议，使得在不可信的主机上进行安全的数据




（a）可信固态盘的架构图　　（b）用户交互示例

**图 1 可信固态硬盘的安全机制。图 1a 中方角和圆角方框分别表示模块和元数据；灰色的方框表示与安全相关；圆角方框中的数值表示图 1b 所示的交互完成时的状态。图 1a 中的 lpn、ppn、uid、pkey、rw、sid 和 skey 分别表示逻辑页地址、物理页地址、用户 ID、用户密钥、页是否可写、会话 ID 和会话密钥。图 1b 中（不在图 1a 中）的 lpn、pkeys、num_sectors、rnd、$E_{pkey}$ 和 token 分别表示起始逻辑页地址、用户表、读/写扇区数、随机数、加密算法和操作验证令牌。**

访问成为可能。可信固态硬盘的接口是主机与它交互的唯一途径；主机必须按照这个接口的协议、通过发送合法命令，才能读取或改变存储内部的状态。我们希望扩展普通的存储接口，使其具备用户验证、操作授权等功能，成为一个安全增强的存储接口。

然而，设计安全增强的存储接口和协议是有多重挑战的。首先，攻击者可以监听主机和存储之间的所有交互；重放攻击（Replay attack）、伪装攻击（Impersonation attack）和中间人攻击（Man-in-the-middle attack）等各种攻击都是可能的。其次，存储器每秒钟处理的 I/O 操作数可以高达 10,000，而其内部的计算资源有限，因此协议对每个 I/O 操作的验证授权开销必须很小。最后，安全增强的存储协议必须与原有的软硬件接口尽量兼容，使得驱动程序的改动尽可能的小，并且不需要任何硬件上的改动。

我们最终设计出来的可信固态硬盘的安全增强的接口和协议的概况见图 1。图 1a 中展示了可信固态硬盘的架构图，其中“增强的安全接口”可以接收扩展的命令，“访问控制层”是安全机制的控制逻辑，“用户表”、“权限表”和“会话表”是与安全相关的数据结构。用户与可信固态硬盘的安全交互由“身份的验证”和“授权的读写”两个阶段组成，图 1b 做了举例说明。

身份验证是第一个阶段，其目的是认证用户身份，并建立会话。每个用户都有一个 ID（uid）和密钥（pkey），在用户向可信固态硬盘注册账号的时候完成分配；用户 ID 是公开的，但用户密钥只有用户和可信固态硬盘知道。用户的身份验证采用密码学的经典方法——挑战-应答协议（如图 1b）。挑战-应答协议，简单地说，就是通过让用户使用他的密钥加密一个随机数来证明它拥有这个密钥的事实，从而让服务方可以认证用户的身份。完成上述身份验证之后，用户和可信固态硬盘之间建立起了一个会话，用户和可信固态硬盘（见图 1a 中的会话表）都拥有这个会话的密钥。凭借这个会话密钥，用户就可以进行授权的读写操作。

授权读写是第二个阶段，用户凭借会话密钥进行读写操作，可信固态硬盘只授权合法的读写操作。普通的块设备的读写操作，逻辑上可以简化地看成调用 read(lpn, num_sectors) 和 write(lpn, num_sectors, data)两个函数，其中 lpn 是逻辑页地址（Logical Page Number），num_sectors 是读写的扇区数目，一个逻辑页包含数个扇区（比如 8 个），一个扇区的大小是 512 字节。为了验证读写操作的合法性，我们为 read 和 write 扩充了两个可选参数（如图 1b），会话 ID（sid）和操作令牌（token），前者是公开的，后者是根据会话密钥生成（生成规则见下一小节“安全分析”）。可信固态硬盘在接收到读写命令后，根据会话 ID 和操作令牌检查操作是否属于一个合法的会话，如果不合法，则拒绝执行；如果合法，则用会话表（见图 1a）查出发出这个操作的用户 ID。对于合法的写操作，页权限表中把涉及到的页标记成属于这个用户；对于合法的读操作，则检查涉及到的页是否全都属于这个用户，如果是，则继续执行这个读操作；否则，拒绝执行。以上是可信固态硬盘的细粒度访问控制的基本原理，更多实现细节见 4.2 小节“访问控制层”。

### 3.3 安全分析

采用基于会话的安全机制是出于两方面的考虑。一方面，大数据应用的数据处理模式往往是把计算任务发送到数据源（在 Apache Hadoop 中是 Map/Reduce 任务发到数据节点）。计算任务为了合法地读取可信固态硬盘中的数据必然需要某种凭证（在不可信主机上），如果凭证是用户密钥，则一旦丢失，危害很大；如果凭证是会话密钥，即使被窃取，攻击者也仅能在相对短暂的会话期间内冒充合法用户。另一方面，密钥越长安全性越好，但各种密码学操作的开销越大。把验证身份的密钥（用户密钥，使用频率低）和授权读写操作的密钥（会话密钥，使用频率高）区分开，前者使用长位数（比如 128 个比特），后者使用短位数（比如 32 个比特），可以在保证安全性的前提下，把开销降到最低。为了防止短位数的会话密钥被暴力破解，可信固态硬盘一旦发现读写操作的会话密钥不合法，则终止会话。

会话建立是在身份验证阶段。这里采用了挑战-应答协议，一种经典的密码学身份验证方法，其安全性得到了广泛的研究。挑战-应答协议保证了身份验证的安全性，因为 1）用户密钥没有被明文传输，不会被泄露；2）每次用户认证的应答由随机数决定，不受重放攻击影响；3）会话密钥是加密的，并只有用户可以解密。这样，身份验证阶段结束时，只有用户和可信固态硬盘了解会话密钥，保证了读写操作可以在会话中安全地进行。

会话建立之后就进入授权读写阶段。读写操作的验证，是以其附带的操作令牌（token）为依据的。一个最直接的设计是把会话密钥作为令牌，只要查找会话表（图 1a）就可以验证操作附带的会话密钥是否正确，从而完成操作的验证。但这样并不安全，因为任何可以监听用户进程与可信固态硬盘交互的进程都可以窃取会话密钥，从而利用会话密钥冒充合法用户。我们最终的设计是令操作令牌

$$token = MAC_{skey}(lpn, num\_sectors, sid)$$

即操作令牌是用户操作命令的消息认证码（Message Authentication Code, MAC）。这样的有三个好处，1）由于操作令牌的计算需要会话密钥（skey），因此只有合法用户的操作才有合法的令牌；2）根据消息认证码的性质，攻击者是无法通过令牌的值推测出会话密钥的值的；3）攻击者无法篡改用户发出的合法命令。

从上面的分析可以看出，可信固态硬盘的安全协议保证只有合法用户才能被授权执行的合法操作，并且在这个过程中不泄露任何密钥信息，即使攻击者（如拥有特权的外部黑客者或内部管理员）可以监听所有操作。

即使设计再周全，如果实现有漏洞，那么最终系统也是不安全的。虽然可信固态硬盘中的固件程序也具有相当的复杂性（约 1 万行 C 代码），但相比操作系统（Linux 的源代码有约 1500 万行[15]）和关系型数据管理系统库（MySQL 的源代码有约 150 万行[16]），可信固态硬盘中的固件程序出现软件错误或安全漏洞的可能性大大低于这些大型系统软件。较小的代码量使得深入的代码检查，或形式验证（Formal Verification）更可行[4]。除了担忧固件程序的正确性外，用户可能还对信任可信固态硬盘的厂商有疑虑。我们认为，这方面可以效法密码协处理器的做法，由可信权威的第三方机构评估产品的安全性，并对合格的产品授予证书[13, 14]，打消或缓解用户的疑虑。

综上所述，我们认为可信固态硬盘为数据提供安全、有效和可靠的隐私性保护，有望成为大数据安全的新基础。

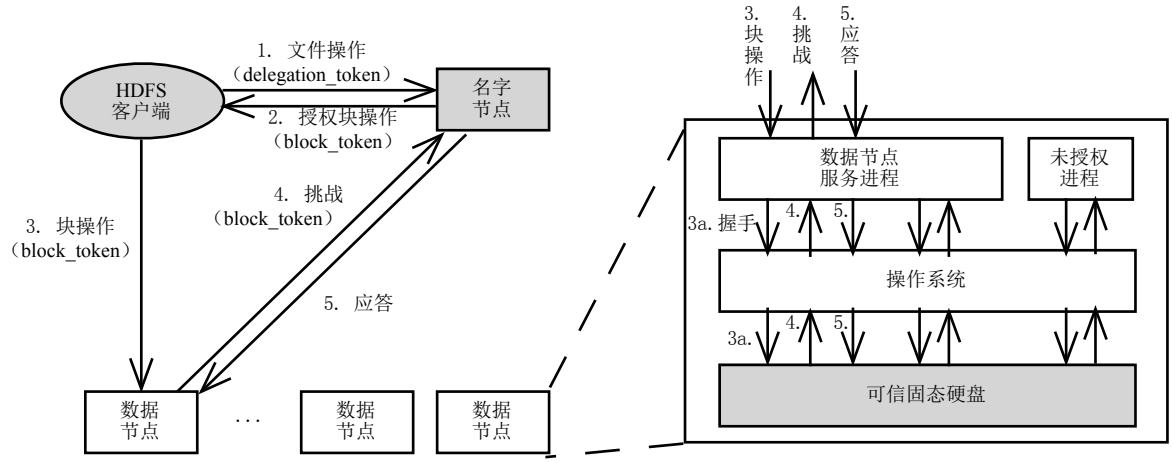



图 2 基于可信固态硬盘的安全增强型 Hadoop

### 3.4 应用举例

本小节以 HDFS（Hadoop Distributed File System）为例子简要介绍可信固态硬盘如何提高大数据系统的数据安全性。

HDFS 的现有安全机制需要名字节点和数据节点协作保证(如图2左边)。名字节点首先验证HDFS客户端发来的文件操作（比如读操作）是否来自一个权限足够的用户。核实用户身份的凭证是其提供的代理令牌（Delegation token)。如果用户合法、权限足够，则把操作所要块的块令牌（Block token）返回给客户端，这个块令牌是加密过的，无法伪造，且过期即失效。随后客户端向数据节点调用块操作时，会把块令牌也发送给数据节点，后者检查块令牌的合法性。如果合法，才授权客户端的块操作。以上就是 HDFS 访问控制的基本原理。

但上面的访问控制是有不足的。HDFS 块是在数据节点上以文件的方式存放的。这意味数据节点的内部管理员或获得特权的外部攻击者，可以绕开上述的 HDFS 访问控制机制，直接读取块所在的文件。可信固态硬盘可以防止这种情况发生。用户的可信固态硬盘秘钥可以存储在名字节点（认为是可信的)，当他需要读取受可信固态硬盘保护的 HDFS 块时，按照 3.2 和 3.3 小节所描述的协议完成身份验证和操作授权即可（如图 2 右边)；即使是拥有特权的攻击者，没有用户的可信固态硬盘秘钥，也无法从可信固态硬盘中非法读取数据。

## 4 设计与实现

可信固态硬盘的原型系统包括固态硬盘的固件程序，操作系统的修改，和用户库函数三个部分。在设计和实现原型系统的过程中，我们主要解决了三个挑战：

1. 提高固态硬盘的基准性能：我们的固态硬盘是基于 OpenSSD Jasmine 平台[5]开发，而后者已有的开源固件实现的是普通的页映射闪存转换层，存在写放大（Write amplification）、并行不充分等问题，因而性能不佳。

2. 设计高效的访问控制机制：块设备最小的读写单元是扇区（512 个字节），因此访问控制也因以扇区作为最小单位，即应该为每个扇区都记录权限信息。然而，OpenSSD 的硬件没有除纠错码（Error correction code）外的带外区域（Out-of-band area）可用于记录权限信息。最直接的解决方法是用一个权限表，但表的大小很大，会导致很多缓存缺失，严重影响性能。

3. 修改复杂的操作系统 I/O 栈：Linux 操作系统的 I/O 软件栈，包括系统调用、虚拟文件系统、缓冲区、通用块设备层和 SCSI 驱动程序层（其中又包括高层、中层和底层驱动）等等，是一个包含诸多组件的多层架构的。这种结构虽然具有模块化、松耦合和灵活性等优点，但用户传递参数（如会话 ID 和操作令牌）到固态硬盘需要经过所有这些层次。尽量少改动就到达到目的，是很有挑战的。

我们解决了以上三个挑战，并分别在接下来的“闪存转换层”，“访问控制层”和“操作系统 I/O 栈”等三个小节阐述。在此之前，有必要对 OpenSSD Jasmine 固态硬盘开发平台做一个介绍。

OpenSSD Jasmine 开发平台[5]是一个开放的固态硬盘参考实现，基于 Indilinx 公司的 Barefoot$^{TM}$ 控制器，后者已经成功应用于众多厂商的固态硬盘

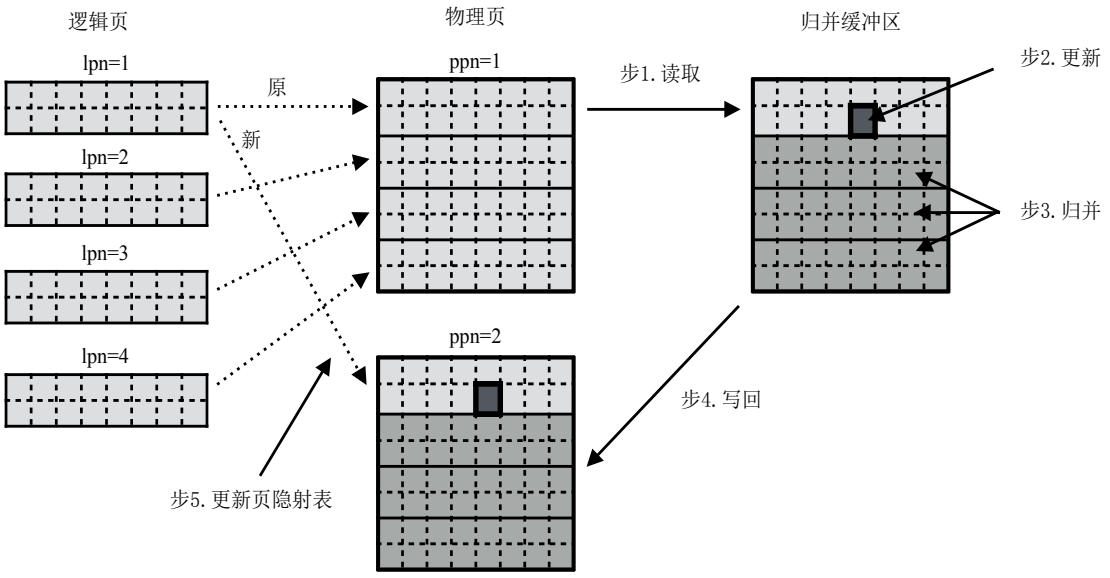



图 3 子页级隐射和归并缓冲区。此图展示了在使用子页级映射和加入归并缓冲区之后闪存转换层更新一个扇区的流程。图中虚线表示逻辑页到物理页的映射关系，实线表示流程中的每个步骤，小虚线格子表示扇区，实粗线格子表示要更新的扇区。lpn 表示逻辑子页序号，ppn 表示物理页序号。浅灰色格子表示更新前的数据；中灰色格子表示归并的数据；深灰色的格子表示更新的数据。更新前，逻辑子页 1 存储于物理页 1；更新后，存储于物理页 2。为了简化图形表示，在本图中标出的子页大小是 8KB，而在我们的实现中是 4KB。

产品。OpenSSD Jasmine 开发板，包括：基于 ARM7 处理器的闪存控制器；96KB 的 SRAM，作为程序的内存空间，存放程序和数据；64MB 的 DRAM，主要作为 I/O 缓冲区；8 个 MLC NAND 闪存模块插口，达到至多 256GB 闪存容量。除了硬件以外，OpenSSD Jasmine 开发平台提供开源的固件程序，后者主要包括固件更新模块、硬件（如内存、闪存、事件队列等）控制模块、SATA 模块，以及最核心的闪存转换层。

### 4.1 闪存转换层（FTL）

闪存转换层（Flash Translation Layer, 后简称 FTL）是基于闪存的固态硬盘的核心引擎。闪存转换层的主要目的，是使基于 NAND 闪存的固态硬盘从使用接口看来与基于机械转盘的传统硬盘无异。FTL 的上层是适用于传统硬盘的块设备接口（读、写操作），下层是 NAND 闪存模块的接口（读、写和擦除操作）。这种转换的必要性是源自 NAND 闪存的物理特性导致的读写不对称：读写操作均以页为单位（在 OpenSSD 中是 32KB），但写操作只能对空白页，非空白页必须擦除后才能写，而擦除比读写操作的时间高得多，且是以块为单位的（在 OpenSSD 中是 4MB，一个块包含 128 个页）。为了缓解高耗时的擦除操作对性能的不利影响，固态硬盘放弃了传统硬盘的原地更新方式，而是异地更新（Out-of-place update），即首先在一个空白页写新数据，然后把旧数据所在的页标记为过时，最后更新页映射表以反映逻辑页与物理页之间对应关系的变化，至此完成更新一个逻辑页的操作。被标记为过时的页要定期清理和擦除，也就是垃圾回收。以上异地更新、垃圾回收等都是 FTL 的工作。

由于 FTL 对固态硬盘的性能至关重要，已经有许多关于 FTL 的研究工作。这些工作根据隐射表的隐射颗粒度可以分为页级 FTL[18,21]、块级 FTL[20] 和混合型 FTL[16,17]。目前学术界普遍的看法是这三种类型的 FTL 中页级映射是最高效的，而且 OpenSSD 中也有开源的页级 FTL 实现，因此接下来主要讨论页级 FTL。顾名思义，页级 FTL 允许逻辑页地址（Logical Page Number, LPN）可以直接转换成物理页地址（Physical Page Number, PPN）。为此，页级 FTL 需要维护一张页级隐射表，为每个逻辑页都需要保存一条记录；整张表都要持久地保存在闪存中，正在使用的部分载入到 DRAM 中缓存起来，以加速读取。这张表的大小往往超出固态硬盘上的 DRAM 容量，但由于实际负载一般都具有较好的局部性，缓存的命中率一般都很高[18]。

OpenSSD 开源固件程序实现了页级 FTL，但性

能不佳。我们 OpenSSD 开发板装有 128GB 闪存，写操作最高能达到 180MB/s 的吞吐量，读能达到 200MB/s。然而，OpenSSD 自带的 FTL 实现只能在少数负载情况（比如以 32KB 为单位的顺序读写操作）发挥硬件的全部潜力。对常见的 4KB 随机读写负载，OpenSSD 的性能降到 10MB/s 左右。如果以此为基准，评估在固态硬盘中加入安全机制对性能的影响，结果难有足够的说服力。为此，我们在把安全机制整合进固件之前，不得不先提高原固件的性能。

4KB 随机写的性能较差主要是因为写放大（Write amplification）。我们考虑比较极端的情况，页级 FTL 如何更新一个扇区。OpenSSD 的物理页是 32KB 大小，根据异地更新的原理，FTL 必须首先把这个扇区原来所在的物理页读出到内存中的一个页缓冲区，然后更新这个缓冲区相应扇区，最后把这个页缓冲区写回到闪存，并更新映射表。根据上述流程，写一个扇区的开销是一次页读加一次页写，即为了写一个 512B 不得不写一个 32KB，这就是“写扩大”。

为了解决写放大的问题，我们引入了子页级隐射和归并缓冲区两个技术。写扩大的根本原因是因为 OpenSSD 物理页的尺寸较大，导致小的写操作效率低。子页级映射，把一个 32KB 的页分为 8 个 4KB 的区域，后者被称为子页（Sub-page）。映射表不再把逻辑页对应到物理页，而是把逻辑子页（Logical Sub-page ）对应到物理页。逻辑子页允许存储在任何一个物理页，但其在物理页中的位置必须与其在逻辑页中的位置一致，即某逻辑页的第 i 个子页也必须存为某个物理页的第 i 个子页。这是为了避免在读取物理页的时候出现其中的子页顺序不对的情况。由于引入子页之后，更新数据的最小单位变为了子页，但毕竟最终写入闪存中还必须以页为单位，因此很自然地引入了归并缓冲区，负责把几个独立的子页更改合并为一个页来写入。同时，多个归并缓冲区一起也可以起到读写缓存的作用，如果逻辑页的数据在缓存中，则直接读取或更新缓存，无需闪存操作。引入子页级隐射和归并缓冲区之后，更新一个扇区并不需要立刻写入扇区，而是首先写入归并缓冲区（如图 3）。优化后的净效果是写 512B 的摊销开销是 1 个读操作，加 1/8 个写操作。

除了写放大外，我们发现 OpenSSD 原 FTL 实现的另一个问题没有充分利用硬件的并行性。固态硬盘的一大优点是其吞吐量可以随闪存模块数量的扩充而线性增加。这要得益于其内部丰富的硬件并行性。以 OpenSSD Jasmine 开发板为例，它拥有四个完全独立的通道（Channel），每个通道有最多个 8 个独立的库（Bank），每个库中包含两个 8 位的闪存芯片，它们一起组成一个 16 位的存储单元；每个通道都有一个 16 位的 I/O 总线和一组控制信号，通道内在任意时刻只能正在有一个接收闪存接口指令的库，但可以各自执行其已经接收的闪存命令（比如读、写、擦出）。因此，固件程序应该尽量做好调度工作，充分发挥这 32 个库的并行性。然而，OpenSSD 原有的 FTL 实现存在诸多因等待阻塞 I/O 完成，比如：1. 当更新子页的一部分时，必须先等子页的未更改部分从闪存载入（因为写回子页的时候要连同未修改的部分一起写回）；2. 当隐射表中某个记录缓存缺失时，必须先等需要的条目从闪存中载入到内存；3. 当希望向某个库发送闪存命令时，必须先等它完成正在执行的命令。

在这些阻塞的闪存 I/O 完成前，CPU 都处于忙等的状态，无法进行任何其他有效的工作（比如接收新命令或给库发送新的闪存命令），因而硬件的并行性也没有得到充分利用。

提高固件程序并行性的根本解决办法是多线程编程。然而，开发固件是在嵌入式编程环境，直接面向底层硬件，连操作系统都没有，更不要说线程这种操作系统中的概念。一个最直接的想法是引入为嵌入式开发而设计的实时操作系统（Real-Time Operating System，RTOS），后者可以提供轻量级的进程，以及高效的进程调度。但 OpenSSD 的 SRAM 只有 96KB，容纳现有的固件程序已然稍显不足，很难再支撑一个操作系统。引入现成的多线程支持不行，只好自己动手。我们自主实现了进程的上下文切换、进程调度、异步 I/O、进程间通讯等一系列与多线程编程有关的、原本由操作系统提供的编程原语。多线程和异步 I/O 机制引入后的一大挑战是保证乱序执行的用户请求的最终结果保持语义的正确性。为此，我们又引入了锁机制，并在开发时小心谨慎地使用互斥锁和共享锁，在语义正确的前提下最大化各种硬件资源的使用率。

实验表明我们对 FTL 的优化是行之有效的（最多达到 10X 的性能提升，见第 5 节）。以往对 FTL 的研究往往都侧重理论，实验结果以模拟为主，而缺乏与实际硬件情况的结合。我们在实际硬件平台上的开发经验表明，算法固然重要，实现技巧对最

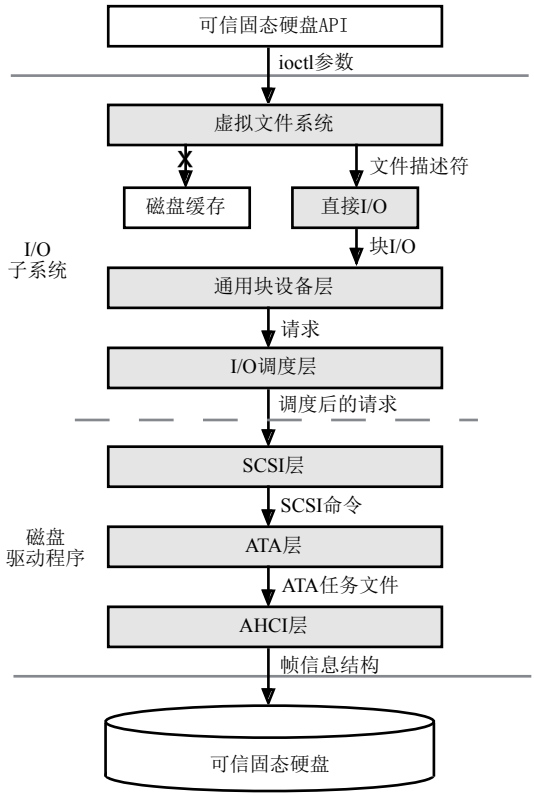



**图 4 修改的操作系统 I/O 栈。修改的操作系统 I/O 栈。本图展示了为向可信固态硬盘传递额外参数（如会话 ID 和操作令牌）而需要修改到操作系统 I/O 组件。图中灰色方框是修改的组件，箭头表示参数传递方向，箭头边是组件之间传递的数据结构的名称。可信固态硬盘的 API 保证 I/O 请求不经过磁盘缓冲，因此图中磁盘缓存是白色方框，表示不涉及这个组件。**

终性能也有很大的影响。

### 4.2 访问控制层（ACL）

访问控制层（Access Control Layer, 后简称 ACL）是可信固态硬盘的安全引擎，与 FTL 紧密结合，确保用户请求被安全地接收和执行。ACL 主要功能有两个，一是用户验证，主要涉及到的数据结构是用户表；二是操作授权，主要用到的数据结构是会话表和权限表（见图 1a）。用户验证的原理已经在 3.2 节中介绍，具体实现中涉及到加密和安全伪随机数生成等常见密码学操作都采用了广泛验证的做法，在此不赘述。固态硬盘的每秒输入输出数（IOPS）可能达到 10000 以上，操作授权伴随每一个用户读写请求。因此，高效的操作授权对达到可信固态硬盘的“高安全保证、低运行开销”的目标至关重要。本小节下面聚焦于如何设计和实现高效的操作授权。

操作授权首先要高效地验证操作密令。在 3.3 小节中，我们给出了一个安全的操作令牌设计，即使用操作的消息认证码（Message Authentication Code, 后简称 MAC）。常见的 MAC 实现依赖于块加密算法或安全哈希函数，计算开销较高。为了在保证安全性的前提下尽量降低开销，我们采用了 UMAC，它被认为是最快的 MAC 算法，软件实现的峰值速度可以到约 1 时钟周期/字节。

操作授权还要高效地查看或修改操作所涉及页面的权限（这里权限主要指数据的拥有者）。由于块设备的最小访问单位是扇区，原则上可信固态硬盘应该为每个扇区维护权限信息。闪存上的每个扇区通常会留一些额外的空间存储系统信息，这个额外空间被称为带外区域。受 OpenSSD 的硬件限制，带外区域仅能用于存放纠错码，没有额外的空间可供存放权限信息。所以，只能在闪存上专门分配一些页以保存权限信息，即权限表。对于 128GB 的固态硬盘，扇区数高达 256M，假设每个扇区的权限信息是 2 个字节，那么权限表的大小达到 512MB；相比之下，对于同等大小的固态硬盘，页映射表也才 128MB。OpenSSD 拥有 64MB 的 DRAM，其中只有不到 20MB 可以作为页映射表和权限表的缓存。如果采用 512MB 权限表的设计，无疑会导致页映射表和权限表的缓存命中率大幅降低，增加很多额外的闪存读写，造成显著的运行时开销。

因此，压缩权限表的大小很有必要的。我们使用了下面两项技术：4KiB 对齐的文件系统，和用户觉察的归并缓冲区。

观察实际应用的 I/O，我们发现磁盘请求的块大小常常是 4KiB 的整数倍。这使我们想到，如果所有磁盘请求都是 4KiB 对齐的（起始地址和请求大小都是 4KiB 的整数倍），那么就不用以扇区为单元记录权限信息，而是以 4KiB 的子页为单位，这样权限表的大小就降到原来的 1/8。而幸运的是，让（几乎）所有磁盘请求都为 4KiB 对齐的是容易办到的，只要 1）在为可信固态硬盘分区的时候指定分区都以 4KiB 对齐，比如常见的分区工具 fdisk 就支持这个功能；2）在为可信固态硬盘上的分区创建文件系统的时候指定文件系统的块大小是 4KiB（或 4KiB 的倍数），比如 Linux 的主流文件系统 Ext4 就可以设置块大小。这样，用户的应用程序经由文件系统向可信固态硬盘发出的磁盘请求就是以 4KiB 对齐的了。注意，ACL 利用 4KiB 对齐文件系统的特点减少权限表的大小，但 ACL 的安

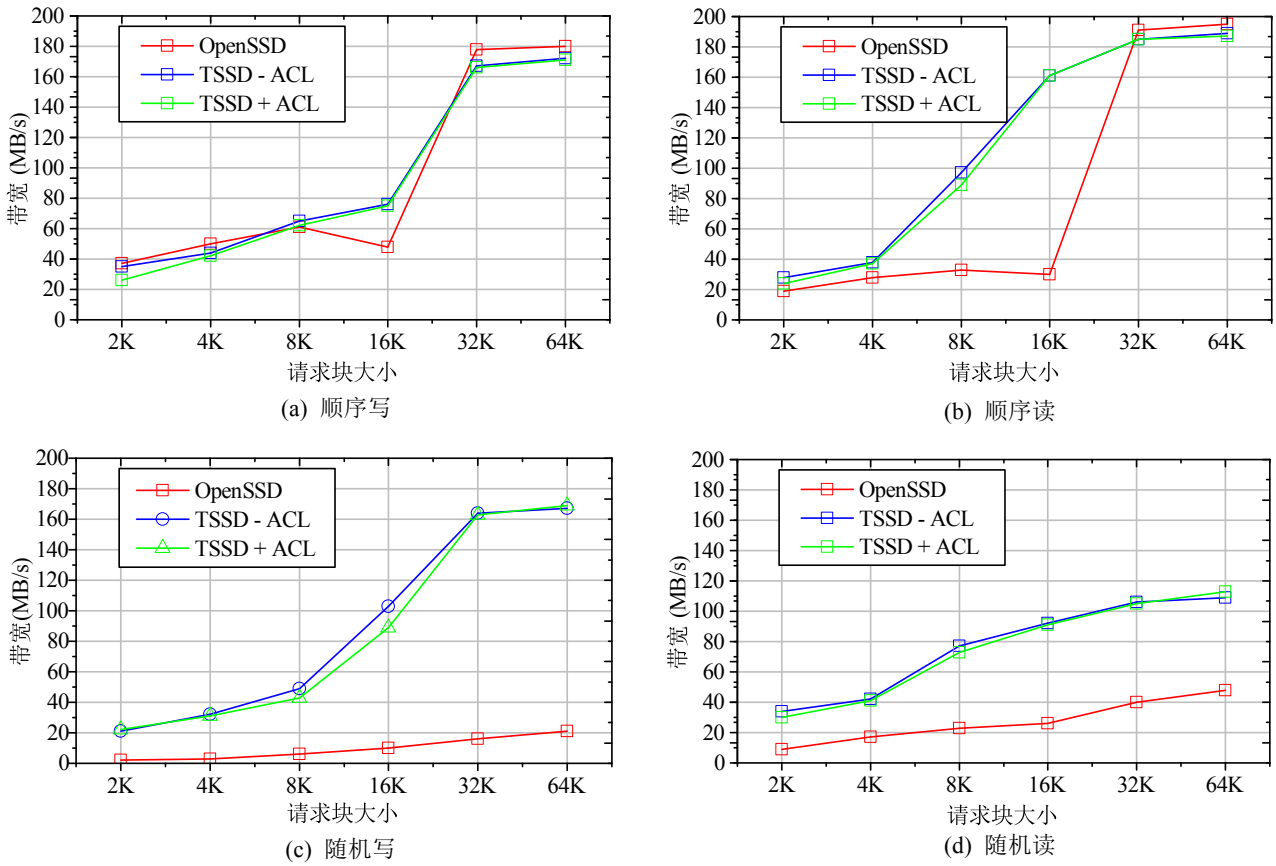



图 5 合成负载的测试结果。本图比较了三个固件：OpenSSD 表示原始的固件，TSSD － ACL 表示可信固态硬盘的固件但禁用访问控制层（ACL），TSSD + ACL 表示可信固态硬盘的固件（带访问控制）。

全性不依赖这点，即非 4KiB 对齐的磁盘请求也可以安全地处理。

为了进一步压缩权限表的大小，ACL 引入用户察觉的归并缓冲区。在 4.1 小节中，我们介绍了归并缓冲区如何解决写放大的问题，以及多个归并缓冲区组成读写缓存的好处。与之前的缓冲区不同的是，一个用户察觉的归并缓冲区只能归并属于同一个用户的子页。这个性质可以保证由用户察觉的归并缓冲区写回到的物理页中都是同一个用户的数据。既然如此，ACL 只需以物理页为单位记录权限信息。这样，最终得到的权限表是以物理页为单位的，其大小是 8MB，只有最初估计的 1/64，完全可以全部缓存在 DRAM 中。

综上所述，可信固态硬盘的访问控制机制经过精心设计后是可以高效实现的。

### 4.3 操作系统I/O栈

可信固态硬盘完全与普通硬盘兼容；这意味着用户程序、操作系统和驱动程序等无需任何改动，就把可信固态硬盘作为普通硬盘正常使用。但如果用户希望读取可信固态硬盘中受保护的数据，或向可信固态硬盘写入受保护的数据，则需要有操作系统的支持。需要指出的是，虽然可信固态硬盘的功能完整性依赖于操作系统，但其安全性是独立于主机软硬件的。

为了读取文件中受可信固态硬盘保护的数据，或向文件中写入受保护的数据，除了操作系统的支持外，用户程序还要使用可信固态硬盘提供的 API，其中最重要的两个函数（C 语言）是：

*int tssd_open(const char* path, int flags);*

*int tssd_use_session_token(int fd, tssd_token_t token);*

tssd_open 是可信固态硬盘提供的文件打开函数，其特别之处在于它强制以直接 I/O（Direct I/O）模式打开文件。直接 I/O 模式常用于自主管理磁盘缓存、无需操作系统缓存的数据密集型程序，如关系型数据库管理系统。操作系统高速磁盘缓存使得其中缓存的页可以被多个不同的进程共享复用，省去很多磁盘 I/O。但这个高速磁盘缓存的优点对于访问控制却是安全隐患，因为非可信固态硬盘授权的用户有可能从操作系统的磁盘缓存读取到属于

表格 1 真实负载的测试结果。本表展示了在多种不同特点的真实负载下，可信固态硬盘的访问控制层对性能的影响。具体来说，它比较了无访问控制层（TSSD - ACL），平凡的访问控制实现（TSSD + 平凡的 ACL）和优化的访问控制层制（TSSD + 优化的 ACL）。响应时间是操作系统向可信固态硬盘发送 I/O 请求的平均响应时间，额外开销是以无访问控制的可信固态硬盘为基准。

| 负载名称 | 负载特征 | | TSSD – ACL(性能基准) | | TSSD + 平凡的 ACL | | TSSD + 优化的 ACL | |
|---|---|---|---|---|---|---|---|---|
| | 读比例 | 平均块大小 | 响应时间 | 额外开销 | 响应时间 | 额外开销 | 响应时间 | 额外开销 |
| Exchange | 6.80% | 24KB | 365 us | N/A | 409 us | 12% | 375 us | **2.7%** |
| BuildServer | 8.36% | 37KB | 451 us | N/A | 640 us | 42% | 461 us | **2.2%** |
| Financial | 15.40% | 3KB | 179 us | N/A | 326 us | 82% | 184 us | **2.8%** |
| TPC-C | 58.44% | 9KB | 243 us | N/A | 411 us | 69% | 246 us | **1.2%** |
| WebSearch | 99.99% | 15KB | 332 us | N/A | 485 us | 46% | 335 us | **0.9%** |

其他用户的数据。以直接 I/O 模式，跳过操作系统磁盘缓存，可以避免上述问题。

tssd_use_session_token 是用于向可信固态硬盘传递本次读写请求的操作令牌的函数。操作令牌是可信固态硬盘授权每个读写请求的凭据。Linux 操作系统的 I/O 软件栈，包括虚拟文件系统、磁盘缓冲、通用块设备层、SCSI 驱动程序（其中又包括高层、中层和底层驱动）等等，是一个包含诸多组件的多层架构。为了把操作令牌传递到最底层的可信固态硬盘，我们修改了 Linux 操作系统 I/O 子系统和 SCSI 子系统中各个有关组件，并扩充了它们之间通信的数据结构（如图 4）。

软件的数据结构容易扩展，硬件的物理接口就难以改动了。AHCI 层向 SATA 接口的固态硬盘发送各种 SATA 命令的数据格式是依据一种叫帧信息结构（Frame Information Structure）的 ATA 标准，它的字段和大小都是业界标准，并且与硬件接口的寄存器相匹配，无法增加额外的字段。幸运的是，在仔细研究 ATA 标准之后，我们发现帧信息结构有 40 个未用或保留的比特可以使用。通过复用帧信息结构已有字段的闲置空间，我们得以把操作令牌和会话 ID 等信息随读写请求本身一起发送给可信固态硬盘。

# 5 实验结果

本节我们给出可信固态盘的初步实验结果。可信固态硬盘的实现和测试代码都已开源，可以访问 http://github.com/tatetian/TrustedSSD 获取。我们的实验环境是一台 Intel 2.4GHz 四核处理器、8GB 内存的服务器，运行 64 位 Ubuntu 12.04 的 Linux 操作系统。可信固态硬盘（OpenSSD 开发板）与主机通过 SATA 2.0 接口连接。我们实验的主要目的是评估在固态硬盘中加入额外的安全机制对整体性能的影响。为此，我们分别在合成和真实的负载上做了测试。

## 5.1 合成负载的实验结果

合成的负载，按操作类型分为读请求、或写请求；按请求块大小可分为 2KB、4KB、8KB、16KB、32KB 和 64KB；按访问模式可分为：顺序和随机。为了完全控制负载的模式，我们编写了专用的测试工具用于生成合成负载。以 4KB 随机读写负载为例，测试工具发送给固态硬盘的 I/O 请求都是读请求，且每个读请求是 4KB 大小，起始地址是随机的。

图 5 中可以展示了在合成负载上的结果。从中可以看出，1）可信固态硬盘大幅提高了原 OpenSSD 固件的性能，尤其是随机读写的情况，最高达到了近 10 倍的性能提升（对 32KB 随机写的情况）；2）可信固态硬盘的访问控制只增加了很小的开销。这说明我们在第 4 节“设计与实现”中介绍对 FTL 和 ACL 的优化是有效的。

## 5.2 真实负载的实验结果

为了进一步验证对 ACL 优化的效果，我们又在真实负载上做了实验。

真实负载是靠重现收集自实际生产环境的存储访问日志来产生的。我们从公开的真实存储访问记录[25, 26]中选择了 Exchange、BuildServer、

Financial、TPC-C 和 WebSearch 等五个特点不同、有代表性的存储访问日志（负载特点见表 1）。我们编写了专门的测试工具用于解析和重现这些负载记录。为了更加贴近真实的使用情景，在真实负载的实验中，我们模拟多个不同用户（8 个用户）同时登陆可信固态硬盘的情况。

表 1 中比较了平凡 ACL 和优化 ACL 相对于禁用 ACL 的可信固态硬盘的性能。可以看到，平凡 ACL 会带来显著的开销，而优化后的 ACL（见 4.2 “访问控制层”）的开销只有不到 3%。由此可见，我们对 ACL 的优化是非常有效的。

结合合成和真实负载的实验结果，我们认为，可信固态硬盘的安全机制仅产生了很小的运行开销。

## 6 结论

本文提出一种解决大数据平台数据存储安全的新思路——存储内安全（In-Storage Security）——把对数据的访问控制从主机上的系统软件下放到底层存储内部。更具体地说，我们设计并开发了可信固态硬盘（TrustedSSD）。它提供安全增强的设备接口和协议，使得用户可以对存储中的数据施以细粒度的访问控制，保护存储中数据的安全性。访问控制层（Access Control Layer, ACL）是可信固态硬盘的安全引擎，与固态硬盘的存储引擎闪存转换层（Flash Translation Layer，FTL）紧密结合，实现高效的用户认证和操作授权。结合用户库函数和操作系统支持，可信固态硬盘可以为数据密集型应用（比如 Hadoop）提供安全的数据存储服务。

## 参 考 文 献


[1] Steven Musil. Google blames software update for lost gmail data[OL]: http://www.cnet.com/news/google-blames-software-update-for-lost-gmail-data/, 2011-02-28.

[2] David Crawford and Vanessa Fuhrmans. Germany Tackles Tax Evasion[OL].http://online.wsj.com/news/articles/SB10001424052748704197104575051480386248538, 2010-02-07.

[3] O’Malley O, Zhang K, Radia S, et al. Hadoop security design[J]. Yahoo, Inc., Tech. Rep, 2009.

[4] Klein G, Elphinstone K, Heiser G, et al. seL4: Formal verification of an OS kernel[C]//Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles.ACM, Big Sky, Montana, USA 2009: 207-220.

[5] David Floyer. Flash price trends disrupt storage[OL]. http://wikibon.org/wiki/v/Flash_Pricing_Trends_Disrupt_Storage.

[6] Computer Systems Laboratory at Sungkyunkwan University. The Open SSD Project[OL]. http://www.openssd-project.org/wiki/The_OpenSSD_Project, 2014-09-04.

[7] Kim S, Oh H, Park C, et al. Fast, energy efficient scan inside flash memory SSDs[C]//Proceeedings of the International Workshop on Accelerating Data Management Systems (ADMS). Seattle, WA, USA 2011.

[8] Do J, Kee Y S, Patel J M, et al. Query processing on smart SSDs: opportunities and challenges[C]//Proceedings of the 2013 international conference on Management of data. ACM, New York, USA 2013: 1221-1230.

[9] Kang Y, Kee Y, Miller E L, et al. Enabling cost-effective data processing with smart ssd[C]//Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on. IEEE, Long Beach ,CA, USA 2013: 1-12.

[10] Kang W H, Lee S W, Moon B, et al. X-FTL: transactional FTL for SQLite databases[C]//Proceedings of the 2013 international conference on Management of data. ACM, New York, USA 2013: 97-108.

[11] Anjo Vahldiek, Eslam Elnikety, Ansley Post, Peter Druschel, Deepak Garg, Johannes Gehrke, Rodrigo Rodrigues USENIX FAST'12 Work in progress, Feb. 2012.

[12] Anjo Vahldiek, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Ansley Post, Rodrigo Rodrigues, Johannes Gehrke. Guardat: A foundation for policy-protected data. Technical Report 2014-002, MPI-SWS, 2014.

[13] Brown K H. Security requirements for cryptographic modules[J]. Federal Information Processing Standards Publication, 1994: 1-53.

[14] THE COMMON CRITERIA. http://www.commoncriteriaportal.org/.

[15] What’s New in Linux 3.10. http://www.h-online.com/open/features/What-s-new-in-Linux-3-10-1902270.html 2013,7,1.

[16] MySQL code size over releases. https://www.flamingspork.com/blog/2013/03/05/mysql-code-size/ 2013,3,5.

[17] Lee S W, Park D J, Chung T S, et al. A log buffer-based flash translation layer using fully-associative sector translation[J]. ACM Transactions on Embedded Computing Systems (TECS), 2007, 6(3): 18.

[18] Lee S, Shin D, Kim Y J, et al. LAST: locality-aware sector translation for NAND flash memory-based storage systems[J]. ACM SIGOPS Operating Systems Review, 2008, 42(6): 36-42.

[19] Gupta A, Kim Y, Urgaonkar B. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings[M]. ACM, 2009.44(3):229-240

[20] Qin Z, Wang Y, Liu D, et al. Demand-based block-level address mapping in large-scale NAND flash storage systems[C]//Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. ACM, Scottsdale, Arizona, USA 2010: 173-182.

[21] Ma D, Feng J, Li G. LazyFTL: a page-level flash translation layer

optimized for NAND flash memory[C]//Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, Athens, Greece 2011: 1-12.

[22] Strunk, John D., et al. "Self-securing storage: protecting data in compromised system." //Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4. USENIX Association, 2000.

[23] Gibson, Garth A., et al. "File server scaling with network-attached secure disks."ACM SIGMETRICS Performance Evaluation Review 25.1 (1997): 272-284.

[24] Wylie, Jay J., et al. "Survivable information storage systems." Computer 33.8 (2000): 61-68.

[25] SNIA, IOTTA Repository Home. http://iotta.snia.org/.

[26] UMass Trace Repository.

http://traces.cs.umass.edu/index.php/Storage/Storage.