基于脚本面向敏感文档的动态指纹溯源架构设计与实现

包立兴[1,2]，赵峰[3]，黄小罗[1]，王洋[1*]

[1]（中国科学院深圳先进技术研究院 深圳 518055）

[2]（河北大学数学与信息科学学院 保定 071000）

[3]（中国科学院新疆生态与地理研究所 乌鲁木齐 830000）

摘要：数据溯源技术可以记录和追踪敏感文档的来源，从而防止文档泄露。传统的网络通路溯源对离线文档缺乏有效跟踪机制，基于加密文件的密钥追踪对共享文件不能保证有效溯源，现有的标注法、反向查询和数据水印技术，往往需要用户参与并在应用层实现，导致溯源的安全力度不够，缺乏透明性和灵活性，系统的整体扩展性不足。该文提出了一种创新的基于脚本的动态指纹溯源架构，该架构基于 Linux 内核实现底层溯源，加强文档溯源的安全性和透明性；基于用户脚本实现指纹追踪算法，提升文档溯源的灵活性和有效性。同时面向多负载共享需求设计指纹驱动算法，确保文档共享的高效性和可扩展性。经验证，本架构对操作系统的影响极小，同时具备出色的可扩展性。在处理单个或多个负载共享的场景时，指纹驱动算法展现了其透明性、实时性和高效性。

关键词 敏感文档；数据泄露；Linux 内核；动态溯源；用户脚本；指纹追踪

中图分类号：TP 302 文献标志码 A doi: 10.12146/j.issn.2095-3135.20240428001

Design and Implementation of Dynamic Fingerprint Traceability Architecture for Sensitive Documents Based on Scripts

BAO Lixing[1,2], ZHAO Feng[3], HUANG Xiaoluo[1], WANG Yang[1*]

[1] (*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, 518055, China*)

[2](*College of Mathematics and Information Science of Hebei University, Baoding, 071000, China*)

[3](*Xinjiang Institute of Ecology and Geography, Chinese Academy of Sciences, Urumqi, 830000, China*)

[*]*Corresponding Author*: *yang.wang1@ siat.ac.cn*

Abstract: Data provenance technology is capable of recording and tracking the origins of sensitive documents to prevent their leakage. Traditional network path tracing methods are ineffective in tracking offline documents, and key tracing for encrypted files does not ensure reliable provenance for shared files. Existing techniques such as annotation, reverse querying, and data watermarking often require user involvement and are implemented at the application layer, resulting in inadequate security, lack of transparency and flexibility, and

来稿日期：2024-04-28 修回日期：2024-05-28

基金项目：第三次新疆综合科学考察项目 (2021xjkk1300)；国家重点研发计划资助(2021YFF1201700)；深圳市科技计划项目(深港澳 C 类，SGDX20220530111001003)

作者简介：包立兴，硕士研究生，研究方向为数据安全存储系统；赵峰，中国科学院中亚生态与环境研究中心信息中心工程师，中国科学院新疆生态与地理研究所工程师；黄小罗，中国科学院深圳先进技术研究院高级工程师，博士生导师，研究方向为 DNA 数据存储算法与技术；王洋（通讯作者），博士，研究员，研究方向为云存储、云计算，E-mail：yang.wang1@ siat.ac.cn。

insufficient overall system scalability. This paper introduces an innovative script-based dynamic fingerprint provenance architecture that utilizes modifications to the Linux kernel to achieve foundational provenance, enhancing the security and transparency of document tracing. The fingerprint tracking algorithm is implemented through user scripts, improving the flexibility and effectiveness of document provenance. Additionally, the fingerprint-driven algorithm is designed to meet the demands of multi-load sharing, ensuring efficient and scalable document sharing. Upon verification, this architecture has a minimal impact on the operating system and exhibits excellent scalability. In scenarios involving single or multiple load sharing, the fingerprint-driven algorithm demonstrates transparency, real-time performance, and efficiency.

Keywords: Sensitive documents; Data leakage; Linux kernel; Dynamic traceability; User script; Fingerprint tracking

Funding: This project is supported by the 3rd Xinjiang Scientific Expedition Program (2021xjkk1300), the National Key R&D Program of China (2021YFF1201700), and the Shenzhen Science and Technology Plan Project (SGDX20220530111001003).

1 引 言

随着互联网信息技术的快速发展，数据密集型领域[1-3]和应用的数据规模正以指数速度膨胀，数据存储技术已从直连附加存储[4]、存储区域网络[5]和网络附加存储[6]进化到如今云计算大数据时代的存储即服务的软件定义存储（Software Defined Storage，SDS）[7]。存储架构的转变满足了数字资源在稳定性、重要性和并发性方面的需求，但其安全性却常被忽略。

文档型数据作为最灵活的数据模型，具有高敏感性和高流动性的特点，因此容易在未经授权或违反规定的情况下引发数据泄露问题。据天际友盟 2022 年上半年的监测，商业机密数据泄露事件中，文档类型的泄露占比达到 73.2%。2023 年，全球公开报道的数据安全事件中，数据泄露事件占比为 67.5%，涉及的数据量超过 51.8TB，总计约 103.8 亿条记录。

溯源[8]是检测数据泄露根源的有力手段，因其能够记录数据的来源、变化和流向[9]，帮助用户实现数据的可信、可控和可审计，而被广泛地运用在物联网、云计算等各种领域[10]。

最初的研究者们尝试通过在数据中附加辅助信息（如背景、作者、时间、出处等[11]）来追踪数据源。例如，polygen 模型[12]通过在数据上添加源标签和中间源标签来解决数据来源问题；Lee 等人[13]提出将数据源的信息作为属性加入到半结构化文档中；Buneman 等人[14,15]设计了基于注释代数的框架来追踪数据在查询过程中的流动；而 Deuth 等人[16]使用标签记录数据派生过程中的规则和事实。

这些方法虽然能够有效记录数据的历史状态和处理信息，但也可能导致存储空间的大量占用。为了解决这个问题，研究者们引入了反向查询技术，这种技术只需要存储少量数据，通过逆置函数[17,18]、逆操作[19]和逆向查询[20,21]，可以从数据结果反推数据[22,23]过程，进而计算数据的来源。反向查询可以简化追踪过程，但此方法受限于逆置函数的复杂性和实现难度。

数字水印[24]技术通过在数据中隐藏标识信息来解决数据溯源问题，它具有鲁棒性、可提取性和不可见性。数字水印分为图片水印、文本水印和数据库水印[25-27]。图片水印根据隐藏位置分为空域[28]和变换域[29]；文本水印主要有基于特定字符和基于自然语言的两种形式；数据库水印针对二维表格数据，通过改变属性值来嵌入水印。这些方法虽然有效，但也存在局限性，例如图像水印可能会降低图像质量，文本水印可能会影响文本的可读性，而数据库水印通常只适用于结构化数据，不适合敏感文档的溯源。

尽管数据溯源领域的研究日益增多，但多聚焦于溯源算法在理论和技术上的创新，而对算法的底层实现机制和在特定架构中的应用探讨却鲜有涉及。此外，现有数据溯源机制

应用端的实现和部署，其本身的安全性会因人为操作不当而受到威胁，如果溯源机制与特定的应用程序或系统紧密集成，那么在不同环境或不同类型的数据中应用相同的溯源算法可能会变得困难，这会限制算法适应新场景的能力，进而影响整个溯源系统的扩展能力。

基于此，本研究提出一种独特的动态溯源架构—内核[30]驱动用户指纹脚本。通过结合现有的哈希算法来生成独特的数据指纹，并利用系统底层内核驱动用户脚本实现数据动态溯源，不仅提高了数据安全性，还为数据溯源系统的完善提供了新的视角。该架构具有以下创新点：

(1) 针对溯源的安全性和灵活性，引入了内核驱动脚本溯源机制，该机制一方面依托内核的安全性和稳定性为上层脚本提供统一透明的硬件溯源接口，另一方面通过用户空间的脚本计算溯源信息，能有效利用用户空间的灵活性和自定义能力来弥补内核自身的局限性。

(2) 针对溯源的有效性和可扩展性，提出了指纹脚本溯源算法，具体地，采用用户信息的哈希串指纹来有效追溯离线文档的来源。同时考虑到多负载文档共享的场景需求，引入了异步和同步两种指纹驱动算法，确保敏感文档共享的效率和可扩展性。

2 动态溯源架构设计与实现

本文中，动态溯源指的是一种在文件从服务器下载之前，由文件服务器动态嵌入溯源标识信息的技术。传统的实现方法涉及在文档下载过程中由应用层程序添加溯源信息，但此方法需要用户或开发人员在应用层进行操作，增加了复杂性和参与门槛，另外应用层的实现可能不够安全，容易遭受数据泄露和篡改的威胁。

本研究的设计思想是将溯源机制下沉到操作系统内核层，利用内核的安全性和稳定性来管理溯源信息，但是内核层在面对用户的个性化安全需求方面存在限制和不够灵活的问题。因此本文提出一种基于 Linux 内核和用户脚本的动态溯源架构，具体地，溯源标识信息由用户空间的脚本生成，动态嵌入动作则由内核对脚本进行实时驱动，该方案将文件处理为系统资源，其安全性由内核控制，灵活性则由用户脚本来确保，从而构建了一个既透明又灵活的动态溯源系统。

如图 1 所示，在 Linux 系统架构中，文件系统如 ext3、ext4 以及特殊的内存文件系统 /proc，都是通过虚拟文件系统（Virtual File System，VFS）的统一接口进行管理的。例如，当用户尝试打开一个位于 ext4 文件系统上的文件时，VFS 会激活相应的文件系统驱动来处理这一操作。文件系统驱动的任务是解析文件系统中的元数据，定位文件的数据块位置，并与设备驱动程序合作，完成与物理硬件的数据交换。这一过程确保了不同文件系统之间的高效协作和数据的准确访问。

本研究的动态溯源架构通过扩展 ext4 文件系统，创建一个名为 myext4 的新文件系统，如图 1 的动态溯源架构所示，新文件系统对文件访问到数据传输的整个链路进行了修改。具体地，监控文件打开的系统调用，通过 Netlink 机制[31]与运行在用户空间的溯源脚本通信，确保在文件读取操作之前完成溯源信息的计算。之后将包含溯源信息的新文件对象返回到用户的下载进程。此时，读和写系统调用直接处理的是已经被添加了溯源信息的数据。最终，DMA 控制器负责将数据从网络缓冲区传输到网卡，完成数据的发送过程。

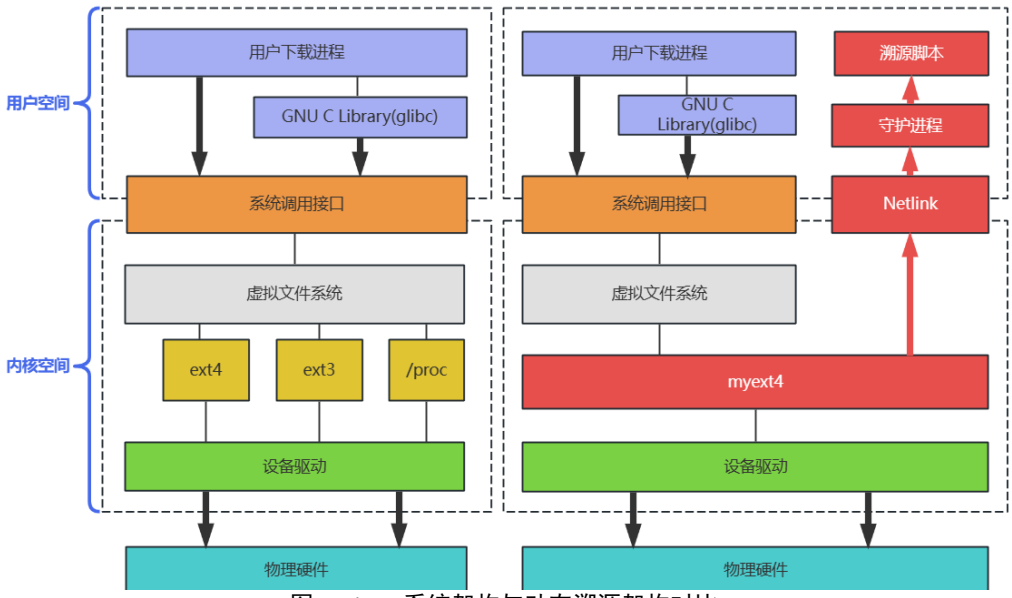

图 1 Linux 系统架构与动态溯源架构对比

Fig. 1 Comparison between Linux system architecture and dynamic traceability architecture

2.1 架构设计

如图 2 所示，在内核空间，以 Linux 常用的 ext4 文件系统为例，将其设计为一个基于脚本的用户级文件系统（myext4），它是一个可加载的模块。可加载模块是辅助文件系统类型和设备等服务的驱动程序，以允许在内核运行时将代码动态加载到内核中，而不需要重新编译或重新安装。这个文件系统包含有对 ext4 文件系统的 open()、read()、write()等各种操作类型的表，并扩展了 open()函数的驱动脚本功能。要使用 myext4，必须和其它文件系统一样挂载到需要的目录。Linux 的 VFS 接口则定义了目录操作，地址操作，节点操作和与 myext4 相关的文件操作表。

在用户空间，守护进程（Daemon）以 Netlink 用户端通信进程的方式出现，充当溯源脚本和 Netlink 用户进程之间的双向通信通道。溯源脚本是用户在文件系统目录下编写的为文件添加溯源信息的脚本。该脚本可以通过继承机制被低层次的文件目录使用。如果文件目录中没有溯源脚本，为避免不必要的进程通信，myext4 组件会智能的过滤掉溯源操作，实现了用户的按需溯源需求。

内核到守护进程之间的通信利用了环状缓冲区和信号量机制，环状缓冲区实现为数组表示的循环队列，每个队列元素表示为一个结构体，该结构体包括文件路径、阻塞信号量、文件返回信息，阻塞信号量用来阻塞缓冲区中位置的文件操作，文件返回信息是脚本执行完后返回的信息。环状缓冲区管理用到两个计数信号量。两个计数信号量 emptyCount 和 fullCount 分别表示可用的空闲缓冲区个数和已经填满的缓冲区个数。因此内核到守护进程之间的单向通信可以看作是一个生产者-消费者模型，内核是生产者，守护进程是消费者。

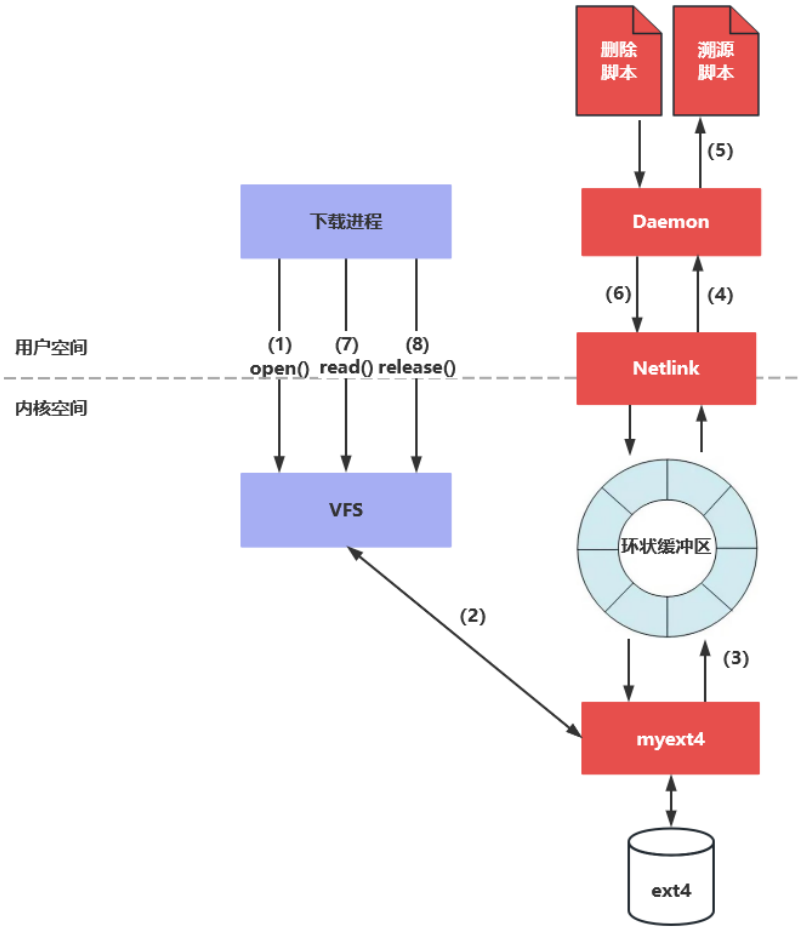

图 2 动态溯源架构通信流程图

Fig. 2 Dynamic traceability architecture communication flow chart

2.2 架构实现

如图 2 所示，当某个用户的文档下载进程通过数据传输协议连接远端服务器下载某个文件时，会在下载进程和用户脚本之间产生如下的通信流程：

步骤(1)：下载进程首先根据文件路径和用户信息执行文件打开操作，文件打开函数首先检查守护进程是否正在运行，如果没有运行就不执行脚本；如果正在运行，VFS 开始从 myext4 装载点的目录结构中搜索访问路径。假设用户想要下载/mnt/myext4/sftpu1/1.pdf 文件，VFS 从/mnt/myext4/开始，在 myext4 的目录结构中查找目录 sftpu1，然后在 sftpu1 的条目中搜索文件 1.pdf。如果路径名的任何部分不存在，则文件打开系统调用将返回一个访问错误。如果找到该文件，执行步骤(2)；

步骤(2)：VFS 调用 myext4 文件操作表的文件打开函数，扩展的文件操作代码先检查 emptyCount 的值[32]，如果为 0 或负数，就阻塞内核的操作；如果为正整数，修改 emptyCount 值，执行步骤(3)；

步骤(3)：将结构体数据填充到环形数组队列中，同时阻塞该队列位置对应文件打开操作，修改 fullCount 信号量的值；

步骤(4)：Netlink 内核通信模块将环形队列中的数据通过 Socket 传输到用户空间，同时修改 fullCount 的值，用户级守护进程获取 Socket 消息体，通过解析消息，得到文件路径；

步骤(5)：用户级守护进程将文件路径信息发送到溯源脚本，脚本读取文件路径和用户名 sftpu1，对目标文件执行溯源计算操作（脚本本身的文件操作会被 myext4 框架过滤），将新数据存到一个新文件中。

步骤(6)：当脚本执行完毕后，守护进程通过 Netlink 将脚本处理得到的新文件路径返回给内核。内核收到信息后，唤醒此文件的文件打开操作，同时修改 emptyCount；打开操作继续查找新文件的节点和目录项，一旦找到，它将替换旧文件的节点和目录项，并释放掉旧文件的节点和目录项，然后执行步骤(2)，VFS 向应用程序返回该文件描述符。

步骤(7)和步骤(8)：下载进程根据文件描述读取带有溯源信息的数据，一旦读取完成，执行 release()系统调用，release 函数和文件打开函数一样被拦截去用户空间执行删除溯源文件脚本，这样用户打开源文件、添加溯源信息、读取溯源文件、删除溯源文件的动作一气呵成，为用户提供了一种透明加载溯源文件的机制。

3 指纹算法

内核底层溯源增强了溯源的安全性和透明性，而用户级溯源脚本则确保溯源的有效性和灵活性。本节内容专注于溯源脚本中溯源信息的计算方法，提出了指纹溯源算法，并针对多文档共享情境下的溯源，定制了两种内核驱动指纹溯源的场景算法。

3.1 指纹溯源算法

指纹溯源算法利用图 2 流程（6）中的溯源脚本来编写，通过在文档中嵌入含有用户个人信息的指纹来追踪文档的使用者。这样，在文档泄露发生时，可以迅速定位泄露源头，有效防止信息进一步泄露和扩散。

指纹溯源算法使用 Python 语言编写， 使用安全散列算法 1（Secure Hash Algorithm，SHA1）压缩用户信息[33]，PyPDF2 模块动态生成含有指纹信息的文件。

SHA1 是一种加密哈希函数，它接受任意长度的输入并产生一个 160 位（20 字节）的哈希值，通常表示为 40 个十六进制数字。本研究将用户名作为哈希函数的输入，产生一个 160 位的和用户名一一对应的哈希串。PyPDF2 是一个开源的 Python 库，专门用于处理 PDF 文件。它不仅支持读取和写入 PDF 文档，还提供了分割和合并 PDF 页面的功能。此外，PyPDF2 还能够为 PDF 文档添加水印、执行加密和解密操作，以及其他多种文档处理功能。

指纹溯源算法主要用到两个函数 createFingerprint()和 addFingerprintToPage()，create Fingerprint()函数负责设置指纹的内容、大小、字体、颜色和透明度，然后通过 canvas 画布在生成一个用户指纹文件；addFingerprintToPage()打开内核传过来的文件名对应的文件，创建输入流，初始化一个 PdfFileReader 对象，然后从这个对象中获取 PDF 文档的每一页，将每一页与用户指纹文件合并并压缩，然后添加到 PdfFileWriter 对象中，由它写入到一个新的指纹文件对象中。

算法 1：指纹溯源算法

输入：源文件 $sourceFile$，用户名为 $user$.

输出：指纹文件 $fingerprintFile$.

```
1  fingerprintContent ← SHA1(user)        //用户名的哈希串作为指纹信息
   //创建指纹内容文件
2  fingerprintPDF ← createFingerprint( fingerprintContent)
3  open(sourceFile,tempFile)              //将源文件以临时文件的形式打开
4  for i ←1 to sourceFile.pageSize()
     //将临时文件和指纹内容文件合并
5    addFingerprintToPage(tempFile,i, fingerprintPDF)
6  end for
```

7 $saveFile(tempFile, fingerprintFile)$ //存储临时文件为指纹文件
8 $deleteFile(fingerprintPDF)$ //删除指纹内容文件
9 **return** *fingerprintFile*

算法 1 展示的是指纹溯源算法的伪代码，该算法的时间复杂度和空间复杂度分析如下：

时间复杂度：第 1 行的 SHA1 哈希函数通常具有固定的时间复杂度，因为它处理的是固定长度的输入（用户名），所以这一步的时间复杂度是 O(1)。第 2 行创建指纹内容文件的操作，时间复杂度也是 O(1)，因为它只是基于哈希串生成一个文件。第 4-6 行的循环是算法的主要部分，它依赖于源文件的页数。对于每一页，都会执行添加指纹的操作，所以这部分的时间复杂度是 O(p)，其中 p 是源文件的页数。算法的总体时间复杂度是由循环中的操作决定的，即 O(p)。

空间复杂度：算法使用了一个额外的文件（fingerprintPDF）来存储指纹内容，所以这需要额外的空间，但这个空间是固定的，所以空间复杂度是 O(1)。同时算法涉及到存储指纹文件（fingerprintFile），假设每个指纹文件大小相同，空间复杂度是 O(s)，其中 s 是指纹文件的大小。

3.2 指纹驱动算法

本研究在指纹溯源算法的基础上，进一步设计了指纹驱动算法，旨在解决多负载共享环境下，内核空间对用户空间指纹脚本的驱动问题，从而确保多文件共享的效率和可用性。具体地，针对下载负载较高但对即时下载需求不敏感的场景，设计了异步指纹驱动算法；针对下载负载较低且对文件的实时性和可靠性有较高要求的场景，设计了同步指纹驱动算法。这两类场景算法都充分利用了内核空间与用户空间的协同工作机制，以保证数据处理过程的高效率和准确性。

算法 2：异步指纹驱动算法

输入：假设有 n 个文件打开进程，每个进程打开的源文件为 $sourceFile_i$，每个进程
的文件打开模式为 $openMode_i$，每个进程的文件打开程序为 $openProgram_i$.

输出：每个源文件对应的指纹文件 $fingerprintFile_i$.

Kernel:

1 初始化环形队列 Q, 空状态信号量 E 为 0, 满状态信号量 F 为 Q 的大小.
2 **for** $i \leftarrow 1$ **to** n
3 **if** $openMode_i$ = O_RDONLY **then** //判断文件打开模式和打开程序
4 **if** $openProgram_i$!= $fingerprintProgram$ **then**
5 **if** $E > 0$ //判断环形队列是否已满
6 $E \leftarrow E - 1$
7 $Q.push(souceFile_i)$
8 $F \leftarrow F + 1$
9 $kernel\_broadcast(Q.pop())$ //内核向用户空间广播数据
10 $F \leftarrow F - 1, E \leftarrow E + 1$
11 **else** $block(openProgram_i)$, **continue** //阻塞该进程
12 **end for**

User:

13 **for** $i \leftarrow 1$ **to** n

14 $user\_recvmsg(sourceFile_i)$ //用户空间进程接收消息

//为源文件添加指纹

15 $fingerprintFile_i \leftarrow addFingerprintToPage(sourceFile_i)$

16 将 $fingerprintFile_i$ 存入共享目录 D.

17 **return** $fingerprintFile_i$

18 **end for**

异步指纹驱动算法的伪代码如算法 2 所示，内核操作只保证多个源文件打开进程的同步、文件打开模式和打开程序条件的正确判断以及数据从内核向用户空间的准确传输；用户脚本操作保证每个指纹文件的静默处理并自动保存至共享目录。这样用户在下载共享文件时，与访问 ext4 文件系统本身一样无需等待指纹脚本处理完成，因为指纹处理在后台进行，处理完的指纹文件会自动保存，用户可以直接下载。

异步指纹驱动算法的时间复杂度分成内核和用户两部分分析：1) Kernel 部分：由于存在一个 for 循环（步骤 2-12），循环体内的操作可以认为是常数时间的操作。因此循环的时间复杂度是 O(n)，其中 n 是需要处理的文件数量；2) User 部分：用户脚本的操作（步骤 13-18）时间复杂度涉及到指纹添加 addFingerprintToPage 函数，也是 O(np)，综合起来，整个算法的时间复杂度是 O(np)。

空间复杂度分析如下：1) Kernel 部分：因为环形队列 Q 的大小是固定的，空间复杂度是 O(1)；2) User 部分：用户的操作涉及到存储指纹文件，假设每个指纹文件大小相同，空间复杂度是 O(ns)，其中 s 是指纹文件的大小。

算法 3：同步指纹驱动算法

输入：假设有 n 个文件打开进程，每个进程打开的源文件为 $sourceFile_i$，每个进程

的文件打开模式为 $openMode_i$，每个进程的文件打开程序为 $openProgram_i$.

输出：每个源文件对应的指纹文件 $fingerprintFile_i$.

Kernel:

1 初始化环形队列 Q, 空状态信号量 E 为 0, 满状态信号量 F 为 Q 的大小.

2 **for** $i \leftarrow 1$ **to** n

3 **if** $openMode_i$ = O_RDONLY **then**

4 **if** $openProgram_i$!= $fingerprintProgram$ **then**

5 **if** $E > 0$

6 $E \leftarrow E - 1$

7 $Q.push(souceFile_i)$

8 $F \leftarrow F + 1$

9 $kernel\_broadcast(Q.pop())$

10 **while** !$kernel\_rcv\_skb(fingerprintFile_i)$

11 $block(openProgram_i)$ //内核未接收到消息时阻塞进程

12 $F \leftarrow F - 1, E \leftarrow E + 1$

//查找源文件目录项

13 $sourceFile_i.dentry \leftarrow lookup\_dentry(sourceFile_i)$

14 $dput(sourceFile_i.dentry)$ //释放源文件目录项的空间

15 $iput(sourceFile_i.dentry.d\_inode)$ //释放源文件的索引节点空间

//查找指纹文件目录项

16 $fingerprintFile_i.dentry \leftarrow lookup\_dentry(fingerprintFile_i)$

//替换源文件目录项

17 $sourceFile_i.dentry \leftarrow fingerprintFile_i.dentry$

18 $sourceFile_i.dentry.d\_inode \leftarrow fingerprintFile_i.dentry.d\_inode$

19 **else** $block(openProgram_i)$, **continue**

User:

20 $user\_recvmsg(sourceFile_i)$

21 $fingerprintFile_i \leftarrow addFingerprintToPage(sourceFile_i)$

22 $user_$sendmsg($fingerprintFile_i$) //用户空间发送消息

23 **return** $fingerprintFile_i$

24 **end for**

同步指纹驱动算法伪代码如算法 3 所示，内核除了执行与异步算法相同的任务外，还需要进程同步锁机制来保证文件打开进程在未接收到用户空间消息时保持阻塞状态；用户脚本处理每个指纹文件，并将数据从用户空间传输回内核。这样用户在下载共享文件时，需要等待用户空间的指纹脚本处理完成，以确保文档的最新状态和减少内存消耗。

同步指纹驱动算法的时间复杂度分析如下：算法的主体是一个从 1 到 n 的循环，在循环内部，有多个条件判断和一个 while 循环（步骤 10），这个 while 循环的次数取决于 kernel_rcv_skb 函数接收到指纹文件的时间。假设平均每个文件的处理时间是常数，则这部分的时间复杂度为 O(1)，因此循环的时间复杂度是 O(n)，其中 n 是需要处理的文件数量；addFingerprintToPage 函数（步骤 21）根据算法 1 可知指纹溯源算法的时间复杂度与文件的页数有关。如果每个文件的页数是 p，则这个函数的时间复杂度是 O(p)。综上，假设每个文件的页数大致相同，则整个算法的时间复杂度是 O(np)。

空间复杂度分析如下：环形队列 Q 的大小是固定的，两个信号量 E 和 F，以及一些临时变量（如 $sourceFile_i$.dentry 和 $fingerprintFile_i$.dentry），但这些都不会随着 n 的增大而增大，所以算法的空间复杂度是 O(1)。

4 实验结构与分析

本节深入探讨了动态溯源架构的可扩展性和通信性能，并对指纹溯源算法处理不同类型 PDF 文档的效率进行了详尽测试。进一步，我们将这一架构应用于一个文档共享系统，该系统为用户提供了一系列安全功能，包括安全认证、数据隔离、安全传输和离线追踪。在此系统上，本文分析了在不同文件共享负载情况下，两种指纹驱动算法的响应性能，以确保系统的高效运行和用户的安全需求得到满足。

4.1 实验环境

实验服务器硬件环境的具体配置为：远程服务器：Intel(R) Xeon(R) Gold 6133 CPU @ 2.50GHz、Cirrus Logic GD 5446 32M、8GB 内存、CentOS Linux release 7.6.1810，内核版本：Linux 5.15.72；本地服务器：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz、

NVIDIA GeForce RTX 3060 Laptop GPU 6GB、16.0 GB 内存、Windows 10 家庭中文版、FileZilla Client 3.62.2。

4.2 动态溯源架构性能分析

在设计支持动态溯源的 myext4 文件系统架构时，应确保其基本功能在未运行用户空间脚本时仍能正常执行，并且性能不低于传统的 ext4 文件系统。此外，当启用用户空间脚本时，架构引入的额外开销应该是最小的，特别是与脚本执行的成本相比。这样可以确保系统即使在执行额外的溯源任务时，也能保持高效率和响应速度。

本实验首先评估在不执行用户脚本的情况下，myext4 框架相较于标准 ext4 的性能损耗，以此来测试框架的可扩展性。

如图 3 所示，实验观察了在打开 200 页全文本文件 1 次、10 次、50 次、100 次和 200 次的情况下，ext4、myext4 异步指纹驱动算法和同步指纹驱动算法下打开文件所需的时间。

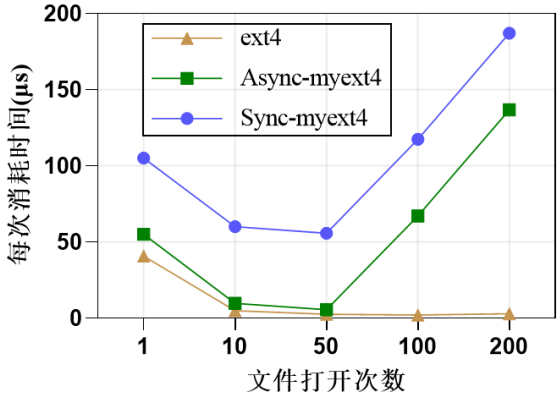

图 3 无脚本时两种算法下框架的开销

Fig.3 The overhead of the framework under the two algorithms without scripts

结果显示，随着文件打开次数的增加，ext4 的性能逐渐稳定，而 myext4 无论是在同步还是异步算法的模式下，每次打开文件的时间消耗都在增加。这一现象主要是由于 myext4 设计中包含的用户脚本过滤机制，以及内核同步机制和 Netlink 消息传递系统的延迟。在现实生活中，文件通常不会被频繁打开，因此在不运行脚本的情况下，由 myext4 引入的微小性能损耗可以忽略，其性能可接近于 ext4。

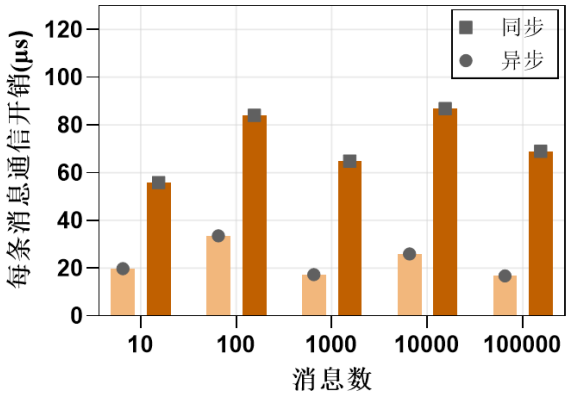

图 4 动态溯源架构通信开销

Fig. 4 The communication overhead of dynamic traceability architecture

本实验测试了 myext4 框架在不同负载下的通信性能，如图 4 所示，通过 Netlink 内核模块，向用户级守护进程发送了不同数量的消息，从 10 条增加到 10 万条，记录了在同步和异步两种算法模式下，每条消息的延迟时间。同步算法模式下，脚本仅进行消息计数，不执行其他操作。异步算法模式则不等待消息处理直接返回。这项测试专注于内核与脚本间的进程间通信（IPC）路径，不包括文件操作或磁盘活动。所有消息内容均相同，仅包

含文件路径信息。实验结果显示，无论消息数量如何，每条消息的通信延迟都保持在 20 至 80 微秒之间，对系统性能的影响微乎其微。因此认为该通信开销在实际应用中可忽略不计。

4.3 指纹溯源算法性能分析

在评估指纹溯源算法的性能时，我们通常会考虑文件大小的影响，预期算法在处理小文件与大文件时会表现出不同的性能。然而，文件的页数和内容复杂性（例如文本、图像和表格的组合）都会对文件大小产生影响。

为了深入探究文件大小和内容复杂性这两个参数如何单独及共同影响指纹溯源算法的性能，本实验综合考量了文件的页数和内容类型。这样做可以全面评估指纹溯源算法对不同 PDF 文件类型的适应性和处理效率，进而揭示算法在处理多样化文件时的效率和灵活性。

首先我们通过不同文件的页数和内容，将文件划分为 KB 级和 MB 级两个大小类别，以探索这些变量如何综合影响指纹溯源算法的性能。图 5 的数据显示，PDF 文件的页数越多，添加指纹的时间也越长。对于页数相同的文件，30 页的纯文本 MB 级文件添加指纹的时间超过了纯文本 KB 级文件。而在 50 页和 70 页的文件测试中，含有图像和表格的 KB 级文件的处理时间反而更长，这是因为这些图像和表格相比纯文本需要额外的处理步骤，比如渲染和合并。

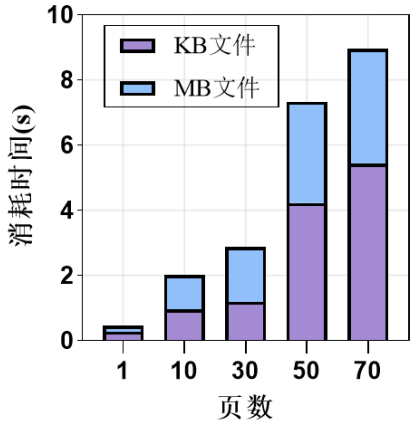

图 5 不同页数的 KB 级和 MB 级文件的算法性能

Fig. 5 Algorithm performance for KB-level and MB-level files with different page numbers

接下来分别研究文件页数和内容各自与算法性能的关系。首先选择了页数相同但内容不同的文件进行比较。结果显示，消耗时间并没有随文件内容呈现出一致的规律。图 6 中，在 66 页的文件测试中，纯文本文件的处理时间最长，而纯图像文件的处理时间最短，这是因为在使用 PyPDF2 库进行指纹添加时，文字内容的比例越高，所需的合并压缩时间也越长，因此读取和写入操作的时间开销也随之增加。而对于 10 页和 30 页的文件，图像文本的文件处理时间最长，其次是纯文本文件，纯图像文件处理时间最短。造成这种现象的原因可能与图像的分辨率、图像复杂度以及图像与文本的比例有关，且这种关系无法准确探究。

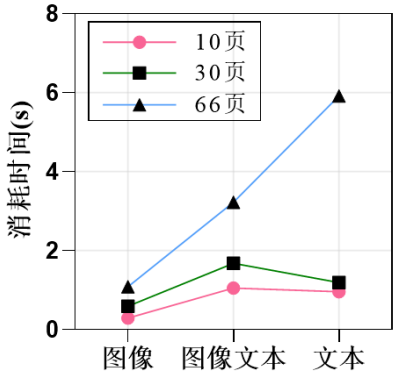

图 6 文件内容和算法性能的关系

Fig. 6 The relationship between the file content and the performance of the algorithm

为了探究指纹溯源算法性能与 PDF 文件页数之间的关系，我们进行了一项对比实验，实验中使用了内容格式为完全相同的纯文本，但页数不同的 PDF 文件。如图 7 所示，实验结果表明，在文件内容保持一致的情况下，随着文件页数的增加，指纹溯源算法所需的处理时间也相应增长。这一现象反映了 PyPDF2 模块在添加指纹时是按照 PDF 的每一页来操作的特点，但这种关系是非线性的。

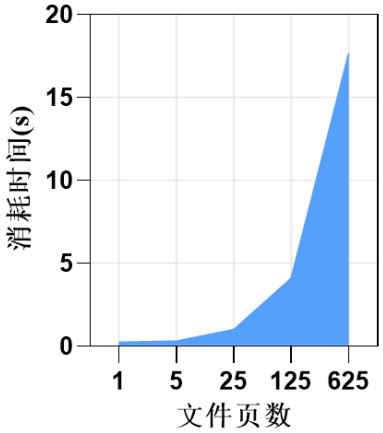

图 7 文件页数和算法性能的关系

Fig. 7 The relationship between the number of pages in a document and the performance of the algorithm

综上所述，指纹溯源算法的性能受多种因素影响，其中文件页数是一个重要的可观因素，内容复杂性的增加，如图像和表格的加入，会导致算法步骤增多，从而延长处理时间，然而图像的分辨率、复杂性和比例等因素会使这种影响呈现不一致性。因此，在评估指纹溯源算法的性能时，应综合考虑这些变量。

4.4 系统共享性能分析

本实验对单个文件的共享性能进行测试，然后将测试结果与使用指纹溯源算法预先静态集成的指纹的性能开销进行对比，以此来评估动态指纹溯源技术的性能影响。

我们假设所有测试 PDF 文件的内容和格式一致，缓冲区大小统一设置为 20，但页数不同，分别为 10 页、50 页和 200 页，以此来模拟不同规模的文档处理场景。这样的页数设计有助于展示算法在处理不同大小文件时的性能表现：10 页的文件代表小型文档，50 页的文件相当于中等规模的报告，而 200 页的文件则类似于大型数据集或书籍。基于这些参数，我们比较了动态指纹溯源和静态指纹溯源的开销，以及 myext4 文件系统框架本身的开销，即内核与用户空间通信产生的进程间通信（IPC）开销。

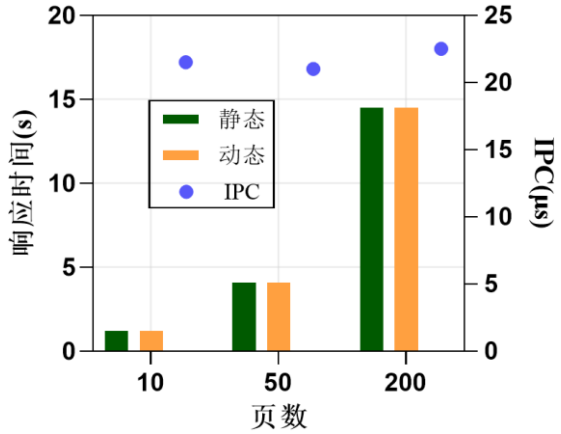

图 8 异步指纹驱动算法单文件共享性能

Fig. 8 The performance of asynchronous fingerprint algorithm for single file sharing

根据图 8 的实验结果，异步驱动算法实现的动态指纹溯源与静态指纹溯源在性能开销上几乎没有差异，且与 PDF 文件的页数成正比，相关系数基本保持不变。这表明在异步驱动算法下，myext4 文件系统的扩展对原有内核文件系统的性能影响微乎其微，同时提供了更高级别的指纹功能，并增加了系统的活跃性。然而，这也要求 SFTP 用户重新访问共享目录以下载指纹文件，从而增加了空间复杂度。

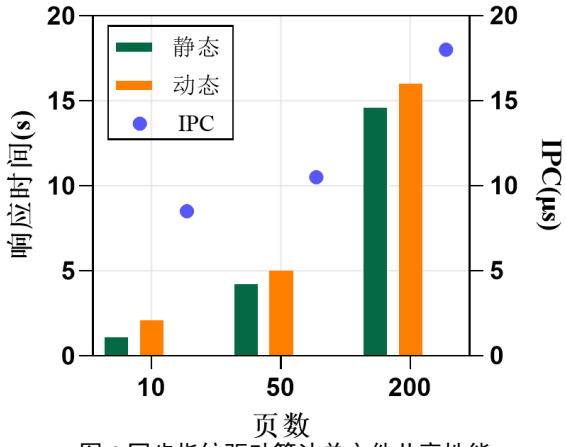

图 9 同步指纹驱动算法单文件共享性能

Fig. 9 The performance of synchronous fingerprint algorithm for single file sharing

图 9 的实验结果显示，在同步驱动算法下，动态指纹溯源与静态指纹溯源的性能开销相比较时，对于 10 页的 PDF 文件，myext4 文件系统的额外开销仅为 0.16 秒。随着文件页数的增加，无论是静态还是动态，时间消耗都呈现上升趋势，但 myext4 的开销仍保持在 1.2 至 1.3 秒之间，这在用户可接受的范围内，因此可以认为这种性能差异不具有决定性影响。此外，同步驱动实现的动态指纹溯源的性能与 PDF 文件页数成正比，但由于算法的开销分摊，这种相关性并不稳定。

当文件共享数量增加，即异步和同步并发请求同时增多时，访问 PDF 文件的页数为 200 页时的性能开销的变化如图 10 和 11 所示。在这种情况下，系统的性能开销会随着并发请求的增加而变化。

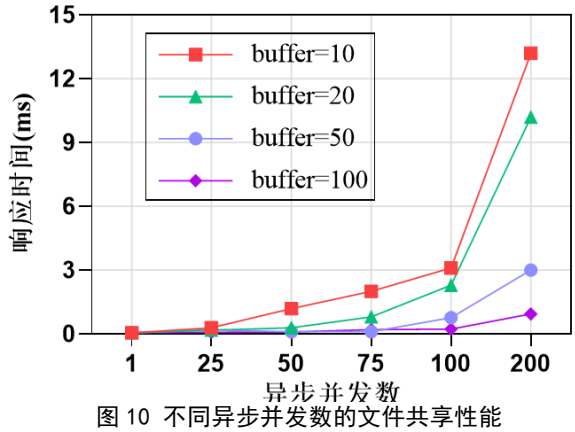

图 10 不同异步并发数的文件共享性能

Fig. 10 The performance of file sharing with different numbers of asynchronous concurrent operations

图 10 展示了在缓冲区大小固定的情况下，异步请求的响应时间会随着并发量的增加而稳定上升。同时，当并发量保持不变时，响应时间会随着共享环形缓冲区大小的增加而降低。值得注意的是，并发量越大，缓冲区大小对响应时间的影响也越显著，响应时间的下降也越明显。

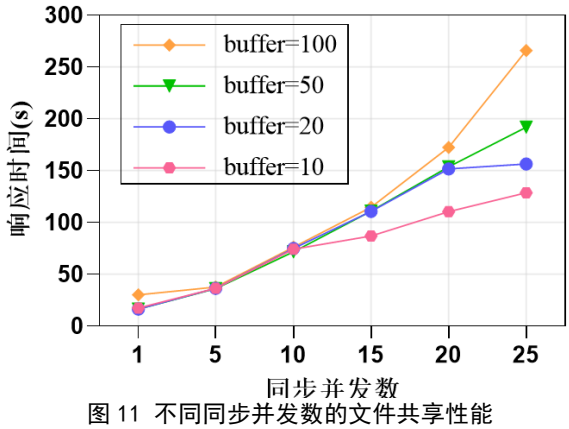

图 11 不同同步并发数的文件共享性能

Fig. 11 The performance of file sharing with different numbers of synchronous concurrent operations

如图 11 所示，在同步算法下，随着并发请求量的增加，响应时间也会相应增长。当缓冲区大小固定时，可以观察到响应时间的增长并不稳定。此外，当并发量保持不变时，响应时间并不总是随着缓冲区大小的增加而减少；有时较大的缓冲区并不比较小的缓冲区表现得更好。这一现象与理论预期不符，理论上认为每个进程应能同时访问自己的缓冲区，从而实现最优的并发性能。

这种差异可能是因为，当进程数小于等于缓冲区数时，每个进程都需要一定的系统资源来管理其缓冲区。这包括内存管理、进程调度和信号量同步等任务。随着缓冲区数量的增加，这些管理任务的开销可能会导致性能降低。特别是当缓冲区数量达到一定水平后，这些开销可能会抵消并发操作所能带来的性能提升。

综上所述， myext4 文件系统的扩展在处理动态指纹溯源时，即使在不同的文件页数和共享并发数下，也能保持良好的性能表现，异步驱动算法下响应时间的增长更为平稳，而同步驱动算法下响应时间的增长可能会因为系统资源管理的开销而变得不稳定，需要注意管理通信开销和空间复杂度。

4.5 讨论

在本研究中，我们通过关键词如“数据溯源”、“溯源机制”和“溯源计算方法”，在 Web of Science、ACM Digital Library、SpringerLink、中国知网和谷歌学术等数据库中进行了全面的文献搜索。在审视国内外数据溯源领域的文献后，可以发现国际上的数据溯源

研究起步早且成果颇丰，已形成一套完善的研究框架，理论基础也较为成熟。而国内的相关研究则相对滞后，成果较少，研究体系仍在逐步构建中，且更多聚焦于实证分析。在区块链技术支持下，以确保数据的可靠性和高质量为目标的数据溯源研究仍处于初级阶段[21]。而现有的标注法、反向查询和数字水印等溯源计算方法，虽然在技术和理论上不断创新，但关于溯源方法的实现机制本身的研究在现有文献中报道较少。

在本研究中，我们探索了一种全新的内核驱动用户指纹脚本的溯源算法底层实现机制-动态指纹溯源架构，此架构与 Scruf 框架[34]都为文件系统的可扩展性和用户自定义功能做出了贡献。Scruf 的方法借鉴了 FUSE[35]的思想，将对内核的文件操作外放到用户空间脚本，但其缺少对多播消息的支持，可扩展性受限。在此基础上，我们进一步优化和拓展了 Scruf 的方法，利用 Netlink 支持异步、全双工和多播消息传递，结合内核的安全性和稳定性，实现了动态的个性化溯源功能。相比于 Scruf，我们的架构更注重文件的安全性和可追溯性，并提供了更高的灵活性和可扩展性。这一架构特别适用于需要高度安全和可追溯性的场景，如在多负载文档共享和离线文档追溯系统中。这样的设计确保了在保护敏感信息的同时，也能满足个性化的安全需求。

5 结论

本文提出了一种新型的基于脚本的动态指纹溯源架构，旨在提高敏感离线文档溯源技术的安全性、灵活性和可扩展性。该架构利用 Linux 内核扩展和 Netlink 机制激活溯源脚本，在文档传输前为文档动态添加指纹。这种方法支持异步和同步模式，适应多负载环境，并通过实验验证了其有效性，为数据追踪系统的发展开辟了新路径。适用于政府、企业和档案馆等机构。

然而，在涉及敏感文档的溯源技术中，保障安全性和隐私也是至关重要的。因此，需要对敏感文档进行加密处理，以防止未经授权的访问，并实施严格的访问控制，确保只有授权用户才能访问，从而在保护个人信息和敏感数据的同时，有效追踪和溯源文档。

参考文献

[1] Sohn J, Choi B, Yoon S W, et al. Capacity of clustered distributed storage[J]. IEEE Transactions on Information Theory, 2018, 65(1): 81-107.

[2] Yang CT, Kristiani E, Wang YT, et al. On construction of a network log management system using ELK Stack with Ceph[J]. The Journal of Supercomputing, 2020, 76: 6344-6360.

[3] LeFevre J, Maltzahn C. SkyhookDM: Data processing in Ceph with programmable storage[J]. USENIX login;, 2020, 45(2).

[4] Kai Z. Research on network data storage Technology based on Autonomous Controllable system[C]//2021 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI). IEEE, 2021: 183-186.

[5] Chukry S,Sbeyti H. Security enhancement in storage area network[C]//2019 7th International Symposium on Digital Forensics and Security (ISDFS). IEEE, 2019: 1-5.

[6] Sasongko H, Hadiwandra TY. Cloud-Based NAS (Network Attached Storage) Analysis as an Infrastructure as A Service (IAAS) Using Open Source NAS4FREE and Owncloud[J]. IT Journal Research and Development, 2022, 6(2): 83-97.

[7] Macedo R, Paulo J, Pereira J, et al. A survey and classification of software-defined storage systems[J]. ACM Computing Surveys (CSUR), 2020, 53(3): 1-38.

[8] 王芳,赵洪,马嘉悦,等.数据科学视角下数据溯源研究与实践进展[J].中国图书馆学报,2019,45(5):79-100.
Wang F, Zhao H, Ma JY, et al. Research and Practice Progress on Data Provenance from the Perspective of Data Science. Journal of the Library Science in China, 2019, 45(5): 79-100.

[9] Zafar F, Khan A, Suhail S, et al. Trustworthy data: A survey, taxonomy and future trends of secure provenance schemes[J]. Journal of network and computer applications, 2017, 94: 50-68.

[10] Pérez B, Rubio J, Sáenz-Adán C. A systematic review of provenance systems[J]. Knowledge and Information Systems, 2018, 57: 495-543.

[11] 明华, 张勇, 符小辉. 数据溯源技术综述[J]. 小型微型计算机系统, 2012, 33(9): 1917-1923.
Ming H, Zhang Y, Fu XH. A Review of Data Provenance Technology. Journal of Small and Micro Computer Systems, 2012, 33(9): 1917-1923.

[12] Wang YR, Madnick SE. A polygen model for heterogeneous database systems: The source tagging perspective[J]. 1990.

[13] Lee T, Bressan S, Madnick SE. Source Attribution for Querying Against Semi-structured Documents[C]//Workshop on Web Information and Data Management. 1998: 33-39.

[14] Buneman P, Khanna S, Tan WC. Computing provenance and annotations for views[C]//Workshop Paper: Workshop on Data Derivation and Provenance (Oct.). Chicago IL. Available at: http://people. cs. uchicago. edu/~ yongzh/position papers. html. 2002.

[15] Buneman P, Khanna S, Tan WC. On propagation of deletions and annotations through views[C]//Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. 2002: 150-158.

[16] Deutch D, Gilad A, Moskovitch Y. Selective provenance for datalog programs using top-k queries[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1394-1405.

[17] Dai C. Theories and approach of data lineage tracing in data warehouse environment[J]. ChangSha, China, 2002.

[18] Alexe B, Chiticariu L, Tan WC. Spider: a schema mapping debugger[C]//Proceedings of the 32nd international conference on Very large data bases. 2006: 1179-1182.

[19] 徐滨, 翁年凤, 樊树海, 等. 面向大规模定制的制造业领域数据溯源模型研究[J]. 机床与液压, 2023, 51(8): 1-7.
Xu B, Weng NF, Fan SH, et al. Research on Data Provenance Model for the Manufacturing Industry in the Context of Large-Scale Customization. Machine Tool & Hydraulics, 2023, 51(8): 1-7.

[20] Fan H. Tracing data lineage using automed schema transformation pathways[C]//British National Conference on Databases. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002: 50-53.

[21] 王晓庆, 孙战伟, 吴军红, 等. 基于数据要素流通视角的数据溯源研究进展[J]. 数据分析与知识发现, 2022, 6(1): 43-54.
Wang XQ, Sun ZW, Wu JH, et al. Research Progress on Data Provenance from the Perspective of Data Element Circulation. Data Analysis and Knowledge Discovery, 2022, 6(1): 43-54.

[22] Woodruff A, Stonebraker M. Supporting fine-grained data lineage in a database visualization environment[C]//Proceedings 13th International Conference on Data Engineering. IEEE, 1997: 91-102.

[23] Buneman P, Khanna S, Tan WC. Data provenance: Some basic issues[C]//FST TCS 2000: Foundations of Software Technology and Theoretical Computer Science: 20th Conference New Delhi, India, December 13–15, 2000 Proceedings 20. Springer Berlin Heidelberg, 2000: 87-93.

[24] Embaby AA, Shalaby MAW, Elsayed KM. Digital Watermarking Properties, Classification and Techniques[J]. International Journal of Engineering and Advanced Technology (IJEAT), 2020, 9(3): 2742-2750.

[25] Agrawal R, Kiernan J. Watermarking relational databases[C]//VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. Morgan Kaufmann, 2002: 155-166.

[26] Kumar S, Singh BK, Yadav M. A recent survey on multimedia and database watermarking[J]. Multimedia Tools and Applications, 2020, 79(27): 20149-20197.

[27] Rani S, Halder R. Comparative analysis of relational database watermarking techniques: An empirical study[J]. IEEE Access, 2022, 10: 27970-27989.

[28] 李伟岸, 熊祥光, 夏道勋. 基于超混沌和 Slant 变换的鲁棒水印算法[J]. 计算机工程与科学, 2020, 42(5): 812-818.
Li WA, Xiong XG, Xia DX. Robust Watermarking Algorithm Based on Hyperchaos and Slant Transform. Computer Engineering and Science, 2020, 42(5): 812-818.

[29] Kadian P, Arora SM, Arora N. Robust digital watermarking techniques for copyright protection of digital data: A survey[J]. Wireless Personal Communications, 2021, 118: 3225-3249.

[30] Tan X, Zhou M, Fitzgerald B. Scaling open source communities: an empirical study of the linux kernel[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020: 1222-1234.

[31] Wan M, Li J, Yao J. Real-Time Self-defense Approach Based on Customized Netlink Connection for Industrial Linux-Based Devices[C]//Collaborative Computing: Networking, Applications and Worksharing: 16th EAI International Conference, CollaborateCom 2020, Shanghai, China, October 16–18, 2020, Proceedings, Part I 16. Springer International Publishing, 2021: 406-420.

[32] 中国科学院深圳先进技术研究院. 敏感文件在线添加水印的方法、装置、设备及存储介质：CN202310672087.6[P]. 2023-12-08.
Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. Method, Apparatus, Equipment, and Storage Medium for Online Watermarking of Sensitive Files: CN202310672087.6[P]. 2023-12-08.

[33] Al-Odat Z, Abbas A, Khan SU. Randomness analyses of the secure hash algorithms, SHA-1, SHA-2 and modified SHA[C]//2019 International Conference on Frontiers of Information Technology (FIT). IEEE, 2019: 316-3165.

[34] Macdonell A C. Trigger scripts for extensible file systems[D]. University of Alberta, 2002.

[35] Szeredi M. Filesystem in userspace[J]. http://fuse. sourceforge. net/, 2001.