Secure and Lightweight Deduplicated Storage via Shielded Deduplication-Before-Encryption

Zuoru Yang, Jingwei Li, and Patrick P. C. Lee

摘要

外包存储应满足大规模数据管理的保密性和存储效率。传统方法通常基于加密后重复数据删除(DaE) 将加密和重复数据删除相结合，它首先执行加密，然后对加密数据进行重复数据删除。我们认为 DaE 具有基本限制，导致安全重复数据删除系统在性能、存储节省和安全性方面存在各种缺陷。在本文中，我们研究了一种称为加密前重复数据删除 (DbE) 的未探索范式，它首先执行重复数据删除并仅加密非重复数据。DbE 的好处是可以减轻重复数据管理造成的性能和存储损失，但其重复数据删除过程不再受加密保护。为此，我们设计了 DEBE，一个基于 DbE 的屏蔽重复数据删除存储系统，通过 Intel SGX 保护重复数据删除。DEBE 建立在基于频率的重复数据删除的基础上，它首先删除空间受限的 SGX enclave 中频繁数据的重复项，然后删除 enclave 外的所有剩余重复项。实验表明，DEBE 优于最先进的 DaE 方法。

关键词：可信执行环境；数据去重；外包存储

1 引言

请在此部分对选题背景，选题依据以及选题意义进行描述。数据外包到公共云存储为面对爆炸性数据增长的低成本、大规模数据存储管理提供了一个合理的解决方案。为了防止数据隐私泄露[1]，客户需要端到端加密，以便他们的外包数据在存储在（不受信任的）公共云存储之前被加密。然而，传统的对称加密禁止跨用户重复数据删除（即从多个客户端删除重复数据），因为每个客户端使用不同的密钥加密自己的外包数据，这意味着来自多个客户端的加密输出也是不同的。

2 背景和动机

2.1 加密后重复数据删除的局限性

重复数据删除是现代存储中广泛部署的一种数据减少技术[2-5]。我们专注于基于块的重复数据删除，它在块的粒度上去除重复的数据。具体来说，重复数据删除的存储系统将输入的文件数据划分为几个块。它通过一个加密的哈希值（例如 SHA-256）来识别每个块，这个哈希值被称为块内容的指纹（假设不同块的指纹碰撞实际上是不可能的[6]）。它维护一个键值存储，称为指纹索引，以跟踪所有现有存储块的指纹，并只存储不重复的块。它还为每个文件存储了一个清单文件，称为文件配方，以跟踪存储中的所有文件块，用于文件重建。此外，它还可以进一步应用压缩技术，删除非重复块中的字节级重复内容，以节省更多存储空间[2,5,7]。

加密后重复数据删除（DaE）结合了重复数据删除和加密，以实现保密性和存储的节约。在 DaE 中，客户端对明文块进行本地加密，并将密文块上传到云端，然后云端对密文块执行重复数据删除。DaE 的一个流行的加密基元是消息锁定加密（MLE）[8]，它正式表明，块加密/解密的密钥来自每个块

的内容，因此，相同的明文块总是被加密成相同的密文块用于重复数据删除。MLE 的一个实例是收敛加密（CE），它根据每个块的指纹推导出每个块的密钥。

CE 容易受到离线暴力攻击，在这种攻击中，对手列举所有可能的明文块以得出其秘密密钥，试图使用每个密钥解密一个密码文本块，如果解密成功，则推断出明文块。DupLESS[9]通过服务器辅助的密钥管理来防御 CE 中的离线暴力攻击，它部署了一个密钥服务器，该服务器根据全局秘密（安全地由密钥服务器拥有）和块的指纹来生成每个块的密钥。此外，DupLESS 实现了基于遗忘伪随机函数（OPRF）的密钥生成[10]，以防止密钥服务器在密钥生成过程中了解块或密钥，并限制客户端的密钥生成请求，以防御在线暴力攻击，其中恶意客户端积极向密钥服务器发出不同明文块的密钥生成请求。

局限: DaE 是构建安全的重复数据删除存储系统的最先进范例。然而，我们认为 DaE 存在三个基本限制。

1. **高密钥管理开销**。DaE 每个块生成一个密钥，导致维护所有基于块的密钥的巨大开销。此外，每个客户端都需要通过自己的主密钥加密其基于块的密钥以进行保护。因此，密钥存储开销与块和客户端的数量成比例地增加，并且对于具有高内容冗余的工作负载（例如，备份[4]，实现服务器辅助密钥管理的 DupLESS[9] 在将块上传到云之前生成用于加密每个块的密钥，即使该块是重复的并且稍后通过重复数据删除删除。由于 DupLESS 在密钥生成中采用 OPRF 和速率限制（见上文），因此其密钥生成被证明是昂贵的[11]。简而言之，DaE 在密钥存储和密钥生成方面都会产生很高的密钥管理开销。
2. **与压缩不兼容**。在 DaE 中，云无法通过压缩进一步节省非重复加密块的额外存储空间，因为加密块具有高熵（几乎随机）的内容。虽然客户端可以在加密之前对明文块应用压缩并上传加密的压缩块，但这会泄露压缩块的长度并引入安全风险[12]。
3. **安全风险**。DupLESS 中的服务器辅助密钥管理使密钥服务器成为单一攻击点。如果对手破坏了密钥服务器并可以访问全局机密，则它可以像 CE 一样通过离线暴力攻击推断出块的秘密密钥。另外，DaE 本质上是确定性的，实现了明文块和密文块之间的一对一映射。对手可以启动频率分析，从去重存储中密文块的频率分布推断出原始明文块[13]。

2.2 先去重再加密

鉴于 DaE 的局限性，我们研究了一种未探索的范例，即加密前重复数据删除 (DbE)，用于安全的重复数据删除存储。其思想是先对明文块进行去重，去除重复项，然后将不重复的明文块加密成密文块存储。

与 DaE 相比，DbE 自然具有多项优势。首先，由于首先应用重复数据删除，DbE 可以像在传统对称加密中一样使用与内容无关的密钥加密每个非重复的明文块，而不会影响重复数据删除。这避免了生成和存储每个块的内容派生密钥，并减少了密钥管理开销（局限 1 解决）。其次，DbE 可以在重复数据删除后对非重复明文块应用压缩以进一步节省存储空间，然后加密压缩的非重复明文块（局限 2 解决）。最后，由于 DbE 可以使用与内容无关的密钥执行加密，因此它不再像 DupLESS 中那样需要密钥服务器来生成每个块的密钥。这消除了密钥服务器中的单点攻击（局限 3 解决）。

然而，DbE 的主要挑战是决定是客户端还是云应该执行不再受加密保护的重复数据删除。我们考虑三种情况：

1. 每个客户端为其自己的明文块维护一个本地指纹索引。它对不重复的明文块进行加密，并将密文块上传到云端。但是，这种方法禁止跨用户重复数据删除。

2. 云维护一个全局指纹索引来跟踪所有客户端的存储块。每个客户端首先将自己明文块的指纹提交到云端，查询是否可以去重。对云端识别出的不重复的明文块进行加密，并将密文块上传至云端。这种方法也称为基于源的重复数据删除[14]，容易受到侧信道攻击，因为任何恶意客户端都可以通过查询目标块是否可以进行重复数据删除来推断是否已经存储了某个目标块。

3. 每个客户端将所有块上传到云端。云根据其全局指纹索引执行重复数据删除，该索引跟踪所有客户端的存储块，然后加密非重复块。这种方法，也称为基于目标的重复数据删除[14]，对客户端隐藏了重复数据删除模式，并且可以安全地抵御边信道攻击。然而，每个客户端都不可避免地会将其明文块暴露给云端。

因此，DbE 在文献中仍未得到探索，而现有研究主要集中在用于安全去重存储的 DaE。

2.3 Intel SGX

在这项工作中，我们通过基于目标的重复数据删除实现了 DbE，并展示了我们如何通过屏蔽执行来保护 DbE。我们使用英特尔 SGX 实施屏蔽执行。由于我们的主要要求是为不受信任的云中的数据处理提供安全的内存区域，我们推测我们的设计可以支持其他屏蔽执行技术（例如 ARM TrustZone 和 AMD SEV）。

SGX 基础知识。SGX 是一组扩展指令，用于 Intel CPU 在称为 enclave 页面缓存 (EPC) 的加密且受完整性保护的内存区域中实现称为 enclave 的屏蔽执行环境。它通过硬件保护确保 enclave 内内容的机密性和完整性。它提供了两个接口来与 enclave 外部的不受信任的应用程序交互：（i）enclave 调用（ECalls），它允许应用程序安全地访问 in-enclave 内容，以及（ii）外部调用（OCalls），它允许 inclave 代码在应用程序中发出函数调用。

挑战。由于 enclave 的资源限制，在 SGX 中实现 DbE 并非易事。首先，EPC 大小有限（例如，最多 128 MiB ）。当 enclave 的内存使用量超过 EPC 大小时，它会将未使用的内存页面加密并逐出到未受保护的主内存，并在将被逐出的页面加载回 EPC 时解密并验证其完整性。这会导致昂贵的 EPC 分页开销。尽管最近的 SGX 设计支持高达 1 TiB 的大型 EPC，但由于失去了完整性树保护，它们提供的安全保证较弱。其次，ECall 和 OCal 都涉及昂贵的硬件操作（例如，刷新 TLB 条目），这会导致显着的上下文切换开销（例如，每次调用大约 8,000 个 CPU 周期）。

3 本文方法

3.1 本文方法概述

DEBE 的实际相关性。DEBE 专注于多租户重复数据删除，它在实践中得到广泛部署（例如，Dropbox、Druva 、Cohesity 和 Memopal ），并且被证明比单租户重复数据删除实现更高的存储节省从多个客户端中删除重复数据。现有的 DaE 方法也专为多租户重复数据删除而设计，而 DEBE 解决了 DaE 的局限性。尽管 DEBE 因受保护的执行而产生成本（例如，来自受信任的第三方的 enclave 验证费用），但它对 DaE 方法的改进（在性能、存储节省和稳健性方面；参见）为云存储提供商提供了激励使用 DEBE 为客户提供安全、高性价比的云存储服务。在 enclave 内保留完整的指纹索引（或简称为完整

索引）可以跟踪所有指纹存储非重复块，但由于有限的 EPC 大小）而导致显着的 EPC 分页开销。或者，在 enclave 之外管理完整索引可以节省 EPC 的使用，但会因为查询完整索引的过多 OCalls 而导致昂贵的上下文切换。

我们提出基于频率的重复数据删除，它执行受 enclave 资源限制的安全重复数据删除。我们的见解是，块的频率（即重复的数量）在实际备份工作负载中存在高度偏差，因此一小部分块可能会导致大部分重复。为了证明这一点，我们对五个真实世界的备份跟踪进行了跟踪分析。我们测量输入块子集的重复率，定义为从块子集派生的重复块的总大小与整个跟踪中重复块的总大小之间的比率（请注意，一个块被称为一个 duplicate chunk，如果它的相同副本已经存储并且可以通过重复数据删除删除）。图 1 显示了重复率与频繁块的最高百分比（按频率降序排列）。例如，在 VM 跟踪中，前 5% 的频繁块导致了大约 97% 的重复率。这意味着如果我们维护一个小的指纹索引来跟踪前 5% 的频繁块，我们可以删除大约 97% 的重复数据并实现高存储节省。

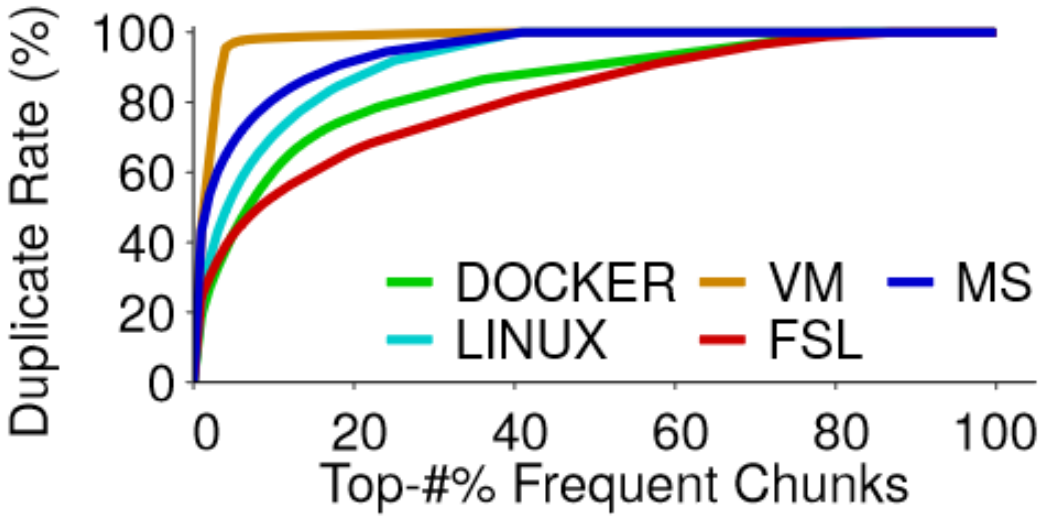


图 1: 重复率与五个真实世界轨迹中频繁块的最高百分比

基于频率的重复数据删除的思想是根据块频率分离重复数据删除过程。它在 enclave 内管理一个小的指纹索引，以从最频繁的块中删除重复项。它还在 enclave 外部维护完整索引，以删除频率较低的块的剩余重复项。基于频率的重复数据删除解决了性能和安全问题。为了性能，它只为 enclave 内最频繁的块管理一个小的指纹索引，以删除大部分重复块。因此，它减轻了 EPC 分页开销。它还减少了上下文切换开销，因为它仅通过 OCalls 查询 enclave 外部的完整索引，以获得频率较低的块的有限部分。为了安全起见，由于最频繁的块更容易受到频率分析的影响，我们仅通过 enclave 内处理删除了最频繁块的重复项。因此，云中的对手无法轻易了解最频繁块的频率，因此也无法了解信息频率分析引起的泄漏是有限的。

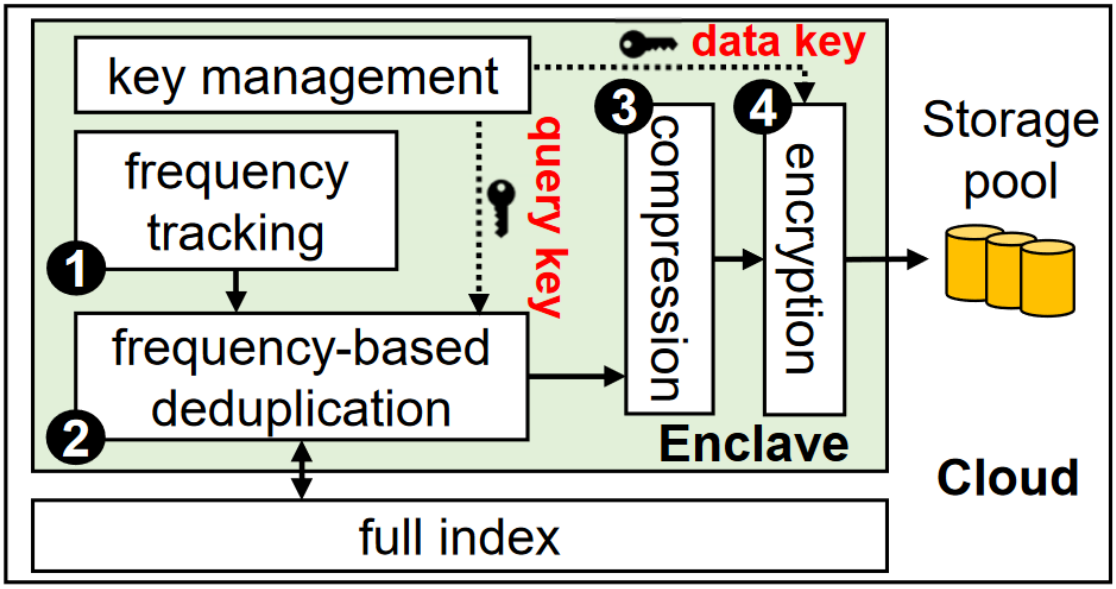


图 2: 方法示意图

enclave 架构和设计路线图。图 2 描述了 DEBE 中 enclave 的架构。最初，enclave 使用一组密钥进行引导，并与每个客户端建立安全数据通道。然后 enclave 跟踪从客户端的数据通道接收到的每个明文块的频率。基于块频率，基于频率的重复数据删除删除了最频繁的明文块的重复项，并与 enclave 外部的完整索引交互以删除剩余的不太频繁的明文块的重复项。enclave 对非重复的明文块进行压缩，并对压缩后的明文块进行加密。最后，enclave 将密文块存储在存储池中。

3.2 密钥管理

enclave 维护一组密钥，用于在去重和压缩之后安全存储块以及与客户端的安全通信。数据键和查询键。enclave 维护两个长期密钥，它们在 enclave 的整个生命周期（即 DEBE 运行的整个持续时间）中保持有效：（i）用于加密和解密安全存储中压缩的非重复明文块的数据密钥，以及 (ii) 在 enclave 外查询完整索引时用于保护明文块信息的查询密钥。当 enclave 被引导时，它通过片上硬件随机数生成器（即 sgx read rand ）初始化数据密钥和查询密钥。这两个密钥都可以通过现有方法（例如，密钥回归）定期更新，而不会影响重复数据删除，因为 DEBE 在加密之前执行重复数据删除。

会话密钥。回想一下，每个客户端都维护一个与 enclave 的数据通道以进行安全数据通信（同时维护一个与云的控制通道以安全地发布存储操作）。每个数据通道都使用短期会话密钥来保护其通信，该密钥对单个通信会话仍然有效。它通过控制通道使用 Diffie-Hellman 密钥交换为数据通道建立会话密钥。会话密钥在客户端通信会话期间保存在 enclave 中，会话完成后将被释放（控制和数据通道也会被释放）。

每个客户端的主密钥。enclave 要求每个客户端通过数据通道为每个存储请求提交一个主密钥。它使用主密钥来保护客户端文件的文件配方，并强制执行客户端对文件的所有权。与会话密钥类似，enclave 只为单个通信会话保留客户端的主密钥，并会在会话结束时销毁主密钥，因此主密钥的存储开销也是有限的。

3.3 频率跟踪

enclave 需要跟踪明文块的频率，以识别最频繁和不太频繁的块，以进行基于频率的重复数据删除。为了减少 EPC 的使用，enclave 使用 Count-Min Sketch（CM-Sketch）[15] 来跟踪具有固定大小空间

和小错误的每个块的近似频率。

CM-Sketch 是一个二维数组，每行有 r 行 w 个计数器。这里的一个关键设计是限制将明文块映射到计数器的计算开销。为此，我们的见解是块指纹被计算为加密哈希（例如，在我们的例子中为 SHA-256），因此我们可以将块指纹视为随机输入值并将其直接映射到计数器而不会损害 CM-Sketch 的准确性。具体来说，对于每个明文块 M，enclave 将 M 的指纹划分为 r 块。它用第 i 个块对 w 取模找到第 i 行（$1 \le i \le r$）中从 0 到 $w-1$ 索引的 w 个计数器之一，并将每个映射的计数器递增 1；这与传统的 CM-Sketch 形成对比，后者使用成对独立的哈希函数将输入映射到不同行的计数器，因此具有额外的计算开销。为了估计块的频率，enclave 使用块的 r 个映射计数器的最小值。默认情况下，我们配置 r = 4、w = 256 K 和 4 字节计数器，因此 CM-Sketch 的整体 EPC 使用量仅为 4 MiB。

3.4 基于频率的重复数据删除

我们介绍了基于频率的重复数据删除的设计，它根据估计的频率分两个阶段删除所有重复的明文块。

第一阶段重复数据删除。enclave 维护一个小的指纹索引，称为 top-k 索引，以对 k 个最频繁的明文块进行重复数据删除。我们将 top-k 索引实现为最小堆和哈希表的组合，如图 4 所示。最小堆区分 top-k 频繁和不太频繁的明文块。它跟踪明文块的 top-k 估计频率，使得根堆条目对应于当前 top-k 估计频率中频率最小的明文块。最小堆中的每个堆条目都存储一个指向哈希表中哈希条目的指针。另一方面，哈希表用于重复检测，如传统的重复数据删除。每个散列条目存储从块指纹到元素元组的映射：（i）指向堆条目的指针（即，堆条目和散列条目相互引用），(iis) 块的估计频率，(iii) 块地址（包括容器 ID 和容器内的内部偏移量），以及 (iv) 压缩块大小（即压缩后块的大小）.

给定一个明文块，为了执行第一阶段重复数据删除，enclave 将从 CM-Sketch 和块指纹中获得的明文块的估计频率作为输入。它首先检查最小堆的根堆条目。如果输入频率小于最小堆的最小频率（即，该块是频率较低的块），则 enclave 跳过查询该块的哈希表并继续进行第二阶段重复数据删除；否则（即，该块是 top-k-frequent 块），enclave 使用输入指纹来查找哈希表。我们有以下两种情况：

1. 如果在哈希表中找到指纹（即块重复），enclave 更新哈希表中的频率并将块地址和压缩块大小添加到文件配方。由于更新频率，它还会根据指向最小堆中堆条目的指针调整最小堆。
2. 如果在哈希表中没有找到指纹（即，该块是一个新的 top-k-frequent 块），enclave 在哈希表中创建一个新的哈希条目并插入一个包含指向新哈希的指针的新堆条目进入最小堆。如果 min-heap 已经存储了 k 个 heap entry，则 enclave 删除 minheap 的当前 root heap entry（频率最小），同时通过 root heap entry 中存储的指针删除 hash table 中对应的 hash entry。由于该块可能已经被存储（但由于其频率较低而未被 top-k 索引跟踪），因此 enclave 还对该块进行第二阶段去重并根据第二阶段重复数据删除的结果。

第二阶段重复数据删除。第二阶段重复数据删除是在第一阶段重复数据删除未删除的明文块上执行的，包括频率较低的块和指纹为 top-k 索引新的新的新的 top-k-frequency 块。由于 EPC 大小有限，DEBE 管理 enclave 之外的完整索引。我们将全索引实现为哈希表，其中每个条目存储从明文块的加密指纹到加密块信息的映射（即块地址和压缩块大小，这两者都由查询加密密文块的密钥）。我们加密指纹和块信息的基本原理是防止云中的任何对手推断明文块，因为完整索引不受 enclave 保护。

给定一个明文块，为了执行第二阶段重复数据删除，enclave 使用查询密钥确定性地加密明文块的指纹（未被第一阶段重复数据删除删除），以便来自不同客户端的重复明文块始终映射到重复的加密指纹以进行跨用户重复数据删除。然后它通过 OCall 查询基于加密指纹的完整索引。如果在全索引中找到加密的指纹，则 OCall 返回加密的块信息，该信息将由 enclave 内的查询密钥解密。然后 enclave 会将地址和压缩块大小更新到文件配方中。否则，如果加密指纹是全索引的新指纹，则 enclave 将此块标识为非重复块，为块分配地址，压缩块以获得其压缩块大小，并对地址和压缩块进行加密大小与查询键。然后它更新加密指纹和完整索引中相应的加密块信息。请注意，由于 OCalls 引起的上下文切换开销是有限的，因为大部分重复项预计已被第一阶段重复数据删除删除。

remark。用于重复数据删除的传统高效索引技术，例如基于相似性[16] 和基于位置的重复数据删除[17] 方法，也可以通过仅将完整索引的一部分加载到 enclave 来解决有限的 EPC 大小。然而，它们只能实现近乎精确的重复数据删除（即无法删除某些重复项），而 DEBE 可以实现精确的重复数据删除。

3.5 存储管理

容器存储。DEBE 管理固定大小容器中的物理块以降低磁盘 I/O 成本。enclave 对去重后的非重复明文块进行压缩，并用数据密钥将压缩后的非重复明文块加密成密文块。它将每个密文块连同初始化向量 (IV) 写入 enclave 内的内存容器中。当内存容器已满时，enclave 会将其发送到云中的持久存储中。请注意，DEBE 仅在去重后为每个非重复密文块存储一个 IV（在 AES-256 中大小为 16 字节），而 DaE 方法在去重前为每个密文块存储加密密钥（在 AES-256 中大小为 32 字节）并且当存在许多重复块时会产生大量的密钥存储开销。

此外，enclave 为每个上传的文件创建和管理文件配方。文件配方中的每个条目都保留文件的每个密文块的块地址和压缩块大小。需要注意的是，当 enclave 向文件 recipe 中添加条目时，不需要对 duplicate chunk 进行压缩来获取其压缩后的 chunk size，因为压缩后的 chunk size 已经存储在 top-k index 和 full index 中。为了保证文件的所有权，enclave 通过客户端的主密钥对文件配方进行加密，并将加密后的文件配方存储为普通文件。由于 enclave 将容器（每个容器包含多个密文块）视为基本 I/O 单元，并且块大小存储在文件配方中（由每个用户的主密钥保护），DEBE 保留了压缩的安全性，因为它避免泄漏压缩块的长度。

下载。要下载文件，客户端会通过安全数据通道向 enclave 发出下载请求及其主密钥。enclave 检索文件配方并使用给定的主密钥解密文件配方。然后它解析解密的文件配方以获得块地址和压缩块大小。为了恢复所有块，enclave 将所请求块的容器 ID 公开给云以通过 OCall 执行 I/O。一旦云将相应的容器提取到未受保护的主内存中，enclave 就会根据其内部偏移量访问密文块，并通过数据密钥对密文块进行解密。最后，它解压缩明文块并将其通过数据通道发送给客户端。

4 复现细节

4.1 与已有开源代码对比

已有的开源代码代码逻辑清晰，各方面处理都很好。作为一个系统原型，基本满足的点都已经满足。但是目前来看复现该论文的主要的困难有三个点：

1. 环境的搭建十分复杂，SDK 与 PSW 每一次的编译时间需要 30 分钟，并且安装具有一定的

依赖关系

2. 项目依赖着 intel 的 sgxssl 库，不同版本 sgxssl 依赖着不同版本的 openssl，这样有很高的耦合性

3. 不同的系统有着不一样的配置规则

为了能让其他同学能快速运行此项目，我做的工作是实现一个松耦合的运行环境，能让项目快速启动并摆脱对不同系统之间强依赖关系。并且对于加密前去重应用来说，容器化部署可以帮助提高效率和安全性:

1. 提高效率：通过将加密前去重应用程序打包进容器，可以更快速地部署和管理应用程序，从而提高效率。

2. 提高安全性：将加密前去重应用程序打包进容器，可以防止应用程序与其他应用程序相互干扰，从而降低漏洞和攻击的风险。

3. 可移植性：通过容器化部署，加密前去重应用程序可以在不同的环境中运行，这有助于提高应用程序的可移植性。

4.2 实验环境搭建

首先需要安装 Docker 环境，在安装完 Docker 环境后需要构建镜像。有两种常见的方式来构建 Docker 镜像：

1. 通过 Dockerfile 构建：Dockerfile 是一个文本文件，其中包含了构建 Docker 镜像所需的说明。您可以使用 Docker 命令行工具的 docker build 命令来构建镜像。

2. 通过复制文件到容器中构建：这种方式通常比较适用于快速构建测试用的镜像。您可以使用 Docker 命令行工具的 docker cp 命令将文件复制到运行中的容器中，然后使用 docker commit 命令将其保存为新的镜像。

还有其他一些更高级的方式来构建 Docker 镜像，例如通过使用 Docker 守护进程的 API 来构建镜像，或者使用一些第三方工具，例如 Packer 来构建镜像。

本次复现用到了上面两种常见方法。AESM (Intel SGX Attestation and Seal Service Manager) 是 SGX 架构的一个组件，它负责处理身份验证和数据加密。AESM 为应用程序提供了一种方法来验证 SGX 保护环境的真实性，并为应用程序加密和存储数据。

SGX 和 AESM 可用于在不可信的环境中保护应用程序和数据的隐私和安全性，如云计算环境或公共云服务。SGX 允许应用程序在保护环境中运行，并使用 AESM 进行身份验证和数据加密。因此 AESM 的容器构建过程应是公开可见，所以在构建 AESM 镜像时，我采用了 Dockfile 方式进行。

对于应用容器来说，配置过程相对复杂，用 Dockfile 编写会使得不同环境下因为网络等问题，镜像构建过程会很长，甚至失败，因此我采用把文件复制进容器中构建。具体来说，把编译好 sgx-ssl 包复制进行容器中，并在容器中安装好所有所需的环境依赖。应用代码也复制进容器中，后续就可以直接启动。

AESM 容器与应用容器通过一个共享的 docker volume 来实现数据交互。Docker volume 是 Docker 中用于存储和管理数据的功能。它允许 Docker 容器之间共享数据，也可以将数据持久化保存在主机

文件系统之外。在本次复现中，两个容器共享的是一个位于/var/run/的 socket 文件 aesmd 。这个文件用于存储 AESM 的运行时数据。

4.3 使用说明

准备工作：安装好 docker 与 sgx driver

启动步骤：

1. 拉取两个 docker 镜像
2. 创建 docker volume
3. 先启动 AESM 容器, 然后再启动应用容器
4. 进入到应用容器指定目录运行容器

应用分为 client 端与 server 端，并可以通过 config.json 来配置存储的位置，块大小，服务器提供服务的 IP 与端口等。用户可以根据自己的情况修改来运行。

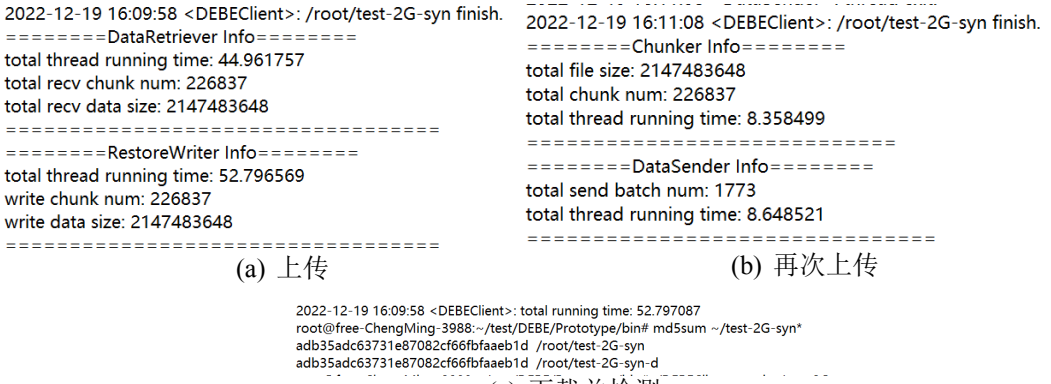
4.4 创新点

整个复现工作的创新点在于容器化启动 SGX。容器化的启动 SGX 创新点包括：

1. 安全隔离：在容器化的 SGX 应用程序中，每个应用程序都会运行在其自己的保护环境中，从而提供了更好的安全隔离。这意味着每个 SGX 应用程序可以独立地运行，并且不会受到其他应用程序的干扰。
2. 资源隔离：容器化的 SGX 应用程序可以被分配独立的 CPU 核心和内存，这有助于提供更好的资源隔离。这意味着 SGX 应用程序可以更好地管理自己的资源，并且不会受到其他应用程序的干扰。
3. 灵活部署：容器化的 SGX 应用程序可以方便地在不同的机器之间部署和移动。这使得 SGX 应用程序的部署变得更加灵活，可以根据需要调整应用程序的部署位置。
4. 更好的可移植性：容器化的 SGX 应用程序可以在不同的操作系统之间移植，因为它是一个轻量级的软件包。这使得 SGX 应用程序可以在不同的硬件平台和操作系统之间运行，并且可以方便地在不同的环境中部署和管理。
5. 更好的可观察性：容器化的 SGX 应用程序可以方便地收集和监控应用程序的运行状态和行为。这可以通过容器化平台提供的监控和日志功能实现。
6. 更好的可管理性：容器化的 SGX 应用程序可以方便地通过容器化平台的管理工具进行管理。例如，可以使用容器化平台的命令行工具或 API 进行 SGX 应用程序的部署、启动、停止和删除。

5 实验结果分析

随机生成 2GB 文件做上传，下载再上传处理。图 3(a) 是对 2GB 文件做上传测试，总用时 52s。对于服务端来说每一个块都是新的，所以需要花较多时间处理。图 3(b)是对该文件再次上传的检测，可以看到总用时只用 8s。对于服务端来说每一个块都是重复的。因此不需要再做存储，可以看到对于 2GB 文件，去重处理并记录相关内容需要时间还是较短的。图 3(c)是对文件下载之后做 md5 算法的完整性检查。



(a) 上传 (b) 再次上传

(c) 下载并检测

图 3: 实验结果

该实验性能对比论文中的结果是一致的，因此容器化部署并不会影响到系统的性能。

6 总结与展望

DEBE 实现了一种未开发的范例，即加密前重复数据删除 (DbE)，用于安全的重复数据删除存储。它建立在 SGX 之上，并应用基于频率的重复数据删除来管理 enclave 中最频繁块的小型指纹索引。我们表明 DEBE 在性能、存储节省和安全性方面优于传统的加密后重复数据删除 (DaE) 方法。本复现方案把 DEBE 方案改进为容器化启动，实现了系统的便捷部署。但是方案中的一些核心问题还未能完全解决，如方案无法抵御侧信道攻击，但是引入了容器化之后可能从容器加密化角度探索解决方案。

对于 SGX 来说，SGX 是一项提供应用程序安全性和隐私性保护的有用技术。然而，SGX 也有一些局限性。例如，SGX 只能在英特尔处理器上使用，并且也存在安全漏洞，可能会被攻击者利用。因此，在使用 SGX 时需要谨慎，并对其进行适当的安全评估。

参考文献

[1] 佘嘉博. Research on Runge Phenomenon[J]. Advances in Applied Mathematics, 2019, 08(08): 1500-1510.

[2] EL-SHIMI A, KALACH R, KUMAR A, et al. Primary Data {Deduplication—Large} Scale Study and System Design[C]//2012 USENIX Annual Technical Conference (USENIX ATC 12). 2012: 285-296.

[3] MEYER D T, BOLOSKY W J. A study of practical deduplication[J]. ACM Transactions on Storage (ToS), 2012, 7(4): 1-20.

[4] WALLACE G, DOUGLIS F, QIAN H, et al. Characteristics of backup workloads in production systems.[C]//FAST: vol. 12. 2012: 4-4.

[5] ZHU B, LI K, PATTERSON R H. Avoiding the disk bottleneck in the data domain deduplication file system.[C]//Fast: vol. 8. 2008: 269-282.

[6] BLACK J. Compare-by-Hash: A Reasoned Analysis.[C]//USENIX Annual Technical Conference, General Track. 2006: 85-90.

[7] SRINIVASAN K, BISSON T, GOODSON G R, et al. iDedup: latency-aware, inline data deduplication for primary storage.[C]//Fast: vol. 12. 2012: 1-14.

[8] BELLARE M, KEELVEEDHI S, RISTENPART T. Message-locked encryption and secure deduplication[C]//Annual international conference on the theory and applications of cryptographic techniques. 2013: 296-312.

[9] BELLARE M, KEELVEEDHI S, RISTENPART T. DupLESS: Server-aided encryption for deduplicated storage[J]. Cryptology ePrint Archive, 2013.

[10] NAOR M, REINGOLD O. Number-theoretic constructions of efficient pseudo-random functions[J]. Journal of the ACM (JACM), 2004, 51(2): 231-262.

[11] REN Y, LI J, YANG Z, et al. Accelerating Encrypted Deduplication via {SGX}[C]//2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021: 957-971.

[12] CHEN D, FACTOR M, HARNIK D, et al. Length preserving compression: marrying encryption with compression[C]//Proceedings of the 14th ACM International Conference on Systems and Storage. 2021: 1-12.

[13] LI J, QIN C, LEE P P, et al. Information leakage in encrypted deduplication via frequency analysis[C]//2017 47th Annual IEEE/IFIP international conference on dependable systems and networks (DSN). 2017: 1-12.

[14] HARNIK D, PINKAS B, SHULMAN-PELEG A. Side channels in cloud services: Deduplication in cloud storage[J]. IEEE Security & Privacy, 2010, 8(6): 40-47.

[15] CORMODE G, MUTHUKRISHNAN S. An improved data stream summary: the count-min sketch and its applications[J]. Journal of Algorithms, 2005, 55(1): 58-75.

[16] BHAGWAT D, ESHGHI K, LONG D D, et al. Extreme binning: Scalable, parallel deduplication for chunk-based file backup[C]//2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems. 2009: 1-9.

[17] LILLIBRIDGE M, ESHGHI K, BHAGWAT D, et al. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality.[C]//Fast: vol. 9. 2009: 111-123.