编号：CACR2022HQYQ66

作品类别：☑软件设计 ☐硬件制作 ☐工程实践 ☐密码应用技术

2022 年第七届全国密码技术竞赛作品设计报告

题目：基于国密算法和 SGX 的密文数据去重系统

2020 年 10 月 28 日

中国密码学会

基本信息表

编号：CACR2022HQYQ66

作品题目：基于国密算法和 SGX 的密文数据去重系统

作品类别：☑软件设计　☐硬件制作　☐工程实践　☐密码技术应用

作品内容摘要：

在云备份系统中，数据去重技术可大幅降低存储空间。但针对密文的数据去重系统的设计却存在不小的挑战。如何在实现高效数据存储的同时为用户数据提供足够的安全保护是目前学术界和工业界面临的关键问题。本题目将结合国产密码算法和Intel SGX技术，通过软硬件结合的方式设计一个安全高效的密文数据去重系统，最大程度地检测并消除系统中的冗余计算和冗余数据存储，并基于备份数据的时间局部性设计了基于时段的去重模式，有效提高了系统的执行效率。

关键词（五个）： 云存储，数据安全，加密去重，国密算法，Intel SGX

目录

1 作品概述

1.1 研究背景

1.1.1 云存储

近年来，随着信息化时代的推进，数据呈现爆发式增长，2019 年，联合国发布的《数字经济报告》强调，数字经济正在成为经济发展的重要推动力。据不完全统计，数字经济占世界 GDP 的 4.5%至 15.5%。如图 1，根据 IDC 分布的《数据时代 2025》预测，全球数据量将从 2018 年的 33ZB 增至 2025 年的 175ZB，增长超过 5 倍。

图 1：数据来源 IDC《数据时代 2025》

中国平均增速快于全球 3%，预计到 2025 年将增至 48.6ZB，占全球数据圈的比例由 23.4%提升至 27.8%。其中，中国企业级数据量将从 2015 年占中国数据量的 49%增长到 2025 年的 69%。如图 2，根据中国信息通信研究院预测，到 2025 年中国数字经济规模将达到 60 万亿元，数字经济将成为经济高质量发展的新动能。

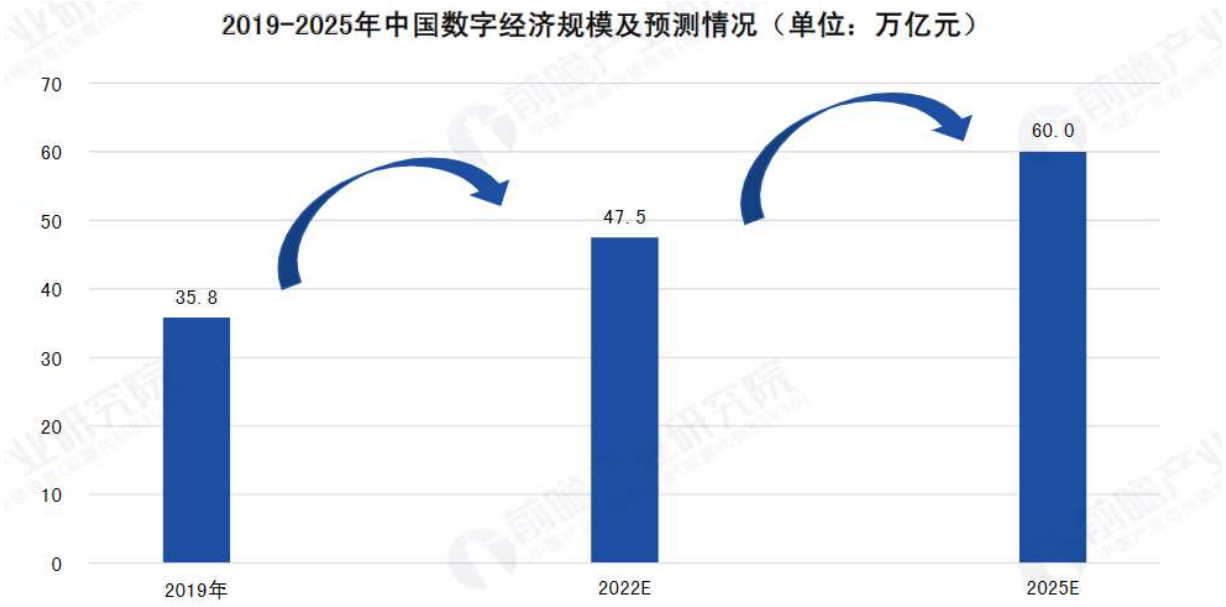

图 2：数据来源中国通信研究院

由于海量数据生成，数据存储压力的持续增长推动了整个存储市场的快速发展。通过提供数据存储和管理，云存储系统成为新时代不可或缺的一部分。云存储本质上是一个云计算系统，允许用户在互联网上存储和共享数据，即把数据存放在通常由第三方托管的多台虚拟服务器，而非专属服务器上。我国云存储行业的发展可以追溯到 2007 年，云计算、云存储的概念在国内开始出现。2011 年，云计算、云存储的概念落地；2012 年，国家将云计算列为重点发展的战略性新兴产业，各大互联网企业纷纷推出自己的云存储应用。从 2015 年到 2020 年，我国云存储市场的规模由 115 亿元上升到接近 400 亿元。如图 3 是来自 CCW 关于我国云存储市场的调查结果，其市场需求持续上涨。

图 3：来源 CCW Research

而在云存储不断发展的过程中，一直存在着两个重要的问题：存储数据的安全问题，以及数据存储的效率问题。如何设计好的数据加密方案以及高效的数据去重方案，成为了云存储发展的关键问题。

1.1.2 数据安全

面对纷繁复杂的云存储，数据的安全显得尤为重要。目前，政府、企业和个人用户正在积极将其数据迁移到云中。如此大量的数据可以创造大量的财富。但是，这会增加可能的风险，例如未经授权的访问，数据泄露，敏感信息泄露和隐私泄露。如图 4 是来自腾讯安全团队云威胁报告，可见公有云恶意木马数量十分庞大。当今云存储的发展也还存在着很多的问题，例如配置策略错误，导致未授权人员可以访问数据；数据治理不足，用户数据的机密性和隐私性无法得到保证。除此之外，云存储还存在一系列其它问题，导致数据的安全性无法得到根本保障。

图 4：来源腾讯安全团队云威胁报告

在过去的几个月中，云存储安全事件频发。微软安全响应中心在当地时间 2022 年 10 月 20 日发布公告，针对 19 日网络安全供应商 SOCRadar 通报的数据泄露事件的调查报告，微软承认了关键事实——即由于公有云服务器端点配置错误，可能导致未经身份认证的访问行为，继而泄漏微软和客户之间的某些业务交易数据以及客户的客人信息。安全公司 Safety Detectives 发现，中国初创公司 Socialarks（笨鸟社交）泄露了 400GB 数据。此次数据泄露是由于 ElasticSearch 数据库设置错误，泄露了总计 408GB，超过 3.18 亿条用户记录，涉及到 11651162 个 Instagram 用户、66117839 个领英用户和 81551567 个 Facebook 用户。

1.1.3 数据存储

数据的高效存储同样也占据重要的地位，正如前文所述，数据呈现爆发式的增长，如何经济高效地管理存储，已成为大数据时代海量存储系统中最具挑战性和最重要的任务之一。实现数据的高效存储可以大大降低开销，节约成本。服务商们想尽了各种办法来提高数据的存储效率，一种常见的技术是数据去重，Microsoft 和 EMC 进行的工作负载研究表明，其生产主存储系统和辅助存储系统中分别有大约 50% 和 85% 的数据是冗余的将重复的数据删除以减少存储数据的开销，或是记录数据在服务器中的存储时间，如果达到一定期限就将数据删除，这样便会减少存储数据的开销。通过识别文件内部和文件之间的常见数据块并仅存储一次，重复数据删除可以通过增加给定存储量的效用来节省成本。重复数据删除是一种有效的数据缩减方法，它消除了文件或子文件级别的冗余数据，并通过其加密安全的哈希签名（即抗碰撞指纹）识别重复的内容。

因此，如何将数据存储和数据安全结合成为了云存储的重中之重，目前比较主流的方向就是加密和数据去重结合起来，实现加密数据的重复数据删除。

1.2 相关知识

1.2.1 Intel SGX

随着云存储需求的提升，移动环境和云平台的安全对硬件和平台安全机制的需要更加迫切，基于硬件的可信执行环境必不可少。2013 年，Intel 推出 SGX(Software Guard Extensions)指令集扩展，旨在以硬件安全为强制性保障，不依赖于固件和软件的安全状态，为用户提供物理级的加密计算环境，通过一组新的指令集扩展与访问控制机制，实现不同程序间的隔离运行，保障用户关键代码和数据的机密性与完整性不受恶意软件的破坏。不同于其他安全技术，Intel SGX 的可信计算基(Trusted Computing Base，简称 TCB)仅包括硬件，避免了基于软件的 TCB 自身存在软件安全漏洞与威胁的缺陷，极大地提升了系统安全保障。

Intel SGX 可针对已知的硬件和软件攻击提供以下保护措施：

• 安全区内存不可从安全区外读写，无论当前的权限是何种级别，CPU 处于何种模式。

• 产品安全区不能通过软件或硬件调试器来调试。（可创建具有以下调试属性的安全区：该调试属性支持专用调试器，即Intel SGX 调试器像标准调试器那样对其内容进行查看。 此措施旨在为软件开发周期提供辅助。）

• 安全区环境不能通过传统函数调用、转移、注册操作或堆栈操作进入。 调用安全区函数的唯一途径是完成可执行多道保护验证程序的新指令。

•安全区内存采用具有回滚保护功能的行业标准加密算法进行加密。 访问内存或将 DRAM 模块连接至另一系统只会产生加密数据（见图5）。

• 内存加密秘钥会随着电源周期（例如，启动时或者从睡眠和休眠状态进行恢复时）随机更改，该密钥存储在 CPU 中且不可访问。

• 安全区中的隔离数据只能通过共享安全区的代码访问。

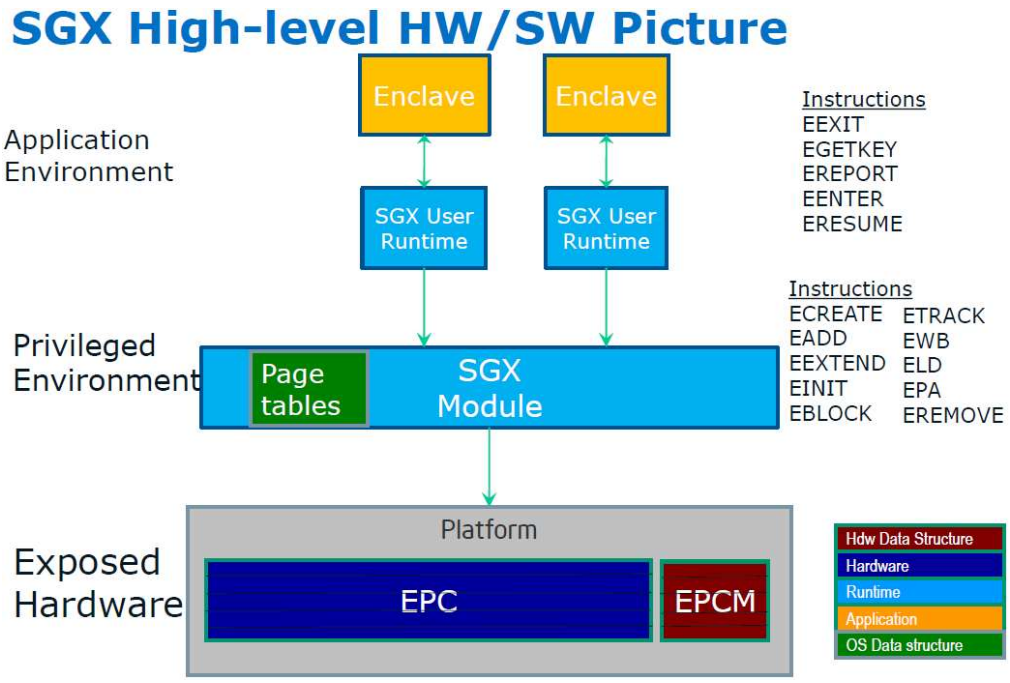

图 5：SGX 应用框架图

此外，Intel SGX 可保障运行时的可信执行环境，恶意代码无法访问与篡改其他程序运行时的保护内容，进一步增强了系统的安全性；基于指令集的扩展与独立的认证方式，使得应用程序可以灵活调用这一安全功能并进行验证.作为系统安全领域的重大研究进展，Intel SGX 是基于 CPU 的新一代硬件安全机制，其健壮、可信、灵活的安全功能与硬件扩展的性能保证，使得这项技术具有广阔的应用空间与发展前景。

1.2.2 重复数据删除（Deduplication）

重复数据删除[1-2]是一种数据缩减技术，通常用于数据的备份系统，旨在减少使用的存储容量。它的工作方式是在某个时间周期内查找不同文件中不同位置的重复可变大小数据块，重复的数据

块用指示符取代。因此，对于高度冗余的数据集，重复数据删除可以极大地减小存储的开销，提高存储的效率。

目前此项技术的基本方法有三种，第一种是基于散列（hash）的方法，例如 Data Domain、飞康、昆腾的 DXi 系列设备都是采用 SHA-1，MD-5 等类似算法将这些进行备份的数据流断成块并且为每个数据块生成一个散列。如果新数据块的散列与备份设备上散列索引中的一个项相匹配，表明该数据已被备份。

第二种方法是基于内容识别的重复删除，这种方法主要是识别记录的数据格式，它采用内嵌在备份数据中的文件系统的元数据识别文件，然后与数据存储库的其他版本逐字节对比。这种方法可以避免散列冲突，但是需要使用支持的备份应用设备以便设备提取元数据。

第三种方法是 Diligent Technologies 用于其 ProtecTier VTL 的技术，它像基于散列的产品那样将数据分成块，并采用自有算法决定给定数据块是否与其他相似。

一种常见的重复数据删除过程如图 6：

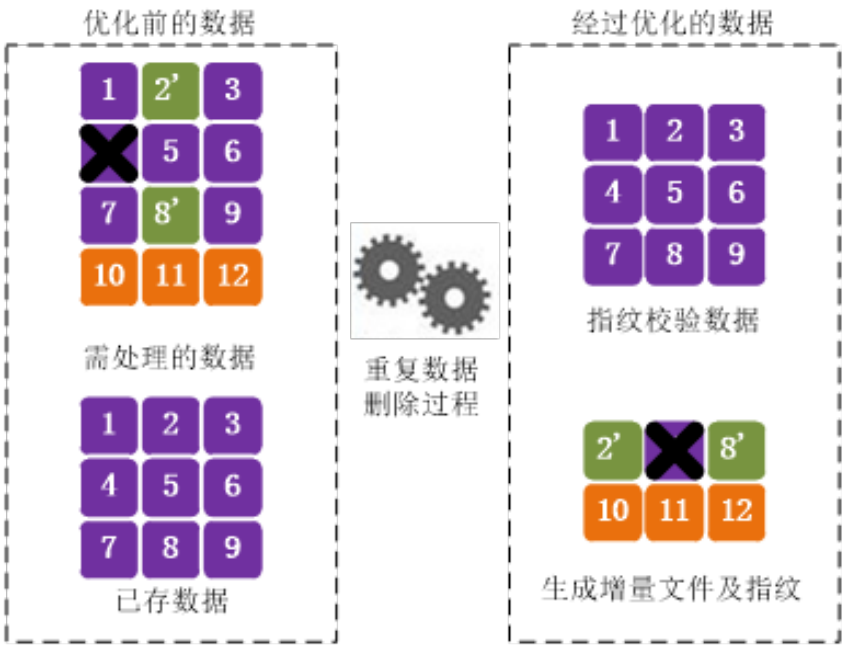

图 6：常见的数据删除过程

1.2.3 加密去重

去重却是和数据加密的目标直接相矛盾的，云存储系统中使用支持去重的加密问题的主要挑战有两点：

• 加密之后的密文需要保留原文的冗余，即原文相同的数据块加密后的密文仍相同（这里的相同不一定是密文的全等，系统只要一种识别包含相同内容的密文的手段即可），这样去重才能

够起作用。

• Cross-user decryption （CUD），即由某个用户加密上传的数据块应该能够被所有有读取权限的用户解密，即使后者不是最初的上传和加密者。

第二点可以用“lockbox”或者key encapsulation 的方式解决。第一点所需的特性我们称之为convergent特性，它是基于去重的数据压缩不可或缺的。但是，它与加密算法的安全性定义有不可调和的矛盾。Semantic security明确禁止原文相等性的检测，即给定两个密文，不应该能够允许对手断定它们加密的是否是同样的数据，否则对手可以利用这一性质攻破前述IND。于是退而求其次，即我们可以适度放宽对安全性的要求，允许密文泄露原文相等性信息，从而使加密后的去重成为可行。

最早提出的方案是Convergent Encryption(CE)[3]，它的想法非常简单：一个数据块d的加密方法为：E(h(d),d)。其中E(key,d)是以key做秘钥加密数据d的对称加密算法，h(x)是一个hash function。也就是说，当需要加密一个数据块d的时候，CE先用数据内容生成key，再用一个symmetric encryption算法（如AES等）加密。

但是一个令密码学研究者不安的状况是，虽然CE已经被广泛应用，它的安全性却始终没有严格的分析。人们曾经做过一些边缘性的工作，比如人们研究过deterministic encryption所能达到的安全性。直到2013年，Mihir Bellare把这些研究都纳入了Message-Locked Encryption（MLE）[4]的框架。简单地讲，MLE是这样一种加密算法，它使用的key是从待加密的原文算出来的。MLE最初被用于重复数据的删除，通过消除用户数据中的冗余副本来提高存储利用率，降低通信成本。但对重复数据删除的实际研究表面，文件块（block）级的重复数据删除可以节省更多的空间。

在加密过程中如何引入额外的秘密且同时保证密文的convergent特性呢？和MLE的思路类似，我们需要由数据产生加密的key（从而保证convergent特性），而我们不希望这一个过程是公开的。一个办法是，由第三方服务用自己的秘钥对数据签名（这里签名算法必须是deterministic的），然后用签名作为伪随机数生成器的种子，从而得到一致的加密key这种方法被称为**Encryption with Signature (EwS)**，其过程可以用图7表示：

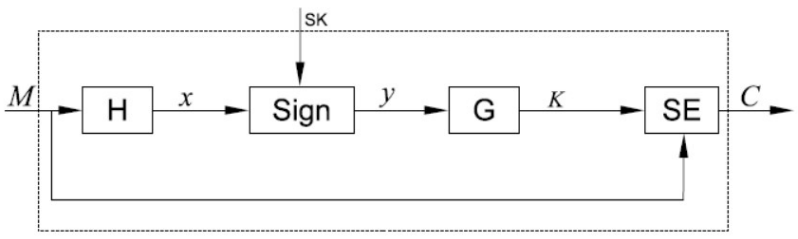

图 7：EwS 流程图

1.2.4 数据所有权证明

在重复数据删除中,存在这样一种攻击:知道文件哈希签名的攻击者可以说服存储服务它拥有该文件，因此服务器允许攻击者下载整个文件，这就要引入数据所有权（其他地方统一改为所有权）证明的概念[5-6]。传统存储系统的数据持有性证明主要采用基于访问的方法，这些系统将数据下载到本地，由用户逐一验证数据的存在性及完整性。这种方式需要频繁地访问服务器上的数据，增加了服务器的负担，严重浪费了网络的带宽资源。在具有数据量较大的应用特征的云存储环境下，这种方式显然效率太低，不适合云存储应用。为了克服此类攻击,引入了所有权证明(Proof-of-Ownership)的概念。现有的数据持有型证明方案之一是基于对称加密体制的方案。基于对称密码学的方案以 RSA 公司的 Juels 和 EMC 公司的 Kaliski 提出的基于岗哨的可恢复证明系统 POR 为典型代表。其基本思想是首先用对称 加密体制将文件加密并用纠错码编码，然后在编码后的文件中的一些随机位置插入和文件数据不可区分的“岗哨”（sentinel)；检查者在挑战时要求服务器返回一些随机位置的岗哨。作者证明只要服务器以大于一定值的概率做出有效的应答，则文件是可以恢复的。同属 RSA 实验室的 Bowers 等人在 Juels 等人研究的基础上提出了一个 POR 的理论框架，用于改进已有 POR 方案，实现更低的存储开销和更高的检错率。Halevi 等人提出了 PoW 的解决方案，PoW (Proof-of-Ownership)是一种加密方法，它为基于源的重复数据删除增加了对侧通道攻击的保护，同时保持了基于源的重复数据删除的带宽节省。它的想法是让云验证客户端确实是密文块的所有者，并被授权对密文块进行完全访问。这确保了受影响的客户端不能查询其他客户端块的存在。具体来说，在基于 PoW 的源数据重删中，客户端将发送到云端的每个指纹都附有 PoW 证明，云通过 PoW 证明来验证客户端是否是对应密文块的真正所有者。云只在证明验证成功后才响应，从而防止任何受影响的客户端识别其他客户端拥有的密文块。数据拥有证明（POD）和可检索性证明（POR）通常用于识别客户端的数据是否可以恢复，所有权证明（POW）用于减少相同数据的多个副本的存储，从而减少存储空间和网络带宽的使用，从而使存储效率提高。

1.2.5 SM 系列国密算法

国密即国家密码局认定的国产密码算法，主要包括 SM1、SM2、SM3、SM4。

1.2.5.1 SM1

SM1[7]算法是对称加密算法，但是该算法不公开，如果想调用该算法，需要通过芯片的接口进

行调用。

1.2.5.2 SM2

SM2[8]和 RSA 都是公钥密码算法，但 SM2 算法是一种更先进安全的非对称加密算法，基于 ECC，而 SM2 分组长度为 256 位，安全强度比 RSA2048 位高。

SM2 算法的加密流程如下：

（1）用户 A 拥有椭圆曲线$y^2 = x^3 + ax + b$，len比特的明文M，用户 B 的公钥P_B

（2）产生随机数$k \in [1, n-1]$，计算椭圆曲线上的点$C_1 = [k]G = (x_1, y_1)$

（3）计算$[k]P_B = (x_2, y_2)$

（4）计算$t = KDF(x_2||y_2, len)$

（5）计算$C_2 = M \oplus t$

（6）计算$C_3 = hash(x_2||M||y_2)$

（7）输出密文$C = C_1||C_2||C_3$

在第四步中出现的 KDF 为密钥派生函数，作用是从一个共享的秘密比特串派生出密钥数据，需要调用密码杂凑函数。SM2 的 KDF 具体步骤如图 8 所示：

密钥派生函数 $KDF(Z, klen)$：

输入：比特串 Z，整数 $klen$（表示要获得的密钥数据的比特长度，要求该值小于 $(2^{32}-1)v$）。

输出：长度为 $klen$ 的密钥数据比特串 K。

a) 初始化一个 32 比特构成的计数器 ct=0x00000001；

b) 对 i 从 1 到 $\lceil klen/v \rceil$ 执行：

b.1) 计算 $Ha_i = H_v(Z \| ct)$；

b.2) ct^{++}；

c) 若 $klen/v$ 是整数，令 $Ha!_{\lceil klen/v \rceil} = Ha_{\lceil klen/v \rceil}$，

否则令 $Ha!_{\lceil klen/v \rceil}$ 为 $Ha_{\lceil klen/v \rceil}$ 最左边的 $(klen - (v \times \lfloor klen/v \rfloor))$ 比特；

d) 令 $K = Ha_1 \| Ha_2 \| \cdots \| Ha_{\lceil klen/v \rceil - 1} \| Ha!_{\lceil klen/v \rceil}$。

图 8：SM2 中 KDF 详细步骤

1.2.5.3 SM3

SM3[9]密码杂凑算法是中国国家密码管理局 2010 年公布的中国商用密码杂凑算法标准，适用于商业密码应用中的数字签名和验证，是在[SHA-256]基础上改进实现的一种算法，采用了

Merkle-Damgard 结构，消息分组长度为 512 位，摘要长度为 256 位。

整个算法的执行流程可以分为四个步骤：消息填充、消息扩展、迭代压缩、输出结果。

（1）消息填充

SM3 消息扩展步骤是以 512 位的明文分组作为输入，所以在一开始就将明文长度填充为 512 的倍数，填充规则如下：

（i）先填充一个”1”，后面加上k个”0”，其中k满足：

$$(n+1+k) \equiv 448(mod\ 512)$$

（ii）追加 64 位的明文长度，这 64 位为明文消息长度的二进制表示。

（2）消息扩展

消息扩展的具体步骤如图 9 所示：

将消息分组$B^{(i)}$按以下方法扩展生成132个字$W_0, W_1, \cdots, W_{67}, W_0', W_1', \cdots, W_{63}'$，用于压缩函数$CF$：

a)将消息分组$B^{(i)}$划分为16个字$W_0, W_1, \cdots, W_{15}$。

b)FOR j=16 TO 67

$W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6}$

ENDFOR

c)FOR j=0 TO 63

$W_j' = W_j \oplus W_{j+4}$

ENDFOR

图 9：SM3 消息扩展步骤

即将 512 位明文划分为 16 个字（每个字 4 个字节/32 位），将这 16 个字作为前 16 个字，再递推生成剩余的 116 个字。此步骤结束后，共产生 132 个字。

（3）迭代压缩

首先我们设置一个初值向量 IV，IV 被放在 A、B、C、D、E、F、G、H 八个 32 位变量中，迭代过程如图 10 所示：

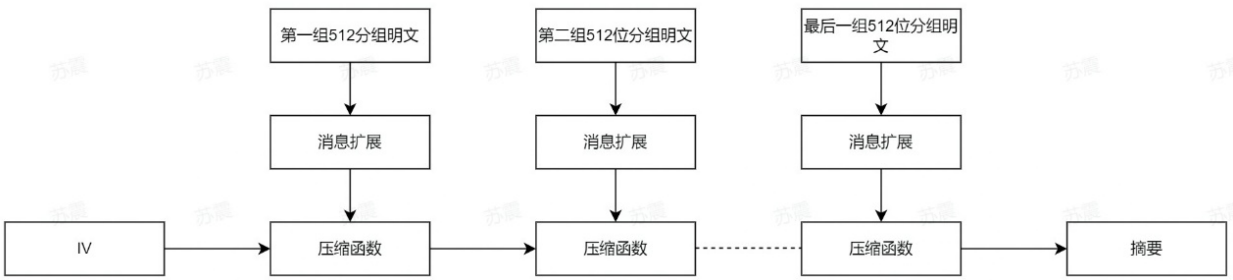

图 10：SM3 迭代压缩整体过程

而压缩函数的步骤如图 11 所示：

令A,B,C,D,E,F,G,H为字寄存器,SS1,SS2,TT1,TT2为中间变量,压缩函数$V^{i+1} = CF(V^{(i)}, B^{(i)}), 0 \leq i \leq n-1$。计算过程描述如下：

$ABCDEFGH \leftarrow V^{(i)}$

FOR j=0 TO 63

$SS1 \leftarrow ((A \lll 12) + E + (T_j \lll j)) \lll 7$

$SS2 \leftarrow SS1 \oplus (A \lll 12)$

$TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W_j'$

$TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$

$D \leftarrow C$

$C \leftarrow B \lll 9$

$B \leftarrow A$

$A \leftarrow TT1$

$H \leftarrow G$

$G \leftarrow F \lll 19$

$F \leftarrow E$

$E \leftarrow P_0(TT2)$

ENDFOR

$V^{(i+1)} \leftarrow ABCDEFGH \oplus V^{(i)}$

其中，字的存储为大端(big-endian)格式。

图 11：SM3 压缩函数

1.2.5.4 SM4

SM4[10]是一种分组密码算法，其分组长度为 128bit，密钥长度也为 128bit。加密算法和密钥扩展算法均采用 32 轮非线性迭代结构，以字为单位进行加密运算。

SM4 的整体流程如下图，主要分为密钥扩展与加密两部分，如图 12 所示：

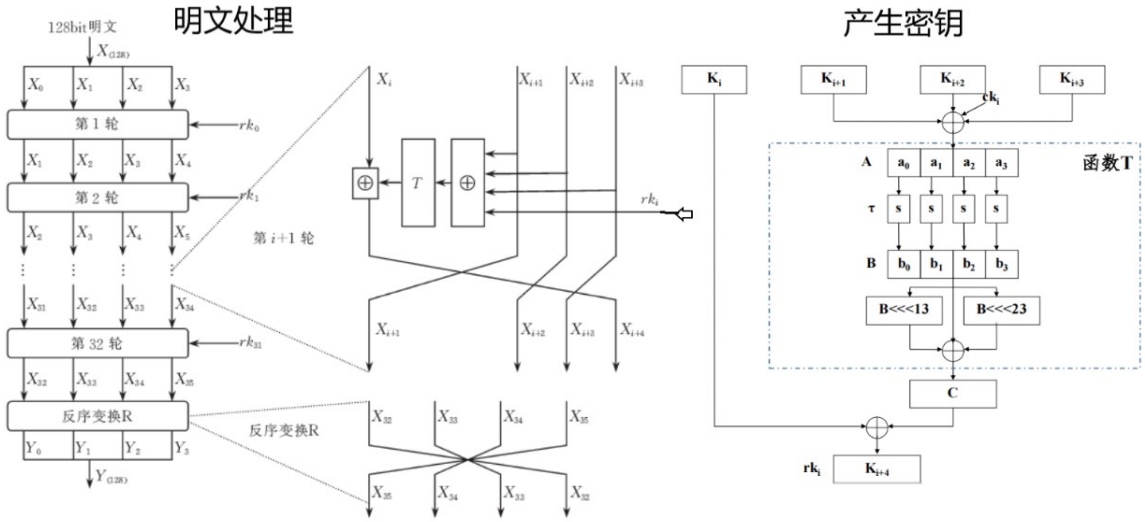

图 12：SM4 算法整体结构

（1）加密过程

加密算法可描述如下：

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)$$

$$= Xi \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i)$$

$$(Y_0, Y_1, Y_2, Y_3) = (X_{35}, X_{34}, X_{33}, X_{32})$$

rk_i为轮密钥（i=0, 1, 2, 3……）

其中，$T: Z_2^{32} \to Z_2^{32}$是一个可逆变换，由非线性变换τ和线性变换L组成，即$T(x) = L(\tau(x))$

（i）非线性变换τ

τ由 4 个并行的 S 盒组成（如图 13）。

设输入为$A = (a_0, a_1, a_2, a_3) \in (Z_2^8)^4$，则输出$B$为：

$$B = (b_0, b_1, b_2, b_3) = (Sbox(a_0), Sbox(a_1), Sbox(a_2), Sbox(a_3))$$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D6	90	E9	FE	CC	E1	3D	B7	16	B6	14	C2	28	FB	2C	05
1	2B	67	9A	76	2A	BE	04	C3	AA	44	13	26	49	86	06	99
2	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	ED	CF	AC	62
3	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	FA	75	8F	3F	A6
4	47	07	A7	FC	F3	73	17	BA	83	59	3C	19	E6	85	4F	A8
5	68	6B	81	B2	71	64	DA	8B	F8	EB	0F	4B	70	56	9D	35
6	1E	24	0E	5E	63	58	D1	A2	25	22	7C	3B	01	21	78	87
7	D4	00	46	57	9F	D3	27	52	4C	36	02	E7	A0	C4	C8	9E
8	EA	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	A1
9	E0	AE	5D	A4	9B	34	1A	55	AD	93	32	30	F5	8C	B1	E3
A	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0D	53	4E	6F
B	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6D	6C	5B	51
C	8D	1B	AF	92	BB	DD	BC	7F	11	D9	5C	41	1F	10	5A	D8
D	0A	C1	31	88	A5	CD	7B	BD	2D	74	D0	12	B8	E5	B4	B0
E	89	69	97	4A	0C	96	77	7E	65	B9	F1	09	C5	6E	C6	84
F	18	F0	7D	EC	3A	DC	4D	20	79	EE	5F	3E	D7	CB	39	48

图 13：SM4 的 S 盒

（ii）线性变换L

设输出为C

$$C = L(B) = B \oplus (B <<< 10) \oplus (B <<< 18)(B <<< 13) \oplus (B <<< 24)$$

（2）密钥扩展算法

（i）常数FK

$$FK_0 = (A3B1BAC6)$$

$$FK_1 = (56AA3350)$$

$$FK_2 = (677D9197)$$

$$FK_3 = (B27022DC)$$

（ii）固定参数CK

共有32个参数$CK_i(i = 0,1,2\ldots\ldots,31)$，每个参数都是一个字

（iii）密钥扩展

假设输入的加密密钥为$MK = (MK_0, MK_1, MK_2, MK_3)$

首先计算

$$(K_0，K_1，K_2，K_3) = (MK_0 \oplus FK_0，MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3)$$

再循环计算轮密钥rk_i

$$rk_i = K_{i+4} = K_i \oplus T'(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i)$$

变换T'与加密过程中的T基本相同，只将其中的线性变换L修改为以下的L'：

$$L'(B) = B \oplus (B <<< 13) \oplus (B <<< 23)$$

2 研究进展

2.1 相关工作

目前对于加密重复数据删除的研究已有很多。加密重复数据删除保留了加密数据的重复数据消除效果，对外包存储具有吸引力。然而，现有的加密重复数据删除方法建立在昂贵的加密原语

上，这会导致性能大幅下降。

将存储管理外包给云是客户（企业或个人）的常见做法，以节省自我管理海量数据的开销，安全性和存储效率是实际外包存储的两个主要目标。为了满足这两个目标，现有研究探索了加密重复数据删除，这是一种将加密和重复数据消除相结合的范例，它总是使用从块内容本身派生的密钥将重复的明文块（来自相同或不同的客户端）加密为重复的密文块；例如，密钥可以是对应块的密码散列。

因此，任何重复的密文块都可以通过重复数据消除来提高存储效率，而所有外包的块都被加密以防止未经授权的访问。加密重复数据删除特别适用于备份应用程序，这些应用程序具有高内容冗余，是外包存储的有吸引力的使用案例。现有的加密重复数据删除方法通常需要高性能开销来实现安全保证。

我们使用最先进的加密重复数据删除系统 DupLESS[11]作为一个代表性示例来解释性能问题。

首先，为了防止对手推断出内容派生密钥，DupLESS 采用了服务器辅助密钥管理，其中部署了专用密钥服务器来管理客户端的密钥生成请求。然而，服务器辅助密钥管理需要昂贵的加密（密码学）操作，以防止密钥服务器在密钥生成期间知道明文块和密钥。

其次，防止对手通过推断重复数据消除模式（也称为侧信道攻击）获得对密文块的未授权访问，DupLESS 可以采用以下方法之一：

（i）执行基于目标的重复数据消除（即上传所有密文块并让云移除任何重复的密文块），从而保护重复数据消除模式免受任何客户端的攻击；

（ii）执行基于源的重复数据删除（即，在客户端删除所有重复的密文块而不上传到云），并且向云进一步证明它确实是密文块的所有者（即，可以访问对应明文块的完整内容），并且被授权对密文块执行重复数据删除。前者需要额外的通信带宽来上传重复的密文块，而后者需要昂贵的加密操作来证明客户端是密文块的所有者。尽管已经提出了各种协议设计来解决加密重复数据删除的性能问题，但它们通常会削弱安全性，增加带宽开销，或降低存储效率。

2.2 基于 SGX 的加密去重系统

2.2.1 SGXDedup

SGXDedup[12]在 DupLESS 中的服务器辅助密钥管理之上，在包内执行高效的加密操作，其有三个主要构建块：

•安全高效的飞地管理：它可以防止密钥服务器泄露，并允许客户端在重新启动后快速引导飞

地。

•可更新盲密钥管理：它生成一个盲密钥，用于基于密钥回归保护密钥服务器内的飞地和每个客户端之间的通信，从而盲密钥可更新以进行动态客户端身份验证。

•基于 SGX 的推测加密：它通过推测加密减轻了安全信道管理的在线加密/解密开销。

SGXDedup 的整体架构如图 14 所示：

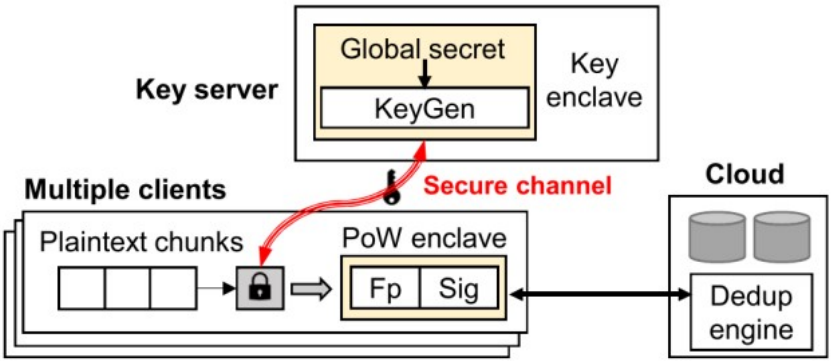

Figure 1: Overview of SGXDedup architecture: a key enclave and a PoW enclave are deployed in the key server and each client, respectively.

图 14：SGXDedup 整体框架

首先，SGXDedup 在客户端和$Key\ server$端分别部署了两个飞地（enclave）。双方通过密钥协商协议，为实现安全通信，生成盲密钥（shared blined key）。

随后，用户将明文信息M的指纹$fg = H(M)$通过安全信道（Secure channel）传输至$Key\ server$端（基于盲密钥加密传输），随后$Key\ server$端在飞地端计算数据加密的密钥K_M：

$$K_M = H(fg||d)$$

其中，d为系统主密钥。在得到K_M后，通过安全信道传输至客户端，客户端对所有的明文块进行加密：

$$C_M = E(K_M, M)$$

随后计算密文的指纹CFG：

$$C_{fg} = H(C_M)$$

以及指纹的签名$Sig_{\{PoWkey\}}(CFG)$。

将密文指纹以及指纹签名传到云端，进行所有权证明，随后对数据进行存储。

• 优点：

（1）在以上方案中，密钥生成减少了原来的 OPRF 协议，提高效率。

（2）由于通信是基于通信密钥的，所以敏感信息不会泄漏。

• **缺点：**

（1）在客户端去重时，去重检测前就直接生成密钥，对明文进行加密，再进行去重，没有实现去重后加密的思路，进行了大量的冗余计算，增加了计算的开销；

（2）基于密文进行所有权证明（PoW），只能验证用户确实拥有密文，不能证明用户拥有明文，敌手可能通过截获通信等方式得到完整密文，通过所有权证明。

2.2.2 DEBE

基于此，又有人研究了 DEBE[13]，这是一种基于屏蔽 DbE 的重复数据消除存储系统，它考虑到了性能、存储节省和安全性。DEBE 也建立在 Intel SGX 之上，它提供了一个被称为包围区的屏蔽执行环境，用于安全的重复数据消除处理。在 Intel SGX 中实现 DEBE 的一个关键挑战是有限的飞地空间。因此，我们提出了基于频率的重复数据消除，这是一种两阶段重复数据消除方案，可以在空间受限的情况下实现安全、轻量级的重复数据删除。具体来说，DEBE 首先对包围区内最频繁的块执行重复数据消除，因为在真实的备份工作负载中，最频繁的数据块通常会导致大量重复。然后，它对飞地之外的其余频率较低的块执行重复数据消除。使用基于频率的重复数据消除，DEBE 具有以下关键优势：

（i）高性能，因为它在第一阶段重复数据消除中删除了大部分重复数据，并且在飞地之外的第二阶段重复数据删除带来了有限的性能开销；

（ii）通过重复数据消除和压缩实现高存储节省；

（iii）安全性，因为它保护了飞地内最频繁的块，这些块更容易受到频率分析攻击。

但是，此方案数据加密全部都是在 Enclave 内部完成，可信环境下的计算量大，并且数据机密性都依赖 Enclave 中的密钥，一旦被泄露就没有任何安全保证；服务器去重，不能够降低通信开销；对数据集要求高，只能用于频率非常不均匀的数据集。

DEBE 的设计相比于 SGXDedup 去除了密钥服务器，只保留客户端和云服务器。客户端也不再部署飞地了。直接在服务器端部署飞地。飞地维护$data\ key$，$query\ key$还有$Session\ key$。飞地与每个客户端都维护一个会话密钥。

DEBE 的结构如图 15 所示：

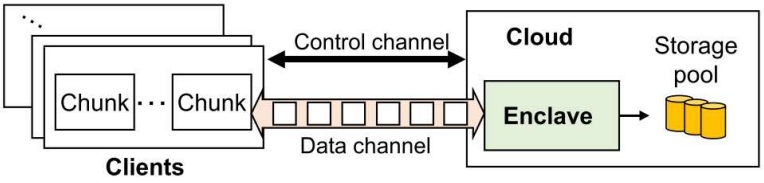

图 15：DEBE 整体架构

上传文件时，客户端将文件拆分为块，通过 control 信道发送上传请求，通过安全的数据信道（$data\ channel$）直接发送明文块至$Enclave$（飞地）。$Enclave$去重并压缩收到的明文块（批量处理），加密所有非重复块，输出至服务器。

$Enclave$在内部维护一个小的索引（top-k），只记录最高频的几个块的索引。

上传块时，$Enclave$恢复出M，使用 CM-sketch 方法进行频率计数。

- 如果 M 的频率是在前 k 个，只需要更新元数据。
- 如果不是，进入第二轮去重。

第二轮去重：服务器维护一个完整的加密索引$C_{fg} = E(k_q, fg)$。在第二轮中，$Enclave$加密M的指纹得到C_{fg}，使用完整索引进行去重检测。若数据是非重复块，$Enclave$使用k_D加密M，输出密文至服务器。

有几种通过 DaE 实现安全的重复数据消除方法。一些方法是从安全角度设计的。随机 MLE 和 iMLE 应用非确定性加密来防止频率泄漏，但它们使用昂贵的原语（例如，非交互式零知识证明、完全同态加密），这些原语尚未准备好实施。研究人员通过分散协议共享密钥，而不依赖于专用密钥服务器，但这会在不同客户端之间引入昂贵的性能开销。TED 通过可配置的存储爆炸来缓解频率泄漏。相比之下，DEBE 实现了 DbE 以同时解决密钥管理开销和安全问题。

此方案虽然实现了在$Enclave$内部去重后加密，只处理了不重复的数据，但数据加密全部都是在$Enclave$内部完成，可信环境下的计算量大，并且数据机密性都依赖$Enclave$中的密钥，一旦被泄露就没有任何安全保证；服务器去重，不能够降低通信开销；对数据集要求高，只能用于频率非常不均匀的数据集。

3 作品功能与性能说明

3.1 基本方案

针对现有加密去重系统中存在的冗余计算和所有权证明等问题，我们提出了基于国密算法和 Intel SGX 的基本方案。方案基于 SGX 实现客户端去重，旨在实现去重后加密，减少了系统中的冗余运算。此外，本方案基于完整的数据明文计算所有权证明值，防止了仅拥有密文而非明文的敌手发起所有权欺骗攻击。

3.1.1 实现流程

设用户外包数据为M，并利用 Intel SGX 技术，分别在客户端和服务器端部署两个飞地：客户端$C_Enclave$、服务器端$S_Enclave$。

• 用户端计算明文的指纹及指纹的密文，并对指纹进行签名：

$$fg = H(M)$$

$$C_{fg} = E(K, fg)$$

$$Sig_k(C_{fg})$$

用户端首先将外包数据M输入至$C_Enclave$中，$C_Enclave$计算数据的哈希值（指纹）$H(M)$（基于国密算法 SM3）。而由$C_Enclave$计算数据哈希值的原因在于，我们目标是基于完整的明文数据进行所有权证明（PoW）。随后$C_Enclave$对指纹用通信密钥K进行加密（基于国密算法 SM4），并对指纹密文进行签名（基于 HMAC-SHA256），其中，K为两个$Enclave$之间的协商密钥。

完毕后，将指纹密文C_{fg}以及其签名$Sig_k(C_{fg})$通过安全信道传输给服务端（如图 16）。

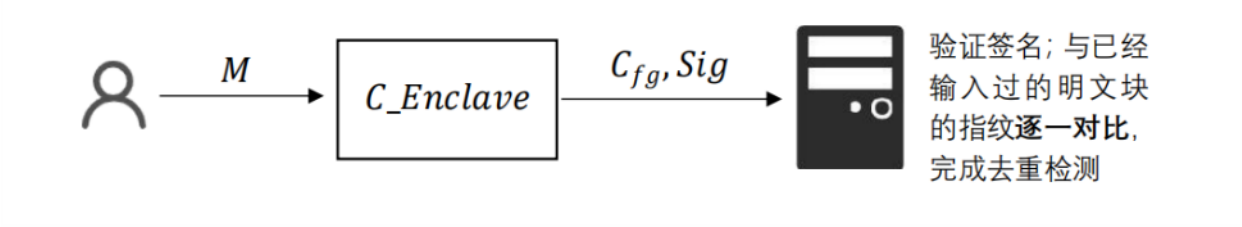

图 16：客户端加密数据上传

• S_Enclave解密得到指纹fg，计算标签T，输出至服务器用于去重：

$$fg = D(k_e, C_{fg})$$

$$T = H(k_T, fg)$$

服务端接收到用户端传输的指纹密文C_{fg}以及其签名$Sig_k(C_{fg})$（基于 HMAC-SHA256），验证签名后，用通信密钥K解密，得到明文数据的指纹，再用存储于$S_Enclave$的密钥k_T来计算指纹的标签T，利用标签比对来进行去重检测（如图 17）。

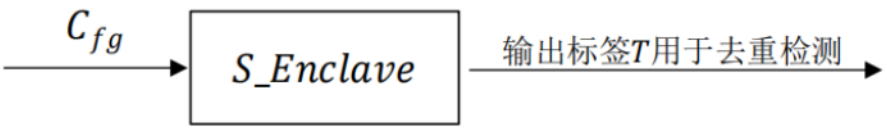

图 17：服务端去重检测

通过标签T检测重复的方法则是与已经存储过的指纹进行逐一比对，实现方法如下：

（1）方案选择将去重检测移出$S_Enclave$，云服务器端存储所有索引。

（2）若已经存储过T，则数据已存储过，只需将密钥返回至用户端。

（3）若没存储过 T，则需要用户端加密上传完整数据。

于此同时，$S_Enclave$也计算明文的加密密钥，并加密此密钥：

$$k_M = H(k_d, fg)$$

$$C_k = E(K, k_M)$$

$S_Enclave$用存储的密钥k_d对指纹进行哈希值运算（基于 HMAC-SHA256）得到加密密钥k_M。然后用通信密钥K对加密秘钥k_M进行加密（基于国密算法 SM4）。

其中k_T、k_d安全存储于$S_Enclave$中。这里将明文加密密钥加密，是因为下一步要将密钥传回用户端，用于明文加密。

• 服务端将标签T、C_k、去重检测结果，即$< T, C_k, 0/1$(表示是否存储) $>$通过安全信道一同发回给用户端（如图 18）：

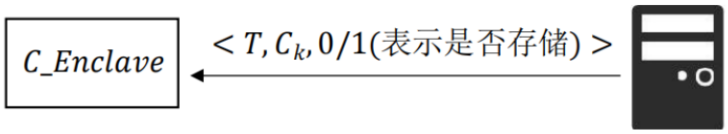

图 18：服务端将结果发送至用户端

在$S_Enclave$计算完标签T、加密密钥的密文C_k以及完成去重检测时，将这三项基于安全信道（协商密钥为K）传输回用户端。

• 用户首先解密C_k得到加密密钥k_M，用k_M加密明文M，并生成计算的签名Sig：

$$k_M = D(K, C_k)$$

$$C_M = E(k_M, M)$$

$$Sig_k(C_M)$$

用户首先用通信密钥K解密C_k得到加密密钥k_M，用k_M加密明文M，并生成密文的签名（同样基于 HMAC-SHA256）$Sig_k(C_M)$。

这里用户发送密文时也需要计算签名（如图 19）的原因在于，我们要防止文件伪造攻击。

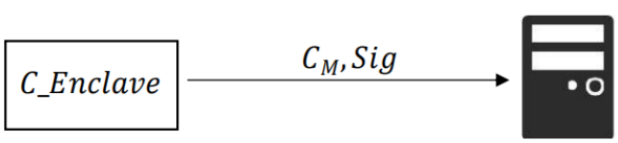

图 19：用户端加密数据后上传云端

• 服务器将密文存储至云端。至此整个方案的过程完毕：

服务端接收用户端发来的密文C_M及其签名$Sig_k(C_M)$，验证签名后，将密文存储至云端，完成整个云存储的过程。

3.1.2 基础方案创新点

相比于现有方案，我们基础方案的创新点如下：

1.充分利用 Intel SGX 技术，实现了安全高效的用户端去重，高效解决了密钥传递，所有权证明和文件伪造攻击等问题；

2.与现有方案相比，移除了第三方服务器，只部署了两个$\boldsymbol{Enclave}$，使得整个系统结构更加清晰明了，减少了资源的开销；

3.现有方案的去重检测方法是只统计高频块。与现有方案相比，我们的基础方案将去重检测过程放到了$\boldsymbol{Enclave}$外，不是采用此前的只统计高频块的特性，这使得去重检测更加准确；

4.现有方案去重检测的顺序为：先对明文块进行加密，后对密文进行去重。这个过程便产生了问题：无论数据块是否重复，都需要先对数据块进行加密，随后进行去重，即对重复的数据块也会进行加密，降低了计算的效率，增大了云存储的整个开销。而我们的现有方案则是先对明文块进行，通过去重后加密，减少了$\boldsymbol{Enclave}$内部的加密操作，大大减小了整体的开销，提高了云存储的效率。

但是，基础方案仍存在一定的局限性：

• 去重检测需要通过与所有的$\boldsymbol{T}$进行比较，去重检测的效率不高

• 不管数据是否重复，都需要$\boldsymbol{S_Enclave}$计算$\boldsymbol{T}$、$\boldsymbol{C_k}$等信息，仍存在冗余计算的问题。

3.2 增强方案

为解决基础方案中存在的局限性，我们提出了增强方案，基于备份数据的时间局部性提高系统中去重检测的执行效率。

设用户外包数据为M，并利用 Intel SGX 技术，分别在客户端和服务器端部署两个飞地：客户端$C_Enclave$、服务器端$S_Enclave$，基于时段进行密文去重检测。

每个时段开始前，$S_Enclave$生成一个时段密钥k_e，作为该时段内两个飞地的协商密钥，并基于安全信道，发送给所有用户端的$C_Enclave$。具体方案如 3.2.1 节所示:

3.2.1 实现流程

• 用户端计算明文的指纹及指纹的密文并对其进行签名：

$$fg = H(M)$$

$$C_{fg} = E(k_e, fg)$$

$$Sig_k(C_{fg})$$

用户端首先将外包数据 M 输入至 C_Enclave 中，C_Enclave 计算数据的哈希值（指纹）$H(M)$（基于国密算法 SM3），目标同样是基于完成的明文数据进行所有权证明（PoW）。随后 C_Enclave 对指纹用k_e进行加密（基于国密算法 SM4），并对指纹密文进行签名（基于 HMAC-SHA256）。其中，k_e为当前时段的时段密钥。计算完毕后，将指纹密文C_{fg}以及其签名$Sig_k(C_{fg})$通过安全信道传输给服务端。（如图 20）

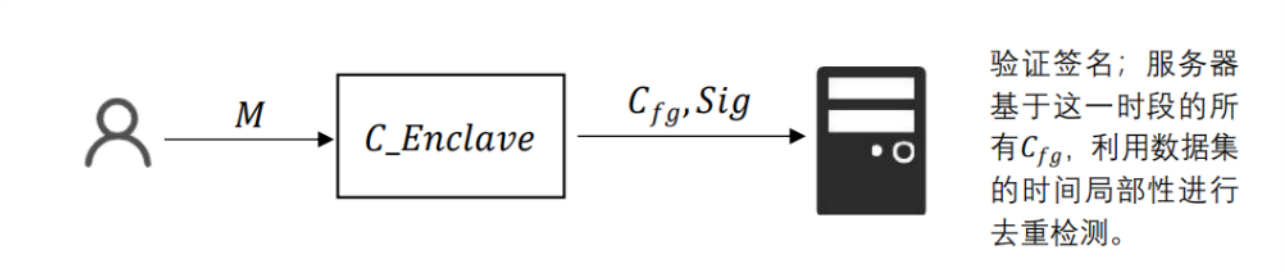

图 20：客户端加密数据上传

• $S_Enclave$解密得到指纹fg，计算标签T，输出至服务器用于去重检测：

$$fg = D(k_e, C_{fg})$$

$$T = H(k_T, fg)$$

服务端接收到用户端传输的指纹密文C_{fg}以及其签名$Sig_k(C_{fg})$（基于 HMAC-SHA256），验证签名后，用时段密钥k_e解密，得到明文数据的指纹，再用存储于$S_Enclave$的密钥k_T来计算指纹的标签T，利用标签比对来进行去重检测（如图 21）。

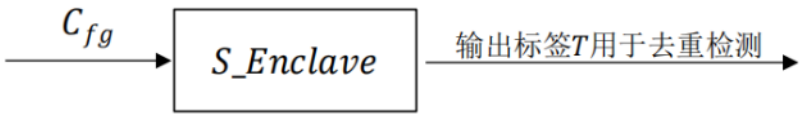

图 21：服务端去重检测

与基础方案不同，在增强方案中，通过标签 T 检测重复的方法则是：我们记录前k个出现频率最高的块，并将每个输入的标签与这k个标签逐一比较，具体步骤如图 22 所示：

（1）方案选择将去重检测移出$S_Enclave$，云服务器端存储所有索引。

（2）若标签T属于这k个标签，则数据已存储过，只需将密钥返回至用户端并更新频率表。

（3）若标签T不属于这k个标签，则需要用户端加密上传完整数据。

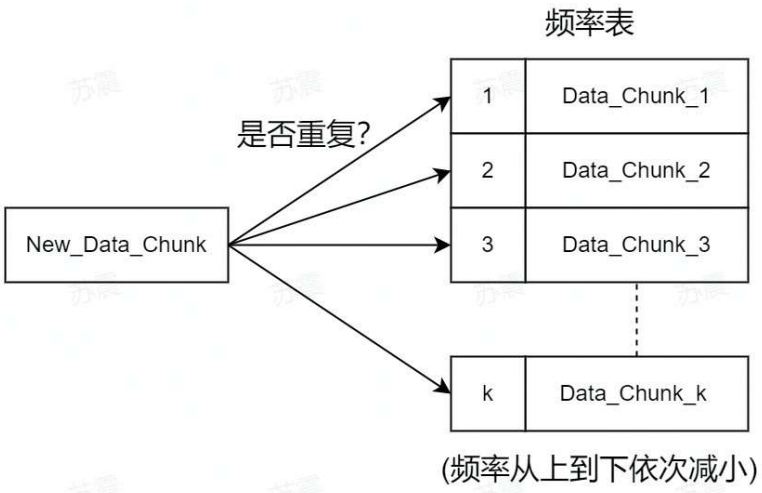

图 22：增强方案去重检测方法

于此同时，$S_Enclave$也计算明文的加密密钥，并加密此密钥。**与基础方案不同，在此我们利用时段密钥k_e对k_M进行加密：**

$$k_M = H(k_d, fg)$$

$$C_k = E(k_e, k_M)$$

$S_Enclave$用存储的密钥k_d对指纹进行哈希值运算（基于 HMAC-SHA256）得到加密密钥k_M。然后用通信密钥K对加密秘钥k_M进行加密（基于国密算法 SM4）。

其中，k_T、k_d安全存在飞地中，用于生成标签和密钥。这里将明文加密密钥加密，是因为下一步要将密钥传回用户端，用于明文加密。

• 服务端将标签T、C_k、去重检测结果，即$< T, C_k, 0/1$(表示是否存储)$ >$通过安全信道一同发回给用户端（如图 23）：

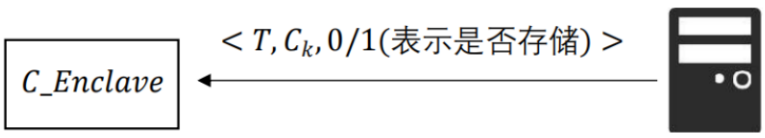

图 23：服务端将结果发送至用户端

在$S_Enclave$计算完标签T、加密密钥的密文C_k以及完成去重检测时，将这三项基于安全信道传输回用户端。

• 用户首先解密C_k得到加密密钥k_M，用k_M加密明文M，并生成计算的签名Sig：

$$k_M = D(k_e, C_k)$$

$$C_M = E(k_M, M)$$

$$Sig_k(C_M)$$

用户首先通过时段密钥k_e解密C_k得到加密密钥k_M，用k_M加密明文M，并生成密文的签名（同样基于 HMAC-SHA256）$Sig_k(C_M)$。

这里用户发送密文时也需要计算签名的原因在于，我们要防止文件伪造攻击（如图 24）。

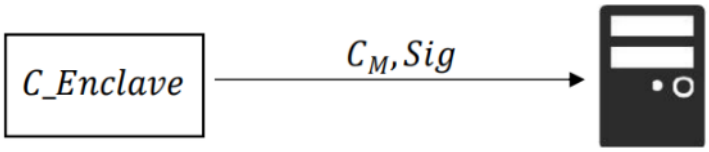

图 24：用户端加密数据后上传云端

• 服务器将密文存储至云端。至此整个方案的过程完毕：

服务端接收用户端发来的密文C_M及其签名$Sig_k(C_M)$，验证签名后，将密文存储至云端，完成整个云存储的过程。

3.2.2 增强方案创新点

与基础方案不同，我们基于每个时段使用统一的时段密钥k_e生成C_{fg}，并且基于k_e来搭建用户端和服务端之间的安全信道。这里应用时段密钥k_e的主要原因是：在备份数据集中，时间局部性是普遍存在的特性。因此应用时段密钥k_e可以很好地利用备份数据集这一特性，大幅降低去重检测的性能开销，服务器可直接重用此前的计算结果，降低了$Enclave$内的运算开销。

3.3 技术指标

本方案的技术指标如下：

（1）系统在模拟/真实数据集上的数据加密上传的时间开销；

（2）系统在模拟/真实数据集上的数据下载解密的时间开销；

4 设计与实现方案

4.1 测试方案

本章在第三章系统实现基础上，对本系统进行测试。为了测试系统的性能，我们选择了 16M、100M、500M 的外包文件进行测试。由于数据集的分布会对系统性能有影响，所以我们的测试方案分为两部分：

• 第一部分：数据首次上传。文件分块后均为非重复的数据块，随后分别用三种方案，即基础方案、增强方案、SGXDedup[12]，来对数据的上传时间进行测试，并比较三种方案的上传时间。

• 第二部分：数据后续上传。而后续上传的文件与首次上传的时候具有一定重复率，我们设置了三个重复率：60%、80%、100%，同时我们将文件的大小限定为 500M，随后分别用三种方案，即基础方案、增强方案、SGXDedup，来对数据的上传时间进行测试，并比较三种方案的上传时间。

4.2 测试环境

本次测试的测试环境如表 1：

表 1：测试环境详细信息

CPU	intel i5-9300
操作系统	Ubuntu18.04
内存	8G RAM
语言	c/c++

4.3 功能测试

我们的功能测试主要分为两个部分：数据的上传、数据的下载。

4.3.1 数据上传

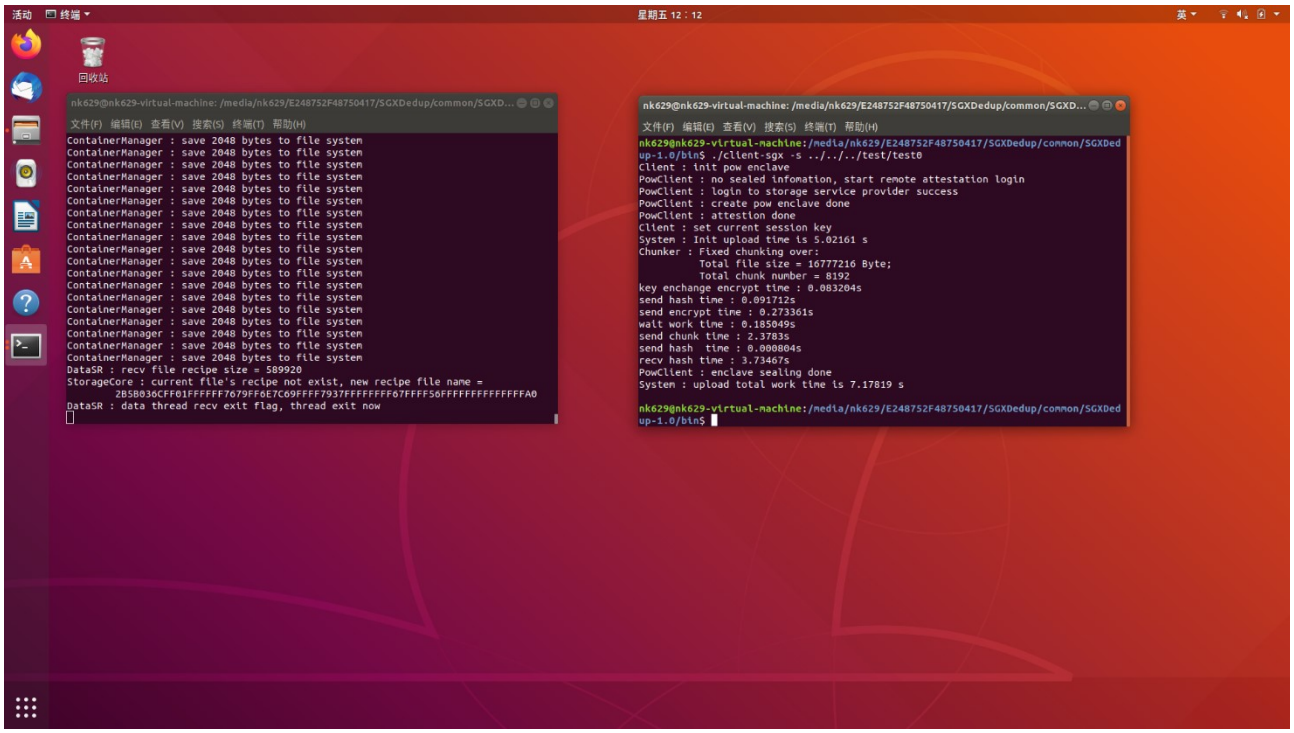

图 25：用户端上传文件

上传文件时，我们输入”-s”**命令**，选择 test 文件夹中的 test0 文件进行上传，从上图中我们可以看到上传的过程：用户首先初始化*Enclave*，并进行认证，随后成功创建*Enclave*。随后我们将文件分块，并利用我们所设计的方案将文件进行上传，每个部分的时间都会显示出来，总时间也会通过命令行输出至屏幕。（见图 25）

4.3.2 数据下载

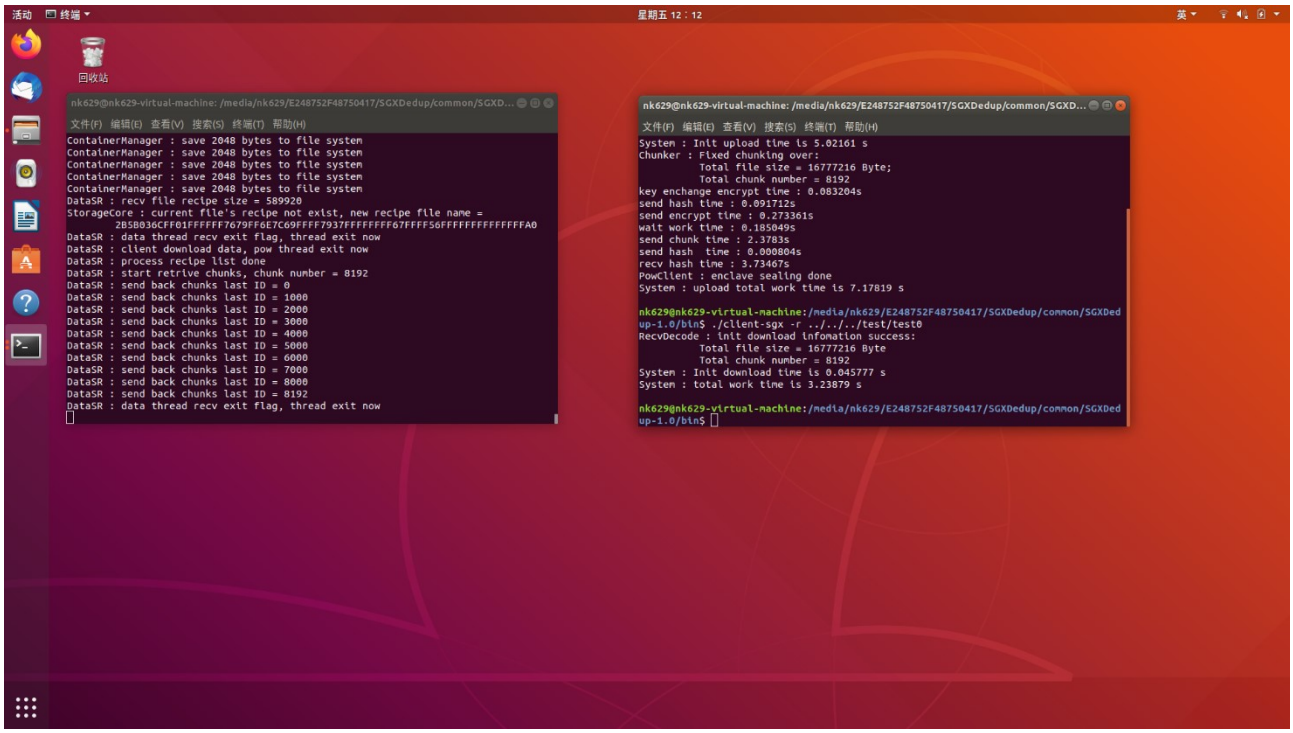

图 26：用户端下载文件

下载文件时，我们输入”-r”命令，选择 test 文件夹中的 test0 文件进行下载，从上图中我们可以看到下载的过程，下载时，所下载文件的大小以及块数都会显示在命令行中，随后文件下载的总时间也会通过命令行输出至屏幕。（见图 26）

4.4 测试数据与结果

4.4.1 测试方案一

数据首次上传时，我们将数据大小分为16M、100M、500M三种大小，随后分别用四种方案测试数据上传时间，并进行比较。

测试结果如表2：

表 2：测试方案一测试结果

方案 / 数据大小/M	基础方案	增强方案	SGXDedup
16	1.75264s	1.90853s	1.69077s

100	12.1390s	12.2383s	12.5159s
500	65.8559s	60.3953s	67.4293s

根据上述表格中的数据，我们可以得到上传不同大小文件的平均时间，并可以绘制图24。

从实验结果可以看出，当执行数据首次上传时，我们提出的基本和增强方案与SGXDedup相比，具有更低的时间开销。当文件大小分别为16M、100M、500M时，我们基本方案与增强方案的时间开销分别为SGXDedup时间开销的：103.66%、112.88%（16M）；96.987%、97.9782%（100M）；97.667%、89.568%（500M）。原因在于我们的方案有效减少了系统中的冗余计算。测试结果折线图见图27。

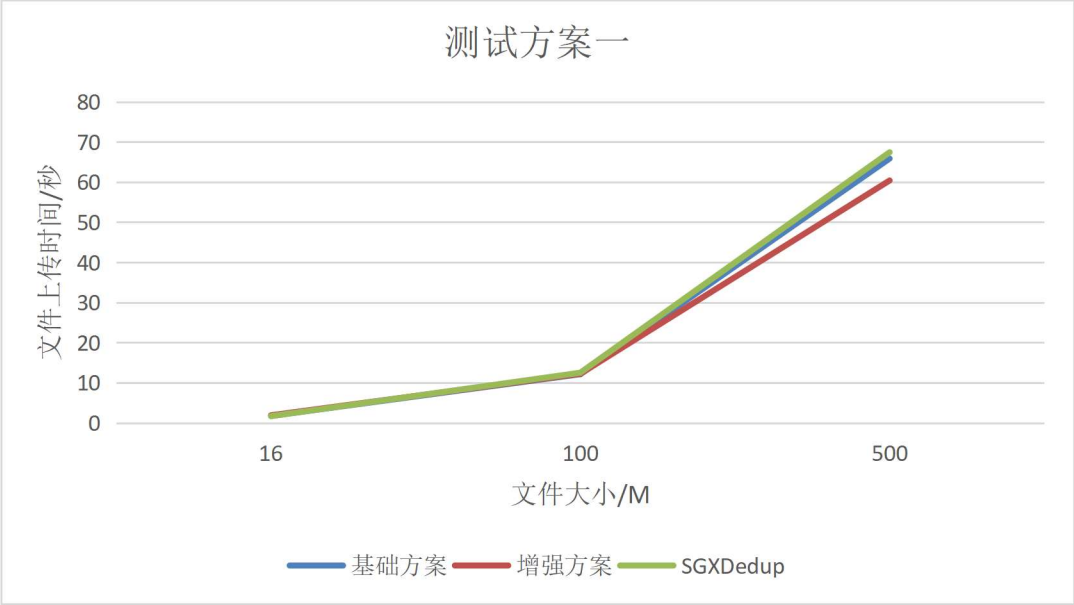

图27：测试方案一测试结果

4.4.2 测试方案二

数据后续上传时，我们限定数据大小为 500M，随后分别用三种方案测试数据上传时间，并进行比较。

测试结果如表 3：

表 3：测试方案二测试结果

方案 / 重复率	基础方案	增强方案	SGXDedup
60%	36.7838s	34.8389s	40.4246s
80%	25.5798s	19.6011s	27.0924s
100%	10.7365s	8.09708s	11.4160s

根据上述表格中的数据，我们可以得到上传每种大小文件的平均时间，并可以绘制图24。

从实验结果可以看出，我们的基础和增强方案与SGXDedup相比，具有明显的性能优势。当文件重复率为60%、80%、100%时，我们基本方案与增强方案的时间开销分别为SGXDedup时间开销的：90.994%、86.182%（重复率60%）；94.417%、72.349%（重复率80%）；94.048%、70.927%（重复率100%）。原因在于，当文件的重复率提高时，本文方案所减少的计算量将更加显著。综上，本文方案对于冗余计算的节省有效提高了系统的执行效率。测试结果折线图见图28。

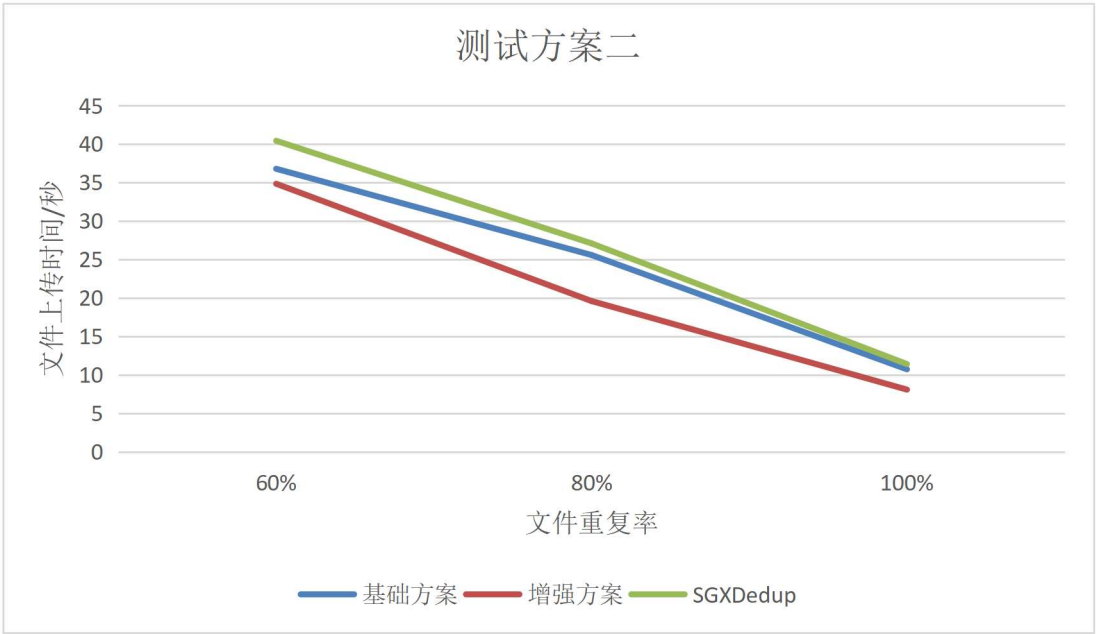

图28：测试方案二测试结果

5 应用前景

随着大数据时代的到来，数据呈现爆发式的的增长，数据开始表现出隐藏的价值。数据分析学，统计学，人工智能，经济学等学科都依赖于数据的质量和数量。面对如此多的数据，如何安全高效地存储数据成为了困扰研究人员的重点问题。目前来说，云存储作为一种广泛使用的方式，是一种可行的解决数据存储的方案。而一种安全可行的云存储框架必须解决两个问题:数据存储的效率和安全性。对于数据存储厂商来说，提高数据存储的效率就可以降低相关硬件成本，同时也减少了数据管理的难度，提高行业竞争力。对于产业界的相关要求，相关研究领域密切关注，提出了众多理论方案。目前，一种已经投入使用的方案是重复数据删除方案。重复数据删除依赖于这样的统计性质：大部分存储的数据都是冗余和重复的，通过识别数据特征并将冗余和重复的数据删除来提高存储效率。目前的研究和实践已经表明：重复数据删除是一种有效可行的数据缩减方法。

另一个颇受社会关注的方面就是云存储的安全性，在最初的存储方案中，很多云存储厂商选择直接存储用户数据的明文，毫无疑问这并不符合上面提到的两种要求，也导致了许多数据泄露问题，造成了重大的财产损失，甚至有些造成社会重大安全事件。近来，数据安全事件频发，数据安全成为了社会热点问题。如何保证云存储数据的安全性，常见的解决方案就是将数据加密，存储数据的密文，数据的安全性就依赖于相关的密码算法和数学难题。

因此，将存储效率和存储安全性兼顾的一种方案就是加密的重复数据删除，加密的重复数据删除保持了重复数据删除对加密数据的有效性，在保证了存储数据安全性的同时保证了数据存储的效率，对于云存储的发展有着非常重要的作用。

本作品基于上述思路，以 Intel 公司发展的 SGX 作为方案的硬件基础，设计出了一个云存储加密数据去重系统，在保证数据安全性同时，探索了数据存储的时间局部性，以此来提高数据存储的效率。根据方案测试效果来看，相比与之前的加密数据去重方案，本方案在去重效率和加密安全性方面均有提升。同时，因为方案的硬件基础 Intel SGX 已经投入使用，同时软件控制系统已经经过初步测试，本作品提供的加密数据去重解决方案已经在理论上具备了工程可行性。

6 结论

大规模云存储系统往往面临两个矛盾的需求：一方面，考虑到费用问题，云存储系统要压缩数据节省存储空间开销；另一方面，考虑到用户的隐私和数据的安全，云存储系统也应对用户的

数据加密后存储。本作品针对于云存储的这个需求，利用加密去重，设计了一个基于 Intel SGX 的加密去重系统，并对基础方案进行了增强。系统主要分为：用户端、服务端两个部分。

首先，用户端在飞地内将明文块用时段密钥加密，计算密文指纹及指纹签名，随后基于安全信道将这些内容发给服务端。服务端随即对签名进行验证，计算数据的标签，随后根据标签，利用数据集的时间局部性进行去重检测。如果数据重复，则只需将密钥发送回用户端；如果不重复，则将标签、经密钥加密后的数据以及是否重复的标志基于安全信道发回用户端。用户端接收到后先用通信密钥解密，再用密钥将非重复的数据块进行加密，通过安全信道传输至服务端，随后放至云端存储。基础方案的创新点在于：我们只部署了$Enclave$，减少了资源的开销，并且进行去重后加密，减少了冗余计算；增强方案的创新点在于：我们设置了时段密钥来搭建用户端与服务端之间的安全信道，并且充分利用数据集的时间局部性，对数据进行去重检测操作，提升了系统整体的效率。

本作品基于 SM3、SM4 以及 HMAC-SHA256。经数据测试，我们所设计的系统相比于现有方案，利用时段密钥，减少了加密去重时的计算开销，提升了整体的效率，为用户提供了一个既安全又高效的加密去重系统。

参考文献

[1] 熊金波,张媛媛,李凤华,等.云环境中数据安全去重研究进展[J].通信学报, 2016, 37(11)：169-180.
XIONG J B, ZHANG Y Y, Li F H, et al. Research progress on secure data deduplication in cloud[J]. Journal on Communications, 2016, 37 (11): 169-180

[2] XIA W, JIANG H, FENG D, et al. A Comprehensive Study of the Past, Present, and Future of Data Deduplication[J]. Proceedings of the IEEE, 2016, 104(9):1681-1710.

[3] DOUCEUR J R, ADYA A, BOLOSKY W J, et al. Reclaiming Space from Duplicate Files in a Serverless Distributed File System[C]. Proceedings of the International Conference on Distributed Computing Systems. IEEE, 2002: 617- 624

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

[5] XU J, CHANG E C, ZHOU J. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage[C]. Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, Hangzhou, China. 2013: 195-206.

[6] HALEVI S, HARNIK D, PINKAS B, et al. Proofs of ownership in remote storage systems[C]. Proceedings of the 18th ACM Conference on Computer and Communications Security, Chicago, Illinois, USA 2011: 491-500

[7] 基于国密 SM1 算法的 CPU 卡应用[J]. 闫永昭,郑金州. 现代电子技术. 2013(15)

[8] SM2 椭圆曲线公钥密码算法综述[J]. 汪朝晖,张振峰. 信息安全研究. 2016(11)

[9]Specification of SM3 cryptographic hash function (in Chinese) , http://www.sca.gov.cn/sca/xxgk/2010-12/17/1002389/files/302a3ada057c4a73830536d03e683110.pdf

[10] GB/T 32907-2016. 信息安全技术 SM4 分组密码算法[S]. 2016

[11] KEELVEEDHI S, BELLARE M, RISTENPART T. DupLESS: Server-aided encryption for deduplicated storage[C]. Proceedings of the 22nd USENIX Conference on Security, Washington, DC, USA 2013: 179-194.

[12] 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.

[13] Yang Z, Li J, Lee P P C. Secure and Lightweight Deduplicated Storage via Shielded {Deduplication-Before-Encryption}[C]//2022 USENIX Annual Technical Conference (USENIX ATC 22). 2022: 37-52.