云中透明的重复数据消除

弗雷德里克·阿姆克内赫特
曼海姆大学
德国曼海姆 68131
armknecht @ uni-Mannheim . de

延斯-马提亚斯·博利
欧洲电气公司实验室
德国海德堡 69115
延 斯-马 提 亚 斯 。
Bohli@neclab.eu

加桑·卡拉姆
NEC 实验室欧洲 69115
德国海德堡

弗兰克·优素福
欧洲电气公司实验室
德国海德堡 69115

摘要

Dropbox 和 Google drive 等云存储提供商在很大程度上依赖重复数据删除，通过只存储每个上传文件的一个副本来节省存储成本。尽管最近的研究报告称，全文件重复数据消除可实现高达 50%的存储缩减，但用户并未直接受益于这些节省，因为有效存储成本与提供给用户的价格之间没有透明的关系。

在本文中，我们提出了一种新的存储解决方案，即 ClearBox，它允许存储服务提供商向其客户透明地证明其存储的(加密的)数据的重复数据删除模式。通过这样做，ClearBox 使云用户能够验证他们的数据在云中占据的有效存储空间，并因此检查他们是否有资格获得降价等好处。ClearBox 对恶意用户和 rational 存储提供者是安全的，并且确保文件只能被它们的合法所有者访问。我们使用亚马逊 S3 和 Dropbox 作为后端云存储来评估 ClearBox 的原型实现。我们的发现表明，我们的解决方案与现有服务提供商提供的 API 一起工作，没有任何修改，并且获得了与现有解决方案相当的性能。

类别和主题描述符

c . 2.0[计算机通信网络]:常规-安全和保护。

泛称

安全，测量，实验。

关键词

云安全；安全的重复数据删除；重复数据消除的透明证明。

2015 年 10 月 12 日至 16 日，美国科罗拉多州丹佛市。

http://dx.doi.org/10.1145/2810103.2813630.

1. 介绍

云存储服务已经成为我们日常生活中不可或缺的一部分。随着越来越多的人操作多种设备，云存储为用户提供了一种方便的方式来存储、访问和无缝同步来自多种设备的数据。

众多提供相对廉价服务的竞争云存储服务也推动了云的日益普及。例如，Dropbox 为其客户提供免费的 2 GB 帐户，谷歌硬盘提供 100 GB，仅售 1.99 美元，而 Box.com 为其商业客户提供无限存储，每月仅售 12 欧元。这些有竞争力的优惠主要是由于现代硬盘价格大幅下跌，从每 GB 20 美元跌至 2014 年的每 GB 几美分[9]。

为了进一步增加利润，现有云存储提供商在存储客户数据时采用了积极的存储效率解决方案。也就是说，现有的云只存储由不同用户上传一次的重复数据(块级或文件级)，从而极大地节省了存储成本。最近的研究表明，跨用户重复数据消除可以在标准文件系统中节省 50%以上的存储成本[35，36]，在备份应用程序中节省高达 90-95%的存储成本[35]。

文献中有大量关于保护重复数据消除的建议(例如。[14，25，42])在云端。所有这些提案都有一个共同的目标，即让云提供商能够对其用户存储的加密数据进行重复数据消除。此类解决方案允许云提供商减少总存储，同时确保存储数据的机密性。

通过这样做，现有解决方案提高了云的盈利能力，但不允许用户直接从重复数据消除对其数据的节约中受益。请注意，云服务提供商根据其存储的数据量向其客户收费，而不管其数据显示的重复数据消除级别如何。但是，与在云中存储不太可能进行重复数据消除的个人文件的客户端相比，使用云作为备份存储的用户应该受益于数据减少(最多 90%)。在图 1 中，我们估计了与现有商品云提供商(如 Dropbox、Google drive 和微软 Onedrive)的每用户价格相比，重复数据消除给每用户带来的成本降低。我们的估计清楚地表明，对于那些数据经过大量重复数据消除的用户来说，有相当大的降价空间。

在本文中，我们解决了这个问题，并提出了一种新的安全存储解决方案，称为 ClearBox，它使云提供商能够透明且可验证地证明存储在云中的每个文件的重复数据消除模式。我们的解决方案依赖于 gate-

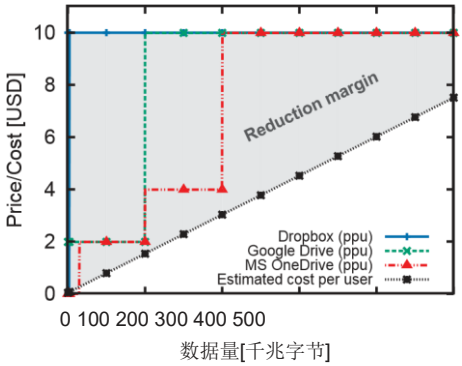


图 1:重复数据消除带来的成本降低与。商品储存供应商的价格。蓝色、绿色和红色曲线分别显示了 Dropbox、Google Drive 和 MS OneDrive 目前的收费价格。黑色虚线描述了数据经过重复数据消除后，亚马逊 S3 [1]中每个用户的估计存储成本。我们假设客户端存储的 50%的数据已经过重复数据消除[36]，数据属于另外两个云用户，并且客户端每天下载其帐户中存储的 0.96%的数据[34]。“缩减幅度”是指用户承担的价格与用户存储在重复数据删除后的有效成本之间的差额。“ppu”指的是每个用户的价格。

在(公共)云服务器上存储文件之前，协调跨用户基于文件的重复数据消除的方法。ClearBox 确保文件只能由合法所有者访问，抵御好奇的云提供商，并使云用户能够验证他们的加密文件在云中占用的有效存储空间(重复数据消除后)。通过这样做，ClearBox 为其用户提供了关于其数据所显示的存储节省的完全透明性；这使得用户能够评估他们是否为他们的钱获得了适当的服务和价格降低——尽管理性网关的目标是利润最大化。

ClearBox 可以与现有的云存储提供商(如亚马逊 S3 和 Dropbox)集成，无需任何修改，并推动了一种新的云定价模式，该模式考虑了数据所经历的重复数据删除级别。我们认为，这种模式不会威胁到云业务的盈利能力，相反，它会极大地激励用户将音乐和视频文件等大型流行数据存储在云中(因为流行数据的存储成本可能更低)。总之，我们在这项工作中做出了以下贡献:

具体实例化:我们描述了一个名为 ClearBox 的云存储方案，它采用了一种新的基于加密树的累加器 CARDIAC，以对数时间(相对于上传文件的客户端数量)证明云中存储的每个文件的重复数据消除模式。ClearBox 还利用所有权证明[23，27]和自动过期的 URL 命令，以便有效地管理对存储在云中的文件的访问控制。

安全性分析:我们为 ClearBox 提供了一个模型，并根据我们的模型分析了我们的建议的安全性。也就是说，我们展示了 ClearBox 使用户能够验证他们的文件所经历的重复数据删除，尽管一个理性的提供商旨在最大化其在系统中的利润。此外，我们展示了 ClearBox 抵抗恶意客户端和好奇的存储提供商。

原型实现:我们使用亚马逊 S3 和 Dropbox 作为后端云存储，实现并评估了一个基于 ClearBox 的原型，我们表明，我们的提议不会损害云用户的体验，并且在客户端之间编排文件操作时不会在网关上产生可容忍的开销。

本文的其余部分组织如下。在第 2 节中，我们介绍了我们的模型和目标。在第 3 节中，我们介绍了 ClearBox 并分析了它的安全性。在第 4 节中，我们评估了一个基于 ClearBox 与亚马逊 S3 和 Dropbox 集成的原型实现。在第 5 节中，我们回顾了该领域的相关工作，并在第 6 节中总结了论文.在附录 A 和 B 中，我们提供了关于我们计划的更多见解。

2. 模型

在第 3 节给出 ClearBox 的完整描述之前，我们在这一节介绍我们的系统和安全模型。

2.1 系统模型

ClearBox 由 C1、C2，... 有兴趣将其文件存储在存储提供商 S 处，以便每个客户端了解其文件经历的重复数据消除级别，即外包相同文件的客户端数量。由于现有的存储提供商不报告存储数据的重复数据消除模式，因此一种简单的方法是依靠分散的方案，即用户在将数据存储到云中之前协调文件上传；然而，这种分散的方案需要用户之间的交互，并且不太可能随着存储相同数据的用户数量的增加而扩展[41]。这就是为什么 ClearBox 使用了一个额外的网关 G，它连接用户和云，并在将数据存储到云上之前编排重复数据删除。请注意，G 是一个逻辑上集中的实体，可以很容易地使用分布式服务器进行实例化。我们认为我们的模型是通用的，并且捕捉了许多实际的部署场景。例如，G 可以是一个独立的服务，通过在现有的公共云上执行重复数据消除来提供更便宜的云存储；或者，G 可以是 S 自己提供的服务，以便为现有的云存储服务提供差异化，等等。

在我们的方案中，我们假设 G 拥有一个由 S 托管的帐户，并协调基于文件的跨用户重复数据消除。通过这样做，G 可以向其用户提供完全透明的存储节省，这是由于其数据表现出的重复数据消除。例如，通用电气可以为数据经过高度重复数据消除的客户提供价格优惠(例如，。如果文件在至少 n 个实体中进行了重复数据消除，则可享受 50%的折扣)。或者，G 可以在存储重复数据消除文件的用户之间公平分配存储成本；例如，假设有 n 个客户端都存储相同的文件 f，那么每个客户端可以有效地被收取存储 f 的成本 n1 的一部分。请注意，这些减少不会威胁到通用汽车的盈利能力，通用汽车仍然可以从各种提供的服务中轻松获利(例如，。故障恢复能力、移动接入、跨设备的数据同步)。相反，我们认为，提供与存储相同文件的其他客户端共享存储成本的选项，与其他竞争对手相比，可能会提供明显的优势。然而，提供商显然不热衷于与用户分享部分利润，并且可能无法正确报告重复数据消除带来的成本降低。因此，这里的挑战在于透明且高效地证明数据存储成本(即，。包括重复数据消除节省),而存储提供商可能无法正确报告其存储的数据的重复数据消除(和访问)模式。

为了与现有存储提供商的操作保持一致，我们假设时间被划分为等长的时期 Ei，例如。12 小时[1]。在每个时期结束时，客户端都会从 G 处收到其文件和重复数据消除模式的列表。给定文件的重复数据消除模式是指云中存储相同重复数据消除文件的用户数量。

与现有的存储提供者类似，ClearBox 支持以下操作:放、获取、删除。此外，ClearBox 支持两个附加协议，即证明和验

证，用于证明/验证文件经历的重复数据消除模式。我们从描述这些协议开始。在这里，表达

【P1:1；。。 *pn:inn]→[P1:out 1；…,。Pn : outn]*

Pn 运行一个交互式协议 π，其中 ini 和 outi 分别表示 Pi 的输入和输出。

卖出协议。

Put 协议在网关和旨在上传文件 f 的客户端 C 之间执行。最初，客户端从文件中获得一个密钥 kFID，他用这个密钥将 f 加密成 f，最终通过 g 上传到 S。接下来，双方都导出一个文件标识 FID，它将作为 f 的一个实际上唯一的句柄。网关 G 在内部维护一个集合 F，该集合包含对(FID，CFID)，其中 CFID 是注册到 FID 引用的文件的客户端集合。如果没有客户端注册到文件标识为 FID 的文件，它会认为 F 不包含表单元素(FID，∑)。鉴于此，网关检查是否有任何用户已经注册到该文件，即，。if (FID，CFID) ∈ F。如果是这样，它将 C 插入 CFID。否则，将创建一个新的集合 CFID = {C}，并将其插入到 f 中。此外，客户端获得一个验证标签 τ，该标签随后允许验证 g 生成的证明。

放 ：[C:f；g:F；s:⊥]→

[C : FID，kFID，τ；G : FID，F；s:f∫]

当我们需要指定特定客户端 C 的验证标签时，我们写 τC。*获取协议。*

当客户端 C 想下载一个文件 f 时，它用 G 发起这个协议，并发送相应的文件 ID FID。网关首先检查 F 是否包含条目(FID，CFID)。在正的情况下，它进一步验证 C ∈ CFID。如果验证通过，将采取措施，最终客户端可以从 S 下载加密文件 f ,*并用 kFID 将其解密为 f。请注意，我们没有指定额外的令牌来验证客户端，因为我们假设客户端和 g 之间存在经过验证的通道。

得到 ：[C : FID，kFIDg:F；s:f∞]→

[C:f；g:⊥；学生:⊥]

删除协议。

该协议允许客户端删除文件，或者更准确地说，取消注册。为此，客户端将文件 ID FID 发送到网关，网关检查是否(FID，CFID)←F 为某个集合 CFID，是否 C ∈ CFID。如果不是这种情况，请求就被忽略，c 接收 f = ⊥. 否则，CFID 将在下一个纪元开始时更新为 CFID。如果 CFID 通过此操作变为空，这意味着没有用户再注册到此文件。因此，G 可以请求 S 删除该文件。

删除 ：[丙:FIDg:F；s:f∞]→

[c:⊥；g:F；学生:⊥]

证明协议。

仅由网关执行的证明过程有两个目的。一方面，它为给定的文件标识 FID 和纪元 E 生成基数证明，证明了|CFID|的上限，即在此纪元内注册到此文件的客户端数量。另一方面，它还包括给定客户 C 关于 CFID 的成员资格证明。形式上，我们有:

A ←证明(FID、E、CFID、C)。

证据甲，即。“证明”的输出包含对|CFID|上限的声明、CFID 的紧凑摘要以及可能的附加信息，以便客户端 C ∈ CFID 可以使用后续的“验证”来确信集合 CFID 的上限及其成员资格。如第 2.2 节所述，|CFID|的上限在我们的模型中是充分必要的。也就是说，G没有动力报告更大的价值|CFID|因为这导致G对客户收费过低，因此利润减少。因此，为了增加利润，G 的兴趣是在每个纪元结束时要求最小可能的上限。

验证协议。

在 ClearBox 中，客户可以验证(使用验证协议)由证明协议生成的证明，以确认他们是文件用户集的一部分，并验证文件用户总数的上限。验证算法由客户端执行，并使用在 Put 过程中生成的验证标记。

它输出接受或拒绝，以指示证据是否被接受。

接受|拒绝←验证(FID，E，A，τ)。

2.2 安全模型

在后续文章中，我们假设客户端和网关之间的通信经过身份验证，以提供不可否认性，并且在需要时进行加密。如前所述，我们假设时间被分成一系列的时期 E1，E2，...,。其中 $E \leq E'$的意思是 E 发生在 E 纪元之前，或者两者都指同一个纪元。如果没有另外提及，我们将总是提及过去发生的时代。此外，我们假设各方都是同步的。也就是说，在每个时间点，所有各方共享当前时代的相同观点(例如，。如果纪元由递增的计数器表示，则其索引号)。

在每个时期，在网关 G 和客户端之间可以执行几个协议运行。协议运行由一个四元组 pi = (C，prot，(in)，(out))表示，其中 C 是客户端，prot 表示协议 Put，Get，Delete 之一，in 表示 C 的输入，out 表示其输出。对于任何过去的纪元 E，我们用 PE 表示在这个纪元内发生的所有协议运行。这里，我们将自己限制为完整的协议运行，即，。这些都没有中止。我们假设这些在纪元内是唯一有序的。p') or $p' < p\, p < p'$这是对 $\mathbb{P}_E = \{p_1, \ldots, p_\ell\}$而言，其中表示在这个时期内运行的协议总数，它适用于$p, p' \in \mathbb{P}_E$ with (当 p 发生$p \neq p'$在之前(当 p 最先发生时)的任何一个协议。我们以简单的方式将这个顺序扩展到所有协议运行的集合。更准确地说，对于 $E \neq E'$ with $E < E'$的任何两个时代，它认为$p < p'$代表所

有的 p ∈ PE 和所有的 $p' \in \mathbb{P}_{E'}$。最后，我们用 $\mathbb{P}_{\leq E} = \bigcup_{E' \leq E} \mathbb{P}_{E'}$来表示 发生在纪元前的所有协议运行(包括纪元前)都类似地定义了 P<E。

定义 1(文件注册)。
我们说，如果存在满足以下两个条件之一的 p = (C，Put，(f)，(FID，kFID，τ)) ∈P≤E，则客户端 C 在某个时间 E 注册到文件 ID FID:

1. *p ∈ PE*
2. *p ∈ P<E，对于整个 $p' \in \mathbb{P}_{<E}$ with $p < p'$，它认为 $p' \neq (C, \text{Delete}, (\text{FID}), (\perp))$.*

我们用 CFID(E)表示在 E 时代注册到 FID 的所有客户的集合。如果所考虑的时代从上下文来看是清楚的，我们就简单地写 CFID。最后，我们用 Ereg(E，C，f，FID，τ)C 在纪元 E 内注册到 FID 的事件，其中 f 是上述协议运行 p 中指定的文件(即上次上传中使用的文件)。显然，它认为 C ∈ CFID(E)当且仅当 Ereg(E，C，f，FID，τ)对某些文件 f 成立。

Def.1 中的第一个条件意味着，如果 C 在 E 纪元内上传了文件，他将被注册到该文件，无论他是否在同一纪元的稍后请求删除该文件。第二个条件意味着，如果上传发生在前一个时期，C 一定没有在更早的时期要求删除它。还要注意，在这里，如果 C 要求在纪元 E 内删除它，他仍然在这个纪元内注册到 FID。同样，如果客户在一个时期内注册了文件，我们允许他在这个时期内的任何时间下载文件。

我们现在准备正式定义正确性和可靠性。*正确性*

对于正确性，有两个要求。首先，一个用 Put 上传文件的用户，必须能够在他注册到这个文件的那些时期用 Get 获得它。由于文件是由它们的文件标识来标识的，我们首先处理文件标识生成过程。也就是说，我们说文件标识生成过程创建 ε-唯一的
文件标识，如果它适用于任何两个协议运行(C，Put，(f)，(FID，kFID))和$(C', \text{Put}, (f'), (FID', k'_{FID}))$
$= FID' | f = f'] = 1 Pr[$国际开发署， (1)
$'| f \neq f'] \leq \varepsilon . Pr[$ FID (2)
第一个条件意味着相同的文件总是导致相同的文件标识，而不同的文件导致相同的文件标识的概率最多为 ε。这允许我们为我们的方案定义正确的访问权限:

定义 2(正确访问)。*假设文件标识生成过程是ε唯一的。我们说，如果该方案适用于任何客户端 C、任何纪元 E 和任何文件f，并且如果事件Ereg(E、C、f、FID)适用，并且如果存在运行$(C, \text{Get}, (FID, k_{FID}), (f')) \in \mathbb{P}_E$的协议，则该方案提供正确的访问:*
$Pr[f' = f] \geq 1 - \varepsilon.$ (3)

之所以不能要求概率等于 1，是因为在某种概率 ε 下，两个不同的文件可能会得到相同的文件 ID。在这些情况下，正确性再也无法保证了。

定义 3(正确证明)。*设 FID 为任意文件 ID，E 为任意纪元。此外，假设 A 是网关为文件 FID 和纪元 e 生成的证明。即 A ←证明(FID，E，CFID)。此外，让 bd 表示在 A 中声明的 |CFID(E)|的上限，让 C 是任意客户。*

如果在 Ereg(E，C，f，FID，τ)和|CFID(E)| ≤ bd 的条件下，认为:

pr[接受←验证(FID，E，A，τ)]] = 1。(4)健全性

我们假设每一方(客户端、网关、服务提供商)都可能行为不端，但目标不同。因此，健全性要求针对所有这些攻击者实现安全性。在接下来的内容中，我们将激励每个攻击者类型，并定义相应的安全目标。

恶意客户端:原则上，我们假设客户端可能出于各种目的而任意恶意:破坏服务、拒绝行为以及非法访问文件。前两个可能会被标准机制所阻挠，比如认证通道。因此，我们只关注第三个:如果用户之前已经将文件 f 上传到 G，并且他仍然注册到 G，那么他必须只能通过 Get 访问文件 f。

定义 4(安全文件访问)。*我们说，该方案提供 ε-安全的文件访问，如果它适用于任何客户端 C、任何纪元 E 和任何 $(C, \text{Get}, (FID, k_{FID}), (f')) \in$的任何文件 f*
当 Ereg(E，C，f，FID，τ)不成立时，则
$Pr[f' = \perp] \geq 1 - \varepsilon.$ (5)

理性网关:我们假设网关和服务提供商都是理性实体，例如。类似的假设见[43]。理性的意思是，只有当网关和存储提供商的策略增加了他们在系统中的利润时，他们才会偏离协议。请注意，网关可以访问服务提供商见证的所有信息(例如，。访问存储的文件)。这意味着任何由 rational 服务提供者发起的攻击都可能由网关执行。同样，一个理性的网关不能通过与服务提供商勾结来发动更强的攻击。因此，在给定一个合理的网关的情况下，我们将我们的分析限制在我们方案的安全性上。

回想一下，网关执行两种类型的交互:证明注册客户端的数量和处理客户端的文件。因此，我们看到了两种不同类型的策略，它们可能允许网关增加其在系统中的优势:(I)向客户端多收费和(ii)非法提取文件内容。

关于第一个策略，如果一个 rational gateway 能够说服客户端使用更低级别的重复数据消除，那么这个 gateway 可以收取更高的价格。另一方面，关于第二策略，网关可以尝试导出关于上传文件内容的信息；请注意，获取关于用户数据的信息对于网关来说在经济上是有益的，因为存储的数据可能被滥用。卖了。因此，需要确保两个安全目标:安全证明和数据机密性，这是我们接下来要形式化的。

定义 5(安全证明)。*设 FID 为任意文件 ID，E 为任意纪元。此外，假设 A 是网关为文件 FID、纪元 E 和客户端 C 生成的证明，即 A ←证明(FID，E，CFID，C)。此外，让 bd 表示在*

a 中声明的|CFID(E)|的上限。我们说，如果 Ereg(E，C，f，FID，τ)或|CFID(E)| > bd 暗示

pr[接受←验证(FID，E，A，τ)|] ⩽ ε。(6)

请注意，在我们的模型中，报告|CFID(E)|的较大值不符合 rational gateway 的利益，因为这将允许客户要求进一步降低成本。

关于上传数据的机密性，请注意，标准语义安全加密方案不能用于我们的上下文，因为它们有效地阻止了数据的重复数据删除[14，25，42]。适合重复数据删除的方案为同一消息的多次加密产生相同的密文，被称为消息锁定加密(MLE)[13，15]。因此，在我们的案例中，可实现的安全性是 MLE 方案的安全性。

为了描述最大似然估计方案的安全性，我们采用了[15]中引入的安全概念，该概念保证了在选择分布攻击下的隐私。

MLE 方案由一组算法(setup、keygen、enc、dec、tag)组成，其中 setup 生成(公共)参数 P，key generation 生成给定 P 的密钥 k 和消息 f；最后，enc 和 dec 是用密钥 k 加密和解密 f 的算法。标签生成算法标签为密文创建一个标签。在我们的模型中，这是由唯一文件标识符 FID 涵盖的，在隐私概念中不发挥作用。

消息空间由算法 M 给出，该算法根据分布对消息 M∞{ 0，1 } *进行采样，并且可以提供附加的上下文信息。消息采样必须是不可预测的，即，。给定上下文信息，预测 M 的输出的概率可以忽略。最大似然估计方案的安全性由以下实验定义:PRV-CDAAMLE，M，不可预测采样算法 M 和对手 A [15]:

1. 环境随机生成一个参数集 P 和一个位 b ← {0，1}。
2. 环境采样一组消息 M 和上下文信息 Z: (M，Z) ← M。由于相同的消息将导致相同的加密，这里的限制是所有消息都是不同的。
3. 对于 M 中的所有消息 M[i]，环境使用 MLE 方案加密消息 M[i]:

$$C0[i] \leftarrow enckeygen(M[i])(M[i])并$$

生成一个相同长度的随机字符串:

$$C1[I]；\{0，1\}|C0[i]|。$$

4. 对手获得集合 Cb 并输出对 b 的猜测:b ← A(P，Cb，Z)
5. 如果 b 等于 b，对手就赢了.在这种情况下，实验返回 1，否则返回 0。

定义 6(数据保密)。*我们说消息采样算法 M 的 MLE 方案是安全的，如果对手在赢得 PRV$-CDAAMLE 中的优势，M 实验是可以忽略的，即。*

$$^{A}_{MLE,\mathcal{M}}\Big] - 1 \leq 2 \cdot Pr\left[1 \leftarrow \text{PRV\$-CDA 内格尔}(\kappa)\right.,$$

对于安全参数κ。

请注意，G 和 S 都必须知道文件大小和文件的下载模式，以便进行正确的核算。因此，隐藏这些信息不能成为我们目标的一部分。

命令	描述
创建桶	创建一个桶 B
放(乙，FID)	上传一个文件 FID 到 B
获得(乙，FID)	从 B 下载一个文件 FID
删除(乙，FID)	从 B 中删除一个文件 FID
生成器 1(命令，测试)	生成一个在时间 t 过期的网址，支持 PUT，GET，DELETE。

表 1:亚马逊 S3 和谷歌云存储公开的示例 API。**COMMAND 指的是一个 HTTP 命令，比如 PUT(B，FID)。**

2.3 设计目标

除了上述安全目标之外，我们建议的解决方案应该满足以下功能需求。与用户直接与 s 接口的标准解决方案相比，ClearBox 应该在当前服务提供商提供的 API 内工作，而不会降低用户所见证的性能。类似于现有的云存储提供商，如亚马逊 S3 和谷歌云存储，我们假设 S 向其客户端公开了一个标准界面，该界面由一些基本操作组成，如存储文件、检索文件、删除文件、生成用于发送存储/检索的 HTTP 命令的签名网址等。(参见。表 1)。在适当的情况下，我们还讨论了 S 是商品云服务提供商的情况，并公开了一个更简单的界面，例如，。不允许使用网址存储文件。

此外，我们的解决方案应该随着用户数量、文件大小和上传文件的数量而扩展，并且在每个时期结束时验证用户文件的重复数据消除模式时，应该会给用户带来可承受的开销。

3. 白盒

在这一节中，我们介绍我们的解决方案，并根据第 2 节中概述的模型分析其安全性。

3.1 概观

ClearBox 确保对数据正在进行有效重复数据消除的用户的存储消耗进行透明证明，而不会损害存储数据的机密性。

为了向客户证明重复数据消除模式，一个简单的解决方案是网关发布与每个重复数据消除文件相关联的所有客户端的列表(例如。在公共公告板上),使得每个客户可以首先检查(I)他是否是该列表的成员，以及(ii)该列表的大小是否对应于存储该文件的网关所提供的价格降低。除了该解决方案不可扩展的事实之外，不公布其客户及其文件的整个列表同样符合 G 的利益；例如，竞争者可以以其他方式了解关于由 G 提供的服务的信息(例如，。总营业额)。

为了解决这个问题，ClearBox 采用了一种新颖的基于 Merkle 树的密码累加器，该累加器由网关维护，以有效地累加在同一时间周期内注册到同一文件的用户的标识。我们的构造确保每个用户都能在每个纪元结束时检查他的 ID 是否正确累积。此外，我们的累加器对累加值的数量进行上限编码，从而使任何与累加器相关联的合法客户端都能够验证(相对于上传同一文件的客户端数量，以对数时间表示)该界限。

显然，要求网关发布所有存储文件的所有累加器的详细信息的解决方案不会随着云中存储的文件数量而扩展。这就是为什么 ClearBox 依赖于一种概率算法，这种算法有选择地揭示每个时期的文件累加器数量的细节。但是，如果网关可以

选择发布哪些文件累加器，那么 G 很容易通过只为所选文件创建正确的累加器来作弊，同时误报剩余文件的重复数据消除模式。在 ClearBox 中，选择在每个时期发布哪些累加器是由外部随机性来源播种的，这种随机性无法预测，但可以由任何实体验证。我们展示了如何使用比特币有效地实例化这样一个安全的随机性来源。这使得任何客户都能够验证取样是否正确，以及他是否获得了承诺的降价——而不会给 G 带来任何不当行为的好处。

ClearBox 通过在访问内容时利用自行过期的 URL，对共享文件实施细粒度的访问控制。也就是说，每当用户希望访问给定的资源时，网关会动态地为该资源生成一个 URL，该 URL 会在一段短暂的时间后过期。如表 1 所示，现有的云 API 支持动态生成过期的 URL。通过这样做，ClearBox 不仅确保了 G 可以限制对存储在云上的数据的访问，还使 G 能够跟踪其用户的访问模式(例如，。用于计费)。ClearBox 还依赖于一个不经意的服务器辅助密钥生成协议，以确保存储的文件用依赖于文件散列和网关秘密的密钥加密。当消息内容是可预测的时，这防止了暴力搜索攻击，但也确保了不知道文件散列的好奇网关/存储提供者不能获得解密文件所需的密钥(因为密钥生成协议是不经意的)。为了防止恶意用户获取文件散列(例如，。通过盗窃/恶意软件)但不拥有完整文件，ClearBox 使用加密文件的所有权证明来验证给定用户确实拥有完整文件。

在接下来的小节中，我们将更详细地介绍 ClearBox 的各个部分，首先是我们将在解决方案中使用的构建块，然后是协议规范，最后是它的安全性分析。

3.2 积木

在详细描述 ClearBox 之前，我们首先概述将在 ClearBox 中使用的构建块。

3.2.1 心脏的

密码累加器(例如。[12，20–22，30，33，39])基本上构成单向成员函数；这些函数可用于回答给定候选是否属于集合的查询。

在接下来的内容中，我们将展示如何构建基数证明累加器(CARDIAC)，该累加器利用 Merkle 树来有效地提供成员资格证明和最大集合基数的(非公开)证明。正如我们在第 3.3 节中所展示的，需要基数证明来向用户证明文件重复数据消除模式。

Merkle 树是一个二叉树，数据存储在树叶中。让 ai，j 表示树中位于第 I 层和第 j 个位置的节点。这里，级别指的是到叶节点的距离(以跳为单位)；显然，叶节点位于距离 0 处.另一方面，级别内的位置是从位置 0 开始从左到右递增计算的；例如，级别 1 最左边的节点由 a1，0 表示.在 Merkle 树中，中间节点被计算为它们各自子节点的散列；即 ai+1，j = H(ai，2j，ai，2j+1)。

给定一棵树的高度，CARDIAC 通过将集合 X 的元素分配给叶节点(从位置 0 开始)来累积这些元素，而剩余的叶节点/用不同的符号 0 填充。我们称之为零叶。可以从零叶计算的节点起着特殊的作用。我们将它们称为开放节点，因为它们的价值是公开的。更正式地说，$a_{0,|X|},\ldots,a_{0,2^\ell}$的零叶是打开。而且，如果 ai，2j 和 ai，2j+1 都是开的，那么 ai+1 也是开的，j = H(i + 1，ai，2j，ai，2j+1)。

我们现在概述由 CARDIAC 提供的主要算法(Acc、ProveM、VerifyM、ProveC、VerifyC)。请注意，ProveM 和 ProveC 将用于实现证明，同样，VerifyM 和 VerifyM 也将用于实例化验证。

$\delta \leftarrow Acc(X)$。该算法将集合 X 的元素累加成一个摘要 δ。在“心脏”中，δ 对应于修改后的 Merkle 树的根节点($a_{\ell,0}$)和树的高度(即$\delta = \mathrm{H}(a_{\ell,0},\ell)$)的散列。

$\pi M \leftarrow ProveM(X, X)$。给定集合 X 和元素 x ∈ X，该算法输出隶属度 πM 的证明，断言 x ∈ X。*πM 由修改后的 Merkle 树中 x 的兄弟路径和根*$a_{\ell,0}$*组成。*

VerifyM(δ，x，πM)。给定 δ，元素 x，它的兄弟路径和 the root $a_{\ell,0}$，当且仅当 δ = H 时，该算法输出真，其中$(a_{\ell,0},\ell)$是兄弟路径的长度，x 的兄弟路径与根$a_{\ell,0}$匹配。

$\pi C \leftarrow ProveC(X)$。给定集合 X，该算法输出证明 πC，证明集合的大小|X|的上界。这里，πC 由 X 的大小、最右边的非零元素 a0、| X |-1、它的兄弟路径和$a_{\ell,0}$树的根组成。

VerifyC(δ，C，πC)。给定摘要 δ、集合 X 的基数|X|以及由 a0、| X | 1、其兄弟路径和$a_{\ell,0}$树的根组成的基数证明，如果满足以下条件，则该算法输出 true:

- 它认为$\delta = \mathrm{H}(a_{\ell,0},\ell)$是兄弟路径的长度，
- 它认为 a0，| X |-1 的$2^{\ell-1} < |X| \le 2^{\ell}$,[5]兄弟路径与根$a_{\ell,0}$匹配，并且
- 兄弟路径上所有打开的节点都包含正确的值。

这里，VerifyC 假设在级别 0 中位置大于| X |-1 的所有叶子都用不包含在 X 中的 0 元素填充。

在附录 A 中，我们表明，除了成员证明之外，我们的心动实例化还提供了一个证明，即具有根$a_{\ell,0}$的 Merkle 树中的非零叶的数量最多为|X|。

3.2.2 与时间相关的随机性

ClearBox 利用了一个依赖于时间的随机性发生器 $T \to \{0,1\}^\ell$种子，其中 T **表示一组离散的时间点。**简而言之，GetRandomness 产生的值是不可预测的，但可以公开重建。

更正式地说，让 cur 表示当前时间。$\{0,1\}^\ell$在输入 t ∈ T 时，如果 t ≤ cur，则 get 随机性输出⊥.种子中的一致随机串，否则 get 随机性输出。我们说，如果只要 t < cur，就不能用明显好于$2^{-\ell}$种子的概率来预测 getrandom(t)的输出，那么 getrandom 是安全的。

类似于[10]，我们通过利用比特币的功能来实例化 GetRandomness，因为后者提供了一种方便的方法(例如。通过应用编程接口)来获得依赖于时间的随机性。

[5] 这个条件最小化了树中开放节点的数量，从而减少了证明/验证成员资格和基数的工作量。

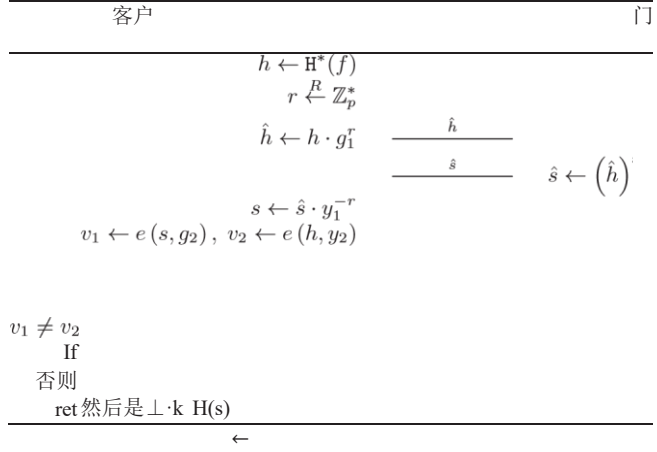


图 2:基于盲 BLS 签名的服务器辅助密钥生成模块。这里，γ1、γ2 分别是 γ1 和 γ2 的 p 阶、g1 阶、g2 阶生成元的两个群，配对函数 e:γ1×γ2→γT，散列函数 H∫:{ 0，1 }∫→γ1，秘密密钥 x ∈Zp∫，对应的公钥 y1 = g1x，y2 = g2x。

也就是说，比特币依靠区块(一种基于哈希的工作证明概念)来确保交易的安全性。具体来说，比特币对等体需要找到一个随机数，当用最后一个块哈希和累积最近交易的 Merkle 树的根进行哈希运算时，整体哈希小于 256 位阈值。

调整比特币中区块生成的难度，使区块平均每 10 分钟生成一次；在[29]中表明，比特币中的区块生成遵循参数 p = 0.19 的移位几何分布。最近的研究表明，可以在比特币上建立一个公共随机性信标——每 10 分钟输出 64 位 minentropy。

考虑到这一点，GetRandomness 然后展开如下。在输入时间 t 时，GetRandomness 输出自时间 t 以来在比特币区块链中出现的最新区块的散列。显然，如果 t > cur 对应于未来的某个时间，那么 get 随机性将输出⊥，因为未来出现的比特币块的散列是无法预测的。另一方面，对于 t ≤ cur 的值(即，。*t 是过去的)通过获取以前的比特币块的散列。*通过这种方式，GetRandomness 使不可信的一方能够对随机性进行采样，而无法提前预测结果。

get 随机性的目的是确保网关证明重复数据消除模式的文件选择是随机选择的，而不是预先计算的。

虽然使用依赖于时间的随机性来源是一个可行的选择，但其他选择也是可以想象的。例如，可以将文件的选择与菲亚特-沙米尔试探法[26]相结合，以确保从网关的角度随机选择选择，并将其与工作证明相结合。如果生成有效工作证明的时间大约是一个时期的持续时间，恶意网关可能无法预计算这些选择。我们把研究外部随机性来源的可行替代方案作为未来研究的一个有趣方向。

3.2.3 服务器辅助密钥生成

ClearBox 采用了一个改编自[14]的遗忘协议，该协议在客户端和网关之间执行，以生成加密存储文件所需的密钥。与[14]不同，我们的协议不依赖于 RSA，而是基于盲 BLS 签名[17，18]。虽然 BLS 签名的验证比 RSA 签名更昂贵，但 BLS 签名比 RSA 签名要短得多，而且网关计算速度更快。

如图 2 所示，我们假设在设置时，网关选择两个 p 阶的组 γ1 和 γ2，以及一个可计算的双线性映射 e:γ1×γ2→γt。此外，网关选择私钥 x ∈ Zp，对应的公钥 y1 = $g_1^x \in \Gamma_1$和 $y_2 = g_2^x \in \Gamma_2$。设 H∫:{ 0，1 }∫→γ1 是一个加密散列函数，它将任意长度的位串映射到 γ1 中的组元素.在存储文件 f 之前，客户端计算 H←H∫(f)，给定随机选择的 r ∈ Zp，通过将其与 g1r 相乘来隐藏它，并将隐藏的散列发送到网关。后者在接收到的消息上导出签名，并将结果发送回客户端，客户端计算未隐藏的签名并验证:e(s，g2) = e$\left(h^x g_1^{rx} g_1^{-rx}, g_2\right) = e\left(h, y_2\right)$.

然后，加密密钥被计算为未加密签名的散列:k ← H(s)。这种密钥生成模块有两个好处:

- 因为协议是不经意的，所以它确保网关不了解任何关于文件的信息(例如。关于文件散列)。另一方面，该协议使客户端能够检查由网关执行的计算的正确性(即，。验证网关的签名)。正如我们稍后所展示的，需要进行这种验证，以防止 rational G 将同一文件的用户注册到重复数据消除级别降低的不同文件版本。
- 通过将网关包含在密钥生成模块中，可以通过对 g 的密钥生成请求进行限速来减缓对可预测消息的暴力攻击。请注意，与[14]类似，该方案并不阻止好奇的 G 对可预测的消息执行暴力搜索，获取散列和相应的密钥 k。在这个意义上，我们的方案提供的安全性降低到现有的最大似然估计方案的安全性。第 3.4 节)。

3.2.4 所有权证明

上述服务器辅助密钥生成确保了没有配备正确文件散列的对手不能获得文件加密密钥 k。然而，错误地获得文件散列的用户将能够主张文件所有权。在这种情况下，网关可以使用所有权证明(PoW)来确保客户端拥有整个文件(而不仅仅是其散列)[16，23，27]。

在本文中，我们依赖于 Halevi 等人的 PoW。[27]据我们所知，这使得网关的计算和存储开销最低[16]。这个 PoW 在文件 f 上计算一个 Merkle 树，这样 Merkle 树的根就构成了对文件的承诺。在续集中，我们用 MTBuf(f)表示 Merkle 树的根，作为给定输入 Buf(f)的[27]的 PoW 的输出，它是文件 f 的编码。验证者可以针对文件的任何块询问证明者，并且证明者能够通过提交该块的认证路径来证明知道被询问的块。为了减少验证 PoW 的挑战数量，在计算 Merkle 树之前，文件被编码成 Buf(f)。在文件较大的情况下，可以通过将 Buf(f)限制为 64 MB 的最大大小来进行额外的权衡。

该方案的安全性基于随机线性码的最小距离。关于[27]的功率的更多细节，我们请读者参考附录 B。

3.3 协议规范

我们现在详细说明 ClearBox 程序的规范。如前所述，我们在续集中假设 G 拥有一个由 S 托管的帐户，C、S 和 G 之间的通信通过经过身份验证和加密的通道进行。

卖出程序的说明。

在 ClearBox 中，当客户端 C 希望上传一个新的文件 f 到 S 上时，C 向 g 发出上传请求。随后，C 和 G 开始执行第 3.2.3 节中描述的服务器辅助密钥生成协议.该协议的结果是密钥 k ← H(s)，其中 s←H∫(f)x 给定一个加密散列函数 H∫。

*c 随后在密钥 k 下使用加密算法 enc 对 f 进行加密，计算并向 G 发送[27]的 PoW 输出的 Merkle 树的根，即 FID ← MTBuf(enc(k，f))，其中 Buf 是一个编码函数(cf。*第 3.2.4 节)。

随后，G 检查是否有任何其他客户端以前存储过由 FID 索引的文件。这里出现了两种情况:

*f 以前没有存储过:在这种情况下，G 发出一个定时 generateURL 命令，允许客户端在一个时间间隔内将数据上传到 G 的账户上。*回想一下，定时生成器命令会产生一个在指定时间段后过期的 URL。在加密文件 f́←enc(k，f)的上传终止后，G 访问 S，使用[27]的 PoW 计算 MTBuf(f́)，并验证其与 FID 匹配。如果验证匹配，G 将与 f *相关联的元数据存储在由 FID 索引的新生成的结构中(如客户端 ID C 和 f *的大小，详见第 4 节)。否则，如果 MTBuf(f́)与 FID 不匹配，G 将删除该文件，并将 C 追加到黑名单中。

*f 以前存储过:在这种情况下，G 请求 C 证明它拥有文件 f。*为此，G 和 C 执行[27]的 PoW 协议(更多细节请读者参考附录 B)。本质上，G 选择一个在 Buf(f́)上计算的 Merkle 树的叶索引的随机数 u，并要求 C 给出所有 u 个叶的兄弟路径。作为响应，C 返回对应于与 Buf(f∑)的 Merkle 树相关联的所选 u 叶的兄弟路径。*如果相对于存储的 FID，所有的兄弟路径都有效，g 接受。*如果验证通过，G 将 C 追加到文件结构 FID 中，并向 C 发送确认。反过来，C 删除文件的本地副本，只需要存储 FID 和密钥 k。

获取程序规范。

要下载带有索引 FID 的文件，C 向 G 提交 FID；后者检查 C 是否是添加到 FID 的元数据结构中的用户列表的成员。如果是这样，G 生成一个定时的 URL，允许 C 从 S.7 下载请求的文件。

请注意，如果 C 没有在本地缓存与 FID 相关联的解密密钥，那么 C 可以利用它对 H∞(f)的知识，通过用 g 执行服务器辅助生成协议来获取相应的密钥。

删除程序的说明。

当 C 想删除文件 FID 时，会通知 g。*g 标记 C，用于在下一个时期从与 FID 相关联的元数据结构中移除(更多详细信息，请参见第 3.3 节)。*如果没有其他客户端注册该文件，G 会向 S 发送删除请求。

证明程序的说明。

在每个时期结束时，G 会证明其客户端文件的重复数据消除模式，例如。在他们的账单里。请注意，如果一个客户端在纪元期间请求删除文件 f，则在纪元结束后，G 只会从订阅 FID 的客户端列表中删除标记的客户端。

在纪元 Ej 结束时，G 的每个客户机 C 的清单包括每个存储文件 f 的 C 的访问次数和 CFID 的基数，这表示注册到 f 的客户机集。这里，我们假设一个静态设置，其中客户端为他们在每个时期内存储的文件付费；这符合现有提供商的功能，例如亚马逊 S3，其依赖固定的时期来测量存储消耗(例如，。亚马逊 S3 的纪元间隔是 12 小时)。因为我们的证明程序是有效的(参见。第 4 节)，我们的解决方案可以通过依赖极小的时期而变得实际动态，在这种情况下，账单可以累积多个时期的细粒度存储消耗。

提供商出具的账单是 G 对其客户文件的重复数据消除(和访问)模式的约束性承诺。账单发出后，G 需要让他的客户相信账单格式正确。请注意，G 保存了客户端验证下载请求的记录，可以用来证明每个文件的访问次数。此外，G 需要向文件 f 的所有客户证明:

- 这些客户端中的每一个都包含在 CFID 存储 FID 的客户端集中。
- CFID 的面积与法案中宣称的一样大。
- 存储 f 的客户端引用同一个 CFID 集。

如第 3.4 节所述，最后一种情况会阻止 G 将同一文件的用户包含到不同的累加器中，从而导致文件重复数据消除模式的报告不足。

前两个证明可以通过 G 使用 CARDIAC 来实现。更具体地说，G 累积存储 FID 的客户端的标识；这里，对于每个用户 Ci ∈ CFID，创建具有值 Ui ← H(FID‖Ci‖Ej‖seedi，j)的叶节点，其中 seedi，j 表示为该客户端和该时期采样的随机数，并且该随机数在账单内传送。正如我们在第 3.4 节中所展示的，这保护了用户隐私，并确保当 G 发布大小和/或成员资格的证明时，任何用户的 ID 都不会清楚地提供给其他用户。所有 Ui ∈ CFID 的累积产生一个用 δ 表示的摘要。如前所述，CARDIAC 使 G 能够向集合中的每个客户端证明(I)该客户端是累积集合 CFID 的一部分，以及(ii)累积集合的大小|CFID|的上限。请注意，基数的上限足以防范 rational gateway。这是因为网关不能从报告更大的|CFID|中获益，因为这可能会降低存储 f 的客户端的价格。

图 3 示出了包括 5 个客户端 U0 的集合的例子，..。*U4.用红色虚线圈出的元素描述了 U2 的成员证明，而用蓝色实线圈出的元素描述了集合基数的证明，该集合基数由最后一个非零元素的兄弟路径组成。*灰色元素由开放节点组成，即，。零叶不与任何客户端相关联，因此用符号 0 填充，以及可以从这些符号中导出的所有节点。请注意，如果相应的兄弟路径包含相同的元素，则可以进一步压缩成员资格和集合基数的证明。

*g 仍然需要显示所有存储 FID 的客户端都在同一个累加器中累加。*这可以通过在公共公告板上发布 FID 和其对应的累加器摘要之间的关联来实现(例如，。在 G)的公共网站上。但是，这种方法不会随着文件数量的增加而扩展

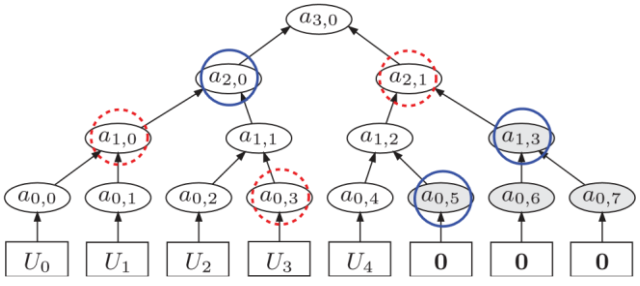


图 3:给定心动图中的一组 5 个元素，集合成员和集合基数的证明草图。

由 g 处理。因此，我们选择只发布随机选择的所有文件子集的关联。这里，G 首先通过调用 GetLayments(tj)获得一个随机种子，其中 TJ 表示纪元 Ej 的票据发行之后的时间点，该时间点由确定性的和众所周知的过程确定。此种子用于对累加器需要在此时期发布的文件进行采样；例如，G 可以提取种子的 ν 位，并将其用作掩码来决定发布哪个文件累加器。在给定时期结束时选择任何文件 FID 的概率随后由 2ν 给出。

请注意，在这种情况下，G 需要计算成员资格的证明，并且只为选定的文件设置基数；该信息被发送到存储所选文件的每个客户端。所选文件的对(FID，δ)随后由 G 发布(例如，。在其自己的网站上)。

验证程序的规范。

验证过程仅由存储文件的客户端执行，文件对应对(FID，δ)已由 G 在纪元结束时发布。请注意，客户端可以通过调用 GetLaminess(TJ)轻松检查哪些 FID 被采样。

给定由 G 为他们的文件发布的成员资格和集合基数的证明，客户端调用 CARDIAC 的 VerifyC 和 VerifyM 算法，以验证他们的 ID 包含在 CFID 存储 FID 的客户端集合中，并且|CFID|如账单中所述。此外，客户检查由 G 发布的对(FID，δ)，以验证他们参考的只有一个 CFID 集。

$2 \cdot \lceil \log_2(|\mathcal{C}_{FID}|) \rceil \lceil \log_2(|\mathcal{C}_{FID}|) \rceil$在这种情况下，隶属关系的证明由高度树的叶子的姊妹路径组成。 因此，成员证明的大小最多是$2 \cdot \lceil \log_2(|\mathcal{C}_{FID}|) \rceil$哈希值，验证最多是哈希函数的$2 \cdot \lceil \log_2(|\mathcal{C}\,\mathrm{FI}|) \rceil$D 执行。另一方面，基数的证明最多需要$2^{\ell-1}$哈希操作来检查开放的叶子(因为我们假设至少一半的叶子被客户端占用)；也就是说，基数证明的大小最多是哈希值，但是验证需要 O(|CFID|)哈希运算。请注意，这最多是在 CARDIAC 中构建 δ 所需工作的一半。如第 4 节所示，该过程效率极高；请注意，如果客户端预先计算并存储树中的开放(零)叶子，VerifyC 和 VerifyM 的验证可以变得更加有效。由于这些节点的值独立于文件标识和客户端标识，因此可以对不同高度的树进行一次预计算。

附加操作。

目录和其他功能:ClearBox 通过在云上的 S 帐户中托管的单个目录结构上工作，对 G 隐藏了客户端的目录结构。这具有减少由 G 承担的开销的好处(即，。没有与路径相关的开销)并最小化向 g 的信息泄漏。

目录创建、目录重命名等目录操作。由用户的软件客户端本地处理。这里，本地目录包含指向存储在其中并外包给云的文件的指针；这使得本地客户端能够执行目录列表和文件重命名等操作，而无需联系 G，从而最大限度地减少了 G 的开销。仅影响存储在云上的客户端文件的操作(例如，。文件删除/创建)被传输到 g。

其他 API:回想一下，ClearBox 利用即将到期的基于 URL 的 PUT 命令(由亚马逊 S3 和谷歌云存储公开[3])使客户端能够将新对象直接上传到 S；在 ClearBox 中，过期的 URL 对于撤销对客户端的数据访问也很重要。

然而，Dropbox、Google drive 等多家商品云服务提供商不支持文件创建的 URL 命令，只提供基于(非过期)URL 的文件下载。为了将 ClearBox 与这样的商品存储提供商集成，我们注意到 ClearBox 的协议规范有以下不同(参见。第 3.3 节):

- 在文件上传时，基于网址的 PUT 被上传文件到 G 的客户端取代，G 再将文件上传到 s。回想一下，G 必须计算上传文件上的 Merkle 树；这可以在 G 将文件上传到 S 之前异步完成，因此减少了 G 的性能损失。
- 文件存储在随机标识符下，可以通过映射到文件标识的永久网址来访问。当用户请求删除文件时，G 会将文件重命名为新的随机选择的 ID。其他需要访问文件的合法客户端必须联系 G，G 会通知他们与重命名的文件对象相对应的新 URL。

在第 4 节中，我们展示了与 Dropbox 接口的 ClearBox 的原型实现，并使用它来评估这种替代技术的性能。

速率限制:类似于[14]，我们对 G 的请求进行速率限制，以防止对可预测文件内容的可能暴力搜索攻击(在辅助密钥生成阶段)，并防止恶意客户端的资源耗尽攻击。为此，我们将客户端在给定时间帧 T1 内可以执行的请求数 Ri 限制在阈值 θmax 内。

3.4 证券分析

在本节中，我们将针对第 2 节中概述的模型讨论 ClearBox 的安全性。

安全访问:我们从展示 ClearBox 确保用 Put 上传文件的客户只要不删除文件，就能用 Get 恢复文件开始。因为我们假设 G 和 S 不会篡改存储协议，所以对健全性论点的威胁只能来自其他恶意客户端。此外，由于“获取”和“验证”过程不会修改存储的数据，因此我们将重点分析“删除”和“放入”操作。

显然，删除过程只是删除了请求删除的用户的访问权限。模拟在这里是不可能的，因为我们假设一个适当的身份管理。也就是说，只有当|CFID| = 0 时，网关才会删除带有标识 FID 的文件。

另一方面，Put 过程只能在第一次上传文件 f 时导致数据的修改。f 的后续上传经过重复数据消除，因此不需要对 s 进行任何数据修改。请注意，在初始上传过程中，恶意客户端可以尝试脱离协议，并构建不适合 f 的 FID，上传另一个文件，或者使用错误的密钥加密文件等。回想一下，G 通过下载上传的文件 f *来验证 FID，以检查 FID 是否=？ *MTBuf(f∑)*。

请注意，如果恶意用户为任何文件 f 创建了一个格式错误的 f∑，即 $f^* \neq \mathrm{enc}(H(H\sum(f)x), f)$，那么这将导致一个随机的 FID 值，该值很可能不会与任何其他文件 ID 冲突。也就是说，f÷将由 S 存储而不进行重复数据消除—这将在不影响其余诚实客户端的情况下，在恶意客户端上产生存储成本(因为他因存储 f÷而被全额收费)。

关于非法客户端访问，我们区分(I)客户端在首次使用 Put 上传文件时获得文件所有权的情况和(ii)客户端在稍后某个时间点使用 Get 访问文件而不拥有文件所有权的情况。对于情况(I)，上传尚未存储的文件对客户端没有帮助。也就是说，客户端不能假装上传带有不同 FID 的文件，因为 G 将检查文件的完整性。当用户请求上传已经存储的文件时，所采用的所有权证明方案(PoW)(参见。第 3.2.4 节和附录 B)确保客户必须知道整个文件，或大于假设泄漏阈值的相应编码数据缓冲区(在文件较大的情况下)。关于 PoW 方案安全性的更多细节可以在附录 b 中找到。在情况(ii)中，当客户端请求访问 f 时，G 首先检查该客户端是否仍订阅 f。请注意，这种验证是由网关在内部

处理的，不会给客户端干扰这一过程的能力。如果授予访问权限，客户端将获得一个临时网址，该网址将在短时间后过期。因此，该信息不能被再次用于进一步的访问；特别是，不再订阅该文件的客户端无法再访问该文件。

请注意，由于依赖经过身份验证的通道，外部对手也无法访问 f。也就是说，我们假设所有交换的消息都经过适当的签名，因此不可能拒绝请求。

安全证明:让 CFID 表示存储文件 f 的一组客户端。

回想一下，f 是用 FID ← MTBuf(enc(k，f))引用的，其中 k←H(H∫(f)x)和 Buf 表示 PoW 中使用的编码。根据我们的服务器辅助加密方案，密钥 k 是根据文件的散列 H∫(f)确定性计算的。由于在整个过程中使用了抗冲突散列函数，因此 FID 唯一地绑定到 f。因此，注册到 f 的所有客户端同样需要相同的 ID FID。

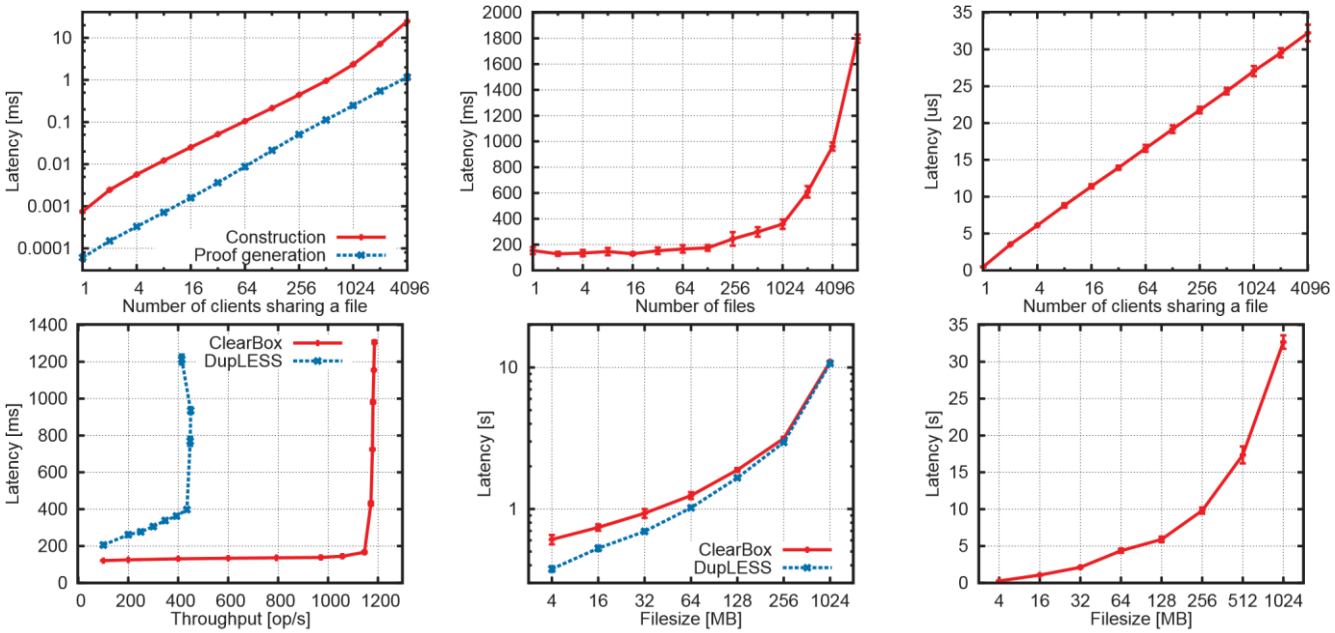
根据 ClearBox 的工作流程，参考相应的文件 ID FID，每个客户 Ci ∈ CFID 都被告知 CFID 的大小。因此，G 首先承诺对 FID，|CFID|。随后，G 对文件子集进行采样，并使用累加器证明重复数据消除模式的正确性。对于这些文件，网关 G 公布 FID 和相应累加器的摘要 δ 之间的关联。这种采样是由 GetRandomness 函数强制执行的，该函数充当不可预测的依赖于时间的随机性源。这使得 G 在学习在纪元结束时提交给

想一下，δ 是对设定的 CFID 的承诺。通过发布采样文件标识的关联列表(FID，δ)，客户端可以确保 G 没有将 CFID 分成单独的子组。关于心脏疾病的安全治疗，我们请读者参考附录 A。

数据机密性:正如第 2 节所解释的，文件大小和用户访问模式的知识对于 G 和 S 运营他们的业务是必不可少的。因此，隐藏这些信息在我们的解决方案中无法实现。在续集中，我们分析了存在好奇的 G 和 s 时存储文件的机密性。

请注意，网关 G 和服务提供商 S 都只看到加密文件，即。f∫←enc(k，f)。因此，如果基础加密方案是安全的，除非密钥 k 被泄露，否则文件的内容将受到保护，不会被任何窃听者窃取。我们的服务器辅助加密方案是消息锁定加密方案的一个实例[14]。由于密钥生成是不经意的，与标准的消息锁定加密对手相比，G 没有任何额外的优势。因此，类似于 MLE 方案，ClearBox 实现了不可预测文件的不可区分性。

请注意，一个好奇的网关 G 可能会猜测流行(可预测)内容 fi 的 H∫(fi)，并计算相应的密钥(因为他知道秘密 x)，以便识别该 fi 是否由某些客户端存储。存储低熵机密文件的客户端可以通过附加一个高熵字符串来防止这种攻击，这样文件就不会再被猜到了。但是，文件的重复数据消除不再可能。对于不知道秘密值 x 的任何其他实体，ClearBox 提供了更强的保护，例如，。服务提供商 s。回想一下，G 速率限制了客户端对加密密钥的请求，以减缓对可预测文件内容的暴力搜索攻击(通过网关的接口)。



(a) Key generation performance at G (b) Overhead of key generation on clients (c) Construction of PoW

(d)心脏构造和一般证明- 心动过速的潜伏期 客户进行的心脏检查 发布的文件。

图 ClearBox 中使用的构建块在许多参数方面的性能评估。

|CFID|时将被采样的文件方面的优势可以忽略不计。由于客户端可以验证 GetRandomness 的输出，因此每个客户端 Ci ∈ CFID 都可以检查 G 是否报告了相应 FID 的 δ。

遵循 CARDIAC 的安全性(参见。附录 A)，G 只能向它的客户证明集合成员和基数，如果基础集合和 δ 计算正确的话。回

4. 亚马逊 S3 和 DROPBOX 的部署

在接下来的内容中，我们使用亚马逊 S3 和 Dropbox 作为后端存储来评估 ClearBox 的原型实现。

4.1 实施设置

我们用 Java 实现了一个 ClearBox 的原型。在我们的实现中，我们依赖 SHA-256、Java 内置随机数生成器和 JPBC 库[7](基于 PBC 密码库[5])来实现 BLS 签名。为了进行基线比较，我们还实施了 Bellare 等人的基于服务器的重复数据消除。[14]并将其与亚马逊 S3 整合.回想一下，在双工模式下，客户端在存储/获取文件时直接与云提供商交互。为了生成密钥，DupLESS 使用基于 RSA 盲签名的服务器辅助遗忘协议[14]。请注意，我们没有实现纯(未加密)云存储系统，因为纯存储的性能可以直接从行速率(即，。网络容量)；在适当的地方，我们将讨论 ClearBox 相对于普通云存储系统的性能。我们在两个双 6 核英特尔至强 E5-2640 上部署了我们的实施，时钟频率为 2.50 千兆赫，内存为 32GB，网络接口卡为 100 兆位/秒。ClearBox 网关和 DupLESS 的辅助服务器运行在一台双 6 核至强 E5-2640 机器上，而客户端位于第二台双 6 核至强 E5-2640 机器上；这确保了 ClearBox 和 DupLESS 之间的公平比较。为了模拟真实的广域网(WAN)，我们依靠 NetEm [38]按照平均为 20 毫秒、方差为 4 毫秒的帕累托分布(模拟广域网的数据包延迟方差[24])来塑造网络接口上交换的所有流量。

我们的实现与亚马逊 S3 和 Dropbox(分别)接口，用于存储用户文件。为了获取比特币块哈希，我们的客户端向比特币块浏览器提供的 getblockhash 工具调用 HTTP 请求[2]。在我们的设置中，每个客户端在一个闭环中调用一个操作，即，。一个客户端最多只能有一个挂起的操作。我们在 G 的机器上产生了多个线程——每个线程对应于一个处理给定客户端的请求/账单的唯一工作者。我们将可以并行产生的最大线程数限制在 100 个.在我们的实现中，网关和客户端将与每个文件相关的元数据信息存储在本地 MySQL 数据库中。

该网关利用了 MySQL 的缓存功能，以降低输入/输出成本。回想一下，MySQL [4]的查询缓存引擎将选择查询的文本映射到它们的结果。对于每个文件，网关存储 FID、每个文件的大小以及共享该文件的客户端的标识。

我们的图中的每个数据点在 10 次独立测量中取平均值；在适当的情况下，我们包括相应的 95%置信区间。为了准确测量(微)延迟，我们利用了 Boyer [19]提出的基准工具。

4.2 性能赋值

在评估 ClearBox 的整体性能之前，我们首先从分析构建块在许多参数方面的性能开始。除非另有说明，否则我们的评估依赖于表 2 中列出的默认参数。

网关辅助密钥生成:在图 4(a)中，我们评估了遗忘密钥生成模块(cf。第 3.2.3 节)关于网关。这里，我们要求网关背靠背地处理密钥生成请求；然后，我们逐渐增加系统中的请求数量(直到吞吐量饱和)，并测量相关的延迟。我们的结果表明，我们的方案在网关上每个客户端密钥生成请求产生了几乎 125 ms 的延迟，并且达到了每秒 1185 个操作的最大吞吐量；这大大提高了双工密钥生成的最大吞吐量(每秒 449 次操作)。这主要是因为与 RSA 签名相比，网关计算 BLS 签名的速度要快得多。相比之下，如图 4(b)所示，客户端验证 BLS 签名比双工中使用的 RSA 变体更昂贵。然而，我们认为，与 DupLESS 相比，我们的方案引入的开销很容易被客户端容忍，因为客户端的工作主要是散列文件；例如，对于 16 MB 的文件，与 DupLESS 相比，我们的建议只会在客户端上产生 213 ms 的额外延迟开销。

所有权证明:图 4(c)描述了 FID 在文件大小方面实例化[27]的 PoW 方案所需的开销。如附录 B 中所解释的，[27]的功率方案

参数	缺省值
默认文件大小	16 兆字节
RSA 模数大小	2048 位
|CFID|小水。PoW 挑战	100 50
椭圆曲线(BLS)	PBC 图书馆曲线

表 2:评估中使用的默认参数。

通过在有限大小的缓冲区(在我们的例子中为 64 MB)中编码原始文件，降低了验证 PoW 的成本。我们的结果表明，FID 的计算在网关和客户端上都产生了可容忍的开销。例如，对于大小为 16 MB 的文件，FID 的计算需要 1.07 秒。

心脏:在图 4(d)中，我们针对|CFID|评估了心脏中累加器和证明生成的构造所产生的开销，即。订阅同一文件 f 的客户端数量。我们的结果表明，由心脏结构引起的潜伏期很容易被 G 容忍；例如，为 1000 个用户构建一个累加器需要 2.34 ms。显然，随着共享同一文件的用户数量的增加，这种开销也会增加。请注意，一旦构造了整个 Merkle 树，证明生成的执行速度将大大快于累加器的构造速度。在这种情况下，G 只需要遍历树，并记录累加器所有成员的兄弟路径。

在图 4(e)中，我们评估了在每个时期结束时，针对被选择用于证明的文件，在心动图中的证明生成在 G 上产生的开销。我们的结果显示，当选择的文件少于 100 个时，延迟约为 150 毫秒，因为这些文件的累加器可以由我们池中的独立线程并行处理。当所选文件的数量增加超过我们的池阈值时(即，。100)，证明生成的等待时间增加，。4000 个文件需要 1 秒钟。

在图 4(f)中，我们评估了客户验证的开销。假设验证只需要 $\lceil \log |\mathcal{C}_{FID}| \rceil$哈希，这个操作只会给客户端带来少量开销。例如，当|CFID| = 1000 时，在 CARDIAC 中验证成员资格和基数只需要 27 µs。

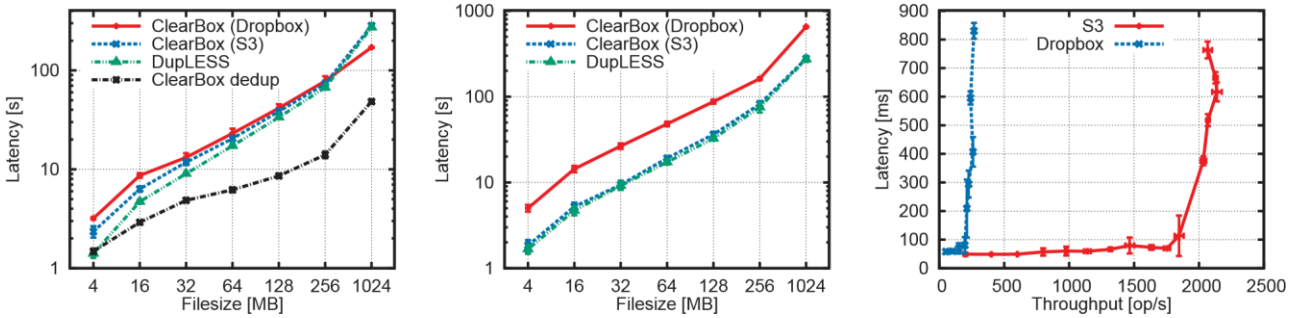
ClearBox:在图 5(a)中，我们评估了用户在 PUT 操作中看到的与文件大小相关的延迟。我们的结果显示，与 S3 的双工相比，ClearBox 实现了相当的性能.例如，在亚马逊 S3 上传 16 MB 文件时，DupLESS 的延迟为 4.71 秒，而 ClearBox 需要 6.33 秒。ClearBox 中的额外开销主要来自用户对 FID 的计算(参见。图 4(c))；延迟主要由上传到亚马逊的文件决定。

相比之下，当用户想要上传已经存储在云上的文件时，ClearBox 的上传性能比 DupLESS 更快，因为用户不再需要上传文件，而是必须执行带有 G 的 PoW 协议——与上传中等大小的文件相比，这种协议的延迟可以忽略不计。回想一下，这是以 G 的额外负载为代价的；相比之下，当用户上传/下载文件时，DupLESS 中的服务器不承担任何负载。

当使用 Dropbox 时，回想一下客户端将文件直接上传到 G，G 又将文件上传到其 Dropbox 帐户(因为 Dropbox 不为文件创建提供 URL 命令)。尽管此过程需要客户端和 G 之间较少的通信回合(以获取签名的 URL)，但我们的结果显示，使用 Dropbox 在 ClearBox 中上传未消除重复数据的文件时产生的延迟略大于其亚马逊 S3 对应方；我们认为，这种差异是由于与亚马逊 S3 相比，我们的客户端和 G 之间的上传带宽较低.请注

意，在 ClearBox 中上传已消除重复数据的文件的性能不依赖于后端云提供商，因此对于我们的亚马逊 S3 和 Dropbox 实施来说是相同的。

在图 5(b)中，我们评估了用户在 ClearBox 的 GET 操作中看到的延迟。我们的结果表明，在 ClearBox 和 DupLESS 中，GET 操作都会导致类似的延迟。回想一下，在 ClearBox 中，客户端必须首先联系 G 并获取定时 GET URL 来下载资源。鉴于 GET 操作的延迟主要由下载速度决定，这一轮额外的通信产生的开销是微不足道的。请注意，与普通云存储系统相比，



投入产出绩效　获得绩效　延迟与吞吐量

图 ClearBox 使用亚马逊 S3 和 Dropbox 作为后端云存储的性能评估。

ClearBox 用户表现出的延迟是可以容忍的。例如，假设线路速率为 100 Mbps，在普通云中下载一个 32MB 的文件几乎需要 3 秒钟；另一方面，在 ClearBox 中下载这个文件会导致 10 秒的延迟。

在图 5(c)中，我们根据获得的吞吐量评估了 ClearBox 中网关的延迟。这里，我们假设 G 处理的 50%的请求对应于 PUT 请求，而剩下的 50%是 GET 请求。我们进一步假设 50%的上传请求对应于已经存储在 s 的文件。将(未消除重复数据的)文件上传到 S 时，我们不会在验证 FID 时测量 G 上产生的开销，因为这种验证是异步的，可以通过以下方式完成 *g 在稍后的时间点*。我们通过模拟来自本地套接字的客户端请求，进一步模拟了 G 的大下载带宽。我们的发现表明，当与亚马逊 S3 集成时，ClearBox 实现了大约每秒 2138 次操作的最大吞吐量.这表明我们的解决方案可以扩展到系统中的大量用户。当与 Dropbox 接口时，ClearBox 的性能会恶化，因为在这种情况下，G 上的负载会大大增加；我们的方案显示的最大吞吐量下降到几乎每秒 289 次操作。

5. 相关著作

云存储系统中的重复数据消除在文献中获得了相当大的关注。

在[28]中，Harnik 等人。描述客户端重复数据消除带来的诸多威胁，在这些威胁中，对手可以通过猜测可预测消息的哈希来了解文件是否已经存储在特定云中。可以使用所有权证明方案(PoW) [23，27]来应对这种泄露，该方案使客户能够证明其拥有整个文件。PoW 的灵感来自可检索性和数据拥有证明(POR/PDP)方案[11，40]，不同之处在于 PoW 在建立时没有预处理步骤。Halevi 等人。[27]提出一种基于 Merkle 树的 PoW 构造，这种构造在构造和验证 PoW 时对服务器产生低开销。徐等。[44]在[27]的 PoW 的基础上构建一个 PoW 方案，支持在有限泄漏设置下的客户端重复数据消除。Di Pietro 和 Sorniotti [23]提出了一种 PoW 方案，以额外的服务器计算开销为代价，降低了[27]的通信复杂度。布拉斯科等人。[16]提出一种基于 Bloom 过滤器的 PoW，进一步降低[23]的服务器端开销。

Douceur 等人。[25]引入了收敛加密的概念，这是一种确定性加密，使用从明文本身导出的密钥对消息进行加密。聚合加密在语义上并不安全[15]，它只为内容不可预测的消息提供机密性。Bellare 等人。[31]提议的 DupLESS，一种服务器辅助加密来执行重复数据删除方案；这里，加密密钥是基于文件的散列和辅助服务器的私有密钥来明确计算的。在[42]中，Stanek 等人。提出一种加密方案，保证不受欢迎的数据的语义安全和受欢迎的文件的较弱的安全性(使用聚合加密)。在[41]中，Soriente 等人。提出了一种在不可知云中分布式实施共享所有权的解决方案。该解决方案可以与 ClearBox 结合使用，使用户能够分布式管理其已消除重复数据的文件的访问控制。

有几个积累隐藏集的建议，如最初的基于 RSA 的构造[12]，已扩展到动态累加器[21]，以及非成员证明[32]。也存在基于双线性群[22，39]和基于散列函数[20，33]的构造。一个相关的概念是零知识集[30，37]。这些结构提供了集合成员的证明。但是，请注意，加密累加器通常不提供有关累加集的信息，如内容或基数。

我们的攻击者模型与[43]中考虑的模型有相似之处，其中云提供商被认为是经济合理的。该方案依赖于沙漏函数，以验证云提供商以加密形式存储文件，这给试图按需应用沙漏函数的理性云提供商带来了很大的限制。

6. 结论

在本文中，我们提出了 ClearBox，它使云提供商能够透明地向其客户证明其存储数据的重复数据消除模式。ClearBox 还对经过重复数据消除的文件实施细粒度访问控制，支持数据机密性，并抵御恶意用户。从 ClearBox 的原型实现中得到的评估结果表明，我们的建议可以很好地适应系统中用户和文件的数量。

据我们所知，ClearBox 是第一个完整的系统，它使用户能够验证他们的数据所展示的存储节省。我们认为，ClearBox 激发了一种新的云定价模式，这种模式承诺在用户之间更公平地分配存储成本，而不会损害数据机密性或系统性能。我们认为，这种模式为用户在云中存储流行数据提供了强大的激励(因为存储流行数据会更便宜)，并阻止个人和独特内容的上传。作为一个副产品，文件的流行还向云用户提供了他们在云中的隐私级别的指示；例如，用户可以验证他的私有文件没有经过重复数据消除，因此没有被泄露。

承认

作者感谢尼科斯·特里亚多普洛斯和匿名审稿人的宝贵反馈和评论。这项工作得到了欧洲联盟(欧盟)在地平线 2020 (H2020)研究和创新方案的信息和通信技术主题下资助的 TREDISEC 项目(政府编号 644412)的部分支持。

7. 参考

[1] 亚马逊 S3 定价。http://aws.amazon.com/s3/pricing/.

[2] 比特币实时统计和工具。http://blockexplorer.com/q.

[3] 谷歌云存储。https://cloud.google.com/storage/.

[4] MySQL 查询缓存。http://dev.mysql.com/文件/refman/5.1/en/query-cache . html。

[5] PBC 图书馆。http://crypto.stanford.edu/pbc/, 2007.

[6] 到 2014 年，云市场将增长两倍以上，达到 1500 亿美元。http://www . MSP today . com/topics/MSP today/articles/364312-cloud-market-will more-2014-reach . htm，2013。

[7] JPBC:基于 Java 配对的密码库。http://gas.dia.unisa.it/projects/jpbc/#. U3HBFfna5cY，2013 年。

[8] 比特币作为随机性的公共来源。https://docs.google.com/presentation/d/ 1vwhm4moza znhxsoj8 facfnk2b _ vxnfbdzgc5 epexfe/view？pli = 1 #幻灯片=id。g3934beb89_034，2014。

[9] 这些是目前最便宜的云存储提供商。http://qz . com/256824/这些是-the-the-chaest-云存储-提供商-rightnow/，2014。

[10] ARMKNECHT，f。BOHLI，。KARAME。O.,。刘，z，。和路透社。A. 可检索性的外包证明。2014 年 11 月 3 日至 7 日在美国亚利桑那州斯科茨代尔举行的 2014 年 ACM SIGSAC 计算机和通信安全会议记录(2014 年)，pp。831–843.

[11] ATENIESE。BURNS。C.,。CURTMOLA，。HERRING，。洛杉矶基斯纳，。PETERSON。名词（noun 的缩写）。J.,。和宋，d。X.
不可信商店中可证明的数据占有。在美国计算机学会计算机和通信安全会议(2007)上，第。598–609.

[12] 北巴里克。和 PFITZMANN，b。无冲突累加器和无树故障停止签名方案。在欧洲墓穴(1997)中，w。梅米，艾德。卷。《计算机科学讲义》第 1233 卷，施普林格，页。480–494.

[13] BELLARE。和基勒维济。交互式消息锁定加密和安全重复数据消除。公钥密码学- PKC 2015 -第 18 届 IACR *国际实践与理论会议*
公钥密码学，美国马里兰州盖瑟斯堡，2015 年 3 月 30 日至 4 月 1 日。卡兹，艾德。卷。《计算机科学讲义》第 9020 卷，斯普林格，第 100-100 页。516–538.

[14] BELLARE。美国基勒维济，。和 RISTENPART，t。双重:服务器辅助加密已消除重复数据的存储。《第 22 届 USENIX 安全会议记录》(美国加州柏克莱，2013 年)，美国证券交易委员会第 13 届会议，USENIX 协会，页。179–194.

[15] BELLARE。美国基勒维济，。和 RISTENPART，t。消息锁定加密和安全重复数据消除。在…里
密码学进展——欧洲密码学 2013，第 32 届年会
理论与应用国际会议
密码技术，希腊雅典，5 月 26 日至 30 日，2013。会议记录(2013 年)，t。约翰逊和 p。问。阮，Eds。卷。《计算机科学讲义》第 7881 期，施普林格，第 100-100 页。296–312.

[16] BLASCO。迪皮埃特罗，。a .奥菲拉，。还有索尼奥蒂，a。使用布隆过滤器进行重复数据消除的可调整所有权证明方案。通信和网络安全，2014 年 IEEE 会议(2014 年 10 月)，第。481–489.

[17] BOLDYREVA。基于 gap-diffie-hellman 群签名方案的高效门限签名、多重签名和盲签名方案。

[18] BONEH。LYNN，。和沙哈姆。短的
韦尔配对的签名。*J. 密码学 17，4 (2004)，297–319*。

[19] BRENT BOYER。健壮的 Java 基准测试。http://www . IBM . com/developer works/library/j bench mark2/j \-bench mark2 \-pdf . pdf。

[20] BULDAS，a。劳德，p，。和利普马。消除
应用于可问责证书管理的反证。*计算机安全杂志 10，3 (2002)，273–296*。

[21] CAMENISCH。还有 LYSYANSKAYA，a。动态的
累加器及其在匿名证书有效撤销中的应用。《密码学进展 2002》(2002)，斯普林格，页。61–76.

[22] 我是达蒙加德。北特里亚多普洛斯。使用双线性映射累加器支持非成员证明。*IACR 密码学电子档案 2008 (2008)，538*。

[23] 迪皮埃特罗。还有索尼奥蒂，a。提高重复数据消除所有权证明的效率和安全性。在第七届美国计算机学会信息、计算机和通信安全研讨会会议录(2012 年，美国纽约州纽约市)中，亚洲计算机安全会议第 12 届会议，美国计算机学会，pp。81–82.[24]d . DOBRE，。KARAME，。李，西，。MAJUNTKE，。北 SURI，。和武科立克。Powerstore:高效、稳健存储的写作证明。在 2013 年美国计算机学会计算机会议录& # 38;《通信安全》(纽约，纽约，美国，2013 年),《通信安全》第 13 期，美国计算机学会，第 10-11 页。285–298.

[25] DOUCEUR。R.,。ADYA，a，。西部 BOLOSKY。J.,。西蒙，华盛顿，。还有 THEIMER，m。在无服务器分布式文件系统中回收重复文件的空间。在国际发展合作中心(2002 年)，第 10-11 页。617–624.

[26] 菲亚特 a。还有沙米尔，a。如何证明自己:
识别和签名问题的实用解决方案。《密码学进展论文集——CRYPTO'86》(伦敦，英国，英国，1987)，斯普林格-弗拉格，页。186–194.

[27] HALEVI。哈尼克，d，。PINKAS，b，。舒曼-皮莱格。远程存储系统的所有权证明。在第 18 届计算机和通信安全会议记录(纽约，纽约，美国，2011 年)中，CCS '11，计算机和通信管理，pp。491–500.

[28] HARNIK。PINKAS，b，。舒曼-皮莱格。云服务中的辅助渠道:云存储中的重复数据消除。*IEEE 安全与隐私 8，6 (2010)，40–47*。

[29] KARAME。O.,。ANDROULAKI，e，。和卡普坤。比特币双消费快速支付。在 2012 年美国计算机学会计算机和通信安全会议记录(美国纽约州纽约市，2012 年)中，CCS '12，美国计算机学会，第 100-100 页。906–917.

[30] KATE，a。ZAVERUCHA。米（meter 的缩写））,。我是戈德堡。多项式的常数大小承诺及其应用。密码学进展-亚洲密码 2010.斯普林格，2010 年，页。177–194.

[31] KEELVEEDHI。BELLARE。和 RISTENPART，t。双重:服务器辅助加密已消除重复数据的存储。作为第 22 届 USENIX 安全研讨会(USENIX 安全会议 13)的一部分提交(华盛顿特区，2013 年)，USENIX，页。179–194.

[32] 李，j。李，北，。和薛。具有有效非成员证明的通用累加器。应用密码学与网络安全，第五届国际会议，ACNS，2007，中国珠海，2007 年 6 月 5-8 日，会议录(2007)，页。253–269.

[33] LIPMAA。在没有可信设置的情况下从欧几里德环保护累加器。2012 年 6 月 26 日至 29 日，在 ACNS 举行的第十届应用密码学与网络安全国际会议上.《会议记录》(2012 年)，第 10 页。224–240.

[34] 刘。黄，十，。傅，。杨格。了解云存储系统中的数据特征和访问模式。在第 13 届 IEEE/ACM 集群、云和网格计算国际研讨会上，CCGrid 2013，荷兰代尔夫特，2013 年 5 月 13-16 日(2013 年)，pp。327–334.

[35] MEYER。T.,。还有西部的博洛斯基。J. 实用重复数据删除研究。《第九届 USENIX 文件和存储技术会议记录》(美国加州柏克莱，2011 年)，FAST'11，USENIX 协会，页。1–1.

[36] MEYER。T.,。还有西部的博洛斯基。J. 实用重复数据删除研究。*跨。存储 7，4(2 月*。2012), 14:1–14:20.

[37] MICALI，s 。 RABIN，。和基里安。零知识集。《计算机科学基础》，2003 年.诉讼程序。*第44 届电气和电子工程师协会年会(2003 年)，电气和电子工程师协会，页*。80–91.

[38] NETEM。Linux 基金会。网站，2009 年.在线提供，网址为 http://www.linuxfoundation.org/协作/工作组/网络/网络。

[39] NGUYEN，1. 双线性对的累加器及其应用。2005 年 2 月 14 日至 18 日在美国加利福尼亚州旧金山举行的 2005 年 RSA 会议上的密码学家的轨迹，会议录(2005 年)，第 100-100 页。275–292.

[40] SHACHAM。和沃特斯。可检索性的简洁证明。在 ASIACRYPTY(2008)中，pp . 90–107.

[41] SORIENTE。KARAME。O.,。RITZDORF，h，。MARINOVIC。和卡普坤。公社:不可知云中的共享所有权。2015 年 6 月 1 日至 3 日在奥地利维也纳举行的第 20 届访问控制模型和技术学术研讨会论文集(2015 年)，第 10-11 页。39–50.

[42] STANEK。SORNIOTTI，。ANDROULAKI，e，。和洛杉矶肯德尔。一种安全的云存储重复数据删除方案。在金融密码学和数据安全-第 18 届国际会议，FC 2014，基督教堂，巴巴多斯，2014 年 3 月 3 日至 7 日，修订论文选集(2014 年)，页。99–118.

[43] 范迪克，男。JUELS，a，。OPREA，a，。RIVEST。长度,。斯特凡诺夫，。北特里亚多普洛斯。沙漏方案:如何证明云文件加密。在 2012 年美国计算机学会计算机和通信安全会议记录(纽约，纽约，美国，2012 年)中，CCS '12，美国计算机学会，第 10-11 页。265–280.

[44] 徐杰。嫦娥。-C，。周杰。无力的云存储中加密数据的防泄漏客户端重复数据消除。在第八届美国计算机学会信息、计算机和通信安全专题讨论会(2013 年，美国纽约)会议录中，亚洲计算机安全会议第 13 届会议，美国计算机学会，第 10-11 页。195–206.

附录 a。心脏的安全性分析

接下来，我们分析了 CARDIAC 的安全性。设一个集合 X，该集合的累计摘要为 δ ← Acc(X)。回想一下，$\delta = \mathtt{H}(a_{\ell,0}, \ell)$ (H 是一个加密安全的散列函数)是对$a_{\ell,0}$根和 Merkle 树高度的承诺。出于我们分析的目的，我们区分了两个安全目标:(I)使用 ProveM 和 VerifyM 证明成员资格，以及(ii)使用 ProveC 和 VerifyC 证明|X|上的上限。由于 Merkle 树的安全性对于第一个目标[20]有很好的理解，因此我们在后续文章中重点分析第二个目标。

由于是使用兄弟路径的长度(间接)验证的，并且假设哈希函数是安全的，因此 CARDIAC 确保 Merkle 树最多可以编码 2 个元素。ProveC 输出最后一个集合元素 a0，| X |-1 的兄弟路径. 验证算法 VerifyC 要求检查同级路径中所有打开节点的值。回想一下，开放节点是那些仅依赖于零叶的节点，因此代表了公开的已知值。事实上，这个验证步骤构成了每个单个零叶的成员证明。

反过来，这确保了至少树的$2^{\ell} - |X|$叶为零。

一个直接的结果是，最多|X|个叶可以是非零的，给出了 X 的上限。请注意，该证明假设位于最后一个非零元素左侧的其他叶子没有设置为零。这一假设可以放心地做出，因为 G 没有动机放置更多的零叶(因为这意味着注册的用户比声称的要少，并且支付给 G 的费用太少)。

输入:长度为 M 位的文件 f，分成长度为 512 位的块。
初始化位置:
计算位长 ，接近
m，但最多 64 MB，即 512 位的块。
用 2 个 512 位的块初始化缓冲区 Buf。
初始化一个 m 行 4 列的表 ptr，它保存指向缓冲区 Buf 的指针。一个临时值四将被初始化为 SHA-256 的四。
还原阶段:
对于每个 I∏m):
更新四:= SHA-256(四；文件[i])，其中文件[i]表示文件的第 I 个块。
初始化 ptr[i] =集群，IV 截断为 4 位..
对于 j = 0 : 3，做
块=文件[i]的循环移位 j∫128 位
在混合阶段将异或块放入位置:
重复 5 次:
对于每个区块，请执行以下操作
j * 128 位的 BlockBuf[i]
如果异或块进入位置
树结构:最终的缓冲区内容由 Buf(f)表示。
使用 Buf(f)的块作为叶子来构建 Merkle 树。 leaves.

图 6:详细的 PoW 协议[27]。

B. 基于[24]的功率

在图 6 中，我们详细说明了 Halevi 等人的所有权证明。[27]. 该协议分为三个阶段:在第一阶段，文件 f 被缩减为最大 64 MB 大小的缓冲区。然后，对缓冲区的内容进行伪随机混合，最后计算缓冲区的 Merkle 树。前两个阶段可以看作是对文件应用一个线性代码，并输出一个缓冲区 Buf(f)。随后，为了验证一个 PoW，验证器从 Buf(f)中请求随机数量的叶子的兄弟路径，并检查验证路径是否与 Merkle 树的根匹配。

缓冲区是用随机线性代码对文件进行编码。代码是使用 SHA-256 生成指针数组 ptr 生成的。由于哈希函数的抗冲突性，因此对文件的任何修改都会导致不同的代码。假设代码的最小距离为 L/3，其中 L 是缓冲区中的块数，任何两个有效缓冲区在至少 L/3 个位置上会有所不同。因此，证明者只有在不知道正确的缓冲区的情况下，才能以(2/3)t 的概率成功回答 t 的挑战。例如，当进行 t = 20 次挑战时，成功的概率最多为 212.请注意，由于部署的哈希函数的抗冲突性，在不完全了解文件的情况下，在前两个步骤中会导出两个不同的随机代码。特别是，缓冲区中的任何块都是不太可能被预测的。我们请读者参考[27]以获得关于该结构安全性的更多细节。