doi: 10.12068/j.issn.1005-3026.2018.08.005

隐藏访问模式的高效安全云存储方案

李宇溪, 周福才, 徐紫枫
(东北大学 软件学院, 辽宁 沈阳 110169)

摘 要: 围绕当前云存储环境中用户数据机密性和隐私泄露问题,提出一个隐藏访问模式的高效安全云存储方案. 该方案首先将文件分为固定大小的数据块,利用伪随机函数和抗碰撞哈希函数将数据块编码、加密,并将密态数据块上传至云服务器的伪随机集合超集内,构建安全云存储结构;同时,设计两轮用户访问策略,在隐藏访问模式的同时降低了存储代价和访问交互次数,实现文件的动态高效更新. 安全分析表明,选择适当的安全参数,方案满足 L_1L_2 - 动态自适应安全性. 实验结果表明,本方案在保证数据机密性的同时,更适用于实际的云存储环境.

关 键 词: 云存储;访问模式;伪随机函数;抗碰撞哈希函数;机密性
中图分类号: TP 309 **文献标志码**: A **文章编号**: 1005-3026(2018)08-1086-06

An Efficient and Secure Cloud Storage Scheme with Hidden Access Patterns

LI Yu-xi, ZHOU Fu-cai, XU Zi-feng
(School of Software, Northeastern University, Shenyang 110169, China. Corresponding author: ZHOU Fu-cai, E-mail: fczhou@mail.neu.edu.cn)

Abstract: Aiming at the problems of data confidentiality and user's privacy leakage in cloud, this paper proposes an efficient and secure cloud storage scheme with hidden access patterns. Firstly, the files are divided into data blocks with fixed sizes, and encrypted by pseudo-random functions and collision-resistant Hash functions. Then, in order to construct the secure cloud storage structure, the blocks are uploaded into pseudo-random collection superset in the cloud server. Meanwhile, a two-round access strategy that hides the access patterns, reduces the storage cost and access interaction is designed. Security analysis shows that the proposed scheme achieves L_1L_2 - dynamic adaptive security. Experimental results show that the proposed scheme can not only protect data confidentiality, but also be more suitable for the actual cloud storage.

Key words: cloud storage; access pattern; pseudo-random function; collision-resistant Hash function; confidentiality

面对大数据和物联网等应用环境中的多元化需求,数据不再像传统信息系统一样采用集中式的存储,云存储等新模式被广泛应用,吸引了越来越多企业和个人用户的关注和使用;然而现有的云存储服务在数据安全保护方面存在严重不足,很多潜在的云存储用户会因为担心数据泄露而不得不使用传统的存储服务,这在很大程度上阻碍了云存储服务的推广.

为了能够让用户放心地将数据存储在云服务器,加密成为确保数据存储安全的隐私保护手段;但是,加密不能保护云存储中数据的访问模式,即用户访问的数据在云存储中的位置、大小以及访问频率等信息,而访问模式通常也蕴含了用户的隐私信息,已成为泄露用户隐私的一种潜在途径.

收稿日期: 2017-04-12
基金项目: 国家自然科学基金资助项目(61772127,61472184); 国家科技重大专项(2013ZX03002006); 辽宁省科技攻关项目(2013217004); 中央高校基本科研业务费专项资金资助项目(N151704002).
作者简介: 李宇溪(1990-),女,辽宁朝阳人,东北大学博士研究生; 周福才(1963-),男,吉林长春人,东北大学教授,博士生导师.

文献[1]指出,基于频率统计的攻击方式,通过分析访问模式能够成功地推断一个加密邮件库搜索内容的概率大约为 80%. 由此可见,在云存储环境中,隐藏访问模式对于保护用户数据隐私十分必要,因此,急需一种高效、安全的云存储方法,能够在不可信环境中对访问模式进行有效保护.

本文从保护数据隐私的角度出发,针对不可信的云存储环境,提出隐藏访问模式的高效安全云存储方案. 该方案首先将文件拆分成固定大小的数据块并进行编码、加密,利用伪随机函数将密态数据块上传至云服务器的伪随机集合超集内,构建安全云存储结构;同时,针对用户的上传、下载以及更新文件等操作,设计两轮访问策略,访问过程中服务器无法获得文件个数、大小以及访问频率信息,在隐藏访问模式的同时实现文件的动态高效更新.

1　相关工作

目前能够实现隐藏访问模式的方法主要有隐私信息检索(private information retrieval, PIR)[2]和不经意随机存取(oblivious random access memory,ORAM). 近年来,许多学者将 PIR 的安全模型进行扩展,提出了 SPIR 模型[3-4],即不仅仅考虑单方的隐私问题,而且要求协议双方都不能获取另一方的访问模式. PIR 协议计算复杂度较高且均面向明文数据,即假设云服务器是可信的,因此不满足本方案的安全模型.

ORAM 最早由 Goldreich 在 1987 年提出[5],其目的并非用于保护用户的隐私,而是用于防止逆向工程等软件的攻击. 2013 年,Stefanov 等人提出了面向轻量级客户端的 ORAM 方法——PathOram[6]. 2016 年,孙晓妮等人在基于二叉树 ORAM 方案的基础上,构造了一个多用户的 ORAM 方案[7],但目前 ORAM 方法仍存在很多问题,因而,需要一种高效安全的云存储方法,能够在不可信环境中对访问模式进行有效的保护.

2　方案模型

图 1 所示为方案的架构图. 方案包括两个类型的实体:用户和云服务器. 其中,用户为数据的拥有者,云服务器向用户提供数据存储服务. 在本场景中,数据以文件形式进行组织,用户将文件进行处理并加密,外包至云服务器进行存储,随后用户可以对云服务器中的存储结构进行访问.

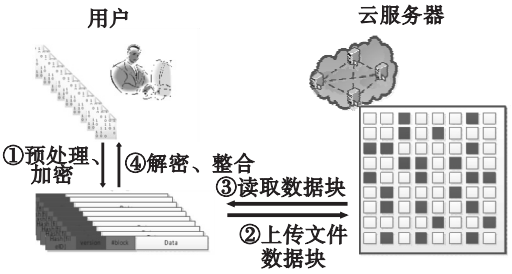

图 1　方案架构

Fig. 1　Scheme architecture

2.1　形式化定义

本方案主要由以下 5 个多项式时间算法或协议组成:

$K \leftarrow \mathrm{KeyGen}(k)$:密钥生成算法,运行于用户端,用于生成用户在方案进行中所需的密钥. 算法输入安全参数 k,输出密钥集合 K.

$D \leftarrow \mathrm{Build}(K, d_0, \{\mathrm{id}_i, \mathrm{data}_i\}_{i=1}^{t})$:构建算法,运行于用户端,主要用来构建存储结构. 算法输入密钥 K、系统中存储数据块的总数量上限 d_0、文件数据 data_i 和文件标志 id_i,输出存储结构 D.

$(\mathrm{S}: D') \leftarrow \mathrm{Upload}(\mathrm{U}: \mathrm{id}_f, \mathrm{data}_f, \mathrm{size}_f, K; \mathrm{S}: D)$:上传协议,主要用于完成向存储结构中写入文件数据. 用户端输入文件标志 id_f、数据 data_f 和文件大小 size_f,服务器端输入存储结构 D,服务器端输出一个新的存储结构 D'.

$(\mathrm{U}: \mathrm{Data}) \leftarrow \mathrm{Download}(\mathrm{U}: \mathrm{id}_f, K; \mathrm{S}: D)$:下载协议,主要用于完成从存储结构中读取文件数据. 客户端的输入为文件标志 id_f 和密钥 K,服务器端输入存储结构 D. 客户端输出文件内容 Data.

$(\mathrm{S}: D') \leftarrow \mathrm{Update}(\mathrm{U}: \mathrm{id}_f, \mathrm{data}_f, K; \mathrm{S}: D)$:更新协议,主要用于更新存储结构中的文件内容. 客户端输入文件标志 id_f 和密钥 K,服务器端输入存储结构 D,服务器端输出更新后的存储结构 D'.

2.2　安全性定义

本文定义 L_1L_2 - 动态自适应安全性. 假设服务器为一个诚实并好奇的敌手,即其会试图获取用户数据,具有从存储结构和访问协议两个过程中不断分析用户数据的能力. 自适应指敌手可以根据之前的质询结果来发起新的质询,拥有更大的攻击能力. 定义 $\mathrm{Real}_{\mathrm{A}}(k)$ 为敌手 A 和用户之间的交互式实验,使用真实的方案算法. 定义 $\mathrm{Ideal}_{\mathrm{A,S}}(k)$ 为敌手 A 和一个有状态的模拟器 S 之间的交互式实验. L_1L_2 - 动态自适应安全性指的是,存在模拟器 S,使得任何具有多项式时间计算能力的敌手都无法区分 $\mathrm{view}_{\mathrm{Real}}$ 和 $\mathrm{view}_{\mathrm{Ideal}}$.

定义 1 L_1L_2 - 动态自适应安全性:令 A 为一个有状态的敌手,S 为一个有状态的模拟器,L_1,L_2 分别为初始化构建阶段和访问阶段(包括上传、下载以及更新)的泄露函数. 考虑以下两个实验:

$\mathrm{Real}_{\mathrm{A}}(k)$:

$(d_0,\{\mathrm{id}_i,\mathrm{data}_i\}_{i=1}^{t})\leftarrow \mathrm{U}(k)$

$K\leftarrow \mathrm{KeyGen}(k)$

$D\leftarrow \mathrm{Build}(K,d_0,\{\mathrm{id}_i,\mathrm{data}_i\}_{i=1}^{t})$

for $1\leqslant i\leqslant q$

$q_i \xleftarrow[\text{each time}]{\text{one query}} \mathrm{A}(D,\mathrm{size}_1,\cdots,\mathrm{size}_{f_{i-1}})$

$\mathrm{size}_{f_i}\xleftarrow{\mathrm{A}}\mathrm{Update}(\mathrm{U}:\mathrm{id}_{f_i},\mathrm{data}_{f_i},K;\mathrm{A}:D)$

or $\mathrm{size}_{f_i}\xleftarrow{\mathrm{A}}\mathrm{Download}(\mathrm{U}:\mathrm{id}_{f_i},K;\mathrm{A}:D)$

or $\mathrm{size}_{f_i}\xleftarrow{\mathrm{A}}\mathrm{Upload}(\mathrm{U}:\mathrm{id}_{f_i},\mathrm{data}_{f_i},\mathrm{size}_{f_i},K;\mathrm{A}:D)$

output $b\leftarrow \mathrm{A}(D,d_0,\mathrm{size}_1,\cdots,\mathrm{size}_{f_i})$

$\mathrm{Ideal}_{\mathrm{A,S}}(k)$:

$(d_0,\{\mathrm{id}_i,\mathrm{data}_i\}_{i=1}^{t})\leftarrow \mathrm{A}(k)$

$D'\leftarrow \mathrm{S}^{L_1(\delta,f)}(k)$

for $1\leqslant i\leqslant q$

$q_i \xleftarrow[\text{each time}]{\text{one query}} \mathrm{A}(D',\mathrm{size}'_1,\cdots,\mathrm{size}'_{f_{i-1}})$

$\mathrm{size}'_{f_i}\xleftarrow{\mathrm{A}}\mathrm{Update}(\mathrm{S}^{L_2(\delta,f)}(k);\mathrm{A}:D')$

or $\mathrm{size}'_{f_i}\xleftarrow{\mathrm{A}}\mathrm{Download}(\mathrm{S}^{L_2(\delta,f)}(k);\mathrm{A}:D')$

or $\mathrm{size}'_{f_i}\xleftarrow{\mathrm{A}}\mathrm{Upload}(\mathrm{S}^{L_2(\delta,f)}(k);\mathrm{A}:D')$

output $b\leftarrow \mathrm{A}(D',d_0,\mathrm{size}'_1,\cdots,\mathrm{size}'_{f_i})$

如果对于所有具有多项式时间计算能力的敌手 A,存在一个模拟器 S,使得

$$|\Pr[\mathrm{Real}_{\mathrm{A}}(k)=1]-\Pr[\mathrm{Ideal}_{\mathrm{A,S}}(k)=1]|\leqslant \mathrm{negl}(k),$$

其中 $\mathrm{negl}(k)$ 为以 k 为输入的可忽略函数,称该方案是 L_1L_2 动态自适应安全的.

3 隐藏访问模式的高效安全云存储方案

3.1 设计思路

方案主要分为构建存储结构和动态访问两部分. 在构建存储结构时,用户首先要对上传至服务器的文件进行处理,将每个文件转换成多个固定形式的数据块格式并进行加密,然后存放在云服务器存储结构的随机位置上;之后用户可以对存储结构进行动态访问,包括下载文件、上传新文件及更新文件等. 在下载过程中,用户需要对服务器进行两轮访问,从云服务器的存储结构中读取出想要查找的文件所对应的数据块,并对其进行解密和整合,最终得到完整的文件内容;更新文件则在两轮访问的基础上,对文件进行修改并重新上传到云服务器中随机位置.

3.2 详细算法与协议

方案选择完全定义域抗碰撞哈希函数 H,伪随机函数 F,完全定义域伪随机函数 P 和一个伪随机生成器 G. 设存储空间 D 中数据块的个数为 n_D,每个数据块的大小为 m_D. 其中为了解决数据块存储中可能出现的冲突问题,引入了膨胀参数 α 来生成比实际数据块个数大的随机序列,并且将每轮交互中的最少操作数据块个数设置为 κ.

1) $\mathrm{KeyGen}(k)$:密钥生成算法. 算法输入安全参数 k,生成伪随机函数 F 的密钥 k_F 以及完全定义域伪随机函数 P 的密钥 k_P,得到存储结构的密钥 $K=(k_F,k_P)$.

2) $\mathrm{Build}(K,d_0,\{\mathrm{id}_i,\mathrm{data}_i\}_{i=1}^{t})$:构建算法. 用于初始化构建存储结构. 算法的主要步骤如下:

设文件集合为 $\{\mathrm{id}_i,\mathrm{data}_i\}_{i=1}^{t}$,其中任意一个文件 $f=\{\mathrm{id}_f,\mathrm{data}_f\}$,$\mathrm{id}_f$ 为文件标志符,data_f 为文件内容. 对 data_f 进行分块处理,每一个数据块都具有两个短头部域:包括初始化为 0 的版本号和 $H(\mathrm{id}_f)$. 除此之外,第一个数据块需要多保存一个头部域以存储记录文件包含的数据块个数 size_f.

对于文件列表中的每一个文件 f:

①利用 id_f、伪随机函数 P 和对应的密钥 k_P,生成伪随机生成器 G 的种子 $\sigma_f=P_{k_P}(\mathrm{id}_f)$;

②将种子 σ_f 作为伪随机生成器 G 的输入,在得到的伪随机序列中选取序列长度为 $\max(|\alpha\cdot\mathrm{size}_f|,\kappa)$ 的前段作为 S_f;

③选择大小为 size_f 的伪随机子序列 $\hat{S}_f\subseteq S_f$,并按增序将文件的 size_f 个数据块写入 D 中对应位置,然后将这些数据块标记为非空;

④对文件的每个数据块进行处理:对于第 i 个数据块 $D[i]$,将其表示为 $v_i\parallel B[i]$(v_i 是版本号),使用伪随机函数 F 和密钥 k_F 处理 $B[i]$,将 $B[i]$ 更新为 $B[i]\oplus F_{k_F}(v_i\parallel i)$.

最后将构建好的存储结构 D 上传至云服务器.

3) $\mathrm{Upload}(\mathrm{U}:\mathrm{id}_f,\mathrm{data}_f,\mathrm{size}_f,K;\mathrm{S}:D)$:上传文件协议. 主要步骤如下:

用户:在已知文件标志 id_f,文件内容 data_f 和文件大小的情况下,利用构建算法构建大小为 size_f 的数据块,发送至云服务器.

服务器：接收到 size_f 个数据块后，将其按照 $\hat{S}_f$ 的增序存入 $D[\hat{S}_f]$ 的空闲位置上，并将这些数据块标记为非空. 得到更新后的结构 D'.

4）Download(U：id_f, K; S：D)：下载文件协议. 协议的主要步骤如下：

用户：利用文件标志 id_f 和密钥 k_P，计算 $\sigma_f = F_{k_F}(\text{id}_f)$，并定义大小为 κ 的集合 S_f^0，其包括了序列 $\Lambda[\sigma_f, \kappa]$ 中前 κ 个整数，并向云服务器发送检索 S_f^0 数据块的请求.

服务器：按照在 $\Lambda[\sigma_f, \kappa]$ 中的顺序将 $D[S_f^0]$ 返回给用户 U.

用户：对返回的数据块进行解密，得到 $D[i] = (v_i \parallel B[i])$. 找到一个标志为 id_f 的数据块，并从其头部域恢复出文件的大小 size_f. 令 $l = \lceil \alpha \cdot \text{size}_f \rceil$，如果 $l < \kappa$，设 $S_f = S_f^0$，否则设 S_f 为在 $\Lambda[\sigma_f, l]$ 中整数的集合. 最后向云服务器请求返回数据块集合 $D[S_f / S_f^0]$.

服务器：将数据块集合 $D[S_f / S_f^0]$ 返回给用户.

用户：解密通过 S_f 被检索的剩余的数据块 $D[S_f / S_f^0]$. 设 $\hat{S}_f$ 表示属于被读取文件的数据块的标志符的集合(即通过检查它们的头部是否匹配 id_f). 如果 $\hat{S}_f$ 不为空，那么按照这些数据块的标志符的增序将它们合并起来，从而恢复文件的内容 data_f.

5）Update(U：id_f, data_f, K; S：D)：更新文件协议. 用于完成客户端用户 U 更新云服务器中某个文件的内容. 协议的主要步骤如下：

用户：执行步骤 4)，将待更新的文件 id_f 下载到本地，随后对文件内容进行修改和更新，命名为 data′. 同时对新内容 data′ 分块编码为 size_f' 个数据块.

①选择 S_f 中一个大小为 size_f' 的子集 $S_f' \subseteq S_f$，在序列 $\Lambda[\sigma_f, l]$ 中找到包含 size_f' 个数据块的一个最短前缀，并且将这些数据块都标记为属于 id_f (即存在于 $\hat{S}_f$ 中)或者将它们都标记为空. 如果 $\text{size}_f' < \text{size}_f$，那么 $\hat{S}_f' \subseteq \hat{S}_f$；否则，$\hat{S}_f \subseteq \hat{S}_f' \subseteq S_f$.

②对所有通过 S_f 检索到的数据块进行处理. 即对于第 i 个数据块 $D[i]$，首先将版本号 v_i 加 1，使用伪随机函数 F 和密钥 k_F 处理 $B[i]$，将 $B[i]$ 更新为 $B[i] \oplus F_{k_F}(v_i \parallel i)$，最终将数据块表示为更新后的 $v_i \parallel B[i]$.

③将加密后的数据块发送给云服务器 S.

服务器：将接收到的数据块更新到 D 中通过 $\hat{S}_f'$ 被检索到的数据块中. 如果 $\text{size}_f' < \text{size}_f$，就将通过 $\hat{S}_f \backslash \hat{S}_f'$ 检索到的数据块置空. 上传被重新加密的新的数据块到服务器中. 要说明的是，$D[S_f]$ 中所有下载得到的数据块在这一步都被重新上传，同时它们的版本号加 1，并重新进行加密. 最后得到更新后的存储结构 D'.

4　安全性证明

针对存储结构构建阶段和访问阶段，分别定义两个泄露函数 L_1 和 L_2 来描述在方案执行期间向服务器泄露的信息. 具体内容如下：

1）构建阶段. 此阶段用户通过输入密钥 K 以及 $(d_0, \{\text{id}_i, \text{data}_i\}_{i=1}^t)$ 生成数据块数组 D，并且将数据块数组 D 发送给服务器，在此过程中，服务器得到的信息为 d_0，即存储的数据块的总数量的上限. 将此信息记为 L_1，即 $L_1((d_0, \{\text{id}_i, \text{data}_i\}_{i=1}^t)) = (d_0)$.

2）访问阶段. 此阶段服务器能够获得的信息主要分为以下三类：①服务器可以获得用户当前进行访问的操作 op；②在某次访问质询过程中，当前需要操作的文件在之前最后一次被访问的情况 j，如果 $j = 0$，则认为此文件之前没有被访问过；③在某次访问质询中，被访问的文件大小 size. 将这些信息记为 L_2，即 $L_2(\text{id}, \text{op}) = (\text{op}, j, \text{size})$. 除此之外，定义存储松弛参数 γ，即服务器中数据块的数量 n_D 与文件被分块处理后的数据块的数量的比值. 需要注意的是，γ 会随着文件的添加(或更新成更大的)而减小，随着文件的删除(或者更新成更小的)而增大.

定理 1　若伪随机函数 F，P 和伪随机生成器 G 具有 CPA 安全性，哈希函数 H 具有抗碰撞性，且方案保证 γ 至少为 $2/(1-1/\alpha)$，并且 $n_D \geqslant \kappa = \omega(\log k)$，则本方案在标准模型下针对诚实并好奇的敌手具有 L_1L_2 - 自适应安全性.

证明　本文构造如下游戏序列来证明任何具有多项式时间计算能力的敌手都不能区分真实的和理想的实验.

游戏 1：这个游戏对应一次真实实验 $\text{Real}_A(1^k)$ 的执行，用户利用敌手提供的文件数据进行存储结构构建，对于任意文件 $f = (\text{id}_f, \text{data}_f)$，利用伪随机函数 P 和哈希函数 H 对其进行分块处理，$\sigma_f = P_{k_P}(\text{id}_f)$，并利用伪随机函数 F 对数据块进行加密. 然后将包含所有文件的数据块的存储结构 D 发送给敌手. 再后，A 向用户发起至多 q 次的质询，其中 q 为多项式级. 在所有 q 次质询结束后，A 所收集到的所有信息构成了 $\text{view}_{\text{Real}}$. A 最后输出一个比特作为整个实验的输出.

游戏 2:游戏 2 与游戏 1 相同,除了以下内容:在构建存储过程中,与敌手交互的不是真正用户,而是一个模拟器 S. 在初始化过程中,S 不能利用伪随机函数 P、哈希函数 H 以及伪随机函数 F 来生成真正的加密数据块,而是利用泄露函数 L_1 的内容,并通过 D 的大小 n_D,构建模拟包含加密数据块的 D';它通过挑选均匀随机比特串来模拟 D 中数据块的内容,其中 D'中每个数据块的版本号都设置为 0.

如果伪随机函数 P 以及伪随机函数 F 的伪随机性成立,哈希函数 H 的抗碰撞性成立,则对于所有多项式时间计算能力的敌手 A,

$$|\Pr[1\leftarrow \mathrm{A}(D,d_0)] - \Pr[1\leftarrow \mathrm{A}(D',d_0)]| \leqslant \varepsilon_1.$$

假设概率 ε_1 是不可忽略的,那么这意味着敌手可以区分包含真实加密数据块的存储结构 D 和包含模拟加密数据块的存储结构 D'. 由于在存储结构 D 中的真实加密数据块是使用伪随机函数产生的,而在 D'中的模拟加密数据块是由随机比特填充的,因此产生了一个矛盾,即敌手能够以不可忽略的概率来区分伪随机函数. 因此概率 ε_1 是可忽略的.

游戏 3:这个游戏对应于一次理想实验 $\mathrm{Ideal}_{\mathrm{A,S}}(1^k)$的执行,与游戏 2 的区别为,与敌手交互的模拟器用到的信息全部为泄露函数 L_2 的内容. 除此之外,另一个区别是异常中止,在之前的游戏中,用户在伪随机子集中不能找到足够的空闲数据块时方案会中止,然而模拟器则不会中止.

如果方案保证存储松弛参数 $\gamma \geqslant 2/(1-1/\alpha)$,并且 $n_D \geqslant \kappa = \omega(\log k)$,则对于所有多项式时间计算能力的敌手 A,

$$|\Pr[1\leftarrow \mathrm{A}(\{\mathrm{op},j,\mathrm{size}\}_1^i)] - \Pr[1\leftarrow \mathrm{A}(\{\mathrm{op}',j',\mathrm{size}'\}_1^i)]| \leqslant \varepsilon_2.$$

在与敌手进行交互之前,S 初始化一个映射,其形式为$(j;\Lambda_j,\mathrm{size}_j)$,它是将一个整数 j(指示访问的序列号)映射成一个在 D 中的数据块序列和在数据块中被访问的文件的大小. 对于敌手第 j 次访问 j^*,首先模拟器通过泄露函数 L_2 创建表条目$(j^*,\Lambda_{j^*},\mathrm{size}_{j^*})$. 考虑以下两种类型.

情况 1:$j=0$,那么 S 会在$[0,n_D]$的范围内选取一个由 l 个不同的整数组成的随机序列,记为 Λ_{j^*}.

情况 2:$j>0$,如果$|\Lambda_j| \geqslant l$,设置 $\Lambda_{j^*}=\Lambda_j$;否则 $j>0$,$|\Lambda_{j^*}|<l$,将 Λ_j 扩展成长度为 l 的随机序列,记为 Λ_{j^*}.

接下来,S 创建模拟 view,即服务器首次接收到通过 Λ_{j^*}得到的首个 κ 条目所检索到的要下载的 κ 个数据块;如果 $l>\kappa$,则下载通过 Λ_{j^*}中的下一个条目所检索到的数据块. 而对于读以外的操作,应该紧接着上传一个新版本(包括数据块版本号的递增,以及作为内容的新的随机字符串),这个新版本是通过 Λ_{j^*}中的首个 l 条目所检索到的数据块的新版本.

方案满足$(d/n_D) \geqslant 2/(1-1/\alpha)$,即 $1-2(d/n_D) \geqslant 1/\alpha$. 因此,除了可忽略概率,对于选择$|S_f|$个数据块,有$(1-2(d/n_D)) \geqslant \mathrm{size}_f$ 个空闲数据块. 同理,S_f^0 至少会有$|\kappa/\alpha|$个空闲数据块. 所以,用户向存储结构中写文件时方案异常中止的概率是可忽略的,即 ε_2 是可忽略的.

综上所述,对于所有敌手 A,实验 $\mathrm{Real}_{\mathrm{A}}(1^k)$与实验 $\mathrm{Ideal}_{\mathrm{A,S}}(1^k)$的输出是一致的,除了可忽略的概率 $\mathrm{negl}(k)$,即

$$|\Pr[\mathrm{Real}_{\mathrm{A}}(1^k)=1] - \Pr[\mathrm{Ideal}_{\mathrm{A,S}}(1^k)=1]| = \mathrm{negl}(k).$$

因此,方案满足 L_1L_2 动态自适应安全.

5 性能分析

5.1 方案评估与对比

将本文构建的方案进行全面的性能评估,并与近几年经典的能隐藏访问模式的 ORAM 方案[6-9]进行效率对比分析,结果见表 1.

表 1 不同方案的性能对比

Table 1 Performance comparison of different schemes

方案	数据块 B 大小	洗牌代价	通信代价	存储代价$_{\text{服务器}}$	存储代价$_{\text{客户端}}$	安全性
文献[6]	$O(\log^2 n)$	$O(\log^2 n/\log B)$	$O(\log n)$	$O(n)$	$O(\log^2 n)$	SH
文献[7]	$\Omega(\log n)$	$\Omega(n)$	$O(\log^3 n)$	$O(n)$	$O(1)$	SH
文献[8]	$O(\log n)$	$O(\log^2 n)$	$O(\log n)$	$O(n\log n)$	$O(\log n)$	SH
文献[9]	$\Omega(\log^6 n)$	$O(B\log n)$	$O(B)$	$O(B\log^2 n)$	$O(B+\log^2 n)$	M
本文	$O(\log^2 n)$	N/A	$O(B)$	$O(n)$	$O(1)$	SH

注:SH(semi honest),半可信服务器模型; M(malicious),恶意服务器模型.

表 1 对比分析了本方案与其他方案的各个指标,从表中可以看出:安全性方面,本方案主要抵抗半可信服务器;通信代价方面,客户端与服务器交互时的通信代价为 $O(B)$,与数据块大小相关,相较于文献[6,8,9]的方案来讲,本方案的通信代价并非与文件总数量直接相关,通信负担较小;除此之外,在存储代价方面,相较于文献[6,9]提出的方案,本方案对服务器端预计客户端的存储代价进行了改进. 值得说明的是,本方案无需本地存储任何文件相关数据,仅需存储密钥等参数,释放了本地存储空间;而且相较于其他所有方案,本方案没有 ORAM 特有的洗牌过程,并且无需服务器对数据本身进行任何计算,降低了服务器的计算代价.

5.2　实验分析

本文对方案进行了实验分析. 实验使用了 2015 年最新版本的 Enron 数据集,在邮件集中选取 3 000 个邮件作为实验文件集合. 分析结果如图 2 所示. 图 2a 表明,随着文件数量的增多,系统构建存储耗时增加,但增幅较小. 文件大小对上传文件耗时的影响如图 2b 所示. 从图 2c 可以看出,文件体积越大,下载耗时越长,并且 A 类文件和 B 类文件的下载耗时相差较大,而 B 类、C 类、D 类文件的下载耗时相差较小且增幅较缓. 更新内容大小对更新文件耗时的影响如图 2d 所示,随着更新内容的增加,更新耗时也不断增加,但增幅较小. 综上所述,本文提出的隐藏访问模式的高效安全云存储方案在时间上有良好的效率,具有较高的实际应用价值.

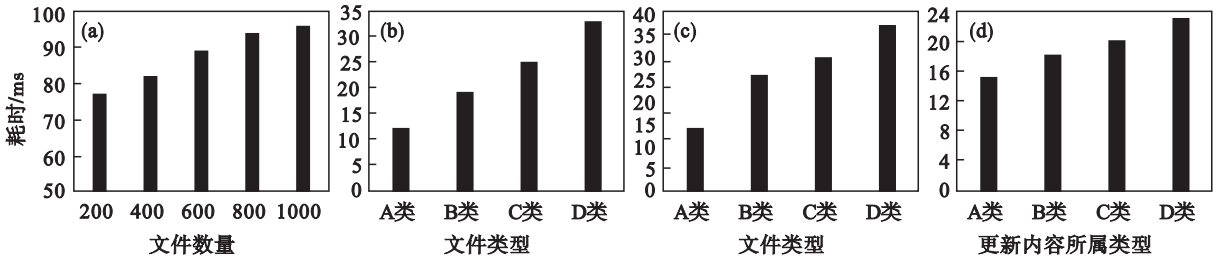

图 2　本文方案的实验分析

Fig. 2　Experimental analysis of the proposed scheme

(a)—构建存储耗时;(b)—上传文件耗时;(c)—下载文件耗时;(d)—更新文件耗时.

6　结　　语

本文提出的隐藏访问模式的高效安全云存储方案,既能解决云存储系统中的数据机密性问题,又能抵抗访问模式泄露. 方案设计二层访问协议,使得用户在不泄露访问模式的前提下,实现对于文件的高效上传、下载及更新,降低了存储代价和访问交互次数. 安全性方面方案满足 L_1L_2 - 动态自适应安全性. 性能对比分析与实验结果表明,本方案的提出对于当前云存储中的数据安全问题有一定的理论意义和实际应用价值.

参考文献:

[1] Cao N, Wang C, Li M, et al. Privacy-preserving multi-keyword ranked search over encrypted cloud data[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(1):222 - 233.

[2] Chor B, Goldreich O, Kushilevitz E, et al. Private information retrieval[C]// Proceedings of the 36th Annual Symposium on Foundations of Computer Science. New York: IEEE, 1995: 41 - 50.

[3] Jarecki S, Jutla C, Krawczyk H, et al. Outsourced symmetric private information retrieval[C]//Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. New York: ACM, 2013: 875 - 888.

[4] Hazay C, Zarosim H. The feasibility of outsourced database search in the plain model[C]//International Conference on Security and Cryptography for Networks. [S. l.]: Springer International Publishing, 2016: 313 - 332.

[5] Goldreich O. Towards a theory of software protection and simulation by oblivious RAMs[C]//Proceedings of the 19th Annual ACM Symposium on Theory of Computing. New York: ACM, 1987: 182 - 194.

[6] Stefanov E, van Dijk M, Shi E, et al. Path ORAM: an extremely simple oblivious RAM protocol[C]//Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. New York: ACM, 2013: 299 - 310.

[7] 孙晓妮,蒋瀚,徐秋亮. 基于二叉树存储的多用户 ORAM 方案[J]. 软件学报,2016,27(6):1475 - 1486.
(Sun Xiao-ni, Jiang Han, Xu Qiu-liang. Multi-user binary tree based ORAM scheme[J]. *Journal of Software*, 2016, 27(6): 1475 - 1486.)

[8] Ren L, Fletcher C W, Kwon A, et al. Ring ORAM: closing the gap between small and large client storage oblivious RAM[J/OL]. [2017 - 01 - 15]. http://citeseerx. ist. psu. edu/viewdoc/download? doi = 10. 1. 1. 691. 5259& rep1& type = pdf.

[9] Devadas S, van Dijk M, Fletcher C W, et al. Onion ORAM: a constant bandwidth blowup oblivious RAM[C]//Theory of Cryptography Conference. Berlin: Springer Berlin Heidelberg, 2016: 145 - 174.