软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(6):1475−1486 [doi: 10.13328/j.cnki.jos.005002]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于二叉树存储的多用户 ORAM 方案*

孙晓妮, 蒋 瀚, 徐秋亮

(山东大学 计算机科学与技术学院,山东 济南 250101)

通信作者: 蒋瀚, E-mail: jianghan@sdu.edu.cn

摘 要: 随着大数据及数据挖掘技术的发展,云计算环境中用户访问模式成为泄露用户隐私的一条途径.不经意随机存取技术(ORAM)是保护用户访问模式的一条有效途径.现有的 ORAM 方案中,大部分只支持单个用户,而唯一支持多用户的 ORAM 方案是基于分层 ORAM 方案设计的,但其混淆过程的计算复杂度高.为了避免出现混淆过程,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.首先,改进了一个代理加密方案,然后在多个用户和服务器之间引入一个代理,利用改进的代理加密机制,将不同用户加密的数据,通过代理再次加密成相同密钥加密的数据存储到服务器.该方案的安全性基于伪随机函数的不可区分性,其最差情况下的计算复杂度和平均计算复杂度均为 $O(\log^2 n)$,比现有的多用户 ORAM 方案的效率要高.

关键词: 云计算;二叉树;不经意随机存取;多用户;访问模式

中图法分类号: TP309

中文引用格式: 孙晓妮,蒋瀚,徐秋亮.基于二叉树存储的多用户 ORAM 方案.软件学报,2016,27(6):1475−1486. http://www.jos.org.cn/1000-9825/5002.htm

英文引用格式: Sun XN, Jiang H, Xu QL. Multi-User binary tree based ORAM scheme. Ruan Jian Xue Bao/Journal of Software, 2016,27(6):1475−1486 (in Chinese). http://www.jos.org.cn/1000-9825/5002.htm

Multi-User Binary Tree Based ORAM Scheme

SUN Xiao-Ni, JIANG Han, XU Qiu-Liang

(School of Computer Science and Technology, Shandong University, Ji'nan 250101, China)

Abstract: With the development of big data and data mining technology, the access pattern becomes a risk of leaking user's privacy in the cloud computing environment. Oblivious random access memory (ORAM) is an effective way to protect the user's access pattern. The existing ORAMs mostly support a single user. The only ORAM supporting multi-user is based on the hierarchical ORAM including a reshuffling phase that may cause high computational complexity. In order to avoid reshuffling, this paper designs a new multi-user ORAM based on binary tree. First, a proxy encryption scheme is improved. Second, a proxy between users and the cloud server is introduced. The data encrypted by different users is encrypted again by the proxy to obtain the final ciphertext encrypted by the same key, and the final ciphertext is stored on the server. The security of the scheme is based on the indistinguishability of the pseudorandom function, and the worst computational complexity and the amortized computational complexity are all $O(\log^2 n)$, achieving higher efficiency than the existing multi-user ORAM schemes.

Key words: cloud computing; binary tree; oblivious random access memory; multi-user; access pattern

云计算作为一种新的计算方式,以其可靠性、共享性、方便性、隔离性、兼容性、低成本、高伸缩性等优势获得了迅速的发展[1−3].云计算在发展过程中面临着很多问题,其安全问题尤为突出[4,5],已经成为用户衡量是

* 基金项目: 国家自然科学基金(61173139, 61572294); 教育部博士点基金(20110131110027)

Foundation item: National Natural Science Foundation of China (61173139, 61572294); Ph.D. Programs Foundation of Ministry of Education of China (20110131110027)

收稿时间: 2015-08-15; 修改时间: 2015-10-09; 采用时间: 2015-12-05; jos 在线出版时间: 2016-01-21

CNKI 网络优先出版: 2016-01-22 11:20:05, http://www.cnki.net/kcms/detail/11.2560.TP.20160122.1120.014.html

否使用云计算服务的一个重要因素,获得了密码研究者极大的关注.

云计算的安全包括数据机密性、数据完整性以及隐私保护.云存储数据的机密性可以通过现有的加密方案来获得.云存储数据的完整性可以通过哈希函数、消息认证码、数字签名等来获得[6].云计算的隐私保护包括身份隐私保护、查询隐私保护和访问模式隐私保护.身份隐私保护可以通过匿名访问来实现;查询隐私保护可以通过安全索引、随机陷门等方法来实现;访问模式隐私保护可以通过动态数据结构、不经意随机存取等方法来实现.

随着数据挖掘技术的迅速发展,访问模式成为泄露用户隐私的一种潜在途径.访问模式指的是在运行过程中所访问的地址序列[7].2010 年 Pinkas[8]指出服务器通过分析用户的访问模式,基于一定的背景知识,可以推断出用户的部分隐私信息,如果服务器监测到用户连续进行特定的几次访问后,都会进行一次股票交易,那么,当用户发起相同或者相似的访问序列时,即使这些内容是加密的,恶意的服务器也可以以极高的概率推断用户将要进行一次股票交易,进而推断出这些序列的内容.此外,服务器还可以统计用户每次访问返回的数据块的次数,推断出数据库中不同区域数据的重要程度.由此可见,在云计算中,隐藏用户的访问模式对于保护用户隐私至关重要.

不经意随机存取(oblivious random access memory,简称 ORAM)技术是隐藏用户访问模式的一种重要方法.当执行一个普通的 RAM 程序时,ORAM 可以将其转换为多个 RAM 步骤,以达到不泄漏访问模式的目的.最平凡的 ORAM 方案是当用户发起请求时,云服务器顺序读取每个数据项,发送给用户,然后用户再将每一个数据项重新加密后发布到云服务器,这样用户既获得了自己需要的数据,也使服务器不知道用户需要的是哪些位置的数据项,从而保护了用户的访问模式.由于平凡的 ORAM 需要传输并加解密所有的数据项,因此,其计算复杂度和通信复杂度巨大,根本无法应用.

基于软件保护的背景,1996 年 Goldreich 和 Ostrovsky[7]提出了两种非平凡的 ORAM 方案:平方根 ORAM 方案和分层 ORAM 方案,其中最高效的方案,对于 n 个数据项,服务器端需要的存储空间为 $O(n\log n)$,而访问每一个数据项,平均需要 $O(\log^3 n)$次数据请求,最差的情况下需要 $O(n\log^2 n)$次数据请求.由于该方案结构复杂,效率不高,因此 ORAM 技术在保护访问模式中的应用研究没有得到广泛的关注.2010 年,Pinkas[8]描述了访问模式泄露对于用户隐私的危害,指出 ORAM 方案可以应用于访问模式保护,并改进了文献[7]中提出的分层 ORAM 方案,提高了效率.该方案中服务器端的数据是分层存储的,当用户访问某数据项时,服务器利用布谷鸟散列计算出该数据项对应每一层的两个位置,将这些位置存储的数据项发给用户,用户通过解密这些数据项,从而获取想要访问的数据内容,此后为了保证数据存储的随机性,用户还需要将这些数据项重新加密写回服务器,并进一步混淆数据项的存储位置.该方案的用户空间复杂度为 $O(1)$,服务器空间复杂度为 $O(n)$,平均计算复杂度为 $O(\log^2 n)$,其中混淆操作最差情况下的计算复杂度为$\Omega(n)$,导致最差情况下的计算复杂度为 $O(n\log n)$.2011 年,Goodrich 和 Mitzenmache[9]发现 Pinkas 的方案在某些情况下,服务器可以高概率地获知用户的访问序列,因而他们提出了一种新的 ORAM 方案,该方案的空间复杂度和计算复杂度与 Pinkas 方案保持同一个数量级.此后,分层 ORAM 方案得到了进一步的研究[10−13].

由于分层 ORAM 方案中混淆过程效率较低,为了避免混淆过程,2011 年,Shi 等人[14]提出了基于二叉树的 ORAM 方案.在该方案中,服务器端数据项以二叉树的形式存储,用户端存储一个索引表,当用户访问某数据项时,用户从索引表中读取到该数据项对应的叶子节点,从而得到包含该数据项的路径(从根节点到其叶子节点的通路),并提供给服务器,服务器依次返回该路径上每一个数据项,用户通过解密数据项,可以获取想要访问的数据内容,此后为了保证数据存储的随机性,用户将他要访问的数据项重新加密后以新的路径存储到服务器,并且使用一个空数据项代替他想要搜索的数据项,与其他数据项一起重新加密后存储到原路径.该方案的平均计算复杂度最好可达到 $O(\log^2 n\log\log n)$,最差情况下的计算复杂度为 $O(\log^3 n\log\log n)$,有效改进了最差情况下的计算复杂度.2013 年 Stefanov 等人[15]采用树的思想,提出了 Path ORAM 方案,极大地改进了文献[14]的方案,将计算复杂度改进为 $O(\log^2 n/\log c)$,其中 c 是描述数据分块大小的参数.但该文并没有对 Path ORAM 方案的安全性进行形式化证明,后期有很多学者基于 Path ORAM 方案进行改进,并给出形式化的安全证明[16,17].

在已有的绝大多数 ORAM 方案中,都是针对单一的用户,或者说多个相互信任的用户(共享秘密密钥)[12,18],但是云并不是为一个用户服务的,实现多用户的ORAM方案具有很强的应用背景.2014年,Zhang等人[19]首次提出了真正意义上的多用户ORAM方案,该方案是基于分层ORAM方案设计的,通过在用户和服务器之间引入匿名器,借助代理加密机制,将不同用户加密的数据,通过匿名器再次加密为相同密钥加密的数据存储到服务器,为了防止单一匿名器权限过大,他们将单一匿名器设计为一系列串联匿名器.Zhang 等人[19]的方案中,用户的空间复杂度为 $O(1)$;服务器端的空间复杂度为 $O(n\log\log n)$;在问询阶段,用户和匿名器的计算复杂度都是 $O(\log n\log\log n)$;在混淆阶段,用户不需要参与,匿名器独立完成该过程,其计算复杂度为 $O(\log^3 n\log\log n)$.该方案进行安全证明时要求充当代理的多个匿名器不能同时妥协.

本文提出了基于二叉树的多用户ORAM方案(binary tree based multi-user ORAM,简称BT-M- ORAM).本文方案中,服务器端采用基于二叉树的 ORAM 方案代替 Zhang 方案[19]中的分层 ORAM 方案,避免了最差情况下分层 ORAM 方案中混淆过程带来的$\Omega(n)$的高计算复杂度.在本文方案的应用背景下,系统中的每一个用户都可以搜索所有用户的数据信息,不需要授权管理功能,因此本文只采用 Dong 等人[20]提出的代理加密方案,同时,为了降低代理端的空间复杂度,对 Dong 等人的方案进行了改进,代理端只需要保存代理主密钥(一个常数和一个哈希函数)就可以实现代理端密钥的计算,并不需要保存每个用户对应的代理密钥.本文提出的 BT-M-ORAM 方案在最差情况下的计算复杂度为 $O(\log^2 n)$,比 Zhang 等人[19]的方案中混淆阶段最差情况下带来的$\Omega(n)$有很大的改进;平均计算复杂度为 $O(\log^2 n)$,比 Zhang 等人[19]的混淆阶段引起的平均计算复杂度 $O(\log^3 n\log\log n)$也要小.

1 预备知识

1.1 基于二叉树的ORAM方案

1.1.1 符号声明

n:需要存储的数据项数目.

D:构造的二叉树的深度 $D=\lceil\log n\rceil$.

s:数据 id,每个数据项拥有唯一的 id.

d:数据内容.

l:数据存储路径所对应的叶子节点.

$P(l)$:从叶子节点 l 到根节点这一条路径.

B:数据明文,形式为 $B=(s,d)$.

B^+:数据明文,形式为 $B^+=(s,d,l)$.

$\perp$:空数据块,数据 id 为 0,数据结构同正常数据(s,d)相同.

index:索引表,用于存储每一个数据项的存储路径所对应的叶子节点,索引表项为(s,l).

index[s]:s 数据项存储路径所对应的叶子节点的编号 l.

x_i^u:用户 u_i 的密钥.

x_i^p:用户 u_i 对应的代理密钥.

1.1.2 基于二叉树的 ORAM 方案

基于二叉树的 ORAM 方案[14]涉及两个主体:用户和服务器.用户既充当数据发布者,又充当数据查询者,用户既可以发布自己的数据到云服务器上,也可以向云服务器发起搜索请求,访问自己的数据.云服务器上数据以二叉树的形式存储,二叉树中的每个节点可以看做是一个独立的 ORAM,每个节点 ORAM 可以存储 $O(\log n)$个数据项.ORAM 的 3 种基本操作如下:

• *ReadAndRemove*(s):当用户想要访问 id 为 s 的数据项时,首先查询本地索引表 *index*,获得该数据项对应的叶子节点 l,然后产生一个随机 D 位的 01 串 l^*,将其存储在一个全局变量中,并将 *index*[s]改写为 l^*.依次读取 $P(l)$路径上的每一个数据项,当读到 s 数据项时,记住其内容,并将其从该节点删除,用$\perp$代替,重新加密写回;当没有读

到 s 数据项时,将读出的数据项重新加密写回.最后,如果找到了 s 数据项,返回其内容,否则返回⊥.

• *Add*(s,d):首先,从全局变量中读取 l,l 是由前面的 *ReadAndRemove*(s)操作产生的,然后用户将(s,d,l)写入根节点.

• *evict*(w):每个 *Add* 操作之后,都要进行该操作.用户从服务器端二叉树的每一层随机选择 w 个节点进行数据下移操作.对选择的每一个节点,用户选取一个数据项(如果该节点有真实数据项,随机选取一个真实数据项,用⊥代替原数据项;如果该节点都是⊥数据项,则选取⊥数据项).如果选取的是⊥数据项,用户向该节点的左右孩子节点均写入重新加密后的⊥数据项;如果选取的是真实数据项,根据该数据项对应的叶子节点 l,将真实数据项重新加密后写入 $P(l)$指定的那个孩子节点,并将⊥数据项重新加密后写入另外一个孩子节点.注意:每次写入左右孩子节点的顺序是固定的.

ORAM 执行读或者写操作时,首先执行 *ReadAndRemove*(s),用户读取 id 为 s 的数据项(s,d,l),并将数据项从二叉树中删除(如果需要改写数据内容,则修改 d)而后执行 *Add*(s,d),将(s,d,l^*)重新写回二叉树.

1.2 代理加密方案系统模型

代理加密系统由系统初始化、密钥生成、用户加密、代理加密、代理解密、用户解密 6 部分组成,即 $\{Init(1^\kappa), KeyGen(x,i), UEnc(x_i^u,m), PEnc(x_i^p,c_i^*(m)), PDec(x_j^p,c(m)), UDec(x_j^u,c_j'(m))\}$ 形式化描述如下:

•系统初始化:可信的密钥分发中心执行,以 κ 为安全参数,初始化系统,生成公开参数 G,g,p 以及秘密参数 x.记为 $Init(1^\kappa)\to\{G,g,p,x\}$.

• 密钥生成:可信的密钥分发中心执行,新加入用户 u_i 时,生成用户密钥 x_i^u.记为 $KeyGen(x,i)\to x_i^u$.

• 用户加密:用户使用密钥 x_i^u,加密明文 m,生成一级密文 $c_i^*(m)$.记为 $UEnc(x_i^u,m)\to c_i^*(m)$.

• 代理加密:代理使用用户 u_i 对应的代理密钥 x_i^p,再次加密一级密文 $c_i^*(m)$,生成最终密文 $c(m)$.记为 $PEnc(x_i^p,c_i^*(m))\to c(m)$.

• 代理解密:代理使用用户 u_j 对应的代理密钥 x_j^p,对密文 $c(m)$进行解密,获得一级密文 $c_j'(m)$.记为 $PDec(x_j^p,c(m))\to c_j'(m)$.

• 用户解密:用户 u_j 使用密钥 x_j^u,解密一级密文 $c_j'(m)$,获得明文 m.记为 $UDec(x_j^u,c_j'(m))\to m$.

2 代理加密方案

BT-M-ORAM 方案中多用户的实现依赖于在用户与服务器之间引入一个代理,数据先通过用户进行初次加密,代理再次加密后变成统一密钥加密的密文,从而实现多个用户参与的 ORAM.代理重加密方案已经得到了广泛的研究,取得了大量的成果,新成果大多关注用户数据授权的研究,数据拥有者可以指定特定的数据使用者来解密被重加密的数据,这样可以提供更加灵活的应用方式,但相比于无授权的代理加密方案,其代价是牺牲了一定的效率.在本文系统背景下,允许用户搜索系统中所有用户的数据,所以不需要涉及到授权,而且 Dong 等人[20]的无授权代理加密方案为基础.由于 Dong 等人的方案中代理端需要存储系统中每个用户的代理密钥,并不适合云环境下海量用户的应用场景.因此,我们改进 Dong 等人的方案,得到一种基于 El Gamal 机制的代理加密方案 *PE*,代理端只需要保存一个常数和一个哈希函数就可以实现代理密钥的计算,并不需要保存每个用户对应的代理密钥.

2.1 代理加密方案

• $Init(1^\kappa)$:可信的密钥分发中心执行.选取一个大素数 q,选取一个循环群 G,g 是群 G 的生成元,公共参数是(G,g,q).从 Z_q 中任意选取一个 t,取一个哈希函数 $h(\cdot,\cdot):Z_q\times Z\to Z_q$.密钥分发中心把 t 和 $h(\cdot,\cdot)$发送给代理,作为代理主密钥.从 Z_q 中选取一个 x,作为秘密的主密钥.

• *KeyGen*(x,i): 可信的密钥分发中心执行.用户 u_i 加入系统时,密钥分发中心执行该算法.执行哈希函数,得到 $x_i^p=h(t,i)$,计算 $x_i^u=x-x_i^p$,将 x_i^u 发送给 u_i,作为 u_i 的密钥.

- $UEnc(x_i^u,m)$: 用户加密算法.从 Z_q 中任取 r 来加密数据,使用 u_i 的密钥 x_i^u 对数据 m 进行加密 $c_i^*(m)=(g^r,g^{rx_i^u}m)$, u_i 将密文 $c_i^*(m)$ 发送给代理.

- $PEnc(i,t,h(\cdot,\cdot),c_i^*(m))$: 代理加密算法.代理接收到 u_i 加密后的密文 $c_i^*(m)$, 首先根据已有的 t 和 $h(\cdot,\cdot)$,计算 $x_i^p=h(t,i)$. 代理使用密钥 x_i^p 对 $c_i^*(m)$ 再次进行加密,执行 $(g^r)^{x_i^p}\cdot g^{rx_i^u}m=g^{rx}m$, 得到最终的密文 $c(m)=(g^r,g^{rx}m)$, 代理将 $c(m)$发送到服务器端存储.

- $PDec(j,t,h(\cdot,\cdot),c(m))$: 代理解密算法.假设 $c(m)$最终要发送给 u_j,代理首先计算 u_j 在代理端的密钥 x_j^p, 然后执行 $g^{rx}m\cdot(g^r)^{-x_j^p}=g^{rx_j^u}m$, 得到代理解密后的一级密文 $c_j'(m)=(g^r,g^{rx_j^u}m)$, 将 $c_j'(m)$ 发送给 u_j.

- $UDec(x_j^u,c_j'(m))$: 用户解密算法.u_j 使用自己的密钥 x_j^u 对 $c_j'(m)$ 进行解密,执行 $g^{rx_j^u}m\cdot(g^r)^{-x_j^u}=m$, u_j 最终获得数据明文 m.

2.2 安全性证明

以上代理加密是基于El Gamal加密方案设计的,其安全性是基于DDH困难问题的.在代理和用户不勾结的情况下,*PE* 方案在选择明文攻击下是不可区分的(IND-CPA).

假设存在一个 PPT 敌手 A,代理加密方案的 CPA 不可区分实验—— $S_{A,PE}^{cpa}$ 如下:

- 执行 $Init(1^\kappa)$, 获得公共参数(G,g,q),代理主密钥 t,散列函数 $h(\cdot,\cdot)$和主密钥 x.
- 执行 $KeyGen(x,i)$,获得 u_i 的密钥 x_i^u.
- 敌手 A 拥有代理主密钥 t 和 $h(\cdot,\cdot)$,并且可以访问用户加密预言机 $UEnc(x_i^u,\cdot)$, 敌手输出两个合法明文 m_0 和 m_1,发送给 u_i.
- 用户 u_i 随机选取 $b\leftarrow\{0,1\}$, 执行 $UEnc(x_i^u,m_b)$, 获得挑战密文 $c_i^*(m_b)=(g^r,g^{rx_i^u}m_b)$, 发送给 A.
- 敌手 A 继续访问用户加密预言机 $UEnc(x_i^u,\cdot)$, 输出 b'.

下面,考虑存在一个 PPT 敌手 A',尝试解决 DDH 难题:

- 参数 $G,g,q,g_1,g_2,g_3,t,h(\cdot,\cdot)$被发送给敌手 A',其中 $g_1=g^\alpha,g_2=g^\beta,g_3\in\{g^\gamma,g^{\alpha\beta}\}$, 其中,$\alpha,\beta,\gamma$是随机选取的,敌手 A'把 $G,g,q,t,h(\cdot,\cdot)$发送给敌手 A.敌手 A'计算 $g^{x_i^u}=g_1\cdot g^{-h(t,i)}$.
- 当敌手 A 访问用户加密预言机的时候,都把明文 m 发送给敌手 A',A'执行 $r\xleftarrow{R}Z_q$, 返回 $(g^r,g^{rx_i^u}m)$.
- 当敌手 A 输出两个明文 m_0 和 m_1 时,敌手 A'执行 $b\xleftarrow{R}\{0,1\}$, 把 $(g_2,g_2^{-h(t,i)}g_3m_b)$ 发送给敌手 A,作为其挑战密文.
- 敌手 A 会输出 b'.如果 $b=b'$,A'输出 1;否则,A'输出 0.

以下分两种情况进行讨论:

若 $g_3=g^\gamma$, 由于γ是随机选择的,所以 g_3 是群 G 中一个随机的元素. $g_2^{-h(t,i)}$ 中也不包含 m_b 的任意信息.所以敌手 A 并没有关于 m_b 的任何额外信息,所以有

$$\Pr[A'(G,g,q,t,h(\cdot,\cdot),g_1,g_2,g^\gamma)=1]=1/2 \tag{1}$$

若 $g_3=g^{\alpha\beta}$, 可以计算出 $g_2^{-h(t,i)}g_3m_b=g^{-\beta h(t,i)}g^{\alpha\beta}m_b=g^{(-h(t,i))}m_b=g^{\beta x_i^u}m_b$, $(g_2,g_2^{-h(t,i)}g_3m_b)$ 可以化简成 $(g^\beta,g^{\beta x_i^u}m_b)$, 是 *PE* 方案的合法密文,所以有

$$\Pr[A'(G,g,q,t,h(\cdot,\cdot),g_1,g_2,g^{\alpha\beta})=1]=S_{A,PE}^{cpa} \tag{2}$$

由于 DDH 问题是困难的,所以有

$$\Pr[A'(G,g,q,t,h(\cdot,\cdot),g_1,g_2,g^{\alpha\beta})=1]-\Pr[A'(G,g,q,t,h(\cdot,\cdot),g_1,g_2,g^\gamma)=1]<negl \tag{3}$$

根据式(1)~式(3)可以推出 $S_{A,PE}^{cpa}<negl+1/2$. 因此,代理加密方案 *PE* 在选择明文攻击下是不可区分的.

3 基于二叉树存储的多用户 ORAM

BT-M-ORAM 方案包括 3 个主体:用户、代理和云服务器.假设存在一个可信的密钥分发中心负责各种初始化工作.可信的密钥分发中心会给每个加入系统的新用户分配一个密钥,给代理分发代理主密钥,用以计算每个用户对应的代理密钥.

服务器端采用二叉树的形式存储密文[14],每一个节点都是一个独立的 ORAM,节点 ORAM 可以是任何形式的 ORAM,比如 Goldreich 和 Ostrovsky[7]提出的平方根 ORAM、分层 ORAM,或者 Damgard[21]提出的基于二叉搜索树的 ORAM,更甚至说仅仅是一个平凡的 ORAM.考虑到现实背景以及本文方案描述的简洁、清晰,本文详细介绍节点为平凡 ORAM 的 BT-M-ORAM 方案.

当用户要上传数据到不可信的服务器时,首先用户用自己的密钥加密数据,将加密后的一级密文发送给代理,代理在不解密的情况下,再次加密,从而获到最终的密文,将其发送给服务器存储.

在单用户的基于树的 ORAM 方案中,当用户需要查询一个数据项时,首先需要检查本地索引表,在多用户场景下,数据使用者检索的数据可能来自于多个数据拥有者,这时,如果依然由各个数据拥有者维护自己的索引表,必然带来应用的不便,因此我们建立了一张联合索引表,记录所有用户的数据索引,并将此索引表存储于代理端.系统模型如图 1 所示.

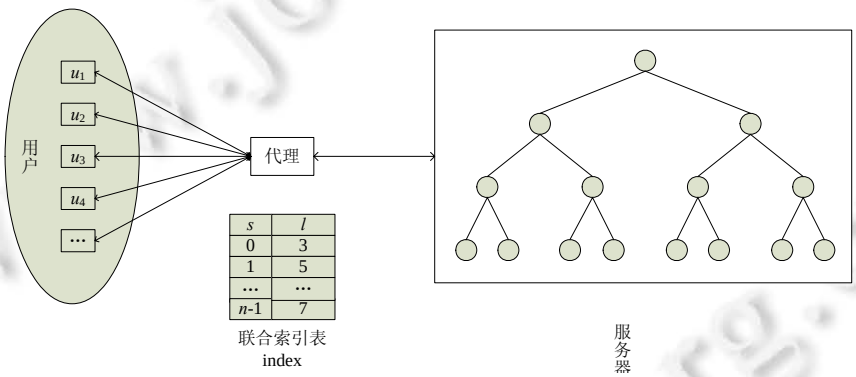

Fig.1 System model

图 1 系统模型

3.1 平凡的节点ORAM

本文 BT-M-ORAM 方案中,二叉树节点采用平凡的 ORAM 方案,存储能力为 $O(\log n)$.由于在用户与服务器端之间引入一个代理,使用第 2.1 节中描述的代理加密方案完成数据加密存储,所以,节点 ORAM 的操作涉及到用户和代理两方的加解密过程.平凡 ORAM 提供两种 ORAM 操作:

• *bucket*.*ReadAndRemove*(*s*):用户在该节点搜索 *s* 数据项,顺序读取服务器端这个节点的每一个数据项,发送给代理,代理解密后发送给用户,用户再次解密获得明文.当读到 id 为 *s* 的数据项时,用户记下这个数据内容,并用 ⊥ 代替 *s* 数据项;如果读到的数据项不是 *s*,则保留原数据项.用户重新加密数据项发送给代理,代理再次加密后写回该块.具体操作见算法 1.

• *bucket*.*Add*(*s*,*d*):逐个读取服务器端该节点的数据项,发送给代理,代理解密后发送给用户,用户再次解密获得明文.当第 1 次读到 ⊥ 时(数据 id 为 0),将要插入的数据项加密后发送给代理,代理再次加密后写入该块,其余情况,只需要用户和代理共同加密原内容写回该块即可.具体操作见算法 2.

算法 1. *bucket*.*ReadAndRemove*(*s*).

输入:要搜索的数据项的数据 id 为 *s*;

输出:*s* 数据项对应的数据内容.

01: *data*←⊥

02: for (every block of this node)
03: $proxy \leftarrow (g^{r}, g^{rx}s', g^{rx}d')$
04: $c_i'(s') \leftarrow (g^{r}, g^{rx_i^u}s') \leftarrow PE.PDec(i,t,h(\cdot,\cdot),(g^{r}, g^{rx}s'))$ //请求是 u_i 发出的
05: $c_i'(d') \leftarrow (g^{r}, g^{rx_i^u}d') \leftarrow PE.PDec(i,t,h(\cdot,\cdot),(g^{r}, g^{rx}d'))$
06: $u_i \leftarrow (g^{r}, g^{rx_i^u}s', g^{rx_i^u}d')$
07: $(s',d') \leftarrow (PE.UDec(x_i^u,(g^{r}, g^{rx_i^u}s')), PE.UDec(x_i^u,(g^{r}, g^{rx_i^u}d')))$
08: if (s'=s)
09: $data \leftarrow (s',d')$
10: $(s_t,d_t) \leftarrow \perp$
11: else
12: $(s_t,d_t) \leftarrow (s',d')$
13: end if
14: $c_i^*(s_t) \leftarrow (g^{r}, g^{rx_i^u}s_t) \leftarrow PE.UEnc(x_i^u, s_t)$
15: $c_i^*(d_t) \leftarrow (g^{r}, g^{rx_i^u}d_t) \leftarrow PE.UEnc(x_i^u, d_t)$ //两次加密取同一个 r
16: $proxy \leftarrow (g^{r}, g^{rx_i^u}s_t, g^{rx_i^u}d_t)$
17: $(g^{r}, g^{rx}s_t) \leftarrow PE.PEnc(i,t,h(\cdot,\cdot),(g^{r}, g^{rx_i^u}s_t))$
18: $(g^{r}, g^{rx}d_t) \leftarrow PE.PEnc(i,t,h(\cdot,\cdot),(g^{r}, g^{rx_i^u}d_t))$
19: write (g^{r},$g^{rx}s_t$,$g^{rx}d_t$) back to this block
20: end for
21: return data

算法 2. *bucket*.*Add*(*s*,*d*).

输入:待插入的新数据(*s*,*d*);

输出:将新数据写入服务器,返回数据 *d*.

01: for (every block)
02: $proxy \leftarrow (g^{r}, g^{rx}s', g^{rx}d')$
03: $c_i'(s') \leftarrow (g^{r}, g^{rx_i^u}s') \leftarrow PE.PDec(i,t,h(\cdot,\cdot),(g^{r}, g^{rx}s'))$ //请求是 u_i 发出的
04: $c_i'(d') \leftarrow (g^{r}, g^{rx_i^u}d') \leftarrow PE.PDec(i,t,h(\cdot,\cdot),(g^{r}, g^{rx}d'))$
05: $u_i \leftarrow (g^{r}, g^{rx_i^u}s', g^{rx_i^u}d')$
06: $(s',d') \leftarrow (PE.UDec(x_i^u,(g^{r}, g^{rx_i^u}s')), PE.UDec(x_i^u,(g^{r}, g^{rx_i^u}d')))$
07: if (the first time s'=0) //即第 1 次读到⊥块
08: $(s_t,d_t) \leftarrow (s,d)$
09: else
10: $(s_t,d_t) \leftarrow (s',d')$
11: end if
12: $c_i^*(s_t) \leftarrow (g^{r}, g^{rx_i^u}s_t) \leftarrow PE.UEnc(x_i^u, s_t)$
13: $c_i^*(d_t) \leftarrow (g^{r}, g^{rx_i^u}d_t) \leftarrow PE.UEnc(x_i^u, d_t)$ //两次加密取同一个 r
14: $proxy \leftarrow (g^{r}, g^{rx_i^u}s_t, g^{rx_i^u}d_t)$
15: $(g^{r}, g^{rx}s_t) \leftarrow PE.PEnc(i,t,h(\cdot,\cdot),(g^{r}, g^{rx_i^u}s_t))$

16: $(g^r, g^{rx} d_t) \leftarrow PE.PEnc(i,t,h(\cdot,\cdot),(g^r, g^{rx_t^u} d_t))$

17: write $(g^r, g^{rx} s_t, g^{rx} d_t)$ back to this block

18: end for

19: return d

3.2 基于二叉树的多用户ORAM方案

- 系统初始化

可信的密钥分发中心首先执行 $PE.Init(1^\kappa)$,然后选择一个伪随机发生器 F,F 可以输出 D 位的伪随机串.当一个新用户 u_i 加入系统时,这个可信的密钥分发中心会执行 $PE.KeyGen(x,i)$,给 u_i 发送一个密钥 x_i^u.

- 数据发布

每个用户会发布自己的数据到服务器端,以便其他用户查找.比如,用户 u_i 要发布新的数据(s,d)到服务器端.数据发布时,都是写入根节点.首先,代理执行伪随机函数 F,得到 l,把 $P(l)$作为新发布数据的初始存储路径,并将(s,l)作为一个索引表项存储在联合索引表 *index* 中.然后,对根节点执行 $bucket.ReadAndRemove(s)$,返回给 u_i 的数据一定为⊥,因为 s 是唯一的,这是第 1 次写入 id 为 s 的数据.最后,对根节点执行 $bucket.Add(s,d)$,从而实现将数据(s,d)以密文的形式存储在服务器端.

- 数据搜索

用户 u_j 要搜索一个 id 为 s 的数据项(注:s 数据项可以是系统内任意一个用户发布的),具体过程如图 2 所示.

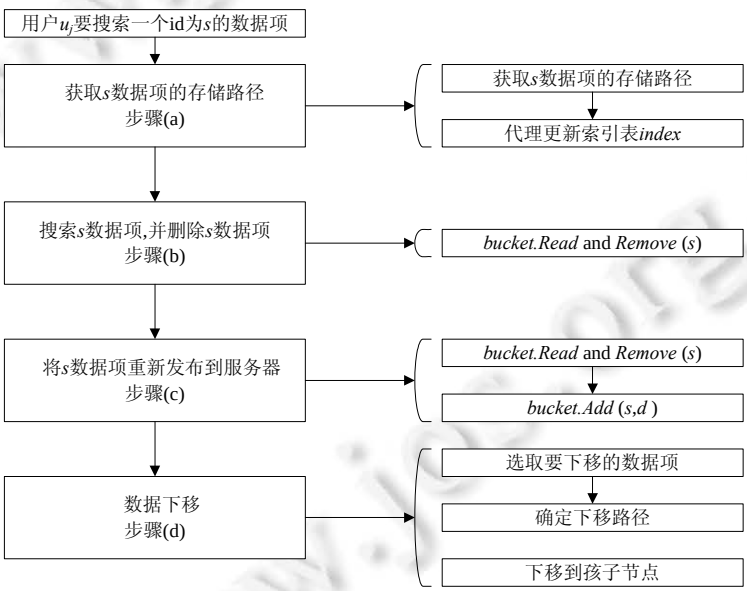

Fig.2 Data search process

图 2 数据搜索过程

(a) 获取 s 数据项的存储路径

- 获取 s 数据项的存储路径:用户给代理发送数据 id s,代理查询本地索引表 *index*,获得 s 数据项对应的叶子节点 l,从而获得 s 数据项的存储路径 $P(l)$.

- 代理更新索引表 *index*:代理执行伪随机函数 F,获得 l^*(s 数据项的新存储路径为 $P(l^*)$),将代理本地索引表 *index* 中 s 数据项对应的索引表项更改为(s,l^*).

(b) 搜索 s 数据项,并删除 s 数据项:对于路径 $P(l)$上的每一个节点,执行 $bucket.ReadAndRemove(s)$操作.如果存在 id 为 s 的数据项,那么 u_j 一定在操作执行过程中获得了 s 数据项的内容,并且将 s 数据项从服务器端删除,

算法返回的数据就是 s 数据项的内容.如果不存在 id 为 s 的数据项,算法返回的数据是$\perp$.

(c) 将 s 数据项重新发布到服务器:返回的数据项(s 数据项或者$\perp$)的存储路径已经在(a)中更新为 $P(l^*)$,对根节点执行 *bucket.ReadAndRemove*(s)和 *bucket.Add*(s,d)将数据项写入根节点.

(d) 数据下移:

对 0~D−1 层的每一层 n,都执行以下操作:

从第 n 层中任意选取两个节点(取 w=2),对于每个节点:

- 选取要下移的数据项:服务器将该节点中的所有数据项依次发送给代理,执行 *PE.PDec*(·,·,·,·),获得一级密文发送给用户,用户执行 *PE.UDec*(·,·),获得数据明文.如果该节点都是$\perp$数据项,则选取$\perp$数据项,记为 $B^{\#}=\perp$;如果该节点存在真实的数据项,任意选取一个真实数据项,记为 $B^{\#}$,并从该节点中删除该数据项.用户执行 *PE.UEnc*(·,·) 加密数据项 $B^{\#}$,获得一级密文 $c^*(B^{\#})$,把($s^{\#}$, $c^*(B^{\#})$)发送给代理.
- 确定下移路径:代理根据 $s^{\#}$查询本地索引表 *index*,获得 $s^{\#}$ 数据项对应的叶子节点 $l^{\#}$,从而获取数据的存储路径 $P(l^{\#})$;若没有查询到对应的索引表项,说明 $B^{\#}$是$\perp$,此时随机选取一个 D 位的 01 串,作为 $l^{\#}$.代理执行 *PE.PEnc*(·,·,·,·) 再次加密 $c^*(B^{\#})$,获得最终密文 $c(B^{\#})$.
- 下移到孩子节点:若 $l^{\#}$的第 n+1 位为 0,把 $c(B^{\#})$写入左侧孩子节点,把$\perp$写入右侧孩子节点;若 $l^{\#}$的第 n+1 位为 1,把$\perp$写入该节点的左侧孩子节点,把 $c(B^{\#})$写入该节点的右侧孩子节点.

4 方案分析

4.1 安全性分析

BT-M-ORAM 方案的目的是在实现多用户的前提下,隐藏访问模式.第 3.2 节中描述的 BT-M-ORAM 方案记为Π,将 Π 中的伪随机函数 F 改为真随机函数 f(输出一个 D 位的真随机 01 串)变成新的方案Π′.存在一个 PPT 敌手 A 攻击方案Π,敌手 A 腐蚀一个用户 u_a,挑战者由未被腐蚀的用户 u_c 和代理充当.A 算法如下:

- 初始化:可信密钥分发中心首先执行 $PE.Init(1^{\kappa})$, 然后选择一个伪随机发生器 F,输出 D 位的伪随机串,最后,可信密钥分发中心执行 *PE.KeyGen*(x,a) 和 *PE.KeyGen*(x,c), u_a 和 u_c 分别获得密钥 x_a^u 和 x_c^u.
- 问询:敌手 A 腐蚀用户 u_a,可以让 u_a 搜索任意数据项;让 u_c 搜索任意一个没被搜索的数据项及访问一个已经被搜索的特定的数据项.
- 请求:敌手 A 让 u_c 随机搜索一个数据项,记为 s_0;敌手 A 让 u_c 再次随机搜索一个数据项,记为 s_1.
- 问询:敌手 A 可以让 u_a 搜索任意数据项;让 u_c 搜索任意一个没被搜索的数据项及搜索 s_0 或 s_1 数据项.
- 挑战:挑战者随机选取 $b \leftarrow \{0,1\}$, u_c 搜索 s_b 数据项.
- 问询:敌手 A 可以让 u_a 搜索任意数据项;让 u_c 搜索任意一个没被搜索的数据项及搜索 s_b 或 s_{1-b} 数据项.
- 应答:敌手 A 输出 b'.

敌手 A 攻击方案Π′时,由于节点采用平凡的 ORAM 方案,因此节点 ORAM 是安全的.当任意一个用户发起一个搜索请求时,代理会通过联合索引表 *index* 查询数据的存储路径 $P(l)$——步骤(a).从步骤(a)可以看出,任意一个 l 都是由真随机函数 f 选取的一个随机的 D 位 01 串,因此步骤(b)访问的节点可以看做是随机访问一条路径上的所有节点.步骤(c)中,对根节点进行写操作,由于节点 ORAM 是安全的,所以步骤(c)也是安全的.步骤(d)中,下移数据时,左右孩子节点都会被重新加密的数据写入,所以也不会泄露任何信息.用户之间是完全独立的,没有任何密钥共享或者交流.整个过程中,A 的视图是一样的,并没有 s_0 和 s_1 数据项任何额外的信息,所以 $S_{A,\Pi'}=1/2$.

下面,假设存在一个 PPT 敌手 A',试图区分伪随机函数 F 与真随意函数 f.A'可访问预言机Θ(·):

- 问询:当 A 让 u_c 随机搜索一个数据项时,A'会访问Θ(·),产生新的叶子节点 l^*,并将路径 $P(l)$(l 是以前 A'访问Θ(·)产生的)上的数据返回给 A.
- 挑战:当 A 已经搜索 s_0 和 s_1 后,A'任意选取 $b \leftarrow \{0,1\}$, 访问Θ(·),获得新叶子节点 l^*,让 u_c 搜索 s_b 数据项.

- 问询:敌手 A 可以让 u_a 搜索任意数据项,通过 A'访问$\Theta(\cdot)$,选取新叶子节点 l^*.
- 应答:A 输出 b',如果 $b=b'$,A'输出 1;否则,A'输出 0.

以下分两种情况进行讨论:

如果$\Theta(\cdot)$为伪随机函数 F,A 作为 A'的子程序运行时的视图与 $S_{A,\Pi}$ 实验中 A 的视图是一样的,所以,

$$\Pr[A'(F)=1]=S_{A,\Pi} \tag{4}$$

如果$\Theta(\cdot)$为真随机函数 f,A 作为 A'的子程序运行时的视图与 $S_{A,\Pi'}$ 实验中 A 的视图是一样的,所以,

$$\Pr[A'(f)=1]=S_{A,\Pi'}=1/2 \tag{5}$$

由于无法区分伪随机函数 F 与真随机函数 f,所以有,

$$\Pr[A'(F)=1]-\Pr[A'(f)=1]<negl \tag{6}$$

由式(4)~式(6)可以推出: $S_{A,\Pi}<negl+1/2$. 所以,BT-M-ORAM 是安全的,其安全性是基于伪随机函数的不可区分性.

4.2 复杂性分析

服务器端以二叉树的形式存储数据,二叉树中每个节点又是一个独立的 ORAM,方案中节点采用平凡的 ORAM.我们首先分析平凡 ORAM 的计算复杂度.假设一个平凡 ORAM 的能力为 L,服务器端的空间复杂度为 $O(L)$,用户的空间复杂度为 $O(1)$,由于平凡ORAM的操作是将每一个数据项都读出并重新加密写回,所以平均和最差计算复杂度都是 $O(L)$.

BT-M-ORAM 中,由于采用了联合索引表的方式记录叶子节点,所以代理的空间复杂度为 $O(n)$.用户不再需要将索引存储在本地,而是存储在代理端,用户的空间复杂度为 $O(1)$.服务器上用来存储数据的二叉树,共 $O(n)$ 个节点,节点采用平凡的 ORAM,能力为 $O(\log n)$,所以服务器端的空间复杂度为 $O(n\log n)$.当用户发起一个搜索请求时,服务器会返回路径 $P(l)$上的每一个节点的数据,数据 ORAM 的树高 $D=\lceil\log n\rceil$,节点 ORAM 的能力为 $O(\log n)$,该过程需要处理的数据项数目为 $O(\log^2 n)$.为了有效防止溢出,会执行一个下移操作,有 wD(w 为常数)个节点参与,每个节点的能力为 $O(\log n)$,所以,下移操作需要处理的数据项数目为 $O(\log^2 n)$.由上可知,每次操作总计算复杂度为 $O(\log^2 n)$——平均和最差情况下的计算复杂度都为 $O(\log^2 n)$.

Zhang 等人[19]提出的多用户 ORAM 方案中,混淆阶段匿名器的计算复杂度最差情况下为$\Omega(n)$,平均计算复杂度为 $O(\log^3 n\log\log n)$,而 BT-M-ORAM 方案中,用户和代理在最差及平均情况下,总的计算复杂度均为 $O(\log^2 n)$,比 Zhang 等人[19]的方案的计算复杂度小得多,复杂度比较见表 1,由此可见,该方案的提出有很重要的意义.

Table 1 Complexity comparison

表 1 复杂度比较

	Zhang 等人[19]的方案	BT-M-ORAM 方案
平均计算复杂度	$O(\log^3 n\log\log n)$	$O(\log^2 n)$
最差计算复杂度	$\Omega(n)$	$O(\log^2 n)$
用户端空间复杂度	$O(1)$	$O(1)$
服务器端空间复杂度	$O(n\log\log n)$	$O(n\log n)$

5 总结与展望

基于二叉树的 ORAM 方案相比较分层 ORAM 方案来说,避免了混淆过程,极大地降低了计算复杂度,借鉴这种思想,改进 Zhang 等人[19]的方案,提出 BT-M-ORAM 方案,服务器端采用了二叉树的存储方式,利用改进的代理加密机制实现了多用户共同参与发布、搜索数据,并有效防止访问模式的泄露.相比较已有的多用户 ORAM,计算复杂度大为降低.

加密数据查询是ORAM的一个很好的应用场景,由于之前ORAM带来的运算量巨大,加密数据查询基本不隐藏访问模式[22−24],但是随着 ORAM 的发展,计算复杂度逐渐降低,隐藏访问模式的加密数据查询可行性变大.如果将 BT-M-ORAM 中的数据结构做一定的修改,可以实现隐藏访问模式的加密数据查询.ORAM 还可以运用

到安全计算以及防止内存攻击等场景中,如此广泛的应用极大地促进了 ORAM 的研究与发展.

References:

[1] Chen K, Zheng WM. Cloud computing: System instances and current research. Ruan Jian Xue Bao/Journal of Software, 2009,20(5): 1337−1348 (in Chinese with English abstract). [doi: 10.3724/SP.J.1001.2009.03493]

[2] Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. A view of cloud computing. Communications of the ACM, 2010,53(4):50−58. [doi: 10.1145/1721654.1721672]

[3] Lou JZ, Jin JH, Song AB, Dong F. Cloud computing: Architecture and key technologies. Journal on Communications, 2011,32(7): 3−21 (in Chinese with English abstract).

[4] Feng DG, Zhang M, Zhang Y, Xu Z. Study on cloud computing security. Ruan Jian Xue Bao/Journal of Software, 2011,22(1): 71−83 (in Chinese with English abstract). [doi: 10.3724/SP.J.1001.2011.03958]

[5] Kaufman LM. Data security in the world of cloud computing. Security & Privacy, 2009,7(4):61−64. [doi: 10.1109/MSP.2009.87]

[6] Su L. Research and implementation of data integrity verification method for cloud computing [MS. Thesis]. Chengdu: University of Electronic Science and Technology of China, 2013 (in Chinese with English abstract).

[7] Goldreich O, Ostrovsky R. Software protection and simulation on oblivious RAMs. Journal of the ACM, 1996,43(3): 431−473. [doi: 10.1145/233551.233553]

[8] Pinkas B, Reinman T. Oblivious RAM revisited. In: Rabin T, ed. Advances in Cryptology-CRYPTO 2010. Berlin, Heidelberg: Springer-Verlag, 2010. 502−519. [doi: 10.1007/978-3-642-14623-7_27]

[9] Goodrich MT, Mitzenmacher M. Privacy-Preserving access of outsourced data via oblivious RAM simulation. In: Aceto L, Henzinger M, Sgall J, eds. Automata, Languages and Programming. Berlin, Heidelberg: Springer-Verlag, 2011. 576−587. [doi: 10.1007/978-3-642-22012-8_46]

[10] Damgård I, Meldgaard S, Nielsen JB. Perfectly secure oblivious RAM without random oracles. In: Ishai Y, ed. Theory of Cryptography. Berlin, Heidelberg: Springer-Verlag, 2011. 144−163. [doi: 10.1007/978-3-642-19571-6_10]

[11] Boneh D, Mazieres D, Popa RA. Remote oblivious storage: Making oblivious RAM practical. Technical Report, 018, 2011.

[12] Goodrich MT, Mitzenmacher M, Ohrimenko O, Tamassia R. Privacy-Preserving group data access via stateless oblivious RAM simulation. In: Rabani Y, ed. Proc. of the 23rd Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: SIAM, 2012. 157−167.

[13] Kushilevitz E, Lu S, Ostrovsky R. On the (in) security of hash-based oblivious RAM and a new balancing scheme. In: Rabani Y, ed. Proc. of the 23rd Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: SIAM, 2012. 143−156.

[14] Shi E, Chan THH, Stefanov E, Li M. Oblivious RAM with O $((\log N)^3)$ worst-case cost. In: Lee DH, Wang XY, eds. Advances in Cryptology–ASIACRYPT 2011. Berlin, Heidelberg: Springer-Verlag, 2011. 197-214. [doi: 10.1007/978-3-642-25385-0_11]

[15] Stefanov E, Van Dijk M, Shi E, Fletcher C, Ren L, Yu XY, Devadas S. Path ORAM: An extremely simple oblivious ram protocol. In: Sadeghi A, ed. Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. New York: ACM, 2013. 299−310. [doi: 10.1145/2508859.2516660]

[16] Ren L, Fletcher CW, Yu X, van Dijk M, Devadas S. Integrity verification for path Oblivious-RAM. In: Leeser M, ed. Proc. of the 2013 IEEE High Performance Extreme Computing Conf. Piscataway: IEEE, 2013. 1−6. [doi: 10.1109/HPEC.2013.6670339]

[17] Chung KM, Liu Z, Pass R. Statistically-Secure ORAM with $O(\log^2 n)$ overhead. In: Sarkar P, Iwata T, eds. Advances in Cryptology–ASIACRYPT 2014. Berlin, Heidelberg: Springer-Verlag, 2014. 62−81.

[18] Williams P, Sion R, Tomescu A. Privatefs: A parallel oblivious file system. In: Yu T, ed. Proc. of the 2012 ACM Conf. on Computer and Communications Security. New York: ACM, 2012. 977−988. [doi: 10.1145/2382196.2382299]

[19] Zhang JS, Zhang WS, Qiao DJ. A multi-user oblivious RAM for outsourced data. Computer Science Technical Report, 2014, 262.

[20] Dong C, Russello G, Dulay N. Shared and searchable encrypted data for untrusted servers. Journal of Computer Security, 2011, 19(3):367−397.

[21] Damgård I, Meldgaard S, Nielsen JB. Perfectly secure oblivious RAM without random oracles. In: Ishai Y, ed. Theory of Cryptography. Berlin, Heidelberg: Springer-Verlag, 2011. 144−163. [doi: 10.1007/978-3-642-19571-6_10]

[22] Cheng FQ, Peng ZY, Song W, Wang SL, Cui YH. An efficient privacy-preserving rank query over encrypted data in cloud computing. Chinese Journal of Computers, 2012,35(11):2215−2227 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2012.02215]

[23] Feng GL, Tan L. Multi-Attribute ranked keyword search over encrypted cloud data. Computer Science, 2013,40(11):131−136,157 (in Chinese with English abstract). [doi: 10.1109/INFOCOM.2014.6848153]

[24] Wang B, Yu S, Lou W, Hou YT. Privacy-Preserving multi-keyword fuzzy search over encrypted data in the cloud. In: Leon-Garcia A, ed. Proc. of the 2014 IEEE on INFOCOM. Piscataway: IEEE, 2014. 2112−2120.

附中文参考文献:

[1] 陈康,郑纬民.云计算:系统实例与研究现状.软件学报,2009,20(5):1337−1348. http://www.jos.org.cn/1000-9825/3493.htm [doi: 10.3724/SP.J.1001.2009.03493]

[3] 罗军舟,金嘉晖,宋爱波,东方.云计算:体系架构与关键技术.通信学报,2011,32(7):3−21.

[4] 冯登国,张敏,张妍,徐震.云计算安全研究.软件学报,2011,22(1):71−83. http://www.jos.org.cn/1000-9825/3958.htm [doi: 10.3724/SP.J.1001.2001.03958]

[6] 苏兰.面向云计算的数据完整性检验方法研究与实现[硕士学位论文].成都:电子科技大学,2013.

[22] 程芳权,彭智勇,宋伟,王书林,崔一辉.云环境下一种隐私保护的高效密文排序查询方法.计算机学报,2012,35(11):2215−2227.

[23] 冯贵兰,谭良.云环境中基于多属性排序的密文检索方案.计算机科学,2013,40(11):131−136,157.

孙晓妮(1990－),女,山东烟台人,硕士生,主要研究领域为密码学,信息安全.

蒋瀚(1974－),男,博士,讲师,主要研究领域为密码学与信息安全,公钥密码学,安全多方计算.

徐秋亮(1960－),男,博士,教授,博士生导师,主要研究领域为公钥密码算法设计与分析,密码协议设计与分析,多方安全计算协议研究,网络信息安全核心技术研究与开发.