doi:10.11959/j.issn.1000-436x.2016010

云存储下多用户协同访问控制方案

史姣丽[1,2]，黄传河[1]，王晶[1]，覃匡宇[1,3]，何凯[1]

(1. 武汉大学计算机学院，湖北 武汉 430072；2. 九江学院信息科学与技术学院，江西 九江 332005；
3. 桂林电子科技大学信息与通信学院，广西 桂林 541004)

摘 要：CP-ABE 被认为是云存储下最适合的数据访问控制方法之一，但它仅适合用户分别读取或者分别修改不同数据的情况，而直接应用 CP-ABE 进行多用户协同数据访问时，会存在修改无序、密文文件大量冗余等问题。多用户协同访问云端数据时，应该在保证机密性、抗共谋的前提下控制合法用户有序地修改同一密文文件，同时云端尽可能减少密文文件副本。针对文件和文件逻辑分块，提出了 2 个多用户协同访问控制方案 MCA-F 和 MCA-B。MCA-F 满足单个数据文件作为最小控制粒度的访问控制需求，该方案采用层次加密结构，云服务器承担部分解密计算，以降低用户解密的计算代价；针对多用户同时写数据的访问控制，提出了对多个用户提交的暂存数据的管理方法。MCA-B 用于文件的逻辑分块作为最小控制粒度的访问控制，该方案设计了文件的逻辑分块机制、基于索引矩阵的表示方法，提出了子数据掩码表示方法以描述多个用户对同一文件不同逻辑分块的写权限；MCA-B 支持用户集合、文件逻辑分块结构的动态变化，而且数据的拥有者和修改者无需一直在线。与现有的方案相比，所提方案不仅具有云存储下多用户协同写数据的访问控制能力，而且读访问控制的用户端存储量和加解密计算量是较小的。

关键词：云存储；访问控制；属性加密；多用户协同访问

中图分类号：TP393.0 **文献标识码：**A

Multi-user collaborative access control scheme in cloud storage

SHI Jiao-li[1,2], HUANG Chuan-he[1], WANG Jing[1], QIN Kuang-yu[1,3], HE Kai[1]

(1. Computer School, Wuhan University, Wuhan 430072, China;
2. School of Information Science and Technology, Jiujiang University, Jiujiang 332005, China;
3. School of Information and Communication, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: CP-ABE was considered as one of most suitable methods of access control in cloud storage. However, it was just fit for reading or modifying different data files respectively. When CP-ABE was applied directly to data access collaborative control by multiple users, there would be such problems as data being modified disorderly. When multiple users access collaboratively the data stored on the cloud, legitimate users should modify the same ciphertext file orderly on the premise of confidentiality and collusion-resistance and the copies of ciphertext file should be generated as few as possible. Two multi-user collaborative access control schemes MCA-F and MCA-B for the file and its logical blocks each were proposed. The MCA-F scheme meets the requirement of access control in which the minimal granularity of control is a single data file. In MCA-F scheme, hierarchical encryption is adopted, a part of decrypting computation is transferred to a cloud server to decrease the computational cost on users when decrypting. In allusion to the simultaneous write-data access control of multiple users, a method is designed to manage semi-stored modified data submitted by menders. The MCA-B scheme is used for the access control in which a logical block of the file is the minimal granularity of control. This scheme designs a mechanism of logical blocking of the file and a representing method based on index matrix, and the representation of sub data mask is put forward to describe write permission of multiple users on different logical blocks of the same file. MCA-B scheme supports the dynamic change of the structure of logical blocks of the file, and the owners or menders do not need to be online always. Compared with the existing schemes, not only do proposed schemes provide multi-user collaborative access control in cloud storage, but also the client storage of reading access control and the computation of encrypting and decrypting are both lesser.

Key words: cloud storage, access control, attribute-based encryption, multi-user collaborative access

收稿日期：2014-10-21；**修回日期：**2014-12-19

通信作者：黄传河，huangch@whu.edu.cn

基金项目：国家自然科学基金资助项目(No.61373040, No.61572370)；**教育部博士点基金资助项目**(No.20120141110073)

Foundation Items: The National Natural Science Foundation of China (No.61373040, No.61572370), The Ph.D. Programs Foundation of Ministry of Education of China (No.20120141110073)

1　引言

云存储为用户提供了随时随地的数据访问。为了细粒度地控制授权用户访问所授权的那部分数据，可以采用 Waters 等提出的 ABE (attribute-based encryption[1] 、 KP-ABE(key-policy attribute-based encryption)[2] 、 CP-ABE(ciphertext-policy attribute-based encryption[3]、FE(functional encryption)[4]等一系列加密方法，构建高效、安全的访问控制方案。其中，CP-ABE 被认为是云存储下最适合的数据访问控制方法之一。

CP-ABE 中，可信授权机构为合法用户分发用户属性私钥。数据拥有者定义访问策略，用访问策略对敏感数据进行加密，将加密后的密文上传到云端。用户访问数据时，从云端下载密文，用自己的属性私钥解密密文，如果用户的属性私钥与密文的访问策略匹配，用户可以解密成功得到明文，否则，用户解密失败。访问策略可以用树、LSSS 矩阵、单调布尔公式等表示，其中，树是最常用的访问策略表示方法。

目前，大部分 CP-ABE 访问控制方法都是用于控制单个用户对云数据的访问，属性集满足访问控制树的用户可以访问数据。但是，很少有论文研究云存储下多用户同时对一个文件进行写操作的访问控制方法。CP-ABE 仅适合用户分别读取或者分别修改不同数据，而直接应用 CP-ABE 进行多用户协同访问数据时，会存在修改无序、密文文件大量冗余等问题。本文拟对 CP-ABE 方法进行完善，以适应云端协同访问控制的应用需求。

分析实际应用，发现数据拥有者 Owner 对数据具有创建、读、写（修改）、删除等权限，而写授权用户（即，数据修改者 Mender）对数据只具有读、写（修改）的权限。由于数据的可读用户集合U_r和可写用户集合U_w之间存在关系：$U_w \subseteq U_r$（即一个用户对数据具有读权限时不一定具有写权限），所以，采用 CP-ABE 进行写权限控制时，每个数据在云端存储时需要携带 2 个访问策略（$policy_r$和$policy_w$），$policy_r$决定了具有读权限的用户集合U_r。$policy_w$决定了具有写权限的用户集合U_w。如果用户u_t的属性集合I_{u_t}与数据 *Data* 的$policy_r$匹配，则该用户读取 *Data* 成功。如果用户u_t的属性集合I_{u_t}与数据 *Data* 的$policy_w$匹配，则该用户对 *Data* 的写操作成功。

对数据而言，每次用户u_t写数据 *Data* 时，有 2 种情况：1）向云端提交的数据 *Data* 是一个新文件；2）向云端提交的数据 *Data* 是一个旧文件。本文研究第 2 种情况，即所提交的数据 *Data* 是对一个旧文件$Data_{\text{file}}$的修改。提交 *Data* 的用户u_t可以是数据拥有者 Owner，也可以是数据的修改者 Mender。如果是 Owner 提交的数据，则 Cloud Server 直接接收 *Data*，并覆盖到$Data_{\text{file}}$上。但如果提交 *Data* 的用户u_t是$Data_{\text{file}}$的修改者，那么数据拥有者 Owner 对 Mender 提交的修改有 2 种处理。1) 听任其修改，直接由 Cloud Server 负责覆盖到$Data_{\text{file}}$。挑战是如何保证 Owner 离线的情况下，可写用户集合U_w中的多个用户能够有序无冲突地将修改数据提交给 Cloud Server，由 Cloud Server 覆盖到$Data_{\text{file}}$。2）经过 Owner 审核通过后才能覆盖到$Data_{\text{file}}$。本文考虑第 2 种处理。分析实际应用，数据应该维持的状态有 3 种：一是记录最新修改后的数据(write-modify)；二是要记录数据的修改日志(write-append)；三是每个 Mender 只看到自己的修改结果(write-copy)。为研究方便，本文只考虑第一种情况，即云 Cloud 只记录 Owner 审核过的最新修改后的数据。现在的挑战是如何高效地让 Owner 审核多个U_w用户上传到 Cloud 的数据修改部分，安全有序地覆盖$Data_{\text{file}}$。

本文主要研究在保证数据机密性、防止共谋攻击的前提下，基于 CP-ABE 方法，在半可信云存储环境中尝试设计多用户协作访问控制方案。

1.1　本文的主要贡献

本文设计了细粒度的灵活访问控制，既考虑了数据文件的创建写，又考虑了数据文件的追加写。Li[6]方案只考虑 PHR 档案数据的创建写，没有考虑 PHR 档案数据的追加写；本文方案适合多授权机构的应用场景。Li[6]方案只考虑一个授权机构的应用场景；本文方案可以控制数据文件或者文件的逻辑分块。Li[6]方案的控制粒度是单个数据文件；本文方案适用于通用云存储系统。

具体地，本文的主要贡献总结如下。

1) 针对单个数据文件作为最小控制粒度的情形，提出了一个多用户协同访问控制方案 MCA-F，用户持有属性私钥，文件用策略树加密后存储于云端。如果用户属性集满足读访问控制树，用户可以读取文件内容。如果用户属性集满足写权限，云端

暂存用户的写数据，由Owner进行写数据的内容审核。采用层次加密结构，云服务器承担部分解密计算，以降低用户解密的计算代价；针对多用户同时写数据的访问控制，提出了对多个用户提交的暂存数据的管理方法。

2) 针对文件的逻辑分块作为最小控制粒度的情形，设计了多用户协同修改文件不同逻辑分块的控制方案（MCA-B）：将文件分割成逻辑分块。子数据掩码SDM（sub data mask）用来控制每个用户对文件不同逻辑分块的写权限。数据修改索引矩阵DM（index matrix of modification）用来记录每个Mender暂存云端的文件逻辑分块。Owner无需总是在线。支持用户集合、文件逻辑分块结构的动态变化。多个用户可以修改同一个数据文件的每一个逻辑分块。

3) 设计了一种云辅助解密的方法。用户可以将大部分解密的计算量委托给云存储服务器，从而减少了每个用户的计算代价。

1.2 相关工作

Deng等[5]注意到多个同行公司在合同制定之后需要共享数据的情形，在不泄漏同行公司内部组织结构的前提下解决了如何产生委托私钥的问题。文献[5]在合数阶群上，利用子群正交性，构建了层次化的CP-ABE方案，适合于大型组织单位的应用场景。

Li等[6]将时间分成时间片Δt，利用散列函数的单向性和数字签名的不可伪造性，设计了PHR（personal health record）应用背景下用户限时写权限控制方案，其主要贡献在于：1）组织单位org无需一直在线，只需要在每个时间片开始的时候周期性分发h_{n-i}即可；2）由Owner签发写权限的时限，有效控制了写权限的开始时间和结束时间。不足之处在于：1）所针对的数据文件是作为一个写授权单元，不符合多个数据分别作为写授权单元的应用场景，也没有对数据文件的每个逻辑部分进行更细粒度的控制；2）Origination（AA）只有一个，不符合多AA的应用场景；3）只考虑了PHR档案数据的创建写，没有考虑PHR档案数据的追加写。

Ferrara等[7]在RBAC访问控制方法中也提到写权限的控制，然而，为了简单起见，设定只有管理者Manager才能写文件。

Zhao等[8]提出了Many-Write-Many-Read的数据共享访问控制概念，但在方案实现时，每个修改者独立进行写操作，没有考虑多个修改者同时进行合作写操作的情形。Ruj等[9]也提出了Many-Write-Many-Read概念：在用户提交信息时，同时提交文件创建时指定的Claim Policy，云端收到Claim Policy之后验证用户身份，如果用户是授权的，则云端允许用户写文件。但论文没有给出具体方案，也没有考虑多个修改者同时进行合作写操作的情形。

Hur等[10]基于提出two-party computation protocol，用在KGC(key generation center)和DSC(data storage center)之间，解决了密钥托管问题(key escrow problem)。

Yang等[11]提出了高效安全的DAC-MACS(data access control for multi-authority cloud storage)方案，支持用户从多授权机构AA获取属性私钥，Owner可以从多授权机构AA获得属性公钥，适合多应用系统的云存储场景。

Yang等[12]针对CP-ABE中Policy动态变化带来的低效率问题，针对不同的访问策略，设计了访问策略动态更新算法。密文存在云端时，动态变化的访问策略追加在密文上。

Herranz等[13]在GDH（CDH<可计算Diffie-Hellman>问题难解决，而DDH<可判定Diffie-Hellman>容易解决）的理论基础下，提出了联合签名方案。不足之处在于，签名方案未考虑用户集合的动态变化。

Lewko等[14]定义了Semi System空间，定义了$Game_k$、$Game_k^N$、$Game_k^T$这3个安全游戏过程，在一系列假设下，给出了从选择安全到完全安全的证明路线：$Game_{\text{real}} \to Game_0$、$Game_{k-1} \to Game_k^N \to Game_k^T \to Game_k$、$Game_Q \to Game_{\text{final}}$。

郭树行等[15]基于动态情景网关，提出了一种协同访问控制模型DSGAC，定义了情景要素，进行了情景的构造和推演，给出了基于情景的协同访问应用架构。

林果园等[16]基于BLP（bell-LaPadula）模型和Biba模型，考虑行为动态调节访问范围，提出了CCACSM模型，提高了云存储访问控制的灵活性。

Li[6]方案是针对PHR医疗系统提出的，具有很大的应用局限性。系统在启动时，就知道写权限开启和禁用的确切时间，例如，医疗系统启动时，就已经知道医生的上班和下班的确切时间。Li[6]方案用散列链和数字签名技术实现了写授权控制。Ferrara等[7]设定只有管理者Manager才能写文

件。Zhao 等[8]和 Ruj 等[9]都考虑了 Many-Write-Many-Read 的情形，但是，Zhao 等[8]只考虑了每个修改者独立进行写操作，没有考虑多个修改者同时进行合作写操作的情形。Ruj 等[9]提出了用 Claim Policy 控制单个用户写授权，但是只给出了思路。郭树行等[15]给出了基于情景的协同访问应用架构，对云存储协同访问控制具体方案的进一步研究具有指导意义。

2　建模

2.1　系统模型

本文常用的符号说明如表 1 所示。系统模型如图 1 所示，系统由 5 个实体组成：证书授权中心(CA, certificate authority)、属性授权中心(AA, attribute authorities)、云(Cloud)、数据拥有者(Owner) 和数据修改者(Mender)。

表 1　　符号说明

符号	说明
I_k	授权机构 AA_k 管理的属性集合
PK_{A_k}	授权机构 AA_k 的公钥
SK_{A_k}	授权机构 AA_k 的私钥
GSK_{u_t}	用户 u_t 的全局私钥
$SK_{u_t,k}$	授权机构 AA_k 发行给用户 u_t 的属性私钥
$(\hat{D}_j, D_j^{'})$	属性私钥 $SK_{u_t,k}$ 中的一部分
Key_{sym}	用来加解密数据文件的对称密钥

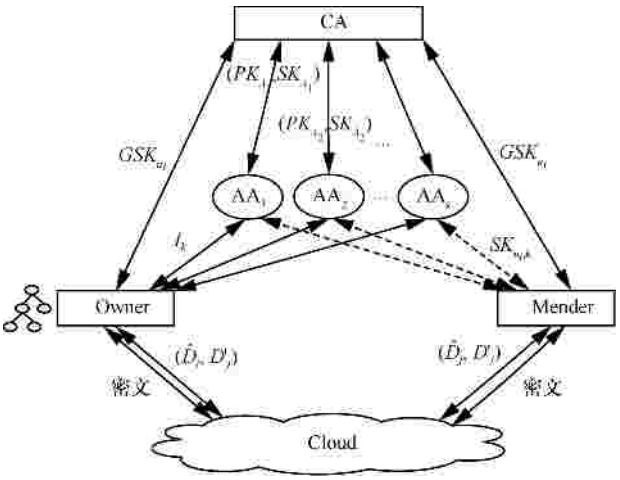

图 1　系统模型

CA：可信任的全局证书授权中心。系统启动时，AA、Owner 和 Mender 都向 CA 申请注册，CA 为它们分别分发全局密钥。

AA：可信任的属性授权中心。负责管理属性集合，为所有用户生成、分发、撤销、更新读授权私钥，同时与 Owner 合作为 Mender 生成、分发、撤销、更新写授权私钥。

Cloud：半信任的公共存储服务提供者。永远在线，负责存储 Owner 和 Mender 提交的数据，并验证用户是否具有写权限，为用户提供随时随地的数据访问服务。

Owner：用读策略树和写策略树对数据进行 CP-ABE 加密，将密文上传云端。当有 Mender 对数据进行修改时，Owner 通过与 Cloud 进行交互，审核所修改的内容，确保 Cloud 上所存储的是最新修改状态（Write-Modify）的数据。

Mender：从 Cloud 获取密文数据后，用自己的属性集合与 $policy_r$ 进行匹配，获得对称密钥 Key_{sym}，读取数据。Mender 可以修改数据，并将修改好的数据用对称加密算法加密后，上传到云端。

2.2　安全定义

防止共谋：即使非法 Mender'拿到合法 Mender 的写权限，也无法修改云端数据文件。而且，Mender 和 Mender'不能通过合并其私钥而修改原本单独不具备写权限的数据。

数据机密性：Owner 和 Mender 提交的数据在云端加密存储，Cloud 或非授权用户无法获知数据的内容。

3　多用户协同访问控制方案

3.1　问题陈述

用 CP-ABE 方法对云存储数据进行访问控制时，多用户同时从云端获取同一数据文件后，各自在用户端对数据进行解密、读取明文时，不会造成混乱。但多用户对云端同一个数据文件进行协同写时，会引发新的问题：1）在保证数据文件机密性的前提下，多个合法用户如何有序地同时写同一个数据文件；2）合法的用户可能对文件的多个逻辑分块具有不同的访问权限，因而，访问控制粒度应该细化到对文件逻辑分块的控制；3）文件的逻辑分块结构可能会因 Owner 的修改而动态变化，数据文件的修改版本动态变化。所以，云存储下的多用户数据协同权限控制方案应能适应上述的动态变化。

针对上述问题，本文提出了多用户协作访问控制方案 MCA-F（见 3.5 节）及 MCA-B 方案（见 3.6 节），前者适用于单个数据文件作为最小控制粒度的情形，后者适用于文件的逻辑分块作为最小控制

粒度的情形。

3.2 基本思想

能够读数据的用户不一定能够写数据，即读授权用户集合不一定等于写授权用户集合。所以，要考虑读权限的用户集合与写权限用户集合不一致的情形。设计了加密层次结构(如图 2 所示)。属性集满足 $Policy_r$ 的用户可以读数据，属性集满足 $Policy_w$ 的用户可以写数据。

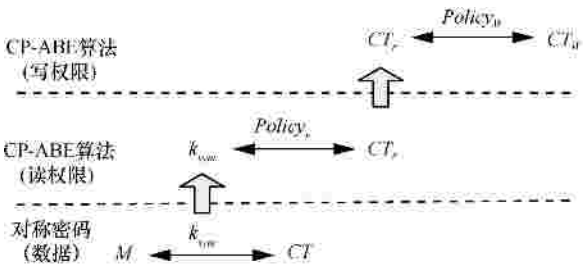

图 2 加密层次结构

考虑 Owner 瓶颈问题，显然不能由 Owner 收集每个 Mender 的修改部分，应该将所有 Mender 的修改部分上传 Cloud，由 Cloud 保留修改部分；考虑到 Owner 或者 Mender 不能永远在线，因而，修改部分不应让 Mender 保留在本地，应该及时上传云端。Owner 上线之后，决策哪些修改部分应该保留，并将最新决策后的版本加密，上传 Cloud。

MCA-B 方案中，Owner 将文件分割成多个逻辑分块，逻辑分块是 MCA-B 方案的最小控制对象。在文件分割表（table of file splitting）中记录文件是如何划分成逻辑分块的。子数据掩码 SDM（sub data mask）也由 Owner 指定。SDM 在 Owner 定义 $policy_w$ 的时候，指定哪些属性集的用户对 $Data_{\text{file}}$ 的每个逻辑分块是否拥有写权限。SDM =1010 表示该拥有属性集的 Mender 可以修改数据文件的第 1 个和第 3 个逻辑分块。也就是说，对于一个数据文件，在满足写权限控制树的前提下，满足逻辑分块访问策略的 Mender 拥有合适的 SDM。Cloud Server 根据每个用户属性私钥解密出 SDM，并以此作为判断用户是否具有逻辑分块写权限的依据。根据解密出来的 SDM，Cloud Server 选择是否暂存（semi-store）用户提交的逻辑分块。

考虑 Owner 上线后，需要对多个 Mender 提交多个 $Data_{\text{file}}$ 部分的检索，本文设计了数据修改索引矩阵 $\boldsymbol{DM}$（index matrix of modification）：以 $Data_{\text{file}}$ 的各个逻辑分块序号为列，以多个 Mender 为行，将 Owner 作为第 0 行。索引矩阵 $\boldsymbol{DM}$ 由 Owner 定义，由 Cloud Server 维护，用来记录每个 Mender 暂存云端的文件逻辑分块。

3.3 加密的层次结构

加密用分层的方法可以提高更新时的灵活性，即底层的密文更新并不影响上层的密文，反之亦然。如图 2 所示，系统设计了 3 层加密结构：最低层是对数据进行加密和解密，用 EncryptData 算法和 DecryptData 算法。中间层是对读权限进行控制，用 EncryptRead 算法和 DecryptRead 算法。最高层是对写权限进行控制，用 EncryptWrite 算法和 DecryptWrite 算法。

用户在往云端写数据时，必须有数据版本的控制。因而 EncryptRead 算法和 EncryptWrite 算法的设计有所区分。也就是说，尽管 EncryptRead 算法和 EncryptWrite 算法都用 CP-ABE 作为基本加密方法，但是二者并不能完全相同。另外，为了更细粒度地控制多用户协同写文件过程，本文 3.6 节也描述了 EncryptWritePart 算法和 DecryptWritePart 算法。

3.4 MCA-F 方案的框架

MCA-F 方案的框架如图 3 所示，详述如下。

Initialization 阶段，调用 InitialCA 算法，生成全局公共参数 $param$ 和主钥 MK_1、MK_2，将 $param$ 和 MK_1 发送给申请注册的 Owner，将 $param$ 和 MK_2 发送给申请注册的 AA_k。调用 $SetupCA$ 算法，为用户生成全局公钥 GPK_{u_t} 和全局私钥 GSK_{u_t}，将 GPK_{u_t} 发送给 AA_k，将 GSK_{u_t} 发送给 Owner 和 Mender。

Key Generation 阶段，调用 $KeyGen$ 算法，生成用户属性私钥 $SK_{u_t,k}$，并将 $SK_{u_t,k}$ 通过密钥交换协议发送给用户（包括可以读数据的 User 和可以写数据的 Mender）。方案中的属性字符串可以在 Key Generation 阶段由 Owner 任意指定，无需在 Initialization 阶段以枚举类型罗列出来，这样灵活性强。

Encryption & Write-Grant 阶段，Owner 定义具有读权限的策略 $policy_r$ 和具有写权限的策略 $policy_w$，调用 EncryptData 算法，用对称密钥 Key_{sym} 对明文数据 M 运行对称密码算法进行加密得到 CT。然后调用 EncryptRead 算法，用 $policy_r$ 对对称密钥 Key_{sym} 进行 CP-ABE 加密得到 CT_r，再调用 EncryptWrite 算法将 CT_r 的一部分 C_r 和 $policy_w$ 作为输入进行 CP-ABE 加密，得到 CT_w，设置 version

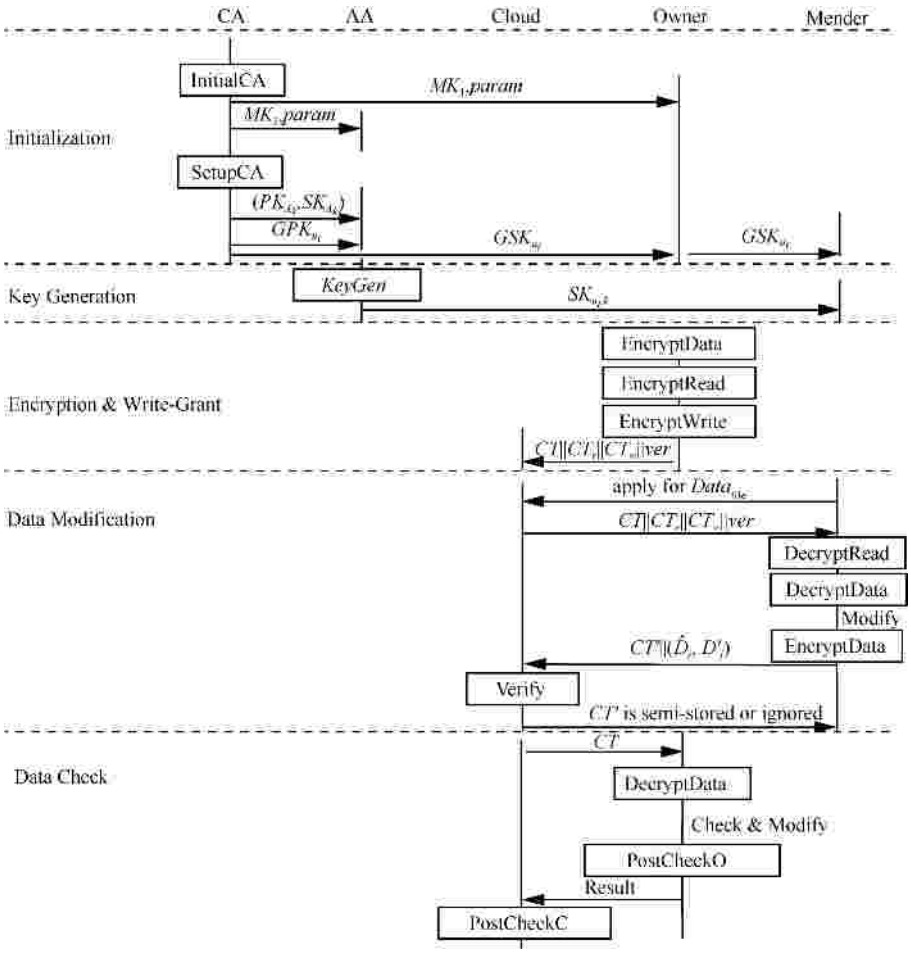

图 3　MCA-P **方案框架**

number=0，**将** $CT||CT_r||CT_w||ver$ **上传到云端。**

Data Modification **阶段，当** Mender **向** Cloud **申请数据时，** Cloud **发送** $CT||CT_r||CT_w||ver$ **给** Mender，Mender **收到之后，使用用户属性私钥** $SK_{u_t,k}$ **运行** DecryptRead **算法进行解密后，可得到对称密钥** Key_{sym}，**并用** Key_{sym} **解密密文** CT，**可以看到数据明文** M，Mender **可以修改数据，然后用对称密钥** Key_{sym} **运行** EncryptData **算法加密，得到** CT' **之后，将属性私钥** $SK_{u_t,k}$ **中的** (D_j, D'_j) **进行机密性保护，得到** $(\hat{D}_j, D'_j)$，**然后连同** CT' **一起发送给** Cloud。Cloud Server **收到** CT' **和** $(\hat{D}_j, D'_j)$ **后，运行** Verify **算法验证** Mender **是否具有写权限：将** $SK_{u_t,k}$ **与** CT_w **作为输入，运行** DecryptWrite **算法得到** CT_{verify}，**如果** CT_{verify} **等于** C_r，**则** Cloud Server **将** CT' **存储为** semi-stored **状态(还未被** Owner **审核)。如果** CT_{verify} **不等于** C_r，**则写失败，** Cloud Server **忽略** CT'。

Data Check **阶段，当** Owner **上线时，从** Cloud **获取** Mender **提交的** CT'，**用对称加密密钥** Key_{sym} **运行** DecryptData **算法解密后，查看修改数据** CT'。**运行** PostCheckO **算法，反馈结果** Result **给** Cloud Server，Cloud Server **根据** Result **的具体内容，运行** PostCheckC **作相应的处理。**

3.5　MCA-F 方案的详细算法

G **和** G_T **表示具有素数阶** p **的双线性群。**g **表示群** G **的生成元。** $e: G \times G \to G_T$ **表示双线性映射。** $H: \{0,1\}^* \to G$ **表示散列函数。**

3.5.1　InitialCA 算法

InitialCA **算法运行在** CA **上。**CA **选择双线性群** G **和** G_T，g **为** G **的生成元。选择** $e: G \times G \to G_T$ **为双线性映射。**CA **选择散列函数** $H: \{0,1\}^* \to G$。**群** G **和散列函数** H **以全局参数** $param$ **的形式公开。**CA **生成主钥如下。**

$MK = (MK_1, MK_2), MK_1 = e(g,g)^a, MK_2 = g^b$，**其**

中，$a,b\in Z_P^*$。

MK_1 发布给所有的 Owner，而 MK_2 发布给所有的授权机构 AA_k。

3.5.2 SetupCA 算法

SetupCA 算法运行在 CA 上。CA 为每个合法的授权机构 AA_k 选择随机数 $a_k\in Z_p$，并为 AA_k 生成公钥 PK_{A_k} 和私钥 SK_{A_k} 如下，其中，$a_k\in Z_p$。

$$PK_{A_k}=g^{a-a_k},SK_{A_k}=g^{a_k-b}$$

私钥 SK_{A_k} 通过密钥交换协议秘密发行给每一个 AA_k。接着，CA 为每个合法的用户 u_t 生成公钥 GPK_{u_t} 和私钥 GSK_{u_t}，其中，$r_t,r_u\in Z_p^*$。

$$GPK_{u_t}=g^{r_t},GSK_{u_t}=g^{r_u-r_t}$$

GPK_{u_t} 公开发布给所有的授权机构 AA_k，GSK_{u_t} 通过密钥交换协议发送给每一个用户 u_t。

3.5.3 KeyGen 算法

KeyGen 算法运行在授权机构 AA_k 上。AA_k 根据每个用户 u_t 的属性集 $I_{u_t,k}$，为每个用户生成属性私钥 $SK_{u_t,k}$。具体过程如下：授权机构 AA_k 为用户的每个属性选择随机数 $r_1,r_2,\text{L},r_j,\text{L}\in Z_p^*$，然后计算用户属性私钥 $SK_{u_t,k}$ 如下

$$SK_{u_t,k}=(D,\{(D_j,D_j^{'})\}_{l_j\in I_{u_t,k}})$$

$$D=MK_2PK_{A_k}SK_{A_k}GPK_{u_t}=g^ag^{r_t}$$

$$D_j=g^{r_t}H(l_j)^{r_j}$$

$$D_j^{'}=g^{r_j}$$

AA_k 通过密钥交换协议将 $SK_{u_t,k}$ 发送给用户。

3.5.4 EncryptRead 算法

EncryptRead 算法运行在 Owner 上。用对称加密算法将明文 M 加密成 CT，加密密钥称为内容密钥 K_{sym}。然后用 CP-ABE 方法加密 K_{sym}，得到 CT_r，CP-ABE 加密时使用数据读策略 $policy_r$。接着 Owner 将 $CT\,||\,CT_r$ 发送给 Cloud。

用于数据读取控制的 CP-ABE 加密过程如下：选择随机数 $s\in Z_p^*$，令 $q_R(0)=s$，其中，R 表示 $Policy_r$ 的根节点。$q_R(x)$ 表示根节点 R 的多项式。Owner 以嵌套的方式设置每个节点 x 的多项式 q_x。每个节点 x 的门限制表示为 k_x，q_x 的度设为 $d_x=k_x-1$。q_x 的常数项 $q_x(0)$ 设置为 $q_{parent(x)}(index(x))$，q_x 的其他项系数是随机选取的整数。CT_r 的计算如下。

$$CT_r=(policy_r,\hat{C},C_r,\{(C_i,C_i^{'})\}_{l_j\in S_T})$$

$$\hat{C}=K_{sym}e(g,g)^{a\cdot s}$$

$$C_r=g^s$$

$$C_i=g^{q_i(0)}$$

$$C_i^{'}=H(l_i)^{q_i(0)}$$

其中，S_T 是访问策略树所有属性（即叶子节点）的集合。

3.5.5 DecryptRead 算法

DecryptRead 算法运行在用户端。用户 u_t 用自己的全局私钥 GSK_{u_t} 修正属性密钥的 D_j 部分

$$\hat{D}_j=D_jGSK_{u_t}=g^{r_t}H(l_j)^{r_j}g^{r_u-r_t}$$

用户 u_t 将 $(\hat{D}_j,D_j^{'})$ 发送给 Cloud。然后，Cloud Server 从访问策略树 $policy_r$ 的根节点 R 开始，以递归方式，运行 DecryptNode 算法得到 $TA_r=DecryptNode_R$，接着 Cloud Server 将 TA_r 和 $CT\,||\,CT_r$ 发送给用户 u_t。

用户 u_t 收到 TA_r 和 CT_r 后，计算内容密钥 K_{sym} 如下。

$$\hat{D}=DGSK_{u_t}=Dg^{r_u-r_t}$$

$$\frac{\hat{C}TA_r}{e(C_r,\hat{D})}=\frac{K_{sym}e(g,g)^{as}e(g,g)^{r_us}}{e(g^s,g^ag^{r_t}g^{r_u-r_t})}=K_{sym}$$

用户得到内容密钥 K_{sym} 后，解密 CT，读取明文 M。

3.5.6 EncryptWrite 算法

EncryptWrite 算法运行在 Owner 上。Owner 可以创建文件，并将写权限授权给修改者 Mender。为了实现写权限的授予，在 $CT\,||\,CT_r$ 后面加上 CT_w，变成 $CT\,||\,CT_r\,||\,CT_w\,||\,ver$ 后上传到云端。其中，CT_w 是将 $policy_w$ 和 CT_r 作为输入运行 EncryptWrite 加密算法得到的，意在控制写权限。具体加密过程如下。

选择随机数 $s_w\in Z_p^*$，令 $q_R(0)=s_w$，构造 CT_w。

$$CT_w=(policy_w,\hat{C}_w,C_w,C_{w,i},C_{w,i}^{'})$$

$$\hat{C}_w=C_re(g,g)^{as_w}$$

$$C_w=g^{s_w}$$

$$C_{w,i}=g^{q_i(0)}$$

$$C_{w,i}^{'}=H(l_i)^{q_i(0)}$$

由于 Owner 在这里是创建数据文件 $Data_{\text{file}}$，因

而，设置数据文件的版本号 ver 为 0。

3.5.7 DecryptWrite 算法

DecryptWrite 算法运行在 Cloud Server 上，代替 Owner 对 Mender 进行写权限验证时所用。

与 DecryptRead 算法相同，用户 u_t 将 $(\hat{D}_j, D_j^{'})$ 发送给 Cloud，根据 $policy_w$ 和 CT_w，Cloud Server 可以通过计算获得 $TA_w = DecryptNode_R$，接着 Cloud Server 选择随机数 $v \in Z_p^*$，令 $TA_w^{'} = TA_w g^v$。然后，将 $TA_w^{'}$ 和 CT_w 发送给用户。用户收到 $TA_w^{'}$ 和 CT_w 就可以计算 T

$$T = \frac{\hat{C}_w TA_w^{'}}{e(C_w, \hat{D})} = \frac{C_r e(g,g)^{as_w} e(g,g)^{r_u s_w} g^v}{e(g^{s_w}, g^a g^{r_t} g^{r_u - r_t})} = C_r g^v$$

其中，$\hat{D}_j$ 在 DecryptRead 算法运行时就已经被计算。

用户将 T 返回 Cloud，Cloud Server 就可以计算 $Tg^{-v} = C_r g^v g^{-v} = C_r$。

3.5.8 Verify 算法

Verify 算法运行在 Cloud Server 上。当 Cloud 收到一个写数据请求分组（包括 $CT^{'} \| (\hat{D}_j, D_j^{'})$）时，Cloud 代替 Owner 验证用户是否是合法的 Mender，具体过程如下。

Cloud Server 将 $(\hat{D}_j, D_j^{'})$ 和 CT_w 作为输入，运行 DecryptWrite 解密算法，得到 C_{verify}。

验证：$C_{\text{verify}} \stackrel{?}{=} C_r$，如果不成立，Cloud Server 认为发送者不是合法的 Mender，因而丢弃 $CT^{'} \| SK_w$。如果上式成立，则 Cloud Server 认为发送者是该数据的合法 Mender，就将 $CT^{'}$ 附在 CT_w 之后，并设置其状态为 semi-stored。

3.5.9 PostCheckO 算法

PostCheckO 算法运行在 Owner 上。Owner 下载 $CT^{'}$，运行 DecryptData 算法解密数据之后，进行审查，会产生 3 种情况。1) Owner 不准备采纳 Mender 对数据的修改。2) Owner 准备采纳修改，并发现 version number($CT^{'}$)等于 version number(CT)+1。3) Owner 准备采纳修改，并发现 version number($CT^{'}$)小于 version number(CT)+1。

对于上述第 1)种情况，Owner 向 Cloud Server 发送删除 $CT^{'}$ 的 message。对于第 2)种情况，Owner 将 version 加 1，发送覆盖 CT 的 message。对于第 3)种情况，Owner 将 version 加 1，重新加密生成 $CT^{''}$，并发送 $CT^{''} \| ver$ 到 Cloud。

3.5.10 PostCheckC 算法

PostCheckC 算法运行在 Cloud Server 上。Cloud 收到 Owner 发来的 message 或者 $CT^{'}$。如果是 PostCheckO 算法中产生的情况 1)发来的 message，则 Cloud 直接删除 $CT^{'}$。如果是情况 2)发来的 message，用 $CT^{'}$ 覆盖 CT，并删除 $CT^{'}$。如果是情况 3)发来的 message 和 $CT^{''} \| ver$，则 Cloud 保留 $CT^{''} \| ver$，删除 $CT^{'}$ 和 $CT \| ver$。

3.6 MCA-B 方案

如果一个数据文件的写权限最小粒度是整个文件，则按照 3.5 节的方案既可。但如果应用系统要求：每个用户对同一个文件的各个逻辑部分拥有不同的写权限，那么，逻辑分块就是方案的最小控制对象。需要设计文件分割表（table of file splitting）、子数据掩码 SDM、数据修改索引矩阵 ***DM***。算法也需要调整：EncryptWrite 算法、DecryptWrite 算法、Verify 算法、PostCheckO 算法和 PostCheckC 算法分别扩展为 EncryptWritePart 算法、DecryptWritePart 算法、VerifyPart 算法、PostCheckOPart 算法和 PostCheckCPart 算法。

3.6.1 EncryptWritePart 算法

EncryptWritePart 算法运行在 Owner 上。Owner 创建数据文件，分割成逻辑分块（logical block），并将写权限授权给 Mender。

为了实现逻辑分块写权限的授予，在 EncryptWrite 算法输出的 $CT \| CT_r \| CT_w \| ver$ 后面加上逻辑分块写权限控制部分 CT_p，将 $CT \| CT_r \| CT_w \| ver \| CT_p$ 后上传到云端。其中，$CT_p = \{CT_p^n\}, CT_p^n$ 是将每一个 $policy_p^n$、子数据掩码 SDM^n 和 C_r 作为输入，运行 EncryptRead 加密算法得到的。具体加密过程如下。

选择随机数 $s_1, s_2, \ldots, s_n, \ldots, s_N \in Z_p^*$，$q_R^n(0) = s_n$。加密过程如下。

$$CT_p^n = (policy_p^n, \hat{C}_n, C_n, C_{n,i}, C_{n,i}^{'})$$

$$\hat{C}_n = (SDM^n \| C_r) e(g,g)^{a \cdot s_n}$$

$$C_n = g^{s_n}$$

$$C_{n,i} = g^{q_i^n(0)}$$

$$C_{n,i}^{'} = H(l_i)^{q_i^n(0)}$$

3.6.2 DecryptWritePart 算法

DecryptWritePart 算法运行在 Cloud Server 上，

代替 Owner 对 Mender 进行逻辑分块写权限进行验证时所用。Cloud Server 收到用户提交的 $(\hat{D}_j, D_j^{'})$ 后，遍历 $CT_p = \{CT_p^n\}$，对每个 CT_p^n 尝试计算 C_r。调用 DecryptNode 算法获得 TA_n。

Cloud Server 选取随机数 $v_1, v_2, \cdots, v_n, \cdots, v_N \in Z_p^*$，令 $TA_w^{n'} = TA_w^n g^{v_n}$。Cloud 将 $TA_w^{n'}$ 和 CT_p^n 发送给用户，用户计算 T_n

$$T_n = \frac{\hat{C}_n TA_n}{e(C_n, \hat{D})} = \frac{(SDM^n \| C_r^{'}) e(g,g)^{as_n} e(g,g)^{r_u s_n} g^{v_n}}{e(g^{s_n}, g^a g^{r_t} g^{r_u - r_t})} = (SDM^n \| C_r^{'}) g^{v_n}$$

然后，用户将 T_n 发回 Cloud。Cloud 即可获得 $T_n g^{-v} = (SDM^n \| C_r^{'}) g^v g^{-v} = (SDM^n \| C_r^{'})$。

3.6.3 VerifyPart 算法

VerifyPart 算法运行在 Cloud Server 上。Cloud Server 将计算所得到的 C_{verify}（即 $C_r^{'}$）和密文 CT_r 中的 C_r 相比较，如果相同，则验证通过，Cloud 得到 SDM^n。SDM^n 表示具体对哪些逻辑部分有写权限，根据 SDM^n，Cloud Server 将 Mender 提交的文件修改部分加上首部信息（Mender 的 ID 和 SDM^n，修改部分的 part 标记）后，附在 $Data_{\text{file}}$ 后面，并将该修改数据的索引信息填入 ***DM*** 矩阵。

3.6.4 PostCheckOPart 算法

PostCheckOPart 算法运行在 Owner 上。Owner 上线之后，根据 ***DM*** 矩阵非常方便地检索某一部分哪些人做了哪些修改，或者检索某个人对哪些部分做了什么修改。对 $Data_{\text{file}}$ 的一个逻辑部分 $Data_{\text{part}}$ 有 3 种处理情形。1) 不采纳 Mender 的修改建议；2) 采纳其中一个 Mender 的修改建议；3) 折中其中 2 个及以上 Mender 的修改建议。对于第 1)个情形和第 2)个情形，Owner 无需再次加密上传 $Data_{\text{part}}$。对于第 3)种情形，Owner 需要将 2 个及以上 Mender 的修改建议糅合，形成新的 $Data_{\text{part}}$，加密并上传。

3.6.5 PostCheckCPart 算法

PostCheckCPart 算法运行在 Cloud Server 上。根据 Owner3 种情况的反馈信息，Cloud 分别作如下处理。1) Cloud 清空 ***DM*** 矩阵，并删除所有的 Mender 提交的修改逻辑块，将 ver+1。2) Cloud 用 Mender 的修改块覆盖原来的密文，清空 ***DM*** 矩阵，删除所有 Mender 提交的逻辑块，ver+1。3) Cloud 用 Owner 提交的 $Data_{\text{part}}$ 覆盖原来的密文，清空 ***DM*** 矩阵，删除所有 Mender 提交的逻辑块，ver+1。

3.6.6 updateCTp 算法

updateCTp 算法运行在 Owner 上。Owner 可能会调整数据文件的逻辑分块结构，因此，该数据文件的逻辑分块更新是由 Owner 发起的。为保证数据文件能被用户有序访问，Owner 需要做如下更新工作。

1) Owner 更新文件分割表（table of file splitting）和 ***DM*** 矩阵结构；

2) Owner 更新子数据掩码 SDM^{*n}；

3) Owner 重新计算该文件后面的 CT_p 中的 $\hat{C}_n$ 部分：$\hat{C}_n^* = (SDM^{*n} \| C_r)(\hat{C}_n / (SDM^n \| C_r))$；

4) Owner 将 $\hat{C}_n^*$ 和最新的 ***DM*** 矩阵结构上传到 Cloud Server。

Cloud Server 收到 Owner 发来的 $\hat{C}_n^*$ 和最新的 ***DM*** 矩阵结构之后，用 $\hat{C}_n^*$ 覆盖 CT_p 中的 $\hat{C}_n$ 部分，并更新云端存储的 ***DM*** 矩阵结构。

值得注意的是，Owner 在调整文件的逻辑分块结构时，可能存在 Mender 写权限的变更。针对这种实际问题，需要考虑 2 种情况：1) 已有的 $policy_p^n$ 能够覆盖需要调整写权限的 Mender；2) 已有的 $policy_p^n$ 无法覆盖 Mender，需要定义新的 $policy_p^n$。对于第 1)种情况，Owner 需要在执行 updateCTp 算法的第 2)步时，调整子数据掩码 SDM^{*n}。对于第 2)种情况，Owner 在执行 updateCTp 算法的第 2)步调整 SDM^{*n}，并在执行 updateCTp 算法的第 3)步时，将新的 $policy_p^n$ 对应的 CT_p^n 添加到 $CT_p = \{CT_p^n\}$ 中。

4 分析

4.1 数据一致性分析

方案采用 ver 版本号区分 Mender 修改数据是在哪个版本上，只区分时间顺序，避免了时钟同步的大代价。

所有的 Mender 写操作都需要经过 Owner 进行审核方能存储于 Cloud。如果一个 Mender 的写权限撤销，则该 Mender 之后提交给 Cloud 的数据在 Owner 审核时，Owner 忽略该 Mender 的修改请求。新加入的 Mender 获得写权限之后，修改数据提交

给 Cloud 时，由 Owner 审核通过即可。

4.2　安全性分析

4.2.1　抗共谋分析

即使用户之间合并其私钥，也不能获得更多权限。假设用户 u_p 的属性私钥是 $SK_{u_t^p,k}$，合并用户 u_q 的属性私钥 $SK_{u_t^p,k}$ 到自己的私钥上，得到新的属性私钥

$$SK_{u_t^p,k}=(D,\{(D_j,D_j^{'})\}_{l_j\in I_{u_t^p,k}}\cup\{(D_j,D_j^{'})\}_{l_j\in I_{u_t^q,k}}\}$$

$$=(g^a g^{r_t^p},\{(g^{r_t^p}H(l_j)^{r_j},g^{r_t^p}g^{v_t^p})\}_{l_j\in I_{u_t^p,k}}\cup$$

$$\{(g^{r_t^q}H(l_j)^{r_j},g^{r_t^q}g^{v_t^q})\}_{l_j\in I_{u_t^q,k}}\}$$

其中，r_t^p 和 r_t^q 是 CA 为用户 u_p 和 u_q 分别选取的随机数，v_t^p 和 v_t^q 是 AA 为不同用户选取的随机数。

为了解密 CT_r，DecryptNode 算法运行如下。

$\forall l_i\in I_{u_t^q,k}$，

$$DecryptNode_x=\frac{e(D_i,C_i)}{e(D_i^{'},C_i^{'})}=\frac{e(g^{r_u^p},g^{q_i(0)})}{e(g^{v_t^q-v_t^p},H(l_i)^{q_i(0)})}$$

因为 $v_t^p\neq v_t^q$，所以参数 s 不能通过 $DecryptNode_x$ 算法计算出来，因而，用户不能通过合并其他用户的属性私钥，而获得更多的权限。

4.2.2　数据机密性安全分析

明文通过对称加密算法加密成 CT 存储于云端，而加密时的内容密钥 Key_{sym} 也用 $policy_r$ 加密成 CT_r 放在云端。只有那些属性集满足 $policy_r$ 的用户才可以解密 CT_r，得到 Key_{sym}，从而使用 Key_{sym}，从 CT 中恢复出明文。非授权用户和 Cloud Server 即使获得 CT 和 CT_r，也不能够恢复明文 M。

4.3　有效性分析

表 2 给出了已有方案（Hur 方案[10]、DAC-MACS[11]）与本文所提出的 2 个方案中之间存储开销方面的比较：本文方案中，Owner 上的存储代价都比较小，更适合轻量级终端应用。但本文方案在 AA_k 和 Cloud 上的存储代价比较大，这是因为本文方案考虑了多用户协同写控制，而其他 2 个方案只考虑了读控制。

本文方案 1(MCA-F)中，AA_k 上的存储负载由主钥 MK_2、AA_k 的全局公钥 PK_{A_k}、AA_k 的全局私钥 SK_{A_k}、AA_k 所管理的属性集、GPK_{u_t} 组成，其中，MK_2 的存储量为 $|p|$，PK_{A_k} 的存储量为 $|p|$，SK_{A_k} 的存储量为 $|p|$，AA_k 管理的属性集的存储量为 $n_{a,k}|p|$，GPK_{u_t} 在 AA_k 上的存储量为 $n_{u_t,k}|p|$。每个 Owner 的存储负载由 AA_k 发行的属性集、主钥 MK_1 组成，其中，AA_k 发行的属性集的总存储量为 $\sum_{k=1}^{N_A}n_{a,k}|p|$，$MK_1$ 的存储量为 $|p|$。每个 Mender 的存储负载由全局私钥 GSK_{u_t}、属性私钥 $SK_{u_t,k}$ 组成，其中，GSK_{u_t} 的存储量为 $|p|$、$SK_{u_t,k}$ 的存储量为 $(1+2\sum_{k=1}^{N_A}n_{a,k,u_t})|p|$。云端的存储负载由 $CT\|CT_r\|CT_w\|ver$、semi-stored 状态的 $CT^{'}\|ver$ 组成，其中，$CT\|CT_r\|CT_w\|ver$ 的存储量为 $(4+2t_r+2t_w)|p|+L$，$CT^{'}\|ver$ 的存储量为 $n_{\text{mender}}L$。

本文方案 2(MCA-B)中，AA_k 和每个 Mender 的存储负载与方案 1(MCA-F)相同。方案 2(MCA-B)中，每个 Owner 的存储负载由 AA_k 发行的属性、主钥 MK_1、文件分割表（table of file splitting）组成，其中，文件分割表的存储量可忽略。云端的存储负载由密文文件、$CT\|CT_r\|CT_w\|ver\|CT_p$、Mender 上传的逻辑分块、数据修改索引矩阵 $\boldsymbol{DM}$ 组成，其中，$\boldsymbol{DM}$ 结构的存储量忽略不计，CT_p 的存储量为 $n(2+2\sum t_n^i)|p|$，Mender 上传的逻辑分块的存储量为 $\sum_{n=1}^{N}\sum_{p=1}^{M}L_p^n$。

表 3 给出了已有方案（Hur 方案[10]、DAC-MACS[11]）与本文所提出的 2 个方案之间核心算法运行时计算代价的比较：本文方案中，用户端（包括 Owner、Mender 和 User）在加密解密时的计算量都比较小。同时，Hur 方案[10]和 DAC-MACS[11]只考虑了单用户独立访问云端数据的情景。

本文方案 1(MCA-F)中，EncryptRead 算法运行在 Owner 上，计算 $\hat{C}$、C_r 和 $\{(C_i,C_i^{'})\}_{l_j\in S_T}$，其中，$\hat{C}$ 和 C_r 分别需要一次 exp，$\{(C_i,C_i^{'})\}_{l_j\in S_T}$ 需要 $2t_r$ 次 exp。DecryptRead 算法运行在 Cloud 上和 Mender/User 上，本文的 DecryptRead 算法在云端的解密过程详见文献[17]。其中，Cloud 上的运算量为 $(k+\log t_r)exp+(2k)\,pairing$，Mender/User 上的运算量为 $exp+pairing$。Verify 算法运行在 Cloud 上，计算量为 $(k+logt_w+1)exp+(2k)\,pairing$。本文方案 2(MCA-B)中，EncryptRead 算法等计算量与方案 1(MCA-F)相同，VerifyPart 算法取代 Verify 算法，计算量为 $(k+\log t_p+1)exp+(2k)\,pairing$。

表 2 **存储代价比较**

方案	AA_k	Cloud	Owner	Mender
文献[10]方案	$(n_{a,k}+3)\lvert p\rvert$	$(3+3t_r)\lvert p\rvert$	$(2N_A+\sum_{k=1}^{N_A} n_{a,k})\lvert p\rvert$	–
文献[11]方案	$(n_{a,k}+3)\lvert p\rvert$	$(3+3t_r)\lvert p\rvert$	$(3N_A+1+\sum_{k=1}^{N_A} n_{a,k})\lvert p\rvert$	–
本文方案 1 (MCA-F)	$(n_{a,k}+3+n_{u_t,k})\lvert p\rvert$	$(4+2t_r+2t_w)\lvert p\rvert+(1+n_{mender})L$	$(1+\sum_{k=1}^{N_A} n_{a,k})\lvert p\rvert$	$(2+2\sum_{k=1}^{N_A} n_{a,k,u_t})\lvert p\rvert$
本文方案 2 (MCA-B)	$(n_{a,k}+3+n_{u_t,k})\lvert p\rvert$	$(4+2t_r+2t_w)\lvert p\rvert+L+n(2+2\sum t_n^i)\lvert p\rvert+\sum_{mender=1}^{N}\sum_{p=1}^{M} L_p^{mender}$	$(1+\sum_{k=1}^{N_A} n_{a,k})\lvert p\rvert$	$(2+2\sum_{k=1}^{N_A} n_{a,k,u_t})\lvert p\rvert$

注：$n_{a,k}$：AA_k 管理的属性个数；n_{a,k,u_t}：AA_k 发行给 u_t 的属性个数；$\lvert p\rvert$：群 G 的一个元素的存储量；$n_{u_t,k}$：AA_k 管理的用户个数；n_{part}：一个文件分割出的逻辑分块数目；n_{mender}：修改同一个文件的 Mender 数量；t_r：控制读权限的访问策略树中属性的数量；t_w：控制写权限的访问策略树中属性的数量；L：一个文件的存储量。t_n^i：控制逻辑分块写权限的访问策略树中属性的数量；L_p^n：每个 Mender 上传的每个逻辑分块的存储量；N_A：密文中 AA_k 的数量。

表 3 **计算代价比较**

方案	EncryptRead 算法(Owner 上)	DecryptRead 算法(Cloud 上)	DecryptRead 算法 (Mender/User 上)	Verify 算法(Cloud 上)	VerifyPart 算法 (Cloud 上)
文献[10]方案	$(2+2t_r)exp$	0	$(k+\log t)exp+(2k+1)pairing$	–	–
文献[11]方案	$(3+6t_r)exp$	$N_A((\sum_{k=1}^{I_{A_k}}(3\,pairing+exp))+2\,pairing)$	exp	–	–
本文方案 1 (MCA-F)	$(2+2t_r)exp$	$(k+\log t_r)exp+(2k)pairing$	$exp+pairing$	$(k+\log t_w+1)exp+(2k)pairing$	–
本文方案 2 (MCA-B)	$(2+2t_r)exp$	$(k+\log t_r)exp+(2k)pairing$	$exp+pairing$	–	$(k+\log t_p+1)exp+(2k)pairing$

注：exp：指数运算；$pairing$：一次 $e(g,g)$ 双线性映射运算；k：用户属性私钥中的属性个数；I_{A_k}：密文中 AA_k 发行的属性集；N_A：密文中 AA_k 的个数；t_p：控制逻辑分块写权限的访问策略树中属性的个数。

5 结束语

本文分析了多用户协作访问云端存储的同一加密文件时产生的实际问题，提出了 2 个实用的多用户协作访问控制方案，拓展了 CP-ABE 方法在云存储中的应用场合。用户采用云辅助解密密文，Owner 采用云辅助的方法管理 Mender 提交的 semi-stored 状态的写数据。用户、Owner 和 Mender 采用低的计算代价和存储代价访问数据，适合轻量级终端用户协作访问云端数据。

参考文献：

[1] SAHAI A, WATERS B. Fuzzy identity-based encryption[C]//Advances in Cryptology - Eurocrypt 2005. Springer, Berlin Heidelberg, c2005: 457-473.

[2] GOYAL O P V, SAHAI A, WATERS B. Attribute based encryption for fine-grained access conrol of encrypted data[C]//13th ACM Conference on Computer and Communications Security. Alexandria, c2006: 89-98.

[3] BETHENCOURT J, SAHAI A, WATERS B. Ciphertext-policy attribute-based encryption[C]//IEEE Symposium on Security and Privacy. California, IEEE, c2007: 321-334.

[4] LEWKO A, OKAMOTO T, SAHAI A, et al. Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption[C]//Advances in Cryptology EUROCRYPT 2010. Springer, Berlin Heidelberg, c2010: 62-91.

[5] DENG H, WU Q, QIN B. Ciphertext-policy hierarchical attribute-based encryption with short ciphertexts[J]. Information Sciences, 2014, 275(8): 370-384.

[6] LI M, YU S C, ZHENG Y. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(1): 131-143.

[7] FERRARA A L, FUCHSBAUER G, WARINSCHI B. Cryptographi-

cally Enforced RBAC[C]//IEEE 26th Computer Security Foundations Symposium (CSF). Louisiana, IEEE, c2013: 115-129.

[8] ZHAO F, NISHIDE T, SAKURAI K. Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems[C]//Information Security Practice and Experience. Springer Berlin Heidelberg, 2011: 83-97.

[9] RUJ S, STOJMENOVIC M, NAYAK A. Decentralized access control with anonymous authentication of data stored in clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(2): 384-394.

[10] HUR J, KANG K. Secure data retrieval for decentralized disruption-tolerant military networks[J]. IEEE/ACM Transactions o Networking, 2014, (22): 16-26.

[11] YANG K, JIA X H, REN K, et al. DAC-MACS: effective data access control for multi-authority cloud storage systems[C]//INFOCOM 2013. Turin, IEEE, c2013: 2895-2903.

[12] YANG K, JIA X, REN K. Enabling efficient access contro with dynamic policy updating for big data in the cloud[C]//INFOCOM 2014. Toronto, IEEE, c2014: 2013 - 2021.

[13] HERRANZ J, RUIZ A, SÁEZ G. New results and applications for multi-secret sharing schemes[J]. Designs, Codes and Cryptography, 2013, 73(3): 841-864.

[14] LEWKO A, WATERS B. New proof methods for attribute-based encryption: achieving full security through selective techniques[C]// Advances in Cryptology CRYPTO 2012. California: Springer, c2012: 180-198.

[15] **郭树行，张禹. 基于动态情景网关的系统协同访问控制模型**[J]. **通信学报**, 2013, 34(Z1): 142-147.
GUO S X, ZHANG Y. Dynamic situation gateway based system cooperations access gated model[J]. Journal on Communications, 2013, 34(Z1): 142-147.

[16] **林果园,贺珊,黄皓. 基于行为的云计算访问控制安全模型**[J]. **通信学报**, 2012, 33(3): 59-66.
LIN G Y, HZ S, HUANG H. Access control security model based on behavior in clond computing environment[J]. Journd on Communications, 2012, 33(3): 59-66.

[17] SHI J L, et al. An access control scheme with direct cloud-aided attribute revocation using version key[C]//ICA3PP 2014. Dalian, Springer International Publishing, c2014: 429-442.

作者简介：

史姣丽（1979-），女，山西运城人，武汉大学博士生，九江学院讲师，主要研究方向为网络安全。

黄传河（1963-），男，湖北随州人，博士，武汉大学教授、博士生导师，主要研究方向为移动互联网、移动 ad hoc **网络、无线传感器网络、无线** mesh **网络、**WDM **网络、物联网、网络安全、分布并行处理。**

王晶（1986-**），女，湖南邵阳人，武汉大学博士生，主要研究方向为网络安全。**

覃匡宇（1974-**），男，壮族，广西马山人，武汉大学博士生，主要研究方向为计算机网络。**

何凯（1987-**），男，湖北黄冈人，武汉大学博士生，主要研究方向为网络安全。**