心於至善

用户可自主控制的链上权限管理模型研究与实现

李轩

东南大学

学校代码: 10286
分 类 号: TP393
密 级: 公开
U D C: 621.3
学 号: 170831

硕士学位论文

用户可自主控制的链上权限管理模型研究与实现

研究生姓名: 李轩
导 师 姓 名: 宋宇波

申请学位类别 工学硕士　　学位授予单位 东南大学
一级学科名称 网络空间安全　　论文答辩日期 2020年5月30日
二级学科名称 ____　　学位授予日期 ____
答辩委员会主席 吴蒙　　评 阅 人 盲审

2020年5月30日

学校代码: 10286
分 类 号: TP393
密 级: 公开
U D C: 621.3
学 号: 170831

东南大学
SOUTHEAST UNIVERSITY

硕士学位论文

用户可自主控制的链上权限管理模型研究与实现

研究生姓名: 李轩
导师姓名: 宋宇波

申请学位类别 工学硕士 学位授予单位 东南大学
一级学科名称 网络空间安全 论文答辩日期 2020年5月30日
二级学科名称 学位授予日期
答辩委员会主席 吴蒙 评 阅 人 盲审

2020年5月30日

东南大学

硕士学位论文

用户可自主控制的链上权限管理模型研究与实现

专 业 名 称: 网 络 空 间 安 全

研究生姓名: 李 轩

导 师 姓 名: 宋 宇 波

RESEARCH AND IMPLEMENTATION OF ON-CHAIN AUTHORITY MANAGEMENT MODEL CONTROLLED BY USERS

A Thesis submitted to

Southeast University

For the Academic Degree of Master of Engineering

BY

Li Xuan

Supervised by:

Prof. SONG Yu-Bo

School of Cyber Science and Engineering

Southeast University

2020/5/30

东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：_____ 日 期：_____

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆、《中国学术期刊（光盘版）》电子杂志社有限公司、万方数据电子出版社、北京万方数据股份有限公司有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等部分内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名：_____ 导师签名：_____ 日 期：_____

摘　要

传统的权限管理技术主要依赖于可信第三方完成权限的分配管理，数据的拥有者无法直接控制数据访问权限的授予和撤销，数据具有泄露的风险。利用区块链去中心化的特性，可以在无可信第三方的场景下实现个人自主可控的数据访问权限管理。为了实现细粒度的权限管理，现有基于区块链的权限管理实现方案通常采用密文策略属性基加密算法 (CP-ABE, Ciphertext-Policy Attribute-Based Encryption)。数据拥有者可以根据数据使用者的自身属性为其生成相应的解密密钥，将权限分配信息通过特定的访问控制结构加密，符合属性要求的数据使用者可通过区块链得到权限分配信息进而获得数据访问权限。但现有基于 CP-ABE 模型和区块链技术相结合的权限管理方案在企业级应用场景下会因为用户数过多出现管理复杂度高的问题；同时数据使用者一旦通过区块链获得数据访问权限后无法进行动态撤销。因此，现有基于区块链的权限管理方案并未完全实现数据拥有者自主可控的权限分配管理。

针对以上问题，本文提出了一种新型的用户可自主控制的链上权限管理方案。该方案通过对 CP-ABE 模型的扩展，实现了权限继承，权限授予，以及权限撤销等权限操作，使其可以灵活的应用在企业级应用场景中。该方案采用多叉树结构实现属性集合的规约描述，在此基础上支持基于角色分配的权限继承；利用基于用户属性匹配的智能合约生成协议实现了数据拥有者对数据使用者的权限授予；并利用时间戳技术以及子集覆盖算法实现了权限撤销管理，有效期一旦过期或数据拥有者主动撤销，已分配的权限即被撤销。本文主要工作以及创新点如下：

1. 针对 CP-ABE 在企业级应用场景下随着用户数的增多而复杂度增加的问题，本文提出了一种基于角色分配的 CP-ABE 扩展模型。该模型在 CP-ABE 的基础上引入属性树的概念，该属性树满足严格偏序关系，可实现数据使用者角色划分与权限继承操作。系统可以根据角色间的权限包含关系构造属性树，树内除根节点与叶子节点外的每一节点代表角色，树内的叶子节点代表每个角色的固有属性。如果属性树内两个角色间存在通路，则代表这两个角色的所有者所拥有的权限具有继承关系。CP-ABE 在进行数据加密时，基于属性树内角色间的可达关系以及角色的固有属性构造访问控制结构实现权限继承，根据角色间的层次关系实现了层次化的权限管理，降低了权限管理的复杂度。

2. 本文提出了一种基于智能合约的链上权限分配模型，该模型可基于属性实现数据拥有者对数据使用者的权限授予。该模型将权限管理的过程编写为智能合约，并将该智能合约部署在区块链节点上，当需要进行权限分配时，模型参与各方通过调用智能合约发起交易实现。数据拥有者可以通过链上交易在不依赖于可信第三

方的情况下直接实现权限分配，权限分配根据数据使用者拥有的属性实现。

3. 针对当前在区块链上利用 CP-ABE 进行权限分配后难以进行权限撤销的问题，本文提出了两种权限撤销机制以实现有效期一旦过期或数据拥有者主动撤销时权限的自动撤销。前者基于时间戳实现，通过为每一数据使用者分配有效访问时间，当时间过期权限即自动撤销；后者通过构造二叉树为不同数据使用者分配不同的版本号，当数据拥有者想主动撤销权限时即可通过改变访问控制结构内的版本号信息来实现权限撤销。通过上述两种机制，在降低权限撤销操作复杂性的同时提升了灵活性。

4. 在上述方案的基础上，完成了链上权限管理原型系统的实现及验证。该原型系统可在区块链上实现数据拥用者自主可控的权限分配、权限继承以及权限撤销功能。经测试表明，在区块链上进行状态数据查询时，吞吐量在 400TPS 以上，在进行交易时，吞吐量在 130TPS 以上，可有效部署在企业级应用场景下。

关键词： CP-ABE, 区块链, 权限管理

Abstract

Traditional permission management techniques rely heavily on trusted third parties to complete the management of the allocation of permissions, where the owner of the data has no direct control over the granting and revocation of access to the data and the data is at risk of being compromised. Using the decentralized nature of the blockchain, it is possible to achieve autonomous and controlled management of individual data access rights without the scenario of a trusted third party. In order to achieve fine-grained permission management, existing blockchain-based permission management implementations usually use the Ciphertext-Policy Attribute-Based Encryption (CP-ABE). The data owner can generate the corresponding decryption key for the data user according to its own attributes, encrypt the permission assignment information through a specific access control structure, and the data user who meets the attribute requirements can get the permission assignment information through the blockchain and obtain data access rights. However, the existing rights management solution based on CP-ABE model and blockchain technology combined with the enterprise application scenario will have high management complexity due to the number of users; at the same time, data users cannot be dynamically revoked once they obtain data access rights through the blockchain. As a result, existing blockchain-based permission management schemes do not fully enable autonomous and controllable permission allocation management by data owners.

In response to the above problems, this paper proposes a novel user-controlled, up-chain permission management scheme. By extending the CP-ABE model, this solution implements permission inheritance, permission granting, and permission revocation operations, which can be flexibly applied in enterprise application scenarios. The scheme adopts a multinomial tree structure to implement the protocol description of the property set, which supports the inheritance of rights based on role assignment; the smart contract generation protocol based on user property matching implements the granting of rights by the data owner to the data user; and the timestamp technique and subset overlay algorithm implements the rights revocation management, which revokes the allocated rights once the validity period expires or the data owner actively revokes the rights. The main work of this paper and its innovations are as follows.

1. In response to the problem that CP-ABE increases in complexity with the number of users in the enterprise application scenario, this paper proposes an extended model of CP-ABE based on role assignment. The model introduces the concept of attribute tree based on CP-ABE, which satisfies the strict bias relationship and enables data user role delineation and permission inheritance operations. The system can construct an attribute

tree based on the permission inclusion relationship between roles, with each node in the tree representing a role except for the root and leaf nodes, and the leaf nodes in the tree representing the inherent attributes of each role. When CP-ABE encrypts data, the access control structure is based on the access relationship between the roles in the attribute tree and the inherent attributes of the roles to achieve the inheritance of rights, and the hierarchical rights management is achieved according to the hierarchical relationship between the roles, reducing the complexity of rights management.

2. In this paper, we propose an up-chain permission allocation model based on smart contracts, which implements the granting of permission by data owners to data users based on attributes. The model writes the process of privilege management as a smart contract and deploys that smart contract on the blockchain node, and when a privilege assignment is required, the model participants initiate the transaction by calling the smart contract. The data owner can implement permission assignments directly through chain transactions without relying on trusted third parties, and permission assignments are implemented based on the attributes owned by the data user.

3. To address the current problem of difficulty in revoking privileges after using CP-ABE for privilege assignment on the blockchain, this paper proposes two privilege revocation mechanisms to achieve automatic revocation of privileges once the expiration date expires or when the data owner actively revokes the privileges. The former is based on timestamp, which is realized by assigning a valid access time to each data user and automatically revoking the privilege when the time expires; the latter is realized by constructing a binary tree to assign different version numbers to different data users, and when the data owner wants to actively revoke the privilege, he can do so by changing the version number information in the access control structure. With both of these mechanisms, flexibility is enhanced while reducing the complexity of revocation operations.

4. The implementation and validation of a prototype system for up-chain rights management was completed on the basis of the above-mentioned scheme. The prototype system enables autonomous and controllable privilege assignment, privilege inheritance, and privilege revocation on the blockchain by the data owner. Tests have shown that the throughput is above 400TPS for status data query on the blockchain and above 130TPS for transactions, which can be effectively deployed in enterprise application scenarios.

Keywords: CP-ABE, blockchain, permission management

目录

插图目录

表格目录

算法目录

第一章 绪论

1.1 研究背景及意义

近几十年来，随着计算机和互联网的出现以及飞速发展，给人们的日常生活方式带来了翻天覆地的变化。其中，最为引人瞩目的即为数据领域的革新，其颠覆了人们处理数据的方式，影响到了现实生活中的各个领域。在医疗领域，电子病历在逐渐取代传统的手写病历，成为各个医疗机构记录患者医疗信息的主要媒介。使用电子形式记录患者的医疗信息，大大减少了纸张的使用，避免了纸张浪费。同时电子病历使用数字化管理，实现了对患者记录更为快速的访问，可以用来进行更协调以及高效的护理。在商务领域，电子商务的出现革新了人们买卖物品的方式，在各种购物平台上，人们选好商品之后，只需填写自己必要的身份地址信息即可完成交易，第三方平台可凭借存储的这些用户个人信息实现送货到家。近年来，随着云存储的出现，更是使得人们可以将无法在本地计算机或者移动终端存储的大量数据托管在第三方平台，同时利用其强大的计算能力协助处理数据，推动了数据存储和处理方式的又一次革新。

任何事物的出现都有其双面性，互联网技术的出现给数据领域带来各种便利的同时，隐私数据泄露问题层出不穷。当前，数据面临的安全状况不容乐观，据报道，仅仅2019 年一月到九月间，全球就发生了 5183 起数据泄漏事件，涉及的数据量达到惊人的79.95 亿条，给世界各国带来了不可估量的损失。随着大数据，物联网，人工智能等信息技术革命的飞速发展，可以预见数据的价值将会越来越高，数据安全问题亟待解决。

权限管理是实现数据隐私保护最重要的手段，国际标准化组织 ISO 将访问控制服务同身份认证服务，数据保密服务，不可否认服务，数据完整性服务作为五大安全服务一起定义在其网络安全体系设计标准中，凸显了访问控制在保证数据安全中的重要性，而访问控制是实现权限管理的最重要的手段。访问控制主要解决的问题是谁在何种情况下可以访问哪些数据，其有效防止了非法用户对隐私数据的访问，实现了权限管理的功能。传统的访问控制技术主要有自主访问控制，强访问控制以及基于角色的访问控制，这些技术在发明之初很好的履行了访问控制的职责，实现了对数据隐私的有效保护。但随着互联网技术的飞速发展，网络环境越来越复杂，新的应用场景如云计算，移动计算等给访问控制提出了新的难题，为了适应新技术的发展，需要一种更为细粒度，安全，高效的新型访问控制技术。

区块链的概念在 2008 年由中本聪[1] 提出，其最初仅仅是为加密货币服务，但由于其分布式，不可篡改，可追溯，公开透明等天然的安全属性，问世之初即迅速获得了各个领域的关注，为各个领域的发展注入了新的活力。在金融服务领域，由于区块链本身就是为加密货币而出现的，区块链在金融服务领域的应用具有天然的优势，得益于区块

链分布式的点对点架构，可以省去与第三方的交易，提升交易安全性的同时提升了效率。在供应链领域[2]，区块链的可追溯性与供应链的场景完美契合，能够实现对药品，食品等与人类日常生活息息相关的物品的流通过程的实时跟踪，并且保证了其安全性。区块链在医疗领域的应用研究也越来越多，文献[3–11]提出了将区块链用于个人电子医疗信息保护的不同架构，利用区块链实现数据获取的各个流程。其他如保险领域，数字版权领域等等均有区块链的身影，可以说，未来针对区块链应用的研究将是最重要的课题之一。利用区块链进行访问控制的研究也在越来越深入，如今，分布式的概念被应用在各个领域，将区块链技术与现有的各类访问控制模型结合起来，用区块链原生的安全特性进行更为有效的权限管理符合当今的发展趋势，为访问控制注入了新的活力。

1.2 研究现状

访问控制技术[12]最早出现于 20 世纪 70 年代，其主要目标是阻止非授权主体对客体的访问并确保已授权主体能够合法访问客体，以此来保护数据的完整性，机密性以及可用性，防止发生数据泄露事件。早期的访问控制技术主要有自主访问控制技术 (DAC, Discretionary Access Control)，强访问控制技术 (MAC, Mandatory Access Control) 以及基于角色的访问控制技术 (RBAC, Role-Based Access Control)。

DAC[13,14]是一种根据主体的身份以及其所属的组进行权限分配的访问控制技术，其中自主的意思是拥有客体访问权限的主体可以自行决定是否授予其他主体权限。主体可以按照自己的意愿有选择的与其他主体共享自己的资源，在实现上主要是利用 ACL 列表来进行授权管理。MAC[15,16]是一种由操作系统决定主体是否对客体具有访问权限的访问控制技术，其不受主体本身行为的控制，主体对客体的任何操作都由一组授权规则来控制，以决定是否允许该操作。

RBAC[17,18]引入了角色的概念，主体不直接与权限相关联，而是为主体分配特定的角色，将权限再分配给角色，通过角色来控制主体对客体是否有访问权限。主体与角色，角色与权限间一般均为多对多的关系，通过改变角色的分配规则，即可灵活控制授权管理。基于角色的访问控制技术因其适用于大规模授权和授权稳定的特性被广泛应用于各个领域，是最为经典的一种访问控制模型。该模型家族的基础为 RBAC96[19]，其主要有四个成员：RBAC0，RBAC1，RBAC2，RBAC3，四个成员的关系如图1-1所示。

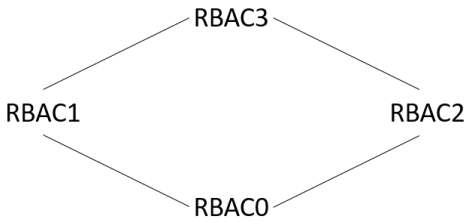


图 1-1 RBAC96 模型关系

其中，RBAC0 为基本模型，主要包含四个要素：用户，角色，会话以及授权。RBAC1 在 RBAC0 的基础上增加了角色继承，RBAC2 在 RBAC0 的基础上加入了一些约束条件，RBAC3 为统一模型，对 RBAC1 和 RBAC2 进行了整合。

随着云计算的出现，研究人员提出了一种基于属性的访问控制模型 (ABAC, Attribute-Based Access Control)。ABAC[20–22] 是一种逻辑访问控制技术，在主体授予客体访问权限时，要综合考虑主体和客体的属性，访问请求发生时的环境状况以及根据这些属性和环境指定的策略，其可以实现 DAC 和 MAC 的功能。RBAC 与 ABAC 有各自的优缺点，RBAC 易于管理但在细粒度和动态授权方面有所欠缺[23–25]，而 ABAC 能够实现细粒度的访问控制，但在属性管理方面开销较大，并且无法对授权进行审计[23–25]。F Li 等人[26] 提出了一种基于行为的访问控制技术，该技术综合了环境，角色，时间等信息，以解决在复杂网络环境下，由于用户，环境，时态等改变带来的数据访问安全问题。Ebrahim[27] 等人提出了一种针对 GSCSs 的访问控制模型，其基本思路是根据追踪到的用户当前所处的位置以及位置之间的空间关系来构造访问控制策略。范艳芳等人[28] 基于 BLP 模型，在访问控制中引入时间限制，提高了访问控制的灵活性。文献[29] 针对当前的访问控制模型的静态特性，缺乏动态职责分离，任务实例级分隔以及实时性等问题，提出了一种动态的访问控制模型 AW-TRBAC，其引入任务和工作流的概念进行身份和访问管理，通过动态的授予用户访问权限以及提供访问管理功能提高了 RBAC 模型的性能。文献[30] 基于 RBAC 提出了一种适用于制造业物联网 (MIoT) 的访问控制模型，方便了 MIoT 内的资源共享。

随着区块链技术的出现，因为其分布式，不可篡改，可追溯等特点，越来越多的研究开始集中于如何将区块链应用到权限管理中。El-Hindi 等人[31] 利用区块链作为存储层并在其上配置数据库层，利用传统的数据管理技术和标准的访问接口实现数据共享。Shangping Wang 等人[32] 提出将分布式存储系统星际文件系统 (IPFS, InterPlanetary File System)，以太坊区块链以及属性基加密算法 (ABE, Attribute-Based Encryption) 结合实现一种针对存储在分布式系统内的数据的细粒度访问控制，同时解决了传统云存储系统中可能出现的单点故障问题。数据拥有者可以将想要分享的数据加密，然后将密钥分配给使用数据的用户，并指定针对数据的访问策略。同时，通过以太坊区块链上的智能合约执行分布在分布式系统上的密文的关键字搜索，解决了在云服务器上无法返回想要搜索的所有结果以及搜索可能出错的问题。

Guy Zyskind 等人[33] 针对人们在使用第三方服务时，会失去对自己数据的完全控制权，当第三方服务在不断收集用户隐私数据时，用户可能完全不知情并且不知道该数据的用途这一安全问题，提出将链上和链下存储结合起来保护隐私数据。访问用户数据需要在区块链上进行交易，交易主要有访问控制管理交易和数据存储检索交易，通过判断交易是否合法来授予访问权限。Yingying Yao 等人[34] 针对在车载云计算环境下，由于车载设备有限的计算资源和存储容量，现存的身份管理服务 (IDaaS, Identity as a Service) 架构并不适用的问题，将改进过的密文策略属性基加密算法 (CP-ABE, Ciphertext-Policy

Attribute-Based Encryption) 与许可链相结合实现了一种适合于车载云计算的能够实现隐私保护的轻量级身份管理服务架构 IDaaSoVCC。在该架构内，与密文相关的地址信息和访问公共服务所必须的令牌被存储在区块链上。

Ouaddah 等人[35,36] 基于区块链提出了一种访问控制架构 FairAccess，其通过使用智能合约分发令牌的方式来进行访问控制。该架构的主要问题是只能基于令牌环授权，有新的访问请求以及令牌过期后都需与数据拥有者建立新的联系，时间成本较高等。针对这些问题，Pinno 等人[37] 提出了一种更适合于物联网环境的访问控制设计 ControlChain。ControlChain 内存储的内容可以分为四部份：关系区块链，上下文区块链，问责区块链以及规则区块链。其中关系区块链负责存储公共凭证以及系统参与实体间的关系；上下文区块链负责存储从传感器处理过的数据和人工输入得到的上下文信息；问责区块链负责记录是否允许访问数据的一些信息；规则区块链负责存储主体对客体的授权规则。H Es-Samaali 等人[38] 提出将区块链应用于大数据的访问控制中，其利用智能合约存储细粒度的与上下文有关的访问控制策略来进行授权决策，实现了分布式的访问控制。

1.3 本文主要内容与章节安排

1.3.1 本文主要工作

传统的权限管理技术主要依赖于可信第三方完成权限管理，数据拥有者无法直接控制数据访问权限的授予与撤销，数据具有一定的泄露风险。区块链的出现很好的解决了这一问题，利用其去中心化的特性，可以在无可信第三方的场景下实现个人自主可控的数据访问权限管理。现有基于区块链的权限管理实现方案通常与 CP-ABE 算法相结合实现更为细粒度的权限管理，数据拥有者可以根据数据使用者的自身属性生成相应的解密密钥，将权限分配信息通过特定的访问控制结构加密，符合属性要求的数据使用者可通过区块链得到权限分配信息进而获得数据访问权限。但现有将 CP-ABE 与区块链技术相结合的权限管理方案在企业级应用场景下会因为用户数过多出现管理复杂度高的问题；同时数据使用者一旦通过区块链获得数据访问权限后无法进行动态撤销，因此现有基于区块链的权限管理方案并未完全实现数据拥有者自主可控的权限分配管理。

针对以上问题，本文提出了一种新型的用户可自主控制的链上权限管理方案。该方案通过对 CP-ABE 模型的扩展，实现了权限继承，权限授予，以及权限撤销等权限管理操作，使其可以灵活的应用在企业级场景中。该方案采用多叉树结构实现属性集合的规约描述，在此基础上实现基于角色分配的权限继承；利用智能合约实现基于用户属性匹配的数据拥有者对数据使用者的权限授予；利用时间戳技术以及子集覆盖算法实现了权限撤销操作，有效期一旦过期或数据拥有者主动撤销，已分配的权限即被撤销。本文完成的主要工作如下：

1. 分析了权限管理的研究背景，概述了权限管理的国内外研究现状。

2. 针对 CP-ABE 在企业级应用场景下随着用户数的增多而复杂度增加的问题，本文提出了一种基于角色分配的 CP-ABE 扩展模型。该模型在 CP-ABE 的基础上引入属性树的概念，该属性树满足严格偏序关系，可实现数据使用者角色划分与权限继承操作。系统可以根据角色间的权限包含关系构造属性树，树内除根节点与叶子节点外的每一节点代表角色，树内的叶子节点代表每个角色的固有属性。如果属性树内两个角色间存在通路，则代表这两个角色的所有者所拥有的权限具有继承关系。CP-ABE 在进行数据加密时，基于属性树内角色间的可达关系以及角色的固有属性构造访问控制结构实现权限继承，根据角色间的层次关系实现了层次化的权限管理。

3. 本文提出了一种基于智能合约的链上权限分配模型，该模型可基于属性实现数据拥有者对数据使用者的权限授予。该模型将权限管理的过程编写为智能合约，并将该智能合约部署在区块链节点上，当需要进行权限分配时，模型参与各方通过调用智能合约发起交易实现。数据拥有者可以通过链上交易在不依赖于可信第三方的情况下直接实现权限分配，权限分配根据数据使用者拥有的属性实现。

4. 针对当前在区块链上利用 CP-ABE 进行权限分配后难以进行权限撤销的问题，本文提出了两种权限撤销机制以实现有效期一旦过期或数据拥有者主动撤销时权限的自动撤销。前者基于时间戳实现，通过为每一数据使用者分配有效访问时间，当时间过期权限即自动撤销；后者通过构造二叉树为不同数据使用者分配不同的版本号，当数据拥有者想主动撤销权限时即可通过改变访问控制结构内的版本号信息来实现权限撤销。

5. 在上述方案的基础上，完成了链上权限管理原型系统的实现及验证，并对其性能进行了测试与分析，该原型系统可在区块链上实现数据拥有者自主可控的权限分配、权限继承以及权限撤销功能。

1.3.2　本文章节安排

本文共分为七章，每一章的主要内容如下：

第一章为绪论，首先介绍了数据隐私的重要性以及数据隐私保护形式的严峻性，凸显了研究权限管理的意义。接着介绍了当前对权限管理的研究现状，讨论了区块链在权限管理中现阶段的发展，最后介绍了本文完成的主要工作以及章节安排。

第二章为相关技术研究，首先介绍了在密码学中广泛使用的双线性映射的概念，然后介绍了属性基加密算法，最后介绍了区块链的相关知识，主要包括区块链的基本概念与智能合约。

第三章介绍了基于角色分配的 CP-ABE 扩展模型，主要内容包括角色的概念，属性树的构造与操作，改进的 CP-ABE 加解密过程以及链上数据的生成。

第四章介绍了利用智能合约实现的基于用户属性匹配的权限分配模型，主要内容包括模型的整体架构设计，智能合约的权限分配流程以及该模型的安全性分析。

第五章介绍了两种权限撤销机制：基于时间戳技术和基于子集覆盖算法的权限撤销，最后给出了这两种机制的安全性证明。

第六章介绍了利用 Hyperledger Fabric 实现的链上权限管理模型原型系统，对其性能进行了测试与分析。

第七章为总结与展望，对本文所做出的工作进行了总结，并分析了提出的链上权限管理方案的不足，展望了下一步需要进行的研究工作。

第二章 相关技术研究

2.1 双线性映射理论

在数论领域，双线性映射指由两个向量空间内的元素结合生成第三个向量空间内的元素的函数操作，并且该函数对于其内的每一个参数均为线性的，矩阵乘法即为一种双线性映射。

令 V，W，Y 为同一个基础域 F 上的向量空间，则双线性映射函数 B 的定义为 $B : V \times W \to Y$。对于任意的 $w \in W$，映射函数 $B_w : v \mapsto B(v, w)$ 代表从向量空间 V 到向量空间 Y 上的线性映射。对于任意的 $v \in V$，映射 $B_v : w \mapsto B(v, w)$ 代表向量空间 W 到向量空间 Y 上的映射。从以上定义可知，如果令双线性映射的第一个参数固定同时令第二个参数不断改变，双线性映射代表线性运算，如果令第二个参数固定，第一个参数不断改变，其仍为线性运算。双线性映射 B 满足以下两个性质:

- 对于任意的 $\lambda \in F$，$B(\lambda v, w) = B(v, \lambda w) = \lambda B(v, w)$；
- 如果 $v_1, v_2 \in V$，$w_1, w_2 \in W$，则有 $B(v_1 + v_2, w) = B(v_1, w) + B(v_2, w)$，$B(v, w_1 + w_2) = B(v, w_1) + B(v, w_2)$。

如果 $V = W$ 且对于任意 V 内的任意元素 v 和 w，有 $B(v, w) = B(w, v)$，则称双线性映射 B 为对称的。如果 Y 为基础域 F，则该双线性映射被称作双线性形式，其应用于许多数学场景下，例如内积，标量积等。以循环环 R 上的模代替基础域 F 上的向量空间，对于双线性映射的定义无需任何改变，同时，该定义还可推广至多元函数上，此时双线性映射扩展为多线性映射。

对于非交换的环 R 和 S，左 R 模 M，右 S 模 N，双线性映射 B 的定义为 $B : M \times N \mapsto T$，其中 T 代表 (R, S) 阿贝尔环。对于 N 内的任意元素 n，$m \mapsto B(m, n)$ 为 R 模同态，对于 M 中的任意元素 m，$n \mapsto B(m, n)$ 为 S 模同态。对于 M 内的所有元素 m，N 内的所有元素 n，R 内的所有元素 r 以及 S 内的所有元素 s，线性映射 B 满足以下两个性质：

- $B(r \cdot m, n) = r \cdot B(m, n)$；
- $B(m, n \cdot s) = B(m, n) \cdot s$。

同时，对于每一参数，B 对于加法运算也为线性的。

由以上对于双线性映射 B 的定义可知，当 $v = 0_V$ 或者 $w = 0_W$ 时，有 $B(v, w) = 0_Y$。以 $0 \cdot 0_v$ 代替 0_v，$0 \cdot 0_w$ 代替 0_w，由 B 的线性特性可知，$B(0 \cdot 0_V, w) = 0 \cdot B(0_V, w) = 0_Y$。

空间（向量空间，模）上的所有线性映射的集合 $L(V,W;Y)$ 为从 $V\times W$ 到 Y 的所有映射的线性子空间。如果 V，W，Y 均为有限维的，那么 $L(V,W;X)$ 也为有限维的，如果 $Y=F$，也就是双线性形式，该空间维度为 $dimV\times dimW$。选择 V 和 W 上的一组基，则每一线性映射可以唯一的表示为矩阵 $B(e_i,f_j)$，反之同样成立，如果 Y 为更高维的空间，显然有 $L(V,W;Y)=dimV\times dimW\times dimY$。

双线性映射最早在密码学中的应用并没有发挥出正面的作用，直到在 2000 年，Joux[39] 利用双线性映射构造了一种三方密钥交换协议。在 2001 年，Boneh 和 Franklin[40] 利用双线性映射提出了一种安全且可用的身份基加密方案 (IBE, Identity-Based Encryption)，从此，双线性映射才被广泛应用于密码学中。

在密码学被广泛使用的双线性映射主要有两种形式[41]，第一种形式为 $e:G_1\times G_1\rightarrow G_T$，其中 G_1 和 G_T 均为素数阶 l 的循环群，第二种形式为 $e:G_1\times G_2\rightarrow G_T$，其中，$G_1$，$G_2$ 以及 G_T 均为素数阶 l 的循环群。由两种形式的定义可知，第一种形式为第二种形式在 $G_2=G_1$ 下的特殊情况。G_1 与 G_2 之间的关系还可以细分为三种类型：第一种类型为 $G_1=G_2$；第二种类型为 $G_1\neq G_2$，但在两个群之间存在有效可计算的同态 $\phi:G_2\rightarrow G_1$；第三种类型为 $G_1\neq G_2$ 并且两个群之间不存在有效可计算的同态。在实际应用中，早期的研究通常将 G_1 与 G_2 设为加法循环群，现在，通常为乘法循环群。令 G_1，G_2，G_T 均为乘法循环群，其阶数为素数阶 q，在域 Z_q 上，双线性映射关系 $e:G_1\times G_2\rightarrow G_T$ 有以下性质：

- 双线性：任意的 $a,b\in Z_q$，$g_1\in G_1$，$g_2\in G_2$，有 $e(g_1^a,g_2^b)=e(g_1,g_2)^{ab}$ 成立；
- 非退化性：$e\neq 1$；
- 可计算性：存在一种计算 e 的高效算法。

在 $G_1=G_2$ 的情况下，该双线性映射为对称的，其代表将一个群里的两个元素映射到第二个群。以上讨论的群均为素数阶的，但其也可以为合数阶。合数阶的双线性映射由 Boneh 等人[42] 引入密码学，其可以利用子群的正交特性实现更加复杂的功能，并且完成安全性证明。

双线性映射在密码学中已被广泛应用于身份基加密，群签名和其他许多研究领域。由于双线性映射细节上的实现十分复杂，许多研究都将其作为黑盒直接使用，这种方式帮助研究人员忽略了双线性映射数学和算法上的细节，能够专注于加密方面的研究。

2.2 属性基加密机制

如今，越来越多的私人隐私数据存储在第三方平台如云数据库中，为了防止储存在第三方平台上的数据的泄露，私人隐私数据需要加密之后再上传。加密数据的一个缺点是数据就只能在粗粒度水平上有选择的分享，例如，将自己的私钥发送给另一个实体。为了实现细粒度的访问控制，Amit Sahai 和 Brent Waters[43] 首先提出了一种基于属性的

加密算法，该算法由基于身份的加密体制发展而来。在基于属性的加密系统中，用户的密钥和密文与一组描述性属性相绑定，只有当密文属性和密钥属性相匹配时，密钥才可以解密相应密文。

在 Sahai 和 Waters 设计的加密系统中，只有密文和密钥之间的属性有 K 个重合时才可以解密。基于属性的加密系统主要分为两种方式，密文策略属性基加密系统 (CP-ABE, Ciphertext-Policy Attribute-Based Encryption)[44] 和密钥策略属性基加密系统 (KP-ABE, Key-Policy Attribute-Based Encryption)[45]。在 KP-ABE 中，加密方用一组描述性属性表示每一密文，每一私钥与一个访问结构相关联，这一访问结构表明了该密钥可以解密什么密文。因为私钥里指定了访问结构，密文仅仅由一组描述性属性表示，所以我们称其为密钥策略属性基加密。在 CP-ABE 中，密文与访问结构相关联，私钥与一组属性相关联，当实体对数据进行加密时，也就是依据属性值指定了一个访问控制结构，当想要解密时，只有用户的属性值满足了密文访问结构的条件，用户才可以正确解密数据。

KP-ABE 与现实中的静态场景相近，加密者将密文和一组属性相关联，然后将该密文存放在服务器上，当允许某些用户访问该数据时，加密者分配访问结构给该用户，如图2-1所示。CP-ABE 与现实中的应用场景较为相似，想要解密的用户将自身属性提交给可信机构，利用属性产生相应的解密密钥，而加密者根据是否想让某些用户能够访问某些数据来制定相应的访问控制结构，如图2-2所示。在 KP-ABE 中，加密者一旦为数据选择了一组属性，其将对谁可以访问其加密的数据失去控制，在这种情况下，加密者必须相信密钥发行方分发了适当的密钥，授予或者拒绝了适当用户的访问权限。在这种加密方式下，密钥发行方决定了谁有权限访问数据，而非加密者，而我们想要让加密方决定谁有权限访问自己加密的数据，CP-ABE 正好可以实现我们的要求。因此，在我们提出的系统中使用的是 CP-ABE，同时，我们对 CP-ABE 进行了一些改进，提出了角色分配与属性树的概念，能够更加灵活的控制用户可以访问的内容。

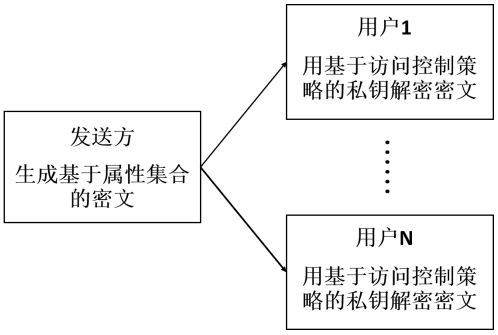


图 2-1 KP-ABE

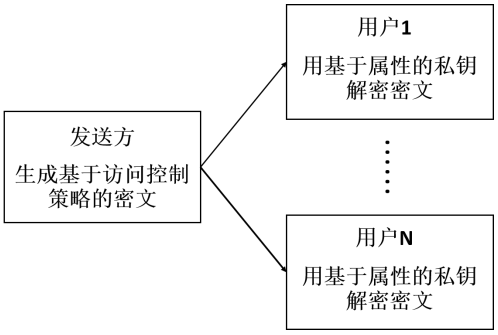


图 2-2 CP-ABE

在上文中提到的基于属性的加密系统中，访问结构和属性是两个重要的概念。首先介绍一下属性集的概念，令 $\{P_1, P_2, P_3, \cdots, P_n\}$ 为一组属性的集合，我们定义一个集合 A，其中 $A \subseteq 2^{\{P_1,P_2,P_3,\cdots,P_n\}}$，如果存在集合 B 和集合 C，当 $B \subseteq A$ 并且 $B \subseteq C$ 时，可以得到 $C \subseteq A$，那么我们就说 A 是单调的。访问控制结构就是 $\{P_1, P_2, P_3, \cdots, P_n\}$ 的一

个非空子集，也就是 $A \subseteq 2^{\{P_1,P_2,P_3,\cdots,P_n\}} \setminus \{\varnothing\}$。集合 A 内包含的属性为授权属性，集合 A 内没有包含的属性为未授权属性。访问控制结构将用户属性分为了两个类别，拥有符合访问控制结构的属性的用户才可以正常解密由该访问控制结构加密的密文。

在 CP-ABE 加密系统中，加密者将数据用特定的访问控制结构加密，该访问控制结构决定了拥有哪些属性的用户可以正确解密，加密者利用访问控制结构实现了细粒度的访问控制。访问控制结构的设计参考了秘密共享方案 (SSS, Secret-Sharing Schemes) 中的方法，SSS[46] 用来将秘密分成不同部分由不同的实体拥有，一方实体拥有的信息是该实体拥有的份额。每一秘密共享方案都实现了一些访问控制结构，这些访问控制结构决定了哪些实体拥有的秘密的份额相组合可以重建出秘密。

Shamir[46] 和 Blackley[47] 第一次提出了使用门限来构建秘密共享方案中的访问控制结构，门限即是一种逻辑运算单元，其有 $n(n \geqslant 1)$ 个输入和一个输出，每一门限可以设定一门限值 $k(0 < k \leqslant n)$。每一输入为 1 或者 0，当 n 个输入中有大于等于 k 个输入都为 1 时，门限输出为 1，反之，输出为 0。当 k 的值为 1 时，该门限可以看作或门；当 k 的值为 n 时，该门限可以看作与门；当门限值 k 在 1 和 n 之间时，代表在 n 个输入内至少有 k 个输入为 1 时，门限输出为 1。三种门限如图2-3所示。

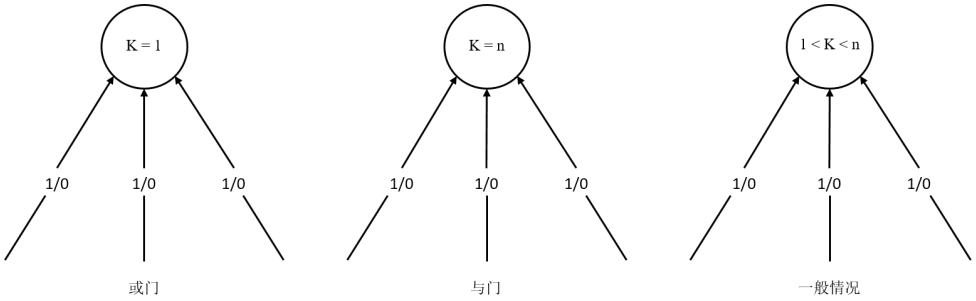


图 2-3 门限

Benaloh[48] 扩展了 Shamir 等人的观点，将访问结构用包含门限的树形结构来表示。在 SSS 中，用户可以制定访问控制树结构，在树形结构内，内部节点包含有或门和与门，叶子节点为不同的实体。任意一组符合访问控制树的实体可以合作重建秘密。因此在 SSS 中，允许各个用户串通合作，如果想要获取秘密，不同用户的串通合作还是必须的。而在 CP-ABE 内，用户的密文与访问控制结构相关联，受秘密共享方案的启发，也可以用访问控制树来表示访问控制结构，可以解密密文的用户必须拥有符合访问控制树的属性。但与秘密共享方案不同，在 CP-ABE 内，用户间的串通是不被允许的。

在 CP-ABE 内，令 T 表示树形的访问控制结构，树内的每一非叶子节点表示一个门限，该门限有孩子节点，并且有一门限值。我们用 num_x 来表示节点 x 的子节点数量，k_x 是该节点的门限值，其中 $0 < k_x \leqslant num_x$。当 $k_x = 1$ 时，该门限表示或门，当 $k_x = num_x$ 时，该门限表示与门。树形结构的每一叶子节点表示一个属性，该叶子节点的门限值为 1。为了使用的方便，我们为访问控制树结构定义几个函数，我们用 $parent(x)$ 来表示节点 x 的父节点。当 x 为叶子节点时，我们使用函数 $att(x)$ 来表示与叶子节点 x 相关联的

属性。对每一叶子节点的孩子节点，访问控制树 T 为其进行了编号，每一孩子节点的编号在 1 到 num_x 之间，我们用函数 $index(x)$ 来表示节点 x 的索引。我们以随机的方式为每一节点分配一个唯一索引。访问控制树结构可以很直观的让用户知道什么样的属性组可以解密相应的密文，一个访问控制树实例如图2-4所示。

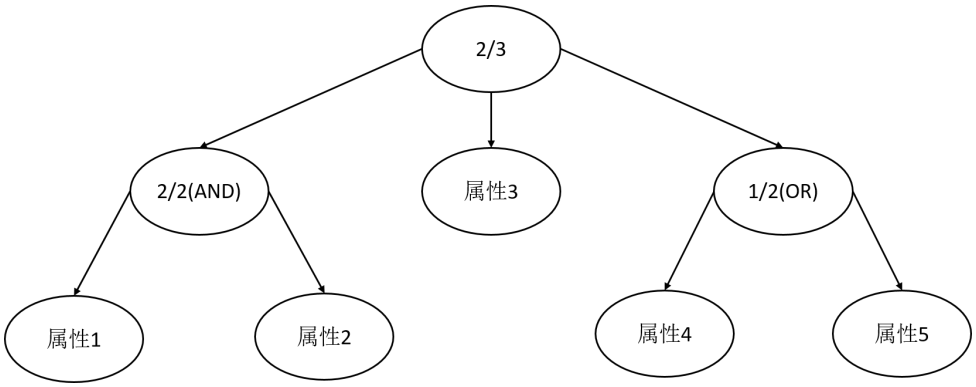


图 2-4 访问控制树

其中，所有的叶子节点都与某一属性相关，比如，在学校里，人员的身份如教师，学生，后勤都可以作为属性。所有的内部节点均表示一个门限，2/3 表示该门限有三个子节点，门限值为 2，也就是用户属性需要满足 3 个子节点中的任意两个。2/2 表示该内部节点有 2 个子节点，门限值为 2，要想满足该节点，用户需要同时满足属性 1 和属性 2，也就是说该节点相当于一个与门。1/2 表示该内部节点有两个子节点，门限值为 1，要想满足该节点，用户需要满足属性 4 或者属性 5，也就是说，该节点相当于或门。图2-4中的访问控制树结构表示，若用户属性集满足（属性 1AND 属性 2）AND 属性 3 或者（属性 1AND 属性 2）AND（属性 4OR 属性 5）或者属性 3AND（属性 4OR 属性 5）时，该用户属性集满足该访问控制树结构，可以正确解密密文。

在 CP-ABE 加密系统中，用户要想解密密文，其属性要满足相应的访问控制树，可以使用递归的算法来进行判断。令 T 为一颗访问控制树，其根节点为 r, 用 T_x 表述 T 中根节点为 x 的子树，我们可以知道，T 和 T_r 是一样的。如果一组属性 γ 满足访问控制树 T_x，我们用 $T_x(\gamma) = 1$ 来表示这种关系。我们递归的计算 $T_x(\gamma)$ 的值，递归过程如下：如果 x 是非叶子节点，我们计算其所有子节点 x' 的 $T_{x'}(\gamma)$ 值，只有当至少 k_x 个子节点的 $T_{x'}(\gamma)$ 值为 1 时，$T_x(\gamma)$ 才返回 1。如果 x 是叶子节点，那么只有当 $att(x) \in \gamma$ 时，$T_x(\gamma)$ 返回 1。

CP-ABE 算法的流程主要包括参数初始化、数据加密、密钥生成、数据解密四个步骤：

- 参数初始化: 初始化算法的输入只有一个隐性的安全参数，除此之外没有其他参数。初始化算法的输出是公共密钥 PK 和主密钥 MK；
- 密钥生成：密钥生成算法的输入为主密钥 MK 和一组描述密钥的属性，这组属性与想要获得消息 M 的用户相关联，密钥生成算法的输出为私钥 SK；

- 数据加密: 数据加密算法的输入为公共密钥 PK，消息 M，以及覆盖特定属性的访问控制结构 A。该算法将会加密消息 M，生成密文 CT，该密文只有那些拥有的一组属性满足访问控制结构的用户才可以正确解密。因此，我们可以认为密文包含访问控制结构 A；

- 数据解密：数据解密算法的输入为公共密钥 PK，密文 CT，该密文包含访问控制结构 A，私钥 SK，该私钥根据用户的一组属性 S 生成。如果一组属性 S 满足访问控制结构 A，那么该用户就可以解密密文，正确返回消息 M。

2.3 区块链技术

2.3.1 基本概念

区块链的概念最初由中本聪[1] 在 2008 年提出，其核心思想是在不需要可信机构签名的情况下为每一区块加上时间戳，并且引入一个困难参数来控制区块被添加到链上的时间，这种设计被中本聪用在比特币的核心设计模块中。比特币是中本聪提出的一种新型加密货币，在利用比特币交易的过程中，不再需要第三方机构，实现了点对点的线上电子现金交易。区块链在比特币中充当记录网络上所有交易的公开账本，使得比特币成为第一种不再依赖可信权威机构或者中心服务器也能解决双花问题的加密货币。

区块链能够防止对存储在其上的数据的更改，其为一种可以高效率的记录两个实体间交易的开放的分布式账本。区块链参与各方可以验证该交易的有效性，交易一旦达成将永久存储在区块链上。为了发挥分布式账本的作用，区块链通常由点对点网络进行管理，在该网络上遵循相应的节点间通信以及验证新的区块的协议。数据一旦被写入区块，在不改变后续所有区块的情况下无法修改，而数据写入区块需要网络上的大多数节点达成共识。

区块链的整体架构如图2-5所示，其主要包含七层：数据层，该层为区块链的核心层，主要负责区块的打包；网络层，主要包括管理区块链的 P2P 网络，交易的传播机制以及验证机制；共识层，主要包括各类共识协议，如 POW，POS，DPOS 等；激励层，主要包括数字货币的发行机制以及分配机制；合约层，该层主要包括用户设计的各类脚本代码，算法机制以及智能合约；查询层，主要包括交易的验证与状态查询；应用层，主要包括区块链当前应用较多的场景[49,50]。

区块链本质上为共享数据库，每一区块内存储若干交易信息，然后将这些区块组织成链状结构即为区块链。区块链的结构如图2-6所示，每一区块中主要包含的内容有：前一个区块的哈希值，一个随机数以及一系列交易。区块链以这种迭代的方式保证前一个区块的完整性，一直到最初的创世区块。

由于每一区块均包含前一区块的哈希值，所以当某一区块被改变时，因为其哈希值也随之改变，所以其后所有的区块都要随之改变。这样需要的计算量非常大，难度非常高，并且区块链的长度也在不断增长，因此可以认为修改区块的内容几乎不可能。在某

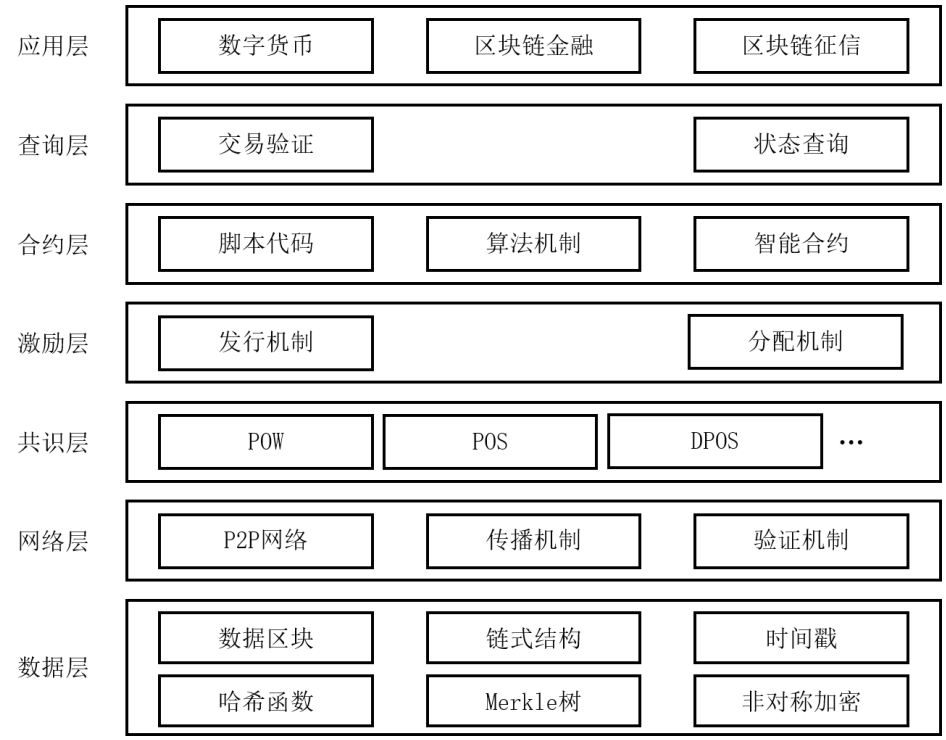


图 2-5 区块链系统整体架构

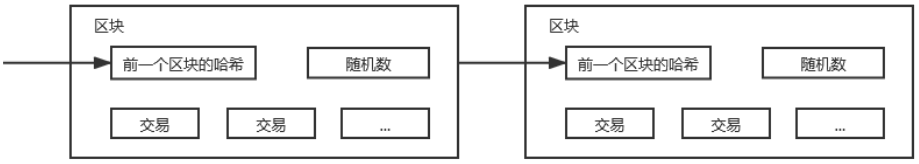


图 2-6 区块链结构

些情况下，两个独立的区块可能被同时产生出来，造成暂时的区块链分叉。为了解决这一问题，在区块链上除了记录基于哈希生成的历史区块，还会设计一种特定的评分算法。该算法依据一定的标准对出现分叉时的两条链进行评分，评分高的链作为当前区块链，没有被加入当前区块链上的区块将变为孤立区块。在出现分叉时，网络中的对等节点可能会保存不同版本的区块链，这些节点仅保存评分较高的链，并将其广播给其他节点，其他节点会将该链作为新的区块链。由以上过程可知，区块链无法保证产生的区块永远保留在链上，通常，区块被替换的可能性随着区块链长度的增长呈指数下降趋势。

以上介绍的区块链分叉为偶然的分叉，除此之外还存在故意的分叉。故意的分叉通过改变区块链各节点遵从的协议规则实现，根据兼容规则的不同，可以分为硬分叉与软分叉。当区块链进行软件升级时，一部分节点更新了区块生成协议，而另一部分节点未更新，此时就会有两个不同版本的协议在区块链中同时使用，通常认为在新协议下，区块生成的规则更加严格。软分叉是指在区块链升级后，被新协议认为合法的区块也被旧协议认为合法，其为向前兼容的。一般来说，运行新协议的节点的算力较大，新节点产

生的区块会被添加到链上而旧节点不会，旧节点所在的区块链上将存在大量的其未验证过的区块，此时，旧节点将逐步升级自身协议，直到软分叉的消失。在硬分叉中，根据新协议生成的区块将不被运行旧协议的节点接受，其不满足向前兼容的特性，如果旧节点不升级其协议，新老节点将各自维护一条自己所认可的链，最终产生两条不同的区块链。

每一区块内的一系列交易被组织成一颗默克尔树，区块内的默克尔树结构如图2-7所示，假设该区块内有 4 个交易，分别为 Tx0，Tx1，Tx2，Tx3。

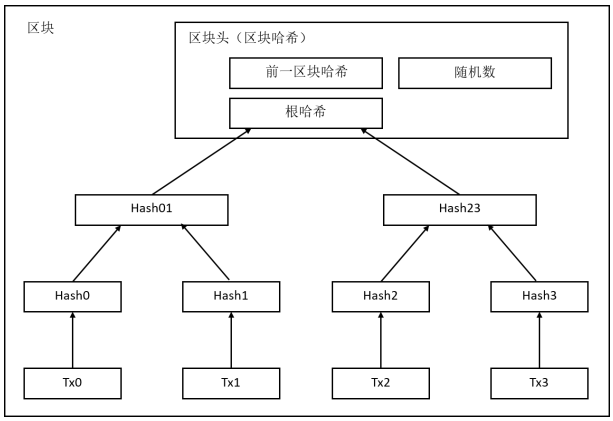


图 2-7 区块内容

我们对这四个交易分别做哈希运算得到 Hash0，Hash1，Hash2，Hash3，然后将 Hash0 和 Hash1 拼接在一起，Hash2 和 Hash3 拼接在一起，分别对拼接后的数据进行哈希运算得到 Hash01，Hash23，再将 Hash01 和 Hash23 拼接在一起进行哈希运算得到根哈希值，即默克尔哈希。在区块头内仅仅存储默克尔哈希，不必存储中间节点的哈希值。当叶子节点表示的交易的内容改变时，默克尔哈希值随之改变，区块哈希值也会改变。之前提到过每一区块都保存前一区块的哈希值，区块哈希值改变后，之后所有区块的哈希值都会改变，默克尔哈希和区块哈希一起，保证了区块内容的不可篡改。不可篡改性是区块链的主要特征之一，其他主要特征还有去中心化，可追溯，匿名性，公开透明等。

区块链将数据通过点对点网络进行存储的去中心化特性使其避免了数据存储在中心节点上的许多风险，其运行在分布式网络中以点对点的形式传递消息。点对点的区块链网络内不存在安全性差的中心节点，避免了黑客的攻击，同样，其也无需担心中心节点的故障问题。区块链的安全性依赖于公钥加密体系，其中，公钥在区块链中充当地址，私钥近似于密码的概念，可以让资产的拥有者访问其存储在区块链上的数字资产或者与区块链当前支持的各种功能进行交互。

比特币系统中的区块链使用的共识机制为工作量证明机制，这里的工作量即为计算随机数值，使得区块的哈希值满足前 k 位为 0，这里 k 可以动态调整以便控制区块生成的速度。基于工作量证明的区块链网络的运行过程主要有以下 6 步：

- 新的交易被广播到全网所有节点；

- 每一节点将一定数量的新的交易打包为一个区块；
- 每一节点为其打包的区块寻找工作量证明，即寻找符合条件的随机数值；
- 当某一节点寻找到工作量证明后，其将区块广播到全网节点；
- 当区块内的所有交易均有效并且之前没有被使用时，节点接受该区块；
- 节点开始使用接受到的区块的哈希来寻找下一区块，这一步表明节点已经接受了该区块。

利用工作量证明作为共识机制的区块链在出现分叉时通过累计工作量证明进行评分，网络中的节点选择累计工作量最大的链作为当前合法的链。累计工作量与链的长度成正比关系，因此，节点始终选择最长的链作为合法的链。如果两个节点同时广播了不同版本的区块，网络上的其他节点可能接受到的区块版本也会不同。这些节点各自在不同版本的区块上工作并扩展该链，直到某一区块链的长度更长，不在该链上的节点将会选择在新的更长的区块链上工作。新交易的广播不必到达所有的节点，只要其广播到了足够多的节点上，这些交易最后都会被包含在区块内。区块的广播也具有容错性，如果一些消息丢失了，节点未收到某个区块，当下个区块到达该节点时，该节点就会意识到其未收到某一区块，该节点会重新请求该区块。在区块链网络中，交易是点对点进行的，不需要可信第三方的介入，在这种情况下，需要一种机制保证零信任情况下的安全交易，区块链使用的是数字签名算法。区块链网络中的每一节点均可以拥有一对公私钥，私钥由自己保管，公钥公开，用户使用私钥对交易进行签名，区块链上的其他所有用户均可以用该私钥相对应的公钥验证交易的有效性。

区块链可以看做为分布式系统，其中每一节点均维护一个区块链副本，由于没有可信第三方机构的参与，需要一种机制保证所有的区块链副本的一致性与正确性，处于一致性状态，这种机制就是共识机制。共识机制就是保证参与到区块链系统中的各个节点上保存的区块链副本以及处理交易的过程是同步的，这些节点要对哪些交易是合法的，哪些交易需要添加到区块链上达成共识。为了保证区块链正常运作，共识机制是至关重要的，其确保了每一位参与者都使用同一份区块链。区块链系统中的每一位参与者都可以请求添加交易到区块链中，因此，保证所有的交易均被检查到以及区块链被所有的节点审核通过是很有必要的。如果没有一个好的共识机制，区块链是很容易遭受各种攻击的。如今，已经有很多共识机制被应用于区块链系统中，几种比较常用的共识机制如下：

- 工作量证明机制 (POW, Proof Of Work)：POW 是第一种应用于区块链中的共识机制，其最初被用于比特币中，在前文中已做过简要介绍。许多其他加密货币都效仿比特币，也使用这种共识机制。工作量证明的过程称作挖矿，每一个参与的节点被称为矿工，矿工挖矿的过程就是解决数学谜题，这个过程需要十分强大的算力。第一个解决该数学谜题的人就获得了创造区块的资格并在创造区块后获得一

份奖励。一般来说提出的数学谜题具有以下几个特点：首先，这些数学谜题均为非对称的，也就是说找到答案需要很长时间，但有了答案验证答案的正确性很简单。第二，解决这些问题的唯一方法为猜测答案，除了不断试错，没有其他方法可以更快的解出这些问题的答案。这也意味者如果参与解题的某一成员想要更快的找到谜题的解决方法，他就需要更多的算力，而拥有更多的算力意味着投入的成本更高。最后，谜题的难度随着挖出来的区块的速度而改变。为了保持新币供应量的稳定性，区块必须在固定的时间块内创造出来。如果区块生成的速度太快，谜题的难度需要提高，如果区块生成的速度太慢，谜题的难度需要降低。在比特币中，该数学谜题即为寻找区块内包含的随机数的值，使得整个区块的哈希值满足前面有若干个零，调整哈希值中零的个数即为调整该数学谜题的解决难度。工作量证明机制的主要缺点是参与者需要拥有很强的算力，耗费很多资源，随着区块链的发展，这一方式是不可取的。在加密货币系统中，挖矿者挖矿成功可以有币值奖励，但区块链如今已不仅仅应用于加密货币中，在其他系统中，可能没有相应的奖励，挖矿者也就没有了挖矿的动力。因此，区块链系统中还需要设计其他的一些共识机制；

- 权益证明机制 (POS, Proof Of Stake)：相比于 POW，POS 的应用环境更为友好。POS 中的基本假定是网络中拥有大多数币值的人作为既得利益者，其有兴趣维持网络的稳定性以保持其拥有币值的高价值。在 POS 中，使用一个随机的过程来决定谁有资格生成下一个区块，用户可以将令牌挂起成为验证器（可以生成区块的用户），也就是说其将令牌锁定一段时间，这样做之后，其就有资格来生成新的区块。决定谁可以生成下一个区块的过程需要将很多因素考虑进来，这些因素决定于区块链的具体设计。但通常来说，拥有最大股份的一方最有可能生成新的区块，另一个因素可能为成员拥有的币值的时间。生成区块的人生成区块后可以得到相应的报酬，具体什么报酬也取决于区块链的设计。通常来说，该用户可以得到所有的或者部分的其生成的区块内所有交易的交易费用，或者可以得到固定数量的代币，这些代币是由于通货膨胀产生的。在 POS 中，用户不需要拥有十分强大的算力就可以生成新的区块，大大节省了资源。与 POW 的另一个主要区别在于，在 POW 中，挖矿者可能并不拥有其正在开采的币值，这意味着他们只是在不改进网络的情况下使得自身利益最大化。而在 POS 中，因为验证者真的持有其要验证的区块链上的币值，所以其会有更大的动力维护网络。POS 也主要用于加密货币使用到的区块链中，有时也会将其与 POW 结合起来使用；

- 委托权益证明机制 (DPOS, Delegated Proof Of Stake)：DPOS 是一种非常快速的共识机制，其因在 EOS 中的实现而闻名，由于其股权加权投票系统，经常被称作数字民主。在 DPOS 中，用户可以使用他们的代币为一定数量的代表投票，他们投票的权重取决于他们拥有的代币多少。例如，A 为一名代表投票 10 个代币，B 为一名代表投票 1 个代币，那么 A 的投票权重是 B 的 10 倍。代表是指想在区块链

网络上生成新区块的个人或者组织，收到最高投票数的代表获得生成区块的权利并可以在生成这些区块后获得奖励。正如 POS 一样，这些代表获得的奖励或者来自交易费用或者是固定数量的代币，这些代币是由于通货膨胀产生的。选出多少代表来生成区块取决于区块链的具体设计。由于代表们想要获得尽可能多的选票，这会激励他们尽可能创造对区块链系统有价值的东西，因为这样可以让他们更有机会获得额外的选票。

以上，我们介绍了比较常用的几种共识机制，这些共识机制多在加密货币中应用，产生新的区块时都会有一定的币值奖励。区块链本质上属于分布式系统，解决分布式系统中的分布式一致性的其他算法如实用拜占庭容错技术 (PBFT, Practical Byzantine Fault Tolerance)[51]，Paxos 协议[52]，Raft 协议[53] 等也可以用于解决区块链中的共识问题。这些机制不涉及代币交易，不需要使用代币来鼓励参与者生成新的区块，区块链应用于除加密货币外的其他场景时可以使用这类共识机制。

区块链主要按其开发程度进行分类，可以分为私有链，联盟链以及公有链。在介绍这三种区块链的区别之前，首先简要介绍一下其共有的特点：

- 仅允许添加：为了成为一条区块链，系统必须遵循区块链的结构，每一个区块都链接到已有区块链的最后一个区块；
- 对等网络：网络中的每一个参与者都持有一份区块链的副本，这些参与者被称为节点，它们以对等的方式交互；
- 共识机制：在区块链系统中，由于没有中心权威机构，需要一种机制来保证节点对在网络上传播的交易的正确性达成共识，以确保没有伪造的交易被写到链上。

接下来，简要介绍一下这三种区块链的主要区别：

- 公有链：如今绝大多数的区块链都属于公有链的范畴，我们称其为公有的是因为任何人都可以查看已发生的交易，并且加入公有链仅仅需要简单的下载必要的一些软件，没有防火墙阻拦其加入并且任何人都可以参与共识机制。由于任何人都可以自由加入区块链并且为其在达成共识的过程中扮演的角色而获得奖励，所以公有链是高度去中心化的。同时，公有链相较于私有链更为抵制审查，由于任何人都可以参加公有链网络，网络中的协议应该实现某种机制防止恶意参与者匿名获得优势。然而，公有链为了安全性确实在性能上有所取舍，许多都遇到了扩展的障碍，吞吐量也相对较弱；
- 私有链：私有链上规定了哪些用户可以在链上进行读写操作，其并非一个去中心化的系统，因为在控制上有明确的层次结构。不过，私有链仍然是分布式的，因为许多节点维护着一份区块链的副本。私有链更适合企业设置，在企业设置中，企业希望不使其网络能被外部访问到的情况下享受区块链的属性。工作量证明机制

在开放环境中很有必要，但在私有链中，没有必要使用这种共识机制。在私有链中，每一参与者的身份都是已知的，不存在工作量证明机制想要抵御的攻击。在私有链中，更为有效的算法是指定验证器，验证器是被选择来承担某些交易验证功能的节点，通常来说，这些节点是需要在每一个区块上签名的一类节点。节点一旦有了恶意行为，需要快速的捕获这些节点并将其从网络中移除；

- 联盟链：联盟链介于私有链和公有链之间，结合了两者的一些特点，其与这两者的主要区别在于共识层面。不同于开放系统中任何人都可以判断区块的有效性，也不同于封闭系统中只有一个实体能验证区块，在联盟链中，有少数同样权力的实体可以充当验证器的功能。联盟链的规则更为灵活，链的可见性可以限制在验证器中或者已授权的个体中，或者两者都有。只要验证器能够达成共识，就可以很容易的做出更改。至于区块链的运作，只要一定数量的实体行为是诚实的，系统就不会出问题。联盟链适用于多个组织在同一行业运行的情况，并且需要一个统一的标准来进行交易或者传递信息，加入联盟链中对一个组织的好处就在于可以在组织间共享信息。

私有链和联盟链也被称为许可链，公有链属于非许可链，所以，区块链也可以分为许可链和非许可链。对于非许可链，任何人都可以匿名参与，除了不可更改的区块链的状态外都是不可信的。另一方面，许可区块链在一组已知的，确定的以及经过审查的参与者之间运行，这些参与者在一个有一定信任度的治理模型下运行。许可链为有共同的目标但并不完全信任彼此的一组实体提供了一种安全的交互方式。

2.3.2 智能合约

智能合约的概念最初在 1994 年由 Nick Szabo 提出，其出现的时间早于区块链这一概念的出现。Szabo 最早将智能合约描述为一组以数字形式定义的承诺，以及参与各方履行这些承诺遵从的协议[54]。之后，Szabo 又将智能合约定义为在无需人工干预下，执行条件自动出现的数字化可计算合同[55]。但智能合约一开始并没有引起研究人员的关注，因为当时没有数字化的平台或者分布式账本技术可以支持智能合约的发展。直到 2008 年比特币的出现，带来了区块链技术，智能合约重回大众视野，很多交易逻辑都可以编写在智能合约内，然后部署到区块链上，协助比特币的交易。

需要注意的是，由于智能合约的概念早于区块链，因此其本身的使用不依赖于区块链，但对智能合约使用的基本要求是可信的网络或机制。如果希望智能合约能够在中心化的可信网络上正常运作，需要一个可信的权威机构执行更改操作以及记录交易。显然，利用区块链可以使得智能合约的使用更为便利与有效。当有触发智能合约运行的行为发生时，区块链网络在不需要第三方干预的情况下自动执行智能合约并产生结果，这使得整个网络能够在不依赖于实体间信任的情况下实现可靠性与防篡改。因此，智能合约如今基本上均配置在区块链或者分布式账本中以协助进行通用的逻辑设计。

为了实现通用性，智能合约的概念已不仅仅局限于合约的传统概念，其可以看做任意的计算机程序。正如以太坊以及 IBM 所使用的智能合约，其已经超出了传统合约的范围。智能合约也可以被看做被安全存储的过程，因为在区块链上或者分布式账本中，一旦部署智能合约的交易达成后，调用智能合约将严格执行其上的代码并产生代码规定的结果，例如实体间一些数据的转移，这一过程无法被第三方操控，这是因为智能合约的实际执行由区块链或者分布式账本控制和审计，而不是任意的中心服务器。智能合约也可以被看做一种计算机协议，其能够以数字化的形式促进，验证或者强制合同的谈判与执行。智能合约可以在没有第三方参与的情况下进行可信交易，这些交易具有可追溯与不可逆的特性。使用智能合约的目的是享受其优于传统合约的安全性，并且减少传统合约执行时耗费的交易成本，如今，各类加密货币都实现了适于各自使用场景的智能合约，方便了货币的交易。

智能合约以分布式的方式运行，在一定条件下可以直接控制数字货币或者资产的转移，智能合约不仅像传统合约一样定义了交易的规则以及违反合约规则的惩罚，而且可以自动执行这些流程。智能合约实现这一功能的方式为：将各种信息作为输入，通过合约中设定的规则为输入赋值，然后执行合同中要求的操作，例如决定资产是归某人所有还是归还资产来源的那一方。如今，智能合约已不单单可以控制资产转移，其可以应用在更广泛的领域内，例如法律程序，保险行业，众筹协议等等，可以用传统合约解决的问题，都可以利用智能合约进行自动化的处理。智能合约有两个主要的关键点，尤其是应用于各种区块链平台上的时候：1. 许多智能合约在网络中同时运行；2. 智能合约可以被动态部署，很多情况下可以由任何人部署。

大多数现存的支持智能合约的区块链平台都遵循排序执行架构，在该架构中，通过共识协议验证交易并将交易排序，接着将这些交易广播到所有对等节点中，然后每个对等节点依次执行这些交易。排序执行架构几乎存在于现有的所有的区块链系统中，从公有链例如以太坊到许可链例如 Tendermint，Chain 以及 Quorum。区块链中基于排序执行架构运行的智能合约的运行结果必须是确定的，否则，共识可能永远不会达成。为了解决不确定性这一问题，许多区块链平台要求使用的智能合约使用非标准化或者特定的语言编写，例如 Solidity，以此来消除不确定性。这一要求阻碍了智能合约的广泛使用，因为其要求开发者为了编写智能合约需要学习新的语言并且可能会出现程序错误。更进一步的，由于所有的交易都被所有节点顺序执行，性能和规模就会受到限制。

智能合约需要在区块链系统中每个节点上执行就要求了系统需要采取复杂的措施来保证整个系统免受潜在的恶意合约的攻击，以此来保证整个系统的弹性。使用智能合约有以下几个潜在的优势：1. 成本效益：智能合约消除了很多运营成本，节省了资源，包括监控合约执行进度的人力资源；2. 处理速度：智能合约运行在自动化流程上，在大多数情况下，可以消除人为的干预，提高合约中规定的业务交易的速度；3. 自治：智能合约由网络自动执行，消除了配置第三方的需要，也消除了由于第三方参与可能导致的风险；4. 可靠性：记录到区块链上的数据不能被改变或者消除，如果智能合约的一方未

能履行其义务，另一方将被智能合约的相关条款保护。智能合约还消除了人为错误的潜在威胁并确保了执行合约的准确性。除了以上四个优点外，智能合约也有一些潜在的缺点，例如缺乏针对区块链，加密货币以及智能合约的国际化的监管阻碍了其在全球经济中的应用。智能合约的实现也比较复杂，其一旦被写入区块链就不可能再被更改，这在安全方面是一个优势但同时也使得合约的参与者无法对智能合约做出任何改变，或者在不配置新智能合约的前提下为合约纳入新的细节。

拜占庭容错技术使得以分布式的形式构建智能合约成为可能，此外，一些区块链平台上使用的具有不同图灵完备性的编程语言使得在智能合约内构造自定义的复杂逻辑成为可能。比特币的交易脚本即可以看作一种智能合约，但由于编写脚本的语言并非图灵完备的，因此其无法完成一些复杂的逻辑。而在被称为区块链 2.0 的以太坊上，实现了利用图灵完备的语言进行智能合约编写的目标，能够帮助用户定义各种复杂逻辑。在 IBM 主导的许可链项目 Hyperledger Fabric 中更是为用户提供了各种编写智能合约的接口，方便了智能合约的使用。

2.4 本章小结

本章主要对提出的用户可自主控制的链上权限管理模型涉及到的相关技术进行了简要介绍，主要包括双线性映射的概念，属性集加密以及区块链技术，在区块链技术部分，主要介绍了区块链的基本概念以及智能合约。

第三章 基于角色分配的 CP-ABE 扩展模型

本章提出了一种基于角色分配的 CP-ABE 扩展模型，该模型在传统 CP-ABE 的基础上引入了角色与角色继承的概念。首先，介绍了 CP-ABE 基本模型，主要由初始化，解密密钥生成，加密以及解密四个步骤构成。然后提出了基于角色分配的扩展模型，主要介绍了角色分配的概念，如何利用属性树构造角色间的继承关系以及针对属性树的操作函数，基于提出的角色分配实现的 CP-ABE 加解密过程。

3.1 CP-ABE 模型

CP-ABE 算法依赖于双线性映射，双线性映射的概念如下：令 G_0 和 G_1 为两个素数阶 p 的乘法循环群，g 为 G_0 的生成元，e 为一个双线性映射，即 $e : G_0 \times G_0 \rightarrow G_1$。双线性映射 e 有如下两个性质：

- 双线性：对于所有的 $u, v \in G_0$ 以及 $a, b \in Z_p$，有 $e\left(u^a, v^b\right) = e\left(u, v\right)^{ab}$；
- 非退化性：$e\left(g, g\right) \neq 1$。

如果对 G_0 的操作以及双线性映射 $e : G_0 \times G_0 \rightarrow G_1$ 都是高效可计算的，我们就称 G_0 为双线性群，注意到映射 e 是对称的，因为 $e\left(g^a, g^b\right) = e\left(g, g\right)^{ab} = e\left(g^b, g^a\right)$。

CP-ABE 的初始化如算法3.1所示。在初始化时，首先选择一个素数阶 p，生成元为 g 的双线性群 G_0。然后，选择两个随机的指数 α 和 β，并且 $\alpha, \beta \in Z_p$。生成的公钥 $PK = G_0, g, h = g^\beta, e\left(g, g\right)^\alpha$，主密钥 MK 为 (β, g^α)。

算法 3.1 CP-ABE 初始化

输入： 空值

输出： 公钥 PK，主密钥 MK

1: 选择一个生成元为 g 的素数阶 p 的双线性群 G_0。
2: 选择两个随机指数 $\alpha, \beta \in Z_p$。
3: 生成公钥 $PK = G_0, g, h = g^\beta, e\left(g, g\right)^\alpha$，主密钥 $MK = (\beta, g^\alpha)$。

公钥 PK 与主密钥 MK 生成后，可以根据 PK 与 MK 实现解密密钥生成以及数据加密。解密密钥生成过程如算法3.2所示，该过程以解密方的一组属性 S 为输入，根据这组属性生成的解密密钥如式3.1所示。

$$SK = (D = g^{(\alpha+r)/\beta}, \forall j \in S : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}) \tag{3.1}$$

算法 3.2 解密密钥生成

输入: 主密钥 MK，属性集 S

输出: 解密密钥 SK

1: 选择随机数 $r \in Z_p$。

2: 对 S 中的每一属性 j，选择随机数 $r_j \in Z_p$。

3: 对 $\forall j \in S$，计算 $D_j = g^r \cdot H(j)^{r_j}$，$D'_j = g^{r_j}$，$D = g^{(\alpha+r)/\beta}$。

4: 输出 $SK = (D, \forall j \in S : D_j, D'_j)$。

在利用 CP-ABE 进行数据加密时，加密方需要指定访问控制树，该访问控制树的叶子节点均与解密方的属性相关。加密方通过控制访问控制树的结构来控制拥有哪些属性的解密方可以解密数据，该加密过程如算法3.3所示。

算法 3.3 CP-ABE 加密过程

输入: 公钥 PK，消息 M，访问控制树 T

输出: 密文 CT

1: 为 T 中的根节点 R 选择一个多项式 q_R，q_R 的阶数 $d_R = k_R - 1$。选择随机数 $s \in Z_p$，令 $q_R(0) = s$，再随机选择其他 d_R 个点构成 q_R。

2: 为 T 中的其他节点 x 各选择一阶数为 d_x 的多项式，$d_x = k_x - 1$，并且 $q_x(0) = q_{parent(x)}(index(x))$，再随机选择其他 d_x 个点构成 q_x。

3: 令 Y 为 T 的叶子节点集，对于 $\forall y \in Y$ 计算 $C_y = g^{q_y(0)}$，$C'_y = H(att(y))^{q_y(0)}$，$\tilde{C} = Me(g,g)^{\alpha s}$，$C = h^s$。

4: 输出密文 $CT = (T, \tilde{C}, C, \forall y \in Y : C_y, C'_y)$。

加密算法的输入为公钥 PK，消息 M，根据访问策略得到的访问控制树 T，输出为密文 CT。加密算法使用访问控制树 T 来加密明文消息 M，加密算法首先为访问控制树 T 内包括叶子节点的每一节点 x 选择一个多项式 q_x。从根节点 R 开始采用自顶向下的方式构造多项式，对树中每一节点 x，其对应的多项式 q_x 的阶数为 d_x，d_x 比该节点的门限值 k_x 少 1，即 $d_x = k_x - 1$。

从根节点 R 开始，加密算法选择一个随机数 s，$s \in Z_p$，并令 $q_R(0) = s$。然后，随机选择 d_R 个其他的点来确定多项式 q_R 的表达式。对于任意其他节点 x，令 $q_x(0) = q_{parent(x)}(index(x))$，其中，$parent(x)$ 代表节点 x 的父节点，$index(x)$ 代表节点 x 在其父节点的子节点中的索引。然后选择任意其他 d_x 个节点来确定 q_x 的表达式。令 Y 为 T 的叶子节点，给定访问控制树的结构 T，可以得到密文如式3.2所示。

$$CT = \left(T, \tilde{C} = Me(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C_y{}' = H(att(y))^{q_y(0)}\right) \tag{3.2}$$

解密方利用根据自身属性生成的解密密钥解密加密数据，该过程如算法3.4所示，解

密算法的输入为密文 CT 以及解密密钥 SK，输出为明文 M 或者错误信息。

算法 3.4 解密算法

输入: 密文 CT，解密密钥 SK

输出: 明文 M 或者错误信息 $\perp$

1: 检查数据使用者的属性集 S 是否满足访问控制树 T。
2: 如果 S 不满足 T，输出 $\perp$。
3: 如果 S 满足 T，计算 $A = e(g,g)^{rq_R(0)} = e(g,g)^{rs}$。
4: 返回明文 $M = \tilde{C}/(e(C,D)/A) = \tilde{C}/(e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs})$。

解密算法为一种递归算法，为了便于说明，我们介绍一下最简单的形式的解密算法。首先定义一个递归算法 DecryptNode(CT,SK,x)，该算法的输入为密文 CT，$CT = \left(T, \tilde{C}, C, \forall y \in Y : C_y, C_y{}'\right)$，解密密钥 SK，SK 与一组属性 S 相关联，以及访问控制树 T 中的节点 x。

如果节点 x 为叶子节点，那么令 $i = att(x)$，其中 $att(x)$ 表示与节点 x 相关联的属性。递归算法 DecryptNode(CT,SK,x) 的定义如下：如果 $i \in S$，如式3.3所示。

$$DecryptNode(CT, SK, x) = \frac{e(D_i, C_x)}{e(D_i{}', C_x{}')} = \frac{e(g^r \cdot H(i)^{r_i}, g^{q_x(0)})}{e(g^{r_i}, H(i)^{q_x(0)})} = e(g,g)^{rq_x(0)} \tag{3.3}$$

如果 $i \notin S$，那么我们定义 $DecryptNode(CT, SK, x) = \perp$。

然后，我们考虑 x 为非叶子节点时的递归情况。在这种情况下，DecryptNode 函数按照如下方式运行：对节点 x 所有的子节点 z，计算 DecryptNode(CT,SK,z)，将其输出保存为 F_z。令 S_x 表示包含任意 k_x 个子节点 z 的一组节点，对于其中的每一个子节点 z，都有 $F_z \neq \perp$。如果找不到这样的一组节点 S_x，那么解密密钥不满足该节点，函数输出为 $\perp$，否则，我们计算：

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S_x'}(0)} \\ &= \prod_{z \in S_x} (e(g,g)^{r \cdot q_z(0)})^{\Delta_{i,S_x'}(0)} \\ &= \prod_{z \in S_x} (e(g,g)^{r \cdot q_{parent(z)}(index(z))})^{\Delta_{i,S_x'}(0)} \\ &= \prod_{z \in S_x} e(g,g)^{r \cdot q_x(i) \cdot \Delta_{i,S_x'}(0)} \\ &= e(g,g)^{r \cdot q_x(0)} \end{aligned} \tag{3.4}$$

并返回该结果。其中 $i = index(z)$，$S_x' = \{index(z) : z \in S_x\}$。$\Delta_{i,S}$ 表示拉格朗日系数，对于 $i \in Z_p$ 以及 Z_p 内的一组元素 S，有 $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$。

以上，我们定义了函数 DecryptNode，接着，我们通过该函数来定义解密算法。解密算法首先将函数 DecryptNode 应用于访问控制树 T 中的根节点 R，如果属性集满足访问控制树，我们令 $A = DecryptNode(CT, SK, r) = e(g,g)^{rq_R(0)} = e(g,g)^{rs}$。明文消息的计算如式3.5所示。

$$\tilde{C}/(e(C,D)/A) = \tilde{C}/(e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs}) = M \tag{3.5}$$

3.2 基于角色分配的扩展模型

3.2.1 角色分配

CP-ABE 使用特定的访问控制结构对数据进行加密，解密密钥根据一组属性生成，当解密方属性满足访问控制结构时，其可正确解密密文。CP-ABE 最大的优势是产生密钥后，加密方可以灵活改变加密时的访问控制结构控制哪些用户可以解密数据，避免了密钥的频繁分发，但其缺陷是效率较低，尤其是在企业级应用中，其无法发挥企业内部人员自身权限间固有的继承关系，造成了授权的冗余。在企业内部，根据人员特定属性，可以在其拥有的权限之间构造继承关系，例如在图3-1所示的企业内部人员架构图中，总经理，经理以及员工这样一种属性表明了这种权限继承关系：总经理拥有的权限应该包含经理拥有的权限，经理拥有的权限应该包含员工拥有的权限。但在利用 CP-ABE 进行权限分配时，其并没有考虑到这种内部固有的权限继承关系。如果通过 CP-ABE 为员工 1 分配了访问数据 D 的权限，经理 1 无法直接获得该权限，因为在加密数据 D 时，并没有在访问控制结构内把经理 1 这一属性包括进来，经理 1 无法解密通过 CP-ABE 加密 D 后得到的密文。如果经理 1 想要获得该权限，需要数据拥有者重新进行数据加密将经理 1 包含进来，增加了数据拥有者权限授予的负担。因此，本章提出角色以及属性树的概念，可以根据企业内部隐含的权限关系实现权限的授予，简化了权限管理的过程。

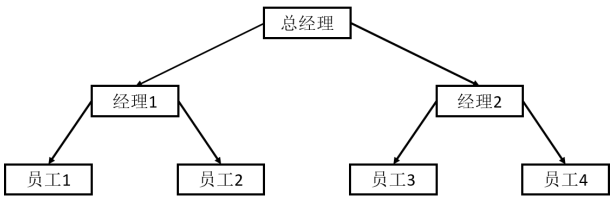


图 3-1 企业内部人员架构图

数据使用者的属性中，存在特殊属性表明数据使用者间的权限继承关系，在图3-1，这一属性为总经理，经理以及员工，本文将该属性定义为角色。同时，为角色定义角色属性，其将角色进一步细化。例如在公司内部可能存在 A 部门与 B 部门，总经理管理 A，B 两个部门，经理 1 管理 A 部门，经理 2 管理 B 部门，员工 1 和员工 2 属于 A 部门，员工 3 和员工 4 属于 B 部门，则总经理这个角色不存在角色属性，经理与员工这两个角色分别存在 A，B 两个属性。通过角色属性可以进行更为细致的权限继承划分，例

如当为 A 部门员工分配权限时，可以指定只有 A 部门经理才可继承此权限。可以看到，企业内部人员的权限继承关系呈现树形结构，因此考虑使用多叉树来实现这种权限继承关系，多叉树内的叶子节点代表角色属性，非叶子节点以及非根节点代表角色。数据拥有者在利用 CP-ABE 加密数据时，根据多叉树内的角色可达关系以及角色属性构造访问控制树实现权限继承，实现了层次化的权限管理，降低了权限管理的复杂度。

3.2.2　属性树构造

由于数据使用者拥有的角色也可以看做其属性，因此，本章将含有角色与角色属性的树形结构直接称为属性树。在属性树内，叶子节点代表角色属性，非叶子节点与非根节点代表角色，通过属性树内角色间的继承关系实现了角色间权限的继承关系。由于角色属性并不直接控制权限间的继承，因此在接下来的讨论中，将不再考虑角色属性，属性树内节点除根节点外均代表角色，同时不再区分角色与属性的概念，统一使用属性说明。

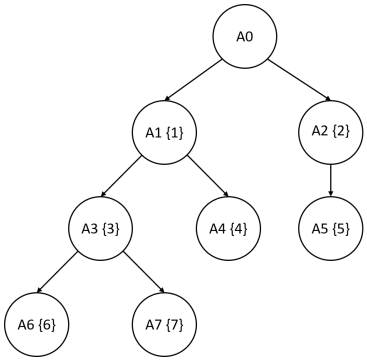


图 3-2 属性树

在本文提出属性树内，利用父子节点来构造属性继承关系，父节点包含子节点的属性，在图3-2所示属性树中，各节点包含的属性如表3.1所示。属性树的根节点 A0 为最大属性，叶子节点没有子属性，其本身属性即为其所有属性。节点 A4，A5，A6，A7 所拥有的属性分别为 4，5，6，7。节点 A3 自身所拥有的属性为 3，其子节点为 A6，A7，则 A3 继承了 A6，A7 所拥有的属性，A3 所拥有的属性为 {3，6，7}。同理，A2 自身属性为 2，其子节点为 A5，则 A2 所拥有的属性为 {2，5}。对于节点 A1，其继承了 A3，A4 的属性，而 A6，A7 又为 A3 的子节点，所以 A1 通过 A3 间接继承了 A6，A7 的属性。按这种属性的继承关系，则根节点 A0 拥有树中所有节点的属性，其拥有的属性集为 {1，2，3，4，5，6，7}。

在属性树中，由于属性间的继承关系有直接继承也有间接继承，无法从树中直接明了地获得两个属性间是否存在继承关系，因此，可以利用可达矩阵来进行表示，矩阵中的元素代表两个属性间是否存在继承关系。设 $T = \langle V, E \rangle$ 为一颗属性树，其有 N 个节点 $V = \{v_1, v_2, v_3, \cdots v_n\}$。该属性树 T 对应的可达矩阵 $P = (p_{ij})$ 中的元素定义为：

表 3.1 节点属性

节点	直接属性	有效属性
A0	ϕ	{1，2，3，4，5，6，7}
A1	{1}	{1，3，4，6，7}
A2	{2}	{2，5}
A3	{3}	{3，6，7}
A4	{4}	{4}
A5	{5}	{5}
A6	{6}	{6}
A7	{7}	{7}

$$p_{ij}=\begin{cases}1 & 若v_i与v_j之间存在通路\\ 0 & 否则\end{cases} \tag{3.6}$$

其中 v_i 到 v_j 之间存在通路代表 v_i 继承了 v_j 的属性，否则两个属性间没有继承关系。由定义可知，属性树对应的可达矩阵的阶数即为属性树中的节点个数 $|V|$，该可达矩阵通过描述两个节点间是否可达，既表示了两个属性间的直接继承关系，也表示了两个属性间的间接继承关系，表达十分简洁明了。

属性树的可达矩阵可以通过属性树的邻接矩阵得到，仍以上述属性树为例，邻接矩阵 $A=(a_{ij})$ 中的元素定义为：

$$a_{ij}=\begin{cases}1 & 若v_i与v_j邻接\\ 0 & 否则\end{cases} \tag{3.7}$$

其中 v_i 与 v_j 之间邻接指 v_i 与 v_j 直接相连，在属性树中，若 $a_{ij}=1$，代表 v_j 为 v_i 的子节点，即属性 v_i 直接继承了属性 v_j。由属性树得到其对应的可达矩阵的过程如算法3.5所示。

利用邻接矩阵 A 得到可达矩阵 P 的常用求法[56] 为首先得到 $B=(A+E)^n=E+A+A^2+\cdots+A^n$，再将 B 中的非零元素均置为 1，零元素不变即可得到 n 步可达矩阵 P，代表两个节点间经过 n 步是否可达。由以上定义可知，邻接矩阵与可达矩阵均为布尔矩阵，布尔矩阵指矩阵中的元素为 0 或者 1，布尔矩阵之间运算时其加法规则如表3.2所示。可以利用布尔矩阵之间的运算规则直接由邻接矩阵得到可达矩阵，同时，采用逐次平方的方式，最多 $(log_2{}^{(n-1)}+1)$ 步布尔平方运算即可由邻接矩阵得到可达矩阵，其中 n 代表矩阵的阶数。

以图3-2中的属性树为例，除去 $A0$ 节点，其对应的邻接矩阵为 A，由 A 得到 B，$B=(A+E)^{(2)}$，再对 B 做平方运算得到 C，$C=B^{(2)}$，矩阵 A，B，C 的值如式3.8所示，

算法 3.5 计算可达矩阵

输入： 属性树 T

输出： 可达矩阵 P

1: 对于 T 中的两个节点 v_1，v_2，如果 v_1 的子节点内存在 v_2，则令 $a_{ij}=1$，由此得到 T 对应的邻接矩阵 $A=(a_{ij})$。
2: 将邻接矩阵 A 与单位矩阵 E 相加得到矩阵 P，$P=A+E$。
3: 对矩阵 P 利用布尔运算法则得到新的 P，$P=P^{(2)}=P\times P$。
4: 重复步骤 3 直到得到的 P 不再改变。
5: 输出可达矩阵 P。

表 3.2 布尔和运算

操作数 1	操作数 2	结果
0	0	0
0	1	1
1	0	1
1	1	1

当得到 C 时，矩阵内的元素不再改变，则可达矩阵 P 即为矩阵 C。

$$\begin{bmatrix}0&0&1&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&1\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{bmatrix},\begin{bmatrix}1&0&1&1&0&1&1\\0&1&0&0&1&0&0\\0&0&1&0&0&1&1\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix},\begin{bmatrix}1&0&1&1&0&1&1\\0&1&0&0&1&0&0\\0&0&1&0&0&1&1\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}\tag{3.8}$$

算法 3.6 计算属性集

输入： 属性 v_i，可达矩阵 P

输出： v_i 继承到的所有属性 v_{all}

1: 对于 V 中的每一属性 V_j，查看 p_{ij} 是否为 1，若为 1，将该属性加入 v_{all}，$v_{all}=v_{all}\cup v_j$。
2: 返回 v_{all}。

从可达矩阵内可以很简洁明了的得到属性之间的继承关系，对于某一属性 v_i，计算其继承到的所有属性 v_{all} 的过程如算法3.6所示。可达矩阵表示了属性集 V 内任意两个

属性间的继承关系，如果 $v_{ij}=1$，代表属性 v_i 继承了属性 v_j，通过遍历 V 内的所有属性，即可获得 v_i 所继承的属性集。

属性树内的节点主要包含三项数据：节点对应的属性值，节点的父节点，由节点的子节点构成的数组。其数据格式可以表示为 {属性：$attr$，父节点：$parentAttr$，子节点：$childNodeList$}，其中 $childNodeList$ 内包含多个子节点。属性树涉及到的动态调整操作主要有以下几项：为属性树添加叶子节点，为属性树添加父节点，删除属性树中某一节点，在属性树两个属性间构造继承关系，解除属性树中两个属性间的继承关系，其具体实现方式如下：

- 为属性树添加叶子节点操作如算法3.7所示，在添加叶子节点时，需要指定要在哪一个属性节点之下添加；

算法 3.7 添加叶子节点

输入： 要添加的属性 $attr$，$attr$ 的父属性 $parentAttr$

输出： 空值

1: 查找属性树中是否存在节点，其属性值为 $parentAttr$，如果不存在，构造节点 $attrNode$，其属性值为 $attr$，以该节点为根节点构造属性树，算法结束。

2: 根据 $parentAttr$ 得到节点 $parentNode$，根据 $attr$ 构造节点 $attrNode$，将 $attrNode$ 添加到 $parentNode$ 的子节点数组中，$attrNode$ 中的父节点值为 $parentNode$。

- 为属性添加父节点的操作如算法3.8所示，除了在属性树中某一属性节点之下添加子属性节点，我们也需要在某一属性节点之上添加父属性节点，在添加父节点时，要指定该父节点在属性树中的子节点；

算法 3.8 添加父节点

输入： 要添加的属性 $attr$，$attr$ 的子属性 $childAttr$

输出： 空值或者 $\perp$

1: 查找属性树中是否存在节点，其属性值为 $childAttr$，如果不存在，算法结束，抛出错误信息 $\perp$。

2: 根据 $childAttr$ 得到子节点 $childNode$，根据 $attr$ 构造节点 $attrNode$，查看 $childNode$ 是否有父节点，若没有，将 $childNode$ 加入 $attrNode$ 的字节点数组中，$childNode$ 的父节点值为 $attrNode$，算法结束。

3: 查找到 $childNode$ 的父节点 $parentNode$，将 $childNode$ 从 $parentNode$ 的子节点数组中删除，将 $attrNode$ 加入 $parentNode$ 的子节点数组中，$attrNode$ 中的父节点值为 $parentNode$。

4: 将 $childNode$ 加入 $attrNode$ 的子节点数组中，$childNode$ 的父节点值为 $attrNode$。

- 删除属性树中的某一节点的操作如算法3.9所示，在属性树中删除的节点可能为叶子节点也可能为非叶子节点，若为叶子节点，直接删除，若为非叶子节点，将该节点删除后，将其子节点的父节点变为该节点的父节点；

算法 3.9 删除节点

输入: 要删除的属性 $attr$

输出: 空值或者 $\perp$

1: 查找属性树中是否存在节点，其属性值为 $attr$，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 根据 $attrNode$ 找到其父节点 $parentNode$，如果不存在，算法结束，抛出错误信息 $\perp$。
3: 将 $attrNode$ 从 $parentNode$ 的子节点数组中删除，如果 $attrNode$ 为非叶子节点，将该节点的子节点的父节点变为 $parentNode$，并将该子节点加入 $parentNode$ 的子节点数组中。

- 属性树两个属性间构造继承关系的操作如算法3.10所示；

算法 3.10 构造继承关系

输入: 父属性 $parentAttr$，子属性 $childAttr$

输出: 空值或者 $\perp$

1: 查找属性树中是否存在节点 $parentNode$，其属性值为 $parentAttr$。节点 $childNode$，其属性值为 $childAttr$。如果 $parentNode$ 或者 $childNode$ 不存在，算法结束，抛出错误信息 $\perp$。
2: 如果 $childNode$ 存在父节点 $oldPrentNode$，将 $childNode$ 从 $oldPrentNode$ 的子节点数组中删除。
3: 将 $childNode$ 加入 $parentNode$ 的子节点数组中，将 $childNode$ 的父节点值置为 $parentNode$。

- 解除属性树中两个属性间继承关系的操作如算法3.11所示。

通过属性树的动态操作函数，可以灵活的构造以及更改属性树结构，从而可以依据企业内部组织架构的变动灵活的实现权限继承。为了防止数据的泄露，需要对数据进行一定的加密，通过控制哪些数据使用者可以解密密文来实现权限的分配。这一过程通过 CP-ABE 算法实现，每一想要获得数据的数据使用者拥有不同的角色与属性，根据这些角色与属性，可以通过 CP-ABE 解密密钥生成算法生成相应的解密密钥。数据拥有者通过改变利用 CP-ABE 加密数据时的访问控制结构，可以控制拥有哪些属性的用户可以解密密文，实现权限的授予。因此，在实现权限继承时，也要在该加密过程进行一定的改

算法 3.11 删除继承关系

输入: 父属性 $parentAttr$，子属性 $childAttr$

输出: 空值或者 $\perp$

1: 查找属性树中是否存在节点 $parentNode$，其属性值为 $parentAttr$，节点 $childNode$，其属性值为 $childAttr$，如果 $parentNode$ 或者 $childNode$ 不存在，算法结束，抛出错误信息 $\perp$。

2: 查找 $parentNode$ 是否有父节点 $grantParentNode$，如果没有，算法结束，抛出错误信息

3: 将 $childNode$ 从 $parentNode$ 的子节点数组中删除，将 $childNode$ 加入 $grantParentNode$ 的子节点数组中，更改 $childNode$ 的父节点值为 $grantParentNode$。

进，将属性树内角色间的可达关系包含在加密时使用的访问控制结构内，实现权限的继承，使得权限管理更有层次化。

3.2.3 数据加解密

数据加密过程由数据拥有者完成,数据拥有者的数据分为两部分:利用对称加密算法加密数据得到的数据密文以及将该对称密钥利用 CP-ABE 加密后得到的密钥密文，数据生成过程如图3-3所示。数据拥有者根据实际应用场景将自身隐私数据按不同标准分块，得到数据集 $D=\{D_1,D_2,\cdots,D_n\}$。接着利用对称密钥集 $K=\{K_1,K_2,\cdots,K_n\}$ 分别进行加密，加密算法为 f，得到的密文集为 $C=\{C_1,C_2,\cdots,C_n\}$，则有 $C_i=f\left(D_i,K_i\right)$。针对隐私数据，为了保证存储在云数据库内的数据完整性，我们利用 HMAC 算法为数据集计算消息摘要集，并将该消息摘要集存储在区块链上，消息摘要集 $H=\{H_1,H_2,\cdots H_n\}$，其中 $H_i=HMAC\left(D_i,K_i\right)$。密文集由数据拥有者上传至数据库，同时，其将密钥集通过 CP-ABE 算法进行加密，将得到的密钥密文集 $CTK=\{CTK_1,CTK_2,\cdots,CTK_n\}$ 上传至区块链，其中 $CTK_i=CP-ABE(K_i)$。

数据拥有者为隐私数据块 D_i 添加数据块说明 $dataNote_i$，利用对称密钥 K_i 将 D_i 加密后上传至云数据库实现持久化存储，同时，利用 K_i 通过 HMAC 算法为数据块 D_i 生成哈希值 H_i，然后将 K_i 通过 CP-ABE 加密后生成密钥密文 CTK_i，数据拥有者将 $dataNote_i$，CTK_i，H_i 构成权限管理元数据上传至区块链，元数据的数据格式为 {数据块说明：$dataNote_i$，密钥密文：CTK_i，HMAC 值：H_i}。

在本章使用的 CP-ABE 中，在对数据进行加密时，除了需要主密钥 MK，还需要指定访问控制树。数据拥有者通过设计加密密钥时的访问控制树的结构实现权限的分配。只有属性集满足访问控制树的数据使用者才可以解密密钥密文从而获得对称密钥，进而解密存储在云数据库中的密文，获得数据访问权限。本章以形式化的语言来描述与访问控制树对应的访问策略，其表明了数据使用者若想解密密钥密文，其自身属性应满足什

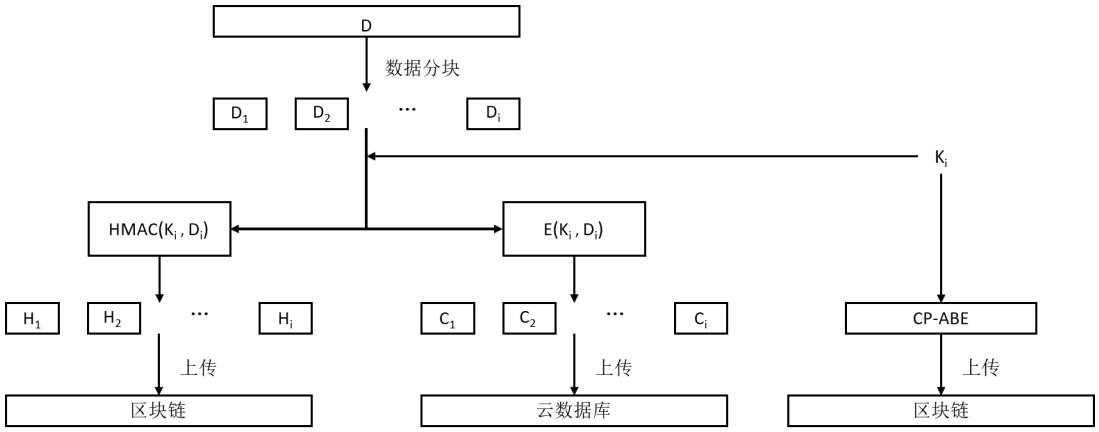


图 3-3 数据生成

么条件。该形式化语言所用到的符号说明如表3.3所示。

表 3.3 形式化语言符号说明

符号	说明
P	访问策略
T_i	门限值 i 的门限
A	属性
R	角色
RA	角色属性
$\wedge$	与
$\vee$	或

根据以上定义，访问策略 $P = R_1 \wedge T_2(A_1, A_2, A_3)$ 表示角色为 R_1 且至少拥有属性 A_1，A_2，A_3 中两个属性的数据使用者可以解密密文。当数据拥有者指定的访问策略内包含角色时，由于角色之间可能存在权限继承的关系，因此其需要参考本章3.2.2节内定义的属性树，计算所有继承该角色的角色集，这一过程可以通过属性树的可达矩阵完成。对于角色 R_j，继承该角色的角色集为 RH_j，如果角色 $R_i \in RH_j$，则在可达矩阵 $P = (p_{ij})$ 内，应有 $p_{ij} = 1$。数据拥有者在使用包含角色的策略进行访问控制时，应将该角色所涉及到的继承该角色的角色集以或的形式包含进去。对于上述策略 P，其应改为 $P = (R_1 \wedge T_2(A_1, A_2, A_3)) \vee T_1(RH_1)$。其中，$RH_1$ 为角色的集合。

更进一步的如果在访问策略内指定了角色 R_j 的同时指定了其角色属性 RA_j，则应查看 RH_j 内的每一角色 R_i 对应的角色属性集 RA_i，如果存在 $RA_{ij} \in RA_i$ 并且 $RA_{ij} = RA_j$，则应将 $R_i \wedge RA_j$ 与其他角色以或的逻辑加入访问策略。例如，如果初始定义的访问策略为 $P = (R_1 \wedge RA_1) \wedge T_2(A_1, A_2, A_3)$，其中 RA_1 为 R_1 的角色属性。数据使用者在加密时，查看属性树发现继承 R_1 的角色为 R_2 与 R_3，同时 R_2 也

有角色属性 RA_1，R_3 没有角色属性 RA_1，则数据拥用者会将原始的访问策略扩展为 $P = ((R_1 \wedge RA_1) \wedge T_2(A_1, A_2, A_3)) \vee T_1(R_2 \wedge RA_1, R_3)$。通过引入角色属性，实现了更加细粒度的权限继承。

数据解密过程由数据使用者完成，其首先从区块链上获取在数据加密过程完成后生成的与权限分配相关的元数据 $\{dataNote_i, CTK_i, H_i\}$，然后使用自己的解密密钥运行 CP-ABE 解密算法解密密钥密文 CTK_i 从而得到对称密钥 K_i。最后使用 K_i 解密存储在云数据库中的数据密文获得数据拥有者隐私数据，利用 K_i 计算 HMAC 值并与 $HMAC_i$ 比对，如果相等，说明该数据未被篡改，数据使用者获得正确的数据。由于数据拥有者在为角色为 R 的数据使用者授予访问权限时，已经将继承角色 R 的角色集 RH 包含在访问控制树内，除了角色为 R 且其他属性符合访问控制树的数据使用者可以解密密钥密文获得对称密钥从而访问数据拥有者隐私数据外，角色为 RH 内某一角色的数据使用者也均可以获得对称密钥进而获得访问隐私数据的权限，实现了权限的继承。更进一步的，如果 RH 内的角色 RH_i 的角色属性与 R 的角色属性有交集 RA，则角色为 RH_i 且角色属性为 RA 的数据使用者才可以继承角色 R 的权限，这符合企业组织内部门的概念，使得权限继承更加细粒度，可以实现更为精确的层次化权限管理。

3.3 本章小结

本章主要介绍了基于角色分配的 CP-ABE 加解密的过程，首先介绍了引入的角色的概念以及其可以如何代表数据使用者间层次化的权限关系。接着提出属性树的概念，介绍了如何使用多叉树的结构表示数据使用者角色间的权限继承以及针对属性树的若干操作。最后，介绍了 CP-ABE 的加解密过程，其中，在数据加密时，介绍了如何将属性树表示的角色间继承关系与访问控制树结合对数据进行加密，实现权限继承的功能，降低权限管理的复杂度。

第四章　基于区块链的权限分配模型

本章介绍了基于智能合约的权限分配模型，该模型通过判断数据使用者属性是否匹配数据拥有者制定的访问策略实现权限授予。整个模型包括区块链系统初始化，角色与属性分配交易，会话交易，链上数据存储，链上数据获取五个步骤，每个步骤均由参与各方通过调用智能合约发起交易完成。本章首先介绍了链上权限分配模型的整体架构，该架构主要包括五层：数据存储层，区块链服务层，访问控制层，链上代码层以及功能层，然后详细介绍了基于智能合约的权限分配流程，最后给出了模型安全性证明。

4.1　权限分配模型架构设计

4.1.1　整体架构描述

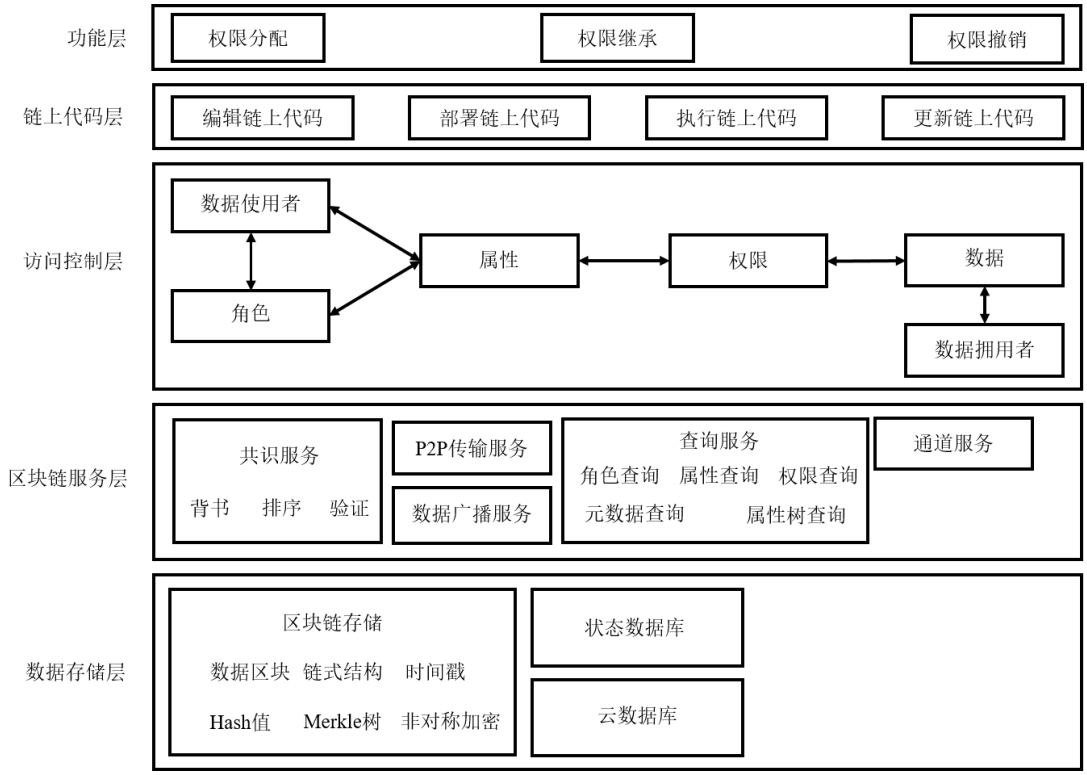


图 4-1 模型整体架构

本文提出的链上权限分配模型整体架构如图4-1所示。整个模型主要分为五层：数据存储层，区块链服务层，访问控制层，链上代码层以及功能层。其中数据存储层主要包括三部分：区块链存储，状态数据库存储以及云数据库存储。区块链服务层主要包括 P2P 传输服务和数据广播服务，共识的达成以及区块链支持的查询服务。访问控制层主

要实现链上权限分配，链上代码层主要包括了链上代码的编辑，部署，执行以及更新。最后，功能层表明了链上权限管理模型实现的功能，包括：权限分配，权限继承以及权限撤销。

4.1.2 数据存储

由于区块链上的存储容量有限以及如果一次交易包含的内容过多将会导致吞吐量的下降，因此，数据拥有者的隐私数据不宜直接存储在区块链上。在本章提出的链上权限分配模型中，数据拥有者的隐私数据经过对称加密后上传至云数据库，实现数据持久化，然后将该对称密钥经过 CP-ABE 加密后上传至区块链，减轻了链上的存储压力。本章提出的链上权限分配模型基于 Hyperledger Fabric 架构，而在该架构内，数据的存储主要包括两个方面：区块链以及状态数据库，二者一起构成了账本的概念，如图4-2所示。其中世界状态即代表了状态数据库，智能合约执行过程中涉及到的业务对象以键值对的形式存储在状态数据库内。

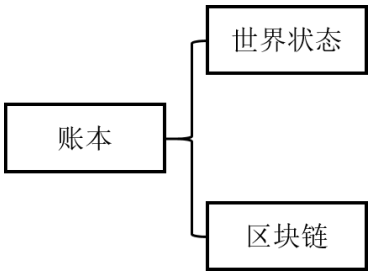


图 4-2 账本

区块链为将数据块以链的形式连接起来的数据结构，其内主要用来存储交易日志，这些日志记录了在链上发生的所有交易信息，交易信息一旦被写入块内，其将不可再被更改。在 Hyperledger Fabric 内，每一数据块内包含的信息如图4-3所示。

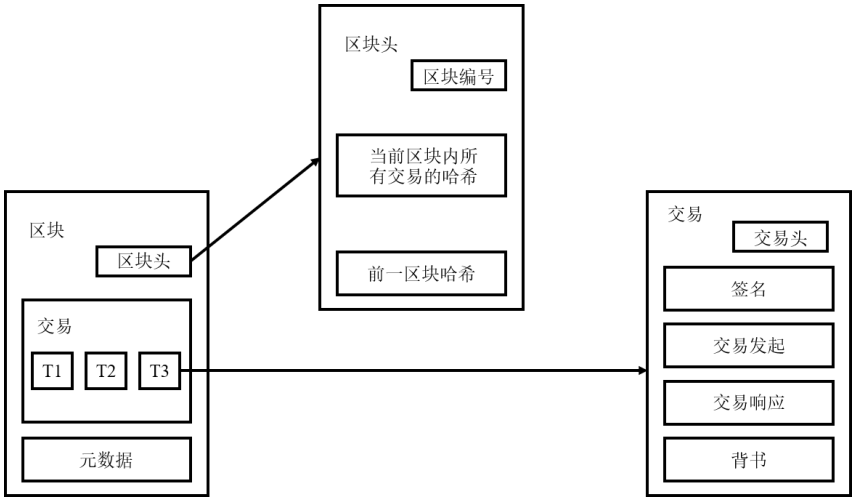


图 4-3 Hyperledger Fabric 区块内容

区块内主要包括区块头，区块数据以及元数据三部分内容，其中区块头内存有当前区块的号码，当前区块内所有交易的哈希以及前一区块的哈希。创世区块的编号为 0，每生成一个区块编号加 1，区块内的所有交易被组织为一颗默克尔树，在区块头内存储该树根节点内的哈希值。区块内的区块数据部分主要包括该区块内的交易信息，在每一交易内，存有交易头，签名，交易发起，交易响应以及背书。交易头内包括了该交易的元数据，签名用来验证交易未被篡改。交易发起内包含利用智能合约发起交易时指定的参数，交易响应为执行智能合约的输出值，如果交易合法，其将被用来修改状态数据库内的相关状态数据。交易是否合法需要判断该交易是否满足背书策略，背书为背书节点对该交易进行签名证明该交易的合法性。元数据主要内容为区块的创建时间。状态数据库以键值对的形式存储业务对象的当前状态，在链上发起交易将改变这一状态。状态数据库主要有两种 LevelDB 与 CouchDB。状态数据库使得使用者可以直接获得业务对象的当前状态，无需根据区块链上的交易信息进行计算。

4.1.3　区块链服务层

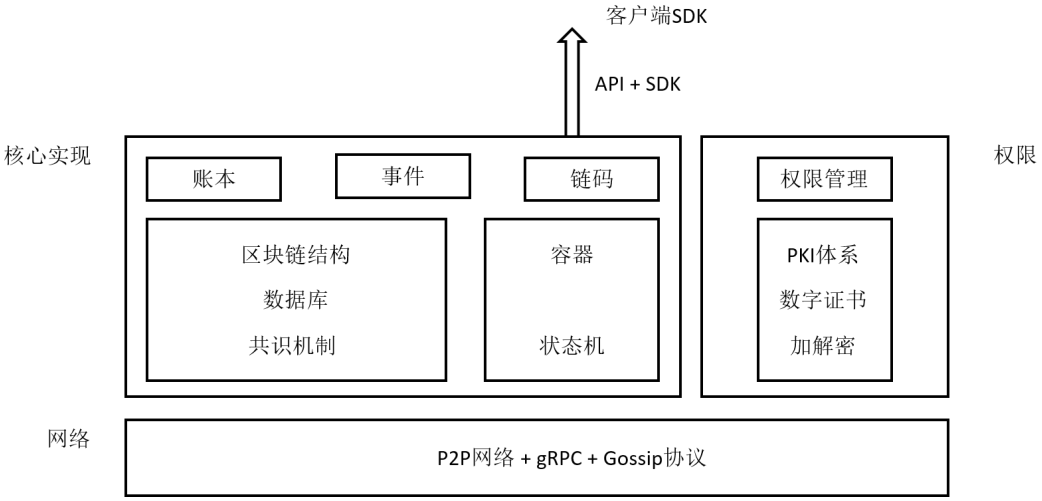


图 4-4 Hyperledger Fabric 整体架构

本章提出的链上权限分配模型基于 Hyperledger Fabric 架构，Hyperledger Fabric 是可以用来配置以及操作联盟链的模块化，可扩展的开源系统，属于许可链的范畴，是由 Linux 基金会主导的 Hyperledger 项目之一。Fabric 是第一个真正意义上的可以用来运行分布式应用的可扩展区块链系统，它支持模块化的共识协议，这个特点可以支持我们设计适合特定用途的系统和可信模型。Fabric 也是第一个可以用标准的，通用的程序设计语言编写分布式应用，并且不需要系统地依赖于原始的加密货币的区块链系统。这与现存的区块链平台形成了鲜明的对比，已有的一些区块链平台要求使用者使用特定领域的语言来编写智能合约，或者依赖于加密货币，在使用上具有一定的局限性，而 Hyperledger Fabric 解决了这一问题。

Hyperledger Fabric 的整体架构如图4-4所示，其主要由五个模块构成：网络模块，核心实现模块，权限模块以及客户端 SDK 模块：

- 网络模块：Hyperledger Fabric 运行在由对等节点组成的 p2p 网络之上，节点间交互使用到了 gRPC 技术，节点利用 Gossip 协议[57] 以一种可扩展的方式广播账本和通道数据。Gossip 消息是连续的，并且通道上的每一节点都不断地接收到多个对等节点发出的当前的一致的账本数据；

- 核心实现模块：该模块运行在网络模块之上，与网络模块一起，提供了区块链网络的主要功能，该模块最重要的就是账本和链码。其中，账本的概念在4.1.2介绍过，其包括区块链以及状态数据库。各个节点内保存的账本的一致性依赖于共识机制。链码实现业务逻辑，其实质等同于智能合约，执行结果导致交易的发生以及状态的改变，链码部署在容器内，其实现依赖于状态机，状态机模块还可以监听整个网络内发生的事件；

- 权限模块：Hyperledger Fabric 使用基础公钥设施 (PKI, Public Key Infrastructure) 产生密钥，分配给相应节点，实现权限控制；

- 客户端 SDK 模块：Hyperledger Fabric 提供了一系列 SDK 来帮助使用者进行链码开发，使用者可以使用 Go，Node.js 以及 Java 来进行开发。同时，Fabric 也提供了一系列 SDK 帮助使用者进行应用的开发，当前可支持的语言有 Node.js 以及 Java。

Hyperledger Fabric 属于联盟链，其可以实现在特定节点间构造联盟并在联盟内进行交易，交易信息不会被联盟外的其他节点获得。在 Hyperledger Fabric 中，这一功能通过通道服务实现，通过在联盟节点间创建通道，保证了链上节点间的秘密通信，通道外的节点无法获知通道内节点间的交易信息以及状态数据库的内容。

由于区块链为分布式系统，不存在中心节点居中调度使得网络上各个节点的状态相同，因此，需要在各个节点间达成一种共识来保证区块链网络上各个节点处于相同的状态。Hyperledger Fabric 中主要通过背书，排序以及验证三个步骤来保证各个节点处于一致状态，这一过程如图4-5所示。

首先，节点在区块链上发起交易，该交易被广播到区块链上的其他节点，其他节点将执行该交易并查看其对状态数据库的更新情况。这些节点在该步不会将该更新应用于状态数据库，这些节点会将其对该交易的响应返回给发起交易的节点，各节点执行交易并返回交易响应的过程被称为对交易背书。在 Hyperledger Fabric 内主要存在两种节点：对等节点与排序服务节点，对等节点主要处理各类交易，而排序服务节点可以控制区块的生成。在各节点对交易背书后，发起交易的节点将包含背书的交易发送给排序服务节点，排序服务节点从区块链网络中的各个节点接收交易，其按照一定的规则将这些交易排序然后打包成区块，这些区块最终会被上链。区块内交易的顺序不一定与排序服务节点接收到该交易的顺序相同，因为可能有多个排序节点同时接受到某一交易的可能，但

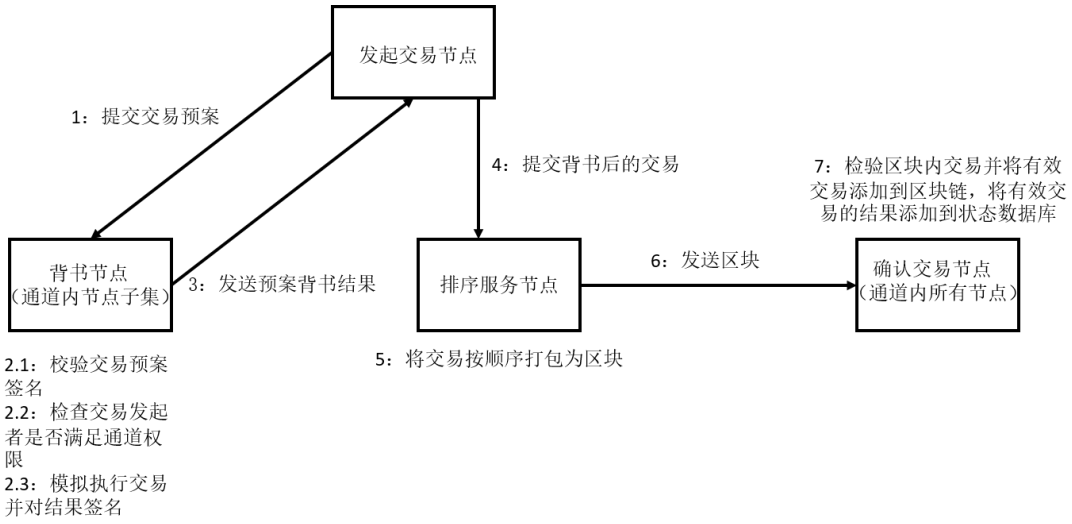


图 4-5 Hyperledger Fabric 共识过程

交易在区块内一定遵循严格的顺序结构，其他节点利用这种顺序结构验证以及执行交易，顺序执行交易保证了各节点的状态相同。区块打包完成后，排序服务节点将区块发送给与其相连的所有节点，未与排序服务节点相连的节点可以通过 gossip 协议从其他节点处获得区块。每一节点以分布式一致的形式验证区块内交易的有效性，交易是否有效由节点对该交易的背书是否满足背书策略确定。各节点将有效的交易加入区块链，并将交易执行结果应用于状态数据库。

4.1.4　访问控制层

访问控制层主要包括通过区块链实现权限分配的流程，本章提出的链上权限分配模型主要利用数据使用者的角色与属性实现权限的分配，而角色也属于属性的一种，因此，本质上来说是通过数据使用者的属性进行权限分配。在该模型中主要存在三个参与方：数据拥有者，数据使用者以及属性管理机构。数据拥有者将其数据经过对称密钥加密后上传至云数据库存储实现数据持久化，加密后的数据在云数据库中的存储是安全的，在没有对称密钥的情况下不可能解密。数据使用者可以根据自身信息向属性管理机构申请角色与属性，属性管理机构为其分配相应的角色与属性。数据使用者得到角色与属性后，可向数据拥有者申请解密密钥。数据拥有者根据数据使用者自身角色与属性生成数据使用者的解密密钥并通过区块链分配给数据使用者。数据拥有者可以通过加密对称密钥生成密钥密文并将该密钥密文上链实现权限分配。在加密对称密钥时使用第三章介绍的基于角色分配的扩展 CP-ABE 模型进行加密，其可在加密时使用的访问控制树内指定拥有哪些角色与属性的数据使用者可以解密密文，同时实现角色间权限的继承。拥有相应角色与属性的数据使用者可以读取区块链上的密钥密文解密后获得对称密钥进而可以访问数据，其他数据使用者由于无法解密密钥密文而无法获得对称密钥，其将无法访问数据，数据拥有者通过改变访问控制策略实现了权限的有效分配。

4.1.5 链上代码层

在 Hyperledger Fabric 内，除了智能合约的概念还有链码的概念，链码即链上代码，智能合约与链码这两个概念本质上相同。通常来说，智能合约内定义了控制业务对象生命周期的交易逻辑，这些业务对象以键值对的形式存储在 HyperLedger Fabric 的状态数据库内，然后智能合约被打包成链码部署在区块链上。可以将智能合约看做控制交易的核心逻辑设计，而链码控制智能合约如何被打包部署。多个智能合约可以定义在同一个链码内，当部署链码时，其内包含的所有智能合约均被部署，可以被应用程序调用。我们可以将智能合约看做与特定业务流程相关的应用于具体领域的程序，而链码是将一组相关的智能合约进行安装与初始化的一种技术容器。

智能合约编写完成后，需要通过链码打包后部署到区块链上，链码一般运行于独立于链上共识节点的安全 Docker 容器内，其通过应用程序发起的交易初始化和管理状态数据库。链码通常用来处理在区块链网络中已经被各成员达成共识的业务逻辑，从这一方面来说，其与智能合约的概念相似，当发起交易时，可以通过调用链码来更新与查询状态数据库，并且链码在得到许可后可以调用其他链码的功能。链码面向不同的用户提供不同的功能，面向开发者，其提供了一系列编写业务逻辑的接口，可以将其看做智能合约，面向使用者，其提供了各种与操作链码相关的命令，这些命令及其说明如表4.1所示。为了避免读者混淆，本章剩余部分统一使用智能合约的概念表示权限分配的业务逻辑。

表 4.1 链上代码层相关命令

命令	说明
peer chaincode install	打包链码并存储在对等节点上
peer chaincode instantiate	部署链码
peer chaincode invoke	调用链码
peer chaincode list	得到通道内已部署的链码或者节点内已安装的链码
peer chaincode package	打包链码
peer chaincode query	调用链码读取已达成共识的数据
peer chaincode signpackage	在打包好的链码上签名
peer chaincode upgrade	更新链码

4.1.6 权限管理功能实现

本文提出的链上权限管理模型主要实现的功能有：权限分配，权限继承，权限撤销。权限分配过程基于数据使用者的自身角色与属性实现。数据拥有者根据数据使用者的自

身角色与属性通过 CP-ABE 的解密密钥生成过程为其生成解密密钥，同时，数据拥有者将自身隐私数据对称加密后上传至云数据库，然后将该对称密钥经过扩展的 CP-ABE 模型加密后生成的密钥密文上传至区块链。数据拥有者可以通过改变加密对称密钥时的访问控制树的结构来授予不同数据使用者权限，不同的访问控制树代表了不同的访问控制策略，通过判断数据使用者角色与属性是否满足数据拥有者制定的访问控制策略实现了权限的分配。

权限继承过程通过属性树实现，该属性树在数据使用者的角色之间构造了一种继承关系，而角色的继承表示了这些角色的拥有者权限间的继承，数据拥有者在制定访问策略时需要依据属性树内角色间的继承关系制定，访问策略内包含了数据使用者间权限的继承关系。权限撤销通过时间戳与二叉树结合实现有效期一旦过期或者数据拥有者主动撤销时权限的自动撤销。在为每一位数据使用者生成解密密钥时均指定了其解密密钥的有效时间以及该解密密钥内包含的版本号信息，数据拥有者可以根据这两个属性控制数据使用者权限的撤销。

4.2　基于智能合约的权限分配流程

4.2.1　模型描述

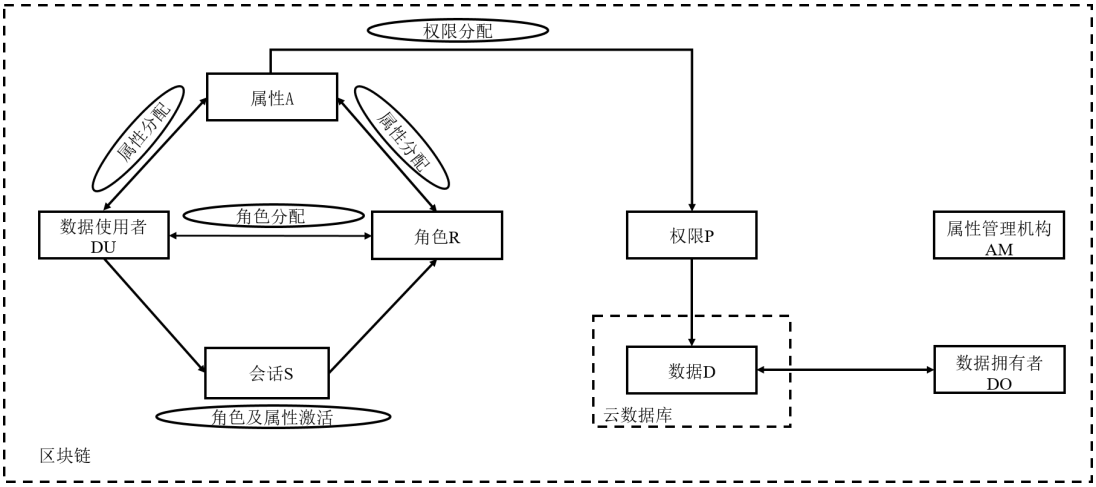


图 4-6 模型描述

整个模型的描述如图4-6所示，权限分配过程在区块链中完成，这一过程主要有三个参与方：数据拥有者，数据使用者，以及属性管理机构。其中，属性管理机构负责数据使用者角色以及属性的分配。模型主要由两部分构成：1. 七个实体和属性管理机构，区块链，云数据库，其中七个实体包括数据拥有者 (DO)，数据使用者 (DU)，数据集 (D)，角色 (R)，属性 (A)，权限 (P) 以及会话 (S)，其具体说明如表4.2所示。2. 六个实体间关系，主要包括数据使用者角色分配 (DURA)，数据使用者属性分配 (DUAA)，角色属性分配 (RAA)，属性权限分配 (APA)，两种会话，分别为 US 与 SR，六个关系的具体说明

表 4.2 模型实体

符号	实体及其含义	说明
DO	$DO=\{DO_1,DO_2,\cdots,DO_n\}$，$DO_i$ 或者 do 表示某一数据拥有者	数据拥有者集合
DU	$DU=\{DU_1,DU_2,\cdots,DU_n\}$，$DU_i$ 或者 du 表示某一数据使用者	数据使用者集合
D	$D=\{D_1,D_2,\cdots,D_n\}$，D_i 或者 d 表示某一数据块	数据集合
R	$R=\{R_1,R_2,\cdots,R_n\}$，R_i 或者 r 表示某一角色	角色集合
A	$A=\{A_1,A_2,\cdots,A_n\}$，A_i 或者 $attr$ 表示某一属性	属性集合
P	$P=\{P_1,P_2,\cdots,P_n\}$，P_i 或者 p 表示对某一数据块的访问权限	权限集合
S	$S=\{S_1,S_2,\cdots,S_n\}$，S_i 或者 s 表示某一会话	会话集合

如表4.3所示。

表 4.3 模型关系

符号	关系及其含义	说明
DURA	数据使用者角色分配 $DURA\subseteq DU\times R$	多对多的数据使用者和角色间的分配
DUAA	数据使用者属性分配 $DUAA\subseteq DU\times A$	多对多的数据使用者和属性间的分配
RAA	角色属性分配 $RAA\subseteq R\times A$	多对多的角色和属性间的分配
APA	属性权限分配 $APA\subseteq A\times P$	多对多的属性和权限间的分配
US	会话 $US\,(du:DU)\rightarrow S$	将数据使用者与会话相关联
SR	会话 $SR\,(s:S)\rightarrow 2^R$	将会话映射到角色集合

4.2.2 智能合约流程

在利用智能合约实现权限分配时，其流程如图4-7所示，整个流程主要有五步：区块链系统初始化，角色与属性分配交易，会话交易，链上数据存储以及链上数据获取。区块链系统初始化主要包括整个区块链网络的初始化和实体状态数据的初始化。AM，DO，DU 分别以不同的身份加入区块链网络，各个实体发起状态数据初始化交易，在状态数据库内生成初始状态数据。角色与属性分配交易主要包括数据使用者向属性管理机构请求角色与属性，属性管理机构根据该数据使用者的身份，证件等信息为其赋予不同的角

色与属性。会话交易主要包括角色的激活与解密密钥的获取，数据使用者可以同时拥有多个角色，其在会话交易中确定自己当前所属角色。角色激活由属性管理机构完成，激活角色的同时会将数据使用者的可用属性同时激活，并为其分配角色属性。数据使用者激活角色与属性后，其向数据拥有者请求解密密钥，数据拥有者根据其角色与属性为其生成解密密钥。链上数据存储由数据拥有者完成，其首先将隐私数据经过对称加密后上传至云数据库，然后将该对称密钥经过第三章介绍的 CP-ABE 扩展模型加密后上传至区块链。链上数据获取由数据使用者完成，其首先向数据拥有者请求数据，数据拥有者同意该请求后，数据使用者从链上得到密钥密文，用自己的解密密钥解密后得到对称密钥进而获得相关隐私数据的访问权限。

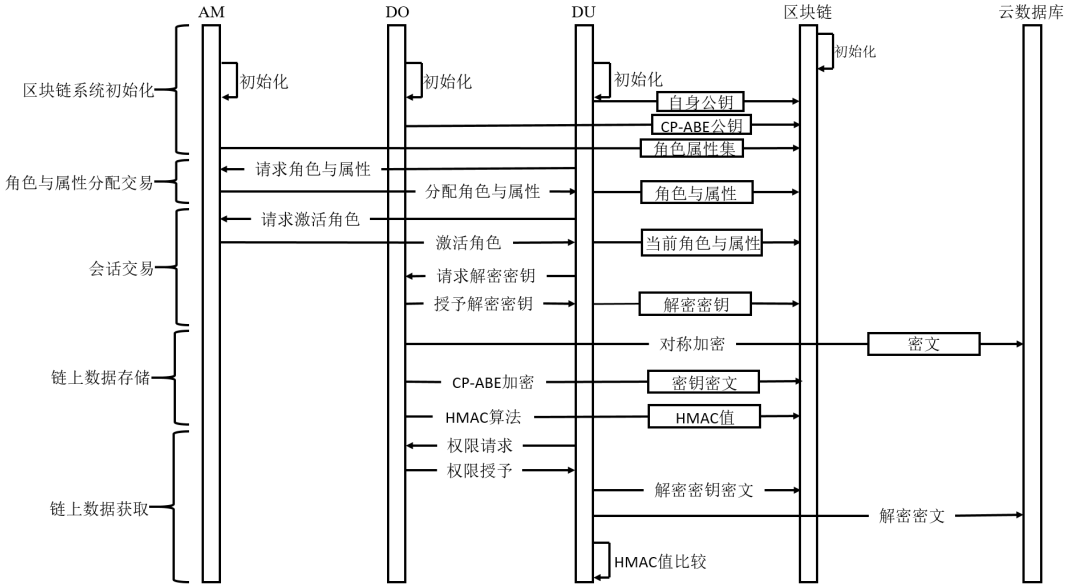


图 4-7 智能合约流程

4.2.3　区块链系统初始化

区块链网络基于 Hyperledger Fabric 构造，网络内主要存在三个组织：数据拥用者组织 R1，数据使用者组织 R2，属性管理机构组织 R3。其中，数据拥有者组织为数据拥有者的集合，数据使用者组织为数据使用者的集合，属性管理机构组织为属性管理机构的集合，三个组织构成的网络拓扑结构如图4-8所示。

R4 为初始化区块链网络的组织，其不参与 AM，DO，DU 之间的交易，R4 内有排序服务节点 O4。R4 通过 O4 根据网络配置文件 NC4 初始化网络，R1，R2，R3 构成了通道 C1，通道的配置文件为 CC1。P1，P2，P3 分别为组织 R1，R2，R3 内的节点，每一节点内均有相同的智能合约以及账本。客户端 A1，A2，A3 可以分别通过 P1，P2，P3 访问账本，调用智能合约，C1，C2，C3，C4 分别为四个组织的证书颁发机构。

区块链网络构造完成后，参与区块链网络的各个实体运行初始化算法4.1来初始化自身状态数据。我们为三个组织中的实体分别分配一个实体身份属性：以 DO 来表示数

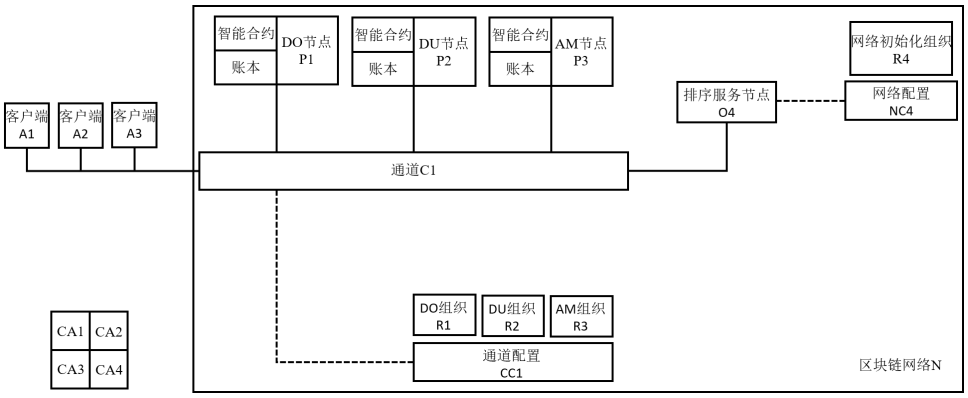


图 4-8 区块链网络

据拥有者，DU 表示数据使用者，AM 表示属性管理机构。

算法 4.1 初始化

输入： 用户 ID $userId$，用户身份属性 $identity$

输出： 空值或者 $\perp$

1: 根据 $userId$ 查看状态数据库内是否存在该用户的相关状态数据，如果存在，算法结束，抛出错误信息 $\perp$。

2: 如果该用户不存在，根据 $identity$ 构造初始状态数据：

(1) 如果 $identity$ 为 DO，构造状态数据项 {key: $userId$, value: {identity: DO, dataList: [], duList: [], PK: CP-ABE 公钥，sk: []}}。

(2) 如果 $identity$ 为 DU，构造状态数据项 {key: $userId$, value: {identity: DU, attrStateList: {}, roleStateList: {}, askUseRoleList: [], currentRoleList: [], currentAttrList: [], askAccessList: [], session: $false$, askForKey: {}, pk: 自身公钥}}。

(3) 如果 $identity$ 为 AM，构造状态数据项 {key: $userId$, value: {identity: AM, attrTree: $attrTree$, roleAttrList: $roleAttrList$}}。

3: 将初始状态数据存入状态数据库。

根据实体在初始化算法中输入的初始化参数的不同，其在状态数据库中存储的状态数据的数据格式不同。传入的初始化参数为 DO，DU 以及 AM 这三种的一种，代表实体想要以何种身份加入区块链网络，分别以这三种参数初始化实体在状态数据库中的状态数据。

我们以键值对的形式表示实体当前状态，不同实体的键均以其 ID 来表示，在初始化时以参数形式输入，不同实体的数据格式略有不同，下面分别进行说明：

- DO：数据使用者列表 ($duList$)，数据集 ($dataList$)，CP-ABE 公钥 (PK)，已分发解密密钥钥列表 (sk)。初始化时，数据拥有者运行 CP-ABE 初始化算法生成公钥

PK 并将其存入区块链，其余均置空。其在状态数据库内存储的状态数据的格式为 {key: $userId$, value: {identity: DO, dataList: [], duList: [], PK: CP-ABE 公钥，sk: []}}；

- DU：角色状态列表 ($roleStateList$)，属性状态列表 ($attrStateList$)，请求激活角色集 ($askUseRoleList$)，请求访问数据列表 ($askAccessList$)，当前激活角色集 ($currentRoleList$)，当前激活属性集 ($currentAttrList$)，当前会话状态 ($session$)，密钥请求 ($askForKey$)，自身公钥 (pk)。初始化时，当前会话状态为 $false$，代表数据使用者未处于会话中，数据使用者运行非对称加密算法产生自身公钥 pk 并将其存入区块链，其余项均置空。其在状态数据库内存储的状态数据的格式为 {key: $userId$, value: {identity: DU, attrStateList: {}, roleStateList: {}, askUseRoleList: [], currentRoleList: [], currentAttrList: [], askAccessList: [], session: $false$, askForKey: {}, pk: 自身公钥}}；
- AM：角色属性集 ($roleAttrList$)，属性树 ($attrTree$)。初始化时，AM 根据当前具体使用场景，向角色属性集内添加角色及该角色对应的属性集，属性树根据当前应用场景内的层次化权限关系构造而成。其在状态数据库内存储的状态数据的格式为 {key: $userId$, value: {identity: AM, attrTree: $attrTree$, roleAttrList: $roleAttrList$}}。

4.2.4　角色与属性分配交易

初始化阶段完成后，数据使用者可向属性管理机构请求角色，数据使用者首先要设定其状态数据中的角色状态列表 $roleStateList$，$roleStateList$ 在状态数据库内存储的数据格式为：{$role_1$: $roleState_1$, $role_2$: $roleState_2$, ... , $role_n$: $roleState_n$}。其中每一项代表当前与数据使用者有关的角色以及该角色的状态，以键值对 {角色：角色状态} 的形式来表示，我们规定角色状态取值有以下三种：

- NA：初始态，为空，角色状态项初始化时，其中每一项角色的状态均为空；
- $REQUEST$：请求态，代表数据使用者正在向属性管理机构申请该角色；
- $ACTIVE$：活跃态，代表数据使用者当前拥有该角色。

数据使用者请求角色的过程如算法4.2所示，其中请求角色集内的每一项为一个角色。数据使用者将想要请求的角色状态置为 $REQUEST$，数据使用者与角色的关系是多对多的，即一个数据使用者可以有多个角色，一个角色可以被多个数据使用者拥有。我们以 $DURA$ 表示数据使用者角色分配集合，以 DU 表示数据使用者集合，以 R 表示角色集合，则这三者的关系为 $DURA \subseteq DU \times R$。

当数据使用者的角色请求状态写入状态数据库后，其是否能获得相应角色需要属性管理机构根据数据使用者的身份以及相关证件信息进行判断，并且角色与属性的授予需

算法 4.2 请求角色

输入： 数据使用者 ID $duId$，请求角色集 $requestRoleList$

输出： 空值或者 $\perp$

1: 根据 $duId$ 查看状态数据库内是否存在该 DU 的相关状态数据，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 根据 $duId$ 得到该数据使用者的状态数据，从该状态数据内得到数据使用者当前角色状态列表 $roleStateList$。
3: 对于 $requestRoleList$ 内的每一项 $requestRole$，构造 $\{requestRole: REQUEST\}$ 并将其存入 $roleStateList$，代表当前对于角色 $requestRole$ 处于请求态。
4: 将 $roleStateList$ 存入数据使用者状态数据中。
5: 将数据使用者状态数据存入状态数据库。

要与数据拥有者达成共识。角色分配交易由属性管理机构发起，其过程如算法4.3所示，其中分配角色集内的每一项为将要分配给数据使用者的角色。

算法 4.3 分配角色

输入： 数据使用者 ID $duId$，分配角色集 $assignRoleList$

输出： 空值或者 $\perp$

1: 根据 $duId$ 查看状态数据库内是否存在该数据使用者的相关状态数据，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 根据 $duId$ 得到该数据使用者的状态数据，从该状态数据内得到数据使用者当前角色状态列表 $roleStateList$。
3: 对于 $assignRoleList$ 内的每一项 $assignRole$，从 $roleStateList$ 中得到其对应的角色状态 $currentState$，将该状态值置为 $ACTIVE$。
4: 将 $roleStateList$ 存入数据使用者状态数据中。
5: 将数据使用者状态数据存入状态数据库。

属性管理机构将数据使用者的角色状态项中对应状态的取值设为 $ACTIVE$。如果属性管理机构未同意将相应角色分配给数据使用者，其将不发起该交易，如果属性管理机构决定分配相应角色，但数据拥有者不同意，该交易将不能达成，数据使用者无法获得相应角色, 该角色仍将处于请求态。

当数据使用者得到角色集后，其继续向属性管理机构请求属性集，流程同请求角色集的流程相同。数据使用者与属性的关系也是多对多的，我们以 $DUAA$ 表示数据使用者属性分配关系，以 DU 表示数据使用者集合，以 A 表示属性集合，则这三者的关系为 $DUAA \subseteq DU \times A$。

数据使用者的状态数据内有属性状态列表，属性状态列表的数据格式同角色状态列表相同，属性状态同样有 NA，$REQUEST$，$ACTIVE$ 三种取值。数据使用者属性请

求交易的算法与数据使用者角色请求交易算法相同，在此不再赘述。

当数据使用者属性请求交易完成后，由属性管理机构发起数据使用者属性分配交易，数据使用者属性分配交易的算法同数据使用者角色分配交易的算法相同，在此不再赘述，其同样需要得到属性管理机构和数据拥有者的共识，该交易才可通过。数据使用者的角色以及属性分配交易完成后，数据使用者获得了角色集以及属性集。

4.2.5　会话交易

在角色与属性分配交易一节中，我们知道数据使用者可以拥有多个角色，数据使用者激活其拥有的角色集的子集即为建立一次会话。会话建立后，数据使用者之后将凭借其激活的角色请求数据访问权限。

一次会话在我们的系统中就是一次数据使用者与属性管理机构的交易，我们以 $session$ 表示数据使用者当前是否处于会话中，如果 $session$ 为 $true$，代表该数据使用者当前处于会话中。数据使用者请求激活角色时，要指定请求激活角色子集 $askUseRoleList$，该集合的每一元素为一个角色，数据使用者请求激活角色的交易如算法4.4所示。

算法 4.4 请求激活角色

输入： 数据使用者 ID $duId$，请求激活角色子集 $askUseRoleList$

输出： 空值或者 $\perp$

1: 根据 $duId$ 查看状态数据库内是否存在该数据使用者的相关状态数据，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 根据 $duId$ 得到当前数据使用者的状态数据。
3: 将 $askUseRoleList$ 内的角色存入数据使用者状态数据中的 $askUseRoleList$ 内并将 $session$ 置为 $true$。
4: 将数据使用者状态数据存入状态数据库。

数据使用者请求激活角色交易完成后，由属性管理机构发起角色激活交易，激活后的角色加入激活角色集 $currentRoleList$。规定当数据使用者发起角色激活交易时，其原有已激活角色将变为非激活的状态，因此要先清空 $currentRoleList$ 内原有的角色。同时，当激活角色时，会将角色属性与数据使用者自身属性一起构成数据使用者当前使用的属性集 $currentAttrList$，$currentAttrList$ 内之前存储的属性集也要被清空，角色激活交易如算法4.5所示。

对于请求激活角色子集内的每一角色，需要判断其在数据使用者角色状态列表内的状态，只有当其处于 $ACTIVE$ 状态时，才可将角色激活并将激活角色加入数据使用者 $currentRoleList$ 数据项内。角色属性集 $roleAttrList$ 由 AM 维护，其中每一项为角色和属性集的一组键值对，其在 AM 的状态数据内的存储形式为 $roleAttrList$: $\{R_1$: $[RA_{11}$, RA_{12}, ...], R_2: $[RA_{21}$, RA_{22}, ...], R_3: $[RA_{31}$, RA_{32}, ...], ...$\}$。当属性管理机构激活数据使用者的角色时，会根据数据使用者当前状态为其分配相应的角色属性 RA 并将该 RA 加入

算法 4.5 激活角色

输入： 数据使用者 ID $duId$，角色属性集 RA

输出： 空值或者 $\perp$

1: 根据 $duId$ 查看状态数据库内是否存在该数据使用者的相关状态数据 $duStateData$，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 从 $duSateData$ 内得到角色状态列表 $roleStateList$，请求激活角色集 $askUseRoleList$，当前角色集 $currentRoleList$，当前属性集 $currentAttrList$，将 $currentRoleList$ 和 $currentAttrList$ 内的角色与属性分别清空。
3: 对于 $askUseRoleList$ 中的每一项，查看其在 $roleStateList$ 内对应的状态，如果为 $ACTIVE$，将该角色加入当前角色集 $currentRoleList$。
4: 将数据使用者属性状态列表 $attrStateList$ 中状态为 $ACTIVE$ 的属性以及角色属性集 RA 内的属性加入 $currentAttrList$ 中。
5: 将 $currentRoleList$ 和 $currentAttrList$ 存入数据使用者状态数据并将 $session$ 置为 $false$。
6: 将数据使用者状态数据存入状态数据库。

数据使用者当前属性集，数据使用者属性状态列表内状态为 $ACTIVE$ 的状态也会被加入当前属性集。

当数据使用者获得相应的角色与属性后，其可以发起请求解密密钥交易。其构造密钥请求项 $askForKey$，其包括数据拥有者 ID 以及当前密钥请求状态 $askForKeyState$，该状态项可取以下值：

- NA：代表初始状态，当数据使用者未请求新解密密钥时的取值；
- ASK：表示数据使用者当前正在请求新的解密密钥；
- $ACCEPT$：代表数据使用者可以获得新的解密密钥。

算法 4.6 密钥请求

输入： 数据使用者 ID $duId$，数据拥有者 ID $doId$

输出： 空值或者 $\perp$

1: 根据 duId 查看状态数据库内是否存在该数据使用者的相关状态数据 $stateData$，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 从 $stateData$ 内取出 $askForKey$ 项，构造数据 {doId：$doId$，askForKeyState：ASK}，将其赋值给 $askForKey$。
3: 将 $stateData$ 存入状态数据库。

数据使用者向数据拥有者请求密钥的交易过程如算法4.6所示。该交易完成后，数据拥有者会发起解密密钥授予交易，其根据数据使用者当前属性与角色生成解密密钥，

然后将该解密密钥利用数据使用者的自身公钥加密后存放在已分发解密密钥列表 sk 内，sk 的数据格式为 sk：[数据使用者 ID：$duId$，解密密钥密文：$duSk$，时间戳：$time$]。其中 $duSk$ 为利用数据使用者自身公钥加密分配给其的解密密钥后生成的解密密钥密文，数据使用者可以读取该列表并用自身私钥解密 $duSk$ 得到相应的解密密钥。同时，每一数据使用者的解密密钥均有一有效时间 $time$，该时间作为生成解密密钥的一个属性存在。

4.2.6　链上数据存储

数据拥有者将自身隐私数据经过对称加密后上传至云数据库，然后将该对称密钥经过 CP-ABE 扩展模型加密后上传至区块链。根据 CP-ABE 扩展模型的定义，数据拥有者在向角色为 $R1$ 的数据使用者授予权限，假定访问策略为 $P1$。其首先在属性管理机构维护的属性树内查找继承角色 $R1$ 的角色集 RR。如果数据拥有者在访问控制策略内未指定 $R1$ 的角色属性，则将 RR 内的角色以或的形式直接加在 $P1$ 上，即新的访问策略为 $P2 = P1 \vee T_1(RR)$，其中 T_1 为门限值为 1 的门限，RR 为一角色集。如果在访问策略内指定了 $R1$ 的角色属性 $RA1$，则查看 RR 内是否存在角色，其角色属性内存在 $RA1$。对于存在 $RA1$ 的角色将该角色与 $RA1$ 相与后再与其他角色一起放入门限值为 1 的门限内。根据以上所述的 CP-ABE 扩展模型的加密方式，在加密阶段即实现了权限的继承，该权限继承关系与属性树内构造的角色间的继承关系相吻合。利用 CP-ABE 扩展模型对对称密钥进行加密后生成密钥密文 CTK，数据拥有者将自身隐私数据的 HMAC 值，元数据说明以及 CTK 上传至区块链，这一过程如算法4.7所示。

算法 4.7 元数据上传

输入： 数据拥用者 ID $doId$，元数据说明 $dataNote$，密钥密文 CTK_i，HMAC 值 H_i

输出： 空值或者 $\perp$

1: 根据 $doId$ 查看状态数据库内是否存在该数据拥用者的相关状态数据，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 根据 $doId$ 得到当前数据拥用者的状态数据，从数据拥用者状态数据内取出数据集 $dataList$。
3: 将 CTK_i 和 H_i 构造为控制访问的元数据 $metaData$，为该元数据添加元数据说明 $dataNote_i$，$metaData$ 的数据格式为：{元数据说明：$dataNote_i$，密钥密文：CTK_i，HMAC 值：H_i}，将该元数据添加到 $dataList$。
4: 将 $dataList$ 写回数据拥用者状态数据。
5: 将数据拥用者状态数据存入状态数据库。

数据使用者得到元数据后，如果其已授权，可以通过该元数据得到隐私数据。元数据说明表明了利用该元数据可以得到哪一部分隐私数据。数据拥有者完成链上数据存储后，其在状态数据库内的数据格式如图4-9所示

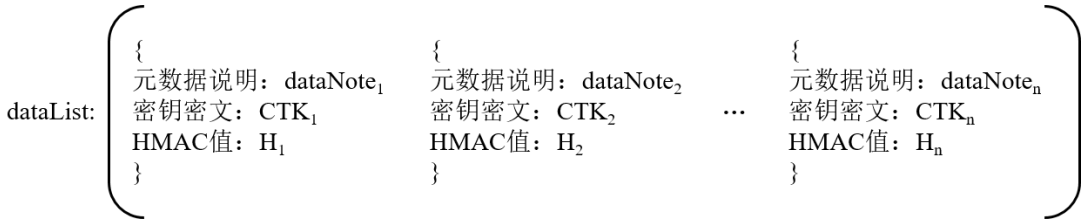


图 4-9 元数据集

4.2.7 链上数据获取

数据使用者在自身状态数据内以请求访问列表 $askAccessList$ 来表示其想要访问的数据集。$askAccessList$ 内的每一项存储的信息主要有三部分：1. 所访问数据的数据拥有者 ID($doId$)，2. 请求访问数据集 ($askDataList$)，3. 当前访问状态 ($currentState$)。其中当前访问状态可取值为 $REQUEST$，$AGREE$ 以及 $REVOKE$，$REQUEST$ 代表当前数据使用者在请求数据访问阶段，$AGREE$ 代表当前数据拥有者已同意数据使用者的访问请求，$REVOKE$ 代表当前数据使用者的访问权限已被撤销。数据请求交易由数据使用者发起，该交易过程如算法4.8所示。

算法 4.8 数据请求

输入： 数据使用者 ID $duId$，请求数据集 $askDataList$

输出： 空值或者 $\perp$

1: 根据 $duId$ 查看状态数据库内是否存在该数据使用者的相关状态数据 $stateData$，如果不存在，算法结束，抛出错误信息 $\perp$。
2: 从 $stateData$ 内取出请求访问列表 $askAccessList$，构造数据 {doId：$doId$，askDataList:$askDataList$，currentState:$REQUEST$}，将其存入 $askAccessList$。
3: 将 $askAccessList$ 存入 $stateData$。
4: 将 $stateData$ 存入状态数据库。

该交易完成后，数据拥有者根据数据使用者的当前属性是否符合相应密文的访问控制结构决定其是否有权限访问数据，如果其有权限，数据拥有者将发起权限授予交易。该交易将数据使用者的 ID，可访问数据集加入自身维护的数据使用者列表 $duList$。$duList$ 内的每一项与一数据使用者相对应，包含的内容有数据使用者 ID $duId$，数据使用者可访问的数据集 $dataList$，数据使用者当前的访问状态 $accessState$，其中 $accessState$ 有两个取值：1.$ACCEPT$，代表该数据使用者可以访问数据。2.$REVOKE$，代表该数据使用者访问权限已被撤销。授予权限时，$accessState$ 取值为 $ACCEPT$，同时，将数据使用者状态数据内的请求访问列表 $askAccessList$ 中的包含在 $dataList$ 内的数据的可访问状态置为 $AGREE$，权限授予交易如算法4.9所示。

数据使用者得知自己拥有相应的权限后，其利用自身解密密钥解密链上元数据中的密钥密文得到对称密钥，然后利用该对称密钥解密数据拥有者存储在云数据库中的密文

算法 4.9 权限授予

输入： 数据拥用者 ID $doId$，数据使用者 ID $duId$, 可访问数据集 $dataList$

输出： 空值或者 $\perp$

1: 根据 $doId$ 查看状态数据库内是否存在该数据拥有者的相关状态数据 $doStateData$，同理，根据 $duId$ 查看 $duStateData$，如果有一个不存在，算法结束，抛出错误信息 $\perp$。
2: 从 $doStateData$ 内取出数据使用者列表 $duList$，构造数据 {duId：$duId$，dataList：$dataList$，accessState：$ACCEPT$}，将其存入 $duList$。
3: 从 $duStateData$ 内取出请求访问列表 $askAccessList$，将 $askAccessList$ 内存在于 $dataList$ 内的数据集的访问状态置为 $AGREE$。
4: 将 $doStateData$ 和 $duStateData$ 分别存入数据拥有者和数据使用者的状态数据。
5: 将状态数据存入状态数据库。

获得其隐私数据同时利用 $HMAC$ 值验证数据的完整性，数据使用者数据访问权限分配成功。

4.3　基于状态机模型的安全性形式化证明

对于本章提出的链上权限分配模型的安全性证明主要包括两个方面[58]：

- 简单安全性，也可以称为可达性，主要考察是否存在未授权的数据使用者能够访问指定数据，如果不存在，即未授权数据使用者无法访问指定数据，则该模型满足简单安全性；
- 简单可用性，主要考察在每一可达状态下，已授权数据使用者是否能正确访问指定数据。如果能，则表明该模型是安全的，能够完成正确的权限分配过程，不会出现已授权数据使用者无法正确访问数据的情况，表明了该模型的可用性。

本文采用状态机模型[59] 来证明链上权限分配模型的安全性。状态机模型即有限状态机[60]，主要研究系统在特定时刻的状态以及状态间的变换过程，其安全性通过系统在安全状态下的规则来说明。状态机模型由各个状态下的状态变量以及状态间的转换函数构成，状态变量的值描述了系统当前状态，而状态转换函数描述了系统是如何变化的。利用状态机模型来证明本章提出的链上权限分配模型的安全性首先要证明该模型的初始状态是安全的，接着证明状态间的各个转移函数也是安全的，如果模型在这两种状况下均为安全的，则可以知道，该模型在任意状态下执行任意状态转换函数均为安全的，一个简单的状态机模型如图4-10所示。

在证明链上权限分配模型安全性之前，首先定义其状态转换系统 $\Sigma = (\Theta, \Psi, s_0, S)$，该状态转换系统为一四元组，其中：

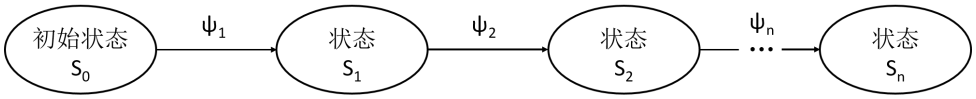


图 4-10 有限状态机

- S 为链上权限分配模型的状态集合；
- $s_0 \in S$ 为该模型的初始状态；
- Θ 为操作事件的集合，其中 $\theta \in \Theta$；
- Ψ 为系统的状态转移函数，$\Psi : \Theta \times S \to S'$，$\Psi_i(\theta, s) = s'$，表示给定状态 s 以及操作事件 θ，通过 Ψ_i 可以得到下一个状态 s'，Ψ 即对应于权限分配的各个流程。

$s \in S$ 表示系统 Σ 某一时刻各个参与实体及实体间关系的状态值集合，令 $s = (L_1, L_2)$,$L_1 = (DO, DU, D, R, A, P)$ 代表模型各个参与实体的状态,$L_2 = (DURA, DUAA, RAA, APA, SR)$ 代表模型实体间各个关系的状态。在初始状态时，L_1，L_2 均为空集 $\varnothing$，如果状态 s 是安全的，则其应满足以下条件：

$$\nexists(du, d)((du, d) \in s \wedge p(du) \notin P) \to access(du) = d \tag{4.1}$$

$$\forall(du, d)((du, d) \in s \wedge p(du) \in P) \to access(du) = d \tag{4.2}$$

其中，$access(du) = d$ 表示数据使用者 du 可以访问数据块 d，式4.1表示了简单安全问题，即不存在任何一个未授权数据使用者可以访问特定数据。式4.2表示了简单可用性问题，即任意已授权数据使用者均可访问特定数据。当状态 s 同时满足式4.1以及式4.2时，证明该状态是安全的，以 $safe(s)$ 表示。定义系统的状态转移的安全性如式4.3所示，当状态转移函数满足该式时，则该状态转移函数为安全保持的。

$$\begin{aligned}&(\forall\psi)(\forall\theta)(\forall s)((\psi \in \Psi) \wedge (\theta \in \Theta) \wedge (s \in S)\\ &\wedge (\psi(\theta, s) = s') \wedge (safe(s))) \to safe(s')\end{aligned} \tag{4.3}$$

模型的安全性依赖于其对应的状态转换系统的安全性，当其状态转换系统的初始状态 s_0 为安全的，并且状态转移函数均为安全保持的，则该状态转换系统 Σ 就是安全的，即该模型为安全的。

在初始状态时，$s_0 = (L_1 = \varnothing, L_2 = \varnothing)$，显然，该状态满足式4.1及式4.2，在该状态时，系统 Σ 是安全的。链上权限分配模型中权限分配不成功的情况主要有以下四种情况：

- 数据使用者角色分配不成功，$(du, r) \notin DURA$；
- 数据使用者属性分配不成功，$(du, attr) \notin DUAA$；

- 角色属性分配不成功，$(r, attr) \notin RAA$；
- 属性权限分配不成功，$(p, attr) \notin APA$。

则上式4.1可以展开为式4.4：

$$\begin{aligned} \nexists(du,d)((du,d) \in s \wedge ((du,r) \notin DURA \vee (du,attr) \notin DUAA \\ \vee (r,attr) \notin RAA \vee (p,attr) \notin APA) \rightarrow access(du) = d \end{aligned} \tag{4.4}$$

数据使用者访问权限分配成功即为上述分配过程均成功，则4.2可以展开为4.5：

$$\begin{aligned} \forall(du,d)((du,d) \in s \wedge ((du,r) \in DURA \wedge (du,attr) \in DUAA \\ \wedge (r,attr) \in RAA \wedge (p,attr) \in APA) \rightarrow access(du) = d \end{aligned} \tag{4.5}$$

假设当前状态 s 为安全的，根据式4.3可知，若经过状态转换函数之后得到的下一个状态 s' 也为安全的，则该状态转换函数为安全保持的。接下来，分别证明链上权限分配过程的各个状态转换函数是安全保持的。区块链系统初始化后，状态改变为：

- 添加数据拥有者，$DO' = DO \cup \{do\}$；
- 添加数据使用者，$DU' = DU \cup \{du\}$；
- 添加属性管理机构 AM；
- 添加角色，$R' = R \cup \{r\}$；
- 为角色添加属性集，$RAA' = RAA \cup \{(r, attr)\}$。

区块链系统初始化改变的状态，均不涉及数据使用者属性的分配以及属性权限的分配，满足安全保持的定义。角色与属性分配交易涉及到的状态改变为：

- 为数据使用者分配角色：$DURA' = DURA \cup \{(du, r)\}$；
- 为数据使用者分配属性：$DUAA' = DUAA \cup \{(du, attr)\}$。

分配角色与属性后，并未激活角色与属性，数据使用者无法通过角色与属性生成解密密钥，无法获得访问权限。同时为数据使用者分配角色与属性并不影响其他数据使用者的权限，因此，该转态转换函数为安全保持的。

会话交易将激活数据使用者的角色与属性，由数据使用者的角色与属性组成数据使用者当前属性。进行完会话交易的数据使用者满足 $(du, r) \in DURA \wedge (du, attr) \in DUAA \wedge (r, attr) \in RAA$。数据拥有者会根据数据使用者的角色与属性通过 CP-ABE 解密密钥生成算法生成相应的解密密钥，数据拥有者利用数据使用者的自身公钥加密该解密密钥生成密文，只有拥有相应私钥的数据使用者才可以解密该密文获得其解密密钥，

不会造成解密密钥的泄露。数据使用者若想获得相关隐私数据的权限，其首先要访问数据拥有者存储在链上的数据，因此该步是否安全保持与链上数据存储是否安全保持相关联。

链上数据存储涉及到的状态改变为数据拥有者将自身隐私数据对称加密后上传至云数据库，同时将控制隐私数据访问权限的元数据存储到链上，该元数据包括将对称密钥经过 CP-ABE 加密后的密文，该状态转换过程完成后，$D' = D \cup d$。对称加密为安全的，数据使用者在无法获得解密密钥的情况下不可能解密密文，而该解密密钥又经过数据拥有者通过 CP-ABE 加密后生成密钥密文上传至区块链。数据使用者若想获得数据拥有者的隐私数据，其首先要能够解密链上的密钥密文。数据使用者是否能解密该密钥密文由数据拥有者在通过 CP-ABE 加密该密钥时使用的访问控制结构所决定，当数据使用者的当前属性满足该访问控制结构时，其可以正常解密。当数据拥有者为拥有相应属性的数据使用者分配权限时，即 $(p, attr) \in APA$，则数据使用者可以利用其解密密钥解密链上密文获得对称密钥进而获得数据拥有者隐私数据。如果数据拥有者未向拥有某些属性的数据使用者分配权限，则有 $(p, attr) \notin APA$，这种情况下，数据使用者解密密钥无法解密链上密文，也就无法获得数据使用者隐私数据。因此，其满足4.4与4.5，该状态转换函数为安全保持的。

最后，链上数据获取仅仅涉及已授权数据使用者获得隐私数据的过程，显然其为安全保持的。因此，链上权限分配模型对应的状态转换系统初始状态为安全的，同时，其各个状态转换函数为安全保持的，因此该模型为安全的。

4.4 本章小结

本章介绍了基于智能合约的权限分配模型，整个模型进行权限分配的过程包括：区块链系统初始化，角色与属性分配交易，会话交易，链上数据存储，链上数据获取。各个过程均通过在区块链上发起交易完成，交易逻辑定义在智能合约内，利用区块链去中心化的特性，实现了在不依赖于可信的三方的情况下根据数据使用者属性匹配的方式实现权限分配及继承。

第五章　链上权限的主动撤销机制

在第四章中，介绍了本文提出的链上权限分配模型，该模型可以安全高效的实现权限管理，同时更为细粒度和灵活。在很多情况下，仅仅能够分配权限无法满足系统要求，还需要能够撤销已分配给数据使用者的访问权限，本章，主要介绍如何撤销已授权数据使用者的访问权限。

在进行权限分配时，主要依靠改变用于进行对称密钥加密的 CP-ABE 中的访问控制结构实现将权限分配给特定的数据使用者。当数据使用者的属性，包括自身属性以及分配给数据使用者的角色以及角色属性，符合访问控制结构时，数据使用者可以正确解密经 CP-ABE 加密过的对称密钥，进而获得数据拥有者隐私数据。因此，实现权限撤销需要实现数据使用者的属性不再满足访问控制结构，无法解密密文获得对称密钥，即主要针对属性基加密实现权限撤销。Yu[61] 等人利用代理重加密技术实现用户权限撤销，但该方案仅支持访问控制结构中均为与策略的权限撤销，同时，当撤销用户权限时，需要重新产生包括代理密钥在内的所有密钥。Liu 等人[62] 提出了一种基于时间戳的代理重加密方案，在经过事先定义好的一段时间后实现用户访问权限的撤销。Peng Zhang 等人[63] 提出在加密阶段将撤销用户的身份信息加入密文，同时，其提出了一种利用属性更新密钥进行属性更新的有效方法，利用该更新密钥可以更新密文，更新已完成属性更新的用户的解密密钥，可以但未完成属性更新的用户的解密密钥。Yanfeng Shi 等人[64] 提出了一种 KP-ABE 环境下的权限撤销方案，用户密文分为两部分，一部分与策略相关，另一部分与用户身份相关，相应的，解密密钥也分为两部分，一部分与用户属性相关，另一部分与用户身份相关。

这些方案在基本的属性基加密方案基础上增加了额外的计算开销，增加了算法复杂性。针对本文提出的模型，在本章提出了静态撤销和动态撤销相结合的权限撤销方案。通过时间戳实现粗粒度的静态撤销，该方式可同时撤销多位数据使用者的访问权限。通过将数据使用者组织为二叉树实现更加细粒度的动态撤销，其可以精确到单个数据使用者的权限撤销。

5.1　基于时间戳的链上权限撤销

利用时间戳来实现数据使用者权限的撤销的基本思路是为每一位已授予数据访问权限的数据使用者指定可访问数据的有效时间。在该方案中，数据拥有者在同意数据使用者的数据请求时，为每一数据使用者指定可访问数据的时间，其存储在数据拥有者维护的数据请求者列表内，数据拥有者维护的数据使用者列表的数据格式如图5-1所示。

数据拥有者在利用 CP-ABE 加密对称密钥时，会在访问控制结构内增加时间比较。

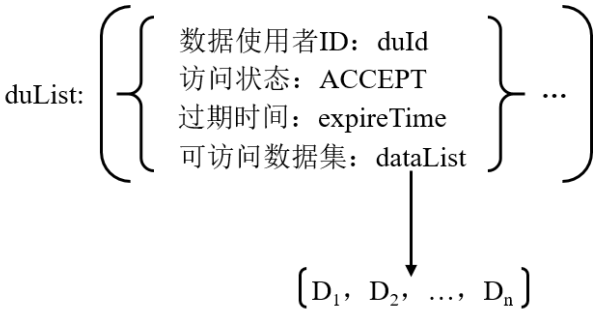


图 5-1 数据使用者列表

记录在数据拥有者的数据使用者列表内的数据使用者均与一有效时间 $expireTime$ 对应，该时间与数据使用者解密密钥的过期时间相同，只有该时间大于当前时间时，数据使用者才可以正确访问数据。例如图5-2中所示 Alice 和 Bob 的访问状态数据，如果在访问控制结构内指定有效时间应在 2020 年 3 月 5 日之后，由于 Alice 的有效时间到 2020 年 3 月 1 日，其在访问控制结构内指定的时间之前，所以其将没有权限访问区块链上的密文，而 Bob 的有效时间到 2020 年 4 月 1 日，其在指定时间之后，所以其仍有权限访问区块链上的密文。

duList		
duId	dataList	expireTime
Alice	[D_1,D_2…]	01/03/2020
Bob	[D_1,D_2…]	01/04/2020

图 5-2 有效时间

在算法3.3中，当数据使用者在用 CP-ABE 加密对称密钥时，在访问控制结构内除了指定数据使用者的属性应满足的条件外，还应指定 $expireTime > now$，now 代表加密对称密钥时的当前时间。当数据使用者可访问数据的权限到期后，需要撤销其访问权限，由于其已经拥有原来的对称解密密钥，因此需要使用新的对称密钥重新对数据拥有者的隐私数据进行加密，然后上传至云数据库。同时，需要将该新的对称加密密钥通过 CP-ABE 加密后上传至区块链。数据拥有者根据维护的数据使用者列表中每一位数据使用者的 $expireTime$ 项决定何时进行重加密交易。在用 CP-ABE 进行重加密时，访问控制结构中的 now 更新为当前时间，则在此时访问权限过期的数据使用者将不能解密新的密文。

我们介绍了利用时间戳实现数据请求者访问权限的撤销，使用该方式进行权限撤销的最大缺陷为只有到达了特定的时间点才可以进行权限撤销操作，并且该时间与数据拥有者为数据使用者生成解密密钥时指定的时间相同，无法动态的实现权限撤销。为了实现更加灵活的权限撤销功能，我们引入撤销列表这一结构，并将数据使用者构造为一颗

二叉树，通过子集覆盖算法，可以更加高效的实现数据使用者权限撤销。

5.2　基于二叉树的链上权限撤销

在撤销部分数据使用者权限时，所有的数据使用者 N 可以分为两个部分：已撤销权限的数据使用者集合 R，未撤销权限的数据使用者集合 $N \setminus R$。对于数据拥有者，所有访问其数据的数据使用者可以以树形结构组织起来，利用树形结构的特点进行高效的数据使用者权限撤销。

本文将数据使用者构造为一颗二叉树 T_{du}，$T_{du} = \langle root_{du}, V_{du}, V_{in}, V, E \rangle$。其中，$root_{du}$ 代表根节点，V_{du} 代表叶子节点，每一叶子节点均与一数据使用者相对应，V_{in} 代表非叶子节点，$root_{du} \in V_{in}$，V 代表所有节点，$V = V_{du} \cup V_{in}$，E 代表树中的边。

如果有 N 个数据使用者，则 $|V_{du}| = N$，我们假设该二叉树为满二叉树且 N 为 2 的幂，这样的一颗二叉树共有 $2N - 1$ 个节点（叶子节点和非叶子节点之和）。对于 $1 \leqslant i \leqslant n$，我们令 v_i 表示树中的某一节点，$v_i \in V$。令 R 表示一组叶子节点，则 R 就代表了一组数据使用者。我们规定 $ST(R)$ 表示 R 以及 $root_{du}$ 引入的最小子树，$ST(R)$ 是唯一的。对于满二叉树内的任意节点 v_i，均有一颗根节点为 v_i 的子树，我们用 S_i 表示该子树的所有叶子节点，也即所有的数据使用者。给定一组被撤销权限的数据使用者列表，这些被撤销权限的数据使用者对应的叶子节点组成 R，令 $S_{v_1}, S_{v_2}, \cdots S_{v_i}$ 表示以 v_i 为根节点的一组子树，其中 v_i 与 $ST(R)$ 中的所有出度为 1 的节点相连但不在 $ST(R)$ 内，则这些子树的叶子节点集 S_i 的并集 $S_1 \cup S_2 \cup \cdots S_i$ 即为所有的未撤销权限的数据使用者。

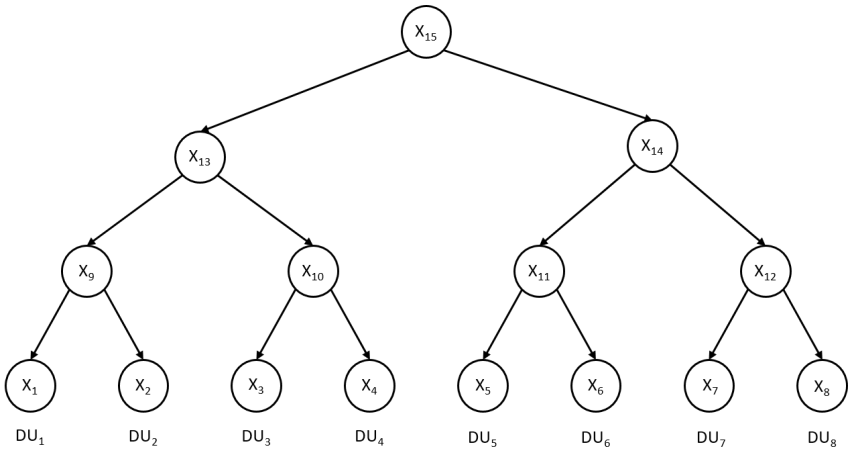


图 5-3 数据使用者树

考虑如图5-3所示的一颗满二叉树，其叶子节点为 $x_1, x_2, x_3, \cdots x_8$，这些叶子节点表示数据使用者 $DU_1, DU_2, DU_3, \cdots DU_8$。如果我们撤销 x_1 和 x_4 的访问权限，x_1 和 x_4 构成节点集合 R，即 $R = \{x_1, x_4\}$，则由 x_1，x_4 以及根节点 x_{15} 构成的最小撤销子树如图5-4所示。由前面的定义可知，以 x_2，x_3，x_{14} 为根节点的子树的所有叶子节点所代表

的数据使用者即为所有未撤销访问权限的数据使用者，即 x_2，x_3，x_5，x_6，x_7，x_8。我们将节点 x_2，x_3，x_{14} 所构成的节点集合表示为 $cover(R)$，即 $cover(R) = \{x_2, x_3, x_{14}\}$。利用 $cover(R)$ 内的每一节点都与未撤销权限的数据使用者的一个子集相对应，以较少的节点数表示了较多的未撤销权限的数据使用者，可以将该算法称为子集覆盖算法，利用该算法优化了进行权限撤销的效率。

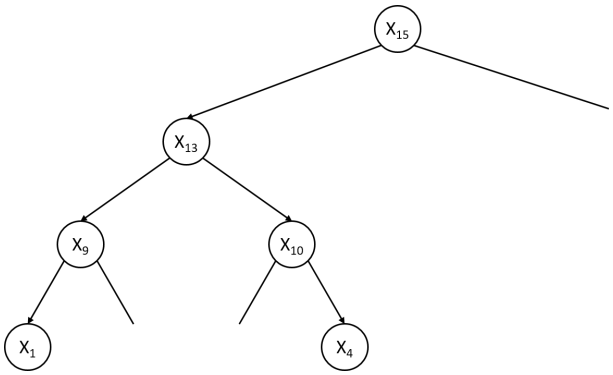


图 5-4 最小撤销子树

本章介绍的利用该数据使用者树进行权限撤销需要在解密密钥生成与数据加密阶段进行改进：

- 在解密密钥生成阶段，该数据使用者树内的每一节点 x_i 内均保存一随机版本号 ver_i，各个节点内保存的版本号均不相同且各不相关。每一数据使用者均对应于一叶子节点，其拥有该叶子节点到根节点的路径上经过的节点的所有版本号。例如在图5-3中，DU_1 拥有的版本号为 $\{ver_1, ver_9, ver_{13}, ver_{15}\}$。当数据拥有者为 DU_1 产生解密密钥时，会将其拥有的版本号作为一部分属性生成解密密钥。例如对 DU_1 来说，其属性内将包括其拥有的版本号集 $\{ver_1, ver_9, ver_{13}, ver_{15}\}$；
- 在数据加密阶段，当数据拥有者想要撤销一部分数据使用者的访问权限时，其首先根据已被撤销权限的数据使用者计算出 $cover(R)$，然后将 $cover(R)$ 内包含的节点内的版本号以或逻辑的访问策略加入访问控制树内。例如在图5-3中，如果撤销 x_1，x_4 的访问权限，则由上文可知，$cover(R)$ 内包含的节点为 $\{x_2, x_3, x_{14}\}$，这些节点内保存的版本号为 $\{ver_2, ver_3, ver_{14}\}$，则数据拥有者在加密数据时，会在访问控制树内加入 $ver_2 \vee ver_3 \vee ver_{14}$ 的访问控制逻辑。

如果数据使用者的权限未被撤销，则其拥有的版本号集与 $cover(R)$ 必有交集，且该集合内仅含一个元素，其属性满足访问控制结构。假设数据请求者为图5-3中的 DU_5，则其到根节点的路径为 $path_{DU_5} = \{x_5, x_{11}, x_{14}, x_{15}\}$，我们计算 $path_{DU_5}$ 与 $cover(R)$ 的交集，图5-3中，$cover(R) = \{x_2, x_3, x_{14}\}$，则 $path_{DU_5} \cap cover(R) = \{x_{14}\}$。$DU_5$ 拥有版本号 ver_{14}，其属性满足 $ver_2 \vee ver_3 \vee ver_{14}$，因此其可以正常解密。如果数据使用者权限已被撤销，其拥有的版本号集与 $cover(R)$ 的交集必为空集，其属性不满足访问

控制结构。例如图5-3中的 DU_1，其到根节点的路径为 $path_{DU_1} = \{x_1, x_9, x_{13}, x_{15}\}$，而 $path_{DU_1} \cap cover(R) = \{\varnothing\}$。将 $cover(R)$ 内的节点中的版本号以或的逻辑加入访问策略，保证了所有未撤销权限的数据使用者仍可正常访问数据，而已被撤销权限的数据使用者将无法访问数据，实现了权限的动态撤销。

由以上描述可知，整个算法的核心为数据使用者二叉树的设计与操作，为了便于讨论，我们之前规定的二叉树为满二叉树，在实际应用中，只要我们设计的二叉树的叶子节点均为数据使用者节点即可。在我们构造的数据使用者二叉树中，引入节点颜色的概念，节点颜色一共有三种取值：

- RED：数据使用者二叉树初始状态时所有节点默认为红节点；
- $BLACK$：由被撤销权限的数据使用者与根节点构成的撤销子树中的所有节点为黑节点；
- $BLUE$：$cover(R)$ 内包含的所有节点为蓝节点。

在图5-3中，如果被撤销访问权限的数据使用者对应的节点为 x_1 与 x_2, 则引入颜色定义后个节点的颜色取值如表5.1所示。

表 5.1 节点颜色取值

节点集	颜色取值
$\{x_1, x_2, x_9, x_{13}, x_{15}\}$	$BLACK$
$\{x_{10}, x_{14}\}$	$BLUE$
$\{x_3, x_4, x_5, x_6, x_7, x_8, x_{11}, x_{12}\}$	RED

可以发现节点颜色有以下几个特点：

- 若某一节点为黑节点，则其到根节点的所有节点均为黑节点。在将某一已撤销叶子节点到根节点的路径上经过的所有节点变为黑节点时，若碰到某一节点为黑节点，则由于该黑节点到根节点的路径上经过的所有节点均为黑节点，所以访问到该黑节点即可，无需再向上访问。在图5-3中，撤销数据使用者节点 x_1 以及 x_2，在撤销 x_1 时，该节点到根节点的所有节点均为红色，则需要从 x_1 一直访问到根节点 x_{15} 并将路径上经过的所有节点变为黑色。当撤销数据使用者节点 x_2 时，我们需要将 x_2 到根节点经过的所有节点变为黑色，但当访问到 x_2 的父节点 x_9 时，其已经为黑色节点，则此时无需再向上访问，可继续查看下一个已撤销的数据使用者节点，如果没有其他撤销数据使用者节点，撤销子树构造完成；
- 非叶子节点的黑节点的子节点中必有一个黑节点。若某一非叶子节点为黑节点，则必存在一个黑叶子节点与该节点相连，而黑叶子节点到某一内部黑节点所经过的所有节点均为黑节点，所以该黑节点必有一子节点为黑节点；

- 红节点的所有后代均为红节点。若红节点的后代中存在一黑节点，则该黑节点到根节点的路径上经过的所有节点均为黑节点，由于该红节点也在这条路径上，则该红节点应该为黑色，与其为红节点矛盾，因此在该红节点的后代中不存在黑节点。利用该特点与上个特点，在计算 $cover(R)$ 时，我们并不需遍历二叉树中的所有节点，当我们确定了与黑节点相连的某一节点为红节点时，将其变为蓝节点，同时将其加入 $cover(R)$。以该蓝节点为根节点的子树中的所有节点均为红节点，所以这些节点中不可能存在蓝节点，因此我们只需遍历以该黑节点的另一子节点为根的子树即可。在图5-3中，如果已撤销的数据使用者子节点为 x_1 和 x_2 时，撤销子树中的所有节点为黑节点。我们从根节点开始遍历，遍历方式使用广度优先遍历，计算 $cover(R)$。当访问到根节点的右子节点 x_{14} 时，由于其是与黑节点相连的红节点，所以将其改为蓝节点并加入 $cover(R)$，此时，我们无需再遍历以 x_{14} 为根的子树，只需遍历以 x_{13} 为根的子树。同理，当访问到 x_{10} 时，将其改为蓝节点并加入 $cover(R)$，无需再访问以 x_{10} 为根的子树，只需访问以 x_9 为根的子树。

利用我们构造的数据使用者二叉树的三个特点，优化了数据使用者撤销过程的效率。效率的提升与已撤销权限的数据使用者对应的数据使用者节点在二叉树叶子节点中的位置有直接关系。如果撤销数据使用者节点在叶子节点中的位置较为集中，无论是构造撤销子树还是计算 $cover(R)$ 的速度都会得到提升。在图5-3中，假设撤销四个数据使用者，我们考虑两种极端情况：

- 撤销数据使用者节点是完全分散的，在叶子节点中，按从左往右的顺序，从撤销第一个节点开始，每隔一个节点为一个撤销数据使用者节点，即为撤销 $\{x_1, x_3, x_5, x_7\}$；
- 撤销数据使用者节点是完全集中的，不再每隔一个节点为一个撤销数据使用者节点，而是从左边第一个撤销数据使用者节点开始，连续四个节点均为被撤销权限的数据使用者节点，即为撤销 $\{x_1, x_2, x_3, x_4\}$。

第一种情况下撤销 x_1 时，x_1 到根节点的所有节点均需要访问一遍，共访问 4 个节点。在撤销 x_3 时，其访问到 x_{13} 已经为黑节点，无需再访问，共访问 3 个节点。同理，撤销 x_5 需要访问 4 个节点，撤销 x_7 需要访问 3 个节点，则构造完成撤销子树共需访问 14 个节点。以满二叉树为例，如果有 N 个数据使用者节点，撤销其中 $N/2$ 个数据使用者节点，若以该情况撤销，则共需访问 $\left(\frac{N \cdot logN}{4} + N\right)$ 个节点。在计算 $cover(R)$ 阶段，从根节点开始遍历，已知访问到叶子节点才会出现红节点，所以整个数据使用者二叉树中的所有节点都需要访问一遍，共需访问 15 个节点。仍以满二叉树为例，如果共有 N 个叶子节点，撤销其中的 $\frac{N}{2}$ 个，共需访问 $2N-1$ 个节点。同理，在第二种情况下，计算撤销子树时，需要访问 $\left(\frac{N \cdot logN + 9N}{8}\right)$ 个节点，在计算 $cover(R)$ 时，访问到 x_{14} 即为蓝节点，则以 x_{14} 为根的子树上的所有节点均无需访问，总共只需访问 $N+1$ 个节点。

数据使用者树内的节点数据格式如图5-5所示。根据以上讨论可以知道针对数据使用者树的操作算法主要有构造数据使用者树与计算 $cover(R)$。构造数据使用者树的过

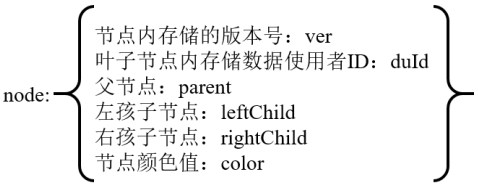


图 5-5 数据使用者树内节点

程如算法5.1所示，使用迭代的方式来构造该二叉树，迭代即通过重复某一过程由初始

算法 5.1 构造数据使用者树

输入： 数据使用者节点集 $duNodes$，版本号集 ver

输出： 空值

1: 如果 $duNodes$ 中的节点数量为 0，1 或者 2，直接构造出数据使用者二叉树，指定树的根节点，算法结束。
2: 以 $duNodes$ 中每相邻两个节点为左右孩子节点构造其父节点 $parentNode$，这些父节点构成父节点集 $parentNodes$。如果 $duNodes$ 中有剩余节点，将其加入 $parentNodes$。
3: 以 $parentNodes$，ver 为参数，继续运行 $construTree$ 算法。

值得到最后的结果，在构造二叉树的过程中，即为重复某一过程，由叶子节点得到根节点。节点内数据使用者 ID 这一项只有叶子节点内有值，表示该叶子节点代表的数据使用者，内部节点这一项不赋值。给定被撤销权限的数据使用者，根据数据使用者树计算 $cover(R)$ 的过程如算法5.2所示。

算法 5.2 计算 $cover(R)$

输入： 数据使用者撤销列表 R

输出： 空值

1: 通过 R 得到数据使用者撤销节点列表 $revokeNodes$。
2: 对于 $revokeNodes$ 内的每一节点 $revokeNode$，重复步骤 3，得到撤销子树 $revokeTree$。
3: 查看 $revokeNode$ 到根节点的每一节点，若其不为黑节点，将其改为黑节点，如果遇到了黑节点，该步骤结束。
4: 广度优先遍历数据使用者二叉树，如果某节点为黑节点，查看其是否有红子节点，若有将该节点变为蓝节点并加入 $cover(R)$，同时，不必再遍历以该子节点为根的子树，若没有，继续遍历。
5: 返回 $cover(R)$。

给定数据使用者撤销列表，我们要能够从数据使用者撤销列表得到 $cover(R)$。我们从数据使用者撤销列表中给定的数据使用者 ID 得到在数据使用者树中其对应的叶子节点，在节点中有颜色值这一属性来标识当前节点的颜色。在撤销数据使用者之前，数据

使用者树内的所有节点颜色均用红节点表示，当撤销部分数据使用者的权限后，将所有撤销数据使用者对应的叶子节点到根节点的路径上经过的所有节点的颜色改变为黑色，由黑节点构成的子树为撤销子树，与撤销子树相连的每一红节点即为 $cover(R)$ 中的节点。数据拥有者在撤销部分数据使用者的权限时，其首先使用新的对称密钥加密相关隐私数据。然后，使用新的访问控制树加密该对称密钥，该访问控制树内除包含的版本号信息不同外，与之前加密时的结构相同。新访问控制树将 $cover(R)$ 内的版本号以或的形式包含进来，实现了权限的撤销。

5.3 链上权限撤销安全性分析

针对本章提出的两种权限撤销机制的安全性证明主要包括两个方面：

- 前向安全性：在撤销部分数据使用者的访问权限后，未被撤销访问权限的数据使用者能够正确访问数据；
- 后向安全性：已被撤销访问权限的数据使用者将无法正常访问数据。

在利用时间戳实现权限撤销时，数据使用者权限被撤销是因为其有效访问时间已过期，根据该过期时间生成的 CP-ABE 的解密私钥已不再满足新加密密文的访问控制结构。因此，其将无法获得新的对称解密密钥，进而无法获得数据拥有者的隐私数据，保证了后向安全性。同时，未被撤销访问权限的用户，其访问时间仍在访问控制结构内指定的有效时间内，可以正确解密密文获得对称解密密钥，满足了前向安全性。

在利用数据使用者树进行权限变更时，通过版本号控制数据使用者是否能正确解密密文，不同的数据使用者具有不同的版本号集。每当有数据使用者的权限被撤销时，数据拥有者会使用新的包含版本号的访问策略利用 CP-ABE 加密新的对称密钥。已被撤销权限的数据使用者拥有的版本号不在该访问策略要求的版本号集内，其属性将不满足访问控制树结构，无法解密密文，保证了后向安全性。如果数据使用者未被撤销权限，则其拥有的版本号满足数据拥有者的访问策略，其仍可正常解密密文，保证了前向安全性。

5.4 本章小结

本章主要针对第三章提出的链上权限分配模型提出了两种权限撤销机制：基于时间戳的权限撤销与基于二叉树的权限撤销。前者将时间戳信息加入 CP-ABE 的访问控制结构，通过规定数据使用者的有效访问时间实现权限撤销；后者将版本号信息加入访问控制结构以及数据使用者的属性内，通过为不同的数据使用者分配不同的版本号以及改变访问策略内要求的数据使用者应该拥有的版本号实现权限撤销。

第六章 链上权限管理系统原型实现

本章介绍了用户可自主控制的链上权限管理系统原型的实现。首先介绍了系统的架构设计，系统主要有三个参与方：数据拥有者，数据使用者，属性管理机构以及云数据库，区块链，在区块链上实现了数据拥有者隐私数据访问权限的管理。然后，介绍了权限管理的具体流程，主要包括数据使用者角色分配，权限链上发布，链上数据获取。最后对该系统的性能进行了测试与分析。

6.1 系统设计与实现

6.1.1 系统开发环境

本章介绍的系统原型实现的开发环境如表6.1所示。

6.1.2 系统架构设计

链上权限管理系统的架构如图6-1所示，系统的参与者共有三方：数据拥有者，数据使用者以及属性管理机构。数据拥有者将自身隐私数据经过对称加密后存入云数据库，同时将该对称密钥经过 CP-ABE 加密后产生密钥密文存入区块链。数据使用者通过区块链向属性管理机构申请角色与属性，属性管理机构根据该数据使用者的身份以及证件等信息通过区块链向数据使用者分配角色与属性。数据使用者根据得到的角色与属性通过区块链向数据拥有者请求密钥，数据拥有者使用数据使用者的角色与属性生成相应的解密密钥并通过区块链分发给数据使用者。数据使用者可以通过区块链得到密钥密文并使用自身解密密钥解密，如果其有权限，将正确解密，然后从云数据库获得数据拥有者隐私数据。如果其未被分配权限，则无法解密，从而无法获得数据拥有者隐私数据。整个

表 6.1 开发环境

名称	参数
CPU	Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz
内存	8.00GB
云服务器操作系统	Ubuntu 16.04 64 位
云服务器内存	2.00GB
云数据库	MySQL 8.0
开发工具	Visual Studio Code 1.44.0、Node.js 12.13.1

系统进行权限管理的流程主要有以下几步：数据使用者角色分配，权限链上发布，链上数据获取。

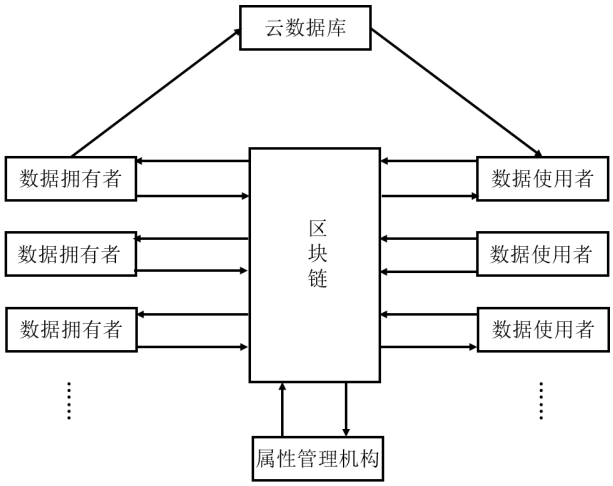


图 6-1 系统架构

6.1.3 数据使用者角色分配

本文以 Hyperledger Fabric 构建区块链网络，区块链网络内共有三个组织：org1，org2，org3，分别代表数据拥有者组织，数据使用者组织以及属性管理机构组织，每一组织内各有一个 peer 节点，同时另有一排序服务节点 orderer，节点说明如表6.2所示。排序服务节点 orderer 根据网络配置初始化区块链网络，同时将 org1，org2，org3 组织成一个联盟，通过通道配置文件构建通道。在 peer 节点上配置链码即本文设计的智能合约，并由各个 peer 节点共同维护分布式账本。客户端可以通过 peer 节点调用链码发起交易，更改状态数据库的内容。交易经由各个 peer 节点背书后提交给 orderer 节点，由 orderer 节点排序打包后广播到区块链网络中的各个节点。

表 6.2 节点说明

节点名称	节点类型	IP 地址
orderer.example.com	排序服务节点	112.124.6.18:7050
peer0.org1.example.com	peer 节点	112.124.6.18:7051
peer0.org2.example.com	peer 节点	112.124.6.18:9051
peer0.org3.example.com	peer 节点	112.124.6.18:11051

区块链网络中的每一节点均由其所属组织的 CA 颁发数字证书，当节点连接到通道时，其数字证书表明了其所属的组织，该功能由通道 MSP 完成，节点可以凭借该数字证书与区块链网络交互。当客户端通过节点发起交易时，该交易被广播到区块链网络中

的其他节点，每一节点分别执行该交易并生成交易执行后的结果。该结果并不会用来更新状态数据库，各个节点仅仅将为该交易结果签名，然后返回给客户端。当客户端得到足够多的交易背书时，其将该交易提交给 orderer 节点。orderer 节点将来自各个客户端的交易排序打包成区块然后将该区块发送给区块链网络中的其他节点。其他节点检查区块中的每一笔交易得到的背书是否满足背书策略，将满足的交易组成区块加入区块链。区块链网络的配置过程如下：

- 根据配置文件 crypto-config.yaml，利用 Hyperledger Fabric 的 cryptogen 工具生成网络中各个参与节点的 x509 证书和签名密钥，其由 MSP 管理，同时在 crypto-config.yaml 文件里也规定了网络拓扑结构；
- 根据配置文件 confogtx.yaml，利用 configtxgen 工具生成创世区块以及发起通道配置交易；
- 通过 Docker 服务利用下载好的镜像文件以及创世区块激活区块链网络；
- 根据通道配置交易创建通道并将各个组织内的节点加入通道；
- 在各个节点上配置链码并初始化，客户端可以通过节点调用链码发起交易。

区块链网络配置完成后，组织中的客户端通过其组织内的节点发起初始化交易。以 DO_i 表示数据拥有者 ID，其身份为 DO，以 DU_i 表示数据使用者 ID，其身份为 DU，以 AM 表示属性管理机构，其身份为 AM。不同身份的实体加入区块链网络后，首先需要运行初始化算法，发起状态数据初始化交易，以初始化 DO_1 为例，该交易的信息如图6-2所示。

```
Transaction ID: 92b8a828a69e7a6b7fe9947de1761bdd6e744c28af8373bcae8fcd45fdaaf007
Args: ["setUp","DO1","DO","7479706520610a7120…"]
{ fcn: 'setUp',
 params:
  [ 'DO1',
    'DO',
    '7479706520610a7120…' ] }
```

图 6-2 状态数据初始化交易

初始化交易完成后，数据拥有者，数据使用者以及属性管理机构的状态数据如图6-3所示。数据拥有者的状态数据内主要包含以下几项：

- 数据集 $dataList$，主要保存自身存储到云数据库内的隐私数据的元数据，数据使用者可以根据该元数据得到数据拥有者的隐私数据；
- 数据使用者列表 $duList$，主要保存访问其数据的数据使用者的信息；

- 解密密钥集 sk，其中保存分配给每一位数据使用者的解密密钥，该解密密钥使用相应的数据使用者的自身公钥加密；
- 公钥 PK，数据拥有者运行 CP-ABE 初始化算法生成的公钥 PK，存放在区块链上实现共享。

Transaction ID: 38d707a61a0eec76ba12ddc73ea7e0aa3b6761ff83d844ecea4b4fb4fb6f5ec3
Args: ["query","DO1"]
{ fcn: 'query', params: ['DO1'] }
==========
{"identity":"DO","dataList":[],"duList":[],"sk":[],"PK":"7479706520610a71203837383037313
037393936363333313235323234333737383139383437353430343938313538303638383331393
939343134323038323131303238363533333939323636343735363330383830323232…"}

Transaction ID: 970dba1ed4dedc327489b048d17e86d60d565a31c6637cfda801602308522ed5
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{},"roleStateList":{},"askUseRoleList":[],"currentRoleList":[]
,"currentAttrList":[],"askAccessList":[],"session":false,"askForKey":{},"pk":"MFwwDQYJKo
ZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ=="}

Transaction ID: ed981766f13dc1ed1a54d783cd695a9ae93c0eb724a2cd03bebfa4e3b237636b
Args: ["query","AM"]
{ fcn: 'query', params: ['AM'] }
==========
{"identity":"AM","attrTree":"{\"_root\":{\"attr\":\"root\",\"parent\":null,\"children\":[]}}","role
AttrList":{“R1":["RA1","RA2"],“R2":["RA3","RA4"],“R3":["RA5","RA6"]}}

图 6-3 初始化状态数据

数据使用者的状态数据内各数据项的说明如表6.3所示。其主要有九项数据。其中 $attrStateList$ 与 $roleStateList$ 分别表示数据使用者当前角色以及属性的状态，如果为 NA，代表初始化状态，$REQUEST$ 代表数据使用者当前正在申请该角色或者属性，$ACTIVE$ 代表数据使用者已被分配该角色以及属性。$askUseRoleList$ 里面保存的是数据使用者请求激活的角色，$currentRoleList$ 表示数据使用者当前已激活角色，$currentAttrList$ 代表数据使用者当前使用的属性。$askAccessList$ 里包含数据使用者当前请求访问的数据的信息，$session$ 表示数据使用者当前是否处于会话中，若为 $true$，则代表其正在请求激活角色。$askForKey$ 表示数据使用者解密密钥的请求状态，pk 为数据使用者的自身公钥，数据拥有者可以用该公钥加密数据使用者的解密密钥。

属性管理机构的状态数据主要有两项：属性树 $attrTree$ 以及当前维护的角色属性列表 $roleAttrList$。在角色属性列表内，表明了各个角色拥有的属性集，图6-3所示的属性管理机构管理 3 个角色，分别为 $R1$，$R2$ 以及 $R3$。其中 $R1$ 拥有属性 $RA1$，$RA2$，$R2$ 拥有属性 $RA3$，$RA4$，而 $R3$ 拥有属性 $RA5$，$RA6$。

参与系统的各方状态数据初始化完成后，进入角色与属性分配交易，角色请求交易以及该交易完成后数据使用者的状态信息如图6-4所示。

表 6.3 DU 初始化状态数据说明

符号	说明
$attrStateList$	DU 当前对于某一属性的状态，可取值为 NA，$REQUEST$，$ACTIVE$
$roleStateList$	DU 当前对于某一角色的状态，可取值为 NA，$REQUEST$，$ACTIVE$
$askUseRoleList$	DU 请求激活的角色集合
$currentRoleList$	DU 当前激活的角色集合
$currentAttrList$	DU 当前使用的属性集合
$askAccessList$	DU 当前请求访问的数据的 DO 的信息
$session$	标识 DU 当前是否处于会话中
$askForKey$	DU 向 DO 请求解密密钥
pk	DU 自身公钥

Transaction ID: f4e1a89faee683b8efd65abacfc701f6794ac48e032e69479076b2fe69334900
Args: ["askForRoles","DU1","R1"]
{ fcn: 'askForRoles', params: ['DU1', 'R1'] }

Transaction ID: 8d4c098a02a4c1cda3a9f1efd0b3c5b2e057f6667cdcb48cd8ae4a7d0f7e1a8d
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{},"roleStateList":{"R1":"REQUEST"},"askUseRoleList":[],"currentRoleList":[],"currentAttrList":[],"askAccessList":[],"session":false,"askForKey":{},"pk":"MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnPxhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ==="}

图 6-4 DU 请求角色

数据使用者状态数据内的角色状态列表 $roleStateList$ 内增加了一项角色 $R1$，并且其当前状态为 $REQUEST$，代表该数据使用者当前正在请求角色 $R1$。

数据使用者请求角色交易完成后，如果属性管理机构同意该数据使用者获得其请求的角色，则其发起角色分配交易 $duRoleAssign$。角色分配交易以及该交易完成后数据使用者的状态数据如图6-5所示，可以看到，该数据使用者请求的 $R1$ 角色已经变为 $ACTIVE$ 状态。

数据使用者获得相应的角色后，其可以通过请求属性交易向属性管理机构请求相应的属性，然后，属性管理机构会根据其身份，证件等信息判断其是否可以获得相应的属性。如果数据使用者请求了属性 $A1$，$A2$，$A3$，但其不可以获得 $A3$，则其 $A1$，$A2$ 属性将会变为 $ACTIVE$ 状态，而 $A3$ 仍为 $REQUEST$ 状态。角色分配与属性分配均由属性管理机构完成，这些交易完成后，数据使用者的状态数据如图6-6所示。

数据使用者得到角色与属性后，其发起会话交易激活角色。会话交易参与双方为数

Transaction ID:b27b40191e7927cf5d7a69a9bbceec4f4e2d728504ab0352a2b7768f4be7febb
Args: ["duRoleAssign","DU1","R1"]
{ fcn: 'duRoleAssign', params: ['DU1', 'R1'] }

Transaction ID: afd21551a879bae21860d912b5c7f734cb79586017d51448f92ca15773f77c58
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{},"roleStateList":{"R1":"ACTIVE"},"askUseRoleList":[],"currentRoleList":[],"currentAttrList":[],"askAccessList":[],"session":false,"askForKey":{},"pk":"MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ==="}

图 6-5 分配 DU 角色

Transaction ID: 6b92b65fd7effc77534ea5e72addf97d18ed6086e56921055043b3aad84f745e
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{"A1":"ACTIVE","A2":"ACTIVE","A3":"REQUEST"},"roleStateList":{"R1":"ACTIVE"},"askUseRoleList":[],"currentRoleList":[],"currentAttrList":[],"askAccessList":[],"session":false,"askForKey":{},"pk":"MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ==
"}

图 6-6 角色与属性分配

据使用者和属性管理机构，该交易主要用来激活数据使用者已分配到的角色，同时确定数据使用者当前使用的属性集。首先，数据使用者要发起请求激活角色交易，数据使用者只可以激活其状态数据库内角色状态列表 $roleStateList$ 内状态为 $ACTIVE$ 的角色。以数据使用者 $DU1$ 为例，由上文可知，$DU1$ 当前角色 $R1$ 为 $ACTIVE$ 状态，$DU1$ 请求激活角色 $R1$ 后，其状态数据如图6-7所示。$DU1$ 状态数据中的 $askUseRoleList$ 内增加了角色 $R1$，同时，其 $session$ 项也变为了 $true$，代表 $DU1$ 当前正处于会话之中。

数据使用者请求激活角色交易完成后，属性管理机构发起数据使用者角色激活交易，该交易的信息以及激活 $R1$ 后 $DU1$ 的状态数据如图6-8所示。属性管理机构根据数据使用者的请求激活角色列表和其角色状态列表激活角色，请求激活角色列表内的角色

Transaction ID: 54671da580dc055fbb817271351f441c68000c7577a30ec0146ea7ae7fe51f40
Args: ["askUseRoles","DU1","R1"]
{ fcn: 'askUseRoles', params: ['DU1', 'R1'] }

Transaction ID: 2c1cab4b15d54f4f1da66c144b183cb109979468aa09f81e0f05ef583c80270c
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{"A1":"ACTIVE","A2":"ACTIVE","A3":"REQUEST"},"roleStateList":{"R1":"ACTIVE"},"askUseRoleList":["R1"],"currentRoleList":[],"currentAttrList":[],"askAccessList":[],"session":true,"askForKey":{},"pk":"MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ==
"}

图 6-7 DU 请求激活角色

存在于角色状态列表内且状态为 $ACTIVE$ 时，该角色才可被激活，被激活的角色将被加入当前角色列表 $currentRoleList$ 内。同时，将与该角色相关的角色属性加入数据使用者的当前属性集内，数据使用者已分配的属性均加入当前属性集。角色激活完成后，将 $session$ 置为 $false$，代表会话已经结束。

Transaction ID: c313e28771150b88b6a4630d312be9f87ea499947c1be7007697b950332e42c9
Args: ["useRoles","DU1","RA1"]
{ fcn: 'useRoles', params: ['DU1', 'RA1'] }

Transaction ID: 437190fe7316fdbc0b9edb47ac829d3919a1382e7800758c18b409e5f3857cb2
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{"A1":"ACTIVE","A2":"ACTIVE","A3":"REQUEST"},"ro
leStateList":{"R1":"ACTIVE"},"askUseRoleList":["R1"],"currentRoleList":["R1"],"current
AttrList":["RA1","A1","A2"],"askAccessList":[],"session":false,"askForKey":{},"pk":"MFw
wDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ=="}

图 6-8 激活 DU 角色

数据使用者激活角色与属性后，其可以向数据拥有者请求解密密钥，数据使用者首先发起密钥请求交易，该交易信息以及该交易完成后数据使用者的状态数据如图6-9所示。在数据使用者的 $askForKey$ 项内增加了一条记录，该记录以键值对的形式存在，代表数据使用者当前在向哪些数据拥有者请求解密密钥以及当前的请求状态，ASK 代表解密密钥请求中，$ACCEPT$ 代表数据使用者已经得到了该数据拥有者生成的解密密钥。

Transaction ID: 7a161a73fdfebe5a5b4db2357a609d2526768561db4ea86b4b69b6ddae77dedd
Args: ["askForKey","DU1","DO1"]
{ fcn: 'askForKey', params: ['DU1', 'DO1'] }

Transaction ID: 42a94df9c11971643689951cd11cdfa12c021b5258d9c234647e835d6d1fcf10
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{"A1":"ACTIVE","A2":"ACTIVE","A3":"REQUEST"},"roleS
tateList":{"R1":"ACTIVE"},"askUseRoleList":["R1"],"currentRoleList":["R1"],"currentAttrList
":["RA1","A1","A2"],"askAccessList":[],"session":false,"askForKey":{"DO1":"ASK"},"pk":"
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ=="}

图 6-9 解密密钥请求

该交易完成后，数据拥有者将发起解密密钥授予交易，该交易的信息及该交易完成后数据拥有者与数据使用者的状态数据如图6-10所示。

数据拥有者在本地维护数据使用者树，该树内的每一节点内均有一随机版本号，树的叶子节点代表不同的数据使用者。当数据拥有者为数据使用者生成解密密钥时，会将其对应的叶子节点到根节点的路径上经过的所有节点内包含的版本号均作为其属性，然后再根据数据使用者当前角色，属性以及过期时间生成解密密钥，最后将该解密密钥使用数据使用者的自身公钥加密后存放在自身解密密钥集 sk 内，并为该密钥指定一过期时间 $time$。同时，将数据使用者 $askForKey$ 内该数据拥有者的解密密钥授予状态变为 $ACCEPT$，代表该数据拥有者已授予数据使用者解密密钥。

Transaction ID: 20a01667750d68e2829792ec8a9bd3e899d1a79b31701621efa6619aeb636a3c
Args: ["grantKey","DO1","DU1","VreupBK6roxa1Z2XcbGWRoEczG4r+tgYrZ+cgsobww8mM6gX14uzuKJjhd/mLu3o6J…"]
{ fcn: 'grantKey',
  params:
   ['DO1',
     'DU1',
     'VreupBK6roxa1Z2XcbGWRoEczG4r+tgYrZ+cgsobww8mM6gX14uzuKJjhd/mLu3o6JASGvqNSfE…‘]}

Transaction ID: f1ea5b696774b3757ff27772e5acdc314ea4ce8c801661f3740f9d7136869259
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{"A1":"ACTIVE","A2":"ACTIVE","A3":"REQUEST"},"roleStateList":{"R1":"ACTIVE"},"askUseRoleList":["R1"],"currentRoleList":["R1"],"currentAttrList":["RA1","A1","A2"],"askAccessList":[],"session":false,"askForKey":{"DO1":"ACCEPT"},"pk":"MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ=="}

Transaction ID: e59ff640a9a8ea290e011d347203cae99228e6048363bbd27857ca230d2c4117
Args: ["query","DO1"]
{ fcn: 'query', params: ['DO1'] }
==========
{"identity":"DO","dataList":[],"duList":[],"sk": [{"duId":"DU1","duSk":"WWQzIGzIGgNlPCXopQCJUp… ","time":"2020-04-26"}], "PK":"7479706520610a712… "}

图 6-10 解密密钥授予

6.1.4 权限链上发布

数据拥有者为了实现对其隐私数据的权限管理，其首先将自身隐私数据分块加密，加密方式采用对称加密，在本章实现时，对称加密算法使用 AES-128。同时利用对称密钥生成数据 HMAC 值，生成 HMAC 值时使用 MD5 消息摘要算法。将加密后的隐私数据密文编码后上传至云数据库，将生成的 HMAC 值上链。数据使用者得到隐私数据后，可以利用该 HMAC 值判断数据的完整性。将加密密文数据的对称密钥利用 CP-ABE 加密后也上链，在 CP-ABE 内指定不同的访问控制结构以实现访问控制。如果数据使用者在访问策略中指定了角色，其需要参考属性管理结构维护的属性树，找到继承自该角色的其他角色，然后使用第三章介绍的 CP-ABE 扩展模型进行对称密钥的加密，实现了角色间权限继承的功能。如果属性树结构如图6-11所示，则其在属性管理机构的状态数据内的存储形式如图6-12所示。

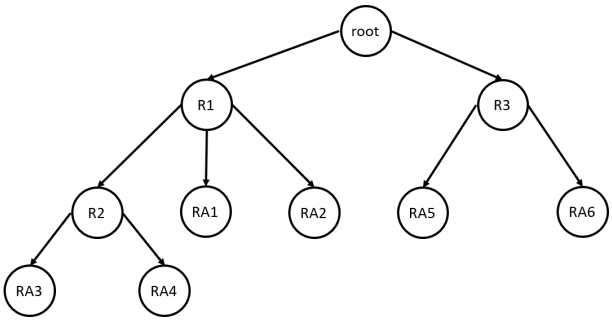


图 6-11 属性树结构

将对称密钥 K 经过 CP-ABE 加密后生成密文文件，假定在加密时使用的访问控制结构为 $R1$ and $A1$，即角色为 $R1$ 且属性为 $A1$ 的数据使用者才可解密该密文。当数据使

Transaction ID: 02aab7f4c5948b7512d7f13729323de727c88b8d888d846eeb4e2920d63bf085
Args: ["query","AM"]
{ fcn: 'query', params: ['AM'] }
==========
{"identity":"AM","attrTree":"{\"_root\":{\"attr\":\"root\",\"parent\":null,\"children\":[{\"attr\":\"R1\",\"parent\":\"~_root\",\"children\":[{\"attr\":\"R2\",\"parent\":\"~_root~children~0\",\"children\":[{\"attr\":\"RA3\",\"parent\":\"~_root~children~0~children~0\",\"children\":[]},{\"attr\":\"RA4\",\"parent\":\"~_root~children~0~children~0\",\"children\":[]}]},{\"attr\":\"RA1\",\"parent\":\"~_root~children~0\",\"children\":[]},{\"attr\":\"RA2\",\"parent\":\"~_root~children~0\",\"children\":[]}]},{\"attr\":\"R3\",\"parent\":\"~_root\",\"children\":[{\"attr\":\"RA5\",\"parent\":\"~_root~children~1\",\"children\":[]},{\"attr\":\"RA6\",\"parent\":\"~_root~children~1\",\"children\":[]}]}]}}","roleAttrList":{"R1":["RA1","RA2"],"R2":["RA3","RA4"],"R3":["RA5","RA6"]}}

图 6-12 属性管理机构状态数据

用者使用该策略加密密文时，由于在属性树内不存在角色继承角色 $R1$，其可以直接使用该策略对对称密钥进行加密。将加密后的文件内容编码和 HMAC 一起上链，这两部分内容构成数据拥有者存储在区块链内的数据列表 $dataList$，其可以作为获得数据拥有者隐私数据的元数据。元数据在状态数据内的存储形式如图6-13所示，主要在 $dataList$ 内增加新的数据项，该数据项主要有三部分内容：数据说明 $dataNote$，HMAC 值 $hmac$ 用来验证数据完整性，密钥密文 ct，其为利用 CP-ABE 加密对称密钥后的密文。

Transaction ID: 1b8c2a5e6b26b7f108f2b81155f04ee9ceb325cc0ed59b9ee39b40547fdaea19
Args: ["query","DO1"]
{ fcn: 'query', params: ['DO1'] }
==========
{"identity":"DO","dataList":[{"dataNote":"D1","hmac":"7cc2d19adae82aea1cc4d6f0ad758070","ct":"000000090000001082ec262dfe6ea…"}],"duList":[],"sk":[{"duId":"DU1","duSk":"WWQzIGzIGgNlPCXopQCJUp… ","time":"2020-04-26"}],"PK":"7479706520610a712…"}

图 6-13 元数据存储

6.1.5　链上数据获取

数据使用者可以通过发起数据请求交易来查看自己是否对相关隐私数据拥有访问权限，该交易的信息及交易完成后数据使用者的状态数据如图6-14所示。数据使用者在其 $askAccessList$ 项内增加了请求访问的数据块信息，当前对于数据拥有者 $DO1$ 的数据块 $D1$ 的数据访问状态为 $REQUEST$，即请求中。

数据请求交易完成后，由其所请求数据的数据拥有者发起访问权限授予交易。$DU1$ 向 $DO1$ 发起数据请求，$DO1$ 查看其角色与属性，如果 $DU1$ 有权限获取相应数据，$DO1$ 发起权限授予交易。该交易完成后，数据拥有者和数据使用者的状态数据如图6-15所示。在 $DO1$ 的状态数据的 $duList$ 项内增加了当前数据使用者访问数据的信息：$DU1$ 可以

Transaction ID: e62b0945093652e418c437dabe2f3dd23de1bdd7627e40e70f1eab4c212b2a61
Args: ["askForData","DU1","DO1","D1"]
{ fcn: 'askForData', params: ['DU1', 'DO1', 'D1'] }

Transaction ID: ccc89842c5ae5a65498447d409933edbbe27488d3e51bb22d874c5e921603c47
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{"A1":"ACTIVE","A2":"ACTIVE","A3":"REQUEST"},"role
StateList":{"R1":"ACTIVE"},"askUseRoleList":["R1"],"currentRoleList":["R1"],"currentAttr
List":["RA1","A1","A2"],"askAccessList":[{"doId":"DO1","askDataList":["D1"],"currentState
":"REQUEST"}],"session":false,"askForKey":{"DO1":"ACCEPT"},"pk":"MFwwDQYJKoZI
hvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ==="}

图 6-14 请求数据

访问数据块 $D1$，有效访问时间到 2020-04-26，这一时间与数据拥有者分配给数据使用者的解密密钥的有效时间相同。$DU1$ 的状态数据的请求访问列表 $askAccessList$ 内请求访问 $DO1$ 的数据块 $D1$ 的访问状态由 $REQUEST$ 变为了 $AGREE$，代表 $DO1$ 已经同意了 $DU1$ 的数据访问请求。

该交易完成后，$DU1$ 可以从区块链上获得相应的元数据。$DU1$ 使用自身解密密钥解密 ct 得到对称密钥 K，然后 $DU1$ 从云数据库中得到利用 K 加密 $D1$ 后生成的密文，$DU1$ 使用 K 解密该密文得到相应的隐私数据。当数据拥有者需要撤销数据使用者权限时，将隐私数据通过新的对称密钥加密，然后将该对称密钥再经过包含新的时间戳以及版本号信息的访问控制树进行加密。

6.2 测试与分析

本节主要从时延与吞吐量两个方面来对系统进行测试，其中时延指从发起交易到区块链系统对该交易做出反应的时间，吞吐量的定义是每秒成功完成的交易数，单位可以用 TPS(Transactions Per Second) 表示。链上权限管理系统原型实现通过调用智能合约发起交易完成，在这个过程中主要涉及到了两类函数：query 与 invoke。query 用来读取当前状态数据库中数据拥有者，数据使用者以及属性管理机构的当前状态，其严格意义上并不算交易，不需要各个节点的共识，只是单纯的查询状态数据的过程，不会改变参与各方的状态数据，但仍可以用吞吐量表示其性能。其他所有函数均属于 invoke，当数据拥有者，数据使用者以及属性管理机构调用这类函数时，将发起交易，改变参与各方在状态数据库内的状态数据。时延与并发的 query 的数量的关系如图6-16所示，其中时延的单位为毫秒。吞吐量与并发的 query 的数量的关系如图6-17所示，其中横坐标并发查询数代表并发的 query 的数量。由图可知，随着并发查询数的增加，时延也会随之增加，但增长较为缓慢，同时，吞吐量也在不断增加，可以维持在 400TPS 以上，可以看到调

Transaction ID: 83d5d78e09115f023bd90f991d235b5ffbdc4be5183c002b1da08ca6681dee21
Args: ["query","DO1"]
{ fcn: 'query', params: ['DO1'] }
==========
{"identity":"DO","dataList":[{"dataNote":"D1","hmac":"7cc2d19adae82aea1cc4d6f0ad758070","ct":"000000090000001082ec…"}],"duList":[{"duId":"DU1","dataList":["D1"],"time":"2020-04-26","accessState":"ACCEPT"}],"sk":
[{"duId":"DU1","duSk":"WWQzIGzIGgNlPCXopQCJUp… ","time":"2020-04-26"}],"PK":"7479706520610a712038373830…"}

Transaction ID: 0bb361614b601811e27b9277819b12df825415d4b2acdf08a41bcb065bdcc5fb
Args: ["query","DU1"]
{ fcn: 'query', params: ['DU1'] }
==========
{"identity":"DU","attrStateList":{"A1":"ACTIVE","A2":"ACTIVE","A3":"REQUEST"},"roleStateList":{"R1":"ACTIVE"},"askUseRoleList":["R1"],"currentRoleList":["R1"],"currentAttrList":["RA1","A1","A2"],"askAccessList":[{"doId":"DO1","askDataList":["D1"],"currentState":"AGREE"}],"session":false,"askForKey":{"DO1":"ACCEPT"},"pk":"MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAISa1jkkC6QHdfPrLvLbzmSegVzzlNnP
xhaqVROCzZm1fZ0U49OchTbjVcFPEQlJU2TlcWIOkKhCUH6HuSVeaqUCAwEAAQ=="}

图 6-15 权限授予

用 query 发起查询的效率很高。

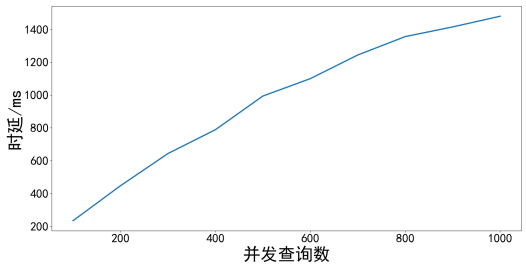


图 6-16 时延

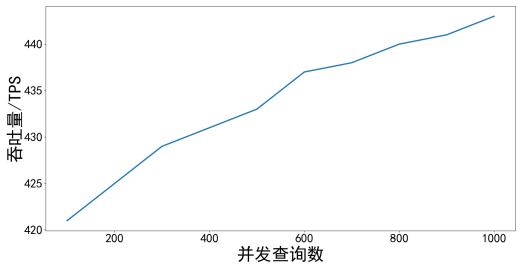


图 6-17 吞吐量

参与各方发起交易的过程即为调用 invoke 函数的过程，随着并发交易数的增加，交易的时延以及吞吐量的变化分别如图6-18和6-19所示。可以看到，调用 invoke 函数的时延要明显高于 query 函数，在并发交易数为 1000 时，已经 5 秒左右。同时，吞吐量也明显小于 query 函数，但也在 130TPS 以上，效率也很高。无论是调用 query 还是 invoke，这里吞吐量都在不断增加的原因初步分析是区块链系统还未达到饱和，还有上升的空间。发起交易时的吞吐量虽然相比发起状态数据查询要偏低，但也维持在一个较高位。

6.3　本章小结

本章主要将本文提出的链上权限管理模型进行了系统原型实现，首先将权限管理过程涉及到的交易分别进行了实现，验证了流程的可行性，其可以实现权限的分配，继承

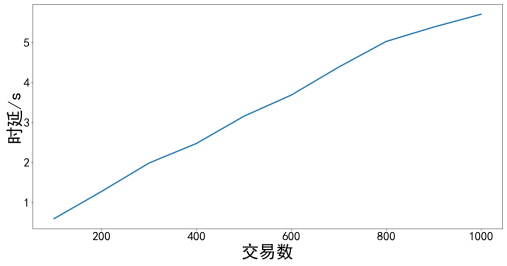


图 6-18 时延

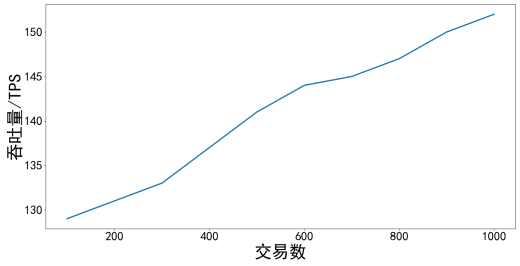


图 6-19 吞吐量

以及撤销。然后对于系统原型实现的性能进行了测试，进行状态数据查询时，其吞吐量在 400TPS 以上，进行交易时，其吞吐量在 130TPS 以上。

第七章 总结与展望

7.1 本文工作总结

随着大数据时代的来临，隐私数据保护形式越来越严峻，如何在保护数据隐私的同时实现数据共享成为了很对研究人员的研究重点，权限管理技术实现了控制用户仅仅可以访问自己已被授权的部分数据，有效保护数据隐私的同时实现了数据共享。传统的权限管理技术通常需要依赖可信第三方实现权限的分配，数据的拥有者无法控制数据访问权限的授予和撤销，数据具有泄露的风险。随着区块链的出现，为权限管理带来了新的方向，区块链本身去中心化的特点使得不依赖于可信第三方实现权限分配成为可能，越来越多的研究集中于如何使用区块链实现高效的权限分配。基于此，本文提出了一种新型的用户可自主控制的链上权限管理模型，该模型将改进的 CP-ABE 算法与区块链结合起来，在去中心化的同时实现了细粒度的权限管理，主要包括权限分配，权限继承以及权限撤销。本文完成的主要工作有以下几点：

1. 本文提出了一种改进的 CP-ABE 模型，该模型在传统 CP-ABE 算法的基础上引入了角色的概念，角色属于一种特殊的属性，其可以代表组织内不同数据使用者所拥有权限的层次化关系，进一步的，角色拥有者角色属性，通过角色属性可以细化这种层次化关系。为了表示这种层次化关系，提出了属性树的概念，属性树满足严格偏序关系，其叶子节点代表角色属性，非叶子节点与非根节点代表角色。角色间存在通路代表这两个角色的所有者所拥有的权限间具有继承关系，数据拥有者在进行数据加密时，会根据这种继承关系制定访问策略从而实现权限继承，构造数据使用者权限间的层次化关系。通过改进的 CP-ABE 模型，可以利用企业内部固有的层次化关系降低其在企业级应用场景下的复杂度。

2. 本文提出了一种基于智能合约的链上权限分配模型，该模型可基于属性实现数据拥有者对数据使用者的权限授予。权限管理过程被编写为智能合约然后部署在区块链节点上，当需要进行权限分配时，模型参与各方通过调用智能合约发起交易实现。数据拥有者可以通过链上交易在不依赖于可信第三方的情况下直接实现权限分配，权限分配根据数据使用者拥有的属性实现。

3. 本文提出了两种权限撤销机制，实现了数据使用者访问数据的有效期一旦过期或者数据拥有者主动撤销数据使用者的访问权限时，实现数据使用者访问权限的自动撤销。前者利用时间戳实现，通过为每一数据使用者分配有效访问时间，当时间过期权限既自动撤销；后者通过构造二叉树为不同数据使用者分配不同的版本号，当数据拥有者想主动撤销权限时即可通过改变访问控制结构内的版本号信息

实现权限撤销。通过上述两种机制，在降低权限撤销操作复杂性的同时提升了灵活性。

4. 本文对链上权限管理原型系统进行了实现及验证。可在区块链上实现数据使用者自主可控的权限分配、权限继承以及权限撤销功能。

7.2 未来研究展望

针对用户数据隐私泄露问题越来越严重的现状，本文提出了一种新型的链上权限管理模型，并根据该模型进行了系统原型的实现及验证，并对其进行了测试与分析。通过总结本文研究内容，发现该模型仍然存在很多改进的空间，未来的研究重点可以放在以下几点：

1. 本文为了实现权限继承引入了属性树的概念，树内角色间的可达关系代表了这些角色所有者的权限间的继承关系，数据拥有者在加密数据时通过将这种可达关系包含进访问控制结构实现了权限的继承。但是，这种方式增加了访问控制结构的复杂度，在加密效率上有所降低，下一步研究如何在尽量高效率的情况下实现权限的继承。

2. 权限管理的整个流程均是在区块链上通过发起交易实现的，交易能否通过的前提是共识是否达成，而共识协议是区块链系统正常运行的核心，本文的研究重点是权限管理如何通过发起区块链上的交易来实现，并没有针对该特定应用场景设计合适的共识协议，实现更加高效，安全的交易过程，因此，下一步需要针对本文提出的模型研究更加完善合适的共识协议。

3. 本文提出的模型针对的权限管理只涉及到读取数据的权限，在权限变更时涉及到的为撤销读取权限，在实际场景下，可能需要写权限以及执行权限，接下来需要继续完善模型，使其支持读权限的同时，也要支持写权限与执行权限，同时实现权限变更的功能。

致谢

时光如白驹过隙，九龙湖畔七年的求学生涯转眼集将结束，值此硕士论文将要完成之际，想要表达对老师与同学由衷的感谢。

首先，需要感谢我的导师宋宇波教授，宋老师爱岗敬业，学识渊博，关爱学生，谈吐幽默，在三年研究生生涯中给予了我莫大的帮助，从毕业设计的选题，内容研究到论文撰写，均是在宋老师的精心指导下完成的。每当我学术方面遇到了困难，他都会与我耐心的讨论，引导我的思路，帮助我解决问题。科研之外，宋老师与我们亦师亦友，与我们交流跑步心得，关心我们的职业发展，热心为我们介绍工作。很幸运能够成为宋老师的学生，再次对宋老师表示由衷的感谢。

然后，需要感谢研究生三年的室友顾志方，李杨，曹凡，你们学业优异，是我科研方面的榜样，同时为人真诚，热心助人，和气友善，与你们一起度过三年的研究生涯是我莫大的幸运，在此，感谢你们三年的陪伴。

感谢同门宋睿，石伟三年来的陪伴，我们一起完成了许多科研与工程项目，互相学习，取长补短，宋睿在学术方面给予了我莫大的帮助，在寻找实习时热心分享给我自己总结的资料，同时，在各个方面扩展了我的知识面。感谢石伟与我一起打球，帮助我拥有了一个健康的体魄。

感谢已毕业的师兄董启宏、张克落、武威、黄强、魏一鸣以及师姐杨慧文、罗平。在我初到实验室时，给予了我莫大的帮助，在科研上为我答疑解惑，给我分享学习资料与经验，积极与我探讨职业规划。在此，对各位师兄师姐表达我深深的感激之情，祝福各位工作生活事事如意。

感谢师弟樊明、杨俊杰、赵灵奇、张仕奇、马小松、徐前川、蒋心造以及师妹祁欣妤、金星妤、陈琪、耿益瑾。你们的到来为实验室带来了新的活力，你们的朝气蓬勃与对待科研的认真态度深深激励着我，鞭策我继续努力勇攀高峰。在此，深深感谢各位师弟师妹，祝福各位学业顺利。

在二十多年的求学生涯中，我的父母给予了我无微不至的照顾与尽心尽责的教育，在此，特别感谢我的父母，你们是我人生路上的灯塔与坚强后盾，祝福你们身体健康，万事如意。

最后，感谢各位评审老师在百忙之中抽出时间评审本文。

参考文献

[1] NAKAMOTO S, et al. A peer-to-peer electronic cash system[J]. Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf, 2008.

[2] CUI P, DIXON J, GUIN U, et al. A blockchain-based framework for supply chain provenance[J]. IEEE Access, 2019, 7:157113-157125.

[3] LINN L A, KOO M B. Blockchain for health data and its potential use in health it and health care related research[C]//ONC/NIST Use of Blockchain for Healthcare and Research Workshop. Gaithersburg, Maryland, United States: ONC/NIST. 2016: 1-10.

[4] LIU X, WANG Z, JIN C, et al. A blockchain-based medical data sharing and protection scheme[J]. IEEE Access, 2019, 7:118943-118953.

[5] SUN Y, ZHANG R, WANG X, et al. A decentralizing attribute-based signature for healthcare blockchain[C]//2018 27th International Conference on Computer Communication and Networks (ICCCN). IEEE, 2018: 1-9.

[6] DASAKLIS T K, CASINO F, PATSAKIS C. Blockchain meets smart health: Towards next generation healthcare services[C]//2018 9th International conference on information, intelligence, systems and applications (IISA). IEEE, 2018: 1-8.

[7] KUMAR T, RAMANI V, AHMAD I, et al. Blockchain utilization in healthcare: Key requirements and challenges[C]//2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom). IEEE, 2018: 1-7.

[8] AZARIA A, EKBLAW A, VIEIRA T, et al. Medrec: Using blockchain for medical data access and permission management[C]//2016 2nd International Conference on Open and Big Data (OBD). IEEE, 2016: 25-30.

[9] WANG H, SONG Y. Secure cloud-based ehr system using attribute-based cryptosystem and blockchain[J]. Journal of medical systems, 2018, 42(8):152.

[10] MIŠIĆ V B, MIŠIĆ J, CHANG X. Towards a blockchain-based healthcare information system[C]//2019 IEEE/CIC International Conference on Communications in China (ICCC). IEEE, 2019: 13-18.

[11] ESPOSITO C, DE SANTIS A, TORTORA G, et al. Blockchain: A panacea for healthcare cloud-based data security and privacy?[J]. IEEE Cloud Computing, 2018, 5(1):31-37.

[12] LAMPSON B W. Protection[J]. ACM SIGOPS Operating Systems Review, 1974, 8(1): 18-24.

[13] HARRISON M A, RUZZO W L, ULLMAN J D. Protection in operating systems[J]. Communications of the ACM, 1976, 19(8):461-471.

[14] WANG Q, JIN H. Data leakage mitigation for discretionary access control in collaboration clouds[C]//Proceedings of the 16th ACM symposium on Access control models and technologies. 2011: 103-112.

[15] SANDHU R S. Lattice-based access control models[J]. Computer, 1993, 26(11):9-19.

[16] HU V C, KUHN D R, XIE T, et al. Model checking for verification of mandatory access control models and properties[J]. International Journal of Software Engineering and Knowledge Engineering, 2011, 21(01):103-127.

[17] LIU L, ÖZSU M T. Encyclopedia of database systems: volume 6[M]. Springer New York, NY, USA:, 2009.

[18] FERRAIOLO D F, SANDHU R, GAVRILA S, et al. Proposed nist standard for role-based access control[J]. ACM Transactions on Information and System Security (TISSEC), 2001, 4(3):224-274.

[19] SANDHU R S, COYNE E J, FEINSTEIN H L, et al. Role-based access control models [J]. Computer, 1996, 29(2):38-47.

[20] HU V C, FERRAIOLO D, KUHN R, et al. Guide to attribute based access control (abac) definition and considerations (draft)[J]. NIST special publication, 2013, 800(162).

[21] 王小明, 付红, 张立臣. 基于属性的访问控制研究进展[J]. 电子学报, 2010, 38(7): 1660-1667.

[22] JIN X, KRISHNAN R, SANDHU R. Reachability analysis for role-based administration of attributes[C]//Proceedings of the 2013 ACM workshop on Digital identity management. 2013: 73-84.

[23] KUHN D R, COYNE E J, WEIL T R. Adding attributes to role-based access control[J]. Computer, 2010(6):79-81.

[24] JIN X, SANDHU R, KRISHNAN R. Rabac: role-centric attribute-based access control [C]//International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, 2012: 84-96.

[25] COYNE E, WEIL T R. Abac and rbac: scalable, flexible, and auditable access management[J]. IT Professional, 2013(3):14-16.

[26] LI F, WANG W, MA J, et al. Action-based access control model and administration of actions[J]. Acta Electronica Sinica, 2008, 36(10):1881-1890.

[27] TARAMESHLOO E, FONG P W. Access control models for geo-social computing systems[C]//Proceedings of the 19th ACM symposium on Access control models and technologies. 2014: 115-126.

[28] 范艳芳, 韩臻, 曹香港, 等. 基于时间限制的多级安全模型[J]. 计算机研究与发展, 2010, 47(3):508-514.

[29] UDDIN M, ISLAM S, AL-NEMRAT A. A dynamic access control model using authorising workflow and task-role-based access control[J]. IEEE Access, 2019, 7:166676-166689.

[30] LIU Q, ZHANG H, WAN J, et al. An access control model for resource sharing based on the role-based access control intended for multi-domain manufacturing internet of things [J]. IEEE Access, 2017, 5:7001-7011.

[31] EL-HINDI M, BINNIG C, ARASU A, et al. Blockchaindb: a shared database on blockchains[J]. Proceedings of the VLDB Endowment, 2019, 12(11):1597-1609.

[32] WANG S, ZHANG Y, ZHANG Y. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems[J]. Ieee Access, 2018, 6: 38437-38450.

[33] ZYSKIND G, NATHAN O, et al. Decentralizing privacy: Using blockchain to protect personal data[C]//2015 IEEE Security and Privacy Workshops. IEEE, 2015: 180-184.

[34] YAO Y, CHANG X, MISIC J, et al. Lightweight and privacy-preserving id-as-a-service provisioning in vehicular cloud computing[J]. IEEE Transactions on Vehicular Technology, 2019.

[35] OUADDAH A, ABOU ELKALAM A, AIT OUAHMAN A. Fairaccess: a new blockchain-based access control framework for the internet of things[J]. Security and Communication Networks, 2016, 9(18):5943-5964.

[36] OUADDAH A, ELKALAM A A, OUAHMAN A A. Towards a novel privacy-preserving access control model based on blockchain technology in iot[M]//Europe and MENA Cooperation Advances in Information and Communication Technologies. Springer, 2017: 523-533.

[37] PINNO O J A, GREGIO A R A, DE BONA L C. Controlchain: Blockchain as a central enabler for access control authorizations in the iot[C]//GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, 2017: 1-6.

[38] ES-SAMAALI H, OUTCHAKOUCHT A, LEROY J P. A blockchain-based access control for big data[J]. International Journal of Computer Networks and Communications Security, 2017, 5(7):137.

[39] JOUX A. A one round protocol for tripartite diffie–hellman[C]//International algorithmic number theory symposium. Springer, 2000: 385-393.

[40] BONEH D, FRANKLIN M. Identity-based encryption from the weil pairing[C]//Annual international cryptology conference. Springer, 2001: 213-229.

[41] GALBRAITH S D, PATERSON K G, SMART N P. Pairings for cryptographers[J]. Discrete Applied Mathematics, 2008, 156(16):3113-3121.

[42] BONEH D, GOH E J, NISSIM K. Evaluating 2-dnf formulas on ciphertexts[C]//Theory of Cryptography Conference. Springer, 2005: 325-341.

[43] SAHAI A, WATERS B. Fuzzy identity-based encryption[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2005: 457-473.

[44] BETHENCOURT J, SAHAI A, WATERS B. Ciphertext-policy attribute-based encryption [C]//2007 IEEE symposium on security and privacy (SP'07). IEEE, 2007: 321-334.

[45] GOYAL V, PANDEY O, SAHAI A, et al. Attribute-based encryption for fine-grained access control of encrypted data[C]//Proceedings of the 13th ACM conference on Computer and communications security. 2006: 89-98.

[46] SHAMIR A. How to share a secret[J]. Communications of the ACM, 1979, 22(11):612-613.

[47] BLAKLEY G R. Safeguarding cryptographic keys[C]//1979 International Workshop on Managing Requirements Knowledge (MARK). IEEE, 1979: 313-318.

[48] BENALOH J, LEICHTER J. Generalized secret sharing and monotone functions[C]// Conference on the Theory and Application of Cryptography. Springer, 1988: 27-35.

[49] WRIGHT A, DE FILIPPI P. Decentralized blockchain technology and the rise of lex cryptographia[J]. Available at SSRN 2580664, 2015.

[50] DI PIERRO M. What is the blockchain?[J]. Computing in Science & Engineering, 2017, 19(5):92-95.

[51] CASTRO M, LISKOV B, et al. Practical byzantine fault tolerance[C]//OSDI: volume 99. 1999: 173-186.

[52] LAMPORT L, et al. Paxos made simple[J]. ACM Sigact News, 2001, 32(4):18-25.

[53] ONGARO D, OUSTERHOUT J. In search of an understandable consensus algorithm[C]// 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). 2014: 305-319.

[54] SZABO N. Smart contracts: building blocks for digital markets[J]. EXTROPY: The Journal of Transhumanist Thought,(16), 1996, 18:2.

[55] SZABO N. Formalizing and securing relationships on public networks[J]. First Monday, 1997, 2(9).

[56] 杨秀文, 严尚安, 曾顺鹏, 等. 关于可达矩阵的求法探讨[J]. 数学的实践与认知, 2003 (11):128-130.

[57] VAN RENESSE R, DUMITRIU D, GOUGH V, et al. Efficient reconciliation and flow control for anti-entropy protocols[C]//proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware. 2008: 1-7.

[58] LI N, TRIPUNITARA M V. Security analysis in role-based access control[J]. ACM Transactions on Information and System Security (TISSEC), 2006, 9(4):391-420.

[59] 熊厚仁, 陈性元, 杜学绘, 等. 基于角色的访问控制模型安全性分析研究综述[J]. 计算机应用研究, 2015, 32(11):3201-3208.

[60] 王婷. 面向授权管理的资源管理模型研究[D]. 解放军信息工程大学, 2011.

[61] YU S, WANG C, REN K, et al. Attribute based data sharing with attribute revocation[C]// Proceedings of the 5th ACM symposium on information, computer and communications security. 2010: 261-270.

[62] LIU Q, WANG G, WU J. Time-based proxy re-encryption scheme for secure data sharing in a cloud environment[J]. Information sciences, 2014, 258:355-370.

[63] ZHANG P, CHEN Z, LIANG K, et al. A cloud-based access control scheme with user revocation and attribute update[C]//Australasian Conference on Information Security and Privacy. Springer, 2016: 525-540.

[64] SHI Y, ZHENG Q, LIU J, et al. Directly revocable key-policy attribute-based encryption with verifiable ciphertext delegation[J]. Information Sciences, 2015, 295:221-231.

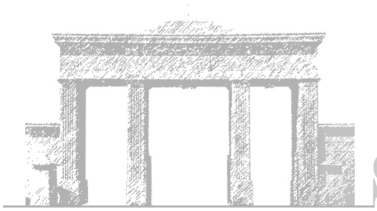
作者简介

李轩（1995.4.26 -），男，河北石家庄人，现为东南大学网络空间安全学院硕士研究生，主要研究方向为用户隐私数据保护、区块链安全。

作者攻读硕士学位期间发表的论文

1. **李轩**, 宋宇波, 王润, 武威, 胡爱群. The Taint Tracking in Mobile Web App to Detect XSS Vulnerability[C]. 第 12 届信息安全漏洞分析与风险评估大会 (VARA), 江苏无锡,2019.7.

心求至善



SOUTHEAST UNIVERSITY