软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(7):1661−1675 [doi: 10.3724/SP.J.1001.2011.03870]

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

多域环境下基于属性的授权委托模型*

吴 槟[1,3+], 冯登国[1,2]

[1](中国科学院 研究生院 信息安全国家重点实验室,北京 100049)

[2](中国科学院 软件研究所 信息安全国家重点实验室,北京 100190)

[3](信息安全共性技术国家工程研究中心,北京 100190)

Attribute-Based Authorization Delegation Model in Multi-Domain Environments

WU Bin[1,3+], FENG Deng-Guo[1,2]

[1](State Key Laboratory of Information Security, Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[2](State Key Laboratory of Information Security, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[3](National Engineering and Research Centre of Information Security, Beijing 100190, China)

+ Corresponding author: E-mail: wubin@is.iscas.ac.cn, http://home.is.ac.cn

Wu B, Feng DG. Attribute-Based authorization delegation model in multi-domain environments. *Journal of Software*, 2011,22(7):1661−1675. http://www.jos.org.cn/1000-9825/3870.htm

Abstract: Traditional identifier-based authorization models are limited to having different authorization paths that will cause the inconsistency in propagation of permission. In addition, unauthorized entities may acquire illegal permission by such an identifier-based authorization path. In order to solve these two problems, an attribute-based authorization delegation model (ABADM), suitable for multi-domain environments is presented. In the ABADM model, the delegation of authority and the propagation of permission are all based on the attribute sets of entities, which ensure that the entities on the same credential chain have the same permission. The model integrates attribute-permission assignation policies inside autonomic domains and the interdomain attributes mapping model. The algorithm for calculating the attribute sets and permissions of entities in the multi-domain environments is proposed. The usage of the ABADM model is illustrated through a common example.

Key words: cross-domain authorization; attribute-based delegation; propagation of permission; trust management

摘 要: 传统的基于实体标识的授权模型会导致由于授权路径的不一致而引起权限传播的不一致,并可能导致不合法的实体进入授权路径获得不应有的权限.针对以上两方面的问题,提出一个基于属性的授权委托模型 ABADM (attribute-based authorization delegation model),将授权模型中权限的传递与权力的委托均建立在实体所具有的属性集合的基础之上,以属性集合作为实体自身的代表进行授权,从而确保拥有相同属性凭证链的实体其权限是一致的.模型将属性与访问权限进行明确区分,提出了域内属性权限分配策略和域间属性计算模型,通过两者的结合给出了在 ABADM 模型中计算实体所拥有的跨域属性集合和访问权限的方法,并结合具体实例对模型的使用进行了说明.

关键词: 跨域授权;基于属性的委托;权限传播;信任管理

* 基金项目: 国家自然科学基金(60803129); 下一代互联网业务试商用及设备产业化专项(CNGI-09-03-03)
收稿时间: 2009-08-03; 修改时间: 2010-03-11; 定稿时间: 2010-04-14

中图法分类号: TP309 文献标识码: A

随着国际互联网的日益发展,越来越多的组织开始联合起来进行协同工作.这种协同存在于多种领域之中,例如军事领域中的联合演习、卫生领域中的合作医疗、教育领域中的教育资源共享以及商业领域中的有偿数据与服务交换.在以上协同环境中,各组织内部独立管理,也称为自治域或管理域;每个域都可以看作是一个自主、独立的实体;每个域通过共享其管理的资源和服务及各域之间的交互访问操作实现协同.

由于各组织间的协同工作变得日益频繁且规模不断扩大,特别是各自治域间需要动态地进行数据或服务的交换,传统的基于实体标识的授权方式暴露出了较多的问题和安全隐患.这一方面是由于在 Internet 这样的开放式环境中各协同工作的自治域是彼此陌生的,在协同关系建立之时难以迅速找到公认的可信第三方以验证和确立彼此间的真实身份,此时,通过实体标识进行授权不仅无法精确地限制被授权方所获得的权力,而且有可能导致不合法的实体进入授权路径获得不应有的权限,影响权限传播的安全性与可控性;另一方面,由于授权过程密切依赖于实体标识,因此会导致由于权力来源的传递路径不一致而引起权限分配与传播控制的不一致.为了解决基于实体标识的授权方式所带来的以上两个方面的问题,我们需要在大规模开放式环境中采用基于属性的授权方式实施有效授权.

信任管理是一种当前流行的基于实体标识的跨域授权方案[1−4],其核心的授权机制是通过委托来实现的,即一个实体可以将分配成员的权力分配给另一实体.而进行访问控制判决的过程即为寻找一条从权力源到资源请求者的权力委托的凭证链(也称为凭证路径)的过程,文献[4]称其为凭证链发现问题.例如,在基于角色的信任管理语言 RT[4,5]中,当进行跨域授权时,域 A 通过颁发授权凭证 $A.r_1 \leftarrow B.r_2$ 将为角色 $A.r_1$ 分配成员的权力委托给了域 B;B 又可以通过签发 $B.r_2 \leftarrow D$ 来使 D 获得 $A.r_1$,由此形成了一条跨域授权的凭证链:$A.r_1 \leftarrow B.r_2 \leftarrow D$.在此授权过程中,$A$ 将为角色 $A.r_1$ 分配成员的权力委托给另一实体(本例中是域 B)的过程是基于实体标识的,该委托潜在地假设域 A 对域 B 非常地了解,并且对域 B 中的角色分配和对应的权限也精确地掌握,而这与大规模开放环境下各域彼此陌生、各域结构与安全策略存在较大差异的实际情况是矛盾的.在不清楚域 B 中的结构、行为与授权策略的情况下,域 A 仅基于实体标识就将权力直接委托给域 B,这种授权方式虽然简便易行,但却是非常不安全的(例如域 B 可能向其他不合法的实体授权,使其获得 $A.r_1$ 对应的权限).在合理的授权方式中,授权方必须对被授权方做出必要的限制,一方面保证被授权方不会做出影响授权方安全的行为,另一方面也保证被授权方具有授权方所要求的性质.而这正是基于属性的授权方式所具有的优势[5−7],即授权方的权力不是依据实体标识进行委托,而是将权力委托给具有一定属性集合的实体,凡是满足该属性集合要求的实体均可获得此种权力.在此种授权方式下,从授权方 A 的角度来看,B 的标识并不能代表实体 B 本身的结构、行为与内部的安全策略,而仅是一个名称的代号,实体的真实概念只有通过对该实体的属性进行刻画才能得以重新建立,能够代表实体自身的只有其所具有的属性集合 ω.因此,本文提出将多域环境下权限的传递与权力的委托建立于实体所具有的属性集合的基础之上,以属性集合 ω 作为实体自身的代表,以保证授权委托模型的安全性.

而从权限传递的角度来看,在基于实体标识的授权方式中,由于系统中的各域具有独立、自主的协作关系,为防止本域访问控制策略的泄漏,资源提供者不可能向请求者详细披露本域的属性/权限分配,而这种属性/权限分配信息的缺失,也会造成远端的资源请求者通过不同的凭证路径取得属性后,由于凭证签发者的不同和授权路径的不一致,获得的权限也不一致.例如,上例中的资源请求者 D 可能会通过两条凭证链 $A.r_1 \leftarrow B_1.r_2 \leftarrow D$ 与 $A.r_2 \leftarrow B_2.r_2 \leftarrow D$ 同时获得域 A 中的权限,但两条权限传播路径会导致 D 获得的权限不一致,造成跨域资源访问的权限传播控制出现问题.本文采用的基于属性的授权方式能够有效防止此类权限分配与传播控制的不一致性.在上例中,从授权方 A 的角度来看,若 B_1 与 B_2 具有相同的属性集合 ω,则资源请求者 D 获得域 A 中权限的凭证链只有一条:$A.r_1 \leftarrow \omega.r_2 \leftarrow D$.因此,基于属性的授权方式保证了权限传播的一致性与授权路径的可控性,从而提高了系统的安全性.

本文针对以上两个方面的问题,提出了一个适合开放式环境的基于属性的授权委托模型(attribute-based authorization delegation model,简称 ABADM),解决了不合法的实体进入授权路径获得不应有的权限和凭证链

路径不一致所导致的访问权限不一致问题,有助于实现 Internet 开放式环境下网络资源的访问控制与权限分配,为构建面向互联网资源协同共享的跨自治域计算环境提供了有力的授权技术支持.该模型完全采用基于属性的授权委托机制,使权力的委托与跨域属性的获得不再通过被委托者的标识,而是依据被委托者是否具有所要求的属性,保证了权限传播的安全性和可控性.由于在凭证链发现过程中没有任何实体标识信息,从而保证拥有相同属性的不同实体授权路径仅有一条,不会出现凭证链路径不一致所导致的访问权限不一致问题.同时,通过构造一种基于属性的环签名凭证(attribute-based ring signatures credential,简称 ABRSC),可以在不需要签发者标识的前提下证明资源请求者拥有所要求的属性.本文对 RT_0[4]语言进行了扩展,使其可以支持 ABRSC 凭证,并提出了一种在扩展后的授权框架中进行凭证发现与扩展的算法.

1 相关工作

实现异构系统间的资源共享与互操作,最初始于数据库领域.Sheth 等人[8]首先提出了联合系统的一个必然特性是单个系统的自治,即每个系统必须独立地进行管理.Gong 等人[9,10]通过安全级别交互映射实现多级安全数据库之间的互操作,解决存在的冲突,提供最大安全互操作方案.但该方案属于 NP 完全的集中式算法,是一种适用于封闭环境和已知群体的约束,难以应用于开放式的、基于未知用户群体的环境.Dawson 等人[11]讨论了在实施基于格的访问控制策略的系统间建立安全互操作的框架,通过策略中介所具有的全局视角实现系统间策略比较、映射等操作.以上工作的主要缺点是可扩展性较差,成员之间要求保持长期、稳定的协同关系,不适合开放式、基于未知用户群体的协同.此外,基于格的访问控制策略多用于军事环境中,因此,以上工作较少应用于商业环境中.

基于角色的访问控制(role-based access control,简称 RBAC)[12]通过引入角色的概念实现了用户和权限的逻辑分离,权限被授予角色,用户通过分配的角色获得相应的权限,显著地降低了授权管理的代价,在商业环境中已成为授权管理的主要方式.文献[13–15]对 RBAC 模型中的委托进行了探讨,并提出了各自的约束机制和实施方案.然而,以上工作采用的约束实施机制大都假设存在一个或者几个中心节点对系统中新发布的委托凭证进行合法性验证,而这是与开放式环境下的各域独立和用户群体未知的特性相冲突的.开放式环境中的每个域都是独立的自治实体,控制一组资源,并可以向原本陌生的用户发布自己的委托授权凭证,很难形成可信的第三方或中心节点在委托凭证发布时进行合法性检查,而只能是在凭证链发现过程中动态地进行检查.此外,这些工作的委托约束大都直接指定后继委托的被委托者的标识,延展性较差且与开放环境下用户群体未知的特性相冲突(后继委托的参与者是不可预知的),因此极大地限制了在实际系统中的应用.

为了实现大规模开放式环境下的跨域授权,Blaze 等人[2]最早提出了信任管理框架,用于描述基于本地策略 P 来判断资源请求 r 是否允许被授权的结构.PolicyMaker[2,3]和 KeyNote[1]是该框架的代表性系统.这些系统给出了策略语法规范,但缺少严格的语义解释和推理规则,也不支持基于属性的授权委托机制,其权力的委托过程需要基于被委托者的标识,因此无法满足实际场景中复杂的跨域授权应用场景,也缺乏良好的扩展性和广泛的适用性.为了进一步获得更好的语义解释和逻辑推理能力,Li 等人[6]为 SPKI/SDSI[16,17]提供了一阶逻辑的语义,并将该语义进一步扩展用于策略概念和策略分析请求的表达.之后,Li 等人[4,5]结合 Delegation Logic[7]与 RBAC 模型[12]的优势提出了信任管理语言 RT.进而,Winsborough 等人[18]基于 RT,给出了一个信任协商协议 TTG.Mao 等人[19]进一步给出了带有参数化角色和逻辑约束的语言 RT_1^C 的分布式凭证链发现算法,增强了原有 RT 语言的灵活性和表达力.SPKI/SDSI,Delegation Logic 和 RT 等虽然支持基于属性的授权委托机制,但并未考虑权限传播控制的问题,在各自治域内属性/权限分配关系不可见的情况下,其授权委托过程仍然是基于签发者标识信息的.因此,在访问控制决策中仍然会出现由于凭证签发者的不同和凭证扩展路径的不同造成资源请求者获得权限不一致的情况,并且无法进行有效的检测和去除.

2 基于属性的授权委托模型 ABADM

在跨域资源共享访问中,对资源进行访问操作的主体(如人、进程、代表进程的密钥等)称为资源请求者或

资源访问者.向资源请求者提供资源进行访问的主体,称为资源提供者或资源拥有者.每个资源提供者均自治地管理一定的资源供资源请求者访问,称这些用于共享访问的资源为对象,资源请求者向被访问的对象所施加的访问行为称为操作.在开放式环境下,资源提供者向环境中默认的所有实体(而不仅仅是隶属于该资源提供者的实体)提供自己所管理的对象进行访问,并由资源提供者对访问请求进行判定.资源请求者是否拥有对该对象实施某种操作的权限,则依赖于 ABADM 模型中的两个重要部分:自治域内属性权限分配策略和域间属性计算模型.其中,前者用于描述单个自治域内属性权限之间的对应关系,资源拥有者根据权限计算函数和资源请求者所拥有的属性集合,计算出请求者在本域内的权限;而后者则通过授权凭证刻画了域间的属性映射关系以及权力的委托路径.通过域间属性计算模型,资源请求者将自身的属性映射到资源提供者所在域的属性集合,进而通过资源提供者所在域的权限计算函数计算出在该域内的权限.计算域间属性映射关系的过程,等效于跨域环境下的授权委托凭证链发现问题.因此,域间属性计算模型的核心就是凭证链的扩展与发现算法,该算法即本文第 3 节将要描述的 *ABProofGraph* 算法.

ABADM 将属性与访问权限进行明确的区分,克服了 RT 中由于两者混淆所带来的无法对权限传递实施约束控制的缺陷,符合了真实环境下各域内属性/权限分配关系不可见的情况.在 RT 等传统的信任管理中,将属性(或 RT 中的角色)等同于权限,忽略了两者之间的区别,目的是有利于信任管理引擎进行一致性验证.然而,这种方式也带来了一定的缺陷:首先,属性/权限分配通常代表了本自治域的访问控制策略,这种信息在本域内是无条件可信的,而对于其他域通常并不是可见的,即使可见也并不是完整的和无条件可信的.因此,传统方式不仅会增加不必要的凭证链发现和验证的代价,而且会造成域内策略的泄漏;其次,从授权管理的角度看,某一自治域的属性结构在较长时间内是相对不变的,而属性对应的一组权限的集合则是变化的,如果将属性与访问权限混淆,由于安全机制的不同和策略的差异,要求某一自治域对另一域中的属性/权限分配情况做出动态、精确的分析,其计算代价是巨大的,也是不现实的,将难以对跨域的权限传递实施约束控制.因此,本文借鉴了文献[20]的思想,对属性与访问权限进行了明确的区分,将属性/权限分配看作是本域的策略,并且该分配关系对外域而言是不可见的,由自治域内的属性/权限分配策略负责根据实体所拥有的属性集合,计算出实体在本域内的权限;而在跨域访问的环境下,则使用域间属性计算模型来建立实体/属性分配,远端的实体通过委托凭证链取得本域中的属性,再通过本域的访问控制策略获得访问权限.

2.1 自治域内属性权限分配策略

定义 1(模型基础元素). *ATTRS*,*OPS*,*OBS* 分别代表本自治域内的属性集、操作集、对象集.$PERMS\subseteq 2^{(OPS\times OBS)}$ 为权限集,其中,×为笛卡尔积操作.

定义 2(属性/权限分配). $PA\subseteq \wp(ATTRS)\times PERMS$ 是本自治域内的属性/权限分配,其中,$\wp(ATTRS)$是 *ATTRS* 的幂集(power set).

设$(\{attr_1,attr_2\},p)\in PA$,则拥有属性集合$\{attr_1,attr_2\}$的实体可以行使权限 *p*.

定义 3(属性层次). $DAH\subseteq ATTRS\times ATTRS$,是本自治域内属性间的直接支配关系.*AH* 是 *DAH* 的自反、传递闭包,代表了属性层次.

属性层次上的祖先属性继承后代属性的所有权限.即 $(attr_1',attr_1)\in AH$ 且$(attr_1,p)\in PA$,则 $(attr_1',p)\in PA$.

定义 4(属性集合层次). $ASH\subseteq \wp(ATTRS)\times \wp(ATTRS)$,是属性集合间的层次关系.对于两属性集合 as_1,as_2,若$\forall attr_2\in as_2,(\exists attr_1\in as_1:(attr_1,attr_2)\in AH)$,则$(as_1,as_2)\in ASH$.

属性集合层次上的祖先属性集合继承后代属性集合的所有权限.即$(as',as)\in ASH$ 且$(as,p)\in PA$,则$(as',p)\in PA$.

定义 5(权限计算函数). 函数 $authorizedPerms(as':\wp(ATTRS))$计算拥有属性集合 *as*′的实体在本自治域内的权限:

$$authorizedPerms(as')=\{(p)|\exists as:\wp(ATTRS),(as',as)\in ASH\wedge(as,p)\in PA\}.$$

本节描述了单个自治域内属性权限分配的策略.通过权限计算函数,资源所在域根据实体所拥有的属性集合计算出实体在本自治域内的权限.一个完整的授权模型还需要解决另一个问题:来自其他域的陌生实体如何通过域间的属性映射关系取得本域的属性集合,而这由域间属性计算模型来解决.

2.2 域间属性计算模型

定义 6(域间模型基础元素). *ENTITYS* 代表跨域模型中的实体集.在域间属性计算模型中,各域均为 *ENTITYS* 集中的一个实体.

ATTRNAMES 代表所有实体中属性名称的并集,*CDATTRS* 代表域间属性集.在域间属性计算模型中,各自治域中 *ATTRS* 的元素称为属性名称(或属性名),而域间属性则是由实体名后加属性名称组成的,即

$$CDATTRS=\{A.attr|A\in ENTITYS,attr\in ATTRNAMES\}.$$

定义 7(授权凭证及类型). *CREDS* 代表实体集 *ENTITYS* 中的实体所拥有的凭证集.其具体形式如下:

- I 型凭证:$A.attr\leftarrow B$.

其中,*A* 和 *B* 是实体名,*attr* 是属性名.该凭证的含义是,*A* 定义 *B* 获得 *A.attr* 属性,或 *A* 承认 *B* 具有属性 *attr*,即 $B\in members(A.attr)$.其中,*members*(*e*)代表 *e* 中所有成员实体的集合.它通常用于定义本域中的实体/属性分配或定义外部某实体与本域属性的关系.

- II 型凭证:$A.attr_1\leftarrow A.attr_2$.

其中,*A* 是实体名,$attr_1$ 和 $attr_2$ 是两个不同的属性名.该凭证的含义是 *A* 定义所有获得 $A.attr_2$ 属性的成员都可以取得 $A.attr_1$ 属性的成员身份,即 $members(A.attr_1)\supseteq members(A.attr_2)$.或者可理解为如果 *A* 承认某实体具有属性 $attr_2$,那么 *A* 就承认该实体具有属性 $attr_1$.该凭证说明,属性 $A.attr_2$ 比 $A.attr_1$ 更强,因为 $A.attr_2$ 的成员可以做 $A.attr_1$ 所被授权的任何事.

- III 型凭证:$A.attr\leftarrow A.attr_1.attr_2$.

其中,*A* 是实体名,*attr*,$attr_1$ 和 $attr_2$ 是属性名.称 $A.attr_1.attr_2$ 为链接属性.

该凭证的含义是 $members(A.attr)\supseteq members(A.attr_1.attr_2)=\bigcup_{B\in members(A.attr_1)} members(B.attr_2)$,或者可理解为,如果 *A* 承认某实体 *B* 具有属性 $attr_1$,并且 *B* 承认某实体 *D* 具有属性 $attr_2$,那么 *A* 就承认 *D* 具有属性 *attr*.

通过 III 型凭证即可实现基于属性的委托(attribute-based delegation):*A* 将 *B* 作为属性 $attr_2$ 的权威,但并不需要知道 *B* 的身份,而是通过 *B* 所具有的属性(即 $A.attr_1$).

- IV 型凭证:$A.attr\leftarrow f_1\cap f_2\cap\ldots\cap f_k$.

其中,*A* 是实体名,*k* 是大于 1 的整数,每个 $f_j(1\leqslant j\leqslant k)$或者是以 *A* 为开始的属性(如 *A.attr*),或者是以 *A* 为开始的链接属性($A.attr_1.attr_2$).称 $f_1\cap f_2\cap\ldots\cap f_k$ 为属性交集.

该凭证的含义是 $members(A.attr)\supseteq(members(f_1)\cap members(f_2)\cap\ldots\cap members(f_k))$.

- V 型凭证:$A.attr'\leftarrow[A.attr_1\cap A.attr_2\cap\ldots\cap A.attr_k].attr$.

其中,*A* 是实体名,*attr*,*attr*′,$attr_1$,$attr_2$,…,$attr_k$ 是属性名.*k* 是大于 1 的整数,称$[A.attr_1\cap A.attr_2\cap\ldots\cap A.attr_k].attr$ 为交集链接属性.

该凭证的含义是 $members(A.attr')\supseteq\bigcup_{B\in members(A.attr_1)\cap\ldots\cap members(A.attr_k)} members(B.attr)$,或可理解为,如果 *A* 承认某实体 *B* 具有属性集合 $attr_1$,$attr_2$,…,$attr_k$,并且 *B* 承认某实体 *D* 具有属性 *attr*,那么 *A* 就承认 *D* 具有属性 *attr*′.

- VI 型凭证: $[A.attr_1].attr_2\leftarrow D$.

该凭证的含义是 $D\in members(A.attr_1.attr_2)=\bigcup_{B\in members(A.attr_1)} members(B.attr_2)$,或者可非形式化地理解为,*A* 定义某实体 *B* 获得 $A.attr_1$ 属性,并将 *B* 作为属性 $attr_2$ 的权威,而 *B* 又承认某实体 *D* 具有属性 $attr_2$.由此,*D* 获得链接属性 $A.attr_1.attr_2$,并且 *A* 不知道 *B* 的身份.

特别地,称链接属性 $A.attr_1.attr_2$ 中的第 2 项 $attr_2$ 为第 2 链接属性,$A.attr_1$ 为第 1 链接属性,*A* 为第 1 链接属性签发者;并为其定义一种特别的第 2 链接属性 *self*,用于表示凭证右端的实体,即代表拥有 $attr_1$ 属性的实体自身.即 $D\in members(A.attr_1.self)=\bigcup_{B\in members(A.attr_1)} members(B.self)=members(A.attr_1)$.

- VII 型凭证:$[A.attr_1\cap A.attr_2\cap\ldots\cap A.attr_k].attr\leftarrow D$.

其中,k 是大于 1 的整数,$[A.attr_1 \cap A.attr_2 \cap \ldots \cap A.attr_k].attr$ 为交集链接属性.与 VI 型凭证相似,称 A 为第 1 链接属性签发者,$A.attr_1 \cap A.attr_2 \cap \ldots \cap A.attr_k$ 为第 1 链接属性,$attr$ 为第 2 链接属性.

该凭证的含义是 $D \in \bigcup_{B \in members(A.attr_1) \cap \ldots \cap members(A.attr_k)} members(B.attr)$, 或者可非形式化地理解为,$A$ 定义某实体 B 拥有 $A.attr_1 \cap A.attr_2 \cap \ldots \cap A.attr_k$ 属性,并将 B 作为属性 $attr$ 的权威,而 B 又承认某实体 D 具有属性 $attr$.由此,D 获得交集链接属性$[A.attr_1 \cap A.attr_2 \cap \ldots \cap A.attr_k].attr$,并且 A 不知道 B 的身份.与 VI 型凭证相似,交集链接属性中的第 2 链接属性也具有特别属性 *self*.

凭证中的实体、属性、链接属性、属性交集、交集链接属性统称为属性表达式(简写为 *expression* 和 *e*).

在以上 7 种凭证之中,I 型~V 型凭证为私钥签发凭证,即对凭证 $A.attr \leftarrow e$,其密码学形式可描述为 $[e \mid attr]_{sk_A}$ 或 $[e \mid A.attr]_{sk_A}$.其中,sk_A 为实体 A 的私钥.称 A 为签发者,e 为凭证体,$base(e)$中的每一个实体称为凭证的主体.其中,函数 $base(e)$定义如下:

$$base(A)=\{A\},$$

$$base(A.attr)=\{A\},$$

$$base(A.attr_1.attr_2)=\{A\},$$

$$base(f_1 \cap \ldots \cap f_k)=base(f_1) \cup \ldots \cup base(f_k),$$

$$base([f_1 \cap \ldots \cap f_k].attr)=base(f_1) \cup \ldots \cup base(f_k).$$

VI 型、VII 型凭证可以采用私钥签发并附加属性凭证的方式,也可以采用基于属性的环签名凭证(ABRSC)的方式.采用环签名的目的是不泄露凭证签发者的身份,在凭证验证时仅能确定是由“环”中的成员所签发的;而基于属性的环签名则是将拥有一组相同属性集合ω的实体组成了一个环,该环的公钥即为属性集合ω.私钥 sk_ω 则根据属性集合ω生成,由该环签名私钥所签发的凭证即可认为是拥有属性集合ω的实体所签发,并且任何人都无法获知该凭证是该环中的哪一实体所生成的.除该环成员以外,任何实体均无法伪造该签名.文献[21]提供了一种可证明安全的基于属性的环签名方案,用于构造 ABRSC 凭证,从而保证在不泄露签发者标识的前提下证明资源请求者拥有所要求的属性.VI 型、VII 型 ABRSC 凭证可统一描述为$[\omega].attr \leftarrow D$,其中,$\omega$为签发者所拥有的属性集合,$D$ 为凭证的主体,$attr$ 为主体所拥有的属性.其密码学形式可描述为 $[\omega \mid attr \mid D \mid para]_{sk_\omega}$,其中,$para$ 为环签名的公共参数.

在 ABADM 模型的授权过程中,每个域自主地定义自己的授权委托凭证.每个凭证都负责为本域中的属性指定成员实体集合(I 型、II 型凭证),或将这种指定成员集合的权力委托给满足一定条件的外域实体(III 型~V 型凭证).对于前者,在将相应的属性凭证签发给成员实体的同时,也将该属性集合ω的私钥 sk_ω传递给成员实体;而对于后者,在对权力进行委托时,必须要指明被委托者所具有的属性.当外域的实体要映射为本域中的属性集合时,则通过构造 VI 型、VII 型凭证向属性签发者证明该实体拥有所要求的中间属性集合,并通过递归地扩展凭证链,最终映射为所要求的属性集合.计算该映射关系的过程,等效于授权委托凭证链的发现问题.因此,域间属性计算模型的核心就是凭证链的扩展与发现算法,即本文所提出的 *ABProofGraph* 算法.该算法是一种目标指引式的凭证链发现算法,它通过构建一个证明图(proof graph)来实现在分布式的凭证集合中发现满足要求的凭证链,使外域中的实体获得向资源提供者所在域的域内属性进行映射的路径.

3 授权委托凭证链的发现

3.1 *ABProofGraph*算法

为了在复杂的非线性凭证图结构中迅速而有效地发现授权委托凭证链,ABADM 模型采用了 *ABProofGraph* 算法.*ABProofGraph* 算法扩展了文献[4]中的 *ProofGraph* 算法,添加了对基于属性的授权方式,特别是 ABRSC 凭证的支持,并增加了针对交集链接属性节点的链接监视器支持.*ABProofGraph* 的核心思想是:由初始节点开始扩展证明图,对每一个当前处理的节点通过其所对应的凭证确定其所关联的属性;对于新确定的属性签发者,则利用新发现的凭证进行关联属性的扩展操作,以导出边的形式将该属性扩展成为另一实体的属

性成员节点(该扩展即基于属性的授权委托).随着扩展过程的进行,证明图不断添加新的节点,并递归地生成新的 ABRSC 凭证边,直至最终获得从初始节点至目标属性节点之间的一条路径,该算法结束.

证明图中的节点代表凭证集合中的属性表达式,算法开始时只有初始和目标两个节点,通过计算不断向图中添加新的节点;而证明图中的边则主要包含如下 5 种类型:

(1) 凭证边 $e \rightarrow A.attr$,e 是属性表达式,代表实体、属性、链接属性、属性交集或交集链接属性.它是由 I 型~V 型凭证 $A.attr \leftarrow e$ 直接得到的.在本文图中采用实线箭头加以表示;

(2) 导出边 $A.attr \rightarrow B.attr_1.attr$.该种边不是直接由凭证得到的,而是由凭证路径 $B.attr_1 \xleftarrow{*} A$ 导出的.由于该边并非凭证集合中实际存在的凭证,因此称为导出边,在本文图中采用点划线箭头加以表示;

(3) 导出边 $A.attr \rightarrow [B.attr_1 \cap \ldots \cap B.attr_k].attr$.它不是直接由凭证得到的,而是由凭证路径 $B.attr_1 \cap \ldots \cap B.attr_k \xleftarrow{*} A$ 导出的.该种类型的边也称为导出边,在本文图中同样采用点划线箭头加以表示;

(4) 导出边 $e \rightarrow f_1 \cap f_2 \cap \ldots \cap f_k$.它不是直接由凭证得到的,而是由凭证图中的其他凭证路径导出的,这些路径包括 $f_1 \xleftarrow{*} e$, $f_2 \xleftarrow{*} e$,..., $f_k \xleftarrow{*} e$.该种类型的边也称为导出边,在本文图中采用点划线箭头加以表示;

(5) ABRSC 凭证边 $D \rightarrow e.D$ 是实体;e 是属性表达式,代表属性、链接属性或交集链接属性.它是由 I 型、VI 型或 VII 型凭证直接得到的,在本文图中采用虚线箭头加以表示.

在证明图中通过以上 5 种类型的边,形成了一条从资源请求者(即初始节点 D)至资源提供者的特定属性(目标属性节点 $A.attr$)的映射路径,该路径即 $D \xrightarrow{*} A.attr$ 的凭证链.根据该凭证链和 A 中的权限计算函数,即可计算出实体 D 在本自治域 A 内的权限.

附录中给出了 *ABProofGraph* 算法的细节.

3.2 凭证链的构建与扩展过程

例 1 中给出了一个虚构的教育资源共享场景中所使用的授权委托凭证集合.该场景中,某地的大学在当地教育部门的牵头下结成协作联盟关系,隶属于联盟中的大学应开放本校的某些教学服务给该联盟中大学的学生.为了防止教学服务被不属于该联盟的其他学校的人员随意访问,由牵头的教育部门发布联盟中的大学名单.为了简明而又不失一般性,本文假设该场景中的访问过程涉及了两所大学和一个教育部门共 3 个自治域,Alice 和 Bob 则分别是两所大学的学生.下面我们通过 *ABProofGraph* 算法对以上场景中的跨域访问过程进行凭证链的构建与扩展,生成最终的证明图;并通过该例说明 ABADM 模型克服了基于实体标识的授权方式中存在的凭证链路径不一致所导致的访问权限不一致问题,保证了权限传播的安全性与一致性.

例 1:教育资源共享.

自治域:*universityA*,*universityB*,*bureau*.

在授权过程中,各自治域所颁发的凭证集合如下:

***universityA* 颁发**:

{*universityA.student*←Alice (1) //Alice 是 *universityA* 的学生

universityA.AllyLeader←*bureau* (2) //*bureau* 是 *universityA* 参与联盟的牵头人

universityA.eduserve←*universityA.AllyLeader.UniStudent* (3) //牵头人所承认大学的学生可以访问本域的教学服务

}

***universityB* 颁发**:

{*universityB.student*←Bob (4) //Bob 是 *universityB* 的学生

universityB.AllyLeader←*bureau* (5) //*bureau* 是 *universityB* 参与联盟的牵头人

universityB.eduserve←*universityB.AllyLeader.UniStudent* (6) //牵头人所承认大学的学生可以访问本域的教学服务

}

***bureau* 颁发**:

{*bureau.ally*←*universityA* (7) //*universityA* 是 *bureau* 牵头的联盟的成员

bureau.university←*universityA* (8) //*universityA* 是 *bureau* 承认的大学

bureau.ally←*universityB* (9) //*universityB* 是 *bureau* 牵头的联盟的成员

bureau.university←*universityB* (10) //*universityB* 是 *bureau* 承认的大学

bureau.UniStudent←[*bureau.ally*∩*bureau.university*].*student* (11) //既参加 *bureau* 牵头的联盟又是大学的实体,其下属学生将拥有 *UniStudent* 属性.

}

以上的 11 个凭证中共有 I 型凭证 8 个,分别由各 I 型凭证的主体存储;其余的凭证则存储于签发者端.

当 *universityA* 域的学生 Alice 访问 *universityB* 的教学服务时,需要构造一条从 Alice 生成的密钥 K_{Alice} 到 *universityB* 中的目标属性集合{*universityB.eduserve*}的凭证链.当 Alice 生成代表自己的密钥 K_{Alice} 时,需要发布初始凭证

$$\text{Alice}.self \leftarrow K_{\text{Alice}} \tag{0}$$

同时,由于 Alice 拥有属性凭证(1)以及 *universityA.student* 属性所对应的私钥 $sk_{\{universityA.student\}}$,因此 Alice 构造 ABRSC 凭证

$$[universityA.student].self \leftarrow K_{\text{Alice}} \tag{12}$$

并将凭证(12)发送至该凭证的第 1 链接属性签发者 *universityA*.该凭证的含义为密钥 K_{alice} 代表拥有 *universityA.student* 属性的实体(本例中即 Alice)自身.*universityA* 收到凭证(12)后,会通过 *ABProofGraph* 算法进行扩展,并根据扩展生成的中间节点,利用其所拥有的属性凭证(7)、凭证(8)和对应的私钥 $sk_{\{bureau.ally,bureau.university\}}$ 构造 VII 型凭证

$$[bureau.ally \cap bureau.university].student \leftarrow K_{\text{Alice}} \tag{13}$$

在将凭证(13)发送至第 1 链接属性签发者 *bureau* 后,*bureau* 通过 *ABProofGraph* 算法进行扩展,并根据扩展生成的中间节点,利用其所拥有的属性凭证(5)和对应属性的私钥 $sk_{\{universityB.AllyLeader\}}$ 构造 VI 型凭证

$$[universityB.AllyLeader].UniStudent \leftarrow K_{\text{Alice}} \tag{14}$$

当 Alice 通过密钥 K_{Alice} 所建立起来的信道将凭证(14)发送至 *universityB* 后,通过与凭证(6)结合,即可最终获得链 $universityB.eduserve \xleftarrow{*} K_{\text{Alice}}$,通过该目标属性集合{*universityB.eduserve*}进一步获得访问 *universityB* 的教学服务的权限.

上述凭证链的构建与扩展过程最终生成的证明图如图 1 所示.证明图中的节点(0)代表 *ABProofGraph* 算法中的初始节点,节点(*)则代表目标节点,节点的标号代表了算法中节点添加的顺序,边的标号则代表了例 1 中对应的凭证序号.

与学生 Alice 的凭证链构建及扩展过程相似,当来自 *universityB* 域的学生 Bob 访问 *universityB* 的教学服务时,通过 *ABProofGraph* 算法也可以构建从密钥 K_{Bob} 至目标属性集合{*universityB.eduserve*}的授权委托路径.与 Alice 的证明图相似,节点(0)代表 *ABProofGraph* 算法中的初始节点,节点(*)代表目标节点,节点的标号代表了节点添加的顺序,边的标号则代表了对应的凭证序号,最终生成的 Bob 的证明图如图 2 所示.

通过对比来自两个不同自治域中的学生 Alice 和 Bob 的凭证链构建与扩展过程以及最终生成的证明图,可以看到:对于资源拥有者 *universityB*,无论资源请求者来自哪个自治域,只要访问相同的 *universityB* 教学服务,其证明图结构都是相同的,说明凭证链的构建与扩展过程的步骤都是相同的,只是个别节点所代表的自治域会有所区别.同样,资源请求者所提供的从初始节点至目标节点的授权委托路径也是相同的,并且与该路径上的实体标识无关.以上特性保证了在 ABADM 模型中,拥有相同属性集合的不同实体在授权委托路径中的作用是平等的,并且经由相同授权委托路径所获得的权限也是一致的.

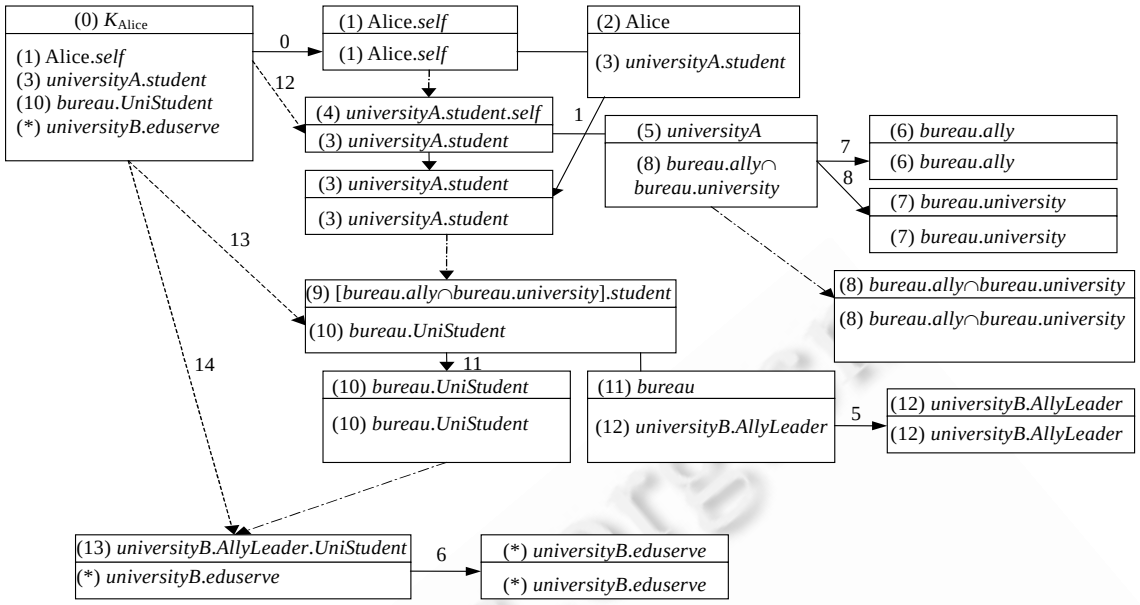

Fig.1 Final proof graph for Alice in example 1

图 1 例 1 中生成的 Alice 的证明图

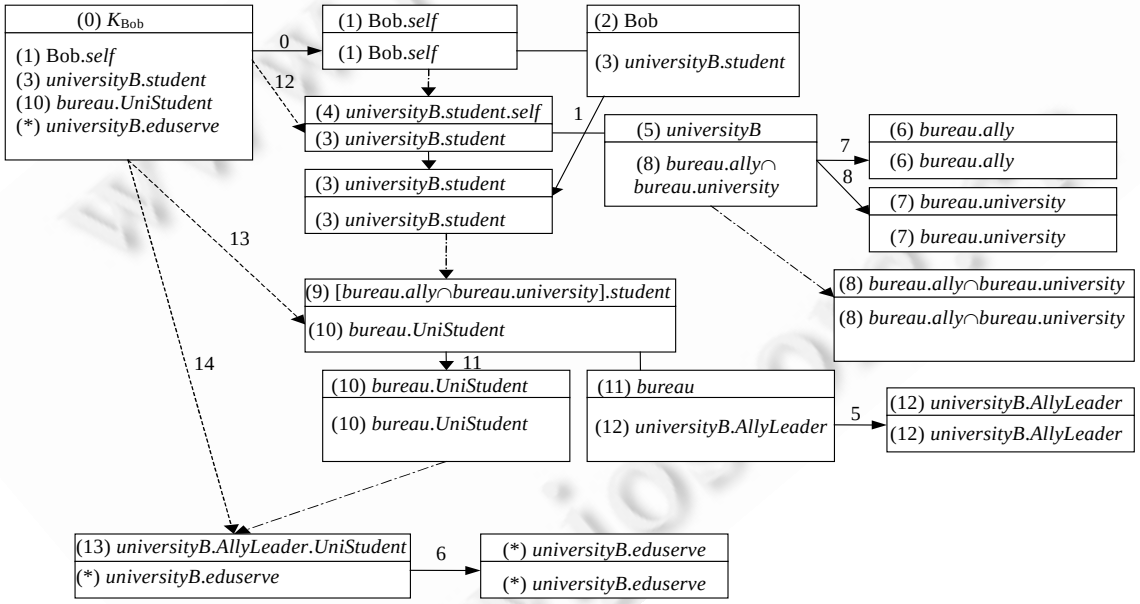

Fig.2 Final proof graph for Bob in example 1

图 2 例 1 中生成的 Bob 的证明图

4 模型的安全性质

在 ABADM 模型的跨域资源访问过程中,资源请求者首先向远端的域提交资源访问请求(*op*,*object*),资源所在域查找包含了该访问权限的本地属性集合,然后使用 *ABProofGraph* 算法计算该资源请求者是否能够取得该

属性集合,从而确定该请求者是否能够行使该权限.

定理 1. 某资源请求实体 e 能够获得本域访问权限 p 当且仅当证明图中存在某路径集合 l 证明他可以取得本域的属性集合 as,满足 $p\in authorizedPerms(as)$.

证明:必要性证明.e 获得权限 p 的唯一方法是通过取得本域中包含了 p 的属性集合 as,且根据定义 5 满足 $p\in authorizedPerms(as)$.通过反证法,假设证明图中不存在这样的路径集合,则必然出现:(1) e 无法取得相关属性集合;或(2) e 所取得的属性集合 as'不满足 $p\in authorizedPerms(as')$.以上两种情况都会导致 e 无法获得访问权限 p,而这与已知前提矛盾.因此,路径集合 l 不存在的假设是错误的.必要性得证.

充分性证明.如果证明图中存在该路径集合 $l, l=\{A.attr_1 \xleftarrow{*} e, A.attr_2 \xleftarrow{*} e,..., A.attr_n \xleftarrow{*} e\}$,则对属性节点集合 $as=\{A.attr_1,A.attr_2,\ldots,A.attr_n\}$中的每一个属性节点执行 *b-activate*()函数,同时对节点 e 执行 *f-activate*()函数,然后通过 *ABProofGraph* 算法的扩展,根据文献[4]的定理 7,e 最终会成为 $as=\{A.attr_1,A.attr_2,\ldots,A.attr_n\}$中每一个属性节点的后向解,$A.attr_1,A.attr_2,\ldots,A.attr_n$ 节点也都会成为节点 e 的前向解.由属性集合 as,根据定义 5 计算 *authorizedPerms*(as),则 e 即可获得访问权限 p.充分性得证. □

5 结束语

传统的信任管理通过委托的方式为开放式环境下的跨域授权提供了一种灵活、有效的解决方案,但是,由于大多数模型都是基于实体标识的授权方式,没有很好地解决不合法的实体可能通过委托进入授权路径获得不应有的权限以及因凭证路径不一致所导致的访问权限不一致这两大安全问题.针对以上缺陷,本文依据开放式系统中各实体彼此陌生这一基本情况,将授权模型中权限的传递与权力的委托均建立于实体所具有的属性集合的基础之上,提出了一个基于属性的授权委托模型 ABADM.通过基于属性的授权委托,使授权过程中的权力委托可以传递给符合相应属性集合要求的陌生实体,而无需依据该实体的标识;并通过此种基于属性的委托所形成的属性凭证链,确保拥有相同的属性凭证链的实体其权限是一致的.此外,从自治域的权限计算对外域不可见的角度,将属性与访问权限进行明确区分,对 ABADM 模型中的域内属性权限分配策略和域间属性计算模型进行了形式化描述,通过添加对基于属性的授权委托方式的支持,给出了计算实体所拥有的跨域属性集合的 *ABProofGraph* 算法,从而实现基于属性控制其所能获得的访问权限.ABADM 实现了一种无需实体标识的对权限传播实施控制的授权委托方案,有助于精确地实现 Internet 环境下网络资源的访问控制.

致谢 在此,我们向本文的评审专家表示感谢,感谢他们对本文所提出的宝贵意见和建议.

References:

[1] Blaze M, Feigenbaum J, Ioannidis J, Keromytis A. The KeyNote trust-management system, version 2. IETF RFC 2704, 1999.

[2] Blaze M, Feigenbaum J, Lacy J. Decentralized trust management. In: Proc. of the '96 IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society Press, 1996. 164−173. http://doi.ieeecomputersociety.org/10.1109/SECPRI.1996.502679 [doi: 10.1109/SECPRI.1996.502679]

[3] Blaze M, Feigenbaum J, Strauss M. Compliance checking in the PolicyMaker trust management system. In: Hirschfeld R, ed. Proc. of the 2nd Int'l Conf. on Financial Cryptography. LNCS 1465, Berlin: Springer-Verlag, 1998. 254−274. [doi: 10.1007/BFb0055488]

[4] Li NH, Winsborough WH, Mitchell JC. Distributed credential chain discovery in trust management. Journal of Computer Security, 2003,11(1):35−86.

[5] Li NH, Mitchell JC, Winsborough WH. Design of a role based trust management framework. In: Heather H, ed. Proc. of the IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society Press, 2002. 114−130. [doi: 10.1109/SECPRI.2002.1004366]

[6] Li NH, Mitchell JC. Understanding SPKI/SDSI using first-order logic. Int'l Journal of Information Security, 2006,5(1):48−64. [doi: 10.1007/s10207-005-0073-0]

[7] Li NH, Grosof BN, Feigenbaum J. Delegation logic: A logic-based approach to distributed authorization. ACM Trans. on Information and System Security (TISSEC), 2003,6(1):128−171. [doi: 10.1145/605434.605438]

[8] Sheth AP, Larson JA. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys (CSUR), 1990,22(3):183−236. [doi: 10.1145/96602.96604]

[9] Gong L, Qian XL. The complexity and composability of secure interoperation. In: Rushby J, Meadows C, eds. Proc. of the IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society Press, 1994. 190−200. [doi: 10.1109/RISP.1994.296581]

[10] Gong L, Qian XH. Computational issues in secure interoperation. IEEE Trans. on Software and Engineering, 1996,22(1):43−52. [doi: 10.1109/32.481533]

[11] Dawson S, Qian S, Samarati P. Providing security and interoperation of heterogeneous systems. Distributed and Parallel Databases, 2000,8(1):119−145. [doi: 10.1023/A:1008787317852]

[12] Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. Role-Based access control models. IEEE Computer, 1996,29(2):38−47. [doi: 10.1109/2.485845]

[13] Wainer J, Kumar A. A fine-grained, controllable user-to-user delegation method in RBAC. In: Ferrari E, Ahn GJ, eds. Proc. of the 10th ACM Symp. on Access Control Models and Technologies. New York: ACM Press, 2005. 59−66. [doi: 10.1145/1063979.1063991]

[14] Bandmann O, Dam M, Firozabadi BS. Constrained delegation. In: Proc. of the 23rd Annual IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society Press, 2002. 131−143. http://doi.ieeecomputersociety.org/10.1109/SECPRI.2002.1004367 [doi: 10.1109/SECPRI.2002.1004367]

[15] Zhang LH, Ahn GJ, Chu BT. A rule-based framework for role-based delegation. In: Sandhu RS, Jaeger T, eds. Proc. of the 6th ACM Symp. on Access Control Models and Technologies. New York: ACM Press, 2001. 153−162. [doi: 10.1145/373256.373289]

[16] Clarke D, Elien JE, Ellison C, Fredette M, Morcos A, Rivest RL. Certificate chain discovery in SPKI/SDSI. Journal of Computer Security, 2001,9(4):285−322.

[17] Ellison C, Frantz B, Lampson B, Rivest RL, Thomas B, Ylonen T. SPKI certificate theory. IETF RFC 2693, 1999.

[18] Winsborough WH, Li NH. Safety in automated trust negotiation. ACM Trans. on Information and System Security (TISSEC), 2006, 9(3):352−390. [doi: 10.1145/1178618.1178623]

[19] Mao ZQ, Li NH, Winsborough WH. Distributed credential chain discovery in trust management with parameterized roles and constraints (short paper). In: Ning P, Qing SH, Li NH, eds. LNCS 4307, Berlin, Heidelberg: Springer-Verlag, 2006. 159−173. [doi: 10.1007%2f11935308_12]

[20] Zhai ZD, Feng DG, Xu Z. Fine-Grained controllable delegation authorization mode l based on trustworthiness. Journal of Software, 2007, 18(8):2002−2015 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/2002.htm [doi: 10.1360/jos182002]

[21] Wang WQ, Chen SZ. An efficient attribute-based ring signature scheme. In: Zhou QH, ed. Proc. of the Int'l Forum on Computer Science-Technology and Applications. Chongqing: IEEE Computer Society Press, 2009. 147−150. [doi: 10.1109/IFCSTA.2009.43]

附中文参考文献:

[20] 翟征德,冯登国,徐震.细粒度的基于信任度的可控委托授权模型.软件学报,2007,18(8):2002−2015. http://www.jos.org.cn/1000-9825/18/2002.htm [doi: 10.1360/jos182002]

附 录

算法 1. *ABProofGraph* 算法.

ABProofGraph 采用面向对象风格的描述方法,包含了两个基本类:*ABProofGraph* 和 *ABProofNode*,以及 3 个辅助类:*ABBLinkingMonitor*,*ABBIntersectionMonitor*,*ABFlinkingMonitor*.

ABProofGraph 类的实例变量定义:

Class *ABProofGraph*

{

initial-node:证明图初始节点;
target-node:证明图目标节点;
nodes:证明图节点;
edges:证明图中的边;
b-proc-queue:后向处理队列;
f-proc-queue:前向处理队列;
}
ABProofNode 类的实例变量定义:
Class *ABProofNode*
{
b-proc-state:后向处理状态; //共 3 种状态取值:*unprocessed*(未处理),*to-be-processed*(加入队列,即将处理),*processed*(已处理)
f-proc-state:前向处理状态; //与后向处理状态取值相同
b-solutions:后向解集合;
f-solutions:前向解集合;
b-sol-monitors:后向解监视器集合;
f-sol-monitors:前向解监视器集合;
}
各类的方法定义:
ABProofGraph: ***run***()
{
while (one of *b-proc-queue* and *f-proc-queue* is not empty)
{if (*b-proc-queue* nonempty)
{*n=b-proc-queue.dequeue*();*n.b-process*();}
if (*f-proc-queue* nonempty)
{*n=f-proc-queue.dequeue*();*n.f-process*();}
if (*initial-node.f-solutions.contains*(*target-node*))
{return;} //获得从初始节点至目标属性节点之间的映射路径,算法结束
}
}
ABProofGraph: ***addNode***(***e***)
{if (a node *n* exists for *e*) {return *n*;}
else {creates one and return it;}}
ABProofGraph: ***addinitialNode***(***e***)
{*n=addNode*(*e*);
initial-node=n;
}
ABProofGraph: ***addtargetNode***(***e***)
{*n=addNode*(*e*);
target-node=n;
}
ABProofGraph: ***find-f-sol-entity***(***n***)

```
{//递归算法寻找节点 n 对应的前向解实体集合
set={};
if (n is an entity node) {set.add(n); return set;}
foreach (n1 in n.f-sol-monitors){
    set=set.union(find-f-sol-entity(n1));      //对前向解实体集合求并集
    }
return set;
}
ABProofGraph: addEdge(e2←e1)
{n1=addNode(e1);n2=addNode(e2);
if (edges.contains(n2←n1))) {return;}          //如该边已存在,则返回
edges.add(n2←n1);
if (e2 is a linked attribute B.attr1.attr2 or an intersection linked attribute [B.attr1∩B.attr2∩…∩B.attrk].attr)
{
    f-sol-entity-set=find-f-sol-entity(n1);    //寻找 e1 对应的前向解实体集合
    foreach (n in f-sol-entity-set) {
        edges.add(n2←n);                       //添加 ABRSC 凭证边
        n.add-b-sol-monitor(n2);               //添加 ABRSC 凭证边后向解监视器
        if (n2.b-proc-state!=unprocessed) {n.b-activate();}
        n2.add-f-sol-monitor(n);               //添加 ABRSC 凭证边前向解监视器
        if (n.f-proc-state!=unprocessed) {n2.f-activate();}
        }
    return;
}
n1.add-b-sol-monitor(n2);                      //添加后向解监视器
if (n2.b-proc-state!=unprocessed) {n1.b-activate();}
n2.add-f-sol-monitor(n1);                      //添加前向解监视器
if (n1.f-proc-state!=unprocessed) {n2.f-activate();}
}
ABProofNode(A.attr): b-process()
{b-proc-state=processed;
creds=find all credentials defining A.attr;
foreach (credential "A.attr←e" in creds)
    {addNode(e);addEdge(A.r←e);}}
ABProofNode(A.attr1.attr2): b-process()
{b-proc-state=processed;n=addNode(A.attr1);
n.add-b-sol-monitor(new ABBLinkingMonitor(A.attr1.attr2));
n.b-activate();}
ABBLinkingMonitor(A.attr1.attr2): add-b-solution(B)
{addNode(B.attr2);addEdge(A.attr1.attr2←B.r2);}
ABProofNode(D): b-process()
{b-proc-state=processed;add-b-solution(D);}
```

ABProofNode($f_1 \cap f_2 \cap \ldots \cap f_k$): ***b-process***()
{*b-proc-state=processed*;
m=new *ABBIntersectionMonitor*($f_1 \cap f_2 \cap \ldots \cap f_k$);
foreach (*j* in 1,...,*k*)
{*n=addNode*(f_j);*n.add-b-sol-monitor*(*m*);*n.b-activate*();}}
ABBIntersectionMonitor($f_1 \cap f_2 \cap \ldots \cap f_k$): ***add-b-solution***(***B***)
{if (*B* is being added for the first time) {*n=addNode*(*B*);*n.f-activate*();}
else if (*B* has been added *k* times) {*addEdge*($f_1 \cap f_2 \cap \ldots \cap f_k$,*B*);}}
ABProofNode(***e***): ***f-process***()
{
f-proc-state=processed; //标明该节点已被处理
if (*e* is an attribute $B.attr_2$)
{*add-f-solution*($B.attr_2$);*n=addNode*(*B*);
n.add-f-sol-monitor(new *ABFLinkingMonitor*($B.attr_2$));
n.f-activate();}
creds=find all credentials having *e* as its body at *base*(*e*);
foreach (credential $A.attr \leftarrow e$) {
addNode(*A.attr*);*addEdge*($A.attr \leftarrow e$);}
if (*e* is an intersection) {return;}
creds=find all credentials contain $f_1 \cap f_2 \cap \ldots \cap f_k$ in which $f_j=e$ for some *j*;
foreach (intersection $f_1 \cap f_2 \cap \ldots \cap f_k$ in *creds*)
{*add-f-solution*($\langle f_1 \cap f_2 \cap \ldots \cap f_k, j \rangle$); //为属性交集添加部分解
n=addNode($f_1 \cap f_2 \cap \ldots \cap f_k$);
n.b-activate();}
}
ABFLinkingMonitor(***B.attr***$_2$): ***add-f-solution***(***A.attr***$_1$)
{*addNode*($A.attr_1.attr_2$);
addEdge($A.attr_1.attr_2 \leftarrow B.attr_2$); //由 $A.attr_1 \leftarrow B$ 生成导出边
}
ABFLinkingMonitor(***B.attr***): ***add-f-solution***(***A.attr***$_1 \cap \ldots \cap$ ***A.attr***$_k$)
{*addNode*([$A.attr_1 \cap \ldots \cap A.attr_k$].*attr*);
addEdge([$A.attr_1 \cap \ldots \cap A.attr_k$].$attr \leftarrow B.attr$); //生成导出边
}
ABProofNode(***e***): ***b-activate***()
{if (*b-proc-state*!=*unprocessed*) {return;}
b-proc-state=to-be-processed; *b-proc-queue.enqueue*(*this*);
foreach (*n* such that *edges.contains*($this \leftarrow n$))
{*n.b-activate*();}}
ABProofNode(***e***): ***f-activate***()
{if (*f-proc-state*!=*unprocessed*) {return;}
f-proc-state=to-be-processed; *f-proc-queue.enqueue*(*this*);
foreach (*n* such that *edges.contains*($n \leftarrow this$))

{*n*.*f-activate*();}}

ABProofNode(***e***): ***add-b-solution***(***s***)

{if (*s* exists in *b-solutions*) {return;}

b-solutions.*add*(*s*);

foreach (*n* in *b-sol-monitors*)

{*n*.*add-b-solution*(*s*);} //将后向解向后传播

}

ABProofNode(***e***): ***add-f-solution***(***s***)

{if (*s* exists in *f-solutions*) {return;}

f-solutions.*add*(*s*);

foreach (*n* in *f-sol-monitors*) {*n*.*add-f-solution*(*s*);}}

if (*e* is an entity *D* && *s* is a partial solution ($f_1 \cap f_2 \cap \ldots \cap f_k j$) && all *k* pieces have arrived)

{*addEdge*($f_1 \cap f_2 \cap \ldots \cap f_k \leftarrow D$);}} //生成导出边

ABProofNode(***e***): ***add-b-sol-monitor***(***n***)

{*b-sol-monitors*.*add*(*n*);

foreach (*s* in *b-solutions*) {*n*.*add-b-solution*(*s*);}}

ABProofNode(***e***): ***add-f-sol-monitor***(***n***)

{*f-sol-monitors*.*add*(*n*);

foreach (*s* in *f-solutions*) {*n*.*add-f-solution*(*s*);}}

吴槟(1980－),男,山东青岛人,博士生,主要研究领域为网络与信息系统安全.

冯登国(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络与信息安全.