计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(11):180−185 [doi: 10.15888/j.cnki.csa.006635]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

可变动 RBAC 模型的密钥管理研究①

蒋　凡[1], 魏弋翔[2], 程绍银[1]

[1](中国科学技术大学 信息安全测评中心, 合肥 230027)
[2](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
通讯作者: 程绍银, E-mail: sycheng@ustc.edu.cn

摘　要: 访问控制在一个信息安全系统中是一个基础的课题. RBAC (基于角色的访问控制模型, Role-Based Access Control) 以不同的角色来定义用户, 这些角色对应了不同的密级. 这使得不同角色中的用户有不同的权限. 基于这一点, 密钥可以用来区分不同角色间的访问权限. 随着人事和任务的变动, 现有 RBAC 的结构也会发生变动. 本文定义了线性、树形和有向无环图三类 RBAC 模型, 从线性结构出发, 讨论角色中用户与密级发生的变化, 推广至树形结构, 提出了一种下级角色的密钥由上级角色的密钥决定的方法, 可以有效地实现线性和树形可变动 RBAC 模型的密钥管理.

关键词: 访问控制; 密钥管理; 线性多层结构; 可扩展方法

引用格式: 蒋凡,魏弋翔,程绍银.可变动 RBAC 模型的密钥管理研究.计算机系统应用,2018,27(11):180–185. http://www.c-s-a.org.cn/1003-3254/6635.html

Key Management for Changeable RBAC System

JIANG Fan[1], WEI Yi-Xiang[2], CHENG Shao-Yin[1]

[1](Information Technology Security Evaluation Center, University of Science and Technology of China, Hefei 230027, China)
[2](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: RBAC model is a solution which defines users with different roles, and the roles are in different classes which mean the users with different roles have different permission. Usually, we can use secret keys to discriminate the different roles. However, the role in this access control model is a security class including some users. Changes will appear in this system with personnel changes frequently. Due to the keys are corresponding to the roles, how to update the keys in these frequent changes is the focus of this study. There are three kinds of model in RBAC, the linear model, the tree model, and Directed Acyclic Graph (DAG). This paper discusses the changes of users and security class from the linear and tree model. The problem in the method where the inferior keys are determined by the superior keys is also discussed. Thus, key management for changeable RBAC system is effectively realized.

Key words: access control; key management; linear hierarchies; extensible method

1　引言

访问控制是信息安全中最为基础的方面之一[1], 而 RBAC 正是一种解决办法. 在 RBAC 中, 一个多级结构被定义为拥有不同权限的不同角色的集合, 将用户映射到角色, 给角色赋予权限, 把用户与权限的直接联系舍弃, 通过角色来联系与管理[2]. NIST 标准 RBAC 模型分为 RBAC0 (core RBAC, 核心 RBAC)、RBAC1 (hierarchy RBAC, 分级 RBAC)、RBAC2 (constraint RBAC, 约束 RBAC) 和 RBAC3 (combine RBAC, 组合 RBAC) 四个子模型[3], 其中在 RBAC1 中引入了角色的

① 收稿时间: 2018-03-22; 修改时间: 2018-04-18; 采用时间: 2018-05-11; csa 在线出版时间: 2018-10-24

继承关系. 这个标准模型统一了人们对于 RBAC 的认识[4]. 于是 RBAC 模型可以被抽象为具有支配关系的集合, 有限继承的 RBAC1 模型正是线性以及树形结构的多层结构. 在此基础上, 可以使用密钥来区分不同角色, 以达到权限与角色对应的目的. 这个需求会出现在大部分的多层结构组织的访问控制系统中. 在这样的系统中的密钥管理面临的一大挑战性的问题就是如何保证角色能获得自身的密钥[5]和期望每一个处于这个系统中的用户都能从它自身的密钥计算出它的下级集合的密钥, 即便是在频繁变化的系统中. 系统的变动是使用中的必然, 容易预见, 用户的角色会发生变化, 每一个角色也有可能会发生变化 (例如增加角色、删除角色, 权限的升级与降级). 所以密钥管理应该在保证安全性的同时高效地生成或改变密钥. 如何在系统发生改变时做出响应即是本文章讨论的重点[6].

为了更容易地理解这个场景, 想象有一个多职位的企业. 其中每一个职位都会有下辖的职位, 例如总经理下有销售经理等. 这样一来, 不同职位对应着不同的角色. 显然, 不同角色对应着不同的访问权限. 这样一来就出现了一个问题: 当某职位有人员调动时, 该角色的密钥应当发生改变, 否则本不该知道密钥的、不属于该角色的成员仍能掌握该密钥.

真正核心的问题在于, 如何保证一个祖先节点能够推测出密级远低于它的节点的密钥[7]. 更进一步地, 如果一个用户离开了这个系统或者离开了其中某个部门, 要想使得该用户原节点之下的信息不再被该用户获知, 则这些节点的密钥都需要作出相应更改.

在这篇文章中, 我们提出了一种新的高效的密钥管理方法. 这篇文章按以下结构组织: (1) 在第 2 节中介绍了该模型的多层结构和密钥分配方法; (2) 在第 3 节中, 对已有的密钥管理方法做了一个回顾; (3) 在第 4 节中, 介绍了我们的模型并阐述了理论基础; (4) 在第 5 节对该方法进行安全性证明与实验; (5) 最后对本文进行了总结.

2 问题定义

2.1 密级模型

正如在前文提到过的一样, 在这个访问控制模型中最基本的是“角色”, 每个角色对应一个密级. 可以如下定义这个问题[8].

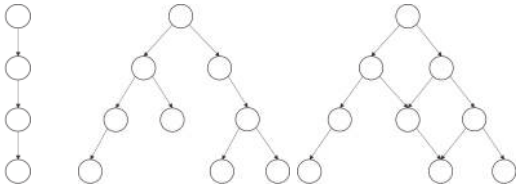
这些实体根据密级被分割为不同的子组, 即成为角色. 每个角色是用户的一个集合,对应了一个密级, 将其称为 SC_i. 这样在定义中, 每个 SC_i 就是一些实体的集合, 所有的 SC_i 组成的集合称作 SC. 现在我们可以定义 SC 的偏序关系. 对于两个类 SC_i 和 SC_j, 若 SC_i 对 SC_j 具有访问权限, 则称 SC_j 从属于 SC_i, 以≤表示这个关系. 这样定义之后有以下几个特性[9]:

1) 传递性: 假设有三个不同的密级集合, SC_i,SC_j,SC_k, 如果有 $SC_i \leq SC_j$ 且 $SC_j \leq SC_k$, 那么 $SC_i \leq SC_k$.

2) 自反性: 对任何密级集合 SC_i 都有 $SC_i \leq SC_i$.

3) 非对称性: 如果 $SC_i \leq SC_j$ 且 $SC_j \leq SC_i$ 那么必定有 $i=j$. 话句话说, $SC_i \leq SC_j \cap SC_j \leq SC_i \Leftrightarrow SC_i$ 和 SC_j 是对应于同一角色的密级集合.

如果在一张图上表示这个结构, 可以明显地看出三种构型: 线性, 树形和有向无环图 (DAG), 如图 1.所示. 在一般情况下, 从有向无环图到线性结构是一个逐步特化的过程. 但是线性结构与树形结构和有向无环图有很大的区别, 因为线性与树形结构的每一个节点均只有一个直接支配节点. 在应用上来说, 这两种结构更符合现实中的情景, 和有限继承的 RBAC 模型契合, 所以本文着重讨论这线性与树形两种结构.



(a) 线性结构 (b) 树形结构 (c) 有向无环图

图 1 三种结构类型示意图

2.2 密钥分配

在一开始, 需要为每个角色分别指定一个初始的密钥. 方法如下[10]:

(1) 为每个角色分别指定一个互不相同的质数. 对于每个 $v \in V$, 选择一个与其他质数 p 均不同的质数 p_v.

(2) 利用这些质数, 通过如下方法, 为每个角色生成一个标记 (token).

$$t_v = \begin{cases} 1 & if\ A_v = V \\ \prod\limits_{u \notin A_v} p_u & \text{otherwise} \end{cases} \quad (1)$$

(3) 随机选择两个不同的大质数 p 和 q, 计算 $n = p \cdot q$. 然后选择根密钥 k_0 使得 $1<k_0<n$.

(4) 对于每一个 $v \in V$, 计算出 $k_v = k_0^{t_v} \bmod n$. 其中 A_v 表示从属于 v 的所有密级集合的集合.

这个标记的优点在于, 非常简单地标记了不同角色之间的从属关系. 对于角色 $SC_i \leq SC_j (i \neq j)$, 一定有 $t_j | t_i$. 这一点很好证明.

对于两个密级集合, 假设 $SC_i \leq SC_j$, 那么若 $A_j = V$, 则 $t_j = 1$, 那么显然 $t_j | t_i$; 否则, $t_i = t_j \cdot \prod_{u \notin A_i \text{且} u \in A_j} p_u$, 得证.

但是, 上述有两点问题. 第一点是计算出的 t_v 可能非常大[11]. 另一点是, 所有的密钥都与 k_0 有紧密的联系, 因此当密钥系统发生了改变时, 不管是角色还是角色中的用户改变, 几乎都必须从根向下更新一遍. 这意味着每当变化发生, 密钥将被重新分配. 这篇文章关注的重点则是如何减少变化发生时的修改次数.

3 KTLHs

Hassen 等人提出的密钥管理方法利用了额外空间来存储密钥表. 他们将此方法称为“对线性多层结构的基于密钥表的密钥管理方法”(KTLHs)[12]. 第一次使用 KTLHs 时, 它会随机生成一个 K_1, 然后每一个 K_{t+1} 都能使用一个单向哈希函数 H 来生成, $K_{t+1} = H(K_t)$. 这样一来, 对于每一个 SC_t 的成员来说, 任意密级集合 SC_u ($SC_u \leq SC_t$) 的密钥可以通过 $u-t$ 次哈希函数来得到. 密钥表将记录下每个密级集合已经得到的密钥以及版本号, 用 K_t^p 来表示密级集合 SC_t 在进行过第 $p-1$ 次更新后的密钥. 更新密钥时, 将会进行以下操作:

3.1 加入/离开

以密级集合 SC_t 发生了改变为例.

(1) 控制中心随机生成一个新的密钥 K_t^{p+1}.

(2) 对每个满足 $SC_c \leq SC_t$ 的 SC_c, 使用单向函数 H 计算出 K_c^{p+1} 并发送至相应的集合.

(3) 对每个满足 $SC_t \leq SC_s$ 的 SC_s, 发送消息对 (t, K_t^{p+1}) 至对应集合.

(4) 每个密级集合更新自己的密钥表.

考虑一个有 5 个密级集合的系统, 其中 SC_3 的一位成员离开系统. 那么 SC_1 的成员所持有的密钥表在运行了上述算法后将如表 1 所示. 这样, 若 SC_1 的成员想要访问 SC_5 的资源, 只需要运行两次 $H(K_3^2)$.

3.2 升级/降级

这个情形是 SC_u 的成员升级或降级至 SC_t, 为了方便描述, 不妨设 $SC_u \leq SC_t$.

表 1 成员持有的密钥表

节点	持有节点	Key
SC_1	1	$K_1^2 (=K_1^1)$
	3	K_3^2
SC_2	2	$K_2^2 (=K_2^1)$
	3	K_3^2
SC_3	3	K_3^2
SC_4	4	K_4^2
SC_5	5	K_5^2

(1) 控制中心随机生成一个新的密钥 K_t^{p+1}.

(2) 然后对每个满足 $SC_{u-1} \leq SC_c \leq SC_{t+1}$ 的 SC_c, 使用单向函数 H 计算出 K_c^{p+1} 并发送至相应的集合.

(3) 满足 $SC_t \leq SC_s$ 的 SC_s 将会收到消息对 (t, K_t^{p+1}); 满足 $SC_{u-1} \leq SC_s$ 的 SC_s 将会收到消息对 (u, K_u^{p+1}).

(4) 每个密级集合更新自己的密钥表.

3.3 密级集合变化

密级集合只会通过两种方式发生改变, 即增加和删除. 这个改变可以视为在改变的节点处做了加入/离开操作. 举例来说, 当 SC_t 被添加的时候, 使原本的 SC_t 降级为 SC_{t+1}, 然后对 SC_t 做 SC_t 上的相应的加入/离开操作.

3.4 空间消耗增多的情形

当连续且不同的改变发生时, 密级集合的密钥表可能表得非常大, 使得这个方法接近或等价于“非依赖方法”. 比如角色中的成员自上而下发生改变时. 在之前举例的情境中, 则是从 SC_2, SC_3 直至 SC_5 发生改变. 在这些改变发生完成后, SC_1 的密钥表将分别记录下所有密级集合的密钥. 如表 2 所示.

4 本文方法

本文提出了一种新的密钥管理方法, 引入参数的密钥管理方法 (Parameters Table-based key management scheme for Hierarchies, PTH). 它在保持与“依赖方法”相同的获取下级密钥的速度的同时比 KTLHs 时的存储空间改变更小.

4.1 模型结构

这个模型是依赖模型的一个改进模型, 在计算下级密钥方面有相同的复杂度, 在系统的变动发生时更新次数有很大的减少. 每个角色仅存储自身的密钥和一个参数, 这个参数用来参与计算下级密钥. 这样保证了系统所消耗的空间与系统的角色数是成正比的, 并

且该比例是稳定的.

表 2 极端情形下成员持有的密钥表

节点	持有节点	Key
SC_1	1	$K_1^5(=K_1^1)$
	2	$K_2^5(=K_2^2)$
	3	$K_3^5(=K_3^3)$
	4	$K_4^5(=K_4^4)$
	5	K_5^5
SC_2	2	$K_2^5(=K_2^2)$
	3	$K_3^5(=K_3^3)$
	4	$K_4^5(=K_4^4)$
	5	K_5^5
SC_3	3	$K_3^5(=K_3^3)$
	4	$K_4^5(=K_4^4)$
	5	K_5^5
SC_4	4	$K_4^5(=K_4^4)$
	5	K_5^5
SC_5	5	K_5^5

4.2 线性模型

如图 2 所示, 每个角色存储它自己的密钥和参数 i 两个信息.

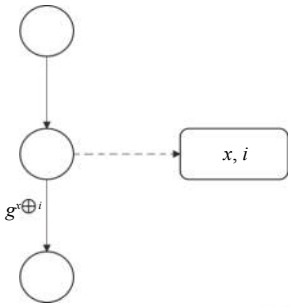


图 2 在变化时发生的更新

K_0 作为根密钥由中心控制器直接随机生成.

对于每个不是根节点的角色, $K_t=H(K_{t-1}, i_{t-1})$, $H(K, i)=g^{K+i}$.

当 SC_t 的成员发生变动时, 以下操作会被执行:

1) SC_t 选择一个随机数 r, 计算出 $K_t^{new}= K_t \cdot g^r$.

2) 计算并更新 i_t^{new}.

3) 上级密钥的参数 i_{t-1}^{new} 更新.

算法 1. 结构的建立与用户变动

Data: 由 CA 生成的 K_0
Result: 各自保存自身密钥的 RBAC 模型

1 initialization;
2 **foreach** 非根节点角色 **do**
3 $K_t=H(K_{t-1},i_{t-1})$;
4 **end**
5 **while** SC_t 发生变动 **do**
6 SC_t 选择随机数 r;
7 计算 $K_t^{new}=K_t \cdot r$;
8 $i_t^{new}=(K_t \oplus i_t) \oplus K_t^{new}$;
9 $i_{t-1}^{new}=(K_{t-1} \oplus i_{t-1}+r) \oplus K_{t-1}$;
10 **end**

可以看出, 每一个变动的只对邻近的角色的有影响, 而且只更新其参数. 每一次变动只有两个密钥集合的信息会被更新.

算法 2. 角色变动

Data: 由 CA 生成的 K_0
Result: 各自保存自身密钥的 RBAC 模型

1 initialization;
2 **while** SC_t 删除 **do**
3 SC_t 的直接指派关系移交至其直接父亲 (假设为 SC_{t-1});
4 $i_{t-1}^{new}=K_t \oplus i_t \oplus K_{t-1}$;
5 **end**
6 **while** SC_t 插入子节点 SC_t+1 **do**
7 SC_t 的直接指派关系移交至 SC_{t+1} ;
8 设置新参数 i_t^{new};
9 计算$K_{t+1}=H(K_t, i_t^{new})$;
10 $i_{t+1}^{new}=K_{t+1} \oplus K_t \oplus i_t$;
11 $i_{t-1}^{new}=K_t \oplus i_t \oplus K_{t-1}$;
12 **end**

在实现与测试中, 密钥与参数进行异或运算作为单向函数的参数, 可以保证密钥与参数的规模在使用过程中保持不变.

4.3 树形模型

因为线性多层结构和树形多层结构有一个重要的共同点, 即都只有一个直接的上级节点, 所以树形多层结构可以被看作是线性多层结构的一个扩展. 有一点区别在于, 树形结构由于子节点更多, 所以需要更多的空间来存储参数序列. 这样, 下级节点的密钥计算如图 3 所示.

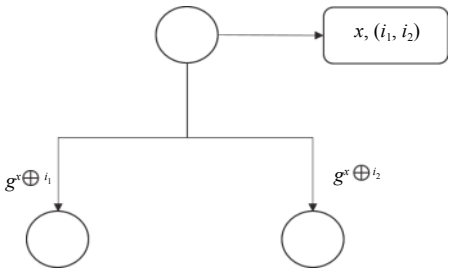


图 3 计算并更新密钥

5 安全性证明及实验

5.1 安全性

定理 1. 这个更新方法保证了一个密级集合不能够获取上级集合的密钥.

证明. 假设有一个函数 Dec 可以通过给定的信息输出上级集合的密钥. 不妨设该函数形为:

$$\mathrm{Dec}(x', g, p) = x, \text{其中} x' = g^{(x+i)}. \quad (2)$$

1) 这里, 该公开模 p 循环群, g 为其生成元.

2) Alice 选择一个随机数 a 作为密钥. 那么公钥则是 (g, g^a, p).

3) 在这个情境下, Dec 则输出 a–i. 而其中 $i = 0$.

这等价于利用 Elgamal 加密算法的公钥输出了其私钥, 而这在目前是公认计算不可行的. 证明完毕.

这个结论的成立也是非常直观的. 对于不同的 x 和 i, 算出的 x'是有可能相同的. Dec 不可能在仅知道 x'的情况下输出确切的 x. 由于这个方法有和 Hassen 等人提出的方法同样的更新时机, 这个方法也满足他们提出的两个安全特性.

5.2 计算复杂度

如果用 N 来代表密级集合的个数 (也即角色数), 用 u_i 表示密级集合 SC_i 的成员数量. SC_i 中成员的变动概率为 p. 那么这个系统发生的变动总数为:

$$\sum_{i=1}^{N} p_i \cdot u_i \quad (3)$$

虽然没有一个特定的规则或公式去求得具体的 p 和 u, 但是显而易见的是, 越下级的 SC_i 的 p 和 u 倾向于越大. 不妨简单地假设 p_i=k_i, u_i=m_i, 其中 k 和 m 是两个指定的参数. 在本文的方法中, 每次变动的更新次数是一致的, 可以算出 $T = \mathrm{O}(N)$; 而在密钥表模型中, T 是不确定的, 在一些极端情况下, 可能出现 $T = \mathrm{O}(N^2)$. 可以说, 密钥表方法是使用了更多的表更新次数换取了更少的获取下级密钥的时间.

5.3 真实场景模拟

基于密钥表的密钥管理几乎只能用于一个纯线性的结构中. 在纯线性的模型中, KTLH 的消耗与角色数呈正相关. 在同样的变动次数下, 其响应时间受系统中角色数的影响. 如图 4 所示.

可以看出, 由于本文提出模型的单向函数基于离散对数, 而处于安全考虑, 其循环群的阶数得较大, 所以在系统初始化时的构建中耗时较长. 但是在其后的频繁变动中, 由于每次独立变动的耗时几乎相同, 其消耗仅与变动次数相关. 在大规模的多角色多层级的系统中, 相较于 KTLH 将拥有更高的效率.

在角色变动中, 有类似的表现. 如图 5 所示.

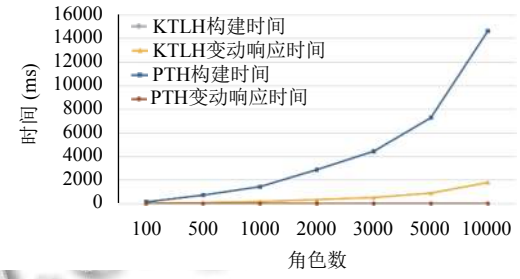


图 4 线性结构用户变动响应效率对比

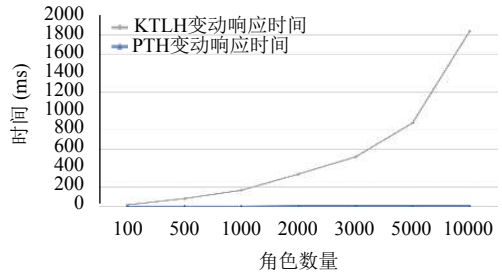


图 5 线性结构角色变动响应效率对比

6 结论与展望

本文介绍了一个可以被用于基于角色的多层系统中的新的密钥管理方法, 这个方法满足 Hassen 等人提出的安全要求. 除了在线性的多层结构中有良好的表现, 这个密钥管理方法可以轻松的扩展至树形结构. 这个方法保持了与依赖方法推测下级密钥相同的复杂度, 并且降低了在变动发生时的更新次数. 该方法的优点在于, 对于一个可能频繁变动的系统来说, 其消耗仅与变动次数相关, 可以更好得应用于一个多角色多层级的大规模系统当中.

参考文献

1 Garrison WC, Shull A, Myers S, *et al*. On the practicality of cryptographically enforcing dynamic access control policies in the cloud. Proceedings of 2016 IEEE Symposium on Security and Privacy. San Jose, CA, USA. 2016. 819–838.

2 李凤华, 苏锟, 史国振, 等. 访问控制模型研究进展及发展趋势. 电子学报, 2012, 40(4): 806–813.

3 Sandhu R, Ferraiolo D, Kuhn D. The NIST model for role-

based access control: Towards a unified standard. Proceedings of the 5th ACM Workshop on Role-based Access Control. Berlin, Germany. 2000. 47–63.

4 Ferraiolo DF, Sandhu R, Gavrila S, *et al*. Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security, 2001, 4(3): 224–274. [doi: 10.1145/501978.501980]

5 Chang CC, Buehrer DJ. Access control in a hierarchy using a one-way trap door function. Computers & Mathematics with Applications, 1993, 26(5): 71–76.

6 Leitner M, Rinderle-Ma S. A systematic review on security in process-aware information systems-constitution, challenges, and future directions. Information and Software Technology, 2014, 56(3): 273–293. [doi: 10.1016/j.infsof.2013.12.004]

7 Lin CH. Dynamic key management schemes for access control in a hierarchy. Computer Communications, 1997, 20(15): 1381–1385. [doi: 10.1016/S0140-3664(97)00100-X]

8 Akl SG, Taylor PD. Cryptographic solution to a problem of access control in a hierarchy. ACM Transactions on Computer Systems (TOCS), 1983, 1(3): 239–248. [doi: 10.1145/357369.357372]

9 Odelu V, Das AK, Goswami A. A secure effective key management scheme for dynamic access control in a large leaf class hierarchy. Information Sciences, 2014, 269: 270–285. [doi: 10.1016/j.ins.2013.10.022]

10 D'Arco P, De Santis A, Ferrara AL, *et al*. Variations on a theme by Akl and Taylor: Security and tradeoffs. Theoretical Computer Science, 2010, 411(1): 213–227. [doi: 10.1016/j.tcs.2009.09.028]

11 Shen VRL, Chen TS. A novel key management scheme based on discrete logarithms and polynomial interpolations. Computers & Security, 2002, 21(2): 164–171.

12 Hassen HR, Bettahar H, Bouadbdallah A, *et al*. An efficient key management scheme for content access control for linear hierarchies. Computer Networks, 2012, 56(8): 2107–2118. [doi: 10.1016/j.comnet.2012.02.006]