Rights Management for Role-Based Access Control

Bart Bouwman
Philips Research / Ordina
Eindhoven, The Netherlands
bart.bouwman@ordina.nl

Sjouke Mauw
University of Luxembourg
Luxembourg
sjouke.mauw@uni.lu

Milan Petković
Philips Research Europe
Eindhoven, The Netherlands
milan.petkovic@philips.com

***Abstract*- Healthcare requires a new approach with respect to the secure management of information. For this purpose we extend the Role Based Access Control model with context awareness, exceptions and delegation. By combining this extended model with common notions from the field of Enterprise/Digital Rights Management we obtain a framework for controlling shared information in a distributed environment.**

I. INTRODUCTION

Advances in information and communication technologies are expected to bring large benefits in the healthcare domain: the introduction of interoperable Electronic Health Record (EHR) systems [1] can reduce the cost of the healthcare system and enhance the overall quality of treatments, whereas Remote Patient Management services [2] will limit the time a patient stays in hospital. However, the application of information technology in a very complex healthcare environment has led to new security requirements, such as privacy concerns related to access of patient records outside the controlled environment of a hospital. These requirements are also recognized by security and privacy regulations (such as EU Directive 95/46 [3] or HIPAA [4] in the US) to which healthcare solutions have to comply.

Modern healthcare architectures tend to be open, interconnected environments. Therefore, sensitive patient records no longer reside on mainframes physically isolated within a healthcare provider, where physical security measures can be taken to defend the data and the system. Patient files are rather kept in an environment where data is outsourced to or processed on partially untrusted servers in order to allow de-centralized access for family doctors, medical specialists, pharmacists, and even non-medical care providers. The currently employed server-centric protection model, which locks the data in a database server and uses a traditional access control model to permit access to data, cannot efficiently deal with the requirements of the new healthcare infrastructures in which patient data may reside outside the control of the central server.

In order to allow sharing of records among different healthcare providers or with external parties, end-to-end security techniques facilitating data-centric protection have been suggested [5,6]: data is cryptographically protected and allowed to be outsourced or even freely float on the network. Rather than relying on different networks to provide confidentiality, integrity and authenticity, data is protected at the end points of the communication. However, this is not straightforward to achieve in the healthcare domain. One of the most important challenges in this domain is the implementation of an advanced role-based access control system (RBAC) which fulfills healthcare requirements. In particular, the introduction of exceptions and context awareness makes it hard to use an existing enterprise or digital rights management system for this purpose.

A particular example requirement to access control mechanisms is that, according to HIPAA, patients have the right to request restrictions (exceptions) with respect to access to their health records. A patient could request restrictions on the disclosure of their records to certain individuals involved in his care that otherwise are permitted (with respect to role-based access control governed by care institutions). Furthermore patients have the right to completely hide certain records in their EHRs from healthcare providers. Similar requirements can also be found in the specifications of national EHR systems that are under development (see for example the UK Spine system [7]). Obviously, this poses additional requirements on confidentiality and access control with respect to health records. Namely, in some cases access is based on roles, in other cases, it is not only based on roles but also on individual's restrictions. Therefore, there is a clear need for a combination of role-based access control and patient-managed access control. Furthermore, complementing access control with context awareness and delegation are of utmost importance in the healthcare domain.

In this work we address the aforementioned problems by extending the traditional role-based access control model to fulfill the new requirements. Then we implement this model in the DRM context allowing its application in new distributed and off-line scenarios of future healthcare. Finally we describe a DRM architecture that encapsulates our solution and discuss several improvements related to license management and permission evaluation.

II. RELATED WORK

Sandhu is often seen as the originator of role-based access control. In one of his papers [8] several role-based access control models are introduced. The four models described in his paper are the base model, the role hierarchies, the constraints model, and the consolidated model. The basic model defines only the smallest set of relations to allow a role-based access control setting. This model contains four entities:

Users, Roles, Permissions, and Sessions. In the consecutive model, role hierarchies are introduced. This extension to the base model defines a relation, which is a partial order, on roles that defines permissions that are inherited from another role. Hierarchies introduce a natural way of reflecting an actual organization's line of responsibility.

Another extension to the basic model introduces constraints on all relations of the model. This allows defining conditions that determine the acceptability of various components in the basic model. Constraints (such as mutually exclusive roles, cardinality, etc.) are a very important part of an RBAC model. The consolidated model of Sandhu combines constraints and role hierarchy. This model is further extended in the next section to fulfill the aforementioned requirements in the healthcare domain.

As already mentioned, context information is very important in the healthcare domain. It can be used to characterize the situation of an entity (e.g. patient records should be accessible to all staff in an emergency room, irrespective of their normal permissions). In the literature, context is very often taken into account as part of the RBAC model. In the papers of Covington [9, 10], environment roles are introduced. These are actually part of the formal model defining the role-based access control system. His architecture is based on an extension of the role-based access control model [11]. A negative aspect about this approach is that one has to model the context in a way that it fits into roles. Another practical issue is that this method increases the administrative complexity in managing access control policies.

While Covington extends the general RBAC model to create context-awareness mainly on the permissions to roles assignment, Wullems [12] defines context-awareness on the users to roles assignment and authentication of users to the system. This approach is designed to augment existing security protocols and therefore he focuses on the implementation of the architecture. In this paper, we will define constraints and context over the complete model and make the instantiation of the formal model more transparent.

III. Extended RBAC Model

This chapter describes the evolution of Sandhu's standard RBAC model [8] into the context-aware RBAC model with exceptions (CARBAC-E) that is suitable for the healthcare domain. We take the consolidated model of Sandhu, as the basis and extend it, first to allow exceptions on permission policies (the DAC-like user and role exceptions) and secondly to allow object categorization.

With the extended model, it is possible to define personalization of default policies. Patients can define exceptions and therefore overrule permissions that are indirectly assigned to users via the membership relation. In the same way as user-based exceptions, role-based exceptions can be defined. They overrule permissions that are directly assigned to roles. Next to that, the model is extended with categories that contain objects. This creates flexibility in policy management, as one can directly use default policies instead of using templates to generate custom policies.

Figure 1 depicts the extended model that combines the above-mentioned features.

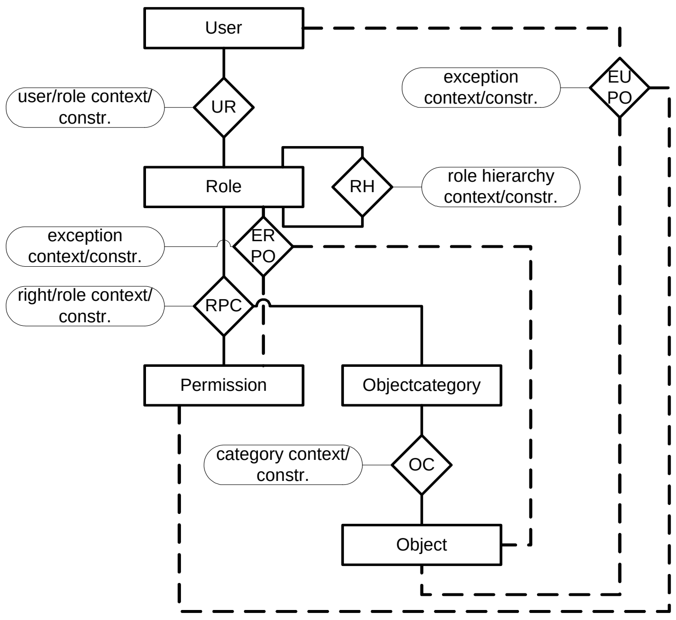

User
user/role context/ constr.
UR
exception context/constr.
EU PO
Role
RH
role hierarchy context/constr.
exception context/constr.
ER PO
right/role context/ constr.
RPC
Permission
Objectcategory
category context/ constr.
OC
Object

Figure 1. The extended model.

A. Formalization

Figure 1 provides an intuitive understanding of the model, but it does not unambiguously define it. To address this point as well as to allow for verification and unambiguous implementation, we provide its formal definition below.

Definition 1: Actions (A). Let A be the set of all actions that can be used in the definition of permissions: A = $\{a_1..a_n\}$, where a single action is represented by the tuple a_i: (action: String). A few examples of actions are view, write, print, etc.

Definition 2: Permission Types (T). Let T be the set of all types that can be used in the definition of permissions: T = {+, -, ?}. A permission can be of type allow (+), deny (-), or not-known (?). Every permission that is not specified is interpreted as a permission with type '?'. The ordering on types is $- \geq + \geq ?$. This means that negative permissions overrule positive permissions. The *max* function, which will be used later in the evaluation phase, is defined on pairs and sets of types (V):

$$\max(t_1, t_2) = \begin{cases} t_1 & \text{if } t_1 \geq t_2 \\ t_2 & \text{if } t_2 > t_1 \end{cases}$$

$$\max(V) = \begin{cases} ? & \text{if } |V| = 0 \\ \text{head}(V) & \text{if } |V| = 1 \\ \max(\text{head}(V), \max(\text{tail}(V))) & \text{if } |V| \geq 2 \end{cases}$$

Definition 3: Users (U). Let U be the set containing all users that take part in the model: U = $\{u_1..u_n\}$, where a single user is represented by the tuple u_i: (userName: String). In practice, a user has more attributes than his name.

Definition 4: Roles (R). Let R be the set containing all roles that are part of the model; R = $\{r_1..r_n\}$, where a single role is represented by the tuple r_i: (roleName: String). Some examples of roles are nurse, doctor, radiologist, etc.

Definition 5: Permissions (P). Let P be the set containing all permissions that can be executed on objects by users assigned to roles: $P = \{p_1..p_n\}$, where a single permission is represented by the tuple p_i: (action: A, type: T [, delegate: Int]). The action attribute describes the actual permission, like view, print, etc. The second attribute is type, which specifies whether the action is allowed or denied. The last attribute is optional and indicates the number of times the permission can be delegated to other roles. It is only allowed on permissions of type +. If this value is not provided, it will be interpreted as a permission that cannot be delegated.

Definition 6: Objects (O). Let O be the set containing all objects that are part of the model: $O = \{o_1..o_n\}$, where a single object is represented by the tuple o_i: (objectReference: URI). The uniform resource identifier (URI) is a reference to the object's resource.

Definition 7: Categories (C). Let C be the set containing all objects categories that are part of the model: $C = \{c_1..c_n\}$, where a single category is represented by the tuple c_i: (categoryName: String). An example of a category is the set of all radiology images for a patient.

Definition 8: Membership (UR). Let UR be the set containing all membership assignments that are part of the model: $UR \subseteq U \times R$. UR is a many-to-many relation defining users' membership in roles. This relation is responsible for the dynamic nature of this model. Using the membership relation a user is easily assigned to permissions that correspond to the specific role.

Definition 9: Role Hierarchy (RH). Let RH be the set defining inheritance on roles that are part of the model: $RH \subseteq R \times R$. A role can inherit permissions from another role. RH is a partial order defining the inheritance of roles. This means that the RH relation must be reflexive, asymmetric and transitive. Therefore we will also use $\geq$ to define this order.

The meaning of $r \geq r'$ is that role r inherits all permissions of role r' and maybe contains additional permission assignments. As we also want to use the notion of direct children, we introduce $\rightarrow$, which is defined as follows:

$r \rightarrow r' \Leftrightarrow r \geq r' \wedge r \neq r' \wedge \neg(\exists r'': r \neq r'' \wedge r' \neq r'': r \geq r'' \geq r')$

Definition 10: Policy (RPC). Let RPC be the set containing all permissions on object categories that roles in the model have: $RPC \subseteq R \times P \times C$. The default permissions that roles have on certain object categories, called RPC, is a many-to-many-to-many relation that binds permissions to categories and roles. The RPC relation is better known as the default policy (in the original RBAC model defined as the role-permission-object relation). This relation is rather static, because the default policy is the same for every document of the same type. Because categories are used, policies are not defined on objects but on categories. All policies remain fairly static for each category. There is no need to create new relations for each new object and therefore no need for templates. Another advantage of this construction is that changes to a default policy immediately hold for all objects because they hold for categories that contain objects.

Definition 11: User-based Policy Exceptions (EUPO). Let EUPO be the set containing all permission exceptions on objects based on users that are part of the model: $EUPO \subseteq U \times P \times O$. The permission exceptions that users have on certain objects called EUPO, is a many-to-many-to-many relation that binds exceptional permissions directly to users. This relation is used to specify individual exceptions on a default policy.

Definition 12: Role-based Policy Exceptions (ERPO). Let ERPO be the set containing all permission exceptions on objects based on roles that are part of the model: $ERPO \subseteq R \times P \times O$. The permission exceptions that roles have on certain objects called ERPO, is a many-to-many-to-many relation that binds exceptional permissions related to an object to all users that are assigned to a particular role.

Definition 13: Categorization (OC). Let OC be the set containing all object categorizations that are part of the model: $OC \subseteq O \times C$. The object categorization is a many-to-many relation that classifies objects into categories.

Definition 14: Complete RBAC Model (CARBAC-E). The extended model is now defined as the following tuple: CARBAC-E = (U, R, P, O, C, UR, RH, OC, RPC, EUPO, ERPO).

B. *Permission Evaluation*

Due to exceptions and a role hierarchy, the extended model requires different permission evaluation than a traditional RBAC model. When the system, based on a user request, determines if the user should have access to a requested record, in our case, the following checks have to be made:

- (step 1) first if there is an exception for the user in question (EUPO), then
- (step 2) exceptions for the role to which the user belongs (ERPO) and
- (step 3) finally the default policy RPC.

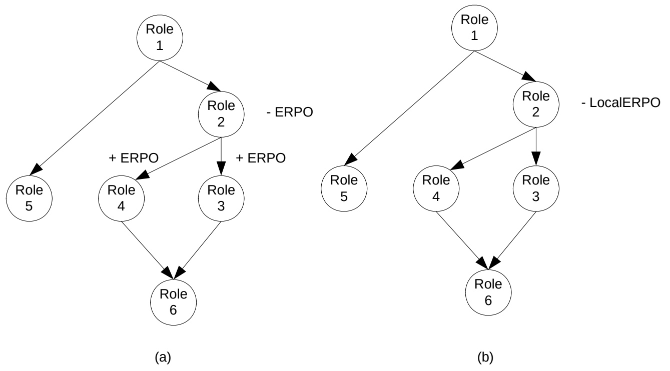
The role hierarchy (the RH relation in our model) introduces additional changes in the process of evaluating permissions. If the evaluation process is not clearly resolved at the level of the role that is in question (resulting in a positive or negative permission), then step 2 and after that step 3 from the procedure defined above have to be recursively performed at the levels, which are higher in the hierarchy (recursively through all parent roles). Appendix A provides a detailed description of such an algorithm.

The introduction of the role hierarchy influences also role-based exceptions (ERPOs), which should also be inherited to allow easy administration[1]. If a patient wants to deny access to a specific document for everyone, it is enough to create an ERPO at the root (public) role from which all roles inherit permissions. However, inheritance of ERPOs could also cause

[1] One can choose not to inherit ERPOs, which means that ERPOs have to be specified for all relevant nodes in the hierarchy (which is definitely much more complex and not in accordance with the idea of role hierarchy and inheritance).

some problems. For example, consider a case where a patient wants to exclude a specific role while keeping the other roles that inherit permission from that role in the access list (an example is given in Figure 2 where a patient excludes role 2 while having role 3, 4 and 6 still be able to access the data). Then a negative permission in an ERPO has to be assigned to role 2, while to avoid that the other roles inherit this permission, positive ERPOs have to be assigned to the roles on the next (lower) level in hierarchy (here roles 3, and 4). This is denoted in Figure 2a. To make this more efficient we suggest to divide ERPOs into local and global ERPOs, where local ERPOs will be valid only for the role for which they are specified (and will not be inherited). On the other hand global ERPOs will be inherited. Then the above-mentioned example can be solved with only one local negative ERPO as shown in Figure 2.

Figure 2. Example local and global ERPO.



IV. Extended RBAC in DRM Context

To enable controlled sharing of information in a distributed system as well as the offline use, we implement this model in a DRM setting (architecture is shown in Figure 3). Consequently, we have to transfer the relevant part of the model to the client and evaluate the permissions client-side. Another quite important reason why we need to evaluate permissions client-side is that they could require context which can only be evaluated in the client-side environment.

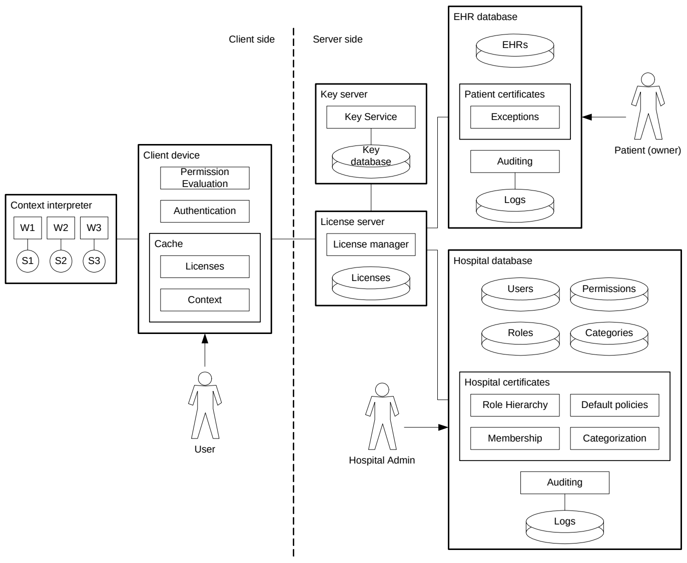
A. *Certificates*

To support client-based evaluation of the model, every relation in that model has to be represented by a certificate. The relations UR, RH, OC, RPC, EUPO, and ERPO are defined below using certificates.

UR = {User, Role, (context/constraints)*}$_{\text{signature}}$
RH = {Role, ChildRole, (context/constraints)*}$_{\text{signature}}$
OC = {Object, Category, (context/constraints)*}$_{\text{signature}}$
RPC = {Role, Category, Permission, (context/constraints)*}$_{\text{sig.}}$
EUPO = {Object, User, Permission, (context/constraints)*}$_{\text{sig.}}$
ERPO = {Object, Role, Permission, (context/constraints)*}$_{\text{sig.}}$

The EUPO certificate states which permissions are granted or denied to specific users, when zero or more context rules or constraints hold. Each permission that is specified in this type of certificate overrules permissions that are specified in other certificates with lower priorities. Similar considerations hold for the ERPO certificate. The only difference is that the exceptions are not user-based but they are role-based. This means that the exception is applied to every user that is assigned to a certain role.

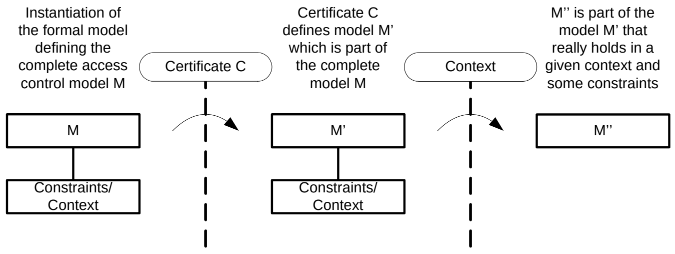
Figure 3. DRM Architecture



B. *Model Instantiation*

As already mentioned, we choose to define constraints and context on top of the access control model, so that the instantiation of the formal model is more transparent. Figure 4 shows how an instantiation is made of the formal model, what happens if one instantiates (part of) the model in certificates/licenses and validates these certificates in a certain context. In this figure, one can clearly see that the definition of the model does not change when the constraints are added. This implies that the formal model of the access control system remains the same throughout the entire usage in a real-life system. Therefore the same formal representation is used at the client side (i.e., the actual access control model for a given user, role, context, etcetera), as well as at the server side.

Figure 4. Usage of context in instantiation of CARBAC-E model


Instantiation of the formal model defining the complete access control model M
Certificate C
Certificate C defines model M' which is part of the complete model M
Context
M'' is part of the model M' that really holds in a given context and some constraints
M
M'
M''
Constraints/ Context
Constraints/ Context

C. *Translation into DRM licenses and permission evaluation*

A straightforward solution for translating the extended RBAC model to DRM licenses is to use certificates for every instance of the relations in the RBAC model (as defined in Section IVA). A license can be used as a container holding all relevant certificates for a particular permission evaluation. Because certificates can specify negative as well as positive permissions, one must be certain that all relevant permissions i.e. certificates are used in the permission evaluation algorithm (in a response to a request from a user to access a data object). To solve this problem a license can be created on-line to make sure that it contains all relevant certificates (or references to certificates) for a specific user and specific data object. Note that certificates can have different origins like some certificates are signed by the government and stored in a global database while others are signed by a hospital and stored in local databases. Exceptions, which are often defined by patients, can come from a different source. A license that contains all relevant certificates can be created in a centralized way, i.e. a central authority can contact all relevant authorities and include their certificates in the license.

Definition 15: License. First, we define a certificate list, where all certificates are signed by trusted authorities or users: $Clist_x \in 2^S$, where the set containing all relevant certificates S signed by one of the parties in x, is defined as:

S = {UR, RH, OC, RPC, EUPO, ERPO}

Now let a license L be defined as:

$L = (Clist_x, ObjectKey)_{signature}$ or $L = (Clist_x, L)_{signature}$

A license is an element of the power set containing all certificates and the key that can be used to decrypt the object. The license only contains relevant certificates defining the relations of the model that could apply to a certain user, requesting certain permission on certain objects. Note that a license can also have license specific attributes like the period in which the license is valid or the number of certificates a license contains. The object key is encrypted in a way that only the recipient can decrypt it and therefore use the policy to enforce the permissions on the object. A key management solution for our approach is described in our previous work [13, 14]. The second definition of a license is used for the delegation of permissions. When a user X delegates a permission, he takes his use-license, which states that he can delegate the permission, and adds the delegated permission including all relevant certificates. Using this formalism, delegations can be made without interference of the certification authority.

D. *Decentralized license management*

However, instead of creating a license for each request, it is much more flexible to allow a client to reuse certificates that are already available, combining certificates from different parties and evaluating permissions off-line. Then, "certificate chains" must be used to dynamically produce a valid license. But then it is indispensable to make sure to verify the completeness of data instead of using licenses to enforce this.

For example, it is possible to optimize the RH certificate so that it does not need the license to verify the completeness of data for the permission evaluation algorithm. The previous RH certificate makes a relation between a role and one of its parent roles. Without a license that binds all relationships of a node in a hierarchy one cannot be sure of a completeness of a set of individual certificates. This could be misused by providing a set of certificates which exclude parent nodes where exceptions are defined. In that way the permission evaluation will not be correct. However, if each certificate lists all the direct parents of a node (see RH below) a client device can independently and unambiguously create the role hierarchy from different individual certificates. If no parent role is listed in the certificate, it means that the end of the role hierarchy is reached i.e. that this role has no parent roles.

$RH = \{Role, ParentRoles\}_{signature}$.

We also have to make sure that EUPO and ERPO certificates are somehow embedded in the other certificates that are necessary to evaluate in order to get the required permission. Otherwise, someone could delete, or simply make these certificates that define exceptions not available. In that way correct evaluation of permission is not guaranteed. If these certificates contain negative authorizations (exceptions) then not presenting these certificates to the permission evaluation algorithm will result in wrong permission evaluation (e.g. granting instead of refusing access).

To solve this problem we propose to include both EUPO and ERPO in an OC certificate. OC certificates describe mapping from objects to categories. This certificate will be bound to the object and stored together with the object. It is necessary to have an OC certificate in permission evaluation, therefore exceptions (or reference to appropriate EUPO and ERPO certificates), which are now stored in this certificate, cannot be avoided.

$OC = \{Object, Category, (Role, Exception)^*, (User, Exception)^*\}_{signature.}$

In this way we ensured that the relevant exceptions will be taken into account when permissions are evaluated off-line without creating a license for each access request. Obviously, revocation of certificates must be ensured to enable timely update of policies (e.g. new exceptions).

V. Conclusions

Information systems in healthcare have domain specific security and privacy requirements. If one would like to apply RBAC in these information systems to reduce the administrative tasks and develop a universal solution, several extensions including context-awareness, personalization of access control policies and delegation must be introduced. Therefore, we have proposed the CARBAC-E model that satisfies these specific requirements. Another aspect that is addressed by this paper is controlled data sharing in an open distributed environment and off-line scenarios. We have designed a DRM architecture and shown how to apply and enforce the proposed model in such environment. The proposed approach increases the data availability which is the most important requirement in healthcare. Furthermore, we have developed an algorithm that handles permission evaluation when conflicting permissions are present. Finally, a flexible method for client enforcement of the proposed model has been discussed.

APPENDIX

This appendix includes a detailed description of a permission evaluation algorithm. It uses three functions each taking care of one step of the algorithm. All these steps use the *max* function over permissions which is defined in Section IIIA. Below, we will provide the definition of each of the three steps of the algorithm. PermEval defines the permission evaluation for a user *u* trying to execute action *a* on object *o*.

```
*** Step 1 ***
PermEval(u, a, o): $U \times A \times O \to T$
    If ($\exists t: t \in T: (u, (a, t), o) \in eupo$):
        PermEval(u, a, o) = max($\{t | (u, (a, t), o) \in eupo\}$)
    Otherwise (u has no direct permission assignments):
        PermEval(u, a, o) = max($\{PermEval_{(erpo)}(r, a, o) | (u, r) \in UR\}$)

*** Step 2 ***
$PermEval_{(erpo)}$(r, a, o): $R \times A \times O \to T$
    If ($\exists t: t \in T: (r, (a, t), o) \in erpo$):
        $PermEval_{(erpo)}$(r, a, o) = max($\{t | (r, (a, t), o) \in erpo\}$)
    Otherwise (r has no direct permission assignments):
        $PermEval_{(erpo)}$(r, a, o) = $PermEval_{(rpc)}$(r, a, o)

*** Step 3 ***
$PermEval_{(rpc)}$(r, a, o): $R \times A \times O \to T$
    If node r is a leaf or ($\exists c: (o, c) \in OC: (\exists t: t \in T: (r, (a, t), c) \in rpc)$):
        $PermEval_{(rpc)}$(r, a, o) = max($\{t | (r, (a, t), c) \in rpc \wedge (o,c) \in OC\}$)
    Otherwise (r has no direct permission assignments):
        $PermEval_{(rpc)}$(r, a, o) = max($\{PermEval_{(erpo)}(r', a, o) | r \to r'\}$)
```

If the algorithm does not return a positive or negative permission (i.e. yields "?"), this will be interpreted as a negative permission.

REFERENCES

[1] R. Charette, “Dying for Data”, IEEE Spectrum, October 2006, pp. 16-21.

[2] D. Simons, T. Egami, J. Perry, “Remote Patient Monitoring Solutions”, in G. Spekowius and T. Wendler (Eds.), *Advances in Healthcare Technology*, Springer, 2006, pp. 505-516.

[3] European Directive 95/46. “Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data”, European Commission, 1996, Retrieved from http://ec.europa.eu/justice_home/fsj/privacy/

[4] U.S. Department of Health & Human Services, “The Health Insurance Portability and Accountability Act of 1996 (HIPAA)”, 1996; Retrieved from http://www.cms.hhs.gov/HIPAAGenInfo/

[5] M. Petković, S. Katzenbeisser, K. Kursawe, “Rights Management Technologies: A Good Choice for Securing Electronic Health Records?”, In the Proceedings of the Information Security Solutions Europe Conference, ISSE 2007.

[6] Milan Petković, Malik Hammouténe, Claudine Conrado and Willem Jonker, "Securing Electronic Health Records using Digital Rights Management", In Proceedings of the 10th International Symposium for Health Information Management Research (iSHIMIR), ISBN 960-87869-5-9, Thessalonica, Greece 2005.

[7] NHS electronic medical records "data spine" privacy and security worries, 2006, Retrieved from http://www.spy.org.uk/spyblog/2006/0/nhs_electronic_medical_records_1.html

[8] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman, “Role-Based Access Control Models”, *IEEE Computer*, vol.29, no.2, Feb 1996, pp. 38-47.

[9] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dey, Mustaque Ahahmed, Gregory D. Abowd, “Securing Context-Aware Applications Using Environments Roles”, in proceedings of the sixth ACM symposium on Access control models and technologies, United States, May 2001, pp. 10-20.

[10] Michael J. Covington, P Fogla, Zhan Zhiyuan, Ahamad, Mustaque Ahahmed, “A context-aware security architecture for emerging applications”, in Computer Security Applications Conference, December 2002, pp. 249-258.

[11] Michael J. Covington, Matthew J. Moyer, Mustaque Ahamed, “Generalized role-based access control for securing future applications”, in Proceedings of the 23rd national information systems security conference (NISSC), USA, October 2000, pp. 40-51.

[12] Chris Wullems, Mark Looi, Andrew Clark, “Towards context-aware security: an authorization architecture for intranet environments”, in proceedings of the 2nd IEEE Annual Conference on pervasive Computing and Communications Workshops, 2004, pp. 132–137.

[13] M. Petković, C. Conrado, M. Hammouténe, “Cryptographically Enforced Personalized Role-Based Access Control”, In S. Fisher-Hubner, K. Rannenberg, L. Yngstrom, S. Lindskog (Eds), *Security and Privacy in Dynamic Environments*, Springer, USA, 2006, pp. 364-376.

[14] Anna Zych, Milan Petković, Willem Jonker, “A Key Management Method for Cryptographically Enforced Access Control”, the 5th International Workshop on Security in Information Systems (WOSIS), Portugal, 2007.