Trust Management Languages and Complexity

Krzysztof Sacha

Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warszawa, Poland
k.sacha@ia.pw.edu.pl

Abstract. Trust management is a concept of automatic verification of access rights against distributed security policies. A policy is described by a set of credentials that define membership of roles and delegation of authority over a resource between the members of roles. Making an access control decision is equivalent to resolving a credential chain between the requester and the role, which members are authorized to use a resource. A credential is an electronic document, formulated using a trust management language. This way, trust management languages are a tool for describing credentials and specifying access control policies in a flexible and modifiable way. This paper discusses the expressive power of trust management languages, describes a new extension to Role-based Trust Managements language RT^T, and evaluates the complexity of algorithm that is used for answering security queries.

Keywords: Access control, trust management, role-based trust management language, credential graph, credential chain.

1 Introduction

The traditional access control mechanism assigns to each protected resource an access control list (ACL), which enumerates the entities that have permissions to access the resource [1]. A decision on whether to allow or to deny access to a resource is based on a verification of identity of the requester. If the number of entities is big and many entities may have the same access rights with respect to resources, then assigning access rights to roles, i.e. sets of entities, rather than to individual entities, can help in reducing the size of the access control problem [2]. Such a schema can easily be implemented in such a system, in which the identity of entities that can make requests for resources is known in advance, e.g. in the form of user accounts that are created before the access to resources is really requested.

Quite another approach to access control is needed in open environments, in which the identity of potential users is not known in advance. For example, the users are identified by public keys and no user accounts are required. Trust management is a concept of decentralized access control, in which the decisions are based on credentials presented by the requesters, regardless of their identity. A credential is an electronic document, which describes a delegation of access rights from one entity (the issuer) to another entity or a group of entities. Such an approach separates a symbolic representation of trust (credentials) from the identity of users.

R. Meersman, T. Dillon, and P. Herrero (Eds.): OTM 2011, Part II, LNCS 7045, pp. 586–602, 2011.

A set of credentials describes a security policy in terms of roles, role membership and delegation of authority between the members of roles. A security policy can be decentralized, if the members of particular roles may issue the credentials locally. Making an access control decision is equivalent to resolving a credential chain between the requester and the role, which is authorized to use the resource. This way, trust management becomes a concept of automatic verification of access rights against security policies [3]. The policies are created externally to the applications, and can be modified without the need to modify the executable program code of the application.

Credentials are implemented as electronic documents, composed of statements in a trust management language. Several languages to describe credentials and security policies have been developed and described in the literature. The goal of this research is to look at syntax, semantics and the expressive power of these languages, make some extensions to a Role-based Trust Managements language RT^T, and evaluate the complexity of an algorithm that is used for answering security queries.

The remaining part of the paper is organized as follows. Related work is described in Section 2. Trust management languages are summarized in Section 3. A motivating example of a policy-based access control system in an open SOA environment is presented in Section 4. Possible system architecture is discussed in Section 5, and an extension to RT^T is described in Section 6. An algorithm for resolving a set of RT^T credentials that define a policy and an evaluation of computational complexity are given in Section 7. The conclusions and plans for further research are in Section 8.

2 Related Work

There can be many independent entities in distributed computing environments, such as SOA systems, with authority to control access to a variety of system resources. Moreover, there can be a need of multiple entities to have input to the access control policy for a single resource. A secure implementation of such a distributed access control mechanism can be based on credentials that convey authorization to use a resource from one entity to another. A *credential* is a digitally signed certificate, which provides a copy of the public key of the issuer, the receiver and a description of the authority that is delegated from the issuer to the receiver of this credential.

A few such distributed authorization environments have been implemented and described in the literature. Examples are PolicyMaker [3], KeyNote [4], SPKI/SDSI [5] and Akenti [6]. All those systems use languages that allow assigning privileges to entities and use credentials to delegate permissions from their issuers to subjects.

In SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) a concept of distributed name space was introduced, according to which an entity could define a local name, related to certain privileges with respect to system resources, and then define members of the local name space or delegate responsibility for the name space membership to another entities. The definition of names and delegation of responsibility was expressed by means of credentials. An algorithm for building the semantics of a given set of credentials was defined with efficiency of order $O(n^3l)$, where n was the number of certificates and l was the length of 'hops' in the longest responsibility delegation chain [5].

The concept of distributed name space was moved at a higher level of abstraction by the introduction of roles and Role-based Trust management languages [7-9].

There is a family of RT languages, with varying expressive power and complexity. The basic language RT_0 allows describing roles and role membership, delegation of authority over roles and role intersection. The issuers as well as the members of roles are entities, and no means for expressing complex security policies, such as threshold policy, exists in this language.

RT^T extends the expressive power of RT_0 significantly, by adding manifold roles and providing extensions to express threshold and separation of duties policies. A manifold role is a role that can be satisfied only by a group of cooperating entities. A threshold policy requires a specified minimum number of entities to agree before access is granted. Separation of duties policy requires that different entities must hold the conflicting roles. If a cooperation of these roles is required before access is granted, then a set of entities is needed, each of which fulfils only one of these conflicting roles. Both of these two policies mean that some roles cannot be fulfilled by a single entity and a group of entities must cooperate in order to satisfy these roles.

Apart of an informal interpretation, a formal definition of the language semantics, which gives meaning to a set of credentials in the application domain, has been provided. In [8], constraint DATALOG was used as the semantic foundation for the RT language family. According to this approach, credentials in RT were translated into DATALOG rules. This way, the semantics was defined in an algorithmically tractable way. A strict, set theoretic semantics of RT_0 was defined in [9] and of RT^T in [10,11]. An important difference between the two (RT_0 and RT^T) is such that RT_0 supports singleton roles only, and the meaning of a role is a set of entities, each of which can fulfill this role, while RT^T supports manifold roles, and the meaning of a role is a set of groups of entities that fulfill the role.

Security queries can be resolved with respect to a set of credentials that define the security policy within a system, by means of a credential graph, which is described in Section 7. The nodes of this graph are roles and groups of entities, which define the meaning of roles. Making a decision on the membership of a group of entities in a given role is equivalent to checking whether a path from this group to the role exists in the graph. An algorithm to construct a credential graph of a set of RT_0 credentials was introduced in [8]. An early version of the algorithm to construct a credential graph of RT^T credentials was described in [11], however, without an estimation of computational complexity. An improved algorithm to construct a credential graph of extended RT^T is presented in Section 7, together with an evaluation of complexity.

3 Trust Management Languages

All RT languages are built upon a set of three basic notions: Entities, role names and roles. An **entity** is someone who can participate in issuing certificates, making requests to access a resource, or both. An entity can, e.g., be a person or a program identified by a public key within a computer system. A **role name** represents access rights with respect to system resources, similar to Admin, Guest or Simple user in Windows operating system, granted by an entity. A **role** represents groups of entities

(may be singletons) that have access rights related to a certain role name and granted by the role issuer. Credentials are statements in a RT language, which are used for describing access control policies, assigning entities to roles and delegating authority to the members of other roles.

In this paper, we use nouns beginning with a capital letter or just capital letters, e.g. *A*, *B*, *C*, to denote groups of entities. Role names are denoted as identifiers beginning with a small letter or just small letters, e.g. *r*, *s*, *t*. Roles take the form of a group of entities (role issuer) followed by a role name separated by a dot, e.g. *A.r*. A credential consists of a role, left arrow symbol and a valid role expression, e.g. $A.r \leftarrow B.s$.

BNF specification of the RT^T syntax can be written as follows.

<credential> ::= <role> ← <role-expression>
<role> ::= <entity-set> . <role-name>
<role-expression> ::= <entity-set>
| <role>
| <role> . <role-name>
| <role> ∩ <role
| <role> ⊕ <role>
| <role> ⊗ <role>

There are six types of role expressions, according to this specification, and six types of credentials in RT^T, which are interpreted in the following way:

$A.r \leftarrow B$ – *simple membership*: a group of entities *B* can satisfy role *A.r*.

$A.r \leftarrow B.s$ – *simple inclusion*: role *A.r* includes all members of role *B.s*. This is a delegation of authority over *r* from *A* to *B*, as *B* may cause new groups of entities to become members of role *A.r* by issuing credentials that define *B.s*.

$A.r \leftarrow B.s.t$ – *linking inclusion*: role *A.r* includes role *C.t* for each *C*, which is a member of role *B.s*. This is a delegation of authority over *r* from *A* to all the members of role *B.s*.

$A.r \leftarrow B.s \cap C.t$ – *intersection inclusion*: role *A.r* includes all the groups of entities that are members of both roles *B.s* as well as *C.t*. This is a partial delegation from *A* to *B* and *C*.

$A.r \leftarrow B.s \oplus C.t$ – *role product*: *A.r* is a manifold role that can be satisfied by a union set of one member of role *B.s* and one member of role *C.t*. However, the same person can play both of these roles.

$A.r \leftarrow B.s \otimes C.t$ – *disjoint role product*: *A.r* is a manifold role that can be satisfied by a union set of one member of role *B.s* and one member of role *C.t*, where both members are disjoint.

This allows expressing separation of duties policy, in which different entities must hold the conflicting roles *B.s* and *C.t*.

In case when the roles are identical, e.g. $A.r \leftarrow B.s \otimes B.s$, the credential can express threshold policy in which two (or more, if we use more credentials) members of *B.s* can jointly fulfill *A.r*.

The syntax of a language describes the rules for constructing language expressions, such as credentials in RT^T. The semantics of a language describes the meaning of expressions in the application domain. Such a definition consists of two parts [12]: A definition of a semantic domain, which gives meaning to the language expressions, and a semantic mapping from the syntax to the semantic domain. A set theoretic semantics is the one that takes sets or power sets of entities as the semantic domain. If singleton roles are considered, the meaning of a role can be a set of entities that fulfill this role. If manifold roles are taken into account, the meaning of a role is a set of groups (sets) of entities that fulfill the role.

Let E be a set of entities and R be a set of role names. Denote the power set of entities by $F = 2^E$. Each element in F is a set of entities from E. Each element in 2^F is a set, composed of sets of entities from E. The semantic mapping can now be described as a function:

$$\hat{S} : 2^E \times R \to 2^F$$

that maps each role from $2^E \times R$ to a set of all such sets of entities, which are members of this role. Such a mapping from the set of RT^T roles to the power set of entities can be defined formally in the language of first-order logic as shown in Table 1, where A, B, C, X, Y are groups of entities, r, s, t are role names, $A.r$, $B.s$, $C.t$ are roles, and $\hat{S}(A.r)$ denotes the semantics of role $A.r$.

Table 1. The interpretation of first-order formulas in RT^T

RT^T credential	Meaning of the credential
$A.r \leftarrow B$	$B \in \hat{S}(A.r)$
$A.r \leftarrow B.s$	$(\forall x)\ (\ x \in \hat{S}(B.s) \Rightarrow x \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s.t$	$(\forall x)\ (\forall y)\ (\ y \in \hat{S}(B.s) \wedge x \in \hat{S}(y.t) \Rightarrow x \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s \cap C.t$	$(\forall x)\ (\ x \in \hat{S}(B.s) \wedge x \in \hat{S}(C.t) \Rightarrow x \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s \oplus C.t$	$(\forall x)\ (\forall y)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(C.t) \Rightarrow x \cup y \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s \otimes C.t$	$(\forall x)\ (\forall y)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(C.t) \wedge x \cap y = \phi \Rightarrow x \cup y \in \hat{S}(A.r)\ )$

SPKI/SDSI allows the first three types of credentials only. However, linking inclusion of arbitrary length is allowed. This does not increase the expressive power of the language, because a linking inclusion of arbitrary length, e.g. $A.r \leftarrow B.s\ldots t.u$, can always be substituted by a pair of credentials $C.v \leftarrow B.s\ldots t$ and $A.r \leftarrow C.v.u$. This way linking inclusion of length l can always be substituted by a set of l credentials with linking inclusions of length 2. Therefore, the complexity $O(n^3 l)$ of the algorithm for building the semantics of a given set of SPKI/SDSI credentials can, in fact, be considered as equal to $O(n^4)$ in terms of the number n of RT credentials.

The language RT_0 allows the first four types of credentials, and supports singleton roles only. The language RT^T allows all six types of credentials and supports manifold roles. The use of manifold roles is inevitable, because the members of roles defined by the last two credentials are always groups rather than single entities.

4 Motivating Example

Consider a student management system of a university composed of a set of nearly independent faculties. The system offers a set of services for the university and for the faculties, according to the concept of service-oriented architecture (SOA). The faculties administer particular instances of each service separately. No centralized security policy for the system exists. Instead, the service owner defines a security policy for each service independently.

The university, the library, each faculty and each student has a public key and is an entity, which can participate in issuing credentials and requesting services from the system. The university defined a role that reflected the university structure and issued the following set of credentials:

$\{University\}.faculty \leftarrow \{IT\}$	// faculty of Information Technology
$\{University\}.faculty \leftarrow \{Chemistry\}$	// faculty of Chemistry
.....	// other faculties of the university

Each faculty of the university defined a set of roles that reflected the main actors of the didactic process and issued a set of credentials:

$\{IT\}.student \leftarrow \{A\}$	// A is an IT student
$\{IT\}.teacher \leftarrow \{X\}$	// X is an IT staff member
$\{IT\}.supervisor \leftarrow \{X\}$	// staff X can supervise students
.....	// other entities within IT faculty

The following types of services were identified for the first release of this system.

1. Library Service. Offered full on-line access to the library resources. The owner of the service was the university, which applied the following security policy: Access was granted to all the students and teachers of all faculties of this university. The access control list for the service contained a role $\{University\}.library$. Security policy was described by a set of two credentials:

$\{University\}.library \leftarrow \{University\}.faculty.student$
$\{University\}.library \leftarrow \{University\}.faculty.teacher$

2. Grade Book. A complex service with two separate entry points: For students – to read the grades, and for teachers – to add new grades. Particular faculties of the university owned separate instances of this service.

(a) The service offered the requesting student read access to all the grades for the requester. Security policy for each instance of the service was defined by each owner. Chemistry decided that access was granted to the students only, and no delegation of the access rights was allowed. The access control list for the service contained a role $\{Chemistry\}.gradeVisitor$. Security policy was described by a single credential:

$\{Chemistry\}.gradeVisitor \leftarrow \{Chemistry\}.student$

IT faculty selected another policy. Access was granted to students, who could delegate permission to another people, and these people could pass the delegation again. Such a policy was described by the following credentials:

$\{IT\}.\ gradeVisitor \leftarrow \{IT\}.student$
$\{IT\}.\ gradeVisitor \leftarrow \{IT\}.\ gradeVisitor.friend$

IT student A could now delegate permission to read his or her grades to another person, e.g. B, by issuing a new credential:

$\{A\}.friend \leftarrow \{B\}$

Because a member of role $\{A\}.friend$ became also a member of $\{IT\}.gradeVisitor$, then B could pass the delegation again to C:

$\{B\}.friend \leftarrow \{C\}$

It is important to note that the permission for delegation of access rights was an individual decision of IT faculty, and other faculties could decide differently. Such a decision can be changed at any time by simply changing the set of existing credentials – in this case, by removing the credential:

$\{IT\}.\ gradeVisitor \leftarrow \{IT\}.\ gradeVisitor.friend.$

If IT faculty removes this credential, then it will implement the same policy as Chemistry. No need for removing all the student's "*friend*" credentials exists.

(b) Teacher's entry point represented in fact a set of services to manipulate grades received by students in particular courses. Only the teachers (one or more) assigned to a course could add or change a grade. Therefore, there was a separate access control list maintained for each course, which was identified by a course number *NN*.

IT faculty decided that an assigned teacher could delegate access to another teacher (an assistant), but the delegates could not pass the permissions again. Such a policy was implemented at IT for a course *NN* by an access control list that contained a role $\{IT\}.grade\_NN$ and the following set of credentials:

$\{IT\}.grade\_01 \leftarrow \{IT\}.teacher\_01$
$\{IT\}.grade\_01 \leftarrow \{IT\}.teacher\_01.assistant \cap \{IT\}.teacher$
$\{IT\}.teacher\_01 \leftarrow \{X\}$
..... // the same for courses 02, 03,…

IT teacher X assigned to a course number 01 could now delegate permission to manipulate grades in the course to another person, e.g. Y, by issuing a new credential:

$\{X\}.assistant \leftarrow \{Y\}$

The delegation was effective only when the delegate was an IT teacher. The delegate could not pass the permission to another person.

3. Supervisor Assignment. The service allowed registering assignment of a student to the selected supervisor. Separate instances of this service were owned by particular faculties of the university. Security policy for each instance of the service was defined

by the owner. IT faculty decided that the assignment was registered if the student as well as the teacher agreed on this fact. The access control list for the service contained a manifold role $\{IT\}.assignment$. Security policy was described by a single credential:

$$\{IT\}.assignment \leftarrow \{IT\}.student \otimes \{IT\}.supervizor$$

Successful registration of student A to supervisor X resulted in adding the pair of entities $\{A, X\}$ to role $\{IT\}.superStudent$, which was done by issuing a new credential by the service:

$$\{IT\}.superStudent \leftarrow \{A, X\}$$

4. Course Registration. The service allowed a student to register for optional program. The owner of the service was a faculty, which defined the security policy for the service: The registration was valid when it was signed jointly by a supervisor and the assigned student. The access control list for the service contained manifold role $\{IT\}.superStudent$ issued by the previous service (Supervisor assignment).

5 System Architecture

An SOA system consists of a number of services that can be located within multiple separate systems from several business and administration domains, interconnected by a computer network. For example, particular instances of the services described in the previous section can be deployed to local servers, and administered by particular faculties of the university. Service clients, i.e. students and teachers, can invoke the services from remote, e.g. personal, computers.

Access rights to services established by the service owners can vary from one service instance to another. If the access rights are expressed through policies and described by sets of credentials, then those credentials must be stored somewhere in the network and presented for verification, at each invocation of a service. The verification of the access rights requires finding a credential chain, which confirms the membership of the requester in the role placed in the access control list of the service. This is a complex process, which can be performed by a special service, called trust management (TM) service, and invoked as part of the client's invocation. A general architecture of the system is shown in Fig. 1.

An important decision to make is to find the right place to store credentials. In practice, credentials can be stored by requesters, by issuers or in a known place in the network. SPKI/SDSI [5] as well as Akenti [6] assumes that credentials are presented by the application, i.e. by the invoked service in Fig. 1. One another possibility is to

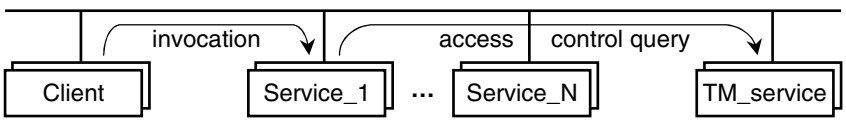

Fig. 1. Services and trust management servers in a SOA system

store at least part of credentials by a TM-service located in the administration domain. For example, each faculty can have a TM-service, which stores credentials issued by this faculty. However, credentials acting as personal certificates that define the attributes of particular entities can be stored by those entities, e.g., a credential:

$$\{IT\}.student \leftarrow \{A\}$$

can function as a student card, which is owned by the student. Credentials issued by particular entities, i.e. students and teachers in our example, to delegate permissions, can be stored by the subject entities.

After logging to a service, the client presents all the credentials, which are in his or her possession, and the service passes those credentials to the TM-service and asks for permission. TM-service builds the credential graph and resolves the query, using the presented credentials as well as the credentials stored in a local memory. TM-service can also look through the network in order to find other credentials.

Decentralized storage of credentials looks attractive, because it fits nicely into the general ideas of distribution and loose coupling that stand behind service-oriented architecture. However, the lack of control over the set of certificates exposes the system on a danger of inconsistency and raises the questions of certificate revocation, validity periods, etc. Neither of these questions can be answered by an analysis of the

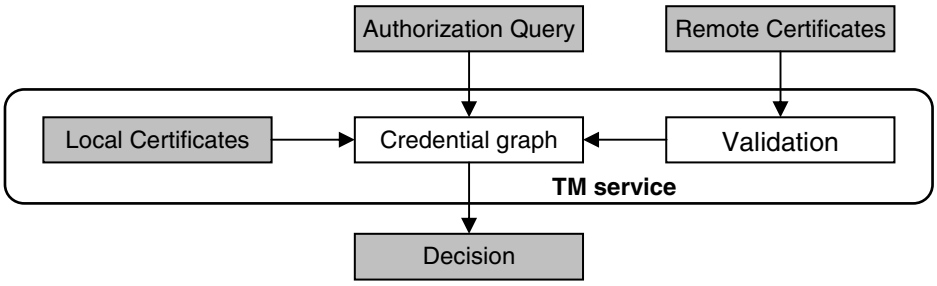

Fig. 2. The structure of a trust management (TM) service

credential graph. Therefore, the validity of certificates need to be assessed externally to the authorization logic [13]. Such a solution splits the process of resolving security queries into two separate layers, shown in Fig. 2, of the authorization logic, which performs a credential graph analysis, and the certificate validation, which searches through certificate revocation lists and verifies the validity periods with respect to the local time. This way, TM-service plays a part of a local certification authority, with the authority to decide, which credentials are taken into account, and which are not.

Single Sign-On. Consider another example adapted from [14]. There is a set of cooperating services (web sites), e.g. airline ticket reservation, car rental and hotel reservation. The users are identified on these sites by means of user accounts that contain access control data, such as user name and password, and other data, e.g. credit card

numbers. When a user requests access to more than one service, the authentication of his or her identity can be done separately on each site, or on one site that asserts the user status to the other services. The latter possibility, which permits the user to enter one name and password in order to use multiple services, is called a single sign-on property of the access control system. It is convenient to the user, who need not repeat a login procedure many times in sequence. The implementation of a single sign-on is among the main goals of Security Assertion Markup Language (SAML 2.0), an OASIS standard [14], which defines a language and a protocol to create, request and pass the identity assertions between the interrelated services.

Trust management is quite another concept, in which the access control data is distributed among credentials, and no user accounts and no login procedure are used. Instead, a user must present a set of credentials when invoking a service. Single sign-on can mean in this case, that the user is not forced to resend the same certificates many times in sequence, when invoking more than one service. The existence of a trust management service can help in solving this problem. When a user accesses the first service, the necessary certificates are sent to TM service. The certificates and the credential graph that is built in the memory of TM service are valid for a predefined period of time, and can be used within this period to resolve queries issued by another services. For example, a teacher at IT, in the previous example, who gained access to the grade book, need not resend $\{IT\}.teacher \leftarrow \{X\}$ credential to access the library.

6 Local Certification Authority

Supervisor Assignment service, described in Section 4, acts as a centralized issuer of the membership credentials for role $\{IT\}.superStudent$, which controls access to Course Registration service. Each member of this role is a pair composed of a student and the assigned supervisor. A decentralized approach to supervisor assignment could relay on credentials issued by supervisors to students, without any contribution of the faculty. For example, if supervisor X agrees to supervise students A and B, then he or she may issue the following credentials to confirm the assignment:

$\{X\}.myStudent \leftarrow \{A\}$
$\{X\}.myStudent \leftarrow \{B\}$

The faculty may also decentralize decision-making on who can deputize X, if X is temporarily unable to perform supervisory duties. To do this, new role $\{X\}.supervisor$ can be introduced, with the membership defined by means of credentials, like:

$\{X\}.supervisor \leftarrow \{X\}$
$\{X\}.supervisor \leftarrow \{Y\}$

If a faculty accepts such a decentralized approach to supervisor assignment, then Supervisor Assignment service becomes useless, and role $\{IT\}.superStudent$, which controls access to Course Registration service, can be defined by a credential:

$\{IT\}.superStudent \leftarrow \{IT\}.supervisor.(supervisor \otimes myStudent)$

Table 2. The meaning of new credentials in extended RT^T

Extensions to RT^T	Meaning of the credential
$A.r \leftarrow B.s(t \oplus u)$	$(\forall x)\ (\forall y)\ (\forall z)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(x.t) \wedge z \in \hat{S}(x.u) \Rightarrow y \cup z \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s(t \otimes u)$	$(\forall x)\ (\forall y)\ (\forall z)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(x.t) \wedge z \in \hat{S}(x.u) \wedge y \cap z = \phi \Rightarrow y \cup z \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s(t \cap u)$	$(\forall x)\ (\forall y)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(x.t) \wedge y \in \hat{S}(x.u) \Rightarrow y \in \hat{S}(A.r)\ )$

Credentials of this type, which join linking inclusion with other operators in a single role expression, do not exist in RT^T or any other Role based Trust management language. However, they can easily be added to the language with the semantics given by first-order formulae shown in Table 2.

The new types of credentials do not increase the expressive power of the language, however, they can help in reducing the number of roles and the number of credentials that are necessary to define a security policy. An equivalent definition of role $\{IT\}.superStudent$ in RT^T requires introduction of a new role and two credentials:

$\{X\}superStudent \leftarrow \{X\}.supervisor \otimes \{X\}.myStudent$
$\{IT\}.superStudent \leftarrow \{IT\}.supervisor.superStudent$

Decentralized approach to supervisor assignment raises a practical problem of resolving a conflict between two different teachers, say X and Y, who can independently agree to supervise a student, e.g. A, and issue credentials:

$\{X\}.myStudent \leftarrow \{A\}$
$\{Y\}.myStudent \leftarrow \{A\}$

Who of the two: X or Y is in this case responsible for signing course registration for student A? The problem can be solved if the supervisor assignment credentials are stored by TM-service (Fig. 1), which decides on the validity of credentials. Layered architecture of the access control mechanism resembles slightly a layered framework for modeling software and security policies introduced in [15].

7 Credential Graph

The semantics of a set P of credentials that define the security policy within a system can be represented by a credential graph. The nodes of this graph are role expressions, which are present in credentials, and the directed edges reflect inclusion of sets of groups of entities, which define the meaning of those expressions. Making a decision on the membership of a group X of entities in role $A.r$ is equivalent to checking whether a path from X to $A.r$ exists in the graph.

Let P be a set of extended RT^T credentials over a set E of entities and a set R of role names. A credential graph of P is defined in the following way.

Definition (Extended RT^T Credential Graph). Credential graph of a set P of extended RT^T credentials is a pair $G_P = (\ N_P, E_P\ )$ comprising a set N_P of nodes, which are role expressions that appear in credentials from P and groups of entities from E,

and a set E_P of directed edges, which are ordered pairs of nodes from N_P. The sets N_P and E_P are the smallest sets that are closed with respect to the following properties:

1) If $A.r \leftarrow e$, where e is a role expression, belongs to P, then the nodes $A.r$ and e belong to N_P and a *credential edge* (e, $A.r$) belongs to E_P.

2) If role expressions $B.s.t$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ belongs to N_P and a path from X to $B.s$ exists in G_P, a *derived edge* ($X.t$, $B.s.t$) belongs to E_P.

3) If role expressions $B.s \cap C.t$, $B.s$, and $C.t$ belong to N_P, then for each $X \subseteq E$, such that paths from X to $B.s$ and from X to $C.t$ exist in G_P, a *derived edge* (X, $B.s \cap C.t$) belongs to E_P.

4) If role expressions $B.s \oplus C.t$, $B.s$ and $C.t$ belong to N_P, then for each X, $Y \subseteq E$, such that paths from X to $B.s$ and from Y to $C.t$ exist in G_P, a *derived node* $X \cup Y$ belongs to N_P and a *derived edge* ($X \cup Y$, $B.s \oplus C.t$) belongs to E_P.

5) If role expressions $B.s \otimes C.t$, $B.s$ and $C.t$ belong to N_P, then for each X, $Y \subseteq E$, such that $X \cap Y = \phi$ and paths from X to $B.s$ and from Y to $C.t$ exist in G_P, a *derived node* $X \cup Y$ belongs to N_P and a *derived edge* ($X \cup Y$, $B.s \otimes C.t$) belongs to E_P.

6) If role expressions $B.s.(t \oplus u)$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ and $X.u$ belong to N_P, node $X.t \oplus X.u$ belongs to N_P. If a path from X to $B.s$ exists in G_P, then a *derived edge* ($X.t \oplus X.u$, $B.s.(t \oplus u)$) belongs to E_P.

7) If role expressions $B.s.(t \otimes u)$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ and $X.u$ belong to N_P, node $X.t \otimes X.u$ belongs to N_P. If a path from X to $B.s$ exists in G_P, and a *derived edge* ($X.t \otimes X.u$, $B.s.(t \otimes u)$) belongs to E_P.

8) If role expressions $B.s.(t \cap u)$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ and $X.u$ belong to N_P, node $X.t \cap X.u$ belongs to N_P. If a path from X to $B.s$ exists in G_P, then a *derived edge* ($X.t \cap X.u$, $B.s.(t \cap u)$) belongs to E_P.

Credential graph G_P of a set P of credentials consists of nodes and edges. Part of the nodes can be defined by a static analysis of credentials from P. These nodes, called static nodes, are roles, which stand at the left hand side of symbol $\leftarrow$, and role expressions, which appear in credentials at the right hand side of symbol $\leftarrow$. Nodes that are added according to properties 6 through 8 are counted as static nodes. Other nodes are created dynamically in the process of building the graph, by repetitive scanning through the set of credentials and executing role expressions with operators $\oplus$ and $\otimes$. These additional nodes are union sets of entities, added to the graph according to properties 4 and 5. Credential edges (property 1) are defined statically, while derived edges (properties 2 through 8) are added dynamically.

One can note that dynamically added nodes can be connected directly only to role expressions of type $B.s \oplus C.t$, $B.s \otimes C.t$ and $B.s \cap C.t$. In order to enhance the efficiency of the graph building algorithm, the necessary search for paths within the graph will be restricted to a subgraph composed of static nodes and all the edges between these nodes. If a path from a node of type $B.s \oplus C.t$, $B.s \otimes C.t$ or $B.s \cap C.t$ to a

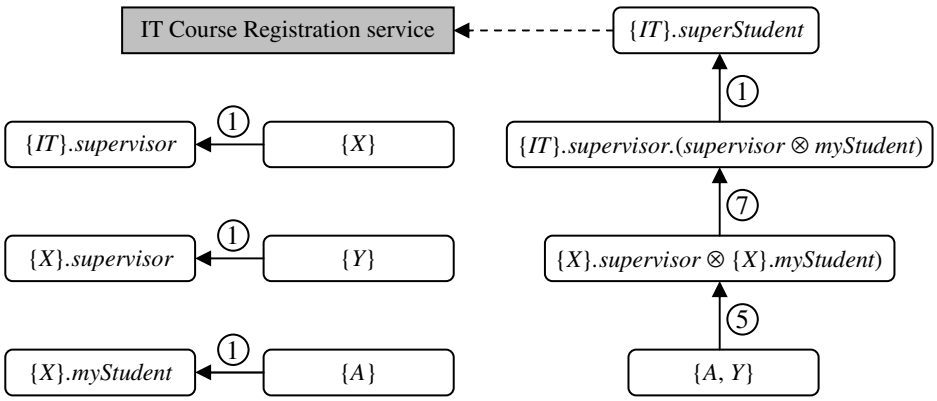

Fig. 3. Credential graph of the Course Registration service

certain node *N* is found, then paths from all groups of entities that are direct predecessors of this node to *N* are also considered.

Example. To observe the construction of a credential graph, assume that the credentials listed in Sections 4, excluding those related to Supervisor Assignment service, and the credentials listed in Section 6 have been issued, and an IT student *A* tries to register for an optional program.

To do this, *A* invokes Course Registration service of IT and presents a request signed jointly by *A* and *Y*. The access control list of the service contains a manifold role {*IT*}.*superStudent*, hence, the membership of pair {*A*, *Y*} in this role must be verified. The service calls TM-server, which looks through the accessible credentials and finds the following ones that are significant for resolving the query:

{*IT*}.*superStudent* ← {*IT*}.*supervisor.superStudent*
{*IT*}.*supervisor* ← {*X*}
{*X*}.*supervisor* ← {*Y*}
{*X*}.*myStudent* ← {*A*}

The authorization logic of the server builds a credential graph shown in Fig. 3. Small circled numbers placed near the edges of the graph refer to the numbers of properties in the definition of credential graph given above. After building the graph, TM-service verifies that a path from {*A*, *Y*} to {*IT*}.*superStudent* exists, and confirms authorization of *A* for registering for an optional course.

The complexity of the algorithm for building the credential graph of a set *P* of extended RT^T credentials can be evaluated with respect to the number of credentials in *P* (the cardinality of *P*), which is considered the input size of the problem. The method of evaluation is by assessing the complexity of each step of the algorithm and then counting the number of repetitions of particular steps. We assume that Dijkstra's algo-

rithm is used for finding paths between two nodes in the graph [16]. The complexity of this algorithm is $O(v^2)$, where v is the number of nodes.

Let n be the number of credentials in P and m be the number of role expressions other than roles and groups of entities in credentials in P. Obviously $m \leq n$. Moreover, let A, B, C, D, E, X, Y denote groups of entities from E.

The Algorithm (Creation of the Credential Graph)

1) For each credential $A.r \leftarrow e$ in P, add nodes $A.r$ and e to N_P and add an edge $(e, A.r)$ to E_P.

Remark. There are $2n$ nodes in the graph that has been built in step 1. The complexity of step 1 is of order $O(n)$.

2) For each node $B.s(t \oplus u)$, $B.s(t \otimes u)$ and $B.s.(t \cap u)$ in N_P, if there exist a pair of nodes $X.t$ and $X.u$ in N_P, where X is an arbitrary group of entities, then add node $X.t \oplus X.u$, $X.t \otimes X.u$ or $X.t \cap X.u$, respectively, to N_P.

Remark. The number of nodes $B.s(t \oplus u)$, $B.s(t \otimes u)$, $B.s.(t \cap u)$ is not greater than m. Hence, a search through N_P in order to find pairs of nodes $X.t$ and $X.u$ is repeated at most m times. The complexity of step 2 is of order $O(n^2)$.

The number of static nodes in N_P is not greater than $3n$.

Loop through the steps 3 through 6:

3) For each node $B.s.t$ find all the reverse paths (i.e. paths that start in $B.s$ and move along edges in the backward direction) from $B.s$ to the other nodes of the graph.
   - If a reverse path exists from $B.s$ to X and role $X.t$ belongs to N_P, then add an edge $(X.t, B.s.t)$ to E_P.
   - If a reverse path exists from $B.s$ to e, where e equals $D.u \oplus E.v$, $D.u \otimes E.y$ or $D.u \cap E.v$, then for each group X of entities, such that X is a direct predecessor of e and role $X.t$ belongs to N_P, add an edge $(X.t, B.s.t)$ to E_P.

Remark. The number of nodes $B.s$, which are the initial nodes in searching for paths, is not greater than the number m of role expressions $B.s.t$. Hence, the search for paths is repeated at most m times.

4) For each node $B.s \cap C.t$ find all the reverse paths from $B.s$ and from $C.t$ to the other nodes of the graph.
   - If a reverse path exists from $B.s$ to X and from $C.t$ to X, then add an edge $(X, B.s \cap C.t)$ to E_P.
   - If a reverse paths exist from $B.s$ to e_1 and from $C.t$ to e_2, where e_1 as well as e_2 are or role expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all groups of entities X that are direct predecessors of e_1 as well as of e_2 and add an edge $(X, B.s \cap C.t)$ to E_P.

Remark. The number of nodes $B.s$ and $C.t$, which are the initial nodes in searching for paths, is not greater than $2m$, hence, the search is repeated not more than $2m$ times.

5) For each node $B.s \oplus C.t$ and $B.s \otimes C.t$ find all the reverse paths from $B.s$ and from $C.t$ to the other nodes of the graph. Select all the pairs of nodes e_1, e_2, such that

paths from $B.s$ to e_1 and from $C.t$ to e_2 exist, and e_1 as well as e_2 are groups of entities or role expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$.

In case of expression $B.s \otimes C.t$, in the following three points take into account only those pairs X, Y, for which $X \cap Y = \phi$.

- If both nodes e_1 and e_2 are groups X and Y of entities, then add node $X \cup Y$ to N_P and add an edge $(X \cup Y, B.s \oplus C.t)$ or $(X \cup Y, B.s \otimes C.t)$, respectively, to E_P.
- If one node, e_1 or e_2, is a group X of entities, while the other node is an expression e, where e equals $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all the direct predecessors of e that are groups of entities. For each such group Y add node $X \cup Y$ to N_P and add an edge $(X \cup Y, B.s \oplus C.t)$ or $(X \cup Y, B.s \otimes C.t)$, respectively, to E_P.
- If both nodes e_1 and e_2 are expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all pairs X, Y of the direct predecessors: X of e_1 and Y of e_2, which are groups of entities. For each of such pair add node $X \cup Y$ to N_P and add an edge $(X \cup Y, B.s \oplus C.t)$ or $(X \cup Y, B.s \otimes C.t)$, respectively, to E_P.

Remark. The number of nodes $B.s$ and $C.t$, which are the initial nodes in searching for paths, is not greater than $2m$, hence, the search is repeated not more than $2m$ times.

6) For each node $B.s.(t \oplus u)$ and $B.s.(t \otimes u)$ find all the reverse paths from $B.s$ to the other nodes of the graph.
   - If a reverse path exists from $B.s$ to X and roles $X.t$ and $X.u$ exist in N_P, then add an edge $(X.t \oplus X.u, B.s.(t \oplus u))$ or $(X.t \otimes X.u, B.s.(t \otimes u))$, respectively, to E_P.
   - If a reverse path exists from $B.s$ to e, where e equals $D.u \oplus E.v$, $D.u \otimes E.y$ or $D.u \cap E.v$, then for each group X of entities, such that X is a direct predecessor of e and roles $X.t$ and $X.u$ exist in N_P, add an edge $(X.t \oplus X.u, B.s.(t \oplus u))$ or $(X.t \otimes X.u, B.s.(t \otimes u))$, respectively, to E_P.

Remark. The number of nodes $B.s$, which are the initial nodes in searching for paths, is not greater than m, hence, the search is repeated not more than m times.

The number of static nodes is not greater than $3n$ in a graph that is searched for paths in steps 3 through 6 of the above algorithm. Therefore, the complexity of finding the paths that begin in a given node is of order $O(n^2)$. The total number of nodes, which are the initial nodes in searching for paths in steps 3 through 6 is also not greater than $2n$. Hence, the search is repeated not more than $2n$ times, and the complexity of a single pass through the loop (steps 3 through 6) is of order $O(n^3)$.

A single pass through the loop corresponds to a single search through the set of n credentials. Each pass adds edges to the static part of the graph. The possibility of adding an edge depends on the existence of certain paths in the graph, which means that it depends on the sequence in which the credentials are processed. Repeating the loop n times guaranties that all the possible sequences of credentials have been exercised. Therefore, the complexity of the entire algorithm is of order $O(n^4)$.

8 Conclusions

The main issue of public key infrastructure has been to provide secure means of authentication entities, based on cryptographic methods and techniques. The next step in developing an approach to the application security could be a research on politics and procedures for authorizing the entities. Trust management is an attempt to define security policies in a decentralized way, based on a delegation of authority. This paper describes a set of trust management languages, discusses their expressive power, suggests an extension to Role-based Trust Managements language RT^T and evaluates the complexity of algorithm that is used for answering security queries in RT^T.

Trust management languages SPKI/SDSI, RT_0, RT^T and extended RT^T are built upon the same set of basic operators for role membership, role inclusion, linking inclusion and role intersection. SPKI/SDSI allows linking inclusion of arbitrary length, while the other languages allow linking inclusion of length two. This is not a significant difference, because a credential with linking inclusion of length n can easily be converted into $n-1$ credentials with the length of linking inclusion not greater than two.

RT^T and extended RT^T support manifold roles and role product operators that are not present in the other languages. This is a significant difference, because the added features allow expressing threshold and separation of duties policies. Extended RT^T allows symmetrical superposition of the linking operator and the other operators of role intersection and role product. This does not increase the expressive power of RT^T and is a 'syntactic sugar' that can help in reducing the number of credentials and thus the size of the access control problem. The selected features of the four trust management languages are shown in Table 3.

Table 3. Roles and role expressions in the four trust management languages and SAML

Language	Roles	Manifold roles	Symmetric linking
SPKI/SDSI	no	no	no
RT_0	yes	no	no
RT^T	yes	yes	no
Extended RT^T	yes	yes	yes
SAML 2.0	no	no	no

The security queries are resolved in trust management languages by building a credential graph and searching for a path within this graph. The complexity of building the graph has been proved polynomial of order $O(n^4)$, with respect to the number n of credentials at hand. It is interesting to observe that the order of complexity is the same for extended RT^T credential graph and for much simpler language used in SPKI/SDSI environment.

Our plans for further research include implementation of a prototype of a trust management service outlined in Section 5.

Acknowledgments. This research was supported in part by the Ministry of Science and Higher Education under the grant number 5321/B/T02/2010/39.

References

1. A guide to understanding discretionary access control in trusted systems. National Computer Security Center, NCSC-TG-003, Maryland (1987)
2. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-Based Access Control Models. IEEE Computer (2), 38–47 (1996)
3. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: 17th IEEE Symposium on Security and Privacy, pp. 164–173. IEEE Computer Society Press (1996)
4. Blaze, M., Feigenbaum, J., Ioannidis, J.: The KeyNote Trust Management System Version 2. Internet Society, Network Working Group, RFC 2704 (1999)
5. Clarke, D., Elien, J.-E., Ellison, C., Fredette, M., Morcos, A., Rivest, R.L.: Certificate chain discovery in SPKI/SDSI. J. Computer Security 9, 285–322 (2001)
6. Thompson, M.R., Essiari, A., Mudumbai, S.: Certificate-Based Authorization Policy in a PKI Environment. ACM Trans. Information and System Security 6(4), 566–588 (2003)
7. Li, N., Mitchell, J.: RT: A Role-Based Trust-Management Framework. In: 3rd DARPA Information Survivability Conference and Exposition, pp. 201–212. IEEE Computer Society Press (2003)
8. Li, N., Winsborough, W., Mitchell, J.: Distributed Credential Chain Discovery in Trust Management. J. Computer Security 1, 35–86 (2003)
9. Czenko, M., Etalle, S., Li, D., Winsborough, W.: An Introduction to the Role Based Trust Management Framework RT. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2007. LNCS, vol. 4677, pp. 246–281. Springer, Heidelberg (2007)
10. Felkner, A., Sacha, K.: The Semantics of Role-Based Trust Management Languages. In: 4th IFIP Central and East European Conference on Software Engineering Techniques, pp. 195–206 (2009)
11. Sacha, K.: Credential Chain Discovery in RT^T Trust Management Language. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2010. LNCS, vol. 6258, pp. 195–208. Springer, Heidelberg (2010)
12. Harel, D., Rumpe, B.: Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff. Weizmann Science Press of Israel, Jerusalem (2000)
13. Chapin, P., Skalka, C., Wang, X.: Authorization in Trust Management: Features and Foundations. ACM Comput. Survey 3, 1–48 (2008)
14. Ragouzis N. et al. (eds.) Security Assertion Markup Language (SAML) V2.0 Technical Overview. OASIS Committee Draft, March 2008. Document ID sstc-saml-tech-overview-2.0-cd-02 (2008), `http://www.oasis-open.org/committees/download.php/27819/`
15. Reith, M., Niu, J., Winsborough, W.: Engineering Trust Management into Software Models. In: International Workshop on Modeling in Software Engineering. IEEE Computer Society (2007)
16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press and McGraw-Hill (2001)