Contents

Chapter 1

Access Control

Sabrina De Capitani di Vimercati
Università degli Studi di Milano, Italy
`sabrina.decapitani@unimi.it`

Pierangela Samarati
Università degli Studi di Milano, Italy
`pierangela.samarati@unimi.it`

Ravi Sandhu
University of Texas at San Antonio, USA
`ravi.sandhu@utsa.edu`

1.1 Introduction

An important requirement of any information management system is to protect information against improper disclosure or modification (known as *confidentiality* and *integrity*, respectively). *Access control* is a fundamental technology to achieve this goal (e.g., [22, 26]). It controls every access request to a system and determines whether the access request should be granted or denied. In access control systems a distinction is generally made among *policies*, *models*, and *mechanisms*. Policies are high level guidelines which determine how accesses are controlled and access decisions determined. A policy is then formalized through a security model and is enforced by access control mechanisms. The access control mechanism works as a *reference monitor* that mediates every attempted access by a user (or program executing on behalf of that user) to objects in the system. The reference monitor consults an *authorization database* to determine if the user attempting to do an operation is actually authorized to perform that operation. Authorizations in this database are usually defined with respect to the identity of the users. This implies that access control requires *authentication* as a prerequisite, meaning that the identity of the requesting user has to be correctly verified (e.g., [8, 19, 25, 31]). The reader is surely familiar with the process of signing on to a computer system by providing an identifier and a password. This authentication establishes the identity of a human user to a computer. More generally, authentication can be computer-to-computer or process-to-process and mutual in both directions. Access control is then concerned with limiting the activity of legitimate users who have been successfully authenticated. The set of authorizations are administered and maintained by a security administrator. The administrator sets the authorizations on the basis of the security policy of the organization. Users may also be able to modify some portion of the authorization database, for example, to set permissions for their personal files. The effectiveness of the access control rests on a proper user identification and on the correctness of the authorizations governing the reference monitor.

The variety and complexity of the protection requirements that may need to be imposed in today's systems makes the definition of access control policies a far from trivial process. For instance, many services do not need to know the real identity of a user but they may need to know some characteristics/properties of the requesting users (e.g., a user can access a service only if she work in an European country), or different systems may need to collaborate while preserving their autonomy in controlling access to their resources [23, 24, 26]. The goal of this chapter is therefore to provide an overview of the access control evolution. This overview begins with a discussion on the classical discretionary, mandatory, and role-based access control policies and models (Sections 1.2-1.4), and then continue with an illustration of the administration policies (Section 1.5) that determine who can modify the accesses allowed by such policies. We then discuss the most recent advances in access control, focusing on attribute and credential-based access control (Section 1.6). Finally, we present our conclusions in Section 1.7.

1.2 Discretionary access control (DAC)

Discretionary access control policies govern the access of users to the information/resources on the basis of the users' identities and authorizations (or rules) that specify, for each user (or group of users) and each object in the system, the access modes the user is allowed on the object. Each request of a user to access an object is checked against the specified authorizations. If there exists an authorization stating that the user can access the object in the specific mode, the access is granted; it is denied, otherwise. In the following, we first describe the access matrix model, which is useful to understand the basic principles behind discretionary access control models and policies, and discuss implementation alternatives. We then illustrate how discretionary policies have been further expanded.

1.2.1 The access matrix model and its implementation

A first step in the definition of an access control model is the identification of the set of *objects* to be protected, the set of *subjects* who request access to objects, and the set of *access modes* that can be executed on objects. While subjects typically correspond to users (or groups thereof), objects and access modes may differ depending on the specific

	File1	File2	File3	File4	Account1	Account2
John	Own R W		Own R W		Inquiry Credit	
Alice	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Bob	R W	R		Own R W		Inquiry Debit

FIGURE 1.1: An example of access matrix

system or application. For instance, objects may be files and access modes may be Read, Write, Execute, and Own. The meaning of the first three of these access modes is self evident. Ownership is concerned with controlling who can change the access permissions for the file. An object such as a bank account may have access modes Inquiry, Credit and Debit corresponding to the basic operations that can be performed on an account. These operations would be implemented by application programs, whereas for a file the operations would typically be provided by the Operating System.

A subtle point that is often overlooked is that subjects can themselves be objects. A subject can create additional subjects to accomplish its task. The children subjects may be executing on various computers in a network. The parent subject will usually be able to suspend or terminate its children as appropriate. The fact that subjects can be objects corresponds to the observation that the initiator of one operation can be the target of another. (In network parlance, subjects are often called initiators, and objects called targets.)

The access matrix is a conceptual model which specifies the rights that each subject possesses for each object. There is a row in the matrix for each subject, and a column for each object. Each cell of the matrix specifies the access authorized for the subject in the row to the object in the column. The task of access control is to ensure that only those operations authorized by the access matrix actually get executed. This is achieved by means of a *reference monitor*, which is responsible for mediating all attempted operations by subjects on objects.

Figure 1.1 shows an example of access matrix where the access modes R and W denote read and write, respectively, and the other access modes are as discussed previously. The subjects shown here are John, Alice, and Bob. There are four files and two accounts. This matrix specifies that, for example, John is the owner of File3 and can read and write that file, but John has no access to File2 and File4. The precise meaning of ownership varies from one system to another. Usually the owner of a file is authorized to grant other users access to the file, as well as revoke access. Since John owns File1, he can give Alice the R right and Bob the R and W rights as shown in Figure 1.1. John can later revoke one or more of these rights at his discretion.

The access modes for the accounts illustrate how access can be controlled in terms of abstract operations implemented by application programs. The Inquiry operation is similar to read since it retrieves information but does not change it. Both the Credit and Debit operations will involve reading the previous account balance, adjusting it as appropriate and writing it back. The programs which implement these operations require read and write access to the account data. Users, however, are not allowed to read and write the account object directly. They can manipulate account objects only indirectly via application programs which implement the Debit and Credit operations. Also, note that there is no

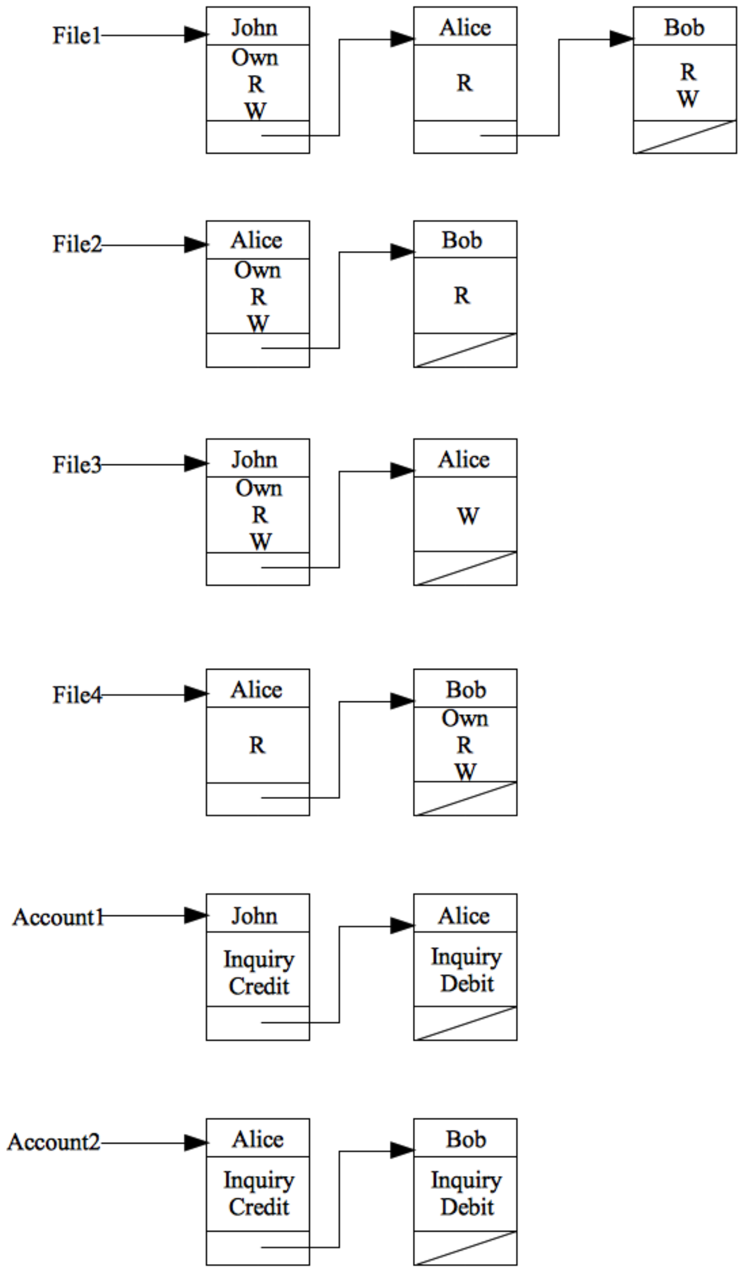

FIGURE 1.2: Access Control Lists corresponding to the access matrix in Figure 1.1

Own right for accounts. Objects such as bank accounts do not really have an owner who can determine the access of other subjects to the account. Clearly, the user who establishes the account at the bank should not be the one to decide who can access the account. Within the bank different officials can access the account depending on their job functions in the organization.

In a large system the access matrix will be enormous in size, and most of its cells are likely to be empty. Accordingly, the access matrix is very rarely implemented as a matrix.

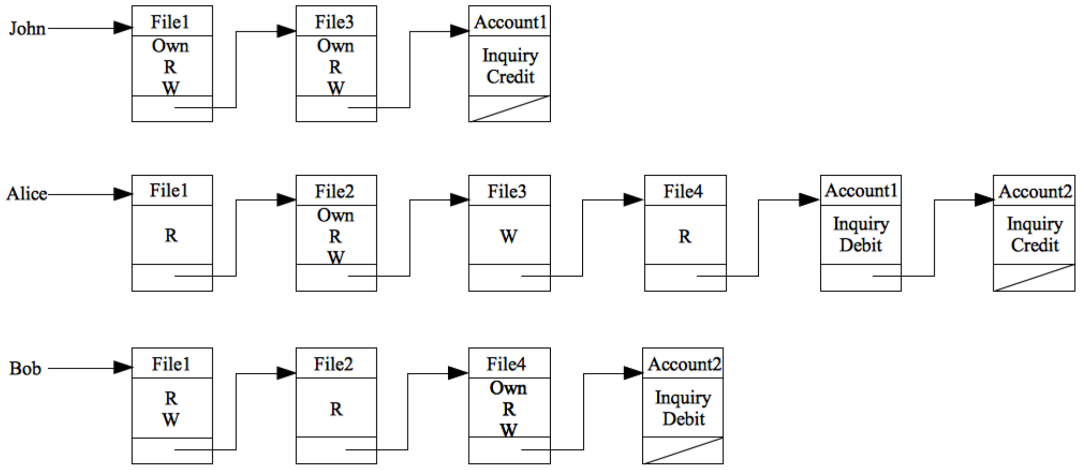

FIGURE 1.3: Capability lists corresponding to the access matrix in Figure 1.1

We now discuss some common approaches for implementing the access matrix in practical systems.

Access Control Lists. Each object is associated with an Access Control List (ACL), indicating for each subject in the system the accesses the subject is authorized to execute on the object. This approach corresponds to storing the matrix by columns. ACLs corresponding to the access matrix of Figure 1.1 are shown in Figure 1.2. Essentially the access matrix column for File1 is stored in association with File1, and so on.

By looking at an object's ACL, it is easy to determine which access modes subjects are currently authorized for that object. In other words, ACLs provide for convenient access review with respect to an object. It is also easy to revoke all accesses to an object by replacing the existing ACL with an empty one. On the other hand, determining all the accesses that a subject has is difficult in an ACL-based system. It is necessary to examine the ACL of every object in the system to do access review with respect to a subject. Similarly, if all accesses of a subject need to be revoked, all ACLs must be visited one by one. (In practice revocation of all accesses of a subject is often done by deleting the user account corresponding to that subject. This is acceptable if a user is leaving an organization. However, if a user is reassigned within the organization it would be more convenient to retain the account and change its privileges to reflect the changed assignment of the user.)

Many systems allow group names to occur in ACLs (see Section 1.2.2). For instance, an entry such as (ISSE, R) can authorize all members of the ISSE group to read a file. Several popular Operating Systems (e.g., Unix) implement an abbreviated form of ACLs in which a small number, often only one or two, group names can occur in the ACL. Individual subject names are not allowed. With this approach the ACL has a small fixed size so it can be stored using a few bits associated with the file.

Capabilities. Capabilities are a dual approach to ACLs. Each subject is associated with a list, called capability list, indicating for each object in the system, the accesses the subject is authorized to execute on the object. This approach corresponds to storing the access matrix by rows. Figure 1.3 shows the capability lists corresponding to the access matrix in Figure 1.1. In a capability list approach it is easy to review all accesses that a subject is authorized to perform by simply examining the subject's capability list. However, determination of all subjects who can access a particular object requires examination of each

Subject	Access mode	Object
John	Own	File1
John	R	File1
John	W	File1
John	Own	File3
John	R	File3
John	W	File3
John	Inquiry	Account1
John	Debit	Account1
Alice	R	File1
Alice	Own	File2
Alice	R	File2
Alice	W	File2
Alice	W	File3
Alice	R	File4
Alice	Inquiry	Account1
Alice	Debit	Account1
Alice	Inquiry	Account2
Alice	Credit	Account2
Bob	R	File1
Bob	W	File1
Bob	R	File2
Bob	Own	File4
Bob	R	File4
Bob	W	File4
Bob	Inquiry	Account2
Bob	Debit	Account2

FIGURE 1.4: Authorization relation corresponding to the access matrix in Figure 1.1

and every subject's capability list. A number of capability-based computer systems were developed in the 1970s, but did not prove to be commercially successful. Modern operating systems typically take the ACL-based approach.

It is possible to combine ACLs and capabilities. Possession of a capability is sufficient for a subject to obtain access authorized by that capability. In a distributed system this approach has the advantage that repeated authentication of the subject is not required. This allows a subject to be authenticated once, obtain its capabilities and then present these capabilities to obtain services from various servers in the system. Each server may further use ACLs to provide finer-grained access control.

Authorization relations. We have seen that ACL- and capability-based approaches have dual advantages and disadvantages with respect to access review. There are representations of the access matrix which do not favor one aspect of access review over the other. For instance, the access matrix can be represented by an authorization relation (or table) as shown in Figure 1.4. Each row, or tuple, of this table specifies one access right of a subject to an object. Thus, John's accesses to File1 require three rows. If this table is sorted by subject, we get the effect of capability lists. If it is sorted by object, we get the effect of ACLs. Relational database management systems typically use such a representation.

1.2.2 Expanding authorizations

Although the access matrix still remains a framework for reasoning about accesses permitted by a discretionary policy, discretionary policies have been developed considerably

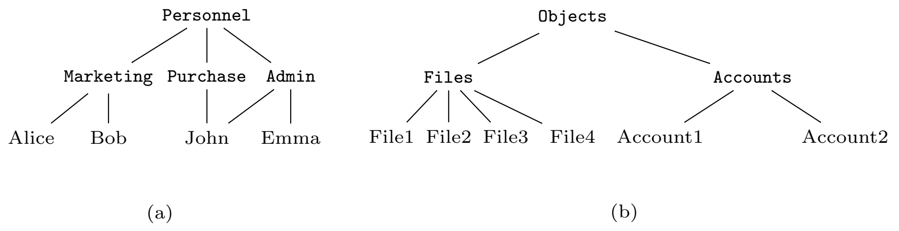

FIGURE 1.5: An example of user group (a) and object (b) hierarchy

since the access matrix was proposed. In particular, early approaches to authorization specifications allowed *conditions* to be associated with authorizations to restrict their validity [33]. Conditions may involve some system predicates, and may specify restrictions based on the content of objects or on accesses previously executed. Another important feature supported by current discretionary policies is the definition of abstractions on users and objects. Both users and objects can therefore be hierarchically organized, thus introducing *user groups* and *classes of objects*. Figures 1.5(a)-(b) illustrate an example of user group hierarchy and object hierarchy, respectively. The definition of groups of users (and classes of objects) requires a technique to easily handle exceptions. For instance, suppose that all users belonging to a group can access a specific object but user u. In this case, it is necessary to explicitly associate an authorization with each user in the group but u, which is clearly a solution that does not take advantage from the definition of user groups. This observation has been the driving factor supporting the development of access control models that combine *positive* and *negative* authorizations (e.g., [10]). In this way, the previous exception can be easily modeled by the definition of two authorizations: a positive authorization for the group and a negative authorization for user u. Hierarchies can also simplify the definition of authorizations because authorizations specified on an abstraction can be propagated to all its members. The propagation of authorizations over a hierarchy may follow different *propagation policies* (e.g., [35, 52]). For instance, authorizations associated with an element in the hierarchy may not be propagated, may be propagated to all its descendants, or may be propagated to its descendants if not overridden.

The use of both positive and negative authorizations introduces two problems: *i) inconsistency*, which happens when for an access there are both a negative and a positive authorization; and *ii) incompleteness*, which happens when some accesses are neither authorized nor denied (i.e., no authorization exists for them). The inconsistency problem is solved by applying a *conflict resolution policy*. There are several conflict resolution policies [35, 45] such as: *no conflict*, the presence of a conflict is considered an error; *denials take precedence*, negative authorizations take precedence; *permissions take precedence*, positive authorizations take precedence; *nothing takes precedence*, neither positive nor negative authorizations take precedence and conflicts remain unsolved; *most specific takes precedence*, the authorization that is more specific with respect to a hierarchy wins (e.g., consider the user group hierarchy in Figure 1.5(a), the effect on John of a positive authorization for the `Admin` group to read File1, and a negative authorization for reading the same file for `Personnel` is that he is allowed to read File1 since `Admin` is more specific than `Personnel`); and *most specific along a path takes precedence*, the authorization that is more specific with respect to a hierarchy wins only on the paths passing through it (e.g., consider the user group hierarchy in Figure 1.5(a), the effect on John of a positive authorization for the `Admin` group to read File1, and a negative authorization for reading the same file for `Personnel` is that there is a conflict for managing John's access to File1 since the negative authorization wins along path ⟨`Personnel`, `Purchase`, John⟩ and the positive authorization wins along path ⟨`Personnel`, `Admin`, John⟩). The incompleteness problem can be solved by adopting a *decision policy*, that is, an *open policy* or a *closed policy.* Open policies are based on explicitly specified authorizations and the default decision of the reference monitor is denial. Open policies are based on the specification of denials instead of permissions. In this case, for each user and each object in the system, the access modes the user is forbidden on the object are specified. Each access request by a user is checked against the specified (negative) authorizations and granted only if no authorization denying the access exists. The combination of a propagation policy, a conflict resolution policy, and a decision policy guarantees a complete and consistent policy for the system.

1.3 Mandatory access control (MAC)

The flexibility of discretionary policies makes them suitable for a variety of systems and applications, especially in the commercial and industrial environments. However, discretionary access control policies have the drawback that they do not provide real assurance on the flow of information in a system. It is easy to bypass the access restrictions stated through the authorizations. For instance, a user who is authorized to read data can pass them to other users not authorized to read them without the cognizance of the owner. The reason is that discretionary policies do not impose any restriction on the usage of information by a user once the user has got it (i.e., dissemination of information is not controlled). By contrast, dissemination of information is controlled in mandatory systems by preventing flow of information from high-level objects to low-level objects. Mandatory policies govern access on the basis of a classification of subjects and objects in the system. Note that the concept of subject in discretionary policies is different from the concept of subject in mandatory policies. In fact, authorization subjects correspond to users (or groups thereof) while in mandatory policies users are human beings who can access the system, and subjects are processes operating on behalf of users.

Each subject and each object in a mandatory system is assigned an *access class*. The set of access classes is a partially ordered set and in most cases an access class is composed by a *security level* and a set of *categories.* The security level is an element of a hierarchical ordered set. In the military and civilian government arenas, the hierarchical set generally consists of Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U), where TS $>$ S $>$ C $>$ U. Each security level is said to dominate itself and all others below it in this hierarchy. The set of categories is a subset of an unordered set, whose elements reflect functional, or competence, areas (e.g., Financial, Demographic, Medical). Given two access classes c_1 and c_2, we say that c_1 dominates c_2, denoted $c_1 \succeq c_2$, iff the security level of c_1 is greater than or equal to that of c_2 and the categories of c_1 include those of c_2.

The semantics and use of the access classes assigned to objects and subjects within the application of a multilevel mandatory policy depends on whether the access class is intended for a secrecy or an integrity policy. In the following, we illustrate secrecy-based and integrity-based mandatory policies.

1.3.1 Secrecy-based model

The main goal of a secrecy-based mandatory policy is to protect the confidentiality of information. In this case, the security level associated with an object reflects the sensitivity

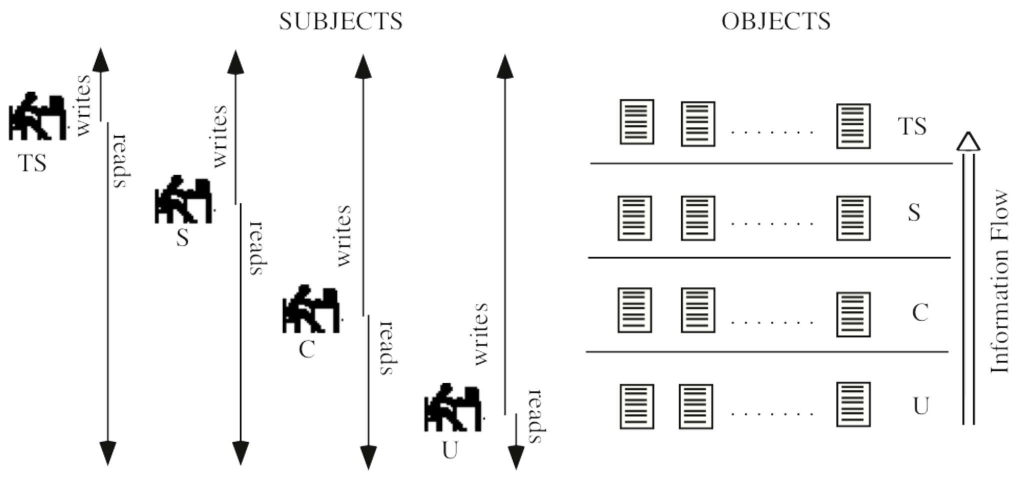

FIGURE 1.6: Controlling information flow for secrecy

of the information contained in the object, that is, the potential damage which could result from unauthorized disclosure of the information. The security level associated with a user, also called *clearance*, reflects the user's trustworthiness not to disclose sensitive information to users not cleared to see it. The set of categories associated with a user reflect the specific areas in which the user operates. The set of categories associated with an object reflect the area to which information contained in the object is referred. Categories enforce restriction on the basis of the need-to-know principle (i.e., a subject should be only given those accesses which are required to carry out the subject's responsibilities). Users can connect to the system at any access class dominated by their clearance. A user connecting to the system at a given access class originates a subject at that access class.

Access to an object by a subject is granted only if some relationship (depending on the type of access) is satisfied between the access classes associated with the two. In particular, the following two principles are required to hold.

Read-down A subject's access class must dominate the access class of the object being read.

Write-up A subject's access class must be dominated by the access class of the object being written.

Satisfaction of these principles prevents information in high level objects (i.e., more sensitive) to flow to objects at lower levels. Figure 1.6 illustrates the effect of these rules. Here, for simplicity, accesses classes are only composed of a security level. In such a system, information can only flow upwards or within the same security class.

To better understand the rationale behind the read-down and write-up rules, it is important to analyze the relationship between users and subjects in this context. Suppose that the human user Jane is cleared to S (again, for simplicity, access classes are only composed

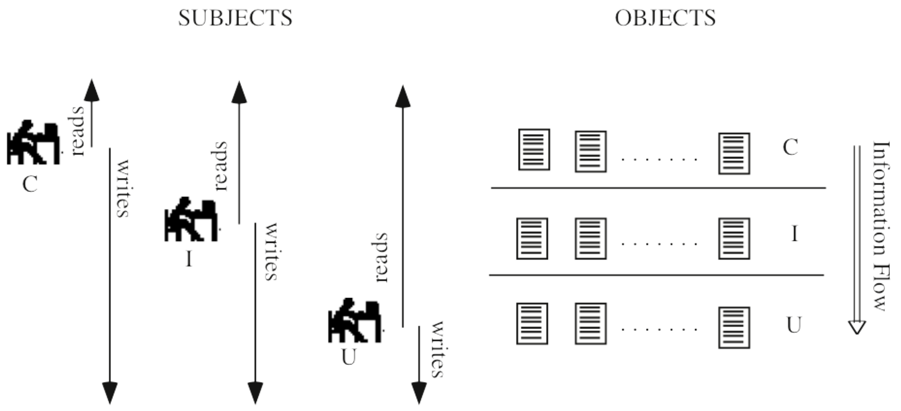

FIGURE 1.7: Controlling information flow for integrity

of a security level), and assume she always signs on to the system as an S subject (i.e., a subject with clearance S). Jane's subjects are prevented from reading TS objects by the read-down rule. The write-up rule, however, has two aspects that seem at first sight contrary to expectation.

- Firstly, Jane's S subjects can write a TS object (even though they cannot read it). In particular, they can overwrite existing TS data and therefore destroy them. Due to this integrity concern, many systems for mandatory access control do not allow write-up; but limit writing to the same level as the subject. At the same time write-up does allow Jane's S subjects to send electronic mail to TS subjects, and can have its benefits.

- Secondly, Jane's S subjects cannot write C or U data. This means, for example, that Jane can never send electronic mail to C or U users. This is contrary to what happens in the paper world, where S users can write memos to C and U users. This seeming contradiction is easily eliminated by allowing Jane to sign to the system as a C, or U, subject as appropriate. During these sessions she can send electronic mail to C or, U and C, subjects respectively.

The write-up rule prevents malicious software from leaking secrets downward from S to U. Users are trusted not to leak such information, but the programs they execute do not merit the same degree of trust. For instance, when Jane signs onto the system at U level her subjects cannot read S objects, and thereby cannot leak data from S to U. The write-up rule also prevents users from inadvertently leaking information from high to low.

1.3.2 Integrity-based model

Mandatory access control can also be applied for the protection of information integrity. The integrity level associated with an object reflects the degree of trust that can be placed in the information stored in the object, and the potential damage that could result from unauthorized modification of the information. The integrity level associated with a user reflects the user's trustworthiness for inserting, modifying or deleting data and programs at that level. Again, categories define the area of competence of users and data. Principles similar to those stated for secrecy are required to hold, as follows.

Read-up A subject's integrity class must be dominated by the integrity class of the object being read.

Write-down A subject's integrity class must dominate the integrity class of the object being written.

Satisfaction of these principles safeguard integrity by preventing information stored in low objects (and therefore less reliable) to flow to high objects. This is illustrated in Figure 1.7, where, for simplicity, integrity classes are only composed of integrity levels, which can be Crucial (C), Important (I), and Unknown (U) with C>I>U. Controlling information flow in this manner is only one aspect of achieving integrity. Integrity in general requires additional mechanisms, as discussed in [16, 53].

Note that the only difference between Figure 1.6 and Figure 1.7 is the direction of information flow, being bottom to top in the former case and top to bottom in the latter. In other words, both cases are concerned with one-directional information flow. The essence of classical mandatory controls is one-directional information flow in a lattice of security classes. For further discussion on this topic see [52].

1.4 Role-based access control (RBAC)

Role-based policies regulate the access of users to the information on the basis of the activities the users execute in the system. Role-based policies require the identification of roles in the system. A role can be defined as a set of actions and responsibilities associated with a particular working activity. Then, instead of specifying all the accesses each user is allowed to execute, access authorizations on objects are specified for roles. Users are given authorizations to play roles. This greatly simplifies the authorization management task. For instance, suppose a user's responsibilities change, say, due to a promotion. The user's current roles can be taken away and new roles assigned as appropriate for the new responsibilities. Another advantage of RBAC is that roles allow a user to sign on with the least privilege required for the particular task at hand. Users authorized to powerful roles do not need to exercise them until those privileges are actually needed. This minimizes the danger of damage due to inadvertent errors or by intruders masquerading as legitimate users. Note that, in general, a user can take on different roles on different occasions. Also, the same role can be played by several users, perhaps simultaneously. Some proposals for role-based access control allow a user to exercise multiple roles at the same time. Other proposals limit the user to only one role at a time, or recognize that some roles can be jointly exercised while others must be adopted in exclusion to one another. In the remainder of this section, we briefly describe the standard RBAC model and then illustrate an extension of RBAC.

1.4.1 Basic RBAC model

In 2004, the National Institute of Standards and Technology (NIST) proposed a U.S. national standard for role-based access control through the International Committee for Information Technology Standards (ANSI/INCITS) [3, 28]. The standard RBAC model is organized in four components, which are briefly described in the following.

Core RBAC. Core RBAC includes five basic elements, that is, users, roles, objects, operations, and permissions. Users are assigned to roles, and permissions (i.e., the association between an object and an operation executable on the object) are assigned to roles.

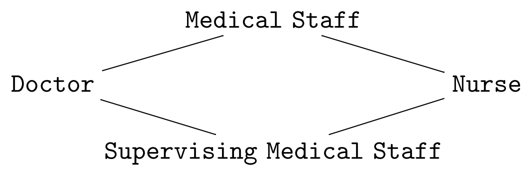

FIGURE 1.8: An example of role hierarchy

The permission-to-role and user-to-role assignments are both many-to-many, thus providing greater flexibility. In addition, Core RBAC includes the concept of session, where each session is a mapping between a user and a set of activated roles, which are a subset of the roles assigned to the user. The permissions available to a user during a session are therefore all the permissions associated with the roles activated by the user in the session.

Hierarchical RBAC. This model introduces the concept of role hierarchy. In many applications there is a natural hierarchy of roles, based on the familiar principles of generalization and specialization. For instance, the `Medical Staff` role may be specialized into `Doctor` and `Nurse` (see Figure 1.8). The role hierarchy has implication on role activation and access privileges: a specialized role inherits the authorizations of its generalizations; also, users authorized to activate a role inherit the authorization to activate the generalizations of the role. For instance, with reference to Figure 1.8, role `Nurse` inherits all authorizations of role `Medical Staff`. Also, users authorized to activate role `Nurse` will also be allowed to activate role `Medical Staff`. A role hierarchy can be an arbitrary partial order or it may be possible to impose some restrictions on the type of hierarchy. For instance, it is possible to require that a role hierarchy has to be a tree, meaning that each role may have only one single direct parent.

Static separation of duty. The RBAC model can be enriched by adding separation of duty constraints. Intuitively, static separation of duty imposes restrictions on the assignments of users to roles. For instance, a user assigned to the `Nurse` role may not be assigned to the `Doctor` role. A separation of duty constraint is defined as a pair where the first element is a set rs of roles and the second element is an integer n greater than or equal to two denoting the number of roles that would constitute a violation. For instance, $\langle\{$`Doctor`, `Nurse`$\}$,2$\rangle$ states that a user may be assigned to one out of the two roles mentioned in the constraint. Since static separation of duty constraints can be also defined in the presence of a role hierarchy, it is important to ensure that role inheritance does not violate the static separation of duty constraints.

Dynamic separation of duty. In the dynamic separation of duty, constraints are enforced at run-time: users can activate any role to which they are assigned but the activation of some roles during a session will rule out their ability to activate another role which is in a separation of duty constraint with the activated roles. An example of dynamic separation of duty is the two-person rule. The first user who executes a two-person operation can be any authorized user, whereas the second user can be any authorized user different from the first. The dynamic separation of duty constraints are still defined as pairs composed by a set rs of roles and an integer n greater than or equal to two and their enforcement requires that no user is assigned to n or more roles from rs in a single session.

In addition to the four components described above, the NIST RBAC standard defines administrative functions related to the creation and management of the basic elements and relations of RBAC, and permission review functions.

1.4.2 Expanding RBAC

The RBAC model has been adopted in a variety of commercial systems (e.g., DB2, Oracle), and has been the subject of many research proposals aimed at extending and enriching the model to support particular domain specific security policies (e.g., web services, social networks), administration models, and delegation. Some proposals are also aimed at integrating RBAC with other technologies such as cryptography, trust mechanisms, and XML-based access control languages (e.g., [1, 30]). Given the huge amount of work on RBAC, it is clearly not feasible to provide a comprehensive summary of the extensions proposed in the literature. A notable example is the *Usage Control Model* (UCON$_{ABC}$) [51], a framework that encompasses access control, trust management, and digital right management. This proposal is interesting since it supports DAC, MAC, and RBAC and is based on attributes characterizing subjects and objects that are used for specifying authorizations (see Section 1.6 for more details about attribute-based access control). In particular, the UCON$_{ABC}$ model integrates authorizations, obligations, and conditions within a unique framework, and includes the following eight components.

Subjects, *objects*, and *rights*. Subjects, objects, and rights have the same meaning of the corresponding concepts also used within the DAC, MAC, and RBAC access control models. A subject is therefore an individual who holds some rights on objects.

Subject and *object attributes.* Each subject and object in the system is characterized by a set of attributes that can be used for verifying whether an access request can be granted. Examples of subject attributes include identities, group names, and roles while examples of object attributes include ownerships, security labels, and so on. A peculiarity of the UCON$_{ABC}$ model is that subject and object attributes can be mutable, meaning that their values may change due to an access.

Authorizations. Authorizations are evaluated for usage decision and return whether a subject can perform the required right on an object. Authorizations are based on subject and object attributes, and are distinguished between pre-authorizations and ongoing-authorizations. A pre-authorization is performed before the execution of the requested right while an ongoing-authorization is performed during the access.

Obligations. An obligation is a requirement that a user has to perform before (pre) or during (ongoing) access. For instance, a pre-obligation may require a user to provide her date of birth before accessing a service. The execution of obligations may change the value of mutable attributes and therefore they may affect current or future usage decisions.

Conditions. Conditions evaluate current environmental or system status. Examples are time of the day and system workload. The evaluation of these conditions cannot change the value of any subject or object attributes.

A family of UCON$_{ABC}$ core models is defined according to three criteria: the decision factor, which may be authorizations, obligations, and conditions; the continuity of decision, which may be either pre or ongoing; and the mutability, which can allow changes on subject or object attributes at different times. According to these criteria, the authors in [51] define 16 basic UCON$_{ABC}$ models and show how these models can support traditional DAC, MAC, and RBAC.

1.5 Administration of authorizations

Administrative policies determine who is authorized to modify the allowed accesses. This is one of the most important, and least understood, aspects of access control.

In mandatory access control the allowed accesses are determined entirely on the basis of the access class of subjects and objects. Access classes are assigned to users by the security administrator. Access classes of objects are determined by the system on the basis of the access classes of the subjects creating them. The security administrator is typically the only one who can change access classes of subjects or objects. The administrative policy is therefore very simple.

Discretionary access control permits a wide range of administrative policies. Some of these are described below.

- *Centralized.* A single authorizer (or group) is allowed to grant and revoke authorizations to the users.

- *Hierarchical.* A central authorizer is responsible for assigning administrative responsibilities to other administrators. The administrators can then grant and revoke access authorizations to the users of the system. Hierarchical administration can be applied, for example, according to the organization chart.

- *Cooperative.* Special authorizations on given resources cannot be granted by a single authorizer but need cooperation of several authorizers.

- *Ownership.* A user is considered the owner of the objects he/she creates. The owner can grant and revoke access rights for other users to that object.

- *Decentralized.* In decentralized administration the owner of an object can also grant other users the privilege of administering authorizations on the object.

Within each of these there are many possible variations [52].

Role-based access control has a similar wide range of possible administrative policies. In this case roles can also be used to manage and control the administrative mechanisms.

Delegation of administrative authority is an important aspect in the administration of authorizations. In large distributed systems centralized administration of access rights is infeasible. Some existing systems allow administrative authority for a specified subset of the objects to be delegated by the central security administrator to other security administrators. For instance, authority to administer objects in a particular region can be granted to the regional security administrator. This allows delegation of administrative authority in a selective piecemeal manner. However, there is a dimension of selectivity that is largely ignored in existing systems. For instance, it may be desirable that the regional security administrator be limited to granting access to these objects only to employees who work in that region. Control over the regional administrators can be centrally administered, but they can have considerable autonomy within their regions. This process of delegation can be repeated within each region to set up sub-regions and so on.

1.6 Attribute and credential-based access control

Emerging distributed scenarios (e.g., cloud computing and data outsourcing) are typically characterized by several independent servers offering services to anyone who needs them (e.g., [21]). In such a context, traditional assumptions for enforcing access control do not hold anymore. As a matter of fact, an access request may come from unknown users and therefore access control policies based on the identity of the requester cannot be applied. Alternative solutions that have been largely investigated in the last twenty years consist in adopting *attribute-based access control* that uses the attributes associated with the resources/services and requesters to determine whether the access should be granted (e.g., [5]). The basic idea is that not all access control decisions are identity-based. For instance, information about a user's current role (e.g., doctor) or a user's date of birth may be more important than the user's identity for deciding whether an access request should be granted.

In the remainder of this section, we first review the basic concepts about attribute and credential-based access control and then describe some solutions based on such a model.

1.6.1 Basic elements of the model

Attribute-based access control differs from traditional discretionary access control since both the subject and the object appearing in an authorization are replaced by a set of *attributes* associated with them. Such attributes may correspond to an identity or a non-identifying characteristic of a user (e.g., date of birth, nationality) and to metadata associated with an object that provide additional context information (e.g., data of creation). In particular, the attributes associated with a user may be specified by the user herself (*declarations*), or may be substantiated by *digital certificates* or *credentials* (e.g., [12, 57]). Metadata associated with resources/services can be in different form (e.g., textual or semistructured data). By analyzing previous works in the area, we identify the following main concepts captured by attribute/credential-based access control models.

Authority. An authority is an entity responsible for producing and signing certificates. A party may accept certificates issued by an authority that it trusts or that has been (directly or indirectly) delegated by an authority that it trusts.

Certificate. A certificate is a statement certified by an authority trusted for making such a statement. Each certificate is characterized by the identity of the issuer, the identity of the user for which the certificate has been issued, a validity period, a signature of the issuing authority, and a set of certified attributes. Certificates can be classified according to different dimensions. Since we focus on the use of certificates in access control, we distinguish between *atomic* and *non-atomic* certificates [7]. Atomic certificates (e.g., X.509) can only be released as a whole, meaning that all attributes in a certificate are disclosed. These certificates are the most common type of certificates used today in distributed systems. Although these certificates can only be released as a whole, there is usually the possibility to refer to specific attributes within a certificate for querying purpose. Given a certificate c including a set $\{a_1, \ldots, a_m\}$ of attributes, we can use the dot notation to refer to a given attribute in c. For instance, given credential `Passport` certifying attributes `name`, `dob`, and `country`, `Passport.name` denotes attribute `name` certified by the `Passport` credential. Non-atomic certificates (e.g., Idemix [15]) allow the selective release of the attributes certified by them. Non-atomic certificates are based on technologies that also permit to certify the possession of a given certificate without disclosing the attributes within it. Abstractions can be defined

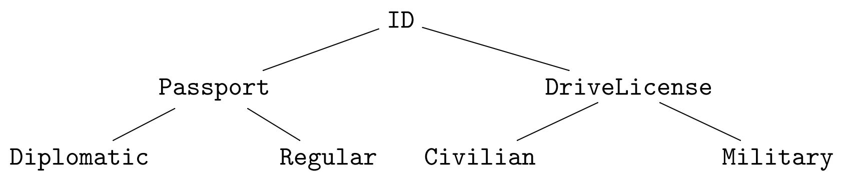

FIGURE 1.9: An example of certificate abstractions

within the domain of certificates. Figure 1.9 illustrates an example of certificate abstractions. The use of abstractions in the policy specification provides a compact and easy way to refer to complex concepts. For instance, the specification of abstraction `Passport` in a policy states that any kind of passport (diplomatic or regular) can be accepted. Abstractions can be formally modeled through a hierarchy $\mathcal{H}$=($\mathcal{C}$,$\prec$), with $\mathcal{C}$ a set of certificate abstractions and $\prec$ a partial order relationship over $\mathcal{C}$. Given two certificate abstractions c_i and c_j, $c_i \prec c_j$ if c_j is an abstraction of c_i.

An important concept captured by several credential-based access control models is *delegation*. The delegation consists in the ability of an authority to produce credentials on behalf of the delegator. Delegation increases flexibility and permits the inexpensive creation of credentials, particularly in an open environment. A *delegation certificate* issued by an authority states that it trusts another authority for issuing certificates that include specific attributes. An authority can delegate other authorities only on given attributes (e.g., a hospital can issue a certificate delegating physicians to certify specific properties of patients) or can give unrestricted delegation to other authorities.

Low-level issues related to the certificate creation, retrieval, validation, and revocation are all usually assumed to be managed by an underlying implementation of the certificate management system (e.g., [38, 41, 44]) and are therefore outside the scope of this chapter.

Policy. A policy defines the rules (authorizations) regulating access to resources. Such authorizations model access restrictions based on generic properties associated with subjects and objects. In general, these restrictions can refer to attributes within specific credentials or can refer to certificate abstractions. In this latter case, a restriction involving a certificate abstraction applies also to its specialized abstractions/credentials. For instance, with respect to the certificate abstractions in Figure 1.9, `ID.dob` represents attribute `dob` certified by a credential of type `ID`. From an analysis of the current attribute and credential-based access control policies it is easy to see that at an abstract level the main elements of an authorization that are common to many proposals are the following.

- *Subject expression.* A subject expression identifies a set of subjects having specific attributes. A subject expression can be seen as a boolean formula of basic conditions defined on attributes. Attributes appearing in basic conditions must be certified or can be declared by a user. For instance, expression `ID.dob`>01-05-1971 denotes all users with a certificate of type `ID` certifying that the date of birth of the users is after January 05, 1971.

- *Object expression.* An object expression identifies the resources/services to be protected. Like for subject expressions, also an object expression can be seen as a boolean formula of basic conditions defined on the metadata associated with resources/services. For instance, assume that `producer` is a metadata attribute associated with objects. Then, expression `producer`="EU" denotes all objects made in an European country.

- *Action.* An action denotes the operation (or group thereof) to which the authorization refers.

Different languages have been developed for the specification of policies, and each of them supports different features. In the remainder of this section, we first describe the policy communication problem (Section 1.6.2), which is a specific problem of the attribute-based access control systems, and then we present some policy languages (Section 1.6.3). In particular, we present logic-based languages, which are expressive but turn out to be not applicable in practice, and then XACML-based languages, which are easy to use and consistent with consolidated technology.

1.6.2 Policy communication

A peculiarity of access control systems based on attributes is that the server offering resources/services evaluates the policies without a complete knowledge of users and their properties. The server has then to communicate to users the policies that they should satisfy to have their requests possibly permitted. For instance, consider a service accessible to all people older than 18 and working in an European country. In this case, the server has to communicate to the user that she has to provide her date of birth and the place where she works before accessing the service. This policy communication problem has been under the attention of the research and development communities for more than a decade and several solutions have been proposed, each addressing different issues (e.g., [29, 38]). A simple policy communication strategy consists in giving the user a list with all the possible sets of certificate that would allow the access. This solution is not always applicable due to the large number of possible alternatives (e.g., with compound credential requests such as "a passport and one membership certificate from a federated association", there may be a combinatorial explosion of alternatives). Automated trust negotiation strategies have been therefore proposed (e.g., [38, 55, 56, 59, 60, 61, 62]), which are based on the assumption that parties may be unknown a-priori and a multi-step trust negotiation process is necessary for communicating policies and for releasing certificates, which are both considered sensitive. The goal of a trust negotiation process is to gradually establish trust among parties by disclosing credentials and requests for credentials. In successful trust negotiations, credentials eventually are exchanged so that the policy regulating access to the required resource/service is satisfied. In [55] two different strategies are described: *eager* and *parsimonious* credential release strategies. Parties applying the first strategy communicate all their credentials if the release policy for them is satisfied, without waiting for the credentials to be requested. Parsimonious parties only release credentials upon explicit request by the server (avoiding unnecessary releases). In [60] the *PRUdent NEgotiation Strategy* (PRUNES) strategy ensures that a user communicates her credentials to the server only if the access will be granted and the set of certificates communicated to the server is the minimal necessary for granting it. Each party defines a set of credential policies that regulates how and under what conditions the party releases its credentials. The negotiation consists of a series of requests for credentials and counter-requests on the basis of the parties' credential policies. In [61] the authors present a family of trust negotiation strategies, called *Disclosure Tree Strategy* (DTS) family, and show that if two parties use different strategies from the DTS family, they are able to establish a negotiation process. In [59] the authors present a *Unified Schema for Resource Protection* (UniPro) for protecting resources and policies in trust negotiation. UniPro gives (opaque) names to policies and allows any named policy P_1 to have its own policy P_2 regulating to what parties policy P_1 can be disclosed. TrustBuilder [62] is a prototype developed to incorporate trust negotiation into standard network technologies. Traust [38] is a third-party authorization service that is based on the TrustBuilder framework for trust negotiation. The Traust service provides a negotiation-based mechanism that allows qualified users to obtain the credentials necessary to access resources provided by the involved server. In [56] the authors introduce a formal framework for automated trust negotiation, and formally define the concept of correct enforcement of policies during a trust negotiation process. In [6] the authors present an expressive and flexible approach for enabling servers to specify, when defining their access control policies, if and how the policy should be communicated to the client. Intuitively, an access control policy is represented through its expression tree and each node of the tree is associated with a disclosure policy (modeled with three colors, namely green, yellow, or red), ensuring a fine-grained support and providing expressiveness and flexibility in establishing disclosure regulations. The disclosure policies state whether an element in the policy can be released as it is or whether it has to be obfuscated. The authors also illustrate how to determine the user's view on the policy (i.e., the policy to be communicated to the user requesting access) according to the specified disclosure policies.

1.6.3 Languages for access control

Languages for access control aim to support the expression and the enforcement of policies [52]. Several access control languages have been developed, and most of them rely on concepts and techniques from logic and logic programming (e.g., [13, 27, 35, 36, 39, 40, 42]). Logic languages are particularly attractive due to their clean and unambiguous semantics, suitable for implementation validation, as well as formal policy verification. Logic languages can be expressive enough to formulate all the policies introduced in the literature, and their declarative nature yields a good compromise between expressiveness and simplicity. Nevertheless, many logic-based proposals, while appealing for their expressiveness, are not applicable in practice, where simplicity, efficiency, and consistency with consolidated technology are crucial. Effectively tackling these issues is probably the main motivation for the success of the eXtensible Access Control Markup Language (XACML) [50]. XACML is an OASIS standard that proposes an XML-based language for specifying and exchanging access control policies over the Web. The language can support the most common security policy representation mechanisms and has already found significant support by many players. Moreover, it includes standard extension points for the definition of new functions, data types, and policy combination methods, which provide a great potential for the management of access control requirements in emerging and future scenarios.

In the following, we survey some logic-based access control languages, and then describe the XACML language along with some extensions aiming at including the possibility of using credentials in the XACML policy specification.

1.6.3.1 Logic-based languages

The first work investigating logic languages for the specification of authorizations is the work by Woo and Lam [58]. They shown how flexibility and extensibility in access specifications can be achieved by abstracting from the low level authorization triples and adopting a high level authorization language. Their language is a many-sorted first-order language with a rule construct, useful to express authorization derivations and therefore model authorization implications and default decisions (e.g., closed or open policy). Their work has been subsequently refined by several authors (e.g., [35]).

Several logic-based languages have been developed for formulating, for example, dynamic policies, inheritance and overriding, policy composition, and credential-based authorizations (e.g., [9, 11, 12, 35]). In particular, specific solutions have addressed the problem of constraining the validity of authorizations through periodic expressions and appropriate temporal operators. For instance, the proposal in [9] shows a temporal authorization model that supports periodic access authorizations and periodic rules, and allows the derivation of new authorizations based on the presence or absence of other authorizations in specific periods of time. Other logic-based access control languages support inheritance mechanisms and conflict resolution policies. Jajodia et al. [35] present a proposal for a logic-based language that allows the representation of different policies and protection requirements, while at the same time providing understandable specifications, clear semantics (guaranteeing therefore the behavior of the specifications), and bearable data complexity. Authorizations are specified in terms of a locally stratified rule base logic. Such a solution allows the representation of different propagation policies (i.e., policies that specify how to obtain derived authorizations from the explicit authorizations), conflict resolution policies, and decision policies that a security system officer might want to use.

The fact that in open environments there is the need for combining access control restrictions independently stated by different parties motivates the development of solutions specifically targeted to the composition of policies (e.g., [11, 54]). These solutions typically do not make any assumption on the language adopted for specifying the given policies and define a set of policy operators used for combining different policies. In particular, in [12] a policy is defined as a set of triples of the form (s,o,a), where s is a constant in (or a variable over) the set of subjects $\mathcal{S}$, o is a constant in (or a variable over) the set of objects $\mathcal{O}$, and a is a constant in (or a variable over) the set of actions $\mathcal{A}$. Here, complex policies can then be obtained by combining policies via specific algebra operators.

Logic-based approaches have been also used for specifying, reasoning about, and communicating protection requirements. In particular, several proposals have addressed the problem of defining and enforcing credential-based authorization policies and trust management (e.g., [12, 34, 43, 48, 62]). In particular, in [12] the authors present a framework that includes an access control model, a language for expressing access and release policies, and a policy-filtering mechanism to identify the relevant policies for a negotiation. Access regulations are specified by logical rules, where some predicates are explicitly identified. The system is composed of two entities: the *client* that requests access, and the *server* that exposes a set of services. Abstractions can be defined on services, grouping them in sets, called *classes*. Server and client interact via a *negotiation process*, defined as the set of messages exchanges between them. Clients and servers have a *portfolio*, which is a collection of credentials (certified statements) and declarations (unsigned statements). Credentials are modeled as *credential expressions* of the form *credential_name(attribute_list)*, where *credential_name* is the credential name and *attribute_list* is a possibly empty list of elements of the form *attribute_name=value_term*, where *value_term* is either a ground value or a variable. The proposed framework allows a client to communicate the minimal set of certificates to a server, and the server to release the minimal set of conditions required for granting access. For this purpose, the server defines a set of *service accessibility rules*, representing the necessary and sufficient conditions for granting access to a resource. More precisely, this proposal distinguishes two kinds of service accessibility rules: *prerequisites* and *requisites*. Prerequisites are conditions that must be satisfied for a service request to be taken into consideration (they do not guarantee that it will be granted); requisites are conditions that allow the service request to be successfully granted. The basic motivation for this separation is to avoid unnecessary disclosure of information from both parties. Therefore, the server will not disclose a requisite rule until after the client satisfies a corresponding prerequisite rule. Also, both clients and servers can specify a set of *portfolio disclosure rules*, used to define the conditions that govern the release of credentials and declarations.

The rules both in the service accessibility and portfolio disclosure sets are defined through a logic language that includes a set of predicates whose meaning is expressed on the basis of the current *state*. The state indicates the parties' characteristics and the status of the current negotiation process, that is, the certificates already exchanged, the requests made by the two parties, and so on. Predicates evaluate both information stored at the site (persistent state) and acquired during the negotiation (negotiation state). Information related to a specific negotiation is deleted when the negotiation terminates. In contrast, persistent state includes information that spans different negotiations, such as user profiles maintained at Web sites.

Since there may exist different policy combinations that may bring the access request to satisfaction, the communication of credentials and/or declarations could be an expensive task. To overcome this issue, the *abbreviation* predicates are used to abbreviate requests. Besides the necessity of abbreviations, it is also necessary for the server, before releasing rules to the client, to evaluate state predicates that involve private information. For instance, the client is not expected to be asked many times the same information during the same session and if the server has to evaluate if the client is considered not trusted, it cannot communicate this request to the client itself.

Communication of requisites to be satisfied by the requester is then based on a filtering and renaming process applied on the server's policy, which exploits partial evaluation techniques in logic programs [12, 47]. Access is then granted whenever a user satisfies the requirements specified by the filtering rules calculated by means of the original policy and the already released information.

1.6.3.2 XML-based languages

With the increasing number of applications that either use XML as their data model, or export relational data as XML data, it becomes critical to investigate the problem of access control for XML. To this purpose, many XML-based access control languages have been proposed (e.g., [14, 20, 37, 50]). As already mentioned, the eXtensible Access Control Markup Language (XACML) is the most relevant XML-based access control language. XACML version 1.0 [49] has been an OASIS standard since 2003. Improvements have been made to the language and incorporated in version 3.0 [50].

XACML supports the definition of policies based on attributes associated with subjects and resources other than their identities. The attributes are assumed to be known during the evaluation time and stored in the XACML evaluation context, or presented by the requester together with the request. While XACML acknowledges that properties can be presented by means of certificates, and in fact, it has been designed to be integrated with the Security Assertion Markup Language (SAML) [2] for exchanging various types of security assertions and for providing protocol mechanisms, it does not provide a real support for expressing and reasoning about digital certificates in the specification of the authorization policies. Intuitively, XACML supports attribute-based access control but does not really support credential-based access control (see Section 1.6.3.3). XACML also supports policies independently specified by multiple authorities on the same resources. When an access request on that resource is submitted, the system has to take into consideration all these policies and their outcomes are combined according to a combining algorithm. Policies defined by different parties may be enforced at different enforcement points. XACML provides a method for specifying some actions, called *obligations*, that must be fulfill in conjunction with the policy enforcement.

Figure 1.10 illustrates the XACML data-flow that consists of the following steps.

- The requestor sends an access request to the *Policy Evaluation Point* (PEP) module, which has to enforce the access decision taken by the decision point.

- The PEP module sends the access request to the *Context Handler* that translates the original request in a canonical format, called *XACML request context*, by inquiring the *Policy Information Point* (PIP) module. PIP provides the values of attributes about the *subject*, *resource*, and *action* (the function to be performed). To this purpose, PIP interacts with the *Subjects*, *Resource*, and *Environment* modules. The *Environment*

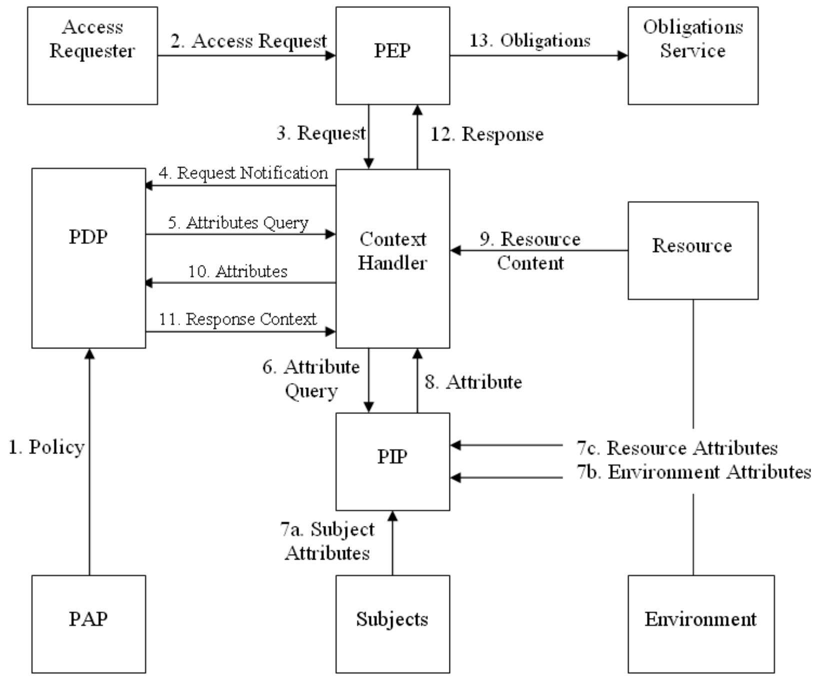

FIGURE 1.10: XACML overview [50]

module provides a set of attributes that are relevant to take an authorization decision and are independent of a particular *subject*, *resource*, and *action*.

- The Context Handler sends the XACML request to the *Policy Decision Point* (PDP). The PDP identifies the applicable policies by means of the *Policy Administration Point* (PAP) module and retrieves the required attributes and, possibly, the resource from the Context Handler.
- The PDP then evaluates the policies and returns the *XACML response context* to the Context Handler. The context handler translates the XACML response context to the native format of the PEP and returns it to the PEP together with an optional set of obligations.
- The PEP fulfills the obligations and, if the access is permitted, it performs the access. Otherwise, the PEP denies access.

The main concepts of interest in the XACML policy language are *rule*, *policy*, and *policy set*. Each XACML policy has a root element that can be either a `Policy` or a `PolicySet`. A `PolicySet` is a collection of `Policy` or `PolicySet`. A XACML policy consists of a *target*, a set of *rules*, an optional set of *obligations*, an optional set of *advices*, and a *rule combining algorithm*. We now describe these components more in details.

- *Target.* It consists of a simplified set of conditions for the *subject*, *resource*, and *action* that must be satisfied for a policy to be applicable to a given request. Note that the definition of the subjects, resources, and actions in a target are based on attributes. For instance, a physician at an hospital may have the attribute of being a researcher, a specialist in some field, or many other job roles. According to these attributes, the physician can be able to perform different functions within the hospital. If all the conditions of a `Target` are satisfied, its associated `Policy` (or `Policyset`) applies to the request. If a policy applies to all entities of a given type, that is, all subjects, actions, or resources, an empty element, named `AnySubject`, `AnyAction`, `AnyResource`, respectively, is used.

- *Rule.* The components of a rule are a *target*, an *effect*, a *condition*, *obligation expressions*, and *advice expressions*. The target defines the set of resources, subjects, and actions to which the rule applies. The effect of the rule can be `permit` or `deny`. The condition represents a boolean expression that may further refine the applicability of the rule. Note that the `target` element is an optional element: a rule with no target applies to all possible requests. Obligation and advice expressions are evaluated and they can be returned to the PEP in the response context. Note that while obligations cannot be ignored, advices can be safely ignored by the PEP.

- *Obligation.* An obligation is an operation that has to be performed in conjunction with the enforcement of an authorization decision. For instance, an obligation can state that all accesses on medical data have to be logged. Obligations are returned by the PDP to the PEP along with the response. Note that, only policies that are evaluated and have returned a response of `permit` or `deny` can return obligations. This means that if a policy evaluates to `indeterminate` or `not applicable`, the associated obligations are not returned to the PEP.

- *Advice.* An advice is a supplementary piece of information that is returned to the PEP with the decision of the PDP.

- *Rule combining algorithm.* Each policy is associated with a rule combining algorithm used for reconciling the decisions each rule make. The final decision value, called *authorization decision*, inserted in the XACML context by the PDP is the value of the policy as defined by the rule combining algorithm. XACML defines eight different combining algorithms: *deny overrides*, *ordered-deny-overrides*, *permit overrides*, *ordered-permit-overrides*, *deny-unless-permit*, *permit-unless-deny*, *first applicable*, and *only-one-applicable* (see [50] for more details about the meaning of these combining algorithms). If no rule applies, the result is `not applicable`. If only one policy applies, the result coincides with the result of evaluating that rule. According to the selected combining algorithm, the authorization decision returned to the PEP can be `permit`, `deny`, `not applicable` (when no applicable policies or rules could be found), or `indeterminate` (when some errors occurred during the access control process). In particular, XACML 3.0 defines an extended set of `indeterminate` values, which includes: `indeterminate{D}` when a policy (rule) could have evaluated to `deny` but not `permit`; `indeterminate{P}` when a policy (rule) could have evaluated to `permit` but not `deny`; `indeterminate{DP}` when a policy (rule) could have evaluated to `deny` or `permit`.

XACML also defines a standard format for expressing requests and responses. The original request submitted by the PEP is translated through the Context Handler in a canonical form, and then forwarded to the PDP to be evaluated. For instance, an application can provide a SAML [2] message that includes a set of attributes characterizing the subject making the access request. This message has to be converted to the XACML canonical form and, analogously, the XACML decision has then to be converted to the SAML format. A request contains attributes for the subject, resource, action, and, optionally, for the environment. Each request includes exactly one set of attributes for the resource and action and at most one set of environment attributes. There may be multiple sets of subject attributes each of which is identified by a category URI. A response element contains one or more results corresponding to an evaluation. Each result contains six elements: `Decision` specifies the authorization decision (i.e., `permit`, `deny`, `indeterminate`, `not applicable`); `Status` indicates if some error occurred during the evaluation process; `Obligations` states the obligations that the PEP must fulfill; `AssociatedAdvice` is optional and reports a list of advices that provide additional information to the PEP; `Attributes` is optional and contains a list of attributes that were part of the request; `PolicyIdentifierList` is optional and corresponds to a list of policy or policy set identifiers that have been applicable to a request.

1.6.3.3 Expanding XACML with credentials

Although designed to be integrated with the Security Assertion Markup Language (SAML) [2] for exchanging security assertions and providing protocol mechanisms, XACML lacks a real support for considering, reasoning, and expressing conditions on certified properties. Recent proposals have tried to overcome this and other limitations that make XACML not yet suitable for open Web-based systems. In particular, the novel features that should be supported by a practical access control language can be summarized as follows [4].

- *Certified information.* The attributes used in the XACML policies are assumed to be known during the evaluation time and stored within the XACML context or presented by the user together with the access request. To represent and manage credentials in XACML it is then necessary to express the fact that some attributes should be presented through given certificates, possibly imposing conditions on the value of these attributes and on the certificates themselves.

- *Abstractions.* Intuitively, abstractions represent a shorthand by which a single concept is introduced to represent a more complex one (e.g., a set, a disjunction, or a conjunction of concepts). For instance, `ID` (abstraction head) can be defined as an abstraction for any element in set {`Passport`, `DriverLicense`} of credentials (abstraction tail). A policy specifying that an access requester must provide an `ID` can then be satisfied by presenting any of the two credentials above.

- *Recursive conditions.* The support for recursive reasoning allows the specification of policies based on chains of credentials and of conditions on data with a recursive structure.

- *Dialog.* The introduction of dialog between the involved parties has the advantages that the server can communicate which information is needed to evaluate an access control policy, and a user can release only the necessary credentials instead of releasing the whole set. A further advantage is that it permits to tackle the issue of the privacy trade-off between providing the whole set of credentials (on the access requester side) and disclosing the whole access control policy (on the server side). The proposal in [4] obtains this result by attaching a disclosure attribute to each condition in an access control policy. This attribute indicates what type of disclosure policy is associated with the condition, and it is enforced by hiding from the access requester the information that cannot be released according to such a disclosure policy. The more (less) of an access control policy is disclosed, the smaller (bigger) is the quantity of information in terms of released credentials that will have to be provided by the user.

The proposal in [4] illustrates how certified information, abstractions, recursive reasoning, and dialog management can be deployed in XACML. In particular, it shows that the integration of XACML with XQuery can be adopted for supporting abstractions and recursive conditions. Credentials and dialog management require a minimal change in the XACML language that consists in the addition of appropriate elements and attributes. Other proposals (e.g., [17, 18, 32, 46]) provide XACML extensions to support trust negotiation.

1.7 Conclusions

In this chapter we introduced the most important concepts related to access control. We first described the discretionary, mandatory, and role-based access control policies, and then we illustrated recent proposals in the area of access control models and languages. In particular, we described novel approaches based on digital certificates, which are more suitable for open scenarios where servers offering services and users requesting such services do not know each other. We also provided an overview of logic-based and XML-based access control languages.

1.8 Defining Terms

Access control: A process that controls every request to a system and determining, based on specified authorizations, whether the request should be granted or denied.

Access matrix: A matrix representing the set of authorizations defined at a given time in the system.

ACL: Access Control List.

Administrative policy: A policy regulating who can modify the allowed accesses.

Authorization: The right granted to a user to exercise an action (e.g., read, write, create, delete, and execute) on certain objects.

Certificate: A statement certified by an authority trusted for making such a statement.

DAC: Discretionary Access Control.

MAC: Mandatory Access Control.

Obligation: An action that must be performed sometime to allow the execution of a given action.

PEP: Policy Evaluation Point.

PDP: Policy Decision Point.

PIP: Policy Information Point.

RBAC: Role Based Access Control.

Role: A job function within an organization that describes the authority and responsibility related to the execution of an activity.

Security mechanism: Low-level software and/or hardware functions that implement security policies.

Security policy: High-level guidelines establishing rules that regulate access to resources.

XACML: eXtensible Access Control Markup Language.

Bibliography

[1] G. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):207–226, November 2000.

[2] A. Anderson and H. Lockhart. *SAML 2.0 profile of XACML*. OASIS, September 2004. http://docs.oasis-open.org/xacml/access_control-xacml-2.0-saml_profile-spec-cd-01.pdf.

[3] *ANSI/INCITS 359 American National Standard for Information Technology – Role Based Access Control*, 2004.

[4] C. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, and M. Verdicchio. Expressive and deployable access control in open web service applications. *IEEE Transactions on Service Computing (TSC)*, 4(2):96–109, April-June 2011.

[5] C.A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. A privacy-aware access control system. *Journal of Computer Security (JCS)*, 16(4):369–392, 2008.

[6] C.A. Ardagna, S. De Capitani di Vimercati, S. Foresti, G. Neven, S. Paraboschi, F.-S. Preiss, P. Samarati, and M. Verdicchio. Fine-grained disclosure of access policies. In *Proc. of the 12th International Conference on Information and Communications Security (ICICS 2010)*, Barcelona, Spain, December 2010.

[7] C.A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, and P. Samarati. Minimising disclosure of client information in credential-based interactions. *International Journal of Information Privacy, Security and Integrity (IJIPSI)*, 1(2/3):205–233, 2012.

[8] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving fingercode authentication. In *Proc. of 12th ACM Workshop on Multimedia and Security (MM&Sec 2010)*, Rome, Italy, September 2010.

[9] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems (TODS)*, 23(3):231–285, September 1998.

[10] E. Bertino, P. Samarati, and S. Jajodia. Authorizations in relational database management systems. In *Proc. of the First ACM Conference on Computer and Communications Security (CCS 93)*, pages 130–139, Fairfax, VA, November 1993.

[11] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, February 2002.

[12] P. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security (JCS)*, 10(3):241–271, 2002.

[13] P. Bonatti and P. Samarati. Logics for authorizations and security. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*. Springer-Verlag, 2003.

[14] D. Box et al. *Web services policy framework (WS-Policy) version 1.1.*, May 2003. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policy.asp.

[15] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proc. of International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 2001)*, Innsbruck, Austria, May 2001.

[16] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison Wesley, 1994.

[17] D.W. Chadwick, S. Otenko, and T.A. Nguyen. Adding support to XACML for dynamic delegation of authority in multiple domains. In *Proc. of 10th Open Conference on Communications and Multimedia Security (CMS 2006)*, Heraklion, Crete, Greece, October 2006.

[18] V.S.Y. Cheng, P.C.K. Hung, and D.K.W. Chiu. Enabling web services policy negotiation with privacy preserved using XACML. In *Proc. of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007)*, Hawaii, USA, January 2007.

[19] S. Cimato, M. Gamassi, V. Piuri, and F. Scotti. Privacy-aware biometrics: Design and implementation of a multimodal verification system. In *Proc. of the Annual Computer Security Applications Conference (ACSAC 2008)*, Anaheim, CA, USA, December 2008.

[20] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):169–202, May 2002.

[21] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. A data outsourcing architecture combining cryptography and access control. In *Proc. of the 1st Computer Security Architecture Workshop (CSAW 2007)*, Fairfax, VA, USA, November 2007.

[22] S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Access control: Principles and solutions. *Software – Practice and Experience*, 33(5):397–421, April 2003.

[23] S. De Capitani di Vimercati and P. Samarati. Access control in federated systems. In *Proc. of the ACM SIGSAC New Security Paradigms Workshop (NSPW 96)*, Lake Arrowhead, CA, USA, September 1996.

[24] S. De Capitani di Vimercati and P. Samarati. Authorization specification and enforcement in federated database systems. *Journal of Computer Security (JCS)*, 5(2):155–188, 1997.

[25] S. De Capitani di Vimercati, P. Samarati, and S. Jajodia. Hardware and software data security. In D. Kaeli and Z. Navabi, editors, *EOLSS The Encyclopedia of Life Support Systems*. EOLSS Publishers, 2001.

[26] S. De Capitani di Vimercati, P. Samarati, and S. Jajodia. Policies, models, and languages for access control. In *Proc. of the Workshop on Databases in Networked Information Systems (DNIS 2005)*, Aizu-Wakamatsu, Japan, March 2005.

[27] J. DeTreville. Binder, a logic-based security language. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2002.

[28] D.F. Ferraiolo and R. Sandhu. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, August 2001.

[29] K. Frikken, M. Atallah, and J. Li. Attribute-based access control with hidden policies and hidden credentials. *IEEE Transactions on Computer (TC)*, 55(10):1259–1270, October 2006.

[30] L. Fuchs, G. Pernul, and R. Sandhu. Roles in information security - A survey and classification of the research area. *Computers and Security*, 30(8):748–769, November 2011.

[31] M. Gamassi, V. Piuri, D. Sana, and F. Scotti. Robust fingerprint detection for access control. In *Proc. of the Workshop RoboCare (RoboCare 2005)*, Rome, Italy, May 2005.

[32] D.A. Haidar, N. Cuppens, F. Cuppens, and H. Debar. XeNA: an access negotiation framework using XACML. *Annales des télécommunications - Annals of telecommunications*, 64(1/2):155–169, January 2009.

[33] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the SCM*, 19(8):461–471, August 1976.

[34] K. Irwin and T. Yu. Preventing attribute information leakage in automated trust negotiation. In *Proc. of the 12th ACM Conference on Computer and Communications Security (CCS 2005)*, Alexandria, VA, USA, November 2005.

[35] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, 26(2):214–260, June 2001.

[36] T. Jim. Sd3: A trust management system with certified evaluation. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2001.

[37] M. Kudoh, Y. Hirayama, S. Hada, and A. Vollschwitz. Access control specification based on policy evaluation and enforcement model and specification language. In *Proc. of Symposium on Cryptography and Information Security, (SCIS 2000)*, Okinawa, Japan, January 2000.

[38] A.J. Lee, M. Winslett, J. Basney, and V. Welch. The Traust authorization service. *ACM Transactions on Information and System Security (TISSEC)*, 11(1):1–3, February 2008.

[39] N. Li, B.N. Grosof, and Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):128–171, February 2003.

[40] N. Li and J.C. Mitchell. Datalog with constraints: A foundation for trust-management languages. In *Proc. of the 5th International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, New Orleans, LA, USA, January 2003.

[41] N. Li and J.C. Mitchell. Understanding SPKI/SDSI using first-order logic. *International Journal of Information Security*, 5(1):48–64, January 2006.

[42] N. Li, J.C. Mitchell, and W.H. Winsborough. Design of a role-based trust-management framework. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2002.

[43] N. Li, J.C. Mitchell, and W.H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM*, 52(3):474–514, May 2005.

[44] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security (JCS)*, 11(1):35–86, 2003.

[45] T. Lunt. Access control policies: Some unanswered questions. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW 88)*, Franconia, NH, USA, June 1988.

[46] U.M. Mbanaso, G.S. Cooper, D.W. Chadwick, and S. Proctor. Privacy preserving trust authorization framework using XACML. In *Proc. of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks (WOWMOM 2006)*, Niagara-Falls, NY, USA, June 2006.

[47] M. Minoux. Ltur: A simplified linear-time unit resolution algorithm for horn formulae and computer implementation. *Information Processing Letter (IPL)*, 29(1):1–12, 1988.

[48] J. Ni, N. Li, and W.H. Winsborough. Automated trust negotiation using cryptographic credentials. In *Proc. of the 12th ACM Conference on Computer and Communications Security (CCS 2005)*, Alexandria, VA, USA, November 2005.

[49] OASIS. *eXtensible Access Control Markup Language (XACML) Version 1.0*, 2003. http://www.oasis-open.org/committees/xacml.

[50] OASIS. *eXtensible Access Control Markup Language (XACML) Version 3.0*, 2010. http://www.oasis-open.org/committees/xacml.

[51] J. Park and R. Sandhu. The ucon$_{ABC}$ usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174, February 2004.

[52] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*. Springer-Verlag, 2001.

[53] R.S. Sandhu. On five definitions of data integrity. In *Proc. of the IFIP WG11.3 Working Conference on Database Security VII (DBSec 93)*, Lake Guntersville, AL, USA, September 1993.

[54] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):286–325, May 2003.

[55] W. Winsborough, K.E. Seamons, and V. Jones. Automated trust negotiation. In *Proc. of the DARPA Information Survivability Conference & Exposition (DISCEX 2000)*, Hilton Head Island, SC, USA, January 2000.

[56] W.H. Winsborough and N. Li. Safety in automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC)*, 9(3):352–390, 2006.

[57] M. Winslett, N. Ching, V. Jones, and I. Slepchin. Using digital credentials on the World-Wide Web. *Journal of Computer Security (JCS)*, 5(2):255–267, 1997.

[58] T.Y.C. Woo and S.S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security (JCS)*, 2(2,3):107–136, 1993.

[59] T. Yu and M. Winslett. A unified scheme for resource protection in automated trust negotiation. In *Proc. of the IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, May 2003.

[60] T. Yu, M. Winslett, and K.E. Seamons. Prunes: An efficient and complete strategy for automated trust negotiation over the internet. In *Proc. of the 7th ACM Conference on Computer and Communications Security (CCS 2000)*, Athens, Greece, November 2000.

[61] T. Yu, M. Winslett, and K.E. Seamons. Interoperable strategies in automated trust negotiation. In *Proc. of the 8th ACM Conference on Computer and Communications Security (CCS 2001)*, Philadelphia, PA, USA, November 2001.

[62] T. Yu, M. Winslett, and K.E. Seamons. Supporting structured credentials and sensitive policies trough interoperable strategies for automated trust. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):1–42, February 2003.