Handbook of *Database Security*

Applications and Trends

Handbook of *Database Security*

Applications and Trends

edited by

Michael Gertz
University of California at Davis
USA

Sushil Jajodia
George Mason University
USA

Springer

Michael Gertz
University of California at Davis
Dept. of Computer Science
One Shields Avenue
Davis, CA 95616-8562
gertz@cs.ucdavis.edu

Sushil Jajodia
George Mason University
Center for Secure Information Systems
Research I, Suite 417
Fairfax VA 22030-4444
jajodia@gmu.edu

Library of Congress Control Number: 2007934795

ISBN-13: 978-0-387-48532-4
e-ISBN-13: 978-0-387-48533-1

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

Preface

Motivation for the book

Database security has been gaining a great deal of importance as industry, military, and government organizations have increasingly adopted Internet-based technologies on a large-scale, because of convenience, ease of use, and the ability to take advantage of rapid advances in the commercial market. Along with the traditional security aspects of data integrity and availability, there is an increasing interest in research and development in data privacy. This is because today's often mission-critical databases no longer contain only data used for day-to-day processing by organization; as new applications are being added, it is possible for organizations to collect and store vast amounts of data quickly and efficiently and to make the data readily accessible to the public, typically through Web-based applications. Unfortunately, if security threats related to the integrity, availability, and privacy of the data are not properly resolved, databases remain vulnerable to malicious attacks and accidental misuse. Such incidents, in turn, may translate into financial losses or losses whose values are obviously high but difficult to quantify, e.g., the loss of the public's trust in the data management infrastructure and services offered by an organization.

In assembling this handbook, we have had a twofold objective: first, to provide a comprehensive summary of the results of research and development activities in various aspects of database security up to this point, and second, to point toward directions for future work in this important and fruitful field of research.

This handbook offers twenty three essays contributed by a selected group of prominent researchers. Given the dynamic nature of the field of database security, we have attempted to obtain a balance among various viewpoints by inviting multiple contributions on the same topic. We believe that this diversity provides a richness generally not available in one book. In some cases, authors have tried to reconcile their differences by contributing a single essay on a topic.

About the book

Essays in this handbook can be roughly divided into following eight areas:

Foundations of Access Control

- Recent Advances in Access Control by Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati
- Access Control Models for XML by Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, and Pierangela Samarati
- Access Control Policy Languages in XML by Naizhen Qi and Michiharu Kudo

Trust Management and Trust Negotiation

- Database Issues in Trust Management and Trust Negotiation by Dongyi Li, William Winsborough, Marianne Winslett, and Ragib Hasan

Secure Data Outsourcing

- Authenticated Index Structures for Outsourced Databases by Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin
- Towards Secure Data Outsourcing by Radu Sion
- Managing and Querying Encrypted Data by Bijit Hore, Sharad Mehrotra, and Hakan Hacıgümüş

Security in Advanced Database Systems and Applications

- Security in Data Warehouses and OLAP Systems by Lingyu Wang and Sushil Jajodia
- Security for Workflow Systems by Vijayalakshmi Atluri and Janice Warner
- Secure Semantic Web Services by Bhavani Thuraisingham
- Geospatial Database Security by Soon Ae Chun and Vijayalakshmi Atluri
- Security Re-engineering for Databases: Concepts and Techniques by Michael Gertz and Madhavi Gandhi

Database Watermarking

- Database Watermarking for Copyright Protection by Radu Sion
- Database Watermarking: A Systematic View by Yingjiu Li

Trustworthy Record Retention and Recovery

- Trustworthy Records Retention by Ragib Hasan, Marianne Winslett, Soumyadeb Mitra, Windsor Hsu, and Radu Sion
- Damage Quarantine and Recovery in Data Processing Systems by Peng Liu, Sushil Jajodia, and Meng Yu

Privacy

- Hippocratic Databases: Current Capabilities and Future Trends by Tyrone Grandison, Christopher Johnson, and Jerry Kiernan
- Privacy-Preserving Data Mining: A Survey by Charu C. Aggarwal and Philip S. Yu
- Privacy in Database Publishing: A Bayesian Perspective by Alin Deutsch
- Privacy Preserving Publication: Anonymization Frameworks and Principles by Yufei Tao

Privacy in Location-based Services

- Privacy Protection through Anonymity in Location-based Services by Claudio Bettini, Sergio Mascetti, and X. Sean Wang
- Privacy-enhanced Location-based Access Control by Claudio A. Ardagna, Marco Cremonini, Sabrina De Capitani di Vimercati, and Pierangela Samarati
- Efficiently Enforcing the Security and Privacy Policies in a Mobile Environment by Vijayalakshmi Atluri and Heechang Shin

Intended audience

This handbook is suitable as a reference for practitioners and researchers in industry and academia who are interested in the state-of-the-art in database security and privacy. Instructors may use this handbook as a text in a course for upper-level undergraduate or graduate students. Any graduate student who is interested in database security and privacy must definitely read this book.

Acknowledgements

We are extremely grateful to all those who contributed to this handbook. It is a pleasure to acknowledge the authors for their contributions. Special thanks go to Susan Lagerstrom-Fife, Senior Publishing Editor for Springer, and Sharon Palleschi, Editorial Assistant at Springer, whose enthusiasm and support for this project were most helpful.

Davis, California, and Fairfax, Virginia — *Michael Gertz*
September 2007 — *Sushil Jajodia*

Contents

List of Contributors

Charu C. Aggarwal
IBM T. J. Watson Research Center, Hawthorne, NY, e-mail: charu@us.ibm.com

Claudio A. Ardagna
Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Crema, Italy, e-mail: ardagna@dti.unimi.it

Vijayalakshmi Atluri
Rutgers University, Newark, NJ, e-mail: atluri@cimic.rutgers.edu

Claudio Bettini
DICo, University of Milan, Italy, e-mail: bettini@dico.unimi.it

Sabrina De Capitani di Vimercati
Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Crema, Italy, e-mail: decapita@dti.unimi.it

Soon Ae Chun
City University of New York, College of Staten Island, Staten Island, NY, e-mail: chun@mail.csi.cuny.edu

Marco Cremonini
Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Crema, Italy, e-mail: cremonini@dti.unimi.it

Alin Deutsch
Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA, e-mail: deutsch@cs.ucsd.edu

Sara Foresti
Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Crema, Italy, e-mail: foresti@dti.unimi.it

Madhavi Gandhi
Department of Mathematics and Computer Science, California State University, East Bay, CA, e-mail: madhavi.gandhi@eastbay.edu

Michael Gertz
Department of Computer Science, University of California at Davis, Davis, CA, e-mail: gertz@cs.ucdavis.edu

Tyrone Grandison
IBM Almaden Research Center, San Jose, CA, e-mail: tyroneg@us.ibm.com

Hakan Hacıgümüş
IBM Almaden Research Center, San Jose, CA, e-mail: hakanh@acm.org

Marios Hadjieftheriou
AT&T Labs Inc., e-mail: marioh@research.att.com

Ragib Hasan
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, e-mail: rhasan@cs.uiuc.edu

Bijit Hore
Donald Bren School of Computer Science, University of California, Irvine, CA, e-mail: bhore@ics.uci.edu

Windsor Hsu
Data Domain, Inc., e-mail: windsor.hsu@datadomain.com

Sushil Jajodia
Center for Secure Information Systems, George Mason University, Fairfax, VA, e-mail: jajodia@gmu.edu

Christopher Johnson
e-mail: chrisjohnson@alum.berkeley.edu

Jerry Kiernan
IBM Almaden Research Center, San Jose, CA, e-mail: jkiernan@us.ibm.com

George Kollios
Computer Science Department, Boston University, Boston, MA, e-mail: gkollios@cs.bu.edu

Michiharu Kudo
Tokyo Research Laboratory, IBM, Japan, e-mail: kudo@jp.ibm.com

Dongyi Li
Department of Computer Science, University of Texas at San Antonio, TX, e-mail: dli@cs.utsa.edu

Feifei Li
Department of Computer Science, Florida State University, FL, e-mail: lifeifei@cs.fsu.edu

Yingjiu Li
School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore, e-mail: yjli@smu.edu.sg

Peng Liu
Pennsylvania State University, PA, e-mail: pliu@ist.psu.edu

Sergio Mascetti
DICo, University of Milan, Italy, e-mail: mascetti@dico.unimi.it

Sharad Mehrotra
Donald Bren School of Computer Science, University of California, Irvine, CA, e-mail: sharad@ics.uci.edu

Soumyadeb Mitra
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, e-mail: mitra1@cs.uiuc.edu

Stefano Paraboschi
University of Bergamo, Dalmine, Italy, e-mail: parabosc@unibg.it

Naizhen Qi
Tokyo Research Laboratory, IBM, Japan, e-mail: naishin@jp.ibm.com

Leonid Reyzin
Computer Science Department, Boston University, Boston, MA, e-mail: reyzin@cs.bu.edu

Pierangela Samarati
Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Crema, Italy, e-mail: samarati@dti.unimi.it

Heechang Shin
Rutgers University, Newark, NJ, e-mail: hshin@cimic.rutgers.edu

Radu Sion
Network Security and Applied Cryptography Lab, Stony Brook University, NY, e-mail: sion@cs.stonybrook.edu

Yufei Tao
Department of Computer Science and Engineering, Chinese University of Hong Kong, Sha Tin, New Territories, Hong Kong, e-mail: taoyf@cse.cuhk.edu.hk

Bhavani Thuraisingham
University of Texas at Dallas, TX, e-mail: bhavani.thuraisingham@utdallas.edu

Lingyu Wang
Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC H3G 1M8, Canada, e-mail: wang@ciise.concordia.ca

X. Sean Wang
Department of Computer Science, University of Vermont, VT, e-mail: xywang@emba.uvm.edu

Janice Warner
Rutgers University, Newark, NJ, e-mail: janice@cimic.rutgers.edu

William Winsborough
Department of Computer Science, University of Texas at San Antonio, TX, e-mail: wwinsborough@acm.org

Marianne Winslett
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, e-mail: winslett@cs.uiuc.edu

Meng Yu
Western Illinois University, Macomb, IL, e-mail: m-yu2@wiu.edu

Philip S. Yu
IBM T. J. Watson Research Center, Hawthorne, NY, e-mail: psyu@us.ibm.com

1

Recent Advances in Access Control

S. De Capitani di Vimercati, S. Foresti, and P. Samarati

Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
26013 Crema, Italy
{decapita,foresti,samarati}@dti.unimi.it

Summary. Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied. Traditional access control models and languages result limiting for emerging scenarios, whose open and dynamic nature requires the development of new ways of enforcing access control. Access control is then evolving with the complex open environments that it supports, where the decision to grant an access may depend on the properties (attributes) of the requestor rather than her identity and where the access control restrictions to be enforced may come from different authorities. These issues pose several new challenges to the design and implementation of access control systems. In this chapter, we present the emerging trends in the access control field to address the new needs and desiderata of today's systems.

1 Introduction

Information plays an important role in any organization and its protection against unauthorized disclosure (*secrecy*) and unauthorized or improper modifications (*integrity*), while ensuring its availability to legitimate users (*no denials-of-service*) is becoming of paramount importance. An important service in guaranteeing information protection is the *access control* service. Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied. An access control system can be considered at three different abstractions of control: access control *policy*, access control *model*, and access control *mechanism*. A policy defines the high level rules used to verify whether an access request is to be granted or denied. A policy is then formalized through a *security model* and is enforced by an access control *mechanism*. The separation between policies and mechanisms has a number of advantages. First, it is possible to discuss protection requirements independently of their implementation. Second, it is possible to compare different access control policies as well as different mechanisms that enforce the same policy. Third, it is possible to design access control mechanisms able to enforce multiple policies.

In this way, a change in the access control policy does not require any changes in the mechanism. Also, the separation between model and mechanism makes it possible to formally prove security properties on the model; any mechanism that correctly enforces the model will then enjoy the same security properties proved for the model.

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. An access control system should be simple and expressive. It should be simple to make easy the management task of specifying and maintaining the security specifications. It should be expressive to make it possible to specify in a flexible way different protection requirements that may need to be imposed on different resources and data. Moreover, an access control system should include support for the following features.

- *Policy combination.* Since information may not be under the control of a single authority, access control policies information may take into consideration the protection requirements of the owner, but also the requirements of the collector and of other parties. These multiple authorities scenario should be supported from the administration point of view providing solutions for modular, large-scale, scalable policy composition and interaction.
- *Anonymity.* Many services do not need to know the real identity of a user. It is then necessary to make access control decisions dependent on the requester's *attributes*, which are usually proved by *digital certificates*.
- *Data outsourcing.* A recent trend in the information technology area is represented by data outsourcing, according to which companies shifted from fully local management to outsourcing the administration of their data by using externally service providers [1, 2, 3]. Here, an interesting research challenge consists in developing an efficient mechanism for implementing selective access to the remote data.

These features pose several new challenges to the design and implementation of access control systems. In this chapter, we present the emerging trends in the access control field to address the new needs and desiderata of today's systems. The remainder of the chapter is organized as follows. Section 2 briefly discusses some basic concepts about access control, showing the main characteristics of the discretionary, mandatory, and role-based access control policies along with their advantages and disadvantages. Section 3 introduces the problem of enforcing access control in open environments. After a brief overview of the issues that need to be addressed, we describe some proposals for trust negotiation and for regulating service access. Section 4 addresses the problem of combining access control policies that may be independently stated. We first describe the main features that a policy composition framework should have and then illustrate some current solutions. Section 5 presents the main approaches for enforcing selective access in an outsourced scenario. Finally, Sect. 6 concludes the chapter.

	Document1	Document2	Program1	Program2
Ann	read, write	read	execute	
Bob	read	read	read, execute	
Carol		read, write		read, execute
David			read, write, execute	read, write, execute

Fig. 1. An example of access matrix

2 Classical Access Control Models

Classical access control models can be grouped into three main classes: *discretionary access control* (DAC), which bases access decisions on users' identity; *mandatory access control* (MAC), which bases access decisions on mandated regulations defined by a central authority; and *role-based access control* (RBAC), which bases access decisions on the roles played by users in the models. We now briefly present the main characteristics of these classical access control models.

2.1 Discretionary Access Control

Discretionary access control is based on the identity of the user requesting access and on a set of rules, called *authorizations*, explicitly stating which user can perform which action on which resource. In the most basic form, an authorization is a triple (s, o, a), stating that user s can execute action a on object o. The first discretionary access control model proposed in the literature is the *access matrix model* [4, 5, 6]. Let S, O, and A be a set of subjects, objects, and actions, respectively. The access matrix model represents the set of authorizations through a $|\mathsf{S}| \times |\mathsf{O}|$ matrix $\mathcal{A}$. Each entry $\mathcal{A}[s, o]$ contains the list of actions that subject s can execute over object o. Figure 1 illustrates an example of access matrix where, for example, user `Ann` can `read` and `write` `Document1`.

The access matrix model can be implemented through different mechanisms. The straightforward solution exploiting a two-dimensional array is not viable, since $\mathcal{A}$ is usually sparse. The mechanisms typically adopted are:

- *Authorization table.* The non empty entries of $\mathcal{A}$ are stored in a table with three attributes: `user`, `action`, and `object`.
- *Access control list (ACL).* The access matrix is stored by column, that is, each object is associated with a list of subjects together with a set of actions they can perform on the object.
- *Capability.* The access matrix is stored by row, that is, each subject is associated with a list indicating, for each object, the set of actions the subject can perform on it.

Figure 2 depicts the authorization table, access control lists, and capability lists corresponding to the access matrix of Fig. 1.

User	Action	Object
Ann	read	Document1
Ann	write	Document1
Ann	read	Document2
Ann	execute	Program1
Bob	read	Document1
Bob	read	Document2
Bob	read	Program1
Bob	execute	Program1
Carol	read	Document2
Carol	write	Document2
Carol	execute	Program2
David	read	Program1
David	write	Program1
David	execute	Program1
David	read	Program2
David	write	Program2
David	execute	Program2

(a)

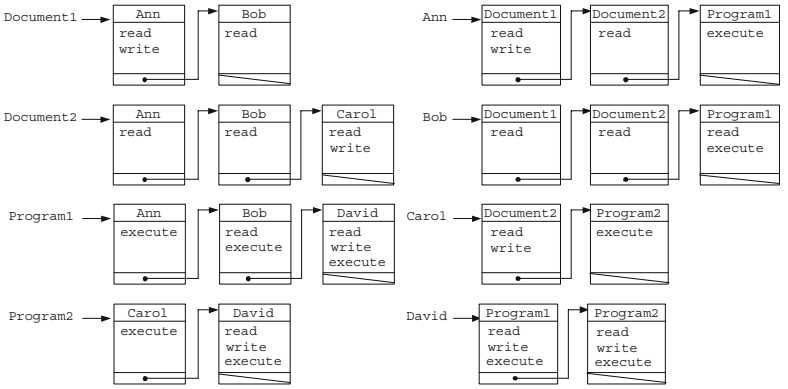

Fig. 2. Access matrix implementation mechanisms

From the access matrix model, discretionary access control systems have evolved and they include support for the following features.

- *Conditions.* To make authorization validity depend on the satisfaction of some specific constraints, today's access control systems typically support conditions associated with authorizations. [5]. For instance, conditions impose restrictions on the basis of: object content (content-dependent conditions), system predicates (system-dependent conditions), or accesses previously executed (history-dependent conditions).

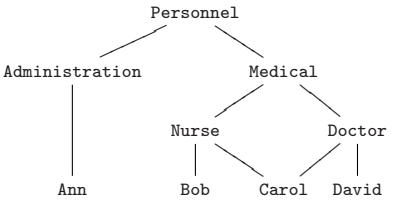

Fig. 3. An example of user-group hierarchy

- *Abstractions.* To simplify the authorization definition process, discretionary access control supports also *user groups* and *classes of objects*, which may also be hierarchically organized. Typically, authorizations specified on an abstraction propagate to all its members according to different *propagation policies* [7]. Figure 3 illustrates an example of user-group hierarchy. Here, for example, an authorization specified for the `Nurse` group applies also to `Bob` and `Carol`.
- *Exceptions.* The definition of abstractions naturally leads to the need of supporting exceptions in authorization definition. Suppose, for example, that all users belonging to a group but u can access resource r. If exceptions were not supported, it would be necessary to associate an authorization with each user in the group but u, therefore not exploiting the possibility of specifying the authorization of the group. This situation can be easily solved by supporting both *positive* and *negative* authorizations: the system would have a positive authorization for the group and a negative authorization for u.
  The introduction of both positive and negative authorizations brings to two problems: *inconsistency*, when conflicting authorizations are associated with the same element in a hierarchy; and *incompleteness*, when some accesses are neither authorized nor denied.
  Incompleteness is usually easily solved by assuming a *default policy*, open or closed (this latter being more common), where no authorization applies. In this case, an open policy approach allows the access, while the closed policy approach denies it.
  To solve the inconsistency problem, different *conflict resolution policies* have been proposed [7, 8], 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.* Conflicts remain unsolved.
  - *Most specific takes precedence.* An authorization associated with an element n overrides a contradicting authorization (i.e., an authorization with the same subject, object, and action but with a different sign) associated with an ancestor of n for all the descendants of n. For instance, consider the user-group hierarchy in Fig. 3 and the autho-

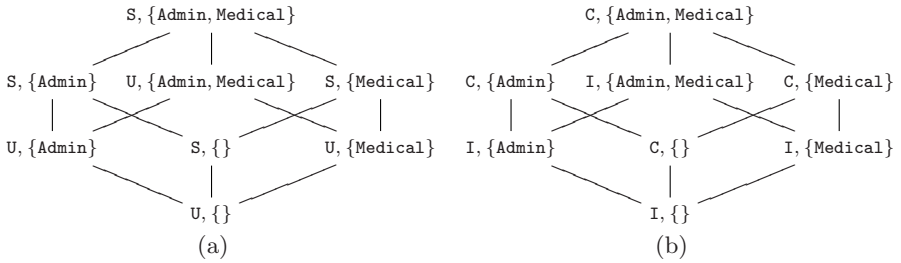

Fig. 4. An example of security (a) and integrity (b) lattices

rizations (Medical,Document1,$+r$) and (Nurse,Document1,$-r$). Carol cannot read Document1, since the Nurse group is more specific than the Medical group.

– *Most specific along a path takes precedence.* An authorization associated with an element n overrides a contradicting authorization associated with an ancestor n' for all the descendants of n, only for the paths passing from n. The overriding has no effect on other paths. For instance, with respect to the previous example, Carol gains a positive authorization from the path ⟨Medical,Doctor,Carol⟩, and a negative one from path ⟨Nurse,Carol⟩.

While convenient for their expressiveness and flexibility, in high security settings discretionary access control results limited for its vulnerability to *Trojan horses*. The reason for this vulnerability is that discretionary access control does not distinguish between *users* (i.e., human entity whose identity is exploited to select the privileges for making the access control decision) and *subjects* (i.e., process generated by a user and that makes requests to the system). A discretionary access control system evaluates the requests made by a subject against the authorizations of the user who generated the corresponding process. It is then vulnerable from processes executing malicious programs that exploit the authorizations of the user invoking them. Protection against these processes requires controlling the flows of information within processes execution and possibly restricting them. Mandatory policies provide a way to enforce information flow control through the use of labels.

2.2 Mandatory Access Control

Mandatory security policies enforce access control on the basis of regulations mandated by a central authority. The most common form of mandatory policy is the *multilevel security policy*, based on the classifications of subjects and objects in the system. Each subject and object in the system is associated with an *access class*, usually composed of a *security level* and a set of *categories*. Security levels in the system are characterized by a total order relation, while

categories form an unordered set. As a consequence, the set of access classes is characterized by a partial order relation, denoted $\geq$ and called *dominance*. Given two access classes c_1 and c_2, c_1 *dominates* c_2, denoted $c_1 \geq c_2$, iff the security level of c_1 is greater than or equal to the security level of c_2 and the set of categories of c_1 includes the set of categories of c_2. Access classes together with their partial order dominance relationship form a *lattice* [9].

Mandatory policies can be classified as secrecy-based and integrity-based, operating in a dual manner.

Secrecy-Based Mandatory Policy [10, 11, 12, 13]. The main goal of secrecy-based mandatory policies is to protect data confidentiality. As a consequence, the security level of the access class associated with an object reflects the sensitivity of its content, while the security level of the access class associated with a subject, called *clearance*, reflects the degree of trust placed in the subject not to reveal sensitive information. The set of categories associated with both subjects and objects defines the area of competence of users and data. A user can connect to the system using her clearance or any access class dominated by her clearance. A process generated by a user connected with a specific access class has the same access class as the user.

The access requests submitted by a subject are evaluated by applying the following two principles.

No-Read-Up. A subject s can read an object o if and only if the access class of the subject dominates the access class of the object.

No-Write-Down. A subject s can write an object o if and only if the access class of the object dominates the access class of the subject.

Consider, as an example, the security lattice in Fig. 4(a), where there are two security levels, `Secret` (`S`) and `Unclassified` (`U`), with `S`>`U`, and the set of categories {`Admin`, `Medical`}. Suppose that user `Ann` has clearance ⟨`S`,{`Admin`}⟩ and she connects to the system as the ⟨`S`,{}⟩ subject. She is allowed to read objects ⟨`S`,{}⟩ and ⟨`U`,{}⟩. She can write objects with access class ⟨`S`,{}⟩, ⟨`S`,{`Admin`}⟩, ⟨`S`,{`Medical`}⟩, and ⟨`S`,{`Admin`,`Medical`}⟩.

Note that a user is allowed to connect to the system at different access classes to the aim of accessing information at different levels (provided that she is cleared for it). Otherwise, these accesses would be blocked by the no-write-down principle.

The principles of the secrecy-based mandatory policy prevent information flows from high level subjects/objects to subjects/objects at lower (or incomparable) levels, thus preserving information confidentiality. However, these two principles may turn out to be too restrictive. For instance, in a real scenario data may need to be downgraded (e.g., this may happen at the end of the embargo). To consider also these situations, the secrecy-based mandatory models can allow exceptions for processes that are *trusted* and ensure that the information produced is *sanitized*.

Integrity-Based Mandatory Policy [14]. The main goal of integrity-based mandatory policies is to prevent subjects from *indirectly* modifying information they cannot write. The integrity level associated with a user reflects then the degree of trust placed in the subject to insert and modify sensitive information. The integrity level associated with an object indicates the degree of trust placed on the information stored in the object and the potential damage that could result from unauthorized modifications of the information. Again, the set of categories associated with both subjects and objects defines the area of competence of users and data.

The access requests submitted by a subject are evaluated by applying the following two principles.

No-Read-Down. A subject s can read an object o if and only if the integrity class of the object dominates the integrity class of the subject.

No-Write-Up. A subject s can write an object o if and only if the integrity class of the subject dominates the integrity class of the object.

Consider, as an example, the integrity lattice in Fig. 4(b), where there are two integrity levels `Crucial` (`C`) and `Important` (`I`), with `C`>`I`, and the set of categories {`Admin`, `Medical`}. Suppose that user `Ann` connects to the system as the ⟨`C`,{`Admin`}⟩ subject. She can read objects having integrity class ⟨`C`,{`Admin`}⟩ and ⟨`C`,{`Admin`,`Medical`}⟩ and she can write objects with integrity class ⟨`C`,{`Admin`}⟩, ⟨`C`, {}⟩, ⟨`I`,{`Admin`}⟩, and ⟨`I`,{}⟩.

These two principles are the dual with respect to the principles adopted by secrecy-base policies. As a consequence, the integrity model prevents flows of information from low level objects to higher objects. A major limitation of this model is that it only captures integrity breaches due to improper information flows. However, integrity is a much broader concept and additional aspects should be taken into account [15].

Note that secrecy-based and integrity-based models are not mutually exclusive, since it may be useful to protect both the confidentiality and the integrity properties. Obviously, in this case, objects and subjects will be associated with both a security and an integrity class.

A major drawback of mandatory policies is that they control only flows of information happening through *overt channels*, that is, channels operating in a legitimate way. As a consequence, the mandatory policies are vulnerable to *covert* channels [16], which are channels not intended for normal communication but that still can be exploited to infer information. For instance, if a low level subject requests the use of a resource currently used by a high level subject, it will receive a negative response, thus inferring that another (higher level) subject is using the same resource.

2.3 Role-Based Access Control

A third approach for access control is represented by *Role-Based Access Control* (RBAC) models [17, 18]. A *role* is defined as a set of privileges that any

user playing that role is associated with. When accessing the system, each user has to specify the role she wishes to play and, if she is granted to play that role, she can exploit the corresponding privileges. The access control policy is then defined through two different steps: first the administrator defines roles and the privileges related to each of them; second, each user is assigned with the set of roles she can play. Roles can be hierarchically organized to exploit the propagation of access control privileges along the hierarchy.

A user may be allowed to simultaneously play more than one role and more users may simultaneously play the same role, even if restrictions on their number may be imposed by the security administrator.

It is important to note that roles and groups of users are two different concepts. A group is a named collection of users and possibly other groups, and a role is a named collection of privileges, and possibly other roles. Furthermore, while roles can be activated and deactivated directly by users at their discretion, the membership in a group cannot be deactivated.

The main advantage of RBAC, with respect to DAC and MAC, is that it better suits to commercial environments. In fact, in a company, it is not important the identity of a person for her access to the system, but her responsibilities. Also, the role-based policy tries to organize privileges mapping the organization's structure on the roles hierarchy used for access control.

3 Credential-Based Access Control

In an open and dynamic scenario, parties may be unknown to each other and the traditional separation between *authentication* and *access control* cannot be applied anymore. Such parties can also play the role of both client, when requesting access to a resource, and server for the resources it makes available for other users in the system. Advanced access control solutions should then allow to decide, on one hand, which requester (client) is to be granted access to the resource, and, on the other hand, which server is qualified for providing the same resource. *Trust management* has been developed as a solution for supporting access control in open environments [19]. The first approaches proposing a trust management solution for access control are PolicyMaker [20] and KeyNote [21]. The key idea of these proposals is to bind public keys to authorizations and to use credentials to describe specific delegations of trust among keys. The great disadvantage of these early solutions is that they assign authorizations directly to users' keys. The authorization specification is then difficult to manage and, moreover, the public key of a user may act as a pseudonym of herself, thus reducing the advantages of trust management, where the identity of the users should not be considered.

The problem of assigning authorizations directly to keys has been solved by the introduction of *digital certificates*. A digital certificate is the on-line counterpart of paper credentials (e.g., a driver licence). A digital certificate is a statement, certified by a trusted entity (the certificate authority), declaring

a set of properties of the certificate's holder (e.g., identity, accreditation, or authorizations). Access control models, by exploiting digital certificates for granting or denying access to resources, make access decisions on the basis of a set of properties that the requester should have. The final user can prove to have such properties by providing one or more digital certificates [22, 23, 24, 25, 26].

The development and effective use of credential-based access control models require however tackling several problems related to credential management and disclosure strategies, delegation and revocation of credentials, and establishment of credential chains [27, 28, 29, 30]. In particular, when developing an access control system based on credentials, the following issues need to be carefully considered [22].

- *Ontologies.* Since there is a variety of security attributes and requirements that may need to be considered, it is important to guarantee that different parties will be able to understand each other, by defining a set of common languages, dictionaries, and ontologies.
- *Client-side and server-side restrictions.* Since parties may act as either a client or a server, access control rules need to be defined both client-side and server-side.
- *Credential-based access control rules.* New access control languages supporting credentials need to be developed. These languages should be both expressive (to define different kinds of policies) and simple (to facilitate policy definition).
- *Access control evaluation outcome.* The resource requester may not be aware of the attributes she needs to gain access to the requested resource. As a consequence, access control mechanisms should not simply return a permit or deny answer, but should be able to ask the final user for the needed credentials to access the resource.
- *Trust negotiation strategies.* Due to the large number of possible alternative credentials that would enable an access request, a server cannot formulate a request for all these credentials, since the client may not be willing to release the whole set of her credentials. On the other hand, the server should not disclose too much of the underlying security policy, since it may contain sensitive information.

In the following, we briefly describe some proposals that have been developed for trust negotiation and for regulating service access in open environments.

3.1 Overview of Trust Negotiation Strategies

As previously noted, since the interacting parties may be unknown to each other, the resource requester may not be aware of the credentials necessary for gaining access privileges. Consequently, during the access control process,

the two parties exchange information about the credentials needed for access. The access control decision comes then after a complex process, where parties exchange information not only related to the access itself, but also to additional restrictions imposed by the counterpart. This process, called *trust negotiation*, has the main goal of establishing trust between the interacting parties in an automated manner. A number of trust negotiation strategies have been proposed in the literature, which are characterized by the following steps.

- The client first requests to access a resource.
- The server then checks if the client provided the necessary credentials. In case of a positive answer, the server grants access to the resource; otherwise it communicates the client the policies that she has to fulfill.
- The client selects the requested credentials, if possible, and sends them to the server.
- If the credentials satisfy the request, the client is granted access to the resource.

This straightforward trust negotiation process suffers of privacy problems, since both the server discloses its access control policy entirely and the client exposes all her certificates to gain access to a resource. To solve such an inconvenience, a *gradual trust establishment* process can be enforced [31]. In this case, upon receiving an access request, the server selects the policy that governs the access to the service and discloses only the information that it is willing to show to an unknown party. The client, according to its practices, decides if it is willing to disclose the requested credentials. Note that this incremental exchange of requests and credentials can be iteratively repeated as many times as necessary.

PRUdent NEgotiation Strategy (PRUNES) is another negotiation strategy whose main goal is to minimize the number of certificates that the client communicates to the server [30]. It also ensures that the client communicates her credentials to the server only if the access will be granted. Each party defines a set of *credential policies* on which the negotiation process is based. The established credential policies can be graphically represented through a tree, called *negotiation search tree*, composed of two kinds of nodes: *credential nodes*, representing the need for a specific credential, and *disjunctive nodes*, representing the logic operators connecting the conditions for credential release. The root of the tree represents the resource the client wants to access. The negotiation process can be seen as a backtracking operation on the tree. To the aim of avoiding the cost of a brute-force backtracking, the authors propose the *PRUNES* method to prune the search tree without compromising completeness or correctness of the negotiation process. The basic idea is that if a credential has just been evaluated and the state of the system has not changed too much, then it is useless to evaluate again the same credential.

A large set of negotiation strategies, called *disclosure tree strategy* (DTS) family [32], has been also defined and proved to be closed. This means that,

if two parties use different strategies from the DST family, they will be able to negotiate trust. A *Unified Schema for Resource Protection* (UniPro) [33] has been proposed to protect the information specified within policies. UniPro gives (opaque) names to policies and allows any named policy P_1 to have its own policy P_2, meaning that the content of P_1 can only be disclosed to parties who satisfy P_2. Another solution is the *Adaptive Trust Negotiation and Access Control* (ATNAC) approach [34]. This method grants (or denies) access on the basis of a *suspicion level* associated with subjects. The suspicion level is not fixed but may vary on the basis of the probability that the user has malicious intents.

It is important to note that in recent, more complicated, scenarios disclosure policies can be defined both on resources and on credentials [22]. In this case, the client, upon receiving a request for a certificate, can answer with a counter-request to the server for another certificate.

3.2 Overview of a Credential-Based Access Control Framework

One of the first solutions providing a uniform framework for credential-based access control specification and enforcement was presented by Bonatti and Samarati [22]. The proposed access control system includes an access control model, a language, and a policy filtering mechanism.

The paper envisions a system composed of two entities: a *client* and a *server*, interacting through a predefined negotiation process. The server is characterized by a set of resources. Both the client and the server have a *portfolio*, which is a collection of credentials (i.e., statements issued by authorities trusted for making them [35]) and declarations (statements issued by the party itself). Credentials correspond to digital certificates and are guaranteed to be unforgeable and verifiable through the public key of the issuing authority.

To the aim of performing gradual trust establishment between the two interacting parties, the server defines a set of *service accessibility rules*, and both the client and the server define their own set of *portfolio disclosure rules*. The service accessibility rules specify the necessary and sufficient conditions for accessing a resource, while portfolio disclosure rules define the conditions that govern the release of credentials and declarations. Both the two classes of rules are expressed by using a logic language. A special class of predicates is represented by *abbreviations*. Since there may exist a number of alternative combinations of certificates allowing access to a resource, *abbreviation predicates* may be used for reducing the communication cost of such certificates. The predicates of the language adopted exploit the current *state* (i.e., parties' characteristics, certificates already exchanged in the negotiation, and requests made by the parties) to take a decision about a release. The information about the state is classified as *persistent state*, when the information is stored at the site and spans different negotiations, and *negotiation state*, when it is acquired during the negotiation and is deleted when the same terminates.

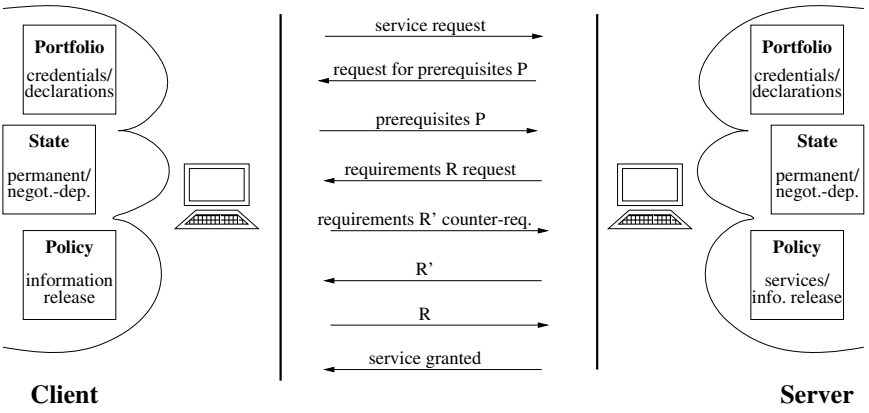

Fig. 5. Client-server negotiation

The main advantage of this proposal is that it maximizes both server and client's privacy, by minimizing the set of certificates exchanged. In particular, the server discloses the minimal set of policies for granting access, while the client releases the minimal set of certificates to access the resource. To this purpose, service accessibility rules are distinguished in *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. Therefore, the server will not disclose a requisite rule until the client satisfies a prerequisite rule. Figure 5 illustrates the resulting client/server interaction. It is important to highlight here that, before releasing rules to the client, the server needs 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.

4 Policy Composition

In many real word scenarios, access control enforcement needs to take into consideration different policies independently stated by different administrative subjects, which must be enforced as if they were a single policy. As an example of policy composition, consider an hospital, where the global policy may be obtained by combining together the policies of its different wards and the externally imposed constraints (e.g., privacy regulations). Policy composition is becoming of paramount importance in all those contexts in which administrative tasks are managed by different, non collaborating, entities.

Policy composition is an orthogonal aspect with respect to policy models, mechanisms, and languages. As a matter of fact, the entities expressing the

policies to be composed may even not be aware of the access control system adopted by the other entities specifying access control rules. The main desiderata for a policy composition framework can be summarized as follows [36].

- *Heterogeneous policy support.* The framework should support policies expressed in different languages and enforced by different mechanisms.
- *Support of unknown policies.* The framework should support policies that are not fully defined or are not fully known when the composition strategy is defined. Consequently, policies are to be treated as black-boxes and are supposed to return a correct and complete response when queried at access control time.
- *Controlled interference.* The framework cannot simply merge the sets of rules defined by the different administrative entities, since this behavior may cause side effects. For instance, the accesses granted/denied might not correctly reflect the specifications anymore.
- *Expressiveness.* The framework should support a number of different ways for combining the input policies, without changing the input set of rules or introducing ad-hoc extensions to authorizations.
- *Support of different abstraction levels.* The composition should highlight the different components and their interplay at different levels of abstraction.
- *Formal semantics.* The language for policy composition adopted by the framework should be declarative, implementation independent, and based on a formal semantic to avoid ambiguity.

We now briefly describe some solutions proposed for combining different policies.

4.1 Overview of Policy Composition Solutions

Various models have been proposed to reason about security policies [37, 38, 39, 40]. In [37, 39] the authors focus on the secure behavior of program modules. McLean [40] introduces *the algebra of security*, which is a Boolean algebra that enables to reason about the problem of policy conflict, arising when different policies are combined. However, even though this approach permits to detect conflicts between policies, it does not propose a method to resolve the conflicts and to construct a security policy from inconsistent subpolicies. Hosmer [38] introduces the notion of meta-policies, which are defined as policies about policies. Metapolicies are used to coordinate the interaction about policies and to explicitly define assumptions about them. Subsequently, Bell [41] formalizes the combination of two policies with a function, called *policy combiner*, and introduces the notion of *policy attenuation* to allow the composition of conflicting security policies. Other approaches are targeted to the development of a uniform framework to express possibly heterogeneous policies [42, 43, 44, 45, 46].

A different approach has been illustrated in [36], where the authors propose an algebra for combining security policies together with its formal semantics. Here, a policy, denoted P_i, 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 S, o is a constant in (or a variable over) the set of objects O, and a is a constant in (or a variable over) the set of actions A. Policies of this form are composed through a set of *algebra operators* whose syntax is represented by the following BNF:

$$E ::= \mathbf{id} | E + E | E \& E | E - E | E \hat{} C | o(E,E,E) | E * R | T(E) | (E)$$
$$T ::= \tau \mathbf{id}.E$$

where **id** is a unique policy identifier, E is a policy expression, T is a construct, called *template*, C is a construct describing constraints, and R is a construct describing rules. The order of evaluation of algebra operators is determined by the precedence, which is (from higher to lower) τ, ., + and & and -, * and ^.

The semantic of algebra operators is defined by a function that maps policy expressions in a set of ground authorizations (i.e., a set of authorization triples). The function that maps policy identifiers into sets of triples is called *environment*, and is formally defined as follows.

Definition 1. *An* environment *e is a partial mapping from policy identifiers to sets of authorization triples. By $e[X/S]$ we denote a modification of environment e such that*

$$e[X/S](Y) = \begin{cases} S & \text{if } Y = X \\ e(Y) & \text{otherwise} \end{cases}$$

The semantic of an identifier X in the environment e is denoted by $[\![X]\!]_e = e(X)$.

The operators defined by the algebra for policy composition basically reflect the features supported by classical policy definition systems. As an example, it is possible to manage exceptions (such as negative authorizations), propagation of authorizations, an so on. The set of operators together with their semantic is briefly described in the following.

- *Addition* (+). It merges two policies by returning their union.

  $$[\![P_1 + P_2]\!]_e = [\![P_1]\!]_e \cup [\![P_2]\!]_e$$

  Intuitively, additions can be applied in any situation where accesses can be authorized if allowed by any of the component policies (maximum privilege principle).
- *Conjunction* (&). It merges two policies by returning their intersection.

  $$[\![P_1 \& P_2]\!]_e = [\![P_1]\!]_e \cap [\![P_2]\!]_e$$

  This operator enforces the minimum privilege principle.

- *Subtraction* ($-$). It deletes from a first policy, all the authorizations specified in a second policy.

$$[\![P_1 - P_2]\!]_e = [\![P_1]\!]_e \setminus [\![P_2]\!]_e$$

  Intuitively, subtraction operator is used to handle exceptions, and has the same functionalities of negative authorizations in existing approaches. It does not generate conflicts since P_1 prevails on P_2 by default.
- *Closure* ($*$). It closes a policy under a set of derivation rules.

$$[\![P * R]\!]_e = \mathbf{closure}(R, [\![P]\!]_e)$$

  The closure of policy P under derivation rules R produces a new policy that contains all the authorizations in P and those that can be derived evaluating R on P, according to a given semantics. The derivation rules in R can enforce, for example, an authorization propagation along a predefined subject or object hierarchy.
- *Scoping Restriction* ($\hat{}$). It restricts the applicability of a policy to a given subset of subjects, objects, and actions of the system.

$$[\![P_1\hat{}c]\!]_e = \{t \in [\![P]\!]_e \mid t \text{ satisfy c}\}$$

  where c is a condition. It is useful when administration entities need to express their policy on a confined subset of subjects and/or objects (e.g., each ward can express policies about the doctors working in the ward).
- *Overriding* (o). It overrides a portion of policy P_1 with the specifications in policy P_2; the fragment that is to be substituted is specified by a third policy P_3.

$$[\![o(P_1, P_2, P_3)]\!]_e = [\![(P_1 - P_3) + (P_2 \& P_3)]\!]_e$$

- *Template*(τ). It defines a partially specified (i.e., parametric) policy that can be completed by supplying the parameters.

$$[\![\tau X.P]\!]_e(S) = [\![P]\!]_{e[S/X]}$$

  where S is the set of all policies, and X is a parameter. Templates are useful for representing policies as black-boxes. They are needed any time when some components are to be specified at a later stage. For instance, the components might be the result of a further policy refinement, or might be specified by a different authority.

Due to the formal definition of the semantic of algebra operators, it is possible to exploit algebra expressions to formally prove the security properties of the obtained (composed) policy.

Once the policies have been composed through the algebraic operators described above, for their enforcement it is necessary to provide executable specifications compatible with different evaluation strategies. To this aim, the authors propose the following three main strategies to translate policy expressions into logic programs.

- *Materialization.* The expressions composing policies are explicitly evaluated, by obtaining a set of ground authorizations that represents the policy that needs to be enforced. This strategy can be applied when all the composed policies are known and reasonably static.
- *Partial materialization.* Whenever materialization is not possible since some of the policies to be composed are not available, it is possible to materialize only a subset of the final policy. This strategy is useful also when some of the policies are subject to sudden and frequent changes, and the cost of materialization may be too high with respect to the advantages it may provide.
- *Run-time evaluation.* In this case no materialization is performed and run-time evaluation is needed for each request (access triple), which is checked against the policy expressions to determine whether the triple belongs to the result.

The authors then propose a method (*pe2lp*) for transforming algebraic policy composition expressions into a logic program. The method proposed can be easily adapted to one of the three materialization strategies introduced above. Basically, the translation process creates a distinct predicate symbol for each policy identifier and for each algebraic operator in the expression. The logic programming formulation of algebra expressions can be used to enforce access control. As already pointed out while introducing algebra operators, this policy composition algebra can also be used to express simple access control policies, such as open and closed policy, propagation policies, and exceptions management. For instance, let us consider a hospital composed of three wards, namely *Cardiology*, *Surgery*, and *Orthopaedics*. Each ward is responsible for granting access to data under its responsibility. Let $P_{Cardiology}$, $P_{Surgery}$ and $P_{Orthopaedics}$ be the policies of the three wards. Suppose now that an access is authorized if any of the wards policies state so and that authorizations in policy $P_{Surgery}$ are propagated to individual users and documents by classical hierarchy-based derivation rules, denoted R_H. In terms of the algebra, the hospital policy can be represented as follows.

$$P_{Cardiology} \& P_{Surgery} * R_H \& P_{Orthopaedics}$$

Following this work, Jajodia et al. [47] presented a propositional algebra for policies with a syntax consisting of abstract symbols for atomic policy expressions and composition operators.

5 Access Control Through Encryption

Since the amount of data that organizations need to manage is increasing very quickly, data outsourcing is becoming more and more attractive. Data outsourcing provides data storage at a low rate, allowing the data owner to

concentrate its activity on its core business where data are managed by an external service provider. The main drawback of this practice is that the service provider may not be fully trusted. The data owner and final users are usually supposed to trust the provider for managing data stored on its server, and to correctly execute queries on it, but the provider is not fully trusted for accessing data content. To solve this problem, different solutions have been proposed in the literature, mainly based on the use of cryptography as a mechanism for protecting data privacy [1, 2, 3]. Most of the proposals in this area focus on issues related to querying encrypted data, to the aim of avoiding server-side decryption, while minimizing client-side burden in query evaluation. Another drawback of existing proposals is that they assume that any client has complete access to the query results, and therefore the data owner has to be involved for filtering out the data not accessible by the client. This would cause an excessive burden on the owner, thus nullifying the advantages of outsourcing data management. On the other hand, the remote server cannot enforce access control policies, since it may not be allowed to know the access control policy defined by the owner. Since neither the data owner nor the remote server can enforce the access control policy, for either security or efficiency reasons, the data themselves need to implement selective access. This can be realized through *selective encryption*, which consists in encrypting data using different keys and distributing the keys so that users can decrypt only the data they are authorized to access.

The problem of enforcing access control policies through selective encryption has been analyzed both for databases and for XML documents. In the following, we briefly introduce the most important proposals for these two scenarios [48, 49, 50].

5.1 Overview of Database Outsourcing Solutions

Let us consider a system composed of a set $\mathcal{U}$ of users and a set $\mathcal{R}$ of resources. A resource may be a table, an attribute, a tuple, or even a cell, depending on the granularity at which the data owner wishes to define her policy. Since this distinction does not affect access control policy enforcement, we will always refer generically to resources. The access control policy defined by the data owner can be easily represented through a traditional access matrix $\mathcal{A}$, where each cell $\mathcal{A}[u,r]$ may assume either the value 1, if u can access r, or the value 0, otherwise (currently only read privileges have been considered). Figure 6 represents an example of access matrix, where there are four users, namely A, B, C, and D, and four resources r_1, r_2, r_3, and r_4.

A first solution that could be adopted for selectively encrypting data for access control purposes consists in using a different key for each resource, and in communicating each user the set of keys used to protect the resources belonging to her capability list (i.e., the set of resources that the user can access). This solution requires each user to keep a possibly great number of