24

A Decentralized Temporal Authorization Model

Elisa Bertino, Claudio Bettini, Elena Ferrari and Pierangela Samarati
Universitá di Milano
Dipartimento di Scienze dell'Informazione, Universitá di Milano, Via Comelico 39/41, Milano, Italy.
Email: {ebertino,bettini,ferrarie,samarati}@dsi.unimi.it

Abstract

This paper describes a temporal authorization model with decentralized administration facilities. The model supports both positive and negative authorizations. Each authorization is associated with a time interval, limiting the validity of the authorization. Authorizations can be specified explicitly or derived through rules. Administration of authorizations is decentralized and is based on different types of administrative privileges together with the grant option. Revocation is recursive in that whenever a user is revoked an authorization, the authorizations he granted may also be revoked.

Keywords

Database Management, Database Security, Temporal Authorization, Access Control

1 INTRODUCTION

Authorization models available today allow the specification of authorizations stating permissions for users to exercise operations on objects. When a user must be allowed for an operation on an object, an authorization is granted to him. The user is then allowed for the access until the authorization is explicitly removed. This simple paradigm does not fit real-life situations where more complex security requirements arise. In this paper we make a step towards enhancing the expressive power of authorization models with respect to the treatment of temporal aspects in the specification of authorizations.

Although time issues in access control and derivation rules for authorizations have received growing attention among the researchers (Steiner et al. 1988, Thomas et al. 1993, Woo et al. 1993), access control mechanisms provided as part of commercial data management systems and research prototypes (Bobrowski 1993, Castano et al. 1995) do not have temporal capabilities. For example, in a typical Relational DBMS (RDBMS) it is not possible to specify, by using the authorization command language, that a user may access a relation only for a day or a month. If such a need arises, authorization management and access control must be implemented as application programs, thus making authorization management very difficult.

S. K. Katsikas et al. (eds.), *Information Systems Security*

A temporal authorization model has been presented by us in a previous paper (Bertino et al. 1996). In this paper we extend the model with decentralized administration of authorizations. The owner of an object can delegate other users the privilege to administer his objects. Two different types of administrative privileges are considered. To allow selective delegation of administrative privileges, we permit authorizations to be specified with the grant option. If a user owns an authorization for an access mode on an object with the grant option for a given time interval, he can grant the privilege (or its negation), as well as the grant option, to other users for each instant for which the authorization with the grant option holds. The resulting model thus provides a high degree of flexibility by supporting decentralized authorization administration, coupled with the possibility of enforcing stricter controls on particular crucial data items through the use of negative authorizations and temporal authorization validity. The decentralized administration model proposed in this paper enforces recursive revocation: whenever a user revokes an authorization for a privilege on an object to another user, all the authorizations that the revokee has granted thanks to the revoked authorization are removed. The revocation is recursively applied to all the users that received the access authorization from the revokee. We extend the semantics of recursive revocation first proposed by Griffiths and Wade in the framework of the System R database model (Griffiths et al. 1976) and formally defined by Fagin (Fagin 1976), to the consideration of temporal authorizations. Since in our model each authorization has a time interval, a revoke request can cause not only the deletion of other authorizations, besides the ones whose deletion is explicitly required, but also the modification of their time intervals or the splitting of one authorization in more than one.

The remainder of the paper is organized as follows. Section 2 illustrates the authorization model. Section 3 discusses the administrative policy that regulates grant and revocation of authorizations and administrative privileges. Section 4 describes the semantics of the revoke operation. Finally Section 5 concludes the paper.

2 THE AUTHORIZATION MODEL

In this section we illustrate our authorization model. In the following U denotes the set of users, O the set of objects, and M the set of access modes.

We consider both positive and negative authorizations. A positive authorization states a permission for a user to exercise a privilege on an object. A negative authorization states a denial for a user to exercise a privilege on an object. Positive authorizations can be granted with the *grant option*. If a user holds a positive authorization for a privilege on an object with the grant option, the user can also grant (and revoke) other users authorizations (positive or negative) for the privilege on the object.* Authorizations are defined as follows.

Definition 1 (Authorization) *An authorization is a 6-tuple* (s,o,m,pn,g,go) *where* $\mathtt{s} \in U$, $\mathtt{o} \in O$, $\mathtt{m} \in M$, $\mathtt{pn} \in \{+,-\}$, $\mathtt{g} \in U$, $\mathtt{go} \in \{\mathtt{yes},\mathtt{no}\}$.

*The model could be easily extended to allow a different administration of positive and negative authorizations.

Tuple (s,o,m,pn,g,go) states that user s can exercise (if pn='+') or cannot exercise (if pn= '−') access mode m on object o, and that this authorization was granted by user g. If go='yes', s can also grant/revoke other users authorizations (positive or negative), for m on o as well as the grant option. Since only positive authorizations can be granted with the grant option, authorizations with pn = '−' have necessarily go = 'no'.

We associate with each authorization a temporal constraint representing the set of time instants in which the authorization holds. We take as our model of time the natural numbers $\mathbb{N}$ with the total order relation $<$. We refer to authorizations together with their temporal constraints as *temporal authorization.*

Definition 2 (Temporal authorization) *A temporal authorization is a triple* (ts, time, auth)*, where* ts $\in \mathbb{N}$ *is the time at which the authorization was granted* †*,* time *is a time interval* $[t_i, t_j]$*, with* $t_i \in \mathbb{N}$, $t_j \in \mathbb{N} \cup \infty$, ts $\leq t_i \leq t_j$*, and* auth *is an authorization.*

The temporal authorization (5,[10,40],(Alice,o_1,write,+,Bob,yes)) states that at time 5 Bob granted to Alice the authorization to write object o_1 between instants 10 and 40. Since the authorization is with the grant option, Ann can grant other users positive or negative authorizations for the write privilege on o_1 for time intervals in [10,40].

Note that, given a temporal authorization A, t_i(A)≥ts(A), i.e., the starting time of the authorization must be greater than or equal to the time at which the authorization is granted. Note also that a user can only grant privileges he owns. Then, if a user holds an authorization for an access mode on an object with the grant option for a time interval $[t_i, t_j]$, his privilege to authorize or deny other users to exercise the access mode on the object is limited to the interval $[t_i, t_j]$. In the following, given a temporal authorization A = (ts,[t_i,t_j],(s,o,m,pn,g,go)), we denote with s(A),o(A),m(A),pn(A),g(A),go(A) the subject, the object, the privilege, the sign, the grantor, and the grant option in A, respectively . Moreover, we denote with ts(A) the time when A has been granted, and with [t_i(A),t_j(A)] the temporal validity of A.

Additional authorizations can be derived from the authorizations explicitly specified. The derivation is based on temporal propositions, used as rules, which allow new temporal authorizations to be derived on the basis of the presence or the absence of other temporal authorizations. Derivation rules can be applied to positive as well as to negative authorizations. Like authorizations, derivation rules have a time interval associated with them representing the set of time instants in which the rule is applied. Derivation rules are defined as follows.

Definition 3 (Derivation rule) *A derivation rule is defined as* ([t_i,t_j], A_l⟨op⟩A_r)*, where* [t_i,t_j] *is a time interval,* $t_i \in \mathbb{N}$, $t_j \in \mathbb{N} \cup \infty$, $t_i \leq t_j$, A_l *and* A_r *are authorizations,* g(A_l) *is the user who specifies the rule,* go(A_l)= 'no'*, and* ⟨op⟩ *is one of the following operators:* WHENEVER, ASLONGAS, WHENEVERNOT, UNLESS.

Note that for sake of simplicity, we restrict rules to the derivation of authorizations

†Timestamps are introduced to prevent cycles among authorizations (Griffiths et al. 1976).

without the grant option. Indeed, allowing grant option in rules would make authorization management cumbersome. The intuitive semantics of derivation rules is as follows[‡]:

- $([t_i,t_j],A_1$ WHENEVER $A_2)$. We can derive A_1 for each instant in $[t_i,t_j]$ for which A_2 is given or derived. For example, rule R_1 in Figure 1, specified by Tom, states that every time in [10,90], Bob can read object o_1 thanks to an authorization with the grant option granted by Tom, also Alice can read object o_1.
- $([t_i,t_j],A_1$ ASLONGAS $A_2)$. We can derive A_1 for each instant t in $[t_i,t_j]$ such that A_2 is either given or derived for each instant from t_i to t. Unlike the WHENEVER operator, the ASLONGAS operator does not allow to derive A_1 at an instant t in $[t_i,t_j]$ if there exists an instant t', $t_i \leq t' \leq t$, such that A_2 is not given and cannot be derived at t'.
- $([t_i,t_j],A_1$ WHENEVERNOT $A_2)$. We can derive A_1 for each instant in $[t_i,t_j]$ for which A_2 is neither given nor derived.
- $([t_i,t_j],A_1$ UNLESS $A_2)$. We can derive A_1 for each instant t in $[t_i,t_j]$ such that A_2 is neither given nor can be derived for each instant from t_i to t. Unlike the WHENEVERNOT, the UNLESS operator does not allow to derive A_1 at an instant t in $[t_i,t_j]$ if there exists an instant t', $t_i \leq t' \leq t$, such that A_2 is given or derived at t'.

(A_1) (5,([10,40],(Bob,o_1,read,+,Tom,yes)))
(A_2) (8,([41,50],(Bob,o_1,read,+,Tom,yes)))
(A_3) (40,([80,100],(Bob,o_1,read,+,Tom,yes)))
(A_4) (20,([30,50],(Ann,o_1,read,−,Bob,no)))
(R_1) ([10,90],(Alice,o_1,read,+,Tom,no) WHENEVER (Bob,o_1,read,+,Tom,yes))
(R_2) ([20,100],(Sam,o_1,read,+,Tom,no) UNLESS (Ann,o_1,read,−,Bob,no))
(R_3) ([30,∞],(John,o_1,read,+,Tom,no) WHENEVERNOT (Alice,o_1,read,+,Tom,no))
(R_4) ([30,200],(Matt,o_1,read,+,Tom,no) ASLONGAS (Bob,o_1,read,+,Tom,yes))

Figure 1 An example of authorizations and derivation rules.

Example 1 Consider the authorizations and derivation rules illustrated in Figure 1. The following temporal authorizations can be derived:

- ([10,50],(Alice,o_1,read,+,Tom,no)), and ([80,90],(Alice,o_1,read,+,Tom,no)), from authorizations A_1, A_2, and A_3 and rule R_1.
- ([20,29],(Sam,o_1,read,+,Tom,no)) from authorization A_4 and rule R_2.
- ([51,79],(John,o_1,read,+,Tom,no)), and ([91,∞],(John,o_1,read,+,Tom,no)) from rules R_3 and R_1 and authorizations A_1, A_2, and A_3.
- ([30,50],(Matt,o_1,read,+,Tom,no)) from authorizations A_1 and A_2 and rule R_4.

The possibility of specifying both positive and negative authorizations introduces potential conflicts. A conflict arises whenever a user holds both a positive and a negative

[‡]We refer the reader to (Bertino et al. 1995) for the formal semantics.

authorization for the same privilege on the same object and the intervals associated with the two authorizations are not disjoint. Conflicts are solved according to the denials-take-precedence principle, i.e., by denying the access in the instants of time in both intervals.

Note that, the presence of rules involving the WHENEVERNOT or UNLESS operator, besides increasing the expressiveness of the model, introduces the problem of generating a unique set of authorizations from a given set of authorizations and rules. The set of derived authorizations could depend on the evaluation order of rules. To solve these difficulties we have introduced appropriate syntactic restrictions on rules that ensure the uniqueness of the set of derivable authorizations (Bertino et al. 1996).

In the following, we use the term TAB (Temporal Authorization Base) to indicate the set of temporal authorizations and derivation rules present at a given time in the system.

3 AUTHORIZATION ADMINISTRATION

The user creating an object receives the `own` privilege on it, that allows the user to grant and revoke other users authorizations on the objects. The owner of an object can also delegate other users the privilege to administer authorizations on the object. Two different administrative privileges are considered: `refer` and `administer`. If a user has the refer privilege on an object, he can specify derivation rules in which the object appears in the authorization at the right of the temporal operator. If a user has the administer privilege on an object he can grant to and revoke from other users authorizations (negative or positive and with or without grant option) on the object either explicitly or through rules.

Decentralized administration of authorizations can also be granted selectively on single privileges through the use of the grant option. Note however that users holding the grant option for a privilege on an object can grant only explicit authorizations for the privilege on the object; they are not allowed to specify rules for the derivation of these authorizations. The reason for this is that rules can be very powerful and computationally expensive. Rules for the derivation of authorizations on an object can be specified only by users holding either the `own` or the `administer` privilege on the object.

Granting and revocation of authorizations and of administrative privileges is enforced through administrative operations. The administrative operations can be classified as follows:

- **Operations involving explicit authorizations.** These operations allow users to grant or revoke explicit authorizations, both positive and negative, on an object. The user requesting them must have the `own` or the `administer` privilege on the object or an authorization for the privilege on the object with the grant option. The revoke operation can be required with reference to a single authorization, or with reference to an access mode on an object with respect to a given time interval. In the last case, the revoke operation results in the deletion or modification of all the temporal authorizations of the revokee for the access mode on the object, granted by the user who revokes the access mode, to exclude the interval for which the access mode on the object is being revoked. This may cause the splitting of an authorization in more than one. Moreover, if the authorizations revoked or modified are with the grant option, the authorizations granted by the revokee may need to be reconsidered. We discuss the semantics of the

revocation in the next section. Note that, a user can revoke only authorizations and rules he granted.

- **Operations involving rules.** These are requests for specifying or deleting rules. The user invoking these operations must have either the `own` or the `administer` privilege on the object appearing at the left of the operator and either the `own`, `administer`, or `refer` privilege on the object appearing at the right of the operator.
- **Operations involving administrative privileges.** These are requests for granting or revoking administrative privileges on an object. They can be executed only by the owner of the object. Note that the revocation of the `administer` privilege on an object to a subject causes the deletion of all the authorizations on the object and all the derivation rules where the object appears in the authorization at the left of the operator, specified by the revokee. If the revokee does not have the reference privilege on the object, also the derivation rules where the object appears in the authorization at the right of the operator are deleted. Similarly, the revocation of the `refer` privilege on an object from a subject causes the deletion of all the derivation rules granted by the revokee where the object appears in the authorization at the right of the operator.

4 REVOCATION OF AUTHORIZATIONS

In the following we consider the case of the revocation of an access mode on an object for a given time interval. All the results shown for this case apply to the revocation with respect to specific authorizations.

Suppose a user revokes an access mode on an object for a given time interval from another user. The TAB resulting from the revoke operation has to be as if the revokee had never received by the revoker an authorization for the access mode on the object for the interval specified in the revoke request. More precisely, the semantics of the revocation of access mode `m` on object `o` from user `y` by user `x` in the interval $[t_1, t_2]$ is:

(i) to modify or revoke the authorizations that `x` had granted to `y` to exclude the interval $[t_1, t_2]$, and

(ii) to modify or revoke the authorizations in TAB to exclude from their time intervals the time instants in which they would not have existed if `x` had never granted to `y` an authorization for `m` on `o`, for the time instants eliminated by step *(i)*.

In the following we represent the sequence of grant operations for an access mode on an object by a labeled graph, where each node represents a user and an arc between node u_1 and u_2 indicates that user u_1 granted the access mode to u_2. Every arc is labeled with a 5-tuple (*id*, *timestamp*, *time*, *sign*, *grant-op*), where: *id* is the identifier of the authorization;[§] *timestamp* is the time when the authorization was granted; *time* is the time interval of the authorization; *sign* is the sign ('+','−') of the authorization; and *grant-op* indicates whether the authorization is with the grant option. We use symbol '*g*' to indicate that the authorization is with the grant option and nothing if the authorization

[§]We assume that each temporal authorization and each derivation rule in the TAB is identified by a unique identifier assigned by the system at the time of its insertion.

is without grant option. A node with no incoming arcs denotes the owner of the object or one of its administrators.

The revocation of authorizations may imply: deleting temporal authorizations, modifying the time interval associated with authorizations, or splitting temporal authorizations in several temporal authorizations (the last operation can be necessary when a subset of the instants associated with a temporal authorization needs to be excluded). The following example illustrates a case of revoke operation.

Example 2 Consider a TAB consisting of the following authorizations:
(A_1) (5, [50, 200], (Bob, o, read, Ann, +, yes))
(A_2) (55, [55, 180], (Chris, o, read, Bob, +, yes))
(A_3) (60, [60, 70], (David, o, read, Chris, −, no))
(A_4) (50, [80, 150], (Bob, read, o, Ellen, +, yes))

The corresponding graph is illustrated in Figure 2(a). Suppose that, at time 52, Ann issues the following command: REVOKE read ON o FROM Bob FROMTIME 60 TOTIME 200. According to the semantics of the revoke operation, the resulting effect on the TAB has to be as if Bob had never received the read privilege on o from Ann for the time interval [60,200]. The only authorization for the read privilege on o granted by Ann to Bob is authorization A_1. As required by Ann, this authorization is modified to exclude time interval [60,200]. Since the authorization is with the grant option, also the authorizations granted by Bob have to be reconsidered. In doing so, the fact that Bob has also other authorizations for the read privilege on o with the grant option (authorization A_4 granted by Ellen) must be considered. As a matter of fact, after A_1 is modified, Bob still remains with the privilege of granting authorizations for the read access mode on o but only for the intervals [50,59] (thanks to Ann) and [80,150] (thanks to Ellen). The authorizations granted by Bob must therefore be restricted to these intervals. Accordingly, authorization A_2 is modified to exclude time intervals [60,79] and [151,180]. This causes the splitting of the authorization in two authorizations, one for interval [55,59] and the other for the interval [80,150]. Since authorization A_2 was with the grant option, again the effect must be propagated and the time instants deleted from A_2 must be deleted from the authorizations Chris has granted. The only authorization granted by Chris is authorization A_3 whose interval must be completely excluded. As a consequence the authorization is deleted. The TAB' resulting after the revoke operation is illustrated in Figure 2(b).

Before formally introducing the semantics of revocation, we need some preliminary definitions on authorizations of TAB.

Definition 4 (Supporting authorization) *Let* A_1 *and* A_2 *be two authorizations. We say that* A_1 *supports* A_2 *at time* t, $t \in [t_i(A_2), t_j(A_2)]$, *(written* $A_1 \xrightarrow{t} A_2$*), iff:*

- A_1 *and* A_2 *are authorizations for the same access mode on the same object:* $m(A_1) = m(A_2)$, $o(A_1) = o(A_2)$;
- *the subject of* A_1 *is the grantor of* A_2: $s(A_1) = g(A_2)$;
- *the timestamp of* A_1 *is smaller than the timestamp of* A_2: $ts(A_1) < ts(A_2)$;
- *authorization* A_1 *is with the grant option:* $go(A_1) =$'yes';
- *time instant* t *belongs to the time interval of authorization* A_1: $t \in [t_i(A_1), t_j(A_1)]$.

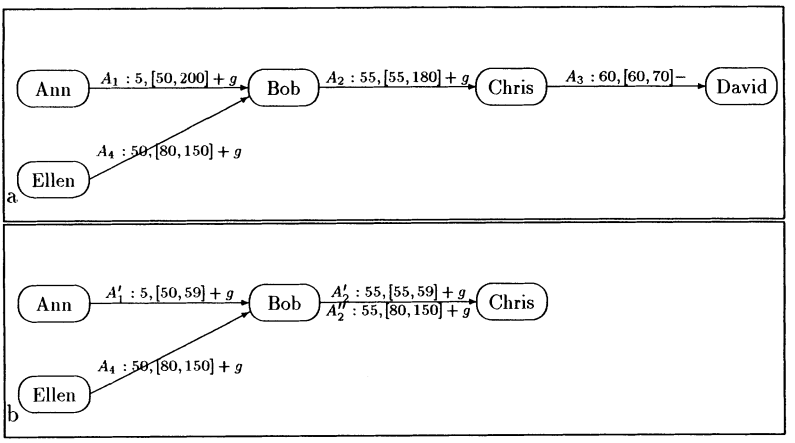

Figure 2 An example of revoke operation.

With reference to the TAB illustrated in Figure 2(a), $\forall t \in [55, 180]$ $A_1 \xrightarrow{t} A_2$, $\forall t \in [60, 70]$ $A_2 \xrightarrow{t} A_3$, and $\forall t \in [80, 150]$ $A_4 \xrightarrow{t} A_2$.

A sequence of authorizations each one supporting the next is called chain and is defined as follows.

Definition 5 (Supporting chain) *Assume that* $A_1, \ldots, A_n$ *are authorizations for access mode* m *on object* o. *We say that* $\langle A_1, \ldots, A_n \rangle^t$, $n \geq 1$, *is a supporting chain for* A_n *at time* t, *iff the grantor of* A_1 *has either the own or the administer privilege on object* o, *and* $A_1 \xrightarrow{t} A_2 \xrightarrow{t}, \ldots, \xrightarrow{t} A_n$.

Each authorization A for a privilege on an object must either be granted by the object's owner, by any of the object's administrators, or by a user who holds the authorization for the privilege on the object with the grant option for all time instants in $[t_i(A), t_j(A)]$. An authorization satisfying this requirement is said to be *legal.*

Definition 6 (Legal authorization) *An authorization* A *is legal in TAB, iff* $\forall t \in [t_i(A), t_j(A)]$, *there exists a supporting chain* $\langle A_1, \ldots, A_n, A \rangle^t$ *for* A, *with* $A_1, \ldots, A_n, A \in$ *TAB.*

Note that several supporting chains can be present in TAB to make a single authorization legal. To formalize the semantics of the revoke operation we use function '*Delete*()' that takes as argument two sets of authorizations S_1 and S_2. For each authorization A in S_2, the function checks if there exists an authorization A′ in S_1 having the same timestamp, subject, object, access mode, sign, grantor and grant option as A and whose time interval is not disjoint from that of A. If the time intervals of A and A′ coincide, then A′ is removed

from S_1. Otherwise, A′ is replaced by a set of authorizations which differ from A′ only for their time intervals, which are the elements of $\{[t_i(A'),t_j(A')]\backslash[t_i(A),t_j(A)]\}$.

In the following we use the notation ⟨x,m,o,y,t_1, t_2⟩ to denote a request by user x to revoke access mode m on object o from user y, from time t_1 to time t_2. We formalize the semantics of the revoke operation by a function named '*rvk*()', defined as follows.

Definition 7 (Rvk function) *Given a TAB containing only legal authorizations, let* ⟨x,m,o,y,t1,t2⟩ *be a request for revocation of access mode* m *on object* o. *Function rvk*() *generates a new temporal authorization base TAB′ defined as:*
TAB′ = Delete(TAB, (REV ∪ RREV)), where:
REV = {(ts,[t'_i,t'_j],auth)| ∃ (ts,[t_i,t_j],auth)∈*TAB, where subject,object, access mode, sign and grantor in* auth *are* y,o,m,'+',x, *respectively, and* [t'_i,t'_j] = ([t_i,t_j]∩[t1,t2])≠∅}.
RREV = {(ts,[t'_i,t'_j],auth)| ∃ A=(ts,[t_i,t_j],auth)∈ *TAB*, [t'_i,t'_j]⊆[t_i,t_j], [t'_i,t'_j]≠∅, *and* ∀t ∈[t'_i,t'_j] ∄ ⟨A_1,...,A_n,A⟩^{t}, *with* A_1,...,A_n,A ∈ *Delete(TAB, REV)*}.

REV denotes the set of authorizations whose revocation is explicitly required, whereas *RREV* (Recursive *REV*) denotes the set of authorizations that, after deleting the authorizations in *REV*, do not have a supporting chain for each instant in their time interval.

Example 3 Consider the TAB and the revoke operation of Example 2.
REV = {(5,[60,200],(Bob,o,read,+,Ann,yes))}.
RREV = {(55,[60,79],(Chris,o,read,+,Bob,yes)),
(55,[152,180],(Chris,o,read,+,Bob,yes)),(60,[60,70],(David,o,read,−,Chris,no))}.
TAB′ = {(5,[50,59],(Bob,o,read,+,Ann,yes)),(55,[55,59](Chris,o,read,+,Bob,yes)),
(55,[80,150],(Chris,o,read,+,Bob,yes)),(50,[80,150],(Bob,o,read,+,Ellen,yes))}.

An algorithm implementing function *rvk()* can be found in (Bertino et al. 1995).

5 CONCLUDING REMARKS AND FUTURE WORK

In this paper we have presented an authorization model which has a number of innovative features, including temporal authorizations and derivation rules. Moreover the model support both positive and negative authorizations for defining exceptions and explicit denials. In addition, our model combines all those features with decentralized authorization administration, thus resulting in a high degree of flexibility. Work that we are undertaking includes periodic authorization models, and additional derivation rules. Moreover, we are developing efficient strategies to perform authorization checking, based on materialization strategies.

REFERENCES

Bertino, E. Bettini, C. Ferrari, E. and Samarati, P. (1995) A decentralized temporal authorization model. Technical Report 148-95, DSI - University of Milano, Italy.

Bertino, E. Bettini, C. Ferrari, E. and Samarati, P. (1996) A temporal access control mechanism for database systems. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

Bobrowski, S. (1993) Safeguarding. *DBMS*, pages 44–52.

Castano, S. Fugini, M.G. Martella, G. and Samarati, P. (1995) *Database security.* Addison Wesley.

Fagin, R. (1976) On an authorization mechanism. *ACM Transactions on Database Systems*, **3**(6):310–9.

Griffiths, P.P. and Wade, B.W. (1976) An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, **1**(3):242–55.

Steiner, J.G. Neuman, C. and Schiller, J.I. (1988) Kerberos: An authentication service for open network systems. In *USENIX Conference Proceedings*, pages 191–202, Dallas, TX.

Thomas, R.K. and Sandhu, R.S. (1993) Discretionary access control in object-oriented databases: Issues and research directions. In *Proceedings 16th National Computer Security Conference*, pages 63–74, Baltimore, MD.

Woo, T.Y.C. and Lam, S.S. (1993) Authorizations in distributed systems: A new approach. *Journal of Computer Security*, **2**(2 & 3):107–36.

Elisa Bertino is professor of computer science in the Department of Computer Science of the University of Milan. She has also been on the faculty in the Department of Computer and Information Science of the University of Genova, Italy. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation in Austin, Texas, at George Mason University. Prof. Bertino is a co-author of the book "Object-Oriented Database Systems - Concepts and Architectures" 1993 (Addison-Wesley International Publ.) and is on the editorial board of the IEEE Transactions on Knowledge and Data Engineering.

Claudio Bettini received a MS degree in Information Sciences in 1987 and a PhD in Computer Science in 1993 from the University of Milan, Italy. He has been an assistant professor at the Computer Science Department of the University of Milan since 1993. His main research interest include temporal logics, description logics, temporal reasoning in knowledge and data bases, and temporal aspects of database security. On these topics he has published several papers. He has been a visiting researcher at IBM Kingston,NY and at George Mason University,VA.

Elena Ferrari received a MS degree in Information Sciences from the University of Milan, Italy, in 1992. Since November 1993, she has been a PhD student at the Department of Computer Science of the University of Milan. Her main research interests include authorization models, database security, and temporal data models.

Pierangela Samarati is an assistant professor of Computer Science at the University of Milan. Her main research interests are information systems security, database security, authorization models, and databases. On these topics she has published several papers. She has been a visiting researcher at Stanford University, CA and at George Mason University, VA. She is currently serving as the the italian representative in the IFIP TC-11 (Technical Committee 11 on Security and Protection in Information Processing Systems). She is co-author of the book "Database Security" (Addison-Wesley, 1995).