Proc. of 7th Annual Computer Security Applications Conference, San Antonio, TX, Dec 2-6 1991, pp 152-164.

A Distributed Implementation of the Extended Schematic Protection Model

Paul Ammann, Ravi S. Sandhu and Gurpreet S. Suri

Center for Secure Information Systems
and
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

Abstract

Protection models provide a formalism for specifying control over access to information and other resources in a multi-user computer system. One such model, the Extended Schematic Protection Model (ESPM), has expressive power equivalent to the monotonic access matrix model of Harrison, Ruzzo, and Ullman [7]. Yet ESPM retains tractable safety analysis for many cases of practical interest. Thus ESPM is a very general model, and it is of interest whether ESPM can be implemented in a reasonable manner. In this paper, we outline a distributed implementation for ESPM. Our implementation is capability-based, with an architecture where servers act as mediators to all subject and object access. Capabilities are made non-transferable by burying the identity of subjects in them, and unforgeable by using a public key encryption algorithm. Timestamps and public keys are used as mechanisms for revocation.

1 Introduction

In distributed systems, it has become vital to have some sort of access control to be able to safely share information and other resources on the network. Over a period of time many access control models have appeared in literature [3, 4, 7, 10], but unfortunately very few have been implemented in actual systems. These models provided a basis for specifying security policies in a multi-user environment.

Security models do not by themselves guarantee security. Systems specified in these models require safety analysis (of access rights) to determine how secure they actually are. Safety analysis issues were first formalized by Harrison, Ruzzo and Ullman (HRU) [7] in the context of the well known access matrix model. Upon analysis of this model it was discovered that the model suffered from a lack of a useful special case for which safety was decidable. In addition the assumptions from which undecidability follows are extremely weak. The Schematic Protection Model (SPM) [10] and various other models were developed in response to these weak safety properties inherent in the HRU model.

It has been shown that SPM has very strong expressive power [13] and at the same time allows for efficient safety analysis under very general conditions [10, 12]. SPM subsumes many other models not only in terms of its expressive power but also in terms of safety analysis [13]. ESPM is derived from SPM by extending the creation operation to allow multiple parents for a child, as opposed to the conventional create operation of SPM which has a single parent for a child [1]. This is the only difference between SPM and ESPM. It has been shown that ESPM is precisely equivalent to HRU's monotonic access matrix model in terms of expressive power and yet it retains SPM's efficient safety properties [2]. In this paper we have focused on providing a distributed capability-based implementation of ESPM.

Our implementation is strongly influenced by the architecture presented by Sandhu and Suri for implementation of the Transform model [14]. Propagation based on links and on the state of the subject, which consists of the capabilities it possesses in its domain, is incorporated in this architecture. Ours is a typical client-server architecture. Capabilities have been made non-transferable by embedding the identity of the user in them. Their freshness is maintained by means of a timestamp. Unforgeability is achieved with public key encryption. Two types of revocation protocols are provided in this implementation.

The paper is organized as follows. Section 2 contains a brief review of ESPM. In section 3 the gen-

eral architecture is discussed followed by details, along with its implementation, in section 4. An example of a security scheme in ESPM is presented in section 5. The paper concludes in section 6.

2 The Extended Schematic Protection Model

In this section we define ESPM. For convenience, we define ESPM directly, rather than first defining SPM and then defining ESPM as an extension. Our review of ESPM here is necessarily brief. Detailed motivation of the various components of the model is given in [1, 10].

ESPM is based on the key principle of protection types, henceforth abbreviated as types. ESPM subjects and objects are strongly typed, *i.e.*, the type of an entity (subject or object) is determined when the entity is created and does not change thereafter. Types are an abstraction of the intuitive notion of properties that are security relevant.

An ESPM scheme is to a large extent, but not exclusively, defined in terms of types. The dynamic privileges in ESPM are tickets of the form Y/r where Y identifies some unique entity and r is a right. The notion of type is extended to tickets by defining type(Y/r) to be the ordered pair type(Y)/r. That is, the type of a ticket is determined by the type of entity it addresses and the right symbol it carries.

ESPM has only two operations for changing the protection state, viz., create and copy. These operations are authorized by rules which are defined by specifying the following (finite) components.

1. Disjoint sets of subject types TS and object types TO. Let $T = TS \cup TO$.
2. A set of rights R. The set of ticket types is thereby $T \times R$.
3. A can-create function:
$$cc: TS \times TS \times ... \times TS \rightarrow 2^T$$
4. Create rules of the form:
$cr_{p_i}(u_1, u_2, ..., u_N, v) = c/R_1^i \cup p_i/R_2^i$ for i = 1..N
$cr_c(u_1, u_2, ..., u_N, v) = c/R_3 \cup p_1/R_4^1 \cup ... \cup p_N/R_4^N$
5. A collection of link predicates $\{link_i\}$.
6. A filter function $f_i : TS \times TS \rightarrow 2^{TXR}$ for each predicate $link_i$.

An ESPM scheme is itself static and does not change. We now explain how the scheme controls and regulates the propagation and creation of access rights.

2.1 The Create Operation

Creation is authorized exclusively by types. Subjects of type $u_1, u_2, ..., u_N$ can (jointly) create entities of type v if and only if $v \in cc(u_1, u_2, ..., u_N)$. N may take on any positive value once cc has been defined (for any given scheme this value is bounded). The case of $N = 1$ corresponds to the conventional creation operation in SPM. The case of $N > 1$ makes ESPM different from SPM by authorizing multiple parents to jointly and cooperatively create a child subject or object. Note that, if type constraints are met, we allow a subject to redundantly participate as more than one parent in a joint create operation.

Tickets introduced as the side effect of creation are specified by create-rules. In the create rules, c is the name of the jointly created entity and p_i is the name of the i^{th} parent. The sets R_1^i, R_2^i, R_3, and R_4^i, for $i = 1..N$ are subsets of R. When subjects $U_1, U_2, ..., U_N$ of type $u_1, u_2, ..., u_N$ create entity V of type v, the parent U_i gets the tickets V/R_1^i and U_i/R_2^i as specified by cr_{p_i}. The child V similarly gets the tickets V/R_3 and U_i/R_4^i for each parent U_i as specified by cr_c. As an example, consider the single parent creation case in which $file \in cc(user)$ authorizes *users* to create *files*; cr_p (*user, file*) = *c/r,w* and cr_c (*user, file*) = ϕ gives the creator r and w tickets for the created file. Note that the superscript i is used to specify a (potentially) different set for each of the N parents. Also note that the parents are not allowed to directly exchange tickets with other parents as a result of creation.

2.2 The Copy Operation

A copy of a ticket can be transferred from one subject to another leaving the original ticket intact. Permission to copy a ticket Y/r depends in part on possession of the ESPM copy flag, c, for that ticket, denoted Y/rc. Possession of Y/rc implies possession of Y/r but not vice versa. It is possible to copy Y/r only, or to copy Y/rc, in which case the ticket may be further copied. Let *dom(U)* signify the set of tickets possessed by U. Three independent pieces of authorization are required to copy Y/r from U to V.

1. Y/rc $\in$ *dom*(U), *i.e.*, U must possess Y/rc for copying either Y/rc or Y/r.
2. There is a link from U to V. Links are established by tickets for U and V in the domains of U and

V. The predicate $link_i(U, V)$ is defined as a conjunction or disjunction, but not negation, of one or more of the following terms for any $r \in R$: $U/\mathrm{r} \in dom(U)$, $U/\mathrm{r} \in dom(V)$, $V/\mathrm{r} \in dom(U)$, $V/\mathrm{r} \in dom(V)$, and $TRUE$. Some examples of link predicates from the literature are given below [8, 9, 13] respectively:

$$\begin{aligned} link_{tg}(U,V) &\equiv V/g \in dom(U) \vee U/t \in dom(V) \\ link_{t}(U,V) &\equiv U/t \in dom(V) \\ link_{sr}(U,V) &\equiv V/s \in dom(U) \wedge U/r \in dom(V) \\ link_{u}(U,V) &\equiv TRUE \end{aligned}$$

3. The final condition is defined by the filter functions f_i, one per predicate $link_i$. The value of $f_i(u, v)$ specifies types of tickets that may be copied from subjects of type u to subjects of type v over $link_i$. Also f_i determines whether or not the copied ticket can have the copy flag. Example values are $T \times R, TO \times R$ and ϕ respectively authorizing all tickets, object tickets and no tickets to be copied via a $link_i$.

In short Y/r can be copied from U to V if and only if there exists some $link_i$ such that:

$$Y/\mathrm{rc} \in dom(U) \wedge link_i(U, V) \wedge y/r \in f_i(u, v)$$

where the types of U, V and Y are respectively u, v and y. To copy Y/rc from U to V, it must also be the case that $y/\mathrm{rc} \in f_i(u, v)$.

2.3 Owner-Based Policy Example

In this section we present an example of a policy specified in ESPM. The example is provided to demonstrate the power of the ESPM framework to easily express a security policy. More detailed examples of ESPM are provided in [1, 10].

In an owner based policy a user is regarded as the owner of all files he* or she has created and has complete discretion regarding access to these files. Most operating system mechanisms are based on this concept. In this context a simple policy is that a user U can authorize another user U' to access file F if and only if U is the owner of F. The following scheme specifies this policy in ESPM.

1. $TS = \{user\}, TO = \{file\}$
2. $R = \{m{:}c\}$†
3. $link_u(\mathrm{X},\mathrm{Y}) \equiv$ **true**
4. $f_u(user, user) = \{file/\mathrm{m}\}$
5. $cc(user) = \{file\}$
6. $cr(user, file) = \{file/\mathrm{mc}\}$

The types *user* and *file* obviously correspond to users and files, respectively. For simplicity, a single right m:c provides access to files. This suffices so long as the policy regarding the dynamics of different rights, such as read, write execute, and append, remains the same. A universal link predicate is defined. Tickets for files, without the copy flag, can be copied across universal links. Owners get a copiable ticket for each created file.

3 The Architecture

We now describe a distributed architecture to implement ESPM. This is a typical client-server architecture, with no centralized authority. In this architecture, all rights are propagated through unforgeable capabilities and the propagation is mediated by the online servers.

Although revocation is not addressed in ESPM, there is a clear need for revocation in practical systems. This architecture supports revocation protocols, though they are of rather coarse granularity. Giving a notation to specify revocation is not difficult; however, analyzing the general safety properties of a model that allows revocation is. Thus a complete treatment of revocation is beyond the scope of this work.

Here we adopt a simple, if standard, approach to implement revocation. We allow some privileged subject, representing say, the security officer, to periodically cause the revocation of capabilities. Below we enumerate some properties that our revocation mechanisms will embody:

1. Revocation occurs quickly.
2. Revocation has minimal adverse effects on third parties.
3. Revocation can be undone.

Our approach is similar to Gong's [6], in that the identity of a capability's holder is recorded within the capability. Our approach extends the role of object servers described in the transform model implementation [14].

*It can be assumed he/she can be used interchangeably anywhere in this paper.

†The notation m:c denotes m or mc.

3.1 Object, Subject and Capability Servers

ESPM defines objects to be passive entities which do not permit objects to hold tickets. A typical example of an object is a file. As in [14], we encapsulate objects in an object server and rely on the object server to mediate all access to the object. The object server implements the response to messages such as "update record X", and ensures that the authorized update does occur and that no other action takes place. For purposes of revocation, object servers maintain a timestamp for each object they manage.

In ESPM subjects may be either passive or active. In either case, the subject may hold tickets within its domain. An example of a passive subject is a directory; an example of an active subject is a user process.

Since both passive and active subjects must respond to certain messages in a specific way, we extend the notion of an object server to subjects. For example, consider a directory that is requested to list the files and directories which it contains. The response clearly does not propagate access rights; nonetheless it is necessary to ensure that only the desired transmission occurs. For instance, a request to list directory entries should not change any entry. Enclosing the subject in a server limits changes to predefined operations and provides a mechanism for checking authorization prior to access. An example that involves an active subject is the response of a child process to an interrupt signal. If the sender of the interrupt holds specific capabilities, for instance if the sender is an ancestor of the child, then the operating system implements the resulting change in the child's control block.

For subjects in ESPM, we are also concerned about the propagation of access rights, and hence about the links that enable subjects to copy tickets to other subjects. To ensure that tickets are not copied unless the relevant link is enabled, we introduce the notion of a capability server and force every valid ticket to be authenticated by the capability server. In addition to controlling links, capability servers generate tickets during creation.

To summarize, we employ object and subject servers to mediate operations that do not change the protection state of the system. We employ capability servers to mediate operations that propagate access rights. Clearly the two types of servers must communicate, in that an object or subject server typically requires the exhibition of specific tickets before allowing an operation to proceed. Although both type of servers must be part of the trusted computing base, a given environment may demand more assurance, for example, for the propagation of access rights than for their use. Thus it is useful to separate object and subject servers from capability servers.

3.2 Raw Tickets

We implement a *raw ticket* with the following four components:

1. Each subject or an object has an identifier which consists of a unique name and a type. The ticket refers to the unique identifier of a subject or object.

2. A timestamp associated with the subject or object to which the ticket refers. Timestamps are useful for the implementation of revocation.

3. One or more right symbols (read (r), write (w) etc.) with or without the copy flag.

4. The identifier of the subject or object whose domain contains the ticket. The identifier again consists of a unique name and type.

According to the above scheme, we represent Y/r $\in dom(X)$ by the tuple (Y, ts, r, X), where Y is an object or subject identifier, ts is a timestamp for Y, r is one or more rights, and X is a subject identifier. The timestamp for an object (subject) Y is managed by the object (subject) server for Y. Raw (i.e., not cryptographically sealed) tickets can be generated at will and thus are not secure. The format of a raw ticket is shown below.

Y	ts	r	X

3.3 Capabilities

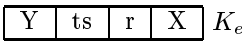
To ensure that a ticket cannot be forged, we cryptographically seal the ticket. We define a *capability* to be a sealed raw ticket. We set three requirements on the sealing process:

1. A ticket can only be sealed by the part of the TCB that is responsible for controlling the propagation of access rights, i.e., the capability server.

2. Only properly sealed tickets can be used by an object or subject server to authenticate access.

3. Only properly sealed tickets can be used by a capability server to propagate access rights.

We choose to use some standard public key based digital signature scheme, such as described in [5] to implement the seal. An authorized process uses a secret encryption key K_e to seal a ticket. A capability, c, is $c = E(t, K_e)$ where E is the public encryption function, t is a ticket of the form (Y, ts, r, X), and K_e is a particular encryption key. The capability formed is shown below.



The corresponding decryption key, K_d, is made public. Thus any holder of c may use the public decryption function D to recover the raw ticket by computing $t = D(c, K_d)$.

3.4 Basic Key Selection

For our basic implementation, we specify an encryption key for each subject. We denote the secret encryption key for subject U by K_e^U. We denote the corresponding public decryption key for U by K_d^U. Subject keys are *not* managed by the subject, but are managed by distributed (and trusted) capability servers.

The digital signature scheme described above only applies to part of the communication necessary in our implementation. In addition, we assume that mutually authenticated, secure channels between processes are supplied as fundamental primitives. Thus when two processes A and B communicate, A is assured that B and only B receives those messages sent by A, and B is assured that the received messages indeed originated from A. Many such schemes have been proposed in literature [5]. In our implementation, object and subject servers rely on these secure channels to respond properly to messages.

4 Implementation

This section describes the various protocols needed for an ESPM implementation. These implementation details are based on the architecture discussed in the previous section.

4.1 Object Access

When a subject U attempts to access an object V, U presents the appropriate capabilities to the object server for V. Such capabilities correspond to tickets of the form $V/\mathrm{r} \in dom(U)$. The object server decrypts the capabilities with the public decryption key K_d^U. For all tickets of the form V/r, the object server for V examines the timestamp in the ticket and compares it with the current timestamp for V. Since the current timestamp for V is maintained by the object server for V, this operation is straightforward. Tickets with out of date timestamps are no longer valid and so are rejected by the object server for V. The requested access to V is granted if the object server validates the necessary capability.

4.2 Subject Access

Subject access is similar to object access. If an operation that does not involve the propagation of access rights is performed on a subject, then the subject server mediates the access. For example, if a directory is requested to list its entries, the subject server for the directory authenticates and implements the access. The difference between object and subject access is that subjects can hold tickets in their domains, and so the subject server for a given subject S must check tickets within the domain of S in addition to tickets outside the domain of S.

4.3 Copy Operation

For propagation of access rights, we define *capability servers*. We associate each subject with a specific capability server. The capability server for a subject U maintains the secret encryption key K_e^U for subject U. Thus each secret encryption key only resides in a single location, and only the capability server for U can seal those tickets that reside in the domain of U. All decryption keys are public and can be broadcast to other capability servers.

We outline the steps by which link authentication takes place. Suppose that U attempts to copy $X/r : c$ to V.

1. U and V each send a sufficient set of capabilities for a given link to the capability server for V.[‡] The capability server for V decrypts each of these capabilities using the public key K_d^U for the capabilities sent by U and the public key K_d^V for the capabilities sent by V. The capability server uses the validated raw tickets to check the relevant link predicate.

2. U sends the capability corresponding to the ticket $X/\mathrm{rc} \in dom(U)$ to the capability server for V.[§]

[‡] Since the link predicate may be a disjunction, there may be more than one sufficient set of capabilities.

[§] Recall that to use the link to send $X/\mathrm{r}{:}c$, there must be a copy flag on the ticket; that is, X/rc must be in the domain of U.

The capability server for V decrypts the capability with the public key K_d^U and thus checks that indeed $X/\mathrm{rc} \in dom(U)$.

3. The capability server checks that $X/\mathrm{r{:}c}$ is allowed to be copied by the filter function for the link. Since the type of subject or object X is recorded in the identifier for X, the capability server is able to make this determination.

4. Additionally, for all tickets of the form V/r, the capability server examines the timestamp in the ticket and compares it with the current timestamp for V. The current timestamp for V is maintained by the subject server for V. To validate the timestamp for tickets for U, the capability server for V consults the latest value published by the subject server for U. (Note that, there may be a propagation delay in the case of revocation.) Tickets with out of date timestamps are no longer valid and so are rejected.

If these four conditions are satisfied, the capability server for V constructs a ticket for $X/\mathrm{r{:}c}$ in the domain of V and seals it with the secret key K_e^V. The resulting capability is sent to V.

The capability server uses the timestamp for X that was given in the capability for the ticket $X/\mathrm{r} \in dom(U)$. The timestamp for an object ticket X/r need not be checked since the object server for X will eventually authenticate the ticket. Similarly, the timestamp for a subject ticket X/r need not be checked since out of date tickets cannot be used to enable any links or authorize any access.

4.4 Creation

Here we concentrate on those aspects of creation that involve type and the proper distribution of creation tickets. We address single parent creation first and then generalize to multiple parent creation.

For single parent creation of objects, an object server first checks that the parent is of the correct type by consulting *cc*. If the parent is of the correct type, the object server creates a new object O and assigns it an initial timestamp. Note that no encryption or decryption keys for O are required. The object server informs the capability server responsible for the parent subject to construct the appropriate capabilities according to the creation rules. For multiple parent creation, the object server informs the capability server responsible for each parent to construct the appropriate capabilities.

For the creation of a subject U, similar rules are followed, except that a subject server takes the place of an object server. The subject server that is responsible for the new subject checks the types of the parents, performs the creation of the new subject, and informs the capability servers responsible for the parent subjects and the (new) child subject to generate capabilities determined by the create rules. It is necessary for the capability server for the child to generate a secret encryption key K_e^U and corresponding public decryption key K_d^U.

4.5 Revocation

We implement revocation in two ways:

1. *Alteration of Encryption and Decryption Keys*: Changing the keys for a specific subject invalidates all of the capabilities in the domain of that subject. Revocation for the domain of U is achieved by changing the keys K_e^U and k_d^U and generating new encryption keys $K_e^{'U}$ and $K_d^{'U}$. The new decryption key $K_d^{'U}$ is made public.

2. *Alteration of a Timestamp*: Changing the timestamp for a subject or a object X invalidates *all* tickets of the form X/r in the domain of any subject. If a capability for a ticket $Y/\mathrm{r} \in dom(X)$, with an out of date timestamp, is used to access object Y (or to link subject Y with some other subject), then the capability is rejected as invalid.

The two algorithms of key and timestamp alteration have the following properties.

1. *Timeliness*: In the case of key alteration, revocation is effective at a capability server as soon as the new public decryption key is received. In the case of timestamp alteration, revocation is effective as soon as the new timestamp is received. Thus there may be propagation delay in revocation. Even though out of date capabilities may be propagated through the system, they cannot be used to enable any link or to access any object, and so do not pose a security problem.

2. *Third party side effects*: In the case of altered keys, assume a third party V holds a capability for the ticket U/r in its domain. V is unaffected by the revocation of a ticket in the domain of U since this capability is encrypted with the key K_e^V. V may continue to use this capability as before. However, in the case of timestamp alteration for U, all third parties are immediately

affected, since capabilities for tickets of the form $U/\mathrm{r} \in dom(V)$ no longer enjoy any currency.

3. *Reversal of revocation*: In the case of key alteration, revocation reversal requires the capability server to generate a replacement capability with the current encryption key. In the case of timestamp alteration, revocation can be undone by arranging for a subject to receive a capability with a current timestamp. In either case, the revocation reversal can occur through continuing evolution according to the ESPM scheme, or by explicit grants from an external authority.

5 The Department Example

In this section, an implementation of ESPM is demonstrated with an example. As in section 2 a security scheme is expressed in ESPM notation and then, in addition, the protocols to implement it are described.

Let us take the example of a department whose subjects (users) frequently work on projects jointly with subjects outside their department. Access rights to outsiders for the internal documents of the department can only be authorized by the head of the department. Members of the department can freely share access rights for internal documents with other members. All the subjects that are to work on the project are created by the security-officer of the department. It is assumed that there exists a universal link between the security officer and the rest of the subjects.

The ESPM scheme for this policy is given below:

1. $TS = \{in,\ out,\ head,\ sec\text{-}off\}$, $TO = \{doc\}$
2. $R = \{\mathrm{r},\ \mathrm{rc},\ \mathrm{w},\ \mathrm{wc},\ \mathrm{t},\ \mathrm{tc}\}$
3. $link_u(\mathrm{X,Y}) \equiv \mathrm{true}$
   $link_t(\mathrm{X,Y}) \equiv \mathrm{Y/t} \in \mathrm{dom(X)}$
4. $f_u(in,\ in) = \{doc/\mathrm{r},\ \mathrm{w}\}$
   $f_u(in,\ head) = \{doc/\mathrm{r},\ \mathrm{w}\}$
   $f_u(sec\text{-}off,\ head) = \{in/\mathrm{t}\}$
   $f_t(in,\ head) = \{doc/\mathrm{rc},\ \mathrm{wc}\}$
   $f_u(head,\ out) = \{doc/\mathrm{r,w}\}$
   All other values of f are empty.
5. $cc(in) = \{doc\}$
   $cc(sec\text{-}off) = \{in,\ head,\ out\}$
6. $cr_p(in,\ doc) = p/\mathrm{rc},\ \mathrm{wc}$
   $cr_p(sec\text{-}off,\ in) = p/\mathrm{tc}$, $cr_c(sec\text{-}off,\ in) = \phi$
   $cr_p(sec\text{-}off,\ head) = \phi$, $cr_c(sec\text{-}off,\ head) = \phi$
   $cr_p(sec\text{-}off,\ out) = \phi$, $cr_c(sec\text{-}off,\ out) = \phi$

In this scheme a reasonable revocation policy could be which allows the department head to revoke access by outside users to internal documents, or to revoke all access to a given project.

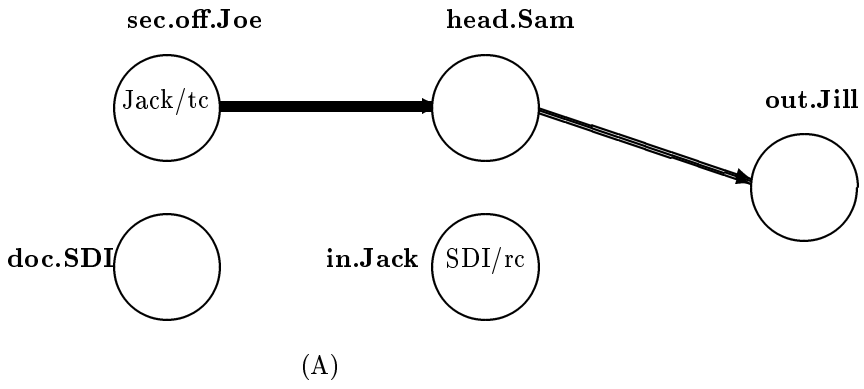
This scheme defines the types involved in a project to be *in* for subjects working inside the department, *out* for subjects working on the project from outside the department, *head* for the head of the department and *sec-off* for the security-officer respectively.

Users of type *in* can create objects of type *doc*. The creator gets the rc and wc tickets for the created document. The creator can then copy these tickets to other users of type *in* or the *head* using the universal link ($link_u$). The copied tickets are themselves without the copy flag so they cannot be further propagated.

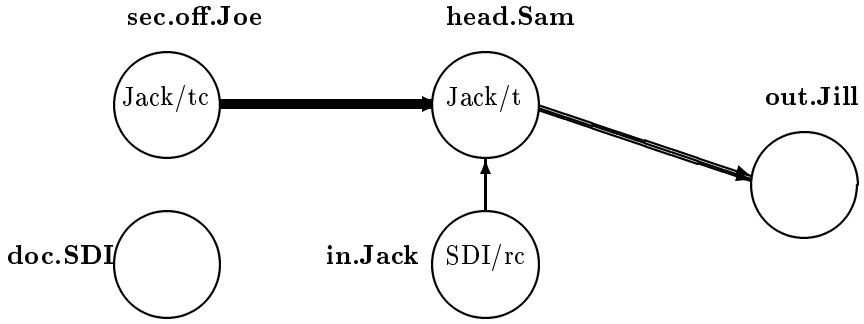
The security-officer creates all the subjects that are to work on the project as allowed by *cc*. By being the parent of type *in* he acquires the right *in*/tc as per the *cr* policy. The *head*, in order to pass the access rights to a user of type *out*, needs to acquire copiable rights for the *doc* from *in* (eg. *doc*/rc, wc) and he can only do this if he possesses the ticket *in*/t. The *in*/t ticket is copied from the *sec-off* to *head* over the universal link ($link_u$). With this ticket in its domain, *head* can copy the rights for *doc* from *in* over the resulting take take link, ($link_t$). This copy operation is regulated by the filter function f_t. In the next operation, the *head* copies the ticket *doc*/r,w to user *out* over the universal link, $link_u$. This copy operation is mediated by a filter function which only allows tickets of the type *doc*/r,w to be copied across it.

The above operations are illustrated in Figure 1. In this figure, subjects and objects are shown as circles. Tickets in the domain of a subject are shown inside the circle. The type and name of a given subject or object is shown next to the circle. Links are shown by arrows. Figure 1A shows the universal links (broad arrows) set up between the various subjects. It also shows Jack possessing the SDI/rc ticket for the SDI document and Joe in possession of the Jack/tc ticket for Jack. In figure 1B one can see the take link (narrow arrow) between Sam and Jack due to propagation of access right Jack/t from Joe to Sam. Figure 1C depicts the propagation of the SDI/rc ticket to Sam from Jack, followed by the copy operation of SDI/r ticket to Jill from Sam. These operations are carried out in our implementation as follows.

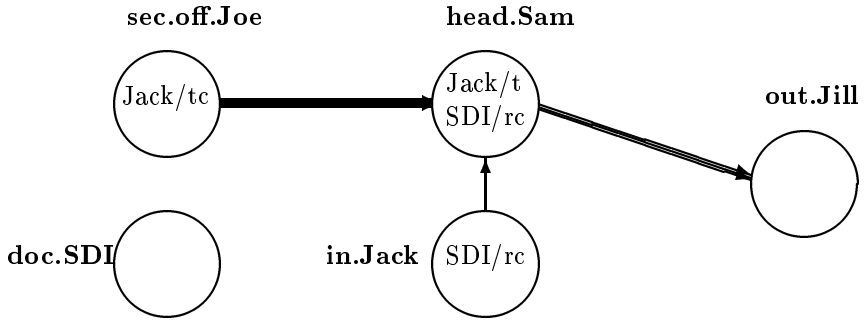
1. First the security-officer, Joe, creates the users that are going to work on a project. Let the inside user be Jack, the head be Sam and the outside user be Jill. For creation, Joe presents his request to the respective subject servers:



(A)



(B)



(C)

Figure 1: Pictorial Representation of the Example

create(Joe, in.Jack)
create(Joe, out.Jill)
create(Joe, head.Sam)

The servers check the *cc* policy against the types of subject involved and the nature of the request. In their internal tables, the respective subject servers store the following associations of the created subject and the associated time stamp:

in.Jack	0

head.Sam	0

out.Jill	0

Then the subject servers check the *cr* policy and inform their respective capability servers to service the request. The capability servers then service the request according to the *cr* rule. The capability servers create capabilities for the created subjects as specified by cr_c (in this case there are none, as none of the subjects get any rights on creation according to the *cr* policy). The capability servers generate the private and public keys for the created subjects, as follows.

K_e^{Jack} and K_d^{Jack} for Jack
K_e^{Jill} and K_d^{Jill} for Jill
K_e^{Sam} and K_d^{Sam} for Sam

The capability server of each child informs the capability server of the parent (i.e., Joe) to compute the capabilities according to cr_p policy. The capability server of Joe computes the following capabilities for Joe:

in.Jack	0	tc	sec-off.Joe	K_e^{Joe}

2. Now Jack, who is working on the project, wants to create a document, say SDI, of the type *doc* so he presents his request to the document server¶.

create(in.Jack, doc.SDI)

When the document server receives the request it proceeds to check the *cc* and the *cr* policy. Since the request conforms to the policies, the document server stores the following tuple in its internal tables:

doc.SDI	0

Then the document server informs Jack's capability server to compute the capabilities according to the *cr* policy with the time stamp 0. The capability server computes the following for Jack:

doc.SDI	0	rc, wc	in.Jack	K_e^{Jack}

Jack now has the ability to access the SDI document. To do so he just presents the capability above. The document server decrypts the capability with Jack's public key:

$($ doc.SDI | 0 | rc, wc | in.Jack $K_e^{Jack})K_d^{Jack}$

If decryption is correct, the document server has authenticated that the capability belongs to Jack, because only Jack's capability server possesses the corresponding private key (i.e., K_e^{Jack}). The document server checks the timestamp from the decrypted capability against the timestamp stored in its internal tables for SDI. If the values match, the validity of the capability is confirmed. Once the validity is confirmed Jack has access to SDI according to the rights present in the decrypted capability.

3. Now suppose Jill needs to work on SDI. She can't access the document as she possesses no access rights for it. According to the policy only the head can grant document access to outsiders, thus access rights for SDI can only be given to her by Sam. To do so Sam needs to copy the rights for SDI from Jack. To copy these rights he needs to have the in.Jack/t ticket. In order to obtain this authorization he requests Joe to initiate the following copy operation:

$copy_u$(sec-off.Joe, head.Sam, in.Jack/t)

Joe can use the universal link to copy this ticket to Sam, thus no link predicate needs to be verified. The capability server for Sam ensures that Joe really does hold *Jack*/tc. The capability server of Sam computes the following capability for Sam.

in.Jack	0	t	head.Sam	K_e^{Sam}

With this capability Sam can get the tickets *doc.SDI*/rc,wc from Jack by means of the copy operation:

$copy_t$(in.Jack, head.Sam, doc.SDI/rc,wc)

¶Note that the document server manages objects only of the type *doc*.

Sam's capability server decrypts the capability

in.Jack	0	t	head.Sam

K_e^{Sam}

with Sam's public key K_d^{Sam} to get:

in.Jack	0	t	head.Sam

The capability server verifies that Sam has the valid capability to set up the link. The capability server for Sam then decrypts the doc.SDI capability with Jack's public key K_d^{Jack} to get:

doc.SDI	0	rc, wc	in.Jack

The capability server checks if Jack has the read and write rights with the copy flag or not. It checks the filter function and, since the function allows the copy operation, the capability server computes a new capability for Sam with the same time stamp as that was in the capability presented by Jack. This new capability is computed as:

doc.SDI	0	rc, wc	head.Sam

K_e^{Sam}

With this capability the Sam can access SDI and also pass the access rights for SDI to Jill.

Note, however, the validity of the timestamp for SDI is not checked at this time. It will only be checked during an attempt to access SDI.

4. Jill gets access to SDI from Sam via the copy operation below:

   $copy_u$(head.Sam, out.Jill, SDI/r,w)

   Since this operation uses the universal link, Jill's capability server does not need to verify existence of the link. Jill's capability server validates the capability held by Sam using Sam's public key, as follows.

   (| doc.SDI | 0 | rc, wc | head.Sam | K_e^{Sam}) K_d^{Sam}

   Jill's capability server then computes the following capability,

   | doc.SDI | 0 | r, w | out.Jill | K_e^{Jill}

   and returns it to Jill. This capability then can then be presented by Jill to the document server, in order to access SDI.

   At the end of this operation the state of the subjects and the various servers is shown in Figure 2. In the Figure, subject Sam is represented by a rectangle within which the capabilities he possesses are shown. Jack and Jill are similarly depicted. Also shown are the capability servers for these subjects with the current encryption keys. Recall that the encryption key K_e is public while K_d is private, known only to the subject's capability server. Subject servers are shown with the subject they manage and the current time stamp. The object is shown as an empty rectangle, since an object cannot possess any capabilities. The document server of the object is shown with the current time stamp.

5. In this example, we have assumed it is the job of the head to revoke further access to the department's internal documents by outsiders once the project is finished. This implementation provides two types of revocation. In one, the timestamp is updated thus invalidating all previous capabilities associated with older timestamps. In the other, the encryption keys of the subject are changed thus invalidating all the capabilities in that subject's domain.

   Consider the situation in Figure 2 which shows the state of various components of the system before revocation is initiated. To see how each type of revocation works, let us suppose the project is complete and there is no further need for either Jack or Jill to have any access to SDI. The command

   revoke(doc.SDI)

   revokes all access to SDI, for all subjects.

   The document server updates the timestamp associated with SDI in its internal tables. The new association shown below is stored in its internal tables.

doc.SDI	1

   Now if either Jack or Jill present their capabilities to access SDI, the server matches the timestamps in the presented capabilities to the timestamp in its internal tables for SDI. When it finds them different, access to SDI is denied. The changes

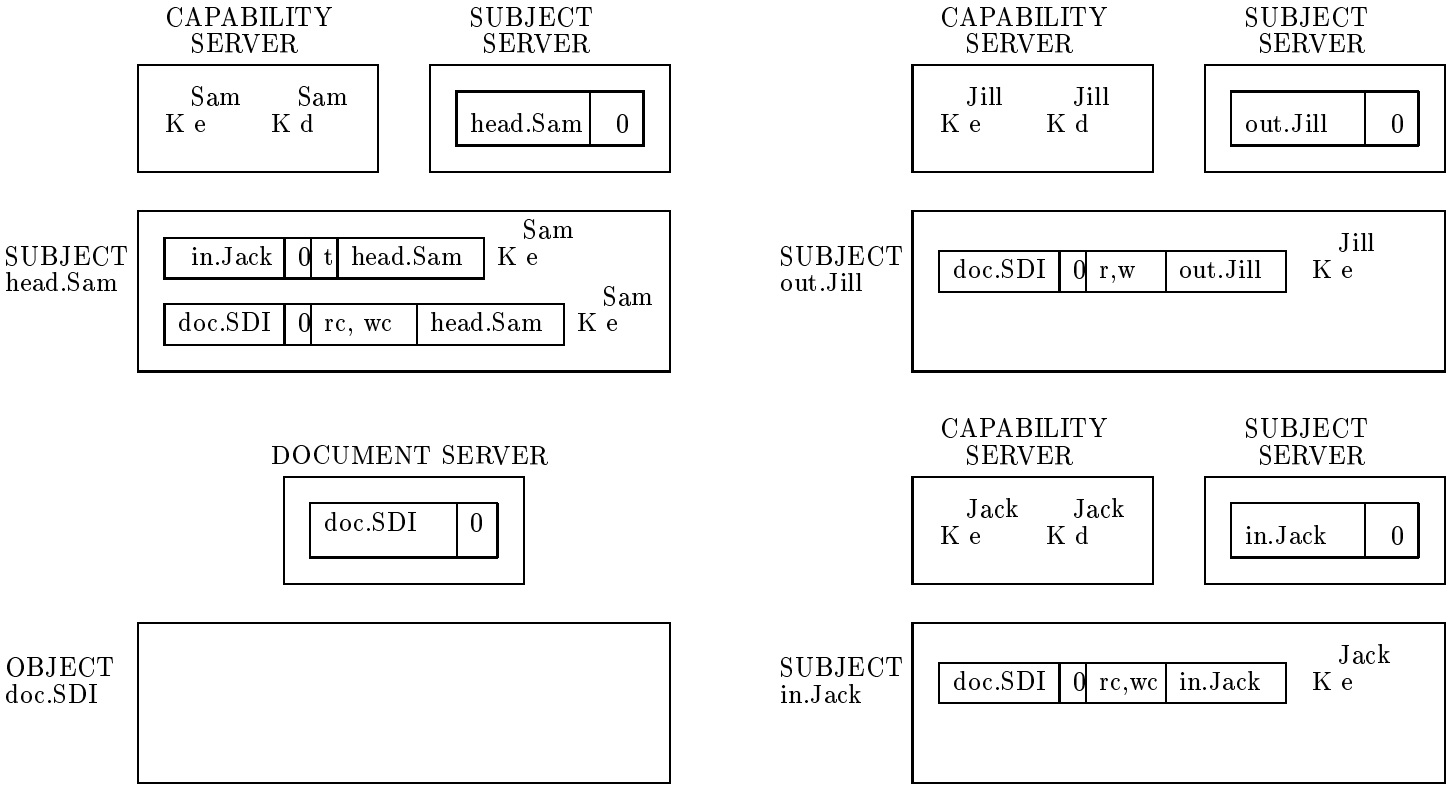


Figure 2: State of Subjects and Objects before Revocation

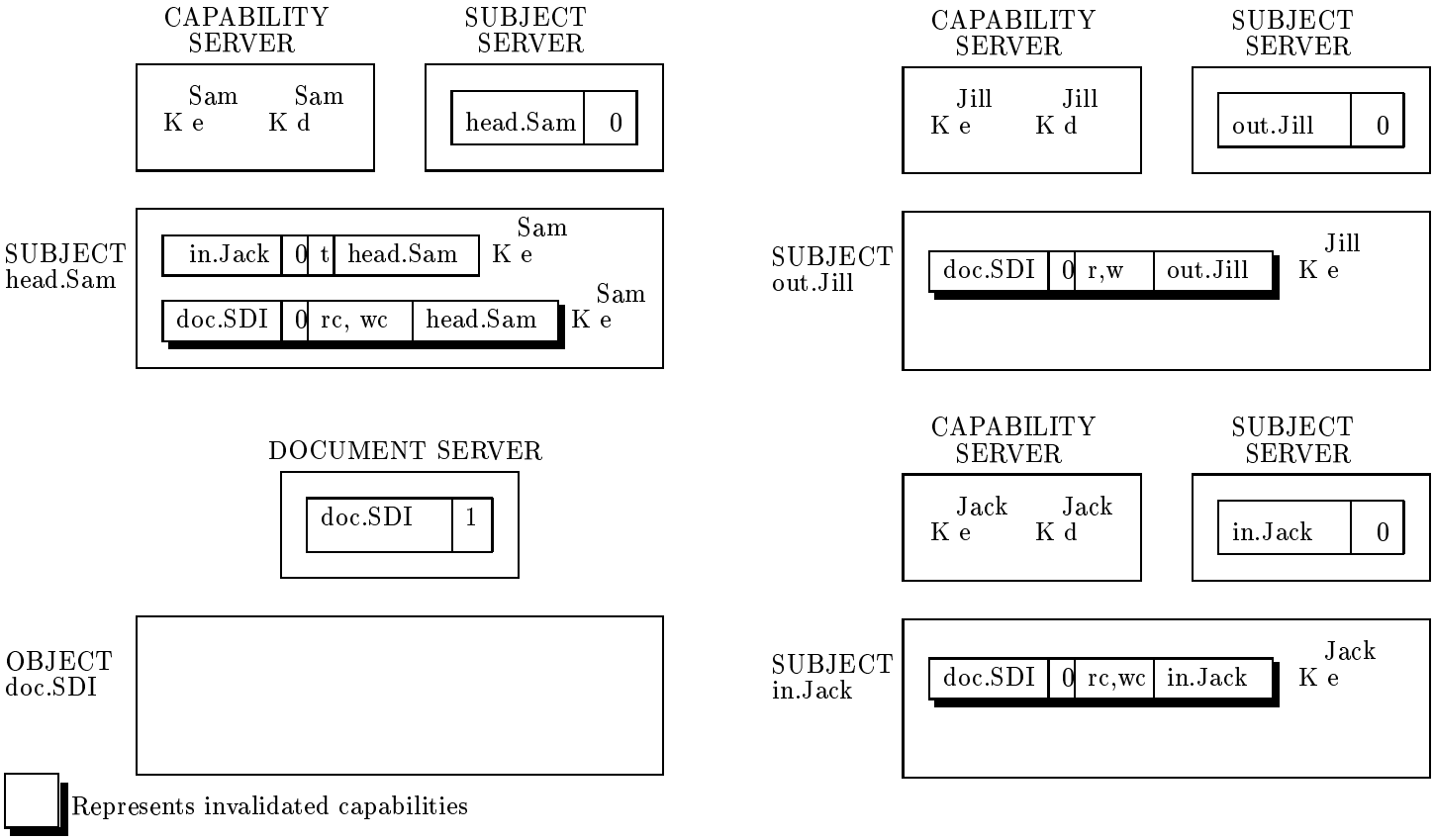


Figure 3: Revocation when the timestamp is changed

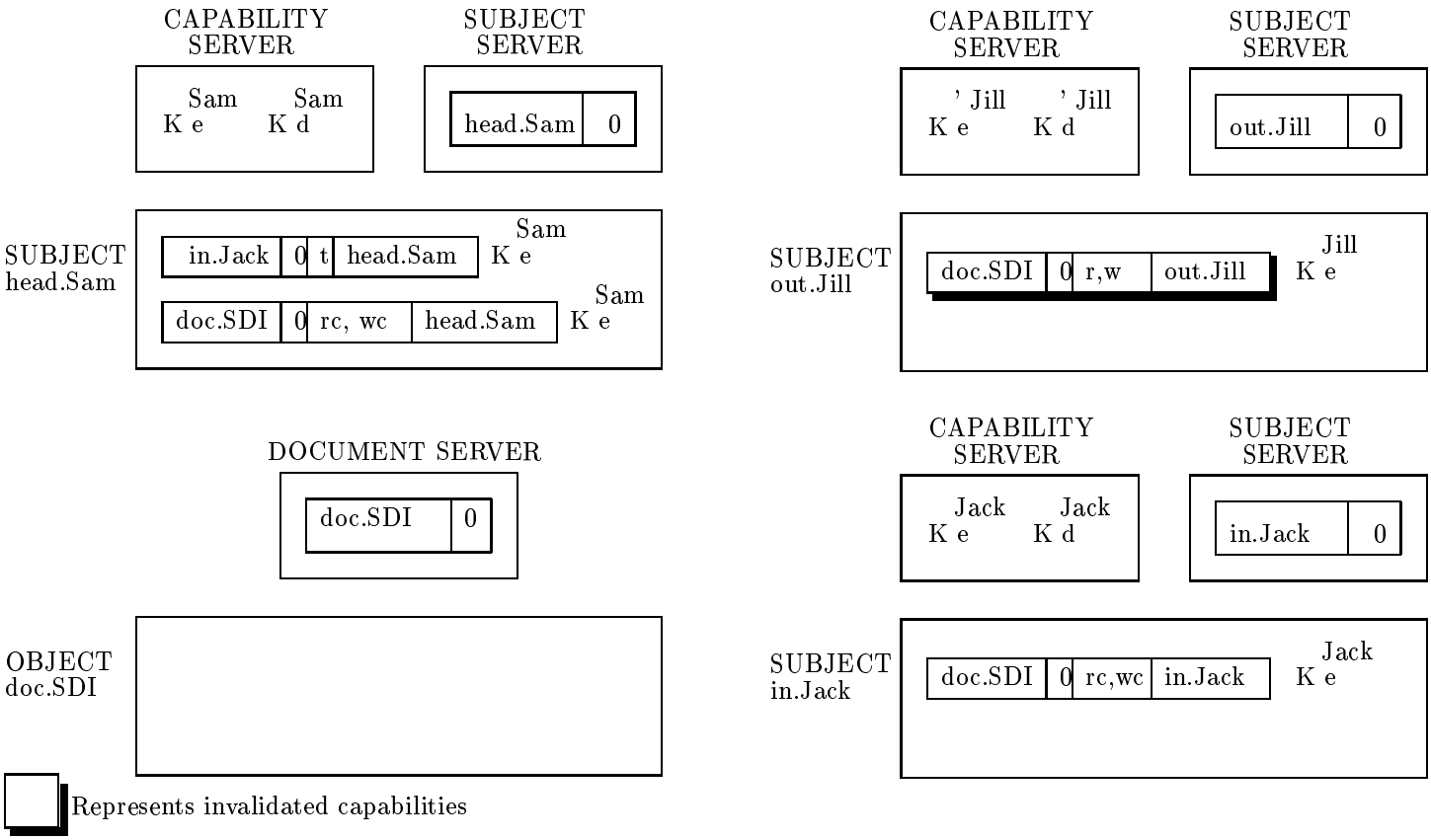


Figure 4: Revocation when Jill's keys are changed

that occur due to this revocation are shown in Figure 3 where the shaded capabilities are the ones which have been revoked.

To illustrate the second type of revocation, let's say the project is not yet complete but for some reason Jill is moved to a new project. Sam decides to revoke Jill's access rights. The command

revoke(out.Jill)

revokes *all* the capabilities held by Jill. The capability server changes the private and public keys for Jill to $K_e^{'Jill}$ and $K_d^{'Jill}$ respectively. The capability server broadcasts the new public key. If Jill tries to access SDI with a capability she possesses, the request will be turned down since the old capability will not decrypt successfully. The effect of this revocation is shown in Figure 4.

This completes the example.

6 Summary and Conclusions

To summarize, we have provided an architecture for a distributed implementation for the Extended Schematic Protection Model. The architecture is based on object and subject servers which act as mediators to objects and subjects respectively. Capability servers set up links and generate encryption keys. Capabilities are made non-transferable by embedding the identity of the user in them, and unforgeable by using a public key encryption algorithm. Two types of revocation have been provided by means of changing the encryption keys of a user, or by updating the timestamp associated with a subject or object.

Though revocation has not been addressed formally in ESPM, in our implementation we have defined it, with certain limitations. Revocation, as presented above, lacks granularity and can have undesirable third party effects. Some suggestions schemes for minimizing effect of revocation on third parties are presented below.

One way is to define encryption keys for each right in R, thus allowing certain capabilities to be revoked while others are unaffected. This technique specifies subsets of capabilities in the domain of a given subject U. As an example, the techniques allows the revocation of "w" access for a given subject U by revoking all capabilities of the form V/w in the domain of U. However, capabilities of the form V/r are unaffected.

Another scheme for minimizing effect of revocation on third parties is to define timestamps for each right in R, again allowing certain capabilities to be revoked while others are unaffected. For a given U, this technique specifies capabilities U/r in the domain of an arbitrary subject V. For example, "w" access for a given file U can be revoked for all subjects V without affecting "r" access for those subjects.

Lastly, we could also include revocation lists so as to explicitly deny access. Together these schemes can provide very fine grain revocation and are areas for future research.

Acknowledgment

We are indebted to Howard Stainer for his support and encouragement, in making this work possible.

References

[1] Ammann, P.E. and Sandhu, R.S. "Extending the Creation Operation in the Schematic Protection Model." *Proc. Sixth Annual Computer Security Applications Conference*, Tucson, Arizona, December 1990, pages 340-348.

[2] Ammann, P.E. and Sandhu, R.S. "Safety Analysis for the Extended Schematic Protection Model." *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1991, pages 87-97.

[3] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, Mitre, Bedford, Massachusetts (1975).

[4] Clark, D.D. and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies." *Proc. IEEE Symposium on Security and Privacy* Oakland, California, April 1987, pages 184-194.

[5] Davies, D.W. and Price, W.L. *Security in Computer Networks*. John Wiley & Sons (1989).

[6] Gong, L. "A Secure Identity-Based Capability System." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1989, pages 56-63.

[7] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D. "Protection in Operating Systems." *Communications of ACM* 19(8):461-471 (1976).

[8] Lipton, R.J. and Snyder, L. "A Linear Time Algorithm for Deciding Subject Security", *Journal of ACM*, 24(3):455-464 (1977).

[9] Lockman, A. and Minsky, N. "Unidirectional Transport of Rights and Take-Grant Control", *IEEE Transactions on Software Engineering*, SE-8(6):597-604 (1982).

[10] Sandhu, R.S. "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes." *Journal of ACM* 35(2):404-432 (1988).

[11] Sandhu, R.S. "Transformation of Access Rights." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1989, pages 259-268.

[12] Sandhu, R.S. "Undecidability of Safety for the Schematic Protection Model with Cyclic Creates." *Journal of Computer and System Sciences*, to appear.

[13] Sandhu, R.S. "Expressive Power of the Schematic Protection Model." *Proc. IEEE Computer Security Foundations Workshop I*, Franconia, New Hampshire, June 1988, pages 188-193.

[14] Sandhu, R.S. and Suri, G.S. "A Distributed Implementation of the Transform Model" *14th National Computer Security Conference*, Washington, DC, October 1991.