Proc. of the 15th NIST-NCSC National Computer Security Conference, Baltimore, MD, October 1992, pages 221-235

Implementation Considerations for the Typed Access Matrix Model in a Distributed Environment

Ravi S. Sandhu and Gurpreet S. Suri[1]

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

ABSTRACT The typed access matrix (TAM) model was recently defined by Sandhu. TAM combines the strong safety properties for propagation of access rights obtained in Sandhu's Schematic Protection Model, with the natural expressive power of Harrison, Ruzzo, and Ullman's model. In this paper we consider the implementation of TAM in a distributed environment. To this end we propose a simplified version of TAM called Single-Object TAM (SO-TAM). We illustrate the practical expressive power of SO-TAM by showing how the ORCON policy for originator control of documents can be specified in SO-TAM. We provide arguments to support our conjecture that SO-TAM is theoretically as expressive as TAM. We show that SO-TAM has a simple implementation in a typical client-server architecture. Our design is based on access control lists as the principal means for enforcing access to subjects and objects. In addition, certificate servers are introduced for generating certificates for checking access rights in those cases where access control lists are insufficient. A major advantage of our design is that atomicity of operations does not require a distributed commit.

Keywords: Access Matrix, Distributed Systems, Secure Architectures, Access Control Lists, Certificates

1 INTRODUCTION

Distributed systems have become the prevalent mode of computing. Modern systems offer a great deal of flexibility in tailoring a user's environment. The physical distribution of data and other resources can be made as transparent as a user wishes. It is important that security researchers and practitioners provide similar flexibility with respect to access control mechanisms.

To provide flexibility in access control we first need a flexible model which can express a rich variety of security policies. In our opinion flexibility is achieved by allowing users to propagate access rights to other users, with a combination of discretionary and mandatory controls. We would like to give individual users as much discretionary choice as possible, within the constraints required to meet the overall objectives and policies of an organization. For example, members of a project team might be allowed to freely share project documents with each other, but only the project leader is authorized to allow non-members to read project documents.

[1]The work of both authors is partially supported by National Science Foundation grant CCR-9202270 and National Security Agency contract MDA904-92-C-5141.

Security models based on propagation of access rights must confront the safety problem. In its most basic form, the safety question for access control asks: is there a reachable state in which a particular subject possesses a particular right for a specific object? There is an essential conflict between the expressive power of an access control model and tractability of safety analysis. The access matrix model as formalized by Harrison, Ruzzo, and Ullman (HRU) [5] has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties.

Recently Sandhu [9] has shown how to overcome the negative safety results of HRU by introducing strong typing into the access matrix model. The resulting model is called the Typed Access Matrix (TAM). TAM combines the positive safety results for the Schematic Protection Model [6] with the natural expressive power of HRU.

The safety problem is closely related to the so-called fundamental flaw of discretionary access control (DAC). DAC is vulnerable to Trojan Horses, in part because Trojan Horse laden programs can surreptitiously modify the protection state without explicit instruction from the users. However, even Trojan Horses are constrained by the authorization for propagating access rights. The Trojan Horse vulnerability of DAC does require that we assume the worst case regarding propagation of access rights in a system. What we need therefore is a model, such as TAM, with strong safety properties and broad expressive power.

In addition to balancing expressive power versus safety analysis, a useful model must also be implementable. Our focus in this paper is on implementation considerations for TAM. It is possible to implement TAM as defined in its full generality. However, such a full-blown implementation would be cumbersome and awkward at best. In this paper we identify a simplified version of TAM called Single-Object TAM (SO-TAM). SO-TAM is particularly suited for implementation in a distributed environment. Moreover it retains most, if not all, of the expressive power of TAM. We provide theoretical arguments to support this claim. We also demonstrate how SO-TAM can enforce the ORCON policy for originator control of documents.

The rest of this paper is organized as follows. Section 2 provides a brief review of the TAM model, following which SO-TAM is defined in Section 3. Section 4 expresses the ORCON policy in SO-TAM. This is achieved by taking the ORCON solution of TAM [9], and manipulating it to fit the requirements of SO-TAM. The basic architecture for implementing SO-TAM is discussed in Section 5. Implementation and protocol details of SO-TAM are covered in Section 6. In Section 7 it is then shown how the ORCON example relates to the implementation. Section 8 gives our conclusions.

2 THE TYPED ACCESS MATRIX MODEL

In this section we briefly review the typed access matrix (TAM) model. In a nutshell, TAM is obtained by incorporating strong typing into the model of Harrison, Ruzzo and Ullman [5]. The principal innovation of TAM is to introduce strong typing of subjects and objects. This innovation is adapted from Sandhu's Schematic Protection Model [6].

As one would expect from its name, TAM represents the distribution of rights in the system by an access matrix. The matrix has a row and a column for each subject and a column for each object. Subjects are also considered to be objects. The $[X, Y]$ cell contains rights which subject X possesses for object Y.

Each subject or object is created to be of a specific type, which thereafter cannot be changed. It is important to understand that the types and rights are specified as part of the system definition, and are not predefined in the model. The security administrator specifies the following sets for this purpose:

- a finite set of access *rights* denoted by R, and

- a finite set of *object types* (or simply *types*) denoted by T.

Once these sets are specified they remain fixed (until the security administrator[2] changes their definition). For example, $T = \{user, so, file\}$ specifies there are three types, viz., user, security-officer and file. A typical example of rights would be $R = \{r, w, e, o\}$ respectively denoting read, write, execute and own.

The *protection state* (or simply *state*) of a TAM system is given by the four-tuple (OBJ, SUB, t, AM) interpreted as follows:

- OBJ is the set of *objects*.
- SUB is the set of *subjects*, $SUB \subseteq OBJ$.
- $t : OBJ \rightarrow T$, is the *type function* which gives the type of every object.
- AM is the *access matrix*, with a row for every subject and a column for every object. The contents of the $[S, O]$ cell of AM are denoted by $AM[S, O]$. We have $AM[S, O] \subseteq R$.

The rights in the access matrix cells serve two purposes. First, presence of a right, such as $r \in AM[X, Y]$ may authorize X to perform, say, the read operation on Y. Second, presence of a right, say $o \in AM[X, Y]$ may authorize X to perform some operation which changes the access matrix, e.g., by entering r in $AM[Z, Y]$. In other words, X as the owner of Y can change the matrix so that Z can read Y.

The protection state of the system is changed by means of TAM commands. The security administrator defines a finite set of TAM commands when the system is specified. Each TAM *command* has the following format:

command $\alpha(X_1 : t_1,\ X_2 : t_2,\ \ldots,\ X_k : t_k)$
  if $r_1 \in [X_{s_1}, X_{o_1}] \wedge r_2 \in [X_{s_2}, X_{o_2}] \wedge \ldots \wedge r_m \in [X_{s_m}, X_{o_m}]$
  then $op_1;\ op_2;\ \ldots;\ op_n$
end

or

command $\alpha(X_1 : t_1,\ X_2 : t_2,\ \ldots,\ X_k : t_k)$
  $op_1;\ op_2;\ \ldots;\ op_n$
end

Here α is the *name* of the command; $X_1,\ X_2,\ \ldots,\ X_k$ are *formal parameters* whose types are respectively $t_1,\ t_2,\ \ldots,\ t_k$; $r_1,\ r_2,\ \ldots,\ r_m$ are rights; and $s_1,\ s_2,\ \ldots,\ s_m$ and $o_1,\ o_2,\ \ldots,\ o_m$ are integers between 1 and k. Each op_i is one of the *primitive operations* discussed below. The predicate following the **if** part of the command is called the *condition* of α, and the sequence of operations $op_1;\ op_2;\ \ldots;\ op_n$ is called the *body* of α. If the condition is omitted the command is said to be an *unconditional command*, otherwise it is said to be a *conditional command*.

A TAM command is invoked by substituting actual parameters of the appropriate types for the formal parameters. The condition part of the command is evaluated with respect to its actual parameters. The body is executed only if the condition evaluates to true.

There are six primitive operations in TAM as follows.

[2] It should be kept in mind that TAM treats the security administrator as an external entity, rather than as another subject in the system.

enter r **into** $[X_s, X_o]$	**delete** r **from** $[X_s, X_o]$
create subject X_s **of type** t_s	**destroy subject** X_s
create object X_o **of type** t_o	**destroy object** X_o
(a) Monotonic Primitive Operations	(b) Non-Monotonic Primitive Operations

We require that s and o are integers between 1 and k, where k is the number of parameters in the TAM command in whose body the primitive operation occurs.

The **enter** operation enters a right $r \in R$ into an existing cell of the access matrix. The contents of the cell are treated as a set for this purpose, i.e., if the right is already present the cell is not changed. The **enter** operation is said to be *monotonic* because it only adds and does not remove from the access matrix. The **delete** operation has the opposite effect of **enter**. It (possibly) removes a right from a cell of the access matrix. Since each cell is treated as a set, **delete** has no effect if the deleted right does not already exist in the cell. Because **delete** (potentially) removes a right from the access matrix it is said to a *non-monotonic* operation.

The **create subject** and **destroy subject** operations make up a similar monotonic versus non-monotonic pair. The **create subject** operation requires that the subject being created does not previously exist. The **destroy subject** operation similarly requires that the subject being destroyed should exist. Note that if the pre-condition for any **create** or **destroy** operation in the body is false, the entire TAM command has no effect. The **create subject** operation introduces an empty row and column for the newly created subject into the access matrix. The **destroy subject** operation removes the row and column for the destroyed subject from the access matrix. The **create object** and **destroy object** operations are much like their subject counterparts, except that they work on a column-only basis.

To summarize, a system in specified in TAM by defining the following.

1. A set of rights R.
2. A set of types T.
3. A set of state-changing commands.
4. The initial state.

We say that the rights, types and commands define the system *scheme*. Note that once the system scheme is specified by the security administrator it remains fixed thereafter for the life of the system. The system state, however, changes with time.

3 SINGLE-OBJECT TAM

In this section we present a simplified version of TAM called Single-Object TAM (SO-TAM). Our principal motivation in defining SO-TAM is to arrive at a model well-suited to a distributed implementation. We, of course, do not wish to lose or compromise the expressive power of TAM in doing so. We conjecture that SO-TAM is theoretically equivalent to TAM. Arguments in support of this conjecture are given at the end of this section. The natural expressive power of SO-TAM is demonstrated in the next section, where we show how the ORCON policy for originator control of documents is specified in SO-TAM.

The principal restriction in SO-TAM is that all primitive operations in the body of a command are required to operate on a single object. An object is represented as a column in the access matrix. Similarly, when a subject is the "object" of an operation, that subject is viewed as a column in the access matrix. SO-TAM stipulates that all operations in the body of a command are confined to a single column.

Now consider the usual implementation of the access matrix by means of access control lists (ACL's). Each object has an ACL associated with it, representing the information in the column corresponding to that object in the access matrix. The restriction of SO-TAM implies that a single command can modify the ACL of exactly one object. These modifications can therefore be done at the single site where the object resides. This greatly simplifies the protocols for implementing the commands. In particular, we do not need to be concerned about coordinating the completion of a single command at multiple sites. There is therefore no need for a distributed two-phase commit for SO-TAM commands.

Commands in SO-TAM are further categorized into the following two classes, depending upon the single or multi-object nature of the condition part of the command.

- **Class I**: In these commands the condition part is also single object, i.e., the tests are confined to the ACL of a single object. Unconditional SO-TAM commands also fall into this class. An example of a **Class I** command is given below.

  command $\alpha(S_1 : t_1, O : t_2, S_3 : t_3)$
      if x $\in [S_1, O]$ **then**
      enter z **into** $[S_2, O]$
  end

- **Class II**: In these commands the condition part is multi-object, i.e., the tests require reference to the ACL of more than one object. An example of a **Class II** command is given below.

  command $\alpha(S_1 : t_1, O : t_2, S_3 : t_3)$
      if x $\in [S_1, O] \wedge$ y $\in [S_1, S_3]$ **then**
      enter z **into** $[S_1, O]$
  end

In **Class I** commands the condition and body of the command reference the ACL of a single object. These commands can therefore be executed completely at the site where this single object resides. In **Class II** commands evaluation of the condition part requires reference to the ACL's of several objects. In general these objects can be located at different sites. Various pieces of the condition will need to evaluated at different sites and then combined together. **Class II** commands therefore require a more complex protocol than **Class I** commands. Implementation of **Class I** and **Class II** commands is discussed in section 6.

Now let us consider the expressive power of SO-TAM. SO-TAM with **Class I** commands alone is quite expressive by itself. In particular it subsumes the various transform models of [7, 10, 11]. SO-TAM with **Class II** has very strong expressive powers. As is shown in the next section it can express the ORCON policy. Moreover SO-TAM can easily model the Extended Schematic Protection Model (ESPM) [1, 2]. SO-TAM therefore inherits the theoretical expressive power of ESPM, which is equivalence to the Harrison, Russo and Ullman (HRU) model [5] for the monotonic case (i.e., no **delete** or **destroy** primitive operations). We conjecture that this equivalence of SO-TAM and HRU will also extend to the non-monotonic case. Formal consideration of this matter is beyond the scope of this paper. SO-TAM also inherits the practical expressive power of ESPM demonstrated in [1, 8]. It should be noted that the expressive power of SO-TAM is obtained without compromise on safety analysis.

4 ORCON IN SO-TAM

In this section we demonstrate the expressive power of SO-TAM by specifying an ORCON (originator control) policy [4]. In doing so we also show how multi-object TAM commands can be reduced to

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			own, read, write	
$S_2 : s$				
...				

(a) Subject S_1 creates an ORCON object O

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			own, read, write	
$S_2 : s$			cread	
...				

(b) S_1 gives S_2 the cread (confined-read) right for O

	$S_1 : s$	$S_2 : s$	$O : co$	$S_3 : cs$	...
$S_1 : s$			own, read, write		
$S_2 : s$			cread		
$S_3 : cs$			read		
...					

(c) S_2, jointly with O, creates the confined subject S_3 to read O

Figure 1: Illustration of the ORCON Policy with multi-object TAM operations

single object operations. Specifically we first review the ORCON solution given in [9]. This solution uses multi-object TAM commands. We then show how to construct equivalent SO-TAM commands.

ORCON requires that the creator (i.e., originator) of a document retains control over granting access to the information in the document. For example, let Tom be the creator of an ORCON document[3] called SDI. Suppose Tom authorizes Dick to read SDI. The ORCON policy requires that Dick cannot propagate the information in SDI to, say, Harry; either directly by granting Harry read access to SDI, or indirectly by granting Harry read access to a copy of SDI. The prohibition that Dick cannot directly grant read access to Harry is straightforward to enforce. The real challenge for the ORCON policy is how to prevent Dick from copying the information from SDI into some other document, say, SDI-Copy and authorizing Harry to read SDI-Copy.[4]

The ORCON solution given in [9] is based on the ability in TAM to have multiple parents jointly create a child subject.[5] Figure 1(a) shows a fragment of the access matrix in which subject S_1 is the creator (and therefore owner) of object O as indicated by own $\in [S_1, O]$. The notation $S_1 : s$ denotes that S_1 is of type s, and similarly for the names on the other rows and columns. The type of O is co for confined object. In Figure 1(b), S_1 gives S_2 the cread (i.e., confined-read) right for O. This right allows S_2 to create jointly with O subject S_3 of type cs (for confined-subject). This creation results in S_3 getting the child right for S_2 and O. By virtue of being the child of S_2 and

[3] An ORCON document is one to which the ORCON policy applies as opposed to, say, ordinary documents to which ORCON does not apply.

[4] Note that Dick as a human being is trusted not to divulge information from SDI to Harry without concurrence of Tom. The problem here is to ensure that Trojan Horse laden subjects executing on behalf of Dick do not surreptitiously leak the information in SDI to Harry.

[5] The solution prohibits subjects spawned by Dick from making copies (or extracts) of SDI. The solution can be extended to allow this with the stipulation that the copies (or extracts) will themselves be originator controlled by Tom.

O and S_2 possessing the cread right, S_3 obtains the read right for O. This results in the situation shown in Figure 1(c). The scheme will ensure that S_3, by virtue of its type being cs, will never be able to write to any object or create any objects.

The definition of the TAM scheme for this ORCON solution is given below.

1. Rights $R = \{\text{own}, \text{read}, \text{write}, \text{cread}\}$
2. Types $T = \{s, cs, co\}$
3. The following TAM commands

   (a) **command** create−orcon−object($S_1 : s, O : co$)
       create object O **of type** co;
       enter {own, read, write} **in** $[S_1, O]$[6]
   end

   (b) **command** grant−confined−read($S_1 : s, S_2 : s, O : co$)
       if own $\in [S_1, O]$ **then enter** cread **in** $[S_2, O]$
   end

   (c) **command** use−confined−read($S_2 : s, O : co, S_3 : cs$)
       if cread $\in [S_2, O]$ **then create subject** S_3 **of type** cs;
           enter read **in** $[S_3, O]$
   end

   (d) **command** destroy−orcon−object($S_1 : s, O : co$)
       if own $\in [S_1, O]$ **then destroy object** O
   end

   (e) **command** revoke−confined−read($S_1 : s, O : co, S_2 : s$)
       if own $\in [S_1, O]$ **then delete** cread **from** $[S_2, O]$
   end

   (f) **command** revoke−read($S_1 : s, O : co, S_3 : cs$)
       if own $\in [S_1, O] \wedge$ read $\in [S_3, O]$ **then destroy subject** S_3
   end

   (g) **command** finish−orcon−read($S_2 : s, O : co, S_3 : cs$)
       if cread $\in [S_2, O] \wedge$ read $\in [S_3, O]$ **then destroy subject** S_3
   end

Use of the first three commands is illustrated in Figure 1. The remaining commands are for revocation of rights and destruction of objects and subjects.

This scheme is not an SO-TAM scheme, because of command (c) which has multi-object operations. In command (c) subject S_3 has to be created and the ACL of object O has to be modified. In general, this requires the command to execute at two sites contrary to the constraints of SO-TAM. All commands other than (c) are actually **Class I** commands, i.e., single-object condition and operations.[7]

This scheme can be easily converted to SO-TAM. We do this by introducing a parent right. Command (c) is replaced by the following two commands.

---

[6] Strictly speaking this should be written as three separate **enter** operations, one for each of the three rights being entered.

[7] One might question how **destroy subject** is a single site operation, since it requires removal of a row from the access matrix potentially affecting a large number of ACL's. However, we don't need to purge these ACL's immediately in an atomic manner.

(c.1) **command** create–confined–subject($S_2 : s, O : co, S_3 : cs$)
create subject S_3 **of type** cs;
enter parent **in** $[S_2, S_3]$;
enter parent **in** $[O, S_3]$;
end

(c.2) **command** get–read($S_2 : s, O : co, S_3 : cs$)
if cread $\in [S_2, O] \wedge$ parent $\in [S_2, S_3] \wedge$ parent $\in [O, S_3]$
then enter read **in** $[S_3, O]$
end

Figure 2 shows how the scenario of Figure 1 plays out with this modification. In the modified scheme we enter the parent privilege during joint creation by command (c.1). Prior to grant of the read privilege to S_3 the condition in command (c.2) tests for the presence of the parent right. This simple manipulation makes the entire scheme an SO-TAM scheme with single-object operations. Note that command (c.1) is a **Class I** command while command (c.2) is a **Class II** command.

5 THE ARCHITECTURE

In this section we describe a client-server based architecture for implementing SO-TAM. This architecture has evolved from our earlier work [2, 10, 11].

5.1 Global Identifiers

Every subject and object is assigned a type when it gets created. The typing is strong and cannot be altered thereafter. Moreover each subject or object in the system has a globally unique identifier i.e., no two subjects or objects in a system can have the same identifiers. We assume the type of a subject or object is embedded in its identifier. These identifiers have the following structure.

type	identifier

The type field denotes the type of the subject or the object. The identifier field uniquely identifies each subject or object among instances of the same type. Uniqueness of object identifiers reduces to requiring each object to have a unique identifier among instances of the same type. If a particular type is managed by more than one server, uniqueness of the identifier can be ensured by having the following structure.

type	server identifier	identifier

Having made this point, we will use the former global identifier structure in rest of this paper.

5.2 Access Control Lists

Each object in the system is managed by an object server. When the object is a subject, we sometimes call the server a subject server. Each server manages a particular type of object, but the same type of object may be managed by several servers. For example, there may be several file servers in the system. Each object resides at exactly one server.

Each object has an Access Control List (ACL) associated with it. The ACL has the following structure.

	$S_1 : s$	$S_2 : s$	$O : co$	$\ldots$
$S_1 : s$			own, read, write	
$S_2 : s$				
$O : co$				
$\ldots$				

(a) Subject S_1 creates an ORCON object O

	$S_1 : s$	$S_2 : s$	$O : co$	$\ldots$
$S_1 : s$			own, read, write	
$S_2 : s$			cread	
$O : co$				
$\ldots$				

(b) S_1 gives S_2 the cread (confined-read) right for O

	$S_1 : s$	$S_2 : s$	$O : co$	$S_3 : cs$	$\ldots$
$S_1 : s$			own, read, write		
$S_2 : s$			cread	parent	
$O : co$				parent	
$S_3 : cs$					
$\ldots$					

(c) S_2, jointly with O, creates the confined subject S_3

	$S_1 : s$	$S_2 : s$	$O : co$	$S_3 : cs$	$\ldots$
$S_1 : s$			own, read, write		
$S_2 : s$			cread	parent	
$O : co$				parent	
$S_3 : cs$			read		
$\ldots$					

(d) S_3 acquires read right for O

Figure 2: Illustration of the ORCON Policy in SO-TAM

	sid1	rights
	sid2	rights
oid	.	.
	.	.
	sidn	rights

To make the construction of the architecture clear we refer to a subject identifier by *sid* and a object identifier by *oid*.

Any access to an object is determined by the rights specified in the ACL for that subject. Similarly all accesses to subjects are dictated by the rights in the ACL possessed by the requesting subject. The ACL's are dynamic in nature and can be manipulated by SO-TAM commands.

5.3 Certificates

In addition each server is associated with a certificate server. The certificate server acts as a mediator for any form of communication between two servers. The certificate server is responsible for creating, encrypting and decrypting certificates for the servers to which it is associated. The certificate generated by a certificate server has the following structure.

oid	Rights	sid

The oid contains the unique identifier for the object in question. The sid is the unique identifier of the subject. The rights field specifies the rights that the subject identified in the sid field has for the object in the oid field.

Since these certificates travel over insecure lines they are made secure by using a public key based encryption algorithm. For this we specify a pair of keys for each server. Out of this pair one of the keys is secret known only to that server's certificate server, while the other one is public and known to all certificate servers. Certificates are doubly encrypted in the usual manner in public-key systems, to ensure their authenticity and confidentiality. They are also time-stamped to avoid replay attacks. Further details are given in the next section. Authentication between users and their servers is assumed. Any authentication protocol from the literature [3] can be employed for this purpose.

6 IMPLEMENTATION OF SO-TAM

The implementation of SO-TAM commands is based on the architecture described in the previous section. All accesses to an object are mediated by the object server responsible for managing that object. Similarly for subject accesses the subject server responsible for that subject mediates the access.

Authentication is also carried out at the time of object/subject access, and must be incorporated into the RPC (Remote Procedure Call) mechanism of the client-server architecture. The servers must authenticate the source of every RPC request. This can be achieved by any of the encryption protocols found in literature [3]. One method would be to provide means for every subject to place its digital signature on every RPC communication to a server. Digital signatures for the reverse communication from object/subject servers to clients can also be incorporated.

We now describe the execution of a primitive operation at a server, followed by protocols for **Class I** and **Class II** commands.

6.1 Primitive Operations

Let us consider each of the primitive operations in turn.

1. **enter** x **into** $[S_1, O]$

   In this operation the server managing object O enters the x right for subject S_1 into the ACL for object O.

2. **delete** x **from** $[S_1, O]$

   For this operation the server managing object O deletes the x right that S_1 has from O's ACL. This operation is exactly the opposite of the **enter** operation.

3. **create subject** S_1 **of type** t_1

   The server who will manage subject S_1 creates S_1 with an empty ACL.

4. **destroy subject** S_1

   The server managing S_1 destroys the subject S_1 and discards S_1's ACL.

5. **create object** O **of type** t_2

   The server who will manage object O creates object O with an empty ACL.

6. **destroy object** O

   The server managing O destroys the object O and discards O's ACL.

6.2 Class I Commands

For an unconditional command the server in question simply executes the primitive operations in the body as indicated above. The operations of conditional **Class I** commands are executed only if the specified condition is satisfied. In **Class I** commands the **if** condition can be tested by the server who manages the object in question, simply by reference to the object's ACL.

A typical command with single-object condition verification is shown below.

command $\alpha(S_1 : t_1, O : t_2)$
  if x $\in [S_1, O]$
  then $op_1; op_2; \ldots; op_n$
end

This command is sent to the server where the listed operations $op_1; op_2; \ldots; op_n$ are to be executed. The command is executed as follows.

1. (a) The server on receiving the request verifies the types of the subjects and objects against the security policy to check the validity of the **command**. Once the validity is confirmed the server tests the **if** conditional statement. If the command fails the validity tests the request is aborted.

   (b) The server checks the ACL for object O to see if S_1 really possesses the x privilege for O and if so it executes the next step, otherwise the request is aborted.

   (c) If the **if** condition is true the server performs the operations op_1; op_2; ...; op_n.

6.3 Class II Commands

Verification of the condition in **Class II** commands requires reference to multiple sites. Our protocol for multi-object condition verification is based on inter-server communications. Various pieces of the condition as verified at individual servers and communicated to server A as certificates. Each server has an associated certificate server to generate the certificate.

Consider the following typical example of multi-object verification of a conditional command.

command $\alpha(S_1 : t_1, O : t_2, S_3 : t_3)$
if x $\in [S_1, O] \wedge$ y $\in [S_1, S_3]$
then $op_1; op_2; \ldots; op_n$
end

As specified by the constraints of SO-TAM all the operations $op_1; op_2; \ldots; op_n$ involve only one server. Let us say this is the server for object O and is called server A. In the above command verification of the condition part involves only one additional site, viz., the site of S_3's server. Let us call S_3's server as server B. The protocol is easily extended to additional sites.

In this command, to verify the conditional **if** statement, server A needs information from the subject server managing subject S_3 as to whether or not S_1 possesses the y right for S_3. This is achieved as follows.

1. (a) Server A checks the security policy to determine the validity of the request. If the validity tests fail the request is aborted.
   (b) Server A checks O's ACL to see whether S_1 possesses the x right for O. If S_1 does indeed possess x for O the command proceeds, otherwise it is aborted.
   (c) Server A further needs information from the subject server managing subject S_3 as to whether or not S_1 has the y right for S_3, so A waits for a certificate from S_3's server. (To prevent A from waiting indefinitely for the certificate to arrive, it waits for a specified period of time and then aborts the command.)
2. (a) Server B, i.e., S_3's server, checks into S_3's ACL to ascertain whether S_1 possesses the y right for S_3. If this is so, B informs B's certificate server to create a certificate and send it to server A. Otherwise server A is notified of a failed condition.
   (b) B's certificate server encrypts the certificate with its own secret key. Then the certificate is again encrypted with the public key of the A's certificate server. The certificate is shown below.

$$(\boxed{S_3 : t_3 \mid \mathrm{y} \mid S_1 : t_1 \mid TS}\ K_d^B)K_e^A$$

   where K_d^B is the secret key of B (known only to B's certificate server), K_e^A is the public encryption key of A (known to all certificate servers) and TS is a timestamp.
3. (a) When A's certificate server receives the certificate it decodes it in two steps. First it applies A's secret key K_d^A, and it applies B's public key K_e^B. If decryption fails or the timestamp is out of date the request is aborted.
   (b) If the certificate is decoded correctly the information it holds is in the clear and server A has the necessary verification it needs to process the **command** request.
   (c) If the condition is met server A executes the requested operations op_1; op_2; $\ldots$; op_n.

7 IMPLEMENTATION OF ORCON

In this section we give a concrete example of the abstract implementation of section 6 by showing how the ORCON policy of section 4 is enforced.

1. Let Tom be a subject of type s who initiates the following command to create the ORCON object SDI of type co.

command create$-$orcon$-$object$(Tom : s, SDI : co)$

The kernel of Tom's host, makes a remote procedure call (RPC) to the object server which is responsible for managing ORCON objects created by Tom. This RPC contains the action requested, the sid and oid; all signed under Tom's digital signature. In this instance, the sid = s.Tom and the oid = co.O.

2. On receiving the request the object server authenticates the request originating from Tom. The server then checks the **command** create−orcon−object with respect to its actual parameters to determine its validity. Once the command is determined to be valid, the object server proceeds to create a new ORCON object SDI with the ACL shown below.

   co.SDI

s.Tom	own,read,write

   The ACL shows Tom to be the owner of the document SDI and possessing own, read and write privileges for it.

3. In this step Tom grants cread (confined read) privilege to Dick (sid = s.Dick). The command is sent to SDI's object server. The request is shown below.

   $$\textbf{command}\ \text{grant−confined−read}(Tom : s, Dick : s, SDI : co)$$

   The server on receiving the RPC authenticates its origin as Tom. Then it performs the validity checks on the request by checking the sids and oids of the subjects and objects involved in the operation. The server then evaluates the condition part of the command. The server looks into SDI's ACL to see if Tom is the owner of SDI. With this fact confirmed, the **if** condition evaluates to true and the server enters cread privilege for Dick into the ACL, as shown below.

   co.SDI

s.Tom	own,read,write
s.Dick	cread

   With this Dick possesses the cread privilege for the confined-object SDI.

4. Now Dick and the object SDI jointly create a new subject Dick′ which is of the type confined-subject (cs). The command shown below is sent to the appropriate subject server.

   $$\textbf{command}\ \text{create−confined−subject}(Dick : s, SDI : co, Dick' : cs)$$

   The subject server on receiving the request authenticates the sender and tests the sids and oids of the subjects and objects involved to determine the validity of the request. Since this is an unconditional command, the subject server proceeds to create a new subject Dick′ with the ACL shown below.

   cs.Dick′

s.Dick	parent
co.SDI	parent

5. Next the read right is obtained by Dick′ via the following command. This command is sent to the object server managing SDI.

   $$\textbf{command}\ \text{get−read}(Dick : s, SDI : co, Dick' : cs)$$

   Like before the object server makes the authentication and validity tests. Then it checks into its ACL to determine whether Dick possesses the cread privilege for SDI. This information completes one part of the **if** statement. For the other part it relies on information from the subject server managing Dick′.

6. The subject server for Dick′ checks into its ACL to determine whether Dick and SDI are parents of Dick′. Since this is the case, the server informs its certificate server which frames two certificates, shown below, to be sent to SDI's object server.

$$(\begin{array}{|c|c|c|c|}\hline \text{s.Dick} & \text{parent} & \text{cs.Dick}' & \text{TS} \\ \hline \end{array}\ K_d^B)K_e^A$$

$$(\begin{array}{|c|c|c|c|}\hline \text{co.SDI} & \text{parent} & \text{cs.Dick}' & \text{TS} \\ \hline \end{array}\ K_d^B)K_e^A$$

where the K_d^B is the secret key of the subject server for Dick′ and K_e^A is the public encryption key of the object server for SDI. Recall that TS is a timestamp.

7. The certificate server for SDI's object server first applies its secret key K_d^A and then the public key K_e^B of the certificate server for the subject server. Now the certificates are in the clear as shown below.

$$\begin{array}{|c|c|c|}\hline \text{s.Dick} & \text{parent} & \text{cs.Dick}' \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|}\hline \text{co.SDI} & \text{parent} & \text{cs.Dick}' \\ \hline \end{array}$$

Now SDI's object server has complete information to evaluate the condition part of the command. Since the condition evaluates to be true, the server updates the ACL by adding the read right for Dick′ for the ORCON object SDI.

$$\text{co.SDI}\ \begin{array}{|l|l|}\hline \text{s.Tom} & \text{own,read,write} \\ \text{cs.Dick}' & \text{read} \\ \hline \end{array}$$

Now Dick′ can read SDI but cannot copy it or pass it to another subject (due to Dick′ being a confined subject).

8. Now suppose Tom wants to revoke the read access to Dick′. To do this he issues the following command.

$$\textbf{command}\ \text{revoke--read}(Tom : s, SDI : co, Dick' : cs)$$

The object server for SDI authenticates the command and performs the regular validity tests on the command. With validity of the command confirmed the server checks SDI's ACL to see whether Tom is the owner of SDI and whether Dick′ has the read privilege for it. Since this is true, the server deletes the read privilege for Dick′ for SDI. The purged ACL is shown below.

$$\text{co.SDI}\ \begin{array}{|l|l|}\hline \text{s.Tom} & \text{own,read,write} \\ \text{cs.Dick}' & \\ \hline \end{array}$$

Since the read privilege is deleted from the ACL all future accesses by Dick′ to read SDI are denied.

This completes the example.

8 CONCLUSION

In this paper we have considered implementation of the Typed Access Matrix (TAM) model, recently defined by Sandhu [9]. TAM has rich expressive power and yet has strong safety properties. We have defined a simplified version of TAM called Single-Object TAM (SO-TAM). We have shown that SO-TAM has a particularly simple and efficient implementation in a distributed environment. This paper demonstrates how the ORCON policy can be expressed in SO-TAM and implemented in the architecture. We conjecture that SO-TAM has the same expressive power as TAM. Theoretical arguments in support of this conjecture have been provided.

The implementation is based on an architecture which makes use of both access control lists and certificates. All accesses to subjects and objects are mediated by subject and object servers respectively. Access control lists are used for this purpose. Each server in addition has a certificate server under its domain. The certificate server has the function of creating and decrypting certificates used for communications between servers over a potentially hostile network.

References

[1] Ammann, P.E. and Sandhu, R.S. "The Extended Schematic Protection Model." *Journal of Computer Security*, to appear.

[2] Ammann, P.E., Sandhu, R.S. and Suri, G.S. "A Distributed Implementation of the Extended Schematic Protection Model." *Seventh Annual Computer Security Applications Conference*, 1991, pages 152-164.

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

[4] Director of Central Intelligence Directive No. 1/7 "Control of Dissemination of Intelligence Information," 4 May 1981.

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

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

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

[8] Sandhu, R.S. "Expressive Power of the The Schematic Protection Model." *Journal of Computer Security*, Volume 1, Number 1, 1992, pages 59-98.

[9] Sandhu, R.S. "The Typed Access Matrix Model" *IEEE Symposium on Research in Security and Privacy*, Oakland, CA. 1992, pages 122-136.

[10] Sandhu, R.S. and Suri, G.S. "A Distributed Implementation of the Transform Model" *14th National Computer Security Conference*, Washington, DC, October 1991, pages 177-187.

[11] Sandhu, R.S. and Suri, G.S. "Non-Monotonic Transformation of Access Rights" *IEEE Symposium on Research in Security and Privacy*, Oakland, CA. 1992, pages 148-161.