*Proc. Ninth Annual Computer Security Applications Conference, Orlando, Florida, December 6-10, 1993, pages 184-194*

# Expressive Power of the Single-Object Typed Access Matrix Model

*Ravi S. Sandhu and Srinivas Ganta*[1]
[6pt]Center for Secure Information Systems
&
Department of Information and
Software Systems Engineering
George Mason University
Fairfax, VA 22030-4444

**ABSTRACT**

The single-object typed access matrix (SOTAM) model was recently introduced in the literature by Sandhu and Suri. It is a special case of Sandhu's typed access matrix (TAM) model. In SOTAM individual commands are restricted to modifying exactly one column of the access matrix (whereas individual TAM commands in general can modify multiple columns). Sandhu and Suri have outlined a simple implementation of SOTAM in a distributed environment using the familiar client-server architecture. In particular the stipulation that each command modifies a single column of the access matrix, is reflected in the desirable property that each command modifies a single access control list corresponding to that column. In this paper we show that TAM and SOTAM are formally equivalent in their expressive power. This result establishes that SOTAM has precisely the same expressive power as TAM, while having a simple implementation at the same time. In a nutshell, this result tells us that manipulation of access control information can be achieved in its most general form by manipulation of a single access control list (ACL) at a time.

[1]The work of both authors is partially supported by National Science Foundation grant CCR-9202270. Ravi Sandhu is also supported by the National Security Agency through contract MDA904-92-C-5141. We are grateful to Dan Atkinson, Nathaniel Macon, Howard Stainer, and Mike Ware for their support and encouragement in making this work possible.

# 1 Introduction

The need for access controls arises in any computer system that provides for controlled sharing of information and other resources among multiple users. Access control models (also called protection models or security models) provide a formalism and framework for specifying, analyzing and implementing security policies in multi-user systems. These models are typically defined in terms of the well-known abstractions of subjects, objects and access rights with which we assume the reader is familiar.

Access controls are useful to the extent they meet the user community's needs. They need to be flexible so that individual users can specify access of other users to the objects they control. At the same time the discretionary power of individual users must be constrained to meet the overall objectives and policies of an organization. One method for achieving the desired flexibility is to allow security administrators to specify policies for propagation rights, which allow some discretionary freedom to users but at the same time impose some non-discretionary rules. Several such policies, and access control models for their specification, have been published in the literature (see for example [1, 5, 6, 7, 9, 11]).

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) [3] has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties.

Recently Sandhu [8] 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 [5] with the natural expressive power of HRU. Although further work on the safety problem—particularly for non-monotonic cases—remains to be done, it is clear from existing results that TAM offers tractable safety analysis in many cases of practical interest.

This brings us to the feasibility of implementing TAM. An implementation of TAM in its complete generality would be very cumbersome and awkward at best. Sandhu and Suri [10] were therefore motivated to define a simplified version of TAM called single-object TAM (SOTAM). They outlined a very simple implementation for SOTAM in a distributed environment using the familiar client-server architecture.

The principal contribution of this paper is to demonstrate that TAM and SOTAM are formally equivalent in terms of expressive power. This result establishes the fact that SOTAM has the most general expressive power possible, while having a simple implementation at the same time.

The rest of the paper is organized as follows. Section 2 gives a brief review of

TAM and SOTAM. Section 3 proves equivalence of the expressive power of TAM and SOTAM. The proof is in two steps. First we show equivalence without creation and destruction of subjects and objects, followed by equivalence with such creation and destruction. Section 4 gives our conclusions.

# 2 The TAM and SOTAM Models

In this section we review the definition of TAM, which was introduced by Sandhu in [8]. Our review is necessarily brief. The motivation for developing TAM, and its relation to other access control models are discussed at length in [8]. Following the review of TAM we briefly review the definition of SOTAM [10].

## 2.1 The Typed Access Matrix (TAM) Model

The principal innovation of TAM is to introduce strong typing of subjects and objects, into the access matrix model of Harrison, Ruzzo and Ullman [3]. This innovation is adapted from Sandhu's Schematic Protection Model [5], and its extension by Ammann and Sandhu [1].

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 $T$.

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. Once these sets are specified they remain fixed, until the security administrator changes their definition. It should be kept in mind that TAM treats the security administrator as an external entity, rather than as another subject in the system.

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 one of the following formats.

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

or

$$
\begin{array}{l}
\textbf{command } \alpha(X_1 : t_1,\, X_2 : t_2,\, \ldots,\, X_k : t_k) \\
\quad op_1;\ op_2;\ \ldots;\ op_n \\
\textbf{end}
\end{array}
$$

Here $\alpha$ 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 $\alpha$, and the sequence of operations $op_1; op_2; \ldots; op_n$ is called the *body* of $\alpha$. 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, grouped into two classes, as follows.

**enter** $r$ **into** $[X_s, X_o]$
**create subject** $X_s$ **of type** $t_s$
**create object** $X_o$ **of type** $t_o$

(a) Monotonic Primitive Operations

**delete** $r$ **from** $[X_s, X_o]$
**destroy subject** $X_s$
**destroy object** $X_o$

(b) Non-Monotonic Primitive Operations

It is required 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 has a unique identity different not only from existing subjects, but also different from all subjects that have ever existed thus far.[1] The **destroy subject** operation requires that the subject being destroyed currently exists. 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.

Two examples of TAM commands are given below.

- **command** *create-file*$(U : user, F : file)$
  **create object** $F$ **of type** $file$

[1] There is some question about whether or not creation should be treated as a monotonic operation. The fact that creation consumes a unique identifier for the created entity, which cannot be used for any other entity thereafter, gives it a non-monotonic aspect. In our work we have always treated creation as a monotonic operation. This is principally because systems without creation are not very interesting. Treating creation as non-monotonic would therefore make the class of monotonic systems uninteresting. Monotonic systems with creation are, however, an important and useful class of systems.

**enter** *own* **in** $[U, F]$
**end**

- **command** *transfer-ownership*$(U : user, V : user, F : file)$

  **if** $own \in [U, F]$ **then**
  **delete** *own* **from** $[U, F]$
  **enter** *own* **in** $[V, F]$
  **end**

The first command authorizes users to create files, with the creator becoming the owner of the file. The second command allows ownership of a file to be transferred from one user to another.

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

1. A set of rights $R$.
2. A set of types $T$.
3. A set of state-changing commands, as defined above.
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.

## 2.2 The Single-Object TAM (SOTAM) Model

SOTAM is a simplified version of TAM, with the restriction 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. SOTAM stipulates that all operations in the body of a command are confined to a single column.

To appreciate the motivation for SOTAM 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 SOTAM 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 SOTAM commands. Further details on an implementation outline of SOTAM are given in [10]. A central result of this paper is that SOTAM has the same expressive power as TAM, in spite of its much easier implementation.

# 3 Equivalence of TAM and SOTAM

This section develops the central result of this paper, which is the formal equivalence of the expressive power of TAM and SOTAM. Since SOTAM is a restricted version of TAM, every SOTAM system is also a TAM system. To establish equivalence we therefore need to show that every TAM system can be simulated by a SOTAM system. The construction to do this is quite intricate.[2] For ease of exposition, and understanding, we develop the construction in several phases. First, in section 3.1, we identify an essential synchronization protocol which is a critical part of the overall construction. Then, in section 3.2, we show that TAM systems without **create** or **destroy** operations can be reduced to SOTAM systems. Finally we show, in section 3.3, how TAM systems with **create** and **destroy** operations can be reduced to SOTAM systems.

## 3.1 Two Column Synchronization Protocol

It is helpful to approach the TAM to SOTAM simulation by first looking at monotonic systems. Recall that a scheme is monotonic if it does not delete any rights, and does not destroy subjects or objects. Let us understand an *operation* to mean a command with specific actual parameters. An important fact in monotonic systems is that once the precondition for a command is satisfied with respect to a given set of existing subjects, no evolution of the protection state can cause the precondition to become false. In other words, once an operation is authorized it will always remain authorized in the future.

Given any monotonic TAM scheme, we can therefore get an equivalent monotonic SOTAM scheme as follows. Each TAM command that modifies $n$ columns is simulated by $n$ SOTAM commands. Each of these SOTAM commands has the same condition as the original TAM command, but each SOTAM command modifies exactly one of the columns modified by the original TAM command. It is easy to see that every sequence of TAM operations can be simulated by the corresponding

[2]It should be kept in mind that our constructions have to deal with the most general case, in order to demonstrate formal equivalence. In practice we can often employ simpler constructions. For example, the ORCON (originator control) example of [8] is very simply converted to a SOTAM system in [10]. The mechanical construction given in this paper would yield a more complex SOTAM system in this case.

SOTAM operations. Conversely, any sequence of SOTAM operations corresponds to a sequence of TAM operations some of which may only be partially completed. However, the SOTAM sequence can be extended to complete all the partial TAM operations. Therefore the two systems are equivalent.

The construction outlined above does not extend to non-monotonic systems. In a non-monotonic system, operations which are currently authorized may have their preconditions falsified due to deletion of access rights by other non-monotonic operations. At the same time, in SOTAM we have no choice but to simulate TAM commands which modify multiple columns with multiple commands. The key to doing this successfully is to prevent other TAM operations from interfering with the execution of a given TAM operation. The simplest way to do this is to ensure that TAM operations can be executed in the SOTAM simulation only one at a time. To do this we need to synchronize the execution of successive TAM commands in the SOTAM simulation (as described in section 3.2).

Thus, surprisingly, the problem of simulating TAM in SOTAM requires solution of a synchronization problem. Moreover, the synchronization must be achieved using SOTAM commands which can modify only one column at a time. This effectively rules out standard synchronization solutions based on semaphores, locks, or similar mechanisms. In effect we have to achieve synchronization without having shared global variables that are writable by concurrent processes.

The basic synchronization problem, which we call *two column synchronization*, is illustrated in figure 1. The solution to this problem turns out to be critical in constructing a SOTAM simulation of a TAM system. For the moment ignore the $SNC$ row and column in figure 1. In figure 1(a) subject $S_1$ possesses the token, represented by the *token* right in the $[S_1, S_1]$ cell. After $S_1$ is done using the token, it is passed on to $S_2$ as indicated in figure 1(b). The *next* right in the $[S_1, S_2]$ cell serves to connect $S_1$ to $S_2$ in sequence, indicating that the token is to be passed from $S_1$ to $S_2$. Similarly, $S_2$ will pass the token on to $S_3$ in turn.[3] For the moment we can ignore typing and, for simplicity, treat all entities as being of the same type $s$.

The TAM command for solving the two column synchronization problem is straightforward, as follows.

> **command** *transfer-token*$(S_1 : s, S_2 : s)$
>     **if** $token \in [S_1, S_1] \wedge next \in [S_1, S_2]$ **then**
>         **delete** *token* **from** $[S_1, S_1]$;
>         **enter** *token* **in** $[S_2, S_2]$;
> **end**

This command modifies both columns $S_1$ and $S_2$, and is therefore not a SOTAM

[3]The exact manner in which the sequence of token passing is encoded in the access matrix is not material to the synchronization problem. For illustrative purposes we have adopted the scheme described here. The construction of section 3.2 uses a slightly different technique.

|  | $SNC$ | $S_1$ | $S_2$ | $S_3$ | ... |
|---|---|---|---|---|---|
| $SNC$ | 0 | 0 | 0 | 0 | ... |
| $S_1$ |  | *token* | *next* |  | ... |
| $S_2$ |  |  |  | *next* | ... |
| $S_3$ |  |  |  |  | ... |
| ... | ... | ... | ... | ... | ... |

(a) $S_1$ possesses the token

|  | $SNC$ | $S_1$ | $S_2$ | $S_3$ | ... |
|---|---|---|---|---|---|
| $SNC$ |  | 0 | 0 | 0 | ... |
| $S_1$ |  |  | *next* |  | ... |
| $S_2$ |  |  | *token* | *next* | ... |
| $S_3$ |  |  |  |  | ... |
| ... | ... | ... | ... | ... | ... |

(b) The token has been transferred to $S_2$

Figure 1: Two Column Synchronization

command.

The transfer-token TAM command can be simulated by four SOTAM commands, which use the $SNC$ row in the access matrix to synchronize. We use three rights denoted 0, 1, and 2, for this purpose. Only one of these rights can be present at a time in a $[SNC, S_i]$ cell, and they do not occur outside of the $SNC$ row. Normally each column $S_i$ has $0 \in [SNC, S_i]$ indicating the quiescent state with respect to the synchronization commands. The meaning of $1 \in [SNC, S_i]$ is that $S_i$ is ready to pass the token. The meaning of $2 \in [SNC, S_j]$ is that $S_j$ is ready to receive the token. The four SOTAM commands to simulate the *transfer-token* TAM command are as follows.

**command** *transfer-token-1*$(S_1, SNC)$
  **if** *token* $\in [S_1, S_1]$ **then**
    **delete** *token* **from** $[S_1, S_1]$;
    **delete** 0 **from** $[SNC, S_1]$;
    **enter** 1 **in** $[SNC, S_1]$;
**end**

**command** *transfer-token-2*$(S_1, S_2, SNC)$
  **if** $1 \in [SNC, S_1] \wedge$ *next* $\in [S_1, S_2]$ **then**

**delete** 0 **from** $[SNC, S_2]$;
**enter** 2 **in** $[SNC, S_2]$;
**end**

**command** *transfer-token-3*$(S_1, S_2, SNC)$
**if** $1 \in [SNC, S_1] \wedge 2 \in [SNC, S_2] \wedge$
$next \in [S_1, S_2]$
**then**
**delete** 1 **from** $[SNC, S_1]$;
**enter** 0 **in** $[SNC, S_1]$;
**end**

**command** *transfer-token-4*$(S_1, S_2, SNC)$
**if** $0 \in [SNC, S_1] \wedge 2 \in [SNC, S_2] \wedge$
$next \in [S_1, S_2]$
**then**
**delete** 2 **from** $[SNC, S_2]$;
**enter** 0 **in** $[SNC, S_2]$;
**enter** *token* **in** $[S_2, S_2]$;
**end**

The correctness of these commands is intuitively obvious, and a formal proof could be easily given. Also note that the **enter** operation in the *transfer-token-4* command can be modified to enter $token'$, rather than $token$, in $[S_2, S_2]$. In this way we can pass a modified token from one column to another by means of SOTAM commands.

We call the protocol described by these four commands as the two column synchronization protocol. As we will see this protocol is repeatedly invoked in the constructions of this paper.

## 3.2 Equivalence Without Create and Destroy

We now prove the equivalence of TAM and SOTAM in the absence of create and destroy operations. This is done by giving a procedure to construct a SOTAM system that can simulate any given TAM system. Every TAM subject or object is simulated in the SOTAM system as a subject (i.e., every column has a corresponding row in the access matrix). In other words the access matrix of the SOTAM system is square. This entails no loss of generality, since TAM subject are not necessarily active entities.

The SOTAM system contains a subject $SNC$ of type *snc*, where *snc* is distinct from any type in the given TAM system. The role of $SNC$ is to enable two column synchronization, as discussed in section 3.1. As we will see, $SNC$ is also used to sequentialize the execution of TAM operations, and to sequentialize the multiple SOTAM operations needed to simulate a given TAM operation.

The SOTAM system also contains the following rights, in addition to the rights defined in the given TAM system.

- $\{0, 1, 2, token, token'\}$
- $\{p_{i,j} \mid j = 1 \ldots n$, for each TAM command $C_i$ (where $C_i$ has $n$ parameters)$\}$

Except for *token*, these rights occur only in the $SNC$ row and column. The *token* right also occurs in the diagonal cells of the SOTAM access matrix. It is assumed, without loss of generality, that these rights are distinct from the rights in the given TAM system.

The initial state of the SOTAM system consists of the initial state of the TAM system augmented in two respects. First, an empty row is introduced for every TAM object, which does not have a row in the given TAM access matrix. Secondly, the $SNC$ subject is introduced in the access matrix with $[SNC, SNC] = \{token, 0\}$, and $[SNC, S_i] = \{0\}$ for all subjects $S_i \neq SNC$.

In the absence of **creates** and **destroys**, the body of a TAM command with $n$ parameters can be rearranged to have the following structure.

**command** $C_i(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n)$
    **if** $\alpha(S_1, S_2, \ldots, S_n)$ **then**
        **enter in/delete from** column $S_1$;
        **enter in/delete from** column $S_2$;
        ...
        **enter in/delete from** column $S_n$;
**end**

That is, the primitive operations occur sequentially on a column-by-column basis. Of course, some of the columns may have no operations, being referenced only in the condition part; but for the general case we assume the above structure.

Let us suppose the above TAM command is invoked with actual parameters $S_1, S_2, \ldots, S_n$.[4] This operation will be simulated by several SOTAM operations. The simulation proceeds in three phases, respectively illustrated in figures 2, 3 and 4. In these figures we show only the relevant portion of the access matrix, and only those rights introduced specifically for the SOTAM simulation. It is understood that the TAM rights are distributed exactly as in the TAM system.

The first phase consists of a single SOTAM command $C_i$-$I$ which tests whether (i) the condition of the TAM command $\alpha(S_1, S_2, \ldots, S_n)$ is true, and (ii) whether

[4]For convenience and readability, we are using the same symbols for the formal parameters of the command $C_i$, as well as for the actual parameters of a particular invocation of $C_i$. The context will make it clear whether the symbol $S_i$ refers to a formal or actual parameter.

*token* $\in [SNC, SNC]$. The former test is obviously required. The latter ensures that the SOTAM simulation of $C_i(S_1, S_2, \ldots, S_n)$ can begin only if no other TAM operation is currently being simulated. It also ensures that once phase I of the simulation of $C_i(S_1, S_2, \ldots, S_n)$ has started, the simulation will proceed to completion before simulation of another TAM command can begin. In other words TAM operations are simulated serially, with no interleaving. The phase I SOTAM command is as follows.

**command** $C_i\text{-}I(S_1 : s_1, S_2 : s_2, \ldots, S_n :$
$s_n, SNC : snc)$

**if** $\alpha(S_1, S_2, \ldots, S_n) \wedge token \in [SNC, SNC]$
**then**
  **enter** $p_{i,1}$ **in** $[S_1, SNC]$;
  **enter** $p_{i,2}$ **in** $[S_2, SNC]$;
  ...
  **enter** $p_{i,n}$ **in** $[S_n, SNC]$;
  **delete** *token* **from** $[SNC, SNC]$;
  **delete** 0 **from** $[SNC, SNC]$;
  **enter** 1 **in** $[SNC, SNC]$;
**end**

The body of this command enters $p_{i,j}$ in $[S_j, SNC]$ for $j = 1 \ldots n$, signifying that $S_j$ is the $j$-th parameter of the TAM command being simulated. It also removes the *token* right from $[SNC, SNC]$, and replaces 0 in $[SNC, SNC]$ by 1 signifying that the *token* can be moved from SNC column. The states of the access matrix, before and after execution of $C_i\text{-}I$, are outlined in figures 2(a) and 2(b) respectively.

In phase II of the simulation the *token* right is passed, in turn, from $[SNC, SNC]$ to $[S_1, S_1]$ to $[S_2, S_2]$, and so on to $[S_n, S_n]$. Each passage requires a total of four SOTAM commands based on the two-column synchronization protocol of section 3.1.[5] The details of the protocol are shown here only for the transfer of the token from column $S_j$ to $S_{j+1}$. The SOTAM commands for transferring the token from $SNC$ to $S_1$, and from $S_n$ to $SNC$ are not explicitly shown, since they are so similar.

Let $\pi(S_1, S_2, \ldots, S_n, SNC)$ represent the following predicate.

$$[p_{i,1} \in [S_1, SNC] \wedge p_{i,2} \in [S_2, SNC] \wedge \ldots \wedge p_{i,n} \in [S_n, SNC]$$

The changes in column $S_j$ in the original TAM command are carried out by a SOTAM command whose condition tests for $\pi(S_1, S_2, \ldots, S_n, SNC)$, and the presence of the *token* in $[S_j, S_j]$. The SOTAM command which carries out the changes in column $S_j$ also removes *token* from $[S_j, S_j]$, and replaces 0 in $[SNC, S_j]$ by 1 signifying that

[5] One difference in detail is that the connection between consecutive columns is achieved by means of the $p_{i,j}$ rights in the $SNC$ column, rather than by the *next* right in the $[S_j, S_{j+1}]$ cell.

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | *token*, 0 | 0 | 0 | | 0 |
| $S_1$ | | | | | |
| $S_2$ | | | | | |
| $\ldots$ | | | | | |
| $S_n$ | | | | | |

(a)

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | 1 | 0 | 0 | | 0 |
| $S_1$ | $p_{i,1}$ | | | | |
| $S_2$ | $p_{i,2}$ | | | | |
| $\ldots$ | | | | | |
| $S_n$ | $p_{i,n}$ | | | | |

(b)

Figure 2: SOTAM Simulation of the $n$-Parameter TAM Command $C_i$: Phase I

the token can be moved to the next column. The SOTAM command for simulating changes by the TAM command in column $j$ is shown below.

**command** $C_i$-$II$-$j$-1($S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc$)
  **if** $\pi(S_1, S_2, \ldots, S_n, SNC) \wedge token \in [S_j, S_j]$
  **then**
    **enter in/delete from** column $S_j$;
    **delete** *token* **from** $[S_j, S_j]$;
    **delete** 0 **from** $[SNC, S_j]$;
    **enter** 1 **in** $[SNC, S_j]$;
**end**

The two-column synchronization protocol, begun above, is then completed to move *token* to $[S_{j+1}, S_{j+1}]$, using the following commands.

**command** $C_i$-$II$-$j$-2($S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc$)
  **if** $1 \in [SNC, S_j] \wedge \pi(S_1, S_2, \ldots, S_n, SNC)$
  **then**

**delete** $0$ **from** $[SNC, S_{j+1}]$;
**enter** $2$ **in** $[SNC, S_{j+1}]$;
**end**

**command** $C_i$-$II$-$j$-3$(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$

**if** $1 \in [SNC, S_j] \wedge 2 \in [SNC, S_{j+1}] \wedge \pi(S_1, S_2, \ldots, S_n, SNC)$

**then**
**delete** $1$ **from** $[SNC, S_j]$;
**enter** $0$ **in** $[SNC, S_j]$;
**end**

**command** $C_i$-$II$-$j$-4$(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$

**if** $0 \in [SNC, S_j] \wedge 2 \in [SNC, S_{j+1}] \wedge \pi(S_1, S_2, \ldots, S_n, SNC)$

**then**
**delete** $2$ **from** $[SNC, S_{j+1}]$;
**enter** $0$ **in** $[SNC, S_{j+1}]$;
**enter** *token* **in** $[S_{j+1}, S_{j+1}]$;
**end**

In this manner, the simulation of the TAM command proceeds on a column-by-column basis, as depicted in figure 3.

Finally, when the token is passed from $[S_n, S_n]$ back to $[SNC, SNC]$, the two-column synchronization passes it back as the $token'$ right, instead of *token*, as shown in figure 4(a). At this point the $SNC$ column is cleaned out to the state of figure 4(b) using the following SOTAM command.

**command** $C_i$-$III(S_1 : s_1, S_2 : s_2, \ldots, S_n : s_n, SNC : snc)$

**if** $\pi(S_1, S_2, \ldots, S_n, SNC) \wedge token' \in [SNC, SNC]$

**then**
**delete** $p_{i,1}$ **from** $[S_1, SNC]$;
**delete** $p_{i,2}$ **from** $[S_2, SNC]$;
...
**delete** $p_{i,n}$ **from** $[S_n, SNC]$;
**delete** $token'$ **from** $[SNC, SNC]$;
**enter** *token* **in** $[SNC, SNC]$;
**end**

The SOTAM system is now ready to simulate another TAM command.

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | $token$, 0 | 0 | 0 | | 0 |
| $S_1$ | $p_{i,1}$ | | | | |
| $S_2$ | $p_{i,2}$ | | | | |
| $\ldots$ | | | | | |
| $S_n$ | $p_{i,n}$ | | | | |

(a)

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | 0 | 0 | 0 | | 0 |
| $S_1$ | $p_{i,1}$ | $token$ | | | |
| $S_2$ | $p_{i,2}$ | | | | |
| $\ldots$ | | | | | |
| $S_n$ | $p_{i,n}$ | | | | |

(b)

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | 0 | 0 | 0 | | 0 |
| $S_1$ | $p_{i,1}$ | | | | |
| $S_2$ | $p_{i,2}$ | | $token$ | | |
| $\ldots$ | | | | | |
| $S_n$ | $p_{i,n}$ | | | | |

(c)

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | 0 | 0 | 0 | | 0 |
| $S_1$ | $p_{i,1}$ | | | | |
| $S_2$ | $p_{i,2}$ | | | | |
| $\ldots$ | | | | | |
| $S_n$ | $p_{i,n}$ | | | | $token$ |

(d)

Figure 3: SOTAM Simulation of the $n$-Parameter TAM Command $C_i$: Phase II

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | $token'$, 0 | 0 | 0 | | 0 |
| $S_1$ | $p_{i,1}$ | | | | |
| $S_2$ | $p_{i,2}$ | | | | |
| $\ldots$ | | | | | |
| $S_n$ | $p_{i,n}$ | | | | |

(a)

| | $SNC$ | $S_1$ | $S_2$ | $\ldots$ | $S_n$ |
|---|---|---|---|---|---|
| $SNC$ | $token$, 0 | 0 | 0 | | 0 |
| $S_1$ | | | | | |
| $S_2$ | | | | | |
| $\ldots$ | | | | | |
| $S_n$ | | | | | |

(b)

Figure 4: SOTAM Simulation of the $n$-Parameter TAM Command $C_i$: Phase III

Note that a $n$-parameter TAM command requires $4(n+1)+1$ (or $4n+5$) SOTAM commands in this construction. The SOTAM simulation operates on $n+1$ columns consisting of the $n$ columns in the TAM command, and the $SNC$ column. The modification of each column is bundled with the first step of the two-column synchronization protocol, giving us $4(n+1)$ commands. One command is required to clean out the $SNC$ column at the end. Various optimizations are possible. Significantly, the SOTAM simulation is linear in the size of the TAM command being simulation.

A proof sketch for the correctness of the construction is given below.

**Theorem 1** For every TAM system $\mathcal{S}_1$ the construction outlined above produces an equivalent SOTAM system $\mathcal{S}_2$.

**Proof Sketch:** It is easy to see that any reachable state in $\mathcal{S}_1$ can be reached in $\mathcal{S}_2$ by simulating each TAM command by SOTAM commands as discussed above. Conversely any reachable state in $\mathcal{S}_2$, with $token \in [SNC, SNC]$, will correspond to a reachable state in $\mathcal{S}_1$. A reachable state in $\mathcal{S}_2$, with $token \notin [SNC, SNC]$, will correspond to a state in $\mathcal{S}_1$ where one TAM command has been partially completed. These claims follow from the property that once the SOTAM command $C_i$-$I$ removes the $token$ right from $[SNC, SNC]$, the entire TAM command $C_i$ must be simulated before the *token* right is returned to $[SNC, SNC]$. Moreover, no SOTAM commands other than those that simulate $C_i$ with the specific parameters used in $C_i$-$I$ can execute until the *token* right is returned to $[SNC, SNC]$. 2

A formal inductive proof can be easily given, but is omitted for lack of space.

## 3.3 Expressive Power With Create and Destroy

We now consider TAM with create and destroy operations. There are several ways in which the construction of section 3.2 can be extended to allow for creation and destruction. We describe one of them here. We have already assumed that the SOTAM system will have a square access matrix, in which every object (column) is also a subject (row). So our focus will be on subject creation.

The primitive TAM operation "**create subject** $S_i$" introduces an empty row and column in the access matrix for the newly created subject $S_i$. A SOTAM command which has this primitive operation in its body is quite restricted in what it can do, since all **enter** and **delete** operations will be confined to the new column $S_i$. We will therefore simulate creation in SOTAM in two steps.

- First we will allow unconditional creation of subjects to occur. However, the created subject will be dormant, indicated by the *dormant* right in the diagonal cell $[S_i, S_i]$. The unconditional creation also introduces the 0 right in the $[SNC, S_i]$ cell. We view the unconditional creation as occurring on demand as needed.
- Dormant subjects will be brought to life by replacing *dormant* in the diagonal cell by *alive*.

A dormant subject cannot be a parameter in any TAM command. This will be ensured by modifying the TAM system so that each TAM command tests for the *alive* right in the diagonal cells for every parameter in the command.

Consider a TAM command which requires $m$ pre-existing subjects or objects, $S_1, \ldots, S_m$, and creates $n - m$ subjects or objects, $S_{m+1}, \ldots, S_n$. We will modify the given TAM command as follows. Let $\alpha(S_1, \ldots, S_m)$ be the given condition in this command. This condition will be supplemented by the tests $alive \in [S_i, S_i]$, for $i = 1 \ldots m$. It will be further supplemented by the tests $dormant \in [S_i, S_i]$, for $i = m + 1 \ldots n$. All create operations in the body of the original TAM command will be discarded. Instead the dormant subjects will be made alive by the operations

> **enter** *alive* **in** $[S_j, S_j]$;
> **delete** *dormant* **from** $[S_j, S_j]$

for $j = m + 1 \ldots n$.

The **destroy** operation can be similarly removed from the body of the given TAM commands, and relegated to a background "garbage collection" activity. To do this every "**destroy** $S_j$" primitive operation is replaced in the TAM command by the following operations

**enter** *dead* **in** $[S_j, S_j]$;
**delete** *alive* **from** $[S_j, S_j]$;

The meaning of $dead \in [S_j, S_j]$ is that $S_j$ has effectively been destroyed, since it cannot function as a parameter in any TAM command. The background garbage collection can be done by introducing the following command for every type $s$.

**command** expunge($S : s$)
  **if** $dead \in [S, S]$ **then**
    **destroy subject** $S$;
**end**

In this manner the original TAM command has been reduced to one which only has **enter** and **delete** operations in its body. The construction of the SOTAM simulation can now proceed as in section 3.2.

# 4 Conclusion

In this paper we have shown that the expressive power of the typed access matrix (TAM) model [8], and the single-object TAM (SOTAM) model are formally equivalent. This is an important result because SOTAM is known to have a simple and efficient implementation in a distributed environment, using the familiar client-server architecture [10]. In a nutshell, this result tells us that manipulation of access control information can be achieved in its most general form by manipulation of a single access control list (ACL) at a time.

The principal conclusion of this paper is that SOTAM has the same expressive power as TAM, while having a simple implementation at the same time. Looking towards future work, our goal is to find other simpler cases of TAM which retain equivalence of expressive power to TAM. The motivation in finding such simpler cases is that even though these simpler cases are equivalent to TAM they could still be easily implemented (than TAM) in a distributed environment. The construction given in this paper is general, and it gives an equivalent SOTAM scheme for every given TAM scheme. In many practical cases there are much simpler constructions, for example see the ORCON construction in [10]. (This situation is analogous to constructions which show that the **goto** statement can be eliminated in programming languages. Those constructions would never be used to actually write a program. But they do establish equivalence of expressive power.)

# References


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

[2] Ammann, P.E. and Sandhu, R.S. "Implementing Transaction Control Expressions by Checking for Absence of Access Rights." *Proc. Eighth Annual Computer Security Applications Conference*, San Antonio, Texas, December 1992.

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

[4] Lampson, B.W. "Protection." *5th Princeton Symposium on Information Science and Systems*, 437-443 (1971). Reprinted in *ACM Operating Systems Review* 8(1):18-24 (1974).

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

[6] Sandhu, R.S. "Transaction Control Expressions for Separation of Duties." *Proc. Fourth Aerospace Computer Security Applications Conference*, Orlando, Florida, December 1988, pages 282-286.

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

[9] Sandhu, R.S. and Suri, G.S. "Non-monotonic Transformations of Access Rights." *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1992, pages 148-161.

[10] Sandhu, R.S. and Suri, G.S. "Implementation Considerations for the Typed Access Matrix Model in a Distributed Environment." *Proc. 15th NIST-NCSC National Computer Security Conference*, Baltimore, MD, October 1992, pages 221-235.

[11] Snyder, L. "Formal Models of Capability-Based Protection Systems." *IEEE Transactions on Computers* C-30(3):172-181 (1981).