The Typed Access Matrix Model

Ravi S. Sandhu

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

Abstract

The access matrix model as formalized by Harrison, Ruzzo, and Ullman (HRU) has broad expressive power. Unfortunately, HRU has weak safety properties (i.e., the determination of whether or not a given subject can ever acquire access to a given object). Most security policies of practical interest fall into the undecidable cases of HRU. This is true even for monotonic policies (i.e., where access rights can be deleted only if the deletion is itself reversible). In this paper we define the typed access matrix (TAM) model by introducing strong typing into HRU (i.e., each subject or object is created to be of a particular type which thereafter does not change). We prove that monotonic TAM (MTAM) has strong safety properties similar to Sandhu's Schematic Protection Model. Safety in MTAM's decidable case is, however, NP-hard. We develop a model called ternary MTAM which has polynomial safety for its decidable case, and which nevertheless retains the full expressive power of MTAM. There is compelling evidence that the decidable safety cases of ternary MTAM are quite adequate for modeling practical monotonic security policies.

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. 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.

The *protection state* of a system is defined by the *privileges*[*] possessed by the individual subjects. Hereafter, we understand state to mean protection state. Once the initial state of a system has been established, the state evolves by the autonomous activity of subjects. A security model provides a framework for specifying the dynamics of the protection state. This is usually done by stating rules which prescribe the authorization for making incremental changes in the state. We call such a collection of rules an *authorization scheme*, often abbreviated simply as *scheme*. To understand the implications of a scheme it must be possible to determine the cumulative effect of authorized incremental changes in the protection state. The incremental state changes authorized by a scheme may appear innocent enough in isolation, although their cumulative effect turns out to be undesirable. So for a given initial state and authorization scheme, we need to characterize protection states that are reachable.

This problem was first identified in [13] where it is called 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 pos-

[*]We view "privilege" as an undefined primitive concept. For the most part, privileges can be treated as synonymous with access rights. However, there are privileges such as security level, type or role, which are usually represented as attributes of subjects and objects rather than as access rights.

sesses a particular privilege for a specific object? It is the fundamental question which a access control model must confront. Since subjects are usually authorized to create new subjects and objects, the system is unbounded; and it is not certain that such analysis will be decidable, let alone tractable, without sacrificing generality.

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) [13] has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties. Safety is undecidable for most policies of practical interest, even in the monotonic version of HRU [14] (which only allows revocation which is itself reversible).

The safety problem is closely related to the so-called fundamental flaw of discretionary access control (DAC). DAC is vulnerable to Trojan Horses, partly because Trojan Horse laden programs can surreptitiously modify the protection state without explicit instruction from the users. These Trojan Horses are, however, constrained by the authorization scheme.† They can modify the protection state only by using commands which are authorized in the current state. Consequently, the Trojan Horse vulnerability of DAC requires that we assume the worst case regarding propagation of access rights in a system. What we need therefore is a model with strong safety properties and broad expressive power. The negative results of HRU have led many researchers to believe that such a model does not exist.

There are two principal contributions in this paper. First, there is the demonstration that strong typing is the key concept for achieving strong safety properties. Strong typing requires that each subject or object is created to be of a particular type which thereafter does not change.‡ In this paper we define the typed access matrix (TAM) model by introducing the notion of strong typing into HRU. We prove that monotonic TAM (MTAM) has strong safety properties similar to those of Sandhu's Schematic Protection Model [21, 22], and its recent extension by Ammann and Sandhu to Extended SPM (ESPM) [2, 3, 4].

Second we show how safety can be made tractable, essentially without loss of expressive power, by using the concepts of local authorization and multi-parent creation from SPM and ESPM. Specifically we define a simplified version of MTAM, called ternary MTAM, for which safety has polynomial complexity (for its decidable cases). This is in contrast to MTAM in which safety is NP-hard (for its decidable cases). At the same time we show that ternary MTAM is formally equivalent in expressive power to MTAM. (Of course, the decidable cases of ternary MTAM and MTAM cannot be co-extensive without implying P=NP.) There is compelling evidence that the decidable safety case of ternary MTAM is quite adequate for expressing practical monotonic security policies.

The rest of the paper is organized as follows. Section 2 gives a brief historical review on the topic of balancing safety versus expressive power in security models. Section 3 gives a formal definition and intuitive explanation of the TAM model and its monotonic version MTAM. In Section 4 we show how the originator-control policy for military documents has a natural expression in MTAM by means of multi-parent creation. Section 5 defines a canonical form for TAM systems and proves that every TAM system has an equivalent specification in this form. Section 6 establishes safety results for MTAM which are very similar to those for SPM and ESPM. It is shown that safety in MTAM's decidable case is NP-hard with respect to the initial size of the access matrix. In Section 7 we develop a model called ternary MTAM which has polynomial safety for its decidable case, and which nevertheless retains the full expressive power of MTAM. Notably, the decidable safety cases of ternary MTAM are quite adequate for modeling practical monotonic security policies. Section 8 concludes the paper.

2 Background

Safety analysis issues were first formalized by Harrison, Ruzzo and Ullman [13] in context of the well-known access matrix model [15]. The resulting model is commonly known as HRU. The matrix has a row for each subject and a column for object. Subjects are also considered to be objects, and thereby have a row and a column in the access matrix. The $[X, Y]$ cell of the matrix contains symbols called rights which authorize subject X to perform operations on object Y. An authorization scheme in HRU is defined by a set of commands. Each command has a condition part and a body. The condition specifies the rights that are required to exist in the matrix before the body can be executed for its actual arguments. The body consists

†We assume that the authorization scheme is enforced by a high-assurance reference monitor. If the reference monitor can be bypassed there is, of course, no basis for security.

‡Strong typing is analogous (but not identical) to tranquility in the Bell-LaPadula style of security models [6], whereby security labels on subjects and objects cannot be changed. The adverse consequences of unrestrained non-tranquility are well known [11, 19, 20].

of a sequence of primitive operations. The primitive operations enter or delete a right from a cell of the matrix, create a new row or column, or destroy an existing row or column.

In the general HRU setting safety is undecidable [13]. HRU does have decidable (albeit NP-complete) safety for the mono-operational case, where the condition part of each command is allowed to be arbitrarily complex but the body can only consist of a single primitive operation. The mono-operational assumption has the unfortunate effect of making creation in HRU essentially useless. A single primitive operation in HRU can create only an empty row and/or column in the matrix. This new row/column is not attached to its creator in mono-operational systems. There is therefore no way of distinguishing the children of one parent from that of another!

A restriction on expressive power that can have substantial benefits for safety analysis is that of monotonicity. Monotonic models do not allow the deletion of access privileges. It must be noted that a strictly monotonic model is too restrictive to be of much practical use, since the ability to delete access privileges is an important requirement. We are really interested in models which can be reduced to monotonic models for purpose of safety analysis. In particular, we can ignore deletion of an access privilege p whenever the deletion can itself be undone by regranting p (see Section 4 for an example). This is by far the most common form of revocation, and it is indeed fortunate that such revocation can be ignored for the purposes of worst case safety analysis.

Unfortunately monotonicity does not seem to help with HRU. Specifically, safety in HRU is known to be decidable for mono-conditional monotonic commands (i.e., commands whose condition part has only one term) but is undecidable even for bi-conditional monotonic commands (i.e., commands whose condition part has exactly two terms) [14]. In practical terms, mono-conditional commands can only test one cell of the access matrix whereas bi-conditional commands can test two cells. There are numerous security policies which need bi-conditional, and more generally, multi-conditional commands. So the HRU demarcation of decidable versus undecidable safety, places many practically useful systems on the undecidable side of this demarcation. To summarize, there does not appear to be any natural and useful special case of HRU for which safety is efficiently decidable. The very weak assumptions from which undecidability follows are most disappointing.

A number of protection models were developed in response to these negative results of HRU. The take-grant model was introduced by Lipton and Snyder [16] and analyzed in considerable detail by a number of authors [7, 8, 9, 17, 25, 26]. The principal insights obtained from the take-grant work can be summarized as follows. Firstly, take-grant is a simple model with linear time algorithms for safety. Yet it falls outside the known decidable classes of HRU (because it is bi-conditional and multi-operational). Secondly, the consequence of the take-grant rules, i.e., symmetry of access rights propagation, is not immediately obvious. Small changes in the rules lead to dramatically different behavior [9, 17]. Thirdly, it is important to analyze the model under different assumptions regarding what subjects can and cannot do [7, 8, 26].

The take-grant model was deliberately designed to be of limited expressive power, so that it would not exhibit the undecidable safety of HRU. There is therefore a substantial gap in expressive power between take-grant and HRU. Sandhu's Schematic Protection Model (SPM) [21] was developed to fill the gap in expressive power between take-grant and HRU, while sustaining efficient safety analysis.

The key notion introduced in SPM is that of *security types*. The intuition is that all instances of a security type are treated uniformly by the authorization scheme. Hereafter, we understand type to mean security type. Every SPM entity (i.e., subject or object) is created to be a particular type, which thereafter does not change. In other words SPM requires strong typing of subjects and objects.

SPM authorizes creation by means of a binary can-create relation *cc* on types. Subjects of type u can create an object of type v if and only if $(u, v) \in cc$. Visualize the *cc* relation as a directed graph, called the *cc*-graph, whose vertices are the types with an edge from u to v if and only if $(u, v) \in cc$. SPM has decidable safety provided the *cc*-graph is acyclic [21], whereas with arbitrary cycles in *cc* safety is undecidable [22]. Moreover, in the special case of loops (i.e., cycles of length one which allow a subject to create subjects of its own type) safety remains decidable if an additional requirement known as attenuating is imposed. Useful security policies do not appear to require arbitrary cycles in the *cc*-graph. Acyclic *cc*, or at most attenuating loops in *cc*, appear to satisfy most requirements [23]. Thus, SPM has a remarkably useful demarcation between its decidable and undecidable cases for safety. This is in sharp contrast to the HRU model which requires multi-conditional and multi-operational commands for most systems of interest, placing them in the undecidable case of HRU.

Despite SPM's demonstrated expressive power [23], attempts to show the equivalence of SPM to monotonic HRU were not successful. This led to the development of Extended SPM (ESPM) by Ammann and Sandhu [2, 4]. ESPM generalizes the conventional single parent creation operation of SPM, to allow multiple parents for a child. ESPM is formally equivalent to monotonic HRU [2, 4], while it retains the positive safety results of SPM [3, 4]. Ammann, Lipton and Sandhu have recently shown that in monotonic models multi-parent creation is strictly more powerful than single-parent creation [5]. This completes our historical review.

3 The TAM and MTAM Models

In this Section we define the typed access matrix (TAM) model, and its monotonic version (MTAM). TAM is defined by introducing strong typing into HRU. Here, we directly give a definition of TAM, indicating along the way how HRU is a special case of TAM.

3.1 Basic Definitions

The first step in defining a TAM system is to define the access rights and types as follows.

Definition 1 There is a finite set of access *rights* denoted by R. □

Definition 2 There is a finite set of *object types* (or simply *types*) denoted by T. There is a set of *subject types* T_S, $T_S \subseteq T$. □

The types and rights are defined when a system is initialized and thereafter T and R remain constant. For example, $T = \{user, so, file\}$ specifies there are three types, viz., user, security-officer and file, with, say, $T_S = \{user, so\}$. A typical example of rights would be $R = \{r, w, e, o\}$ respectively denoting read, write, execute and own. We emphasize that the types and rights are specified as part of the system definition, and are not predefined in the model.

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. The $[X, Y]$ cell contains rights which subject X possesses for object Y. This is formalized below.

Definition 3 The *protection state* (or simply *state*) of a system is the four-tuple (SUB, OBJ, t, AM) interpreted as follows:

- SUB is the set of *subjects*.
- OBJ is the set of *objects*, $SUB \subseteq OBJ$. We say that members of $OBJ - SUB$ are the *pure objects*, i.e., objects which are not also subjects. Pure objects only have a column and no row in the access matrix.
- $t : OBJ \rightarrow T$, is the *type function* which gives the type of every object. It is the case that $t : SUB \rightarrow T_S$ and $t : (OBJ - SUB) \rightarrow (T - T_S)$, i.e., the type function maps subjects to subject types and pure objects to pure object types.
- AM is the *access matrix*, with a row for every subject in SUB and a column for every object in OBJ. We denote the contents of the (S, O) cell of AM by $[S, O]$. We have $[S, O] \subseteq R$.

When we need to explicitly identify the state, we will do so by means of a superscript, i.e., SUB^k, OBJ^k, t^k, AM^k and $[S, O]^k$ all refer to state k. □

HRU is a special case of TAM in which there only two types, say, *subject* and *object*. Every subject in SUB is of type *subject* and every pure object in $(OBJ - SUB)$ is of type *object*.[§]

3.2 TAM Commands

The rights in the access matrix cells serve two purposes. Firstly, presence of a right, such as r, in $[X, Y]$ may authorize X to perform, say, the read operation on Y. Secondly, presence of a right, say o, in $[X, Y]$ may authorize X to perform some operation which changes the access matrix, e.g., by entering r in $[Z, Y]$. In other words, X as the owner of Y can change the matrix so that Z can read Y. The focus of TAM is on this second purpose of rights, i.e., the authorization by which the access matrix itself gets changed.

The protection state of the system is changed by means of commands defined as follows.

Definition 4 A TAM *command* has the format shown in Table 1 where: α is the *name* of the command; X_1, X_2, ..., X_k are *formal parameters* whose types are respectively t_1, t_2, ..., t_k; r_1, r_2, ..., r_m are rights; and s_1, s_2, ..., s_m and o_1, o_2, ..., o_m are integers between 1 and k. Each op_i is one of the *primitive operations* shown in Table 2, in which $r \in R$ and

[§]Strictly speaking, the situation is slightly more subtle since HRU parameters are not typed, i.e., some formal parameters in HRU can be substituted by subjects or pure objects. This can be simulated in TAM by having multiple versions of the same command, so as to allow for all possible type combinations of formal parameters.

```
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
```

(a) Conditional Commands

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

(b) Unconditional Commands

Table 1: TAM Commands

```
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 Operations

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

(b) Non-Monotonic Operations

Table 2: TAM Primitive Operations

s and o are integers between 1 and k. The meaning of each primitive operation is formally given in Table 3. □

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*. Examples of TAM commands will be discussed in Section 4.

A TAM command is invoked by substituting actual parameters of the appropriate types for the formal parameters. TAM makes no statement about who initiates the command. This is consistent with worst-case safety analysis. 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 (and the pre-conditions for all **create** and **destroy** operations are satisfied). The commands are executed serially, i.e., there is no interleaving of operations from different commands. (Alternately, we can assume an interleaved model of execution with a serializability requirement.)

3.3 TAM Primitive Operations

There are six primitive operations in TAM. They are formally defined in Table 3, where the states before and after the operation are denoted as (SUB, OBJ, t, AM) and (SUB', OBJ', t', AM') respectively.

The **enter** operation enters a right into an existing cell of the access matrix. As described in Table 3, this operation leaves the set of objects, subjects and their types unchanged. It also leaves all cells except one unchanged. In the one cell that is (possibly) changed a single right is entered. The contents of each 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** removes 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. This requirement can be easily checked.

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.[¶]

[¶]One could argue that **create object** and **destroy object** should properly be called **create pure object** and **destroy pure object**. We have followed the original terminology of [13].

1. **enter r into $[X_s, X_o]$**
   $OBJ' = OBJ$
   $SUB' = SUB$
   $t'(O) = t(O)$ for all $O \in OBJ$
   $[S,O]' = [S,O]$ if $(S,O) \neq (X_s, X_o)$
   $[X_s, X_o]' = [X_s, X_o] \cup \{r\}$

2. **delete r from $[X_s, X_o]$**
   $OBJ' = OBJ$
   $SUB' = SUB$
   $t'(O) = t(O)$ for all $O \in OBJ$
   $[S,O]' = [S,O]$ if $(S,O) \neq (X_s, X_o)$
   $[X_s, X_o]' = [X_s, X_o] - \{r\}$

3. **create subject X_s of type t_s** (where $X_s \notin OBJ$)
   $OBJ' = OBJ \cup \{X_s\}$
   $SUB' = SUB \cup \{X_s\}$
   $t'(O) = t(O)$ for all $O \in OBJ$
   $t'(X_s) = t_s$
   $[S,O]' = [S,O]$ for all $(S,O) \in SUB \times OBJ$
   $[X_s, O]' = \phi$ for all $O \in OBJ'$
   $[S, X_s]' = \phi$ for all $S \in SUB'$

4. **destroy subject X_s** (where $X_s \in SUB$)
   $OBJ' = OBJ - \{X_s\}$
   $SUB' = SUB - \{X_s\}$
   $t'(O) = t(O)$ for all $O \in OBJ'$
   $t'(X_s)$ =undefined
   $[S,O]' = [S,O]$ for all $(S,O) \in SUB' \times OBJ'$

5. **create object X_o of type t_o** (where $X_o \notin OBJ$)
   $OBJ' = OBJ \cup \{X_o\}$
   $SUB' = SUB$
   $t'(O) = t(O)$ for all $O \in OBJ$
   $t'(X_o) = t_o$
   $[S,O]' = [S,O]$ for all $(S,O) \in SUB \times OBJ$
   $[S, X_o]' = \phi$ for all $S \in SUB'$

6. **destroy object X_o** (where $X_o \in OBJ - SUB$)
   $OBJ' = OBJ - \{X_o\}$
   $SUB' = SUB$
   $t'(O) = t(O)$ for all $O \in OBJ'$
   $t'(X_o)$ =undefined
   $[S,O]' = [S,O]$ for all $(S,O) \in SUB' \times OBJ'$

Note: If the pre-condition for any create or destroy operation in the body is false, the entire TAM command has no effect.

Table 3: Interpretation of TAM Primitive Operations

3.4 Definition of TAM and MTAM

The following definitions complete our formal definition of TAM and MTAM.

Definition 5 A TAM *authorization scheme* consists of a finite set of rights R, a finite set of types T, and a finite collection of commands. □

Definition 6 A TAM *system* is specified by a TAM scheme.and the initial state of the system, i.e., $(SUB^0, OBJ^0, t^0, AM^0)$. □

Definition 7 The *monotonic typed access matrix* (MTAM) model is identical to TAM except that the **delete, destroy subject and destroy object** primitive operations are omitted. □

Note that once the scheme has been defined it remains fixed for the life of the system. The system state, however, changes with time.

3.5 Expressive Power

There is ample evidence that TAM and MTAM are very expressive in the range of policies that they can express. TAM inherits the expressive power of HRU, which is the most general access control model to date. MTAM inherits the expressive power of monotonic HRU in general, and SPM [23] and ESPM [4] in particular.

Note that disjunctive conditions are easily modeled in TAM by having one command for each component of the condition. Thus the condition "if $P \vee Q$" can be expressed by two command which are otherwise identical but which respectively have the conditions "if P" and "if Q." The absence of rights, however, cannot be tested in TAM commands. The TAM model can, of course, be easily extended to allow tests for absence of access rights. This will, however, only further aggravate the safety problem. Budd [10] has demonstrated a very simple system which tests for absence of rights and has NP-complete safety.

MTAM, being monotonic, is not able to represent certain desirable non-monotonic aspects of access control. For instance the following transfer command cannot be represented in a monotonic model.

```
command transfer-r($S_1 : s, S_2 : s, O : o$)
    if $r \in [S_1, O]$ then
        enter $r$ in $[S_2, O]$;
        delete $r$ from $[S_1, O]$;
end
```

Another useful facility, which cannot be represented in a monotonic model, is a countdown right, i.e., a right which can be used a fixed number of times before it expires.

Having noted these limitations of monotonic models, it is equally important to understand that many policies with non-monotonic components can be reduced to monotonic policies for purpose of safety analysis. In particular most of the common revocation policies fall into this category. We will give a concrete example of this assertion in the next Section.

4 The ORCON Policy

In this Section we discuss the ORCON (originator controlled) policy for control of information in documents. This policy has been discussed in a number of recent papers [1, 12, 18] as an example of a policy in the military sector that is difficult to implement within the classic Bell-LaPadula model [6]. In practice the ORCON policy occurs as one aspect of a larger policy context, which will typically include the usual label-based non-disclosure controls (i.e., simple-security and the $\star$- property). Our discussion here will focus exclusively on the ORCON component of the policy. The solution can be easily extended to the larger context mentioned above.

The ORCON policy 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[||] 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.[**]

Our solution to the ORCON problem is based on the ability in TAM to have multiple parents jointly create a child subject. The solution is illustrated in Figure 1.[††] 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 S_3 of type *cs* (for confined-subject). S_3 obtains the read right for O as part of the creation command. 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 $R = \{\text{own}, \text{read}, \text{write}, \text{cread}, \text{parent}\}$
2. Types $T = \{s, cs, co\}$ with $T_S = \{s, cs\}$
3. The commands are as follows.

```
(a) command create-orcon-object($S_1 : s, O : co$)
            create object $O$ of type $co$;
            enter own in $[S_1, O]$
            enter read in $[S_1, O]$
            enter write in $[S_1, O]$
    end

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

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

(d) command revoke-cread($S_1 : s, S_2 : s, O : co$)
        if own $\in [S_1, O]$ then
            delete cread from $[S_2, O]$
    end

(e) command destroy-orcon-object($S_1 : s, O : co$)
        if own $\in [S_1, O]$ then
            destroy object $O$
    end
```

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

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

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

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			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	parent	
$S_3 : cs$			read		
...					

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

Figure 1: Illustration of the ORCON Policy in TAM

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

(g) command finish-orcon-read($S_2 : s, S_3 : cs$)
        if parent $\in [S_2, S_3]$ then
            destroy subject $S_3$
    end
```

The first three commands given above are monotonic. They respectively authorize the three steps shown in Figure 1. The remaining four commands are non-monotonic. They are examples of revocation commands which are reasonable in this context. Commands d, e and f give the owner of a ORCON object O the authority to revoke access to O. Command g gives the parent of a confined subject the authority to destroy it.

It is important to appreciate that the non-monotonic commands of this ORCON scheme can be ignored for purpose of safety analysis, because the effect of each of these commands is itself reversible. In this sense MTAM is sufficient to model this ORCON solution for purpose of safety analysis. Reversibility of the non-monotonic commands is most apparent for command d, which can be immediately undone by command b. Command g can be compensated by command c, i.e., another subject can be created to fill the role of the destroyed subject. Command e can be similarly compensated by commands a and b. Finally, command f can be compensated by command c.

5 Canonical Schemes

In this Section we show that every MTAM scheme can be converted to an equivalent scheme in which all primitive create operations occur only in unconditional commands. This result is important for the subsequent safety analysis of Section 6. It is also of interest in its own right, because it establishes that there is no need to have conditional creation in access control systems.

To establish this result we introduce the following definition.

Definition 8 A command α is a *creating command* if the **create subject** or **create object** primitive operation occurs in its body. Otherwise α is *non-creating*. An MTAM scheme is in *canonical form* if and only if all creating commands are unconditional. □

We have the following result.

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			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	parent	
$S_3 : cs$			child		
...					

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

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

(d) S_3 acquires read right for O

Figure 2: Illustration of the ORCON Policy in Canonical Form

Theorem 1 Every MTAM scheme has an equivalent scheme in canonical form.

Proof Sketch: Let α be a conditional creating command. We outline a construction to replace α by two commands: a non-conditional creating command α' and a conditional non-creating command α''. (There are many different ways this can be accomplished. We outline only one possibility here.) First, we augment the given MTAM scheme with an additional right that indicates that an object is "alive." Liveness is encoded by presence of the "alive" right in the diagonal cells, i.e., alive $\in [O, O]$ signifies that object O is alive. (This requires that every object is also a subject, which can be assumed without loss of generality by introducing an otherwise empty row for each pure object.) We simulate conditional creation by conditionally controlling the presence of rights indicating liveness. To ensure that objects can participate in MTAM commands only if they are marked as being alive, we augment the conditional expression in each MTAM command to check that all actual parameters are alive. Now, because of monotonicity, we can rearrange the body of α so that all the **create** primitive operations occur before the **enter**'s. We construct α' by (i) using the same parameters as α, (ii) extracting the **create** operations of α into α', and (iii) appending some number of **enter** operations to enter special rights unambiguously binding the parents and children of this command. (We ensure these special "binding" rights can only be introduced by the creating commands.) We construct α'' by (i) using the same parameters as α, (ii) taking the condition part of α and augmenting it with additional tests to check for the binding rights introduced by α', (iii) extracting the **enter** operations of α into α'', and (iv) appending one **enter** operation per child object of α to mark that child as being alive. □

Clearly this construction has linear complexity in the number of commands. Henceforth, we can assume without loss of generality that all MTAM schemes are in canonical form.

To illustrate the intuition behind this construction consider how the ORCON policy of Section 4 can be expressed in canonical form. Looking at the monotonic commands of the ORCON scheme, we see that command c is a conditional creating command. We can replace this command by the following two commands, to make the scheme canonical.

```
(c.1) command create-csubject($S_2 : s, O : co, S_3 : cs$)
          create subject $S_3$ of type $cs$;
          enter parent in $[S_2, S_3]$;
          enter child in $[S_3, O]$
      end

(c.2) command perform-read($S_2 : s, O : co, S_3 : cs$)
        if cread $\in [S_2, O] \wedge$ parent $\in [S_2, S_3] \wedge$
                            child $\in [S_3, O]$ then
          enter read in $[S_3, S_2]$;
      end
```

The scenario of Figure 1 is now played out as shown in Figure 2. Note how the "child" right binds S_3 to O, prior to conditional entry of the "read" right in $[S_3, O]$. The "parent" right similarly binds S_2 to S_3. In this example we did not use the "alive" right, because S_3 in the create-csubject command cannot accomplish anything until the condition of the perform-read command is true.

6 Safety Analysis of MTAM

The general undecidability of safety in MTAM follows from the undecidability results for monotonic HRU [13, 14], which were reviewed in Section 2. In particular safety is undecidable for bi-conditional commands (i.e., commands which have more than one term $r_i \in [X_{s_i}, X_{o_i}]$ in the condition part).

MTAM, however, has decidable safety cases identified on the basis of the types of subjects and objects involved in creation operations. These cases cannot even be formulated in HRU, due to the absence of typing. Our principal result in this Section is that safety is decidable for acyclic MTAM schemes (which are defined in Section 6.1). Acyclic MTAM schemes are sufficient to express most practical monotonic security polices. The safety results, moreover, extend to certain kinds of cycles which then appear to cover all practical needs for monotonic policies.

6.1 Acyclic Schemes

We begin by introducing the following definition.

Definition 9 Let α be a creating command with formal parameters $(X_1 : t_1, X_2 : t_2, \ldots, X_k : t_k)$. We say t_i is a *child type* in α if one of the following primitive operations **create subject** X_i **of type** t_i, or **create object** X_i **of type** t_i occurs in the body of α. Otherwise, we say t_i is a *parent type* in α. □

For example, command c is the only creating command in the ORCON scheme of Section 4. In this command s and co are parent types, whereas cs is a child type.

Note that a type can be a parent type and a child type in the same command. For example, consider the following command:

```
command foo($S_1 : u, S_2 : u, S_3 : v, S_4 : w, O : o$)
        create subject $S_2$ of type $u$;
        create subject $S_3$ of type $v$;
end
```

Here, u is a parent type because of S_1 and a child type because of S_2. Also, w and o are parent types and v is a child type.

This leads us to the following concept of an acyclic scheme.

Definition 10 The *creation graph* of an MTAM scheme is a directed graph with vertex set T and an edge from u to v if and only if there is a creating command in which u is a parent type and v is a child type. A MTAM scheme is *acyclic* if and only if its creation graph is acyclic; otherwise the scheme is said to be *cyclic*. □

The creation graph for the foo command shown above has the following edges: $\{(u, u), (u, v), (w, u), (w, v), (o, u), (o, v)\}$. A scheme which includes this command will be cyclic. The ORCON scheme of Section 4 is acyclic. Its creation graph has the following edges: $\{(s, co), (s, cs), (co, cs)\}$.

6.2 The Unfolding Construction

At a broad level, we adopt the following strategy for safety analysis:

1. Starting with the given initial state, first create as many objects as are necessary to account for the worst-case propagation of access rights. Call this the *unfolded state* of the system.

2. Given the unfolded state, perform all non-creating commands until the state does not change any further. Call this the *maximal state* of the system.

A specific safety question such as, "Can subject S obtain right r for object O?" is then answered by looking at the maximal state and seeing whether or not $r \in [S, O]$ in this worst-case state.

The second step in this procedure is guaranteed to terminate because the unfolded state has a finite number of subjects, objects and rights. Therefore the non-creating monotonic commands will eventually be unable to change the state. The problem lies in the first step where we need some computable criteria to determine when all the necessary create operations have occurred. In other words we need to be able to recognize an unfolded state.

The undecidability results tell us that, in general, it is not possible to construct, or recognize, an unfolded state (even though one is guaranteed to exist in a monotonic system). However, for an acyclic creation graph there is a simple algorithm for constructing the unfolded state. To develop this algorithm we define the following partial ordering on the creating commands.

Definition 11 Let α and β be distinct creating commands in a given scheme. We say $\alpha < \beta$ (α precedes β) if and only if (i) some child type of α is also a parent type of β, or (ii) there is a directed path in the creation graph from a child type of α to some parent type of β. □

For acyclic creation graphs, $<$ is clearly a strict partial ordering (i.e., it is transitive, anti-symmetric and irreflexive).

We have the following unfolding algorithm for constructing the unfolded state from the given initial state. We assume, without loss of generality, that the given MTAM scheme is in canonical form (i.e., all creating commands are unconditional).

Algorithm 1 *Unfolding Algorithm*

1. Linearly order the creating commands consistent with the $<$ relation. (This is the familiar operation of topologically sorting a partial order.)

2. Begin with the given initial state.

3. Proceed down the ordered list of creating commands and apply each command once to each possible tuple of parent objects. □

```
command foo($U : u, V : v$)
        create subject $V$ of type $v$;
        enter parent in $[U, V]$;
end

command bar($U : u, V : v, W : w$)
        create subject $W$ of type $w$;
        enter parent in $[U, W]$;
        enter parent in $[V, W]$;
end
```

(a) Creating Commands with foo < bar

$t(U)$	$=$	u	$\pi(U)$	$=$	U
$t(V_1)$	$=$	v	$\pi(V_1)$	$=$	V_1

(b) Initial State 0 with $OBJ^0 = \{U, V_1\}$

$t(U)$	$=$	u	$\pi(U)$	$=$	U
$t(V_1)$	$=$	v	$\pi(V_1)$	$=$	V_1
$t(V_2)$	$=$	v	$\pi(V_2)$	$=$	$\text{foo}_2(U)$
$t(W_1)$	$=$	w	$\pi(W_1)$	$=$	$\text{bar}_3(U, V_1)$
$t(W_2)$	$=$	w	$\pi(W_2)$	$=$	$\text{bar}_3(U, \text{foo}_2(U))$

(c) Unfolded State u with $OBJ^u = \{U, V_1, V_2, W_1, W_2\}$

Table 4: Example of Unfolding and Pedigree

An example is given in Table 4 (ignore the π function for the moment). Table 4(a) shows two creating commands with foo < bar (because v, the child type of foo is also a parent type of bar). The initial state, shown in Table 4(b), has two objects U and V_1, respectively of type u and v. The unfolded state u, in Table 4(c), is derived from the initial state by the following sequence of commands: foo(U, V_2), bar(U, V_1, W_1) and bar(U, V_2, W_2).

The following lemma is easily proved.

Lemma 1 The unfolding algorithm terminates for acyclic creation graphs.

Proof Sketch: No application of a creating command can result in either that command, or any other command considered before it, being applicable to a previously unconsidered tuple of parent objects. Thus for each creating command, there are a fixed number of applications possible. Since each command is considered only once, the algorithm eventually terminates. □

Note that the unfolded state introduces a possibly exponential number of new objects, relative to the number of objects in the initial state.

6.3 The Pedigree Function

Next, we define the pedigree function for the purpose of relating objects from arbitrary derived states to objects in the unfolded state.

Definition 12 For any derived state h the *pedigree function* π gives the pedigree of every object in OBJ^h, recursively, as follows:

1. The pedigree of an object in the initial state is simply the name of that object, i.e., if $V \in OBJ^0$ then $\pi(V) = V$.
2. The pedigree of an object created subsequent to the initial state, i.e., $V \notin OBJ^0$, is defined as follows: Let V be created as the k^{th} argument in the creating command α with parent arguments $U_1, U_2, \ldots, U_m$ (ordered left to right). Then, $\pi(V) = \alpha_k(\pi(U_1), \pi(U_2), \ldots, \pi(U_m))$ □

An example of the pedigree function is given in Table 4. The equation $\pi(W_2) = \text{bar}_3(U, \text{foo}_2(U))$ is interpreted as follows: W_2 was created as the third parameter of the bar command invoked with U as the first parent and an object with pedigree $\text{foo}_2(U)$ as the second parent.

We have the following property for the pedigree function.

Lemma 2 For acyclic MTAM schemes, for every possible value of the pedigree function there is exactly one object in the unfolded state with that value.

Proof Sketch: In an acyclic scheme there can only be a finite number of generations. By construction the unfolded state includes one representative of each pedigree in each generation. A formal proof can be given by induction on the generation number of a given object defined recursively as follows: the generation number of an object is one more than the maximum of the generation numbers for its parents. For the basis case the generation number of the initial objects is defined as 1. □

In other words the unfolded state contains exactly one object of every possible pedigree. We show below that the object of pedigree x in the unfolded state is able to simulate all objects of pedigree x in any possible derived state.

6.4 Safety for Acyclic Schemes

We are now ready to sketch a proof of the central result of this Section.

Theorem 2 Safety is decidable for systems with acyclic MTAM schemes.

Proof Sketch: Assume without loss of generality that the MTAM scheme is in canonical form, i.e., all creating commands are unconditional (Note that the construction of Theorem 1 does not change the acyclic nature of the scheme). We claim that every history for a given system can be simulated by a history without create operations applied to the unfolded state. More precisely, for an acyclic MTAM scheme for every history H which derives state h from the initial state there exists a history G which derives state g from the fully unfolded state u such that

$$(\forall (S,O) \in SUB^h \times OBJ^h) : [S,O]^h \subseteq [\pi(S), \pi(O)]^g \tag{1}$$

On the basis of equation (1) we have the following algorithm for deciding safety.

1. Use the unfolding algorithm to construct the unfolded state u.
2. Perform all non-creating commands until the state does not change any further. Call this state m.

Due to monotonicity and construction of m it follows that for all states g derived from u by non-creating commands, we have

$$(\forall (S,O) \in SUB^u \times OBJ^u) : [S,O]^g \subseteq [S,O]^m \tag{2}$$

By specializing equations (1) and (2) to SUB^0 and OBJ^0 (i.e., the initial set of subjects and objects) it follows that for all derivable states h we have

$$\begin{aligned}(\forall (S,O) \in SUB^0 \times OBJ^0) &: \\ [S,O]^h \subseteq [\pi(S), \pi(O)]^g &\subseteq [\pi(S), \pi(O)]^m\end{aligned} \tag{3}$$

Since $\pi(S) = S$ and $\pi(O) = O$ in this case, we have

$$(\forall (S,O) \in SUB^0 \times OBJ^0) : [S,O]^h \subseteq [S,O]^m \tag{4}$$

From equation (4) it is clear that the state m allows us to answer all safety questions with respect to the initial set of subjects and objects.

It remains to prove the existence of G with the property asserted in equation (1). G is obtained from H by replacing the individual command of H as follows, while preserving their relative order.

1. Ignore all creating commands in H.

2. Replace every non-creating command $\alpha(X_1 : t_1, X_2 : t_2, \ldots, X_k : t_k)$ in H by $\alpha(\pi(X_1) : t_1, \pi(X_2) : t_2, \ldots, \pi(X_k) : t_k)$ in G.

The proof proceeds by proving the two assertions given below, from which the required equation (1) follows.

- *Assertion 1.* Every command in G is authorized (i.e., its condition part is true before the command is executed).

- *Assertion 2.* $r \in [S, O]^h \Rightarrow r \in [\pi(S), \pi(O)]^g$

These assertions can be proved by induction on the number of non-creating commands in H. For the basis case, assume H consists entirely of unconditional creating commands so G is empty. Assertion 1 is therefore trivially true. Assertion 2 is easily verified from construction of the unfolded state. The induction step follows from the manner in which the non-creating commands of H are simulated in G. □

6.5 Complexity of Safety Analysis

Unfortunately the complexity of safety analysis in MTAM is most likely intractable due to the following result.

Theorem 3 Safety is NP-hard (i.e., no better than NP-complete) for systems with acyclic MTAM schemes.

Proof Sketch: Harrison, Ruzzo, and Ullman [13] have shown that monotonic mono-operational HRU, without creates, has NP-Complete safety analysis. (This is not precisely the result shown in [13]. However, the proof of Theorem 1 and its NP-completeness corollary in [13] essentially establish this fact.) Any model which can subsume monotonic mono-operational HRU therefore has NP-hard safety. □

Clearly the above proof applies to MTAM schemes which have no creation. We therefore have the following corollary.

Corollary 1 Safety is NP-hard for MTAM schemes without creation.

6.6 Safety with Attenuating Loops

For a number of technical reasons, it is important to extend the decidable safety results for acyclic MTAM to include the case of creation with "attenuating loops." A *loop* is a cycle of length one in the creation graph. Loops allow a parent to have children of its own type. In SPM and ESPM the safety algorithm for acyclic creation extends, with minor modifications, to include loops with some additional assumptions (called *attenuating*) regarding rights introduced by loop creations [3, 4, 21]. The intuitive idea behind the attenuating requirement is that the child in a loop creation should be no more powerful than its parent. Attenuating loop creations are required in SPM to implement some of the constructions of [23]. They will be needed in MTAM for similar reasons. Fortunately, a suitable concept of "attenuating" loops can be formulated in MTAM with minor modifications to the safety analysis algorithm for the acyclic case. Due to lack of space this important detail is omitted here.

7 The Ternary MTAM Model

In the previous Section we have seen that although safety is decidable for acyclic MTAM schemes, its complexity is NP-hard (even if creation is completely eliminated). In this Section we show how this negative result can be circumvented by a simple technique, without significant loss of expressive power. Specifically we show that the following model has polynomial time safety for its acyclic cases.

Definition 13 Ternary TAM is identical to TAM except that all commands are limited to three parameters. Ternary MTAM is the monotonic version of ternary TAM. □

We say that ternary MTAM has local authorization for its commands, in the sense that only a small fragment of the access matrix is examined to evaluate the condition part.

Note that the ORCON scheme of Section 4 is ternary. In fact ternary MTAM formally has the same expressive power as MTAM, as shown below.

Theorem 4 Ternary MTAM is equivalent in expressive power to MTAM.

Proof Sketch: The following sequence of reductions give us the desired result: MTAM $\leq$ monotonic HRU $\leq$ ESPM with two-parent creates $\leq$ ternary MTAM. We sketch each reduction in turn from left to right. First, it is clear that any MTAM scheme can be expressed as a monotonic HRU scheme. One way to do this is by encoding the type information of MTAM in the diagonal cells of the access matrix so that

$t_o \in [O, O]$ signifies that object O is of type t_o. (This construction requires that every object is also a subject, which can be assumed without loss of generality by introducing an otherwise empty row for each pure object.) Second, it has been shown by Ammann and Sandhu [2, 4] that any monotonic HRU scheme can be expressed in the ESPM model. Moreover this can be achieved by means of two-parent creates (which have two parents jointly creating one child). Third, with the above encoding of types in the diagonal cells, the copy operation of ESPM can be simulated in ternary MTAM. Therefore ESPM with two-parent creates can be simulated in ternary MTAM. □

Remarkably, in spite of this equivalence of expressive power, we have the following result for ternary MTAM.

Theorem 5 Safety for acyclic ternary MTAM is decidable in polynomial time in the size of the initial access matrix.

Proof Sketch: Note that the canonical form for ternary MTAM obtained by the construction in the proof of Theorem 1 is itself ternary. So we can assume without loss of generality that ternary MTAM schemes are in canonical form (i.e., all creating commands are unconditional). The result is then established by analysis similar to that in [3, 4, 21]. The proof consists of establishing that (i) the unfolded state introduces a polynomial number of new objects (relative to the number of objects in the initial state), and (ii) the number of non-creating command invocations is at most polynomial relative to the number of objects in the unfolded state. □

This result is in sharp contrast to the NP-hard result for acyclic MTAM.

There is no contradiction between Theorems 3, 4 and 5. The implication is simply that the acyclic cases of ternary MTAM and MTAM cannot be co-extensive without having P=NP. The equivalence result of Theorem 4 does not speak about the acyclic cases or about the relative sizes of the simulations.

Binary and Unary MTAM

We conclude this Section with a brief consideration of the relative power of some other variations of MTAM. In analogy to ternary MTAM, binary MTAM limits its commands to two parameters and unary MTAM does so to one parameter. Unary MTAM is obviously too restricted to be of any use. Binary MTAM amounts to either (i) having single parent creation where one parameter creates the other, or (ii) creation of both parameters (Recall there is no concept of who invokes the command in TAM, and therefore spontaneous creation of this kind is meaningful). Spontaneous creation does not affect the result of Ammann, Lipton and Sandhu that multi-parent creation is strictly more powerful than single-parent creation [5] in monotonic models.. Therefore binary MTAM is strictly weaker than ternary MTAM. In fact MTAM with binary creating commands, but without any restrictions on non-creating commands, is strictly weaker than ternary MTAM.

8 Conclusion

In this paper we have defined the typed access matrix model (TAM) by introducing the notion of strong typing into the well-known Harrison, Ruzzo and Ullman model (HRU) [13, 14]. We have shown that monotonic TAM (MTAM) has decidable, but NP-hard, safety for its acyclic creation cases. Further, we have shown that ternary MTAM has polynomial time safety analysis for its acyclic cases, even though it is in general equivalent to MTAM. Ternary MTAM thus has strong safety properties similar to those of Sandhu's Schematic Protection Model [21, 22, 23] and its recent extension by Ammann and Sandhu [2, 3, 4]. The expressive power of ternary MTAM has been shown to be equivalent to MTAM in general.

Our results establish that (i) strong typing is crucial to achieving a useful demarcation between decidable and undecidable safety, and (ii) ternary monotonic commands are critical for tractable safety analysis.

Besides providing rich expressive power and strong safety analysis, a useful security model must also be implementable with a high degree of assurance. In recent work we have considered the implementation of TAM in a distributed environment. Our initial results are promising [24]. Our goal is to find some small set of efficient primitives to implement a simplified form of TAM, which nevertheless theoretically retains TAM's full expressive power.

Acknowledgments

I am indebted to Howard Stainer and Sylvan Pinsky for their support and encouragement, making this work possible.

References

[1] Abrams, M., Heaney, J., King, O., LaPadula, L., Lazear, M. and Olson, Ingrid. "Generalized Framework for Access Control: Toward Prototyping the Orgcon Policy." *NIST-NCSC National Computer Security Conference*, 257-266 (1991).

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

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

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

[5] Ammann, P.E., Lipton, R.J. and Sandhu, R.S. Private communication.

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

[7] Bishop, M. and Snyder, L. "The Transfer of Information and Authority in a Protection System." *7th ACM Symposium on Operating Systems Principles*, 45-54 (1979).

[8] Bishop, M. "Theft of Information in the Take-Grant Protection Model." *Computer Security Foundations Workshop*, 194-218 (1988).

[9] Biskup, J. "Some Variants of the Take-Grant Protection Model." *Information Processing Letters* 19(3):151-156 (1984).

[10] Budd, T.A. "Safety in Grammatical Protection Systems." *International Journal of Computer and Information Sciences* 12(6):413-431 (1983).

[11] Denning, D.E. "A Lattice Model of Secure Information Flow." *Communications of ACM* 19(5):236-243 (1976).

[12] Graubart, R. "On the Need for a Third Form of Access Control." *NIST-NCSC National Computer Security Conference*, 296-303 (1989).

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

[14] Harrison, M.H. and Ruzzo, W.L. "Monotonic Protection Systems." In DeMillo et al (Editors). *Foundations of Secure Computations*. Academic Press (1978).

[15] 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).

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

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

[18] McCollum, C.J., Messing, J.R. and Notargiacomo, L. "Beyond the Pale of MAC and DAC - Defining New Forms of Access Control." *IEEE Symposium on Security and Privacy*, 190-200 (1990).

[19] McLean, J. "A Comment on the 'Basic Security Theorem' of Bell and LaPadula." *Information Processing Letters* 20(2):67-70 (1985).

[20] McLean, J. "Specifying and Modeling Computer Security." *IEEE Computer* 23(1):9-16 (1990).

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

[22] Sandhu, R.S. "Undecidability of the Safety Problem for The Schematic Protection Model with Cyclic Creates." *Journal of Computer and System Sciences*, in press.

[23] Sandhu, R.S. "Expressive Power of the The Schematic Protection Model." *Journal of Computer Security*, in press.

[24] Sandhu, R.S. and Suri, G.S. "Implementation Considerations for the Typed Access Matrix Model in a Distributed Environment." George Mason University, Technical Report, Feb. 1992.

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

[26] Snyder, L. "Theft and Conspiracy in the Take-Grant Model." *Journal of Computer and Systems Sciences* 23(3):337-347 (1981).