Modeling Trusted Processing Architectures for Mandatory Access Control

Thomas H. Hinke

Computer Science Department, University of Alabama in Huntsville, Huntsville, U.S.A.

This paper introduces a trusted architecture graph (TAG) model, which can be used for modeling the semantics of trusted architectures designed to enforce mandatory access control. The TAG permits the modeling of various types of trusted functions, storage functions and processing functions and their interconnection through various types of links. The value of the TAG and the associated TAG notation is that they provide a uniform way of representing different trusted architectures that may be described either informally in a natural language, or formally (but voluminously) described in design documents or programming code. By providing a concise yet expressive description of the architecture, the various features of one architecture can be readily compared with another's. This paper provides some examples that illustrate how various trusted database management system architectures can be formulated in the TAG notation such that their significant differences can be readily observed.

Keywords: Computer Security, Security Architectures Modeling

1. Introduction

This paper describes the semantics and modeling constructs of the trusted architecture graph (TAG), which can be used to provide a high-level description of hardware/software architectures that are trusted to enforce a mandatory access control security policy. The TAG notation provides a uniform notation for describing various trusted architectures that have been presented in the literature. By being able to describe these architectures in a common notation, crucial architecture issues, that distinguish one architecture from another, can be compared and contrasted.

The TAG model is more than a pictorial representation to accompany a textual description of an architecture. The TAG model can be used to describe the top-level security architecture of a trusted system with the TAG constructs taken from a canon of trusted architecture components. The TAG model provides sufficient detail so that fundamentally distinct security architectures can be distinguished from each other. The main architectural features that the TAG notation is intended to describe are the gross division of trusted and untrusted software and the nature of the access control/flow control functions to be provided by the trusted functions. TAG is not used to express the functional capability of the untrusted components other than in natural language labels or description. TAG is also not intended to provide a notation for detailed design. The description of the detailed design is left to other approaches, such as formal specifications or descriptive top-level specifications (National Computer Security Center, 1985).

The remainder of this section will describe the security policy addressed by TAG, introduce trusted system architectures and provide some background on related research. Following the conclusion of this introductory material, Section 2 describes the semantics of the TAG constructs, with Section 3 presenting a graphical notation. Section 4 demonstrates the use of TAG for describing a number of trusted architectures that have been described in the literature and presents the results of an architecture-level security analysis of the various database management system architectures described in TAG. Section 5 concludes with a summary of the contributions and conclusions of this research.

1.1. Mandatory Security Policy

This paper describes the modeling of the hardware and software architectures of systems trusted to enforce a mandatory access policy. The mandatory access policy is one of the two security policies that are commonly recognized within the security community and by such trusted system evaluation groups as the U.S. National Computer Security Center (National Computer Security Center, 1985). The other major policy is the discretionary access policy. One of the reasons for focusing on the mandatory policy is that it is the most critical of the two and the one that affects the selection of an architecture.

Mandatory policy is based on hierarchically ordered security levels such as unclassified < confidential < secret < top secret, and nonhierarchically ordered categories, which may be associated with particular types of intelligence, projects or technology. Together these form an access class. Access classes are related by the dominates relationship. A set of categories A is said to dominate a set of categories B if the set of categories associated with B is a subset of the set of categories associated with the categories of A. An access class X is said to dominate an access class Y if the set of categories associated with X dominates the set of categories associated with Y and the hierarchical component of X is greater than or equal to the hierarchical component of Y. An access class X is said to strictly dominate an access class Y if X dominates Y but is not equal to Y.

A model of a mandatory access policy that has been found acceptable by the U.S. National Computer Security Center is the Bell and LaPadula security model (National Computer Security Center, 1985; Bell & LaPadula, 1976). In this model a subject (which provides the active locus of control) can read an object (which models the passive storage of data or programs) if the access class of the subject dominates the access class of the object. However, to prevent the inadvertent or intentional (via malicious code) writing down of sensitive data, the Bell and LaPadula model requires that a subject can write into an object only if the access class of the object dominates the access class of the subject. This means that a subject cannot write down in security level. This prevents malicious code that may exist within programs being executed by a user from writing data into an object where it could be read by some subject not cleared to access the original data. This no-write-down property is called the *-property. For the purpose of describing the TAG model the Bell–LaPadula model will be used as our model of a mandatory access policy.

It should be noted that the Bell–LaPadula model is primarily concerned with unauthorized disclosure of data, although limited protection against unauthorized modification is provided by write controls. The model does not address the prevention of unauthorized denial of service. In keeping with the disclosure nature of current security architectures, this paper will focus primarily on the disclosure issues in the architecture.

1.2. Security Architectures

A fundamental mechanism that should be present in all security architectures that purport to provide security with a high level of assurance is the security reference monitor (Anderson, 1972). A security reference monitor must satisfy the following three conditions:

1. It must be invoked on every access by a subject to an object;
2. It must be protected from unauthorized modification, and
3. It must correctly enforce a desired security policy.

Systems are called "trusted" when the combination of their software and hardware architectures are trusted to enforce a desired security policy, such as Bell and LaPadula. A trusted system is designed with a combination of trusted and untrusted software. Depending upon its role in the system, software is called "trusted" if it satisfies one or both of the the following criteria:

1. It is trusted to enforce the desired security policy, and
2. It is trusted to be nonmalicious

In most trusted architectures, trusted software enforces the mandatory and discretionary security policy. The trusted software associated with a trusted system architecture is called the trusted computing base (TCB) (National Computer Security Center, 1985). Software that does not enforce a security policy, but is non-malicious, can be used to enhance some security architectures, as is noted in the crypto-seal database example that will be presented in this paper.

The basis for trust in a system is provided by a combination of structuring the architecture to minimize the amount of required trusted code and ensuring that the required trusted code has appropriate assurance technology applied to it to validate its trust. This assurance technology can range from good software engineering practices to formal specification and verification techniques. In contrast, untrusted software can be assumed to perform any activity — authorized or unauthorized — with the effect of the activity constrained by the system architecture so that no unauthorized disclosure will result.

The third component of a trusted architecture is hardware, which is normally assumed to be trusted at least to the extent that trusted software is trusted. The evaluation of hardware trust is beyond the current trusted system evaluation criteria (National Computer Security Center, 1985).

Because the development of trusted software can add a significant amount of additional assurance requirements and cost beyond that associated with untrusted software, a strong goal in the development of trusted systems is to minimize the amount of trusted software. The objective is to use a small amount of trusted code to provide the trust for a large amount of untrusted code. One of the techniques used to permit trusted code to control large bodies of untrusted code is to use multiple instantiations of untrusted code for each subject or security level. This concept was first applied to secure database management systems by Hinke and Schaefer in a design which based all of the DBMS security on the operating system security kernel, leaving the DBMS code with no security enforcement role for mandatory security (Hinke & Schaefer, 1975; Woods Hole Summer Study, 1983). This multiple instantiation approach was continued in the SeaView secure DBMS prototype for mandatory enforcement, but with the modification that the DBMS code provided discretionary security enforement. (Lunt & Hsieh, 1990; Lunt & Boucher, 1994) By replicating the untrusted software among subjects and ensuring that there is no communication between subjects except as provided by the trusted software, the untrusted software cannot serve as a conduit of data flow between different subjects (Pfleeger, 1989). The next section will describe previous work that is related to modeling trusted systems.

1.3. Previous Work

The first known security model was proposed by Weissman (Weissman, 1969) and modeled only the security policy to be enforced in a system. This work was followed by contributions of the reference monitor and the mandatory policy of Bell and Lapadula which have already been noted. These form the basis for the U.S. Department of Defense Orange Book (National Computer Security Center, 1985). TAG is intended to model reference-monitor-based architectures that enforce a Bell and LaPadula-type security policy.

Early examples of architectural modeling were presented in the Woods Hole report on secure database management systems (Woods Hole Summer Study, 1983) and the secure database management system design methodology work of Hinke (Hinke, 1986). In these early papers, various secure architectures were represented by boxes. However, there are little or no semantics attached to the boxes.

TAG corrects this deficiency by providing graph-based modeling concepts that provide security-oriented semantics. While there have been many such graph-based semantic models such as the ER-model of Chen's (Chen, 1976) or conceptual graph models of Sowa, based on the work of the 19th-century work of Charles Peirce (Sowa, 1984), none of this work was oriented toward security semantic modeling. A security orientation for the modeling of data was provided by Smith (Smith, 1990), who developed a graph-based language for representing the security semantics of fine-grained data such as would be stored in a secure database management system (DBMS), but this work did not address architecture modeling. TAG is unique

in its orientation to the modeling of security architectures.

2. Trusted Architecture Model

The TAG model uses the following six types of objects: storage entities (SE), processing entities (PE), processing environments (PEV), ports (P), links (L), and various types of flow controllers (FC). These objects are combined to represent a TAG model of an architecture. At the highest level of abstraction, a trusted architecture is represented as a directed graph G = (N, L), where N is a set of nodes and L is a set of links. The nodes represent processing entities, storage entities and various types of trusted flow controllers. The links provide the connections between the various entities. Each of these constructs will be explained in the following paragraphs.

2.1. Storage Entity

The storage entity represents the passive objects that provide persistent storage of data. In the TAG model, storage entities are not assumed to have any processing capability. Any processing that is required to retrieve data from the SE is assumed to be provided by a processing entity or one of the trusted-flow controllers. An SE is characterized by whether or not it contains labeled data and the nature of the labels. An SE that contains only unlabeled data is termed an UD (unlabeled-data) SE, while one that contains only labeled data is termed an LD (labeled-data) SE. If the labeled data is crypto-sealed, then this is so indicated by the TAG notion, which will be described shortly.

2.2. Processing Entity

A processing entity models the active system objects that perform the execution of instructions. It can be a single process, a single processor or a system comprised of multiple processors. As will be noted, a PE can be constructed from other PE's. When not otherwise indicated, the PE will be composed of untrusted code. If the PE is to perform a trusted function, then the trusted function will be identified as a trusted flow controller.

A processing entity can be formally modeled as a graph PE = (N, L), where N is a set of nodes and L is a set of links connecting the nodes. The set of nodes is drawn from the set of objects {PEV, P, FC }, where PEV represents a set of processing environments; P represents a set of ports associated with the processing entity, and FC represents a flow controller.

The processing entity can be characterized by the nature of both its internal flow policy and of its input and output data flow. To characterize the dataflow policy, we assume that the PE is partitioned into one or more processing environments and describe the policy in terms of the flow of data between processing environments.

Three internal flow policies are possible: unconstrained (UF), constrained (CF) and no flow (NF). Unconstrained flow means that there is no security reference monitor to mediate the flow of data between processing environments; hence the PE provides no security protection. In effect, this is equivalent to a PE's having only a single processing environment.

The constrained-flow system permits the flow of data between different processing environments, but only under controlled conditions that satisfy some security policy. To support a constrained-flow policy would require that a trusted flow controller mediate all data flow between the various processing environments and ports within the processing entity. A constrained-flow processing entity is capable of supporting the controlled sharing of data of different sensitivities between uses with different access privileges. The no-flow system carries the constrained-flow to the limit by completely restricting the flow of data between processing environments.

The nature of the PE's input and output dataflow can be characterized in terms of the dataflow supported by its ports, where all dataflow into and out of a PE is assumed to flow through a port. If the PE is to process data that is classified at a single security level, data can be input through a single unlabeled-flow input (SUI) port. If the PE is to support the processing of data at multiple security levels, there are two input options: use multiple unlabeled-flow input (MUI) ports or use a single multi-level labeled-flow input (MLI) port. In a similar way, the output of a PE can be character-

ized by whether all output flows through a single unlabeled-flow output (SUO) port, multiple unlabeled-flow output (MUO) ports or a single multilevel labeled-flow output (MLO) port. When it is not important to identify the precise realization of the multilevel input mechanisms, the multilevel input can be indicated as M*I, where "*"is a "wild card" that could be a "U" or an "L". Similarly, multilevel output can be indicated as M*O, where the precise nature of the output ports (MUO or MLO) is not pertinent.

This section has described those characteristics of a PE that are seen by its users, but not how the PE is designed internally to provide such characteristics. These external characteristics can be provided through a number of different internal architectures. The basic building blocks of these internal architectures are the processing environment, along with a number of other components, which include various types of trusted components, a storage entity and various types of links to connect these components. Each of these will be described in the following sections.

2.3. Processing Environment

A processing environment represents a program execution environment in which multiple processing threads could exist simultaneously. All data and programs within a particular processing environment are accessible to all of the processing threads that exist within the same environment. Access by a processing thread within one processing environment to programs or data within another processing environment depends upon the nature of the path joining the two, where a path is comprised of single links or combinations of nodes and links. If two processing environments are directly connected by a link then they form a single processing environment, since links do not have any ability to enforce an access policy. If however, all paths between two processing environments include a trusted flow controller, the nature of the access will depend upon the specific type of trusted flow controller that is used. If a processing environment is to support the enforcement of a security policy based on a reference monitor, then all processing environments should be connected to a trusted flow controller appropriate for the desired policy that is to be enforced.

Since a processing environment provides at least one execution thread, it can also be considered a processing entity. However, we use the term "processing environment" rather than "processing entity" when we are describing an internal data flow property of an architecture, since it is the access class of the processing environment that is critical, rather than the processing capability of the entity.

2.4. Ports

A port represents the termination of links at processing entities, flow controllers and processing environments. Ports are assigned an access class that determines the highest access class of data that is permitted to move through the port.

Two types of ports are possible: unlabeled- and labeled-flow. The unlabeled-flow port provides the PE with the ability to input or output a stream of data that is regarded as uniformly classified at a single level — it contains no labels. Since there are no security-relevant labels, this type of port requires no trusted software to append or interpret security labels.

In contrast, the labeled-flow port is capable of handling data that contains security labels as part of the flow. This port preserves the actual classification of the data, thus avoiding the need to classify all data within the flow at the highest access class of the flow. For input, the port must have trusted software for interpreting the security labels so that a trusted flow controller is able to route the data properly. For output, the port must have trusted software to label all data. In addition — for both input and output — the port's trusted software must prevent the intermixing of data with different security levels.

2.5. Links

Links are used to interconnect storage entities, processing entities, processing environments, ports and flow controllers. The links are characterized by three attributes: direction, label characteristics and level composition. The direction of the link indicates the direction of the data flow that it supports. A link can be unidirectional or bidirectional. The labeling characteristics indicate whether the data that flows

through the link is labeled with a security label or unlabeled.

The level composition gives an indication of the security levels that provided the original source for the data within the flow. If the original source of the data was single level, then the current level composition will also be single level and indicated with by the "SL" notation. If, however, the flow originated from multilevel sources but does not contain labels, then the data flow is considered to be system-high data, indicated by the "SH" notation. All of this system-high data must, however, be considered to be classified at a security level that dominates all of the data in the flow since (due to lack of security levels) it can no longer be separated into its original security levels.

If the data flow is labeled and originated from multiple security levels, then it is a multilevel flow and indicated with the "ML" notation. With a multilevel flow it is possible to differentiate between data of different access classes and recover the original access class labels of the data.

The final type of link is the trusted link, which carries data whose security semantics must be preserved. It is more general than the multilevel link, since the multilevel link is assumed to contain only labels which must be preserved. For example, a trusted link would permit the transfer of access control data, whose unauthorized modifications could change a subject's access privileges, granting it a higher level of access privilege than was desired. The trusted link can be shown in two different ways. The first is to show an explicit trusted link. The second is to draw the components physically touching each other, but leaving out the explicit link. This is the approach used for the TAG architectures shown in this paper.

2.6. Flow Controller

The flow controller mediates the flow of data between the various processing environments or between ports and flow controllers. All but the degenerate null flow controller are trusted. The trusted flow controller serves as a security reference monitor (Anderson, 1972). The trusted flow controller (TFC) represents a component that can impose various types of restrictions on the flow of data passing through it. The TFC would be implemented with trusted code.

The semantics of the various type ot TFC's can be specified by the nature of the flow policy that they enforce. The flow policy can be characterized in terms of the access class (AC) of various combinations of six flow attributes. The first four flow attributes consist of processing environments that play various roles in data flow. These include the source of the data (source), destination of the data (destination) and the initiator of the request for data (initiator). The fourth role identifies those processing environments that retain any persistent memory of the request for data (memory). The fifth flow attribute is the port through which the data passes. The ports can be distinguished as input (p_in), output (p_out) or just (port) if there is no need to make a distinction. The final flow attribute is the labeled container that holds the data within a multilevel link or entity (container). While most of these attributes are self-explanatory, a few words need to be said about some of these.

The container models the ability of data flows (links) to contain data at multiple access classes. Different containers within the data flow contain data at different levels, each labeled with the access class of the data that it contains.

The memory attribute is used to represent the possibility that a particular flow control policy may or may not leave some residue of a data request at the data source. If residue is left, then that represents a potential covert channel (an unintended channel used to leak data in violation of mandatory policy), since the existence of the residue alone could provide notification that a dominant processing environment made a data request. If the content of this residue or the timing of the appearance of a request can be modulated, then information can potentially be transferred from the processing environment of the initiator to the processing environment of the source of the data. If the initiator's processing environment strictly dominates the source's processing environment, then we have a potential breach of security if Trojan horse code (malicious code whose purpose is to surreptitiously leak data out of the processing environment, while at the same time performing some legitimate functions to mask its malicious function) can leak information from the dominant processing environment using Trojan horse code.

With these attributes we can now describe and formally characterize the semantics of each of the flow controllers to be described in the following sections.

Identification/Authentication (I&A)

The identification and authentication (I&A) TFC is used to identify and authenticate the processing entities that terminate the various links that may be connected to a PE. The I&A component can be characterized in terms of its subject granularity and security scope. The subject granularity options include the complete processing entity (E) and/or the people who are users of that processing entity (P). If, for example, system "A" performs E granularity authentication of system "B", this means while "A" can authenticate the "B" system, the "A" authentication system is not capable of distinguishing between the various users of the "B" system.

The security scope of the authentication can include just the identity of the entity and/or person, or the authorized security level of the entity and/or person. Identity authentication is termed "I" authentication and level authentication is termed "L" authentication.

The I&A TFC can be specified by indicating the granularity and scope of the authentication that is performed using the notation P(wx)E(yz), where wxyz are I or L or null. If an I&A controller supports the authentication of the identity and security level of both individual users and the processing entity from which they are performing the access, this is indicated as I&A = P{IL}E{IL}.

For initiators emanating from single level PE's, the I&A TFC validates that the AC(initiator) = AC(port), where (port) is the port assigned to an authenticated link. For initiator requests that come through trusted flow controllers, the I&A TFC must ensure that the access class of the port associated with the link is within the permitted access class range of the requesting flow controller.

One-Way Flow (OWF)

The one-way-flow (OWF) TFC is used to restrict the flow of data and associated control information to a single direction. The restriction on the flow in the opposite direction is absolute — no data or control information is permitted to flow in a direction opposite that permitted by the OWF TFC.

The OWF TFC is characterized by the following security constraints: AC(source) $<$ AC(destination) AND AC(memory) $\geq$ AC(source) AND AC(initiator) = AC(source).

Read Down/No Write Down (RDNWD)

The read down/no write down (RDNWD) TFC permits a processing environment to read data that is at a lower access class than the current operating access class of the processing environment. This read must be performed by software or hardware that is trusted not to leak data from the processing environment that initiated the read request to any other untrusted processing environment.

For software to support the RDNWD function, it must meet one of the following two criteria. The first is that it must be trusted in the sense that it has undergone the necessary assurance, as defined by the cognizant evaluation organization, to ensure that it correctly enforces the desired security constraint (i.e., not writing down in the course of performing the read). If software does not satisfy the first criterium it must satisfy the second.

The second criterium is that it must be encapsulated by trusted software. This latter approach, suggested by Sandhu (Thomas & Sandhu, 1992) for object-oriented systems, is to have the read performed by untrusted software that is encapsulated and memoryless. It must be encapsulated so that it cannot leak data to other untrusted software, that could ultimately leak data to users within the lower processing environment. It must be memoryless so that data from previous requests is not leaked in the results to subsequent requests.

The RDNWD TFC is characterized by the following security constraints: AC(initiator) $>$ AC(source) AND AC(source) $<$ AC(destination) AND AC(memory) $\geq$ AC(initiator).

No-Flow (NF)

The no-flow (NF) TFC is used to totally restrict the flow of data between two processing environments. This models the separation between processing environments within the NF type of PE described previously.

The NF TFC is characterized by the following constraints: AC(source) = AC(destination) = AC(memory) = AC(initiator).

Unrestricted Flow (UF)

The unrestricted-flow (UF) TFC permits the unrestricted flow of data in either direction. However, it is trusted to handle multiple flows and not intermix them. The purpose of the UF trusted flow controller is to serve as an interface between a processing entity and one or more links.

The UF TFC is characterized by the following constraint: AC(container_in) = AC(container_out).

Flow Reducer (FR)

The flow-reducer (FR) TFC is a variation of the unrestricted-flow TFC in that flow is permitted in both directions. However, the flow through the FR is unbalanced, in that the flow in the preferred direction is unrestricted, while the flow in the attenuated direction is highly restricted. However, in contrast to the OWF controller, the flow even in the attenuated direction is not attenuated to zero. It can still be used to forward control information, and thus can be used to leak unauthorized data from one processing environment to another.

The FR TFC is characterized by the following security constraints: AC(initiator) > AC(source) AND AC(source) < AC(destination) AND AC(memory) $\geq$ AC(source). As can be seen, this characterization is identical to that of the RDNWD TFC, except that the memory of the transfer can exist at the access class of the source.

Flow Splitter

The trusted-flow splitter (TFS) provides the interface between multiple, unlabeled data flows and a single labeled data flow. In one direction, the TFS is trusted to transform one or more streams of unlabeled data into a stream of labeled data. In the opposite direction, the TFS is trusted to transform a stream of labeled data into one or more streams of unlabeled data. The TFS thus serves as a multiplexer/demultiplexer.

The trusted flow splitter (TFS) can be modeled as a component that provides one of the following combinations of multiple unlabeled flows and a single labeled flow, modeled as a set of various types of ports:

1. Unidirectional TFS that transforms a labeled flow into multiple unlabeled flows: (MLI $\rightarrow$ MUO).
2. Unidirectional TFS that transforms multiple unlabeled flows into a single labeled flow: (MUI $\rightarrow$ MLO).
3. Bidirectional TFS that provides bidirectional flows between a single labeled flow and multiple unlabeled flows: ((MLI, MLO) $\leftrightarrow$ (MUI, MUO)).

Since flow splitters have different constraints, depending upon the direction of the flow, each of the types will be characterized with a different set of constraints. The unidirectional flow splitter (MLI $\rightarrow$ MUO) is characterized by the following constraints: AC(container_in) = AC(port_out) AND AC(initiator) = AC(container_in). The unidirectional flow splitter (MUI $\rightarrow$ MLO) is characterized by the following constraints: AC(port_in) = AC(container_out) AND AC(initiator) = AC(port_in). The bidirectional TFS is just the combination of these two.

Crypto-sealer

For data input, the crypto-sealer TFC computes and appends a crypto-seal (Gifford, 1982; Pfleeger, 1989) to the data. The data itself is not encrypted, but used by the cryptographic algorithm only to compute a cryptographic checksum or crypto-seal. One approach to computing a cryptographic checksum is to use an encryption algorithm, such as the Data Encryption

Standard (National Bureau of Standards, 1977) in cipher feedback mode (National Bureau of Standards, 1980; Pfleeger, 1989). Under this mode of operation, a 64-bit register is encrypted during each cycle. The least significant byte of the resulting 64-bit result of the encryption is exclusively OR'ed with the next byte of the text to be encrypted and the single byte output as an encrypted byte. This single byte is also shifted into the least significant byte of the register, with the most significant byte of the register discarded. The encryption cycle is then executed again on the 64-bit register. The 64-bit result from each encryption cycle is a function of the encryption algorithm used, the encryption key and all of the previous data. The result from the final cycle can thus serve as a cryptographic checksum, since it is a function of all of the data that has been processed.

While both a normal checksum, a cyclical redundancy check (CRC) and the cryptographic algorithm can perform similar functions in detecting data modification and may use known algorithms, the contrast lies in the fact that the cryptographic algorithm also uses a key that can be kept secret. Thus, in contrast to a normal checksum or CRC, the adversary or his program is unable to mimic the computation.

For data output from a crypto-seal protected system, the crypto-sealer computes the cryptographic checksum of the data that is flowing through it and compares this newly computed checksum with the one that accompanies the data to be output. If the newly computed cryptographic checksum matches the checksum already associated with the data, then the data is permitted to pass. Otherwise, the crypto-sealer acts as a no-flow controller by preventing the data from flowing through the controller.

The crypto-sealer TFC is characterized by the following constraints: AC(source) = AC(container).

3. TAG Modeling Notation

Figure 1 shows the symbols and associated annotations that comprise the TAG modeling notation. These symbols fall into four classes: processing entities, trusted flow controllers, storage entities and links. This notation will be described briefly in this section.

The general processing entities are indicated by plain rectangles with the notation "PE" below the rectangle. The internal data flow policy supported by the PE is indicated by a line that divides the PE into two representative processing environments in the case of the constrained-flow and no-flow PE and the absence of such a line in the unconstrained-flow PE. As shown in the no-flow example, the nature of the data processed in each processing environment can be optionally included. In this example, the upper processing environment has access to system-high data, while the lower processing environment has access to only system-low data. These annotations could be replaced with particular security level labels if desired.

The figure shows various types of trusted-flow controllers. The trust that differentiates these trusted components from untrusted TAG components is indicated by the shading. The control nature of these components is indicated by the dashed line that divides the symbol. The type of symbol is indicated by the label that sits below the symbol, although as can be noted, the annotation is not normally required, since each symbol is unique. Only in the case where an identification and authentication flow controller is coupled with an unrestricted flow controller to provide both authentication and data separation and routing will the annotation normally be included.

The three types of storage entities are shown. The SE annotation below the symbol should be included to differentiate the storage entity from the unconstrained flow processing entity, since the UD notation might be mistaken for a security level label.

Links are indicated with directed lines indicating direction of flow, and with circles indicating the labeling characteristics and access class composition of the data flowing through the link. Examples are shown for various types of links including trusted links (which do not carry a level composition indication for this example, but could); untrusted, system-high; untrusted, system-low; and labeled, multilevel.

The next section presents some examples of TAG descriptions for a number of trusted systems.

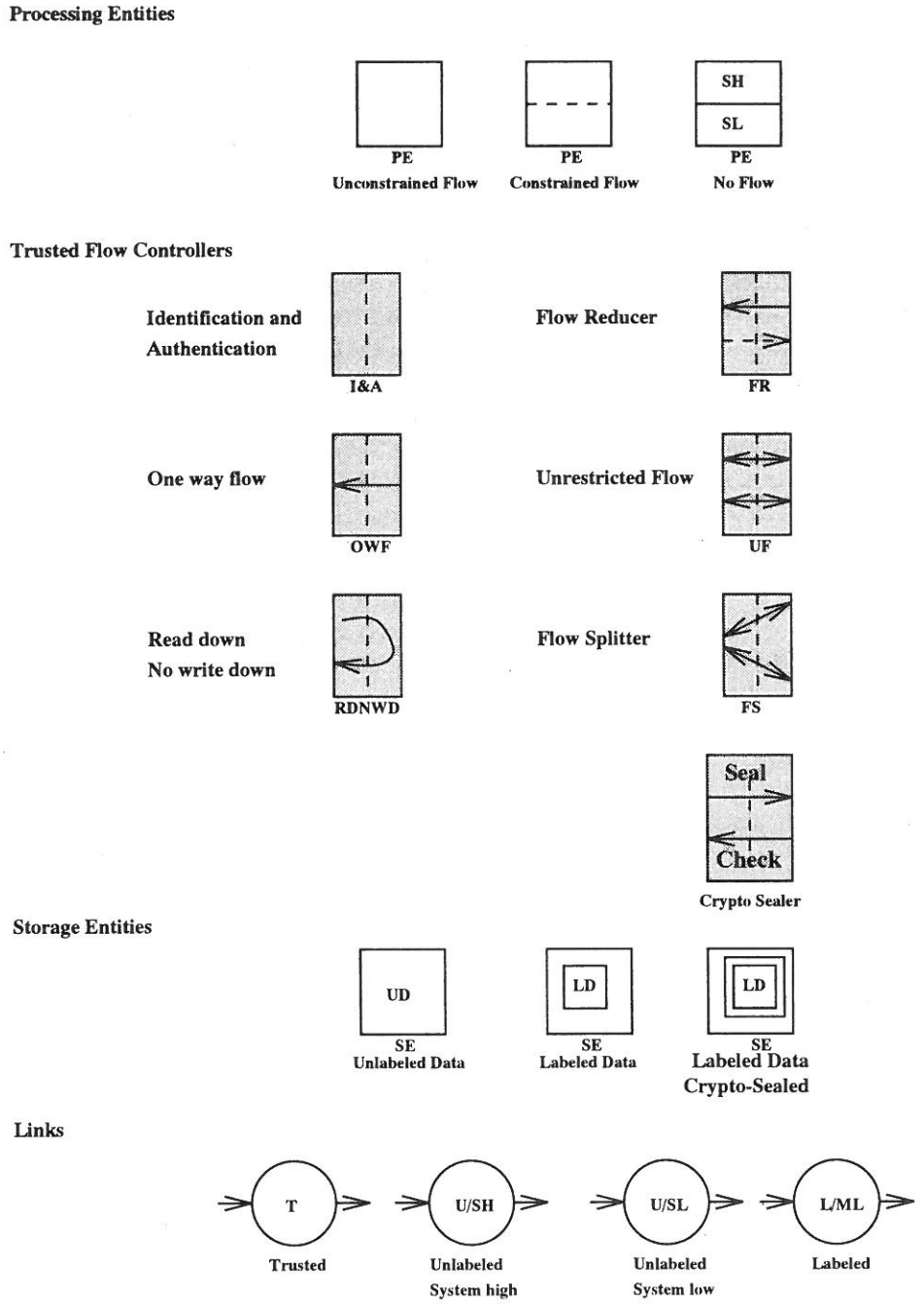

Fig. 1. TAG Graphical Notation

4. TAG Examples

This section shows a number of examples of trusted architectures specified in the TAG notation. The first one shown is the one-way guard, and the remaining examples are various trusted database management system architectures.

For each of the database management systems, the results of an architecture-level security analysis that was conducted on the TAG representation is presented. A security vulnerability that exists at the architecture level will exist within an implementation. However, it is the case that a good security architecture can be compromised with a poor implementation. The security analysis techniques being used here are based on the work of (Hinke, 1986). This work categorized the various architectural-level security vulnerabilities into the following four classes:

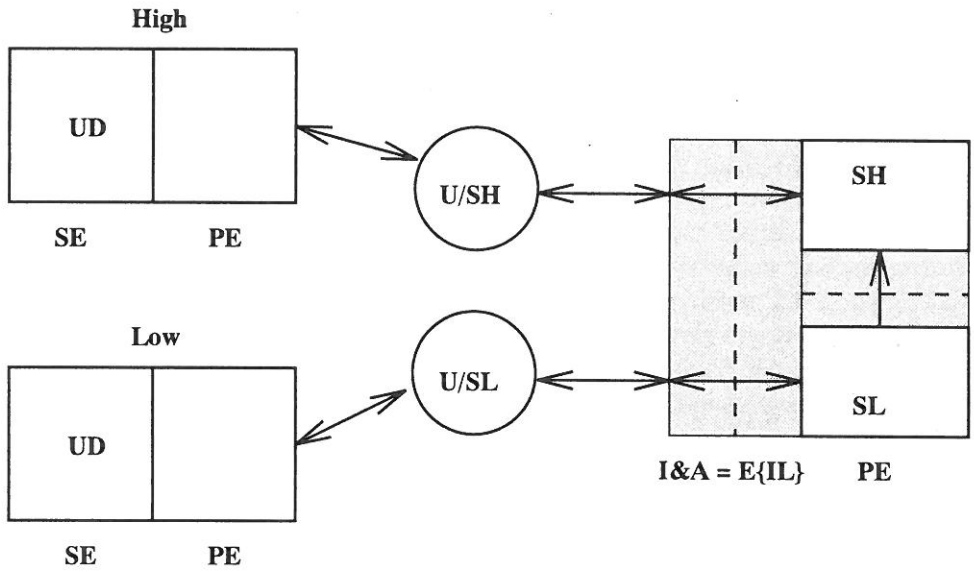

Fig. 2. One-Way Guard

Vulnerability 1 – Grants Unauthorized Request: A system permits a legitimate user to be granted access to requested data to which he is not authorized access. This vulnerability violates the basic security condition aspect of the Bell and LaPadula security policy model.

Vulnerability 2 – Direct Write Down: A system permits data to be written in a port, container, storage entity or processing entity whose access class is strictly dominated by the access class of the data. This is a violation of the Bell and LaPadula *-property.

Vulnerability 3 – Internal Covert Channel: – A System permits the leakage of data from within the system to processing environments whose access class is strictly dominated by the access class of the data using an architectural-level covert channel.

Vulnerability 4 – System as Covert Channel: The system itself can be used as a covert channel so that data can be leaked between external systems using the trusted architecture.

The vulnerabilities are hierarchically ordered in the sense, that the lower the number, the more severe the vulnerability in terms of the amount of data that could be compromised. In the analysis which follows, the DBMS architectures will be characterized by the lowest numbered vulnerability that applies.

4.1. One-Way Guard

This system is analogous to an electronic diode applied to the data flow within a computer system. Figure 2 shows the graphical representation of such a guard that allows data to flow from a system-low PE to a system-high PE. Both of these PE's include storage as indicated by their SE component. Both of these PE's and associated SE's could represent workstations or hosts on a network.

The guard itself consists of two unconstrained flow PE's (one processing system-high data and the other processing system-low data) connected to each other by a one-way flow TFC. The PE's are connected to their respective workstations through an identification and authentication TFC. As indicated, this trusted component also includes an unrestricted-flow TFC, which serves to provide data stream separation and routing for moving the data from the guard input to the respective guard PE's. As a complete system, the guard itself would be described as a controlled-flow PE with multiple unlabeled input and output ports.

4.2. Label-Checking Filter DBMS

The label-checking filter DBMS, which could also be called a label-checking guard DBMS, is one of the simplest security mechanisms to provide a limited amount of security to a DBMS. The label-checking filter serves as a trusted mediator between users and a DBMS. While the untrusted DBMS contains sensitive data classified at different security levels, it does not provide any security mediation itself. All of the security mediation is provided by the filter. An example of such a filter approach is the query modification approach used for Ingres (Stonebraker & Wong, 1974) and ACCAT and FORSCOM guards (Soleglad, 1981).

A TAG model of the label-checking filter DBMS is shown in Figure 3. This architecture is characterized by a trusted-flow controller that provides for identification and authentication of both the identity and level of the users attempting to access the system. The access control function for the data itself is performed by a trusted flow-controller that mediates the query prior to passing it on to the DBMS. This means that this architecture can successfully counter Vulnerability 1.

The primary disadvantage of this approach is that the trusted-flow controller is not capable of identifying the security level of the returned data. Hence, if the untrusted DBMS code attempted to output data whose access class was higher than the subject making the request, the trusted-flow controller would not be able to detect this intentional or accidental error. This can be seen in the fact that the TAG model shows an untrusted PE between the SE and the trusted-flow controller. This means that this architecture is subject to Vulnerability 2. This vulnerability is remedied in the next architecture.

4.3. Crypto-sealing Filter DBMS

The crypto-sealing data filter DBMS has an architecture similar to the basic filter DBMS in that it sits between the users and the untrusted DBMS. It is identical to the basic filter DBMS in the way that it performs request mediation. It differs from the basic filter DBMS in that it performs mediation of the response as well as the request.

The response mediation is based on the data and the labeling being indelibly connected via a cryptographic checksum which is computed when data enters the system (Denning, 1984). The internal structure of the crypto-sealing filter is shown in Figure 4.

This architecture is not subject to Vulnerability 1, since the trusted flow controller can mediate requests. Also, since the trusted-flow controller can now validate the access class of the output, this means that this architecture can successfully counter Vulnerability 2.

This architecture is subject to Vulnerability 3, since the untrusted system-high DBMS PE has access to both high and low data and can output data to both high and low PE's. Untrusted Trojan horse code within the DBMS could encode high data that it can view in terms of a code expressed in terms of low data that it can forward to a low PE. However, if the untrusted DBMS PE could be considered non-malicious, based on it having been developed by appropriately cleared personnel, then this architecture

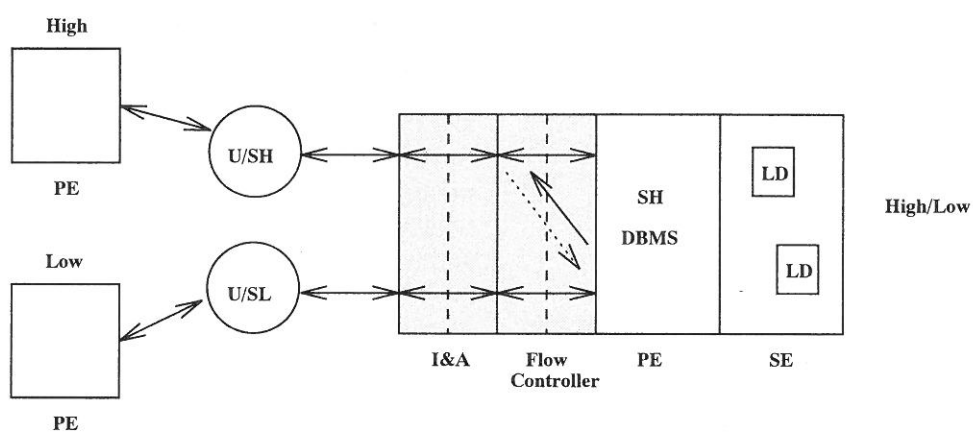

Fig. 3. Label Checking Filter

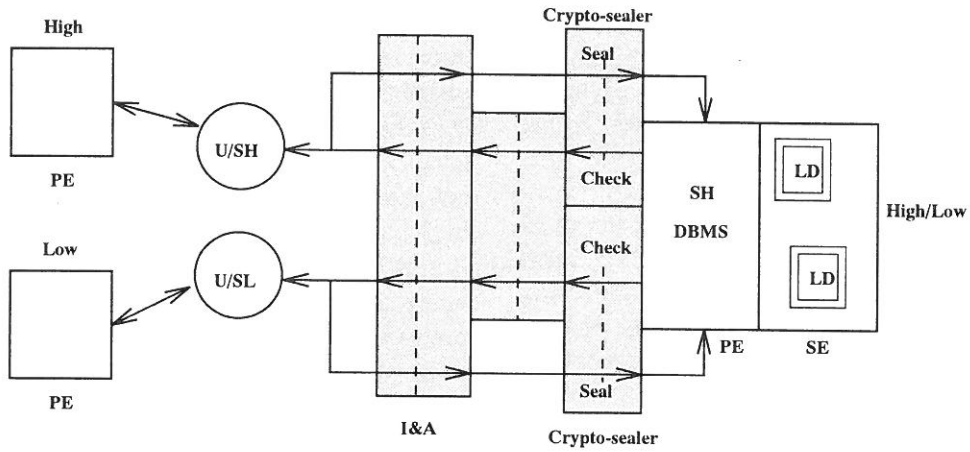

Fig. 4. Crypto-Sealing Filter

would not be open to vulnerability 3, since it would contain no Trojan horse code to perform the encoding.

4.4. Disjoint Data — Multilevel Secure Database Management System

This backend architecture includes a trusted, multilevel frontend processor and multiple, untrusted single-level backend processors. An example of this architecture is the Unisys SD-DBMS system design (McCollum & Notargiacomo, 1991). The disjoint data backend architecture is shown in Figure 5.

In this architecture, each backend processor has access to only a single level of data, thus requiring the frontend processor to query multiple backend processors to satisfy a multilevel query. What characterizes this architecture is the use of a flow reducer in the trusted frontend processor to permit a query to be sent from the system-high processing environment of the trusted frontend to the low backend processor. Requests for a single level of data are handled by the untrusted backend processor of the appropriate access class. A significant advantage of this architecture is that the backend processors can be implemented with commercial of-the-shelf processors, since they support only an unconstrained-flow policy.

The disjoint data backend architecture is not susceptible to Vulnerability 1, since it has a trusted flow controller to validate requests. Since the backend PE's cannot write down to lower level requesting PE's (workstations), the system is not subject to Vulnerability 2. This architecture can also resist Vulnerability 3, since backend PE's cannot view data whose access class strictly dominates the access class of the output of the PE. This architecture is, however, susceptible to Vulnerability 4. This can be detected from the TAG representation by noting the U/SH link from the TFC to the low backend processing entity/storage entity. This U/SH link is carrying the request to a processing entity and associated storage entity that has memory. This request could have been initiated at a system-high workstation (frontend PE). In this case, the result would be that data originating at a high processing entity has reached a low backend processing entity/storage entity. If this backend PE contained Trojan horse code, it could proceed to forward the data to the low frontend processing entity. This is possible since once data reaches a backend processing entity, there is no way for the TFC to validate that the data provided by a low access class backend is anything but low. The possibility that the low backend PE may become contaminated with high data is indicated by the "-h" annotation of the link security levels emanating from this low backend.

4.5. Replicated Data — Multilevel Secure Database Management System

A contrast to the disjoint data multilevel secure DBMS is the replicated data multilevel secure DBMS. The SINTRA system developed by the U.S. Naval Research Laboratory is an example of a replicated data DBMS (Kang & et al., 1994; Frosher & Meadows, 1990).

Both the replicated and disjoint data architectures represent backend DBMS architectures, but as the TAG representation indicates, these are significantly different architectures. In the replicated data, backend DBMS architecture, shown in Figure 6, each backend processor contains all of the data that is dominated by the security level of that processor. Thus, for example, an unclassified backend processor would have only unclassified data, but a top secret processor would contain top secret, secret, confidential and unclassified data.

To support this proliferation of data copies, a one-way data flow must be provided from the low to the high processing environment. It is important for the reader to recognize the somewhat subtle difference between a one-way capability from low to high and a read-down capability from high to low. In the one-way, low-to-high data flow, the low processing environment initiates the transfer when its data changes. The high processing environment is not capable of making a request to initiate the data flow. However, in the read-down capability, it is the high processing environment that must be capable of initiating the data flow. Since this requires the cooperation of software in the low processing environment, this requires a data flow from the high to the low processing environment, which is not provided in this architecture, but must be provided in the architecture described in the previous section. Thus in contrast to the disjoint data architecture, this architecture does not have any of the write-down vulnerabilities of the previous architecture. It thus provides all of the vulnerability-countering capabilities of the previous architecture and also counters Vulnerability 4, since there is no downward flow of data.

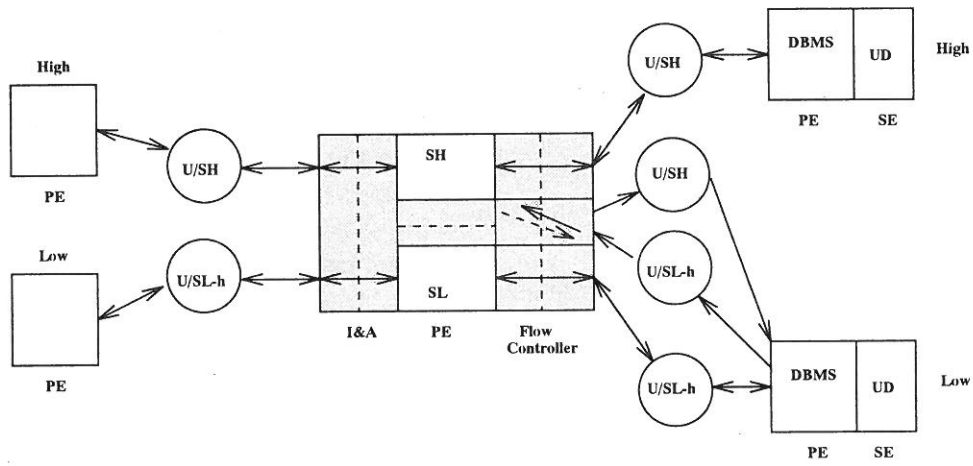

Fig. 5. Disjoint Data Backend

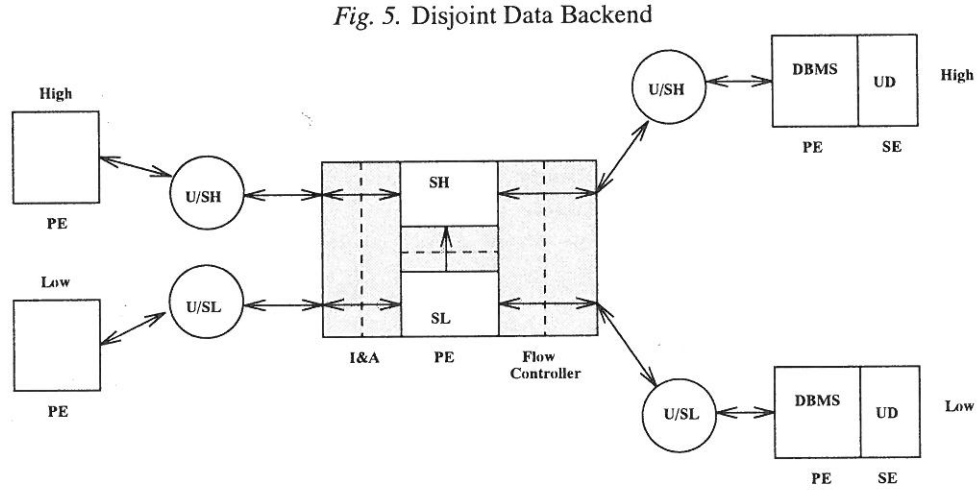

Fig. 6. Replicated Data Backend DBMS

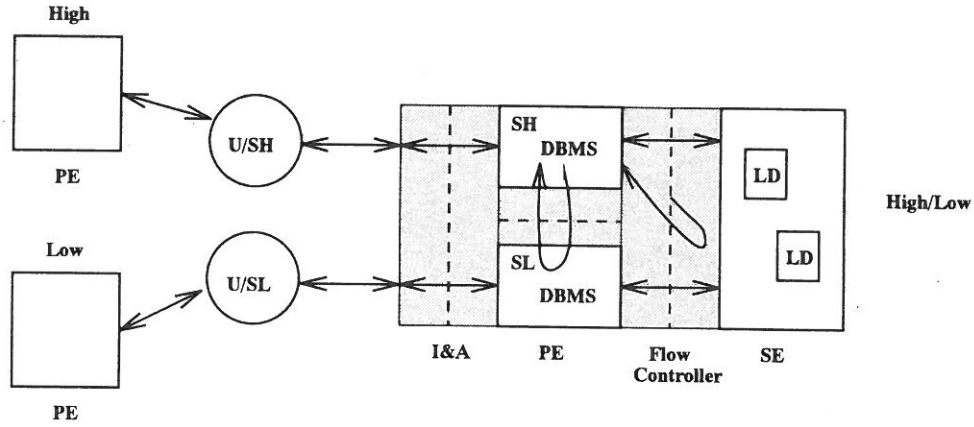

Fig. 7. Security Kernel Based DBMS

4.6. Security Kernel-Based DBMS

This architecture, shown in Figure 7, represents the more traditional approach to providing multilevel secure DBMSs. Examples of security kernel-based DBMS architectures are the SRI International SeaView system (Lunt & Boucher, 1994; Lunt & Hsieh, 1990) and TRW's ASD system (Hinke, 1988).

This architecture is modeled as three trusted-flow controllers. A frontend flow controller provides identification and authentication of users as well as maintaining the separation of different flows. The backend flow controller provides a read-down/no-write-down access to a storage component that contains labeled data. This read-down/no-write-down policy is possible since the trusted-flow controller has direct access to the data through the hardware of the storage entity. Thus, the path that retrieves the data consists totally of trusted software (the flow controller) and hardware (the storage component). The final flow controller that models the kernel is an optional (at least for the DBMS) read-down/no-write-down flow controller that provides separation between multiple untrusted processing environments, each of which contains an instance of untrusted DBMS code. This architecture could also be supported by an inter-processing environment trusted flow controller that enforced a no-flow policy, such as that provided by a separation kernel (Goguen & Meseguer, 1982).

This architecture is not vulnerable to any of the vulnerability classes listed since each untrusted DBMS can only read and write data at the level of the DBMS.

5. Conclusions

This paper makes a number of important contributions to the computer security research field. The first is that it provides a graphical model that can be used to provide a uniform representation for various trusted architectures. In this way, different architectures can be described, analyzed, compared, contrasted and discussed, based on a uniform representation of each. The second contribution is that the model is then used to represent the security guard and a number of the more significant trusted DBMS architectures that have been suggested over the years. By providing a common representation, the various architectures can be compared and contrasted.

For future work, this notation could provide a first step toward a trusted architecture computer-aided design system, with the TAG constructs providing the building blocks for system design. Then the TAG representation of an architecture could be subjected to analysis, such as the system-level security analysis presented in this paper.

References

Anderson, J. P. (1972). Computer Security Technology Planning Study. Tech. rep. ESD-TR-73-51 (AD-758206, U.S. Air Force Electronic Systems Division.

Bell, D. E., & LaPadula, L. J. (1976). Secure Computer Systems: Unified Exposition and Multics Interpretation. Tech. rep. MTR-2997 Rev. 1, MITRE Corp., Bedford, MA.

Chen, P. P.-S. (1976). The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions of Database Systems.*

Denning, D. E. (1984). Cryptographic Checksums For Multilevel Database Security. In *Proceeding of the 1984 Symposium on Security and Privacy.*

Frosher, J. N., & Meadows, C. (1990). Achieving a Trusted Database Management System Using Parallelism. In Spooner, D. L., & Landwehr, C. C. (Eds.), *Database Security, III: Status and Prospects, Results of the IFIP Working Group 11.3 Workshop on Database Security.* North-Holland.

Gifford, D. K. (1982). Cryptographic Sealing for Information Security and Authentication. *Communications of the ACM.*

Goguen, J. A., & Meseguer, J. (1982). Security Policies and Security Models. In *Proceedings of the 1982 Symposium on Security and Privacy.*

Hinke, T. H. (1986). Secure Database Management System Architectural Analysis. In *Proceedings of the AIAA/ASIS/DODCI Second Aerospace Computer Security Conference.* American Institute of Aeronautics and Astronautics.

Hinke, T. H. (1988). A1 Secure DBMS Design. In *A Postscript to the Proceedings of the 11th National Computer Security Conference.*

Hinke, T. H., & Schaefer, M. (1975). Secure Data Management System. Tech. rep. RADC-TR-266 (AD-A019201), Rome Air Development Center, AFSC, Griffiss AFB, N. Y.

Kang, M. H., & et al. (1994). Achieving Database Security Through Data Replication: The SINTRA Prototype. In *Proceedings of the 17th National Computer Security Conference.*

Lunt, T. F., & Boucher, P. K. (1994). The SeaView Prototype:Project Summary. In *Proceedings of the 17th National Computer Security Conference.*

Lunt, T. F., & Hsieh, D. (1990). The SeaView Secure Database System: A Progress Report. In *Proceedings of the European Symposium on Research on Computer Security (ESORICS 90).*

McColl̈um, C. D., & Notargiacomo, L. (1991). Distributed Concurrency Control with Optional Data Replication. In *Proceedings 5th IFIP WG 11.3 Working Conference on Database Security.*

National Bureau of Standards (1977). *Data Encryption Standard.* FIPS PUB 46, U.S. Government Printing Office.

National Bureau of Standards (1980). *DES Modes of Operation.* FIPS PUB 81, U.S. Government Printing Office.

National Computer Security Center (1985). *Department of Defense Standard: Department of Defense Trusted Computer System Evaluation Criteria.* DoD 5200.28.STD.

Pfleeger, C. P. (1989). *Security in Computing.* Prentice Hall, Englewood Cliffs, NJ.

Smith, G. W. (1990). Modeling Security-Relevant Data Semantics. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy.*

Soleglad, M. (1981). ACCAT and FORSCOM Guard Systems. In *Proceedings of the Fourth Seminar on the DOD Computer Security Initiative.* National Bureau of Standards.

Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading, MA.

Stonebraker, M., & Wong, E. (1974). Access Control in a Relational Database Management System by Query Modification. In *Proceedings 1974 ACM National Conference.*

Thomas, R. K., & Sandhu, R. S. (1992). Implementing the Message Filter Object-Oriented Security Model without Trusted Subjects. In *Proceedings of the 6th IFIP WG 11.3 Working Conference on Database Security.*

Weissman, C. (1969). Security Controls in the ADEPT-50 Time-Sharing System. In *Proceedings of the 1969 Fall Joint Computer Conference.* American Federation of Information Processing Societies Press.

Woods Hole Summer Study (1983). *Multilevel Data Management Security.* Air Force Studies Board, National Research Council.

Received: April, 1994
Accepted: September, 1995

Contact address:
Thomas H. Hinke
Computer Science Department
University of Alabama in Huntsville
Huntsville, AL, 35899, U.S.A.
phone: (205) 895-6455
fax: (205) 895-6239
e-mail: thinke@cs.uah.edu

THOMAS H. HINKE an Associate Profesor on Computer Science who joined the UAH Computer Science Faculty in 1990 after a 16-years career in industry. He received his Ph.D. in Computer Science from the University of Southern California. He has been an active researcher in the area of computer security, including database security since the mid-1970's.