# Security controls in the ADEPT-50 time-sharing system

*by* C. WEISSMAN

*System Development Corporation*
Santa Monica, California

*"Authority intoxicates/And makes mere sots of magistrates"—Butler*

## FOREWORD

At present, the system described in this paper has not been approved by the Department of Defense for processing classified information. This paper does not represent DOD policy regarding industrial application of time- or resource-sharing of EDP equipment.

## INTRODUCTION

Computer-based, resource sharing systems are, and contain, things of value; therefore, they should be protected. The valuables are the information data bases, the processes that manipulate them, and the physical plant, equipment, and personnel that form the system plexus. An extensive lore is developing on the subject of system protection.[1,2] Petersen and Turn[3] discuss in considerable detail the substance of protection of non-military information systems in terms of threats and countermeasures. Ware[4,5] contrasts "security" and "privacy" for viewing protection in militarys ystems as well. This paper describes the security controls implemented in the ADEPT–50 time-sharing system[6]—a resource sharing system designed to handle sensitive information in classified government and military facilities.*

Our approach to security control is based on a set theoretic model of access rights. This approach appears natural, since the important objects of security are sets of things—users, terminals, programs, files—and the operators of set theory—membership, intersection, union—are easily programmed for, and quickly performed by, computer. The formal model defines time-sharing security control of user, terminal, job and file security objects in terms of equations of access based upon their security profiles—a triplet of Authority, Category, and Franchise property sets. The correspondence of these properties to government and military Classification, Compartments, and Need-to-Know is demonstrated. Implementation of the model in the ADEPT–50 Time-Sharing System is described in detail, as are features that transcend the model including initialization of the security profiles, the LOGIN decision procedure, system integrity checks, security residue control, and security audit trails. Other novel features of ADEPT security control are detailed and include: automatic file classification based upon the cumulative security history of referenced files; the "security umbrella" of the ADEPT job; and once-only passwords. The paper concludes with a recapitulation of the goals of ADEPT security control, approximate costs of implementation and operation of the security controls, and suggested extensions and improvements.

Historically, protection of a sensitive computer facility has been attained by limiting physical access to the computer room and shielding the computer complex

* Development of ADEPT was supported in part by the Advanced Research Projects Agency of the Department of Defense.

from electromagnetic radiation. This "sheltered" approach promotes one-at-a-time, batch usage of the facility. Modern hardware and software technology has moved forward to more powerful and cost/effective time-shared, multi-access, multiprogrammed systems. However, three features of such systems pose a challenge to the sheltered mode of protection: (1) concurrent multiple users with different access rights operating remote from the shielded room; (2) multiple programs with different access rights co-resident in memory; and (3) multiple files of different data sensitivities simultaneously accessible. These features appear to violate traditional methods of accountability based upon a single user (or multiple users with like clearances) operating within strictly controlled facilities. The problem is of such magnitude that no time-sharing system has yet been certified for use in the manner described! However, some multi-access systems are in operation in a classified mode,[7,8] and a number of design approaches have been suggested.[9,10,11,12]

In addition to the usual goal of building an effective time-sharing system,[13] the ADEPT project began with a number of security objectives as well:

1. Build a security control mechanism that supports heterogeneous levels and types of classifications.
2. Design the security control mechanism in such a manner that it is itself unclassified until primed by security configuration parameters, a point strongly supported by Baran[14] regarding communicatons security.
3. Construct the security control mechanism as an isolated portion of the total time-sharing system so that it may be carefully scrutinized for correctness, completeness, and reliability.
4. Do the above in as frugal a manner as possible, considering costs to design, fabricate, and operate. Good system performance is our principal criterion in selecting among alternative technical solutions, as noted by the author elsewhere.[15]

In approaching our task, we recognize security as a total system problem involving hardware, communication, personnel, and software safeguards. However, our focus is primarily on monitor software, and its interfaces with the other areas. This view is not parochial: our hardware is a standard IBM 360 model 50; communication security is an established field of study with considerable technological know-how;[14] and the policy, doctrine, and procedures for personnel behavior in classified environments are extensive, with legal foundations. Thus, our only degree of freedom is the control we build into the time-sharing executive software.

*A security control formalism*

A formal model of software security control for access to sensitive portions of ADEPT is developed here.

### Security objects

Four kinds of security objects are to be managed by our model: user, terminal, job, and file. Let $u$ denote some user; $t$ some terminal; $j$ some job; and $f$ some file.

### Security properties

Each security object is described by a security profile that is an ordered triplet of security properties—Authority (A), Category (C), and Franchise (F). Authority is a set of hierarchically ordered security jurisdictions. Category is a set of discrete security jurisdictions. Franchise is a set of users licensed with privileged security jurisdiction.

The property "Authority" is defined as a set A, where

$$A = \{a^0 < a^1 <, \cdots, < a^\omega\} \qquad (1)$$

and the specific members, $a^i$, of the set are security jurisdictions hierarchically ordered.

"Category" is a discrete set of specific compartments, $c^i$,

$$C = \{c^0, c^1, \cdots, c^\psi\} \qquad (2)$$

Compartments are mutually exclusive security sanctuaries with discrete jurisdictions.

"Franchise" is a security jurisdiction privileged to a given set of users, i.e.,

$$F = \{u | u \text{ is a user}\} \qquad (3)$$

For a given terminal, $t$, let a given Authority set, A, be denoted by $A_t$, or in general, let a given security object, $\alpha$, denote a given property, P, for $\alpha$ as $P_\alpha$. Hence we can speak of $A_u$, or $C_j$, etc., to mean the specific Authority set for a given user, $u$, or the specific Category set for a given job, $j$, respectively.

Four important sets (of users) arise with respect to the Franchise property, namely, Franchise for files, terminals, jobs, and users. To distinguish the sense in which a given user is being considered, we subscript $u$ by the security object under consideration. Hence, $u_f$ means the user with jurisdiction to file $f$; $u_t$ and $u_j$ are similarly defined. For completeness, we define $u_u$ as simply $u$. We can now define Franchise for each security object.

$$F_u = \{u\} \quad (4)$$

$$F_t = \{u_t^0, u_t^1, \cdots, u_t^\lambda\} \quad (5)$$

$$F_j = \{u_j^0, u_j^1, \cdots, u_j^\mu\} \quad (6)$$

$$F_f = \{u_f^0, u_f^1, \cdots, u_f^\nu\} \quad (7)$$

Equation (4) states that the Franchise for a user is restricted to himself; his jurisdiction is unique, and no other user is so endowed. Equation (5) states that the terminal Franchise is possessed by $\lambda$ different users who have jurisdiction over the terminal $t$. Likewise, equations (6) and (7) define the job and file Franchise sets.

In security discussions, one hears the familiar phrase, "he needs a higher-level clearance." We can now define "higher level" with our model.

Let $\alpha$ and $\beta$ be security objects and let $\rho$ be some function such that $\rho(A_\alpha) \epsilon A$.
Then,

$$A_\alpha \geq A_\beta \leftrightarrow \rho(A_\alpha) \geq \rho(A_\beta) \quad (8)$$

$$C_\alpha \geq C_\beta \leftrightarrow C_\alpha \supseteq C_\beta \quad (9)$$

$$F_\alpha \geq F_\beta \leftrightarrow F_\alpha \supseteq F_\beta \quad (10)$$

Equation (8) claims that the Authority of a security object, $A_\alpha$ is at a "higher level" than another security object $A_\beta$ when the specific authority, $a_\alpha$ is greater than the specific authority, $a_\beta$.

It is implicit in equations (1) and (8) that the specific authorities, $a^i$, must be numerically encoded for the magnitude relationships to hold. Equations (9) and (10) define $P_\alpha$ to be greater than $P_\beta$ if and only if $P_\beta$ is a subset of $P_\alpha$.

Events may alter the membership of property sets. Let $P_f^e$ be the $e$th $P_f$ in a given context.

Define the Authority history, $A_h$, at the $e$th event as

$$A_h(0) = a_f^0 \quad (11)$$

$$A_h(e) = \max(A_h(e-1), \rho(A_f^e)), e > 0 \quad (12)$$

Likewise, define the Category history $C_h$, at the $e$th event as

$$C_h(0) = \phi \quad (13)$$

$$C_h(e) = C_h(e-1) \cup C_f^e, e > 0 \quad (14)$$

Equations (11) through (14) recursively define two useful sets that accumulate a history of file references as a function of file reference events, $e$. A history of the highest Authority, $A_h$, is defined by equation (12) as either the previous set, $A_h(e-1)$, or the current set, $\rho(A_f^e)$, whichever is larger in the sense of equation (8). Equation (11) gives the initial condition as some low specific file authority, $a_f^0$. Equation (14) defines the highest Category history as the union of the previous set, $C_h(e-1)$, and the current set, $C_f^e$; while equation (13) states that the union is initially the empty set.

Though $F_h$ could be defined in our model, no need is seen at this time for a Franchise history. More will be said about these history sets later.

## Property determination

Table I presents in a $3 \times 4$ matrix a summary of the rules for determining the security profile triplets, $P_\alpha$. We shall examine these rules here. For the user $u$, $A_u$ and $C_u$ are given constants, and $F_u$ is given by equation (4). For the terminal $t$, $A_t$ and $C_t$ are given constants, and $F_t$ is given by equation (5). Given $A_u$ and $A_t$, we determine $A_j$ as:

$$A_j = \min(A_u, A_t) \quad (15)$$

Likewise, given $C_u$ and $C_t$, we determine $C_j$ as:

$$C_j = C_u \cap C_t \quad (16)$$

Equation (6) gives $F_j$ to complete the job security profile triplet.

An existing file has its security profile predetermined with $A_f$ and $C_f$ as given constants, and $F_f$ as given by equation (7). However, a new file—one just created—derives its security profile from the job's file access history according to the following:

$$A_f = A_h(e) \quad (17)$$

$$C_f = C_h(e) \quad (18)$$

$$F_f = u_j^i \quad (19)$$

From equations (11) through (14) we see how the Authority and Category histories accumulate as a function of event $e$. These events are the specific times when files are accessed by a job. To maintain security

TABLE I—Security property determination matrix

| Object \ Property | Authority A | Category C | Franchise F |
|---|---|---|---|
| User, u | Given Constant | Given Constant | u |
| Terminal, t | Given Constant | Given Constant | $u_t^i$ |
| Job, j | $\min(A_u, A_t)$ | $C_u \cap C_t$ | $u_j^i$ |
| File, f | *Existing file* Given Constant | *Existing file* Given Constant | $u_f^i$ |
| | *New file* $\max(A(_he-1), \rho(A_f^e)), e > 0$ | *New file* $C_h(e-1) \cup C_f^e, e > 0$ | $u_j^i$ |

integrity, these histories can never exceed (i.e., be greater than) the job security profile. This is specified as,

$$A_h(\infty) \rightarrow A_j \qquad (20)$$

$$C_h(\infty) \rightarrow C_j \qquad (21)$$

For $e = 0$, we see the properties initialized to their simplest form. However, as $e$ gets large, the histories accumulate, but never exceed the upper limit set by the job. $A_h(e)$ and $C_h(e)$ are important new concepts, discussed in further detail later. We speak of them, affectionately, as the security "high-water mark," with analogy to the bath tub ring that marks the highest water level attained.

The Franchise of a new file is always obtained from the Franchise of the job given by equation (6). When $i = \mu = 0$, the job is controlled by the single user $u_j$ who becomes the owner and creator of the file with the sole Franchise for the file.

## Access control

Our model is now rich enough to express the equations of access control. We wish to control access by a user to the system, to a terminal, and to a file. Access is granted to the system if and only if

$$u \epsilon U \qquad (22)$$

where $U$ is the set of all sanctioned users known to the system.
Access is granted to a terminal if and only if

$$u \epsilon F_t \qquad (23)$$

If equations (22) and (23) hold, then by definition

$$u = u_t = u_j \qquad (24)$$

Access is granted to a file if and only if

$$P_j \geq P_f \qquad (25)$$

for properties A and C according to equations (8) and (9), and

$$u_j \epsilon F_f \qquad (26)$$

If equations (25) and (26) hold, then access is granted and $A_h(e)$ and $C_h(e)$ are calculated by equations (12) and (14).

## Model interpretation

Three different dimensions for restricting access to sensitive information and information processes are possible with the security profile triplet. The generality of this technique has considerable application to public and military systems. For the system of interest, however, the Authority property corresponds to the Top Secret, Secret, etc., levels of government and military security; Category corresponds to the host of special control compartments used to restrict access by project and area; such as those of the Intelligence and Atomic Energy communities; and the Franchise property corresponds to access sanctioned on the basis of need-to-know. With this interpretation, the popular security terms "classification" and "clearance" can be defined by our model in the same dimensions—as a min/max test on the security profile triplet. Classification is attached to a security object to designate the minimum security profile required for access, whereas clearance grants to a security object the maximum security profile it has permission to exercise. Thus, legal access obtains if the clearance is greater than or equal to the classification, i.e., if equation (25) holds.

Another observation on the model is the "job umbrella" concept implied by equations (22) through (26); i.e., the derived clearance of the job (not the clearance of the user) is used as the security control triplet for file access. The job umbrella spreads a homogeneous clearance to normalize access to a heterogeneous assortment of program and data files. This simplifies the problem of control in a multi-level security system. Also note how the job umbrella's high-water mark (equations (11) through (14)) is used to automatically classify new files (equations (17) and (18)); this subject is discussed further below.

A final observation on the model is its application of need-to-know to terminal access, equation (23). This feature allows terminals to be restricted to special people and/or special groups for greater control of personnel interfaces—i.e., systems programmers, computer operators, etc.

*Security control implementation*

The selection of a set theoretic model of security control was not fortuitous, but a deliberate choice biased toward computational efficiency and ease of implementation. It permits the clean separation and isolation of security control code from the security control data, which enables ADEPT's security mechanisms to be openly discussed and still remain safe—a point advocated by others.[14,16] We achieve this safety by "arming" the system with security control data only once at start-up time by the SYSLOG procedure discussed later. Also, the model improves the credibility of the security system, enhancing its understanding and thereby promoting its certification.

## Security objects: Identity and structure

Each security object has a unique identification (ID) within the system such that it can be managed individually. The form of the ID depends upon the security-object type; the syntax of each is given below.

### User identification

For generality of definition, each user is uniquely identified by his *user:id*, which must be less than 13 characters with no embedded blanks.

The *user:id* can be any meaningful encoding for the local installation. For example, it can be the individual's Social Security number, his military serial number, his last name (if unique and less than 13 characters), or some local installation man-number convention. The set of all *user:ids* constitutes the universal set, *U*.

### Terminal identification

All peripheral devices in ADEPT are identified uniquely by their IBM 360 device addresses. Besides interactive terminals, this includes disc drives, tape drives, line printer, card reader-punch, drums, and 1052 keyboard. Therefore, *terminal:id* must be a two-digit hexadecimal number corresponding to the unit address of the device.

### Job identification

ADEPT consists of two parts: the Basic Executive (BASEX), which handles the allocation and scheduling of hardware resources, and the Extended Executive (EXEX), which interfaces user programs with BASEX. ADEPT is designed to operate itself and user programs as a set of 4096-byte pages. BASEX is identified as certain pages that are fixed in main core, whereas EXEX and user programs are identified as sets of pages that move dynamically between main and swap memory. A set of user programs are identified as a job, with page sets for each program (the program map) described in the job's environment area, i.e., the job's "state tables." Every job in ADEPT has an environment area that is swapped with the job. It contains dynamic system bookkeeping information pertinent to the job, including the contents of the machine registers (saved when the job is swapped out), internal file and I/O control tables, a map of all the program's pages on drum, *user:id*, and the job security control parameters. The environment page(s) are memory-protected against reading and writing by user programs, as they are really swappable extensions of the monitor's tables.

The *job:id* is then a transitory internal parameter which changes with each user entrance and exit from the system. The *job:id* is a relative core memory address used by the executive as a major index into central system tables. It is mapped into an external two-digit number that is typed to the user in response to a successful LOGIN.

### File identification

ADEPT's file system is quite rich in the variety of file types, file organization, and equipment permitted. There are two file types: temporary and permanent.

Temporary files are transitory "scratch" disc files, which disappear from the system inventory when their parent job exits from the system. They are always placed on resident system volumes, and are private to the program that created them.

Permanent files constitute the majority of files cataloged by the system. Their permanence derives from the fact that they remain inventoried, cataloged, and available even after the job that created or last referenced them is no longer present, and even if they are not being used. Permanent files may be placed by the user on resident system volumes or on demountable private volumes.

There are six file organizations from which a user may select to structure the records of his file: Physical-sequential, S1; non-formatted, S2; index-sequential, S3; partitioned, S4; multiple volume fixed record, S5; and single volume fixed record, S9. Regardless of the organization of the records, ADEPT manages them as a collection, called a file. Thus, security control is at the file level only, unlike more definitive schemes of sub-element control.[8,10–12]

All the control information of a file that describes type, organization, physical storage location, date of creation, and security is distinct from the data records of the file, and is the catalog of the file.

All cataloged ADEPT files are uniquely identified by a four-part name; each part has various options and defaults (system assumptions). This name, the *file:id*, has the following form:

*file:id* : : = *name*, *form*, *user:id*, *volume:id*

*Name* is a user-generated character string of up to eight characters with no embedded blanks. It must be unique on a private volume as well as for Public files (described below).

*Form* is a descriptor of the internal coding of a file. Up to 256 encodings are possible, although only these seven are currently applicable:

1 = binary data
2 = relocatable program
3 = non-relocatable program
4 = card images
5 = catalog
6 = DLO (*Del*ayed *O*utput)
7 = line images

*User:id* corresponds to the owner of the file, i.e., the creator of the file.

*Volume:id* is the unique file storage device (tape, disc, disc pack, etc.) on which the file resides. For various reasons, including reliability, ADEPT file inventories are distributed across the available storage media, rather than centralized on one particular volume. Thus, all files on a given disc volume are inventoried on that volume.

## Security properties: Encoding and structure

Implementation of the security properties in ADEPT is not uniform across the security objects as suggested by our model, particularly the Franchise property. Lack of uniformity, brought about by real-world considerations, is not a liability of the system but a reflection of the simplicity of the model. Extensions to the model are developed here in accordance with that actually implemented in ADEPT.

### Authority

Authority is fixed at four levels ($\omega = 3$ for equation (1)) in ADEPT, specifically, UNCLASSIFIED, CONFIDENTIAL, SECRET, and TOP SECRET in accordance with Department of Defense security regulations. The Authority set is encoded as a logical 4-bit item, where positional order is important. Magnitude tests are used extensively, such that the high-order bits imply high Authority in the sense of equation (8).

### Category

Category is limited to a maximum of 16 compartments ($\psi \leq 15$ for equation (2)), encoded as a logical 16-bit item. Boolean tests are used exclusively on this datum. The definition of (and bit position correspondence to) specific compartments is an installation option at ADEPT start-up time (see SYSLOG). Typical examples of compartments are EYES ONLY, CRYPTO, RESTRICTED, SENSITIVE, etc.

### Franchise

Property Franchise corresponds to the military concept of need-to-know. Essentially, this corresponds to a set of *user:ids*; however, the ADEPT implementation of Franchise is different for each security object:

1. User: All users wishing ADEPT service must be known to the system. This knowledge is imparted by SYSLOG at start-up time and limited to approximately 500 *user:ids* ($\max(U) \leq 500$).

2. Terminal: Equation (5) specifies the Franchise of a given terminal, $F_t$, as a set of *user:ids*. In ADEPT, $F_t$ does not exist. One may define all the users for a given terminal, i.e., $F_t$; or alternatively, all the terminals for a given user. Because SYSLOG orders its tables by *user:id*, the latter definition was found more convenient to implement.
3. Job: The Franchise of a job is the *user:id* of the creator of the job at the time of LOGIN to the system. Currently, only one user has access to (and control of) a job ($\mu = 0$ for equation (6)).
4. File: Implementation of Franchise for a file ($F_f$), is more extensive than equation (7). In ADEPT, we wish to control not only who accesses a file, but also the quality of access granted. We have defined a set of four exclusive qualities of access, such that a given quality, q, is defined if

$$q \in \{\text{READ, WRITE, READ-AND-WRITE, READ-AND-WRITE-WITH-LOCKOUT-OVERRIDE}\} \quad (27)$$

ADEPT permits simultaneous access to a file by many jobs if the quality of access is for READ only. However, only one job may access a file with WRITE, or READ-AND-WRITE quality. ADEPT automatically locks out access to a file being written to avoid simultaneous reading and writing conflicts. A special access quality, however, does permit lockout override. Equation (7) can now be extended as a set of pairs,

$$F_f = \{(u_f^0, q^0), (u_f^1, q^1), \cdots, (u_f^\gamma, q^\gamma)\} \; ; \quad (28)$$

where $q^i$ are not necessarily distinct and are given by equation (27).

The implementation of equation (28) is dependent upon $\gamma$, the number of franchised users. When $\gamma = 0$, we have the ADEPT Private file, exclusive to the owner, $u_f^0$; for $\gamma = \max(U)$, we have the Public file; values of $\gamma$ between these extremes yield the Semi-Private file. $\gamma$ is implicitly encoded as the ADEPT "privacy" item in the file's catalog control data, and takes the place of $F_f$ for all cases except a Semi-Private file. For that case exclusively, equation (28) holds and an actual $F_f$ list of *user:id, quality* pairs exists as a need-to-know list. The owner of a file specifies and controls the file's privacy, including the composition of the need-to-know list.

## Security control initialization: SYSLOG

SYSLOG is a component of the ADEPT initialization package responsible for arming the security controls. It operates as one of a number of system start-up options prior to the time when terminals are enabled. SYSLOG sets up the security profile data for *user:id* and *terminal:id*, i.e., the "given constants" of Table I.

SYSLOG creates or updates a highly sensitive system disc file, where each record corresponds to an authorized user. These records are constructed from a deck of cards consisting of separate data sets for *compartment* definitions, *terminal:id* classification, and *user:id* clearance. The dictionary of *compartment* definitions contains the less-than-9-character mnemonic for each member of the Category set. Data sets are formed from the card types shown in Table II. Use of *passwords* is described later in the LOGIN procedure.

An IDT card must exist for each authorized user; the PWD, DEV, SEC, and CAT card types are optional. Other card types are possible, but not germane to security control, e.g., ACT for accounting purposes. More than one PWD, DEV, and CAT card is acceptable up to the current maximum data limits (i.e., 64 *passwords*, 48 *terminal:ids*, and 16 *compartments*).

A variety of legality checks for proper data syntax, quantity, and order are provided. SYSLOG assumes the following default conditions when the corresponding card type is omitted from each data set:

| | |
|---|---|
| PWD | No *password* required |
| DEV | All *terminal:ids* authorized |
| SEC | A = UNCLASSIFIED |
| CAT | C = null (all zero mask) |

This gives the lowest user clearance as the default, while permitting convenient user access. Various options exist in SYSLOG to permit maintenance of the internal SYSLOG tables, including the replacement or deletion of existing data sets in total or in part.

The sensitivity of the information in the security control deck is obvious. Procedures have been developed at each installation that give the function of deck creation, control, and loading to specially cleared security personnel. The internal SYSLOG file itself is protected in a special manner described later.

## Access control

A fundamental security concern in multi-access sys- is that many users with different clearances will be simultaneously using the system, thereby raising the

TABLE II—SYSLOG control cards

| *Card Type* | *Purpose* |
|---|---|
| DICT | Identifies start of data set of *compartment* definitions. |
| *compartment*$_1$ ··· *compartment*$_{16}$ | Defines up to 16 *compartments.* |
| TERMINAL | Identifies start of data sets of terminal definitions. |
| UNIT *terminal:id* | Identifies start of a terminal data set. |
| IDT *user:id* | Identifies start of a user data set. |
| PWD *password* ··· *password* | Defines legal *passwords* for *user:id* up to 64. |
| DEV *terminal:id*$_1$ ··· *terminal:id*$_{48}$ | Defines legal terminals for *user:id* up to 48. |
| SEC *Authority* | Defines *user:id* Authority. |
| CAT *compartment*$_1$ ··· *compartment*$_{16}$ | Defines *user:id* Category set. |

possibility of security compromise. Since programs are the "active agents" of the user, the system must maintain the integrity of each and of itself from accidental and/or deliberate intrusion. A multifile system must permit concurrent access by one or more jobs to one or more on-line, independently classified files.

ADEPT is all these things—multiuser, multiprogram, and multifile system. Thus, this section deals with access control over users, programs, and files.

## User access control: LOGIN

To gain admittance to the system, a user must first satisfy the ADEPT LOGIN decision procedure. This procedure attempts to authenticate the user in a fashion analogous to challenge-response practices.

The syntax of the ADEPT LOGIN command, typed by a user on his terminal, is as follows:

/LOGIN *user:id password accounting*

Figure 1 pictorially displays the LOGIN decision procedure based upon the user-specified input parameters. *User:id* is the index into the SYSLOG file used to retrieve the user security profile. If no such record exists (i.e., equation (22) fails), the LOGIN is unsuccessful and system access is denied. If the security profile is found, LOGIN next retrieves the *terminal:id* for the keyboard in use from internal system tables, and searches for a match in the *terminal:id* list for which the *user:id* was franchised by SYSLOG. An unsuccessful search is an unsuccessful LOGIN.

If the terminal is franchised, then the current *password* is retrieved from the SYSLOG file for this *user:id* and matched against the *password* entered as a keyboard parameter to LOGIN. An unsuccessful match is again an unsuccessful LOGIN. Furthermore, the terminal is ignored (will not honor input) for approximately 30 seconds to frustrate high-speed, computer-assisted, penetration attempts. If, however, the match is successful (equation (22) holds), the current *password* in the SYSLOG file for this *user:id* is discarded and LOGIN proceeds to create the job clearance.

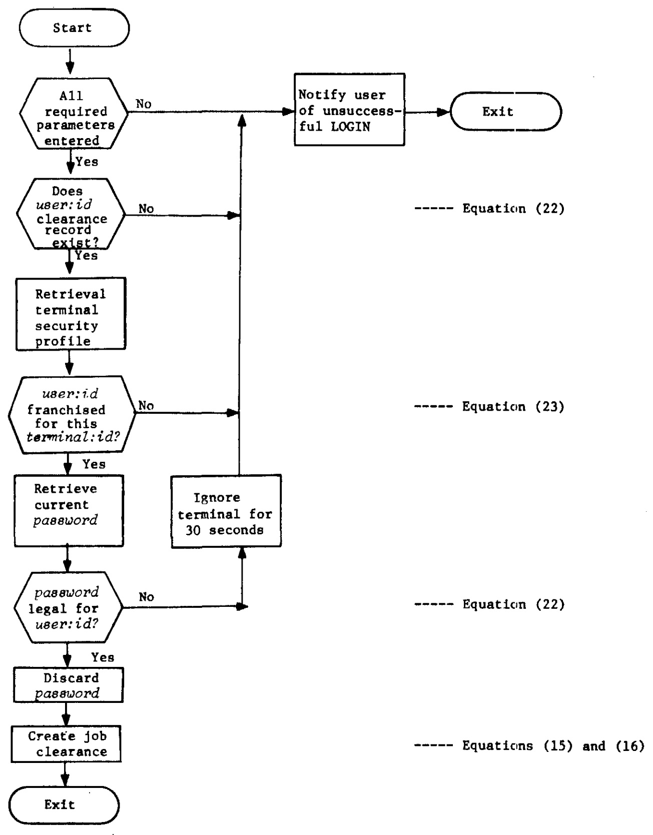
Start
All required parameters entered
No
Yes
Notify user of unsuccessful LOGIN
Exit
Does user:id clearance record exist?
No
Yes
----- Equation (22)
Retrieval terminal security profile
user:id franchised for this terminal:id?
No
Yes
----- Equation (23)
Retrieve current password
Ignore terminal for 30 seconds
password legal for user:id?
No
Yes
----- Equation (22)
Discard password
Create job clearance
----- Equations (15) and (16)
Exit

Figure 1—LOGIN decision procedure

*Passwords* in ADEPT obey the same syntax conventions as *user:id*. (See the earlier description of User Identification.) Although easily increased, currently SYSLOG permits up to 64 *passwords*. Each successful LOGIN throws away the user *password*; 64 successful LOGINs are possible before a new set of *passwords* need be established. If other than random, once-only *passwords* are desired, the 64 *passwords* may be encoded in some algorithmic manner, or replicated some number of times. Once-only *passwords* is an easily implemented technique for user authentication, which has been advocated by others.[2,7] It is a highly effective and secure technique because of the high permutability of 12-character-*passwords* and their time and order interdependence, known only to the user.

Once the authentication process is completely satisfied, LOGIN creates the job security profile according to equations (15) and (16) of our model. That is, the lower Authority of the user and the terminal becomes $A_j$, and the intersection (logical AND) of the user and terminal Category sets becomes the Category of the job, $C_j$. For example, a user with TOP SECRET Authority and a Category set (1001 1001 0000 1101) operating from a SECRET level terminal with a Category set (0000 0000 0000 0010) controls a job cleared to SECRET with an empty Category set.

## Program access control: LOAD

As noted earlier, the ADEPT Executive consists of two parts: BASEX, the resident part, and EXEX, the swapped part. EXEX is a body of reentrant code shared by all users; however, it is treated as a distinct program in each user's job. Up to four programs can exist concurrently in the job. Each operates with the job clearance—the job clearance umbrella.

LOAD is the ADEPT component used to load the programs chosen by the user; it is part of EXEX and hence operates as part of the user's job with the job's clearance. Programs are cataloged files and as such may be classified with a given security profile. As is described in "File Access Control" below, LOAD can only load those programs for which the job clearance is sufficient. Once loaded, however, the new program operates with the job clearance.

In this manner, we see the power of the job umbrella in providing smooth, flexible user operation concurrent with necessary security control. Program files may be classified with a variety of security profiles and then operate with yet another, i.e., the job clearance. By this technique security is assured and programs of different classifications may be operated by a user as one job. It permits, for example, an unclassified program file (e.g., a file editor) to be loaded into a highly classified job to process sensitive classified data files.

## File access control: OPEN

Before input/output can be performed on a file, a program must first acquire the file by an OPEN call to the Cataloger. Each program must OPEN a file for itself before it can manipulate the file, even if the file is already OPENed for another program. A successful OPEN requires proper specification of the file's descriptors—some of which are in the OPEN call, others of which are picked up directly by the Cataloger from the job environment area (e.g., job clearance, *user:id*)—and satisfactory job clearance and *user:id* need-to-know qualifications according to equations (25) and (26) of our model. Equation (25) is implemented as (8) as a straightforward magnitude comparison between $A_j$ and $A_f$. Equation (25) is implemented as (9) as an equality test between $C_f$ and $(C_j \wedge C_f)$. We use $(C_j \wedge C_f)$ to ensure that $C_f$ is a subset of the job categories; i.e., the job umbrella. Lastly, equation (26) is a NOP if the file is Public; a simple equality test between $u_j$ and $u_f$ if the File is Private; and a table search of $F_f$ for $u_j$ if the file is Semi-Private. These tests do increase processing time for file access; however, the tests are performed only once at OPEN time, where the cost is insignificant relative to the I/O processing subsequently performed on the file.

The quality of access granted by a successful OPEN, and subsequently enforced for all I/O transfers, is that requested, even if the user has a greater Franchise. For example, during program debugging, the owner of a file may OPEN it for READ access only, even though READ-AND-WRITE access quality is permitted. He thereby protects his file from possible uncontrolled modification by an erroneous WRITE call.

Considerable controversy surrounds the issue of automatic classification of new files formed by subset or merger of existing files. The heart of the issue is the poor accuracy of many such classification techniques[17] and the fear of too many over-classified files (a fear of operations personnel) or of too many under-classified files (a fear of the security control officers). ADEPT finesses the problem with a clever heuristic—most new files are created from existing files, hence classify the new file as a private file with the composite Authority and Category of all files referenced. This is achieved in ADEPT by use of the "high-water mark."

Starting with the boundary conditions of equations (11) and (13), the Cataloger applies equations (12) and

(14) for each successful file OPEN, and hence maintains the composite classification history of all files referenced by the job. For each new and temporary file OPEN, the Cataloger applies equations (17), (18), and (19); they are reapplied for each CLOSE of a new file, to update the classification (due to changes in the high-water mark since the OPEN) when the file becomes an existing cataloged file in the inventory. The scheme rarely underclassifies, and tends to overclassify when the new file is created late in the job cycle, as shown by boundary equations (20) and (21).

*Trans-formal security features*

ADEPT contains a host of features that transcend the formalism presented earlier. They are described here because they are integral to the total security control system and form a body of experience from which new formalisms can draw.

### Computer hardware

ADEPT operates on an IBM System 360/50 and is, therefore, limited to the hardware available. Studies by Bingham[9] suggest a variety of hardware features for security control, many of which are possessed by System 360.

IBM System 360 can operate in one of two states: the Supervisor state, or the Problem state. ADEPT executive programs operate in the Supervisor state; user programs operate in the Problem state.

A number of machine instructions are "privileged" to the Supervisor state only. An attempt to execute them in the Problem state is trapped by the hardware and control is returned to the executive program for remedial action. ADEPT disposes of these alarms by suspending the guilty job. (A suspended job may be resumed by the user.) Clearly, instructions that change the machine state are privileged to the executive only.

Another class of privileged instructions consists of those dealing with input/output. Problem state programs cannot directly access information files on secondary memory storage devices such as disc, tape, or drum. They must access these files indirectly by requests to the executive system. The requests are subjected to interpretive screening by the executive software.

Main memory is selectively protected against unauthorized change (write protected). We have also had the 360/50 modified to include fetch protection, which guards against unauthorized reading of—or executing from—protected memory. The memory protect instructions are also privileged only in the Supervisor state.

ADEPT software protects memory on a 4096-byte "page" basis (the hardware permits 2048-byte pages), allowing a non-contiguous mosaic of protected pages in memory for a given program. To satisfy multiprogramming, many different protection groups are needed. Through the use of programmable 4-bit hardware masks, up to 15 different protection groups can be accommodated in core concurrently. ADEPT executive programs operate with the all-zero "master key" mask, permitting universal access by all Basic and Extended Executive components.

There are five classes of interrupts processed by System/360 hardware: input/output, program, supervisor call, external, and machine check. Any interrupts that occur in the Problem state cause an automatic hardware switch to the Supervisor state, with CPU control flowing to the appropriate ADEPT executive interrupt controller. All security-vulnerable functions including hardware errors, external timer and keyboard actions, user program service requests, illegal instructions, memory protect violations, and input/output, are called to the attention of ADEPT by the System/360 interrupt system. The burden for security integrity is then one for ADEPT software.

### Monitor software

Inducing the system to violate its own protection mechanisms is one of the most likely ways of breaking a multi-access system. Those system components that perform tasks in response to user or program requests are most susceptible to such seduction.

On-Line debugging

The debugging program provides an on-line capability for the professional programmer to dynamically look at and change selected portions of his program's memory. DEBUG can be directed to access sensitive core memory that would not be trapped by memory protection, since, as an EXEX component operating in the Supervisor state, DEBUG operates with the memory protection master key. To close this "trap door," DEBUG always performs interpretive checks on the legality of the debugging request. These checks are based upon address-out-of-bounds criteria, i.e., the requested debugging address must lie within the user's program area. If not, the request will be denied and the user warned, but he will not be terminated as has been suggested.[7]

## Input/output

Input/output in System/360 is handled by a number of special-purpose processors, called Selector Channels. To initiate any I/O, it is necessary for a channel program to be executed by the Selector Channel.

SPAM, the BASEX component that permits symbolic input/output calls from user programs, is really a special-purpose compiler that produces I/O channel programs from the SPAM calls. These channel progams are subsequently delivered and executed by the ADEPT Input/Output Supervisor, IOS.

SPAM permits a variety of calls to read, write, alter, search for, and position to records within cataloged files. To achieve these ends, SPAM depends upon a variety of control tables dynamically created by the Cataloger in the job environment.

The initiating and subsequent monitoring of channel program execution is the responsibility of the BASEX Input/Output Supervisor, IOS. IOS is called to execute a channel program (EXCP). System components, such as SPAM, branch to IOS at a known entry point that is fetch-protected against entry in the Problem state. IOS is off-limits to user programs attempting to access cataloged storage. For protection against unauthorized EXCP requests, IOS always performs legality checks before executing a channel program. These checks begin by examination of the device addressed by the channel program. If it is the device address for cataloged storage, further checks are made to determine the machine state of the calling program. That state must be Supervisor state for the call to be honored. A call in the Problem state would indicate an illegal EXCP call from a user program.

IOS m. k [illegible] ther checks to guarantee the validity of an I/C request [illegible] checks to see that the specified buffer areas for the [illegible] transfer do not overlay the channel program itself, an [illegible] lie within the user's program memory area, i.e., do not modify or access system or protected memory.

Covert I/O violations are also forestalled since I/O components take direction from information stored in the job environment—an area read- and write-protected from Problem state programs.

## Classified residue

Classified residue is classified information (either code or data) left behind in memory (i.e., core, drum, or disc) after the program that referenced it has been dismissed, swapped out, or quit from the system. The standard solution to the problem is to dynamically purge the contaminated memory (e.g., overwrite with random numbers, or zeros). In a system supporting over ¼ billion bytes of memory, that solution is unreasonable and in conflict with high performance goals. ADEPT's solution to the dilemma of denying access to classified residue while maintaining high performance depends upon techniques of controlled memory allocation.

1. *Core Residue*

   As noted earlier, all core storage is allocated as 4096-byte pages. These pages are always cleared to zero when allocated, thereby overwriting any potential residue.

   Via the program's page map, the ADEPT executive system labels all code and data pages (they need not be contiguous) belonging to a given program with a single hardware memory protection key, thereby prohibiting unauthorized reading or writing by other, potentially co-resident user programs that may be in execution. Furthermore, BASEX keeps a running account of the status and disposition of all pages of core.

   The Loader and Swapper components of ADEPT always work with full 4096-byte pages. Unfilled portions of pages at load time are kept cleared to zero as when they were allocated, and the full 4096 bytes are swapped into core, if not already resident, each scheduled time slice. Further, newly allocated pages are marked as "changed" pages, thus guaranteeing subsequent swap out to drum.

   With these procedures, ADEPT denies access by a user or program to those pages of core not identified as part of his program, and clears core residue by over-writing accessible core at load and swap times.

2. *Drum Residue*

   ADEPT always clears a drum page to zero before it is allocated. The page may subsequently be cleared again to user-specified data. ADEPT also maintains a drum map that notes the disposition of all drum pages (800 pages for the IBM 2303 drum). Drum input/output, like all ADEPT I/O, is controlled by executive privileged instructions.

3. *Disc Residue*

   Disc files in ADEPT are maintained as "dirty" memory. That is, the large capacity of the file system makes it infeasible to consider automatic over-writing techniques for residue control; therefore, deleted disc tracks are returned to the available storage pool contaminated and unclean. It then becomes the burden of the

ADEPT file system to control any unauthorized file access, whether to cataloged files or uncataloged disc memory.

Team work between the Cataloger, SPAM and IOS components of ADEPT achieves this control via legality checking of all OPEN and I/O requests.

For example, all disc packs are labeled internally and externally with their *volume:id*, and this label is checked at the time of mounting by the Cataloger OPEN procedure to assure proper volume mounting. Tapes may also be labeled and checked as a user option.

Of particular note, SPAM always assumes that an end-of-file (EOF) immediately follows the last record written in a new file, and it prohibits reading beyond that EOF. Contaminated tracks allocated to new files cannot be read until they are first written. The act of writing advances the EOF and the user simultaneously over-writes the classified residue with his own data. The user cannot skip over the EOF, and the EOF location is itself protected in the job environment area.

4. *Tape Residue*

No special features for tape residue control are implemented in ADEPT. Tape residue control is easily satisfied by manual, off-line tape degaussing prior to ADEPT use.

System files

Equation (28) led us to examine Private, Semi-Private, and Public files. ADEPT possesses two additional file privacies that transcend our model; both are system files. Privacy-4 system files are the need-to-know lists created by the Cataloger itself for Semi-Private files. Privacy-5 system files are private system memory for the SYSLOG files and the catalogs themselves.

Access to these files is restricted to the system only. Special access checks are made that differ from those of equations (25) and (26). First, a special *user:id* is required that is not a member of $U$ (i.e., not in the SYSLOG file). Second, the program making the OPEN call must be in Supervisor state. Third, the program making the OPEN call must be a member of a short list of EXEX programs. The list is built into the Cataloger at the time of compilation. In this manner, access to system files is severely restricted, even to system programs.

Security service commands

ADEPT provides a variety of service commands that involve security control. The commands are listed in Table III. Note that commands VARYON, VARYOFF, REPLACE, LISTU, AUDIT, AUDOFF, and WRAP-UP are restricted to a particular terminal—the Security Officer's Station.

TABLE III—Security service commands

| *Command* | *Purpose* |
|---|---|
| AUDIT* | Turns on security audit recording. |
| AUDOFF* | Turns off security audit recording. |
| CHANGE | Enables the owner of a file to change any of the access control information of the file. |
| CREATE | Enables a user to create a Semi-Private file and its need-to-know list. |
| LISTU* | Lists by *terminal:id* all the current logged in *user:ids.* |
| RECLASS | Enables a user to raise or lower his job clearance between the bounds of the original LOGIN and current high-water mark clearance. |
| RELOG | Like LOGIN, but reconnects a user to an already existing job, as when a remote terminal drops off the communications line. |
| REPLACE* | Enables a user to move his job to another terminal or to reclassify a given device. |
| SECURITY | Print on the user's terminal approximately every 100 lines (or only by requestd the job high-water mark (or clearance by request) as a reminder to the user an) as a classification stamp of the level of current security activity. |
| VARYON/VARYOFF* | Permits terminals to be varied on- and off-line for flexibility in system maintenance and configuration control. |
| WRAPUP* | Shuts down system after a specified elapsed time. |

* Restricted to Security Officer's Station only.

## Audit

The AUDIT function records certain transactions relating to files, terminals, and users, and is the electronic equivalent of manual security accountability logs. Its purpose is to provide a record of user access in order to determine whether security violations have occurred and the extent to which secure data has been compromised. The AUDIT function may be initiated only at start-up time, but may be terminated at any time. All data re recorded on disc or tape in real time so the data is safe if the system malfunctions. An aux'liary utility program, AUDLIST, may be used to list the AUDIT file. The information recorded is shown in Table IV.

Implementation of AUDIT is quite straightforward, a product of general ADEPT recording and instrumentation.[18,19] AUDIT is an EXEX component that is called by, and at the completion of, each function o be recorded. The information to be recorded is pass d to AUDIT in the general registers. Additional I/O overhead is the primary cost incurred in the operation of AUDIT, for swapping and file maintenance. This cost is nominal, however, amounting to less than one percent of the CPU time.

## SUMMARY

In summary we may ask: How well have we met our goals? First, we believe we have developed and success-

TABLE IV—Security events and information audited by ADEPT-50

| EVENT | TIME | STATUS | JOB SECURITY PROFILE | USER SECURITY PROFILE | ACCOUNT NUMBER | USER:ID | TERMINAL:ID | CPU TIME | NEW TERMINAL | TERMINAL SECURITY PROFILE | FILE NAME | FILE OWNER ID | FILE SECURITY PROFILE | FILE FORM | FILE VOLUME NUMBER | PROSE CATEGORY NAMES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LOGIN | X | X | X | X | X | X | X | | | | | | | | | |
| LOGOUT | X | X | | | | | | X | | | | | | | | |
| OPEN FILE | X | X | | | | | | | | | X | X | X | X | X | |
| REOPEN[1] FILE | X | X | | | | | | | | | X | X | X | X | X | |
| CHANGE FILE | X | X | | | | | | | | | X | X | X | X | X | |
| CLOSE FILE | X | X | | | | | | | | | X | X | | X | | |
| DELETE FILE | X | X | | | | | | | | | X | X | | X | | |
| RECLASS | X | X | X | | | | | | | | | | | | | |
| REPLACE | X | X | | | | | X | | X | X | | | | | | |
| DEVICE LIST[2] | X | | | | | | | | | X | | | | | | |
| CATEGORY DICTIONARY[3] | X | | | | | | | | | | | | | | | X |
| RESTART[4] | X | | | | | | | | | | | | | | | |
| WRAPUP[5] | X | | | | | | | | | | | | | | | |

[1] This is the "OPEN existing file" command.

[2] A list of all the terminal devices and their assigned security and categories is recorded at each system load.

[3] A list of the prose category names is recorded at each system load.

[4] Whenever the system is restarted on the same day (and AUDIT had been turned on earlier that day) the time of the restart is recorded.

[5] The time that the AUDOFF action was taken, or the time that the WRAPUP function called AUDIT, to terminate the AUDIT function.

fully demonstrated a security control mechanism that more than adequately supports heterogeneous levels and types of classification. Of note in this regard is the LOGIN decision procedure, access control tests, job umbrella, high-water mark, and audit trails recording. The approach can be improved in the direction of more compartments (on the order of 1000 or more), extension of the model to include system files, and the implementation of a single Franchise test for all security objects. The implementation needs redundant encoding and error detection of security profile data to increase confidence in the system—though we have not ourselves experienced difficulty here. The increase in memory requirements to achieve these improvements may force numerical encoding of security data, particularly Category, as suggested by Peters.[7]

Second, SYSLOG has been highly successful in demonstrating the concept of "security arming" of the system at start-up time. Our greatest difficulty in this area has been with the human element—the computer operators—in preparing and handling the control deck. In opposition to Peters,[7] we believe the operator should not be "designed out of the operation as much as possible," but rather his capabilities should be upgraded to meet the greater levels of sophistication and responsibility required to operate a time-sharing system.[20] He should be considered part of line management. ADEPT is oriented in this direction and work now in progress is aimed at building a real-time security surveillance and operations station (SOS).

Third, we missed the target in our attempt to isolate and limit the amount of critical coding. Though much of the control mechanism is restricted to a few components—LOGIN, SYSLOG, CATALOGER, AUDIT—enough is sprinkled around in other areas to make it impossible to restrict the omnipotent capabilities of the monitor, e.g., to run EXEX in Problem state. Some additional design forethought could have avoided some of this dispersal, particularly the wide distribution in memory of system data and programs that set and use these data. The effect of this shortcoming is the need for considerably greater checkout time, and the lowered confidence in the system's integrity.

Lastly, on the brighter side, we were surprisingly frugal in the cost of implementing this security control mechanism. It took approximately five percent of our effort to design, code, and checkout the ADEPT security control features. The code represents about ten percent of the 50,000 instructions in the system. Though the code is widely distributed, SYSLOG, security commands, LOGIN, AUDIT, and the CATALOGER account for about 80 percent of it. The overhead cost of operating these controls is difficult to measure, but it is quite low, in the order of one or two percent of total CPU time for normal operation, excluding SYSLOG. (SYSLOG, of course, runs at card reader speed.) The most significant area of overhead is in the checking of I/O channel programs, where some 5 to 10 msec are expended per call (on the average). Since this time is overlapped with other I/O, only CPU bound programs suffer degredation. AUDIT recording also contributes to service call overhead. In actuality, the net operating cost of our security controls may be zero or possibly negative, since AUDIT recordings showed us numerous trivial ways to measurably lower system overhead.

## ACKNOWLEDGMENTS

I would like to acknowledge the considerable encouragement I received in the formative stages of the ADEPT security control design from Mr. Richard Cleaveland, of the Defense Communications Agency (DCA). I would like to thank Mrs. Martha Bleier, Mr. Peter Baker, and Mr. Arnold Karush for their patient care in designing and implementing much of the work I've described Also, I wish to thank Mr. Marvin Schaefer for assisting me in set theory notation. Finally, I would like to applaud the ADEPT system project personnel for designing and building a time-sharing system so amenable to the ideas discussed herein.

## REFERENCES

1 A HARRISON
*The problem of privacy in the computer age: An annotated bibliography*
RAND Corp Dec 1967 RM-5495-PR/RC

2 L J HOFFMAN
*Computers and privacy: A survey*
Stanford Linear Accelerator Center Stanford Univ Aug 1968 SLAC-PUB-479

3 H E PETERSEN R TURN
*System implications of information privacy*
Proc SJCC Vol 30 1967 291-300

4 W H WARE
*Security and privacy in computer systems*
Proc SJCC Vol 30 1967 279-282

5 W H WARE
*Security and privacy: Similarities and differences*
Proc SJCC Vol 30 1967 287-290

6 R LINDE C WEISSMAN C FOX
*The ADEPT-50 time-sharing system*
Proc FJCC Vol 35 1969 Also issued as SDC Doc SP-3344

7 B PETERS
*Security considerations in a multi-programmed computer system*
Proc SJCC Vol 30 1967 283-286

8 RYE CAPRI COINS OCTOPUS SADIE Systems

NOC Workshop National Security Agency Oct 1968

9 H W BINGHAM
*Security techniques for EDP of multi-level classified information*
Rome Air Development Center Dec 1965 RADC-TR-65-415

10 R M GRAHAM
*Protection in an information processing utility*
ACM Symposium on Operating Systems Principles Oct 1967 Gatlinburg Tenn

11 L J HOFFMAN
*Formularies—Program controlled privacy in large data bases*
Stanford Univ Working Paper Feb 1969

12 D K HSIAO
*A file system for a problem solving facility*
Dissertation in Electrical Engineering Univ of Pa 1968

13 J I SCHWARTZ C WEISSMAN
*The SDC time-sharing system revisited*
Proc ACM Conf 1967 263-271

14 P BARAN
*On distributed communications: IX, security, secrecy, and tamper-free considerations*
RAND Corp Aug 1964 RM-3765-PR

15 C WEISSMAN
*Programming protection: What do you want to pay?*
SDC Mag Vol 10 No 8 Aug 1967

16 J P TITUS
*Washington commentary—Security and privacy*
CACM Vol 10 No 6 June 1967 379-380

17 I ENGER et al
*Automatic security classification study*
Rome Air Development Center Oct 1967 RADC-TR-67-472

18 A KARUSH
*The computer system recording utility: Application and theory*
System Development Corp March 1969 SP-3303

19 A KARUSH
*Benchmark analysis of time-sharing systems: Methodology and results*
System Development Corp April 1969 SP-3343

20 R R LINDE P E CHANEY
*Operational management of time-sharing systems*
Proc 21st Nat ACM Conf 1966 149-159