1389

SLAC-117

MAS ER

The Formulary Model for Access Control and Privacy in Computer Systems

Lance J. Hoffman

SLAC Report No. 117

May 1970

AEC Contract AT(04-3)-515

STANFORD LINEAR ACCELERATOR CENTER
Stanford University · Stanford, California

SLAC-117
UC-32
(MISC)

THE FORMULARY MODEL FOR ACCESS CONTROL AND PRIVACY IN COMPUTER SYSTEMS

LANCE J. HOFFMAN
STANFORD LINEAR ACCELERATOR CENTER
STANFORD UNIVERSITY
Stanford, California 94305

PREPARED FOR THE U.S. ATOMIC ENERGY
COMMISSION UNDER CONTRACT NO. AT(04-3)-515

May 1970

Reproduced in the USA. Available from the Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia 22151.
Price: Full size copy $3.00; microfiche copy $ .65.

[illegible]

ABSTRACT

This thesis presents a model for engineering the user interface for large data base systems in order to maintain flexible access controls over sensitive data. The model is independent of both machine and data base structure, and is sufficiently modular to allow cost-effectiveness studies on access mechanisms. Access control is based on sets of procedures called formularies. The decision on whether a user can read, write, update, etc., data is controlled by programs (not merely bits or tables of data) which can be completely independent of the contents or location of raw data in the data base.

The decision to grant or deny access can be made in real time at data access time, not only at file creation time as has usually been the case in the past. Indeed the model presented does not make use of the concept of "files," though a specific interpretation of the model may do so. Access control is not restricted to the file level or the record level, although the model permits either of these. If desired, however, access can be controlled at arbitrarily lower levels, even at the bit level. The function of data addressing is separated from the function of access control in the model. Moreover, each element of raw data need appear only once, thus allowing considerable savings in memory and in maintenance effort over previous file-oriented systems.

Examples of the use of formularies in a system currently running on the IBM 360/67 are given. One recent cost study using the model is also described.

ACKNOWLEDGEMENTS

The author is deeply indebted to Professor William F. Miller for his encouragement and advice during the research and writing of this dissertation. The research environment he has provided at the Stanford Linear Accelerator Center (SLAC) Computation Group makes it a pleasure to work there. Even more important is the warmth and interest he shows in all his students. His advice has been, at the same time, timely, competent, and unobtrusive.

Many other members of the Stanford Computer Science Department and the Stanford Linear Accelerator Center have also contributed their ideas and help, in particular, John Levy, Robert Russell, and Victor Lesser. I wish to thank Professors Harold Stone, Edward Feigenbaum, and Jerome Feldman for their constructive readings of this thesis. Henry Bauer was very helpful with the interactive programs. The formulary idea was initially suggested by the use of syntax definitions ("field formularies") for input/output data descriptions, as described in (Castleman[1967]).

Part of the excellent research environment at SLAC is due to the very helpful and competent technical staff. My thanks go to the SLAC library for tracking down articles, both technical and nontechnical, on the topics involved, and to the SLAC Technical Information Department for translating chicken-scratches into meaningful illustrations. I appreciate the interest and assistance of Jorge Bruguera at the Stanford Computer Science Department library. I wish to thank Kathleen Maddern for her innumerable retypings of this dissertation, Linda Lorenzetti, SLAC Program Librarian, and Carla West, the Executive Secretary of the SLAC Computation Group.

This research was supported in part by the United States Atomic Energy Commission, the Cowell Student Health Service of Stanford University, and the Stanford University Computer Science Department.

A companion report, "The engineering of access control mechanics in physics data bases," (SLAC Report No. 118) is in preparation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This thesis presents a model for engineering the user interface for large data base systems in order to maintain flexible access controls over sensitive data. The model is independent of both machine and data base structure, and is sufficiently modular to allow cost-effectiveness studies on access mechanisms. Access control is based on sets of procedures called formularies. The decision on whether a user can read, write, update, etc., data is controlled by programs (not merely bits or tables of data) which can be completely independent of the contents or location of raw data in the data base.

The decision to grant or deny access can be made in real time at data access time, not only at file creation time as has usually been the case in the past. Indeed the model presented does not make use of the concept of "files," though a specific interpretation of the model may do so. Access control is not restricted to the file level or the record level, although the model permits either of these. If desired, however, access can be controlled at arbitrarily lower levels, even at the bit level. The function of data addressing is separated from the function of access control in the model. Moreover, each element of raw data need appear only once, thus allowing considerable savings in memory and in maintenance effort over previous file-oriented systems.

Specifically not considered in the model are privacy problems associated with communication lines, electromagnetic radiation monitoring, physical security, wiretapping, equipment failure, operating system software bugs, personnel, or administrative procedures. Cryptographic methods are not dealt with in any detail, though provision is made for inclusion of encrypting and decrypting operations in any particular interpretation of the model.

Specific interpretations of the model can be implemented on any general-purpose computer; no special time-sharing or other hardware is required. The only proviso is that all requests to access the data base must be guaranteed to pass through the data base system.

CHAPTER II

ACCESS CONTROL METHODS

A. Access Control in Existing Systems

In most existing file systems which are concerned with information privacy, passwords (Crisman [1965], Babcock [1967]) are used to provide software protection for sensitive data. Password schemes generally permit a small finite number of specific types of access to files. Each file (or user) has an associated password. In order to access information in a file, the user must provide the correct password. These methods, while acceptable for some purposes, can be compromised by wiretapping, electromagnetic radiation monitoring, and other means. Even if this were not the case, there are other reasons (Lampson [1969]) why password schemes, as implemented to date, do not solve satisfactorily the problem of access control in a large computer data base shared by many users.

One of these reasons is that passwords have been associated with files. In most current systems, information is protected at the file level only — it has been tacitly assumed that all data within a file is of the same sensitivity. The real world does not conform to this assumption. Information from various sources is constantly coming into common data pools, where it can be used by all persons with access to that pool. A problem arises when certain information in a file should be available to some but not all authorized users of the file.

In the MULTICS system (Corbato and Vyssotsky [1967]) for example, if a user has a file which in part contains sensitive data, he just cannot merge all his data with that of his colleagues. He often must separate the sensitive data and save that in a separate file; the common pool of data does not contain this sensitive and possibly highly valuable data. Moreover, he and those he permits to access this sensitive data must, if they also wish to make use of the nonsensitive data, create a distinct merged file, thus duplicating information kept in the system; if some of this duplicated data must later be changed, it must be changed in all files instead of only one. Figure 1, taken from Hoffman's survey of computers and privacy (Hoffman [1969]), graphically illustrates this situation by depicting memory allocation under existing systems and under a more desirable system.

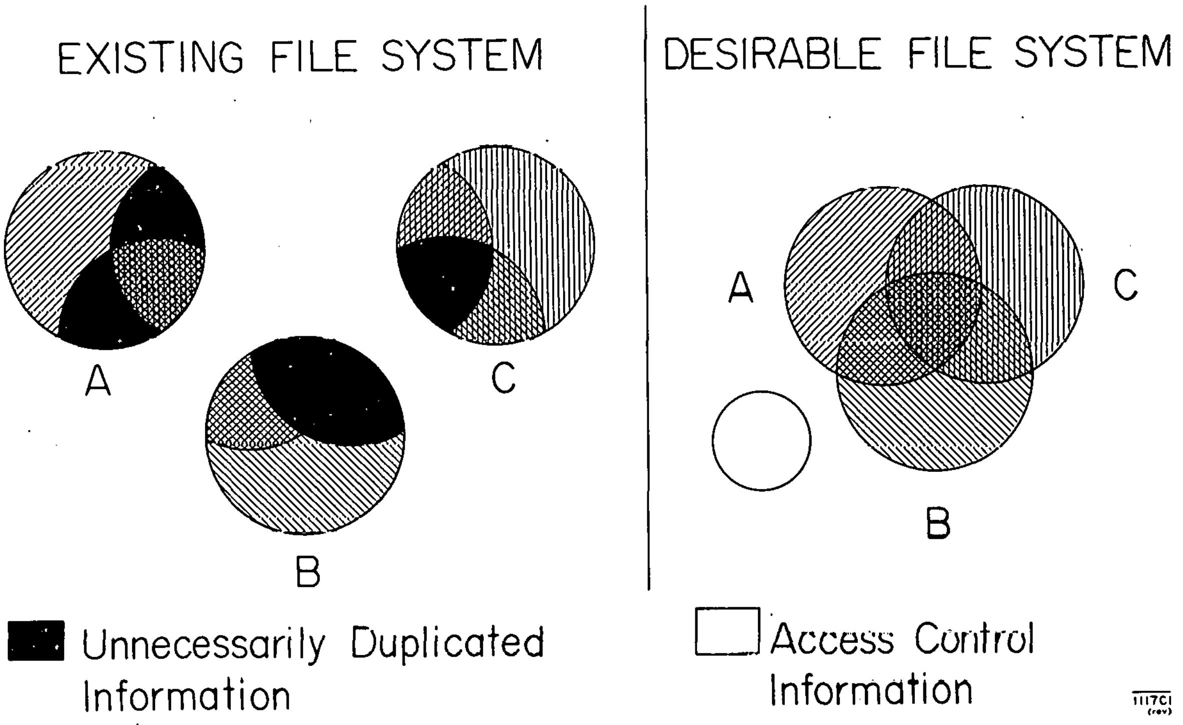

EXISTING FILE SYSTEM
DESIRABLE FILE SYSTEM
A
B
C
A
B
C
Unnecessarily Duplicated Information
Access Control Information
1117C1 (rev)

FIG. 1--Use of computer storage in file systems

The file management problems presented and the memory wastage (due to duplication of data) tend to inhibit creation of large data bases and to foster the development of smaller, less efficient,* overlapping data bases which could, were the privacy problem really solved, be merged.

*A simple cost model for information systems is presented in (Arvas [1968]) p. 34. He there derives a simple rule to determine when it is more efficient to consolidate files and when it is more efficient to distribute copies of them.

Several years ago Bingham (Bingham [1965]) suggested the use of User's Control Profiles to associate access control with a user rather than a file. This allows users to operate only on file subsets for which they are authorized and to some extent solves the memory wastage problem. Weissman has recently described a working system at SDC which makes use of security properties of users, terminals, and files (Weissman [1969]). He presents a set-theoretic model for such a system. His model does not deal with access control below the file level.

Hsiao (Hsiao [1968]) has recently implemented a system using authority items associated with users. Hsiao's system controls access at the record level, one step beneath the file level. In it, access control information is stored independently of raw data, and thus can be examined or changed without acutally accessing the the raw data. Hsiao's system and the TERPS system at West Sussex County in England (Stone [1968]) are the first working systems which control access at a level lower than the file level.

B. Access Control in Proposed Systems

Some other methods have been proposed for access control, but not yet implemented. These include a scheme which essentially assigns a sensitivity level to each program and data element in the system (Graham [1968]), another which allows higher-level programs to grant access privileges to lower-level programs (Dennis and Van Horn [1966]), and still others which place access control at the segment level via machine hardware and "codewords" (Iliffe [1968] Evans and LeClerc [1967]). These methods may prove acceptable in many contexts. However, they are not general enough for all situations. If distinct sensitivity levels cannot be assigned to data, as is sometimes the case, Graham's scheme cannot be used. The other methods, while working in principle on a computer with hardware segmentation, seem unfeasible and uneconomical on a computer with another type of memory structure such as an associative memory (Feldman [1965], Ewing and Davies [1964], Gall [1964], McAteer [1964], Raffel [1964]) or a Lesser memory (Lesser [1968]). These objections are covered in more detail in (Hoffman [1969]).

C. Desirable Characteristics for an Access Control Method

It seems desirable to devise a method of access control which does not impose an arbitrary constraint (such as segmentation or sensitivity levels) on data or programs. This method should allow efficient control of individual data elements (rather than of files or records only). Also, it should not extract unwarranted costs in storage or elsewhere from the user who wants only a small portion of his data protected. The method should be independent of both machine and file structure, yet flexible enough to allow a particular implementation of it to be efficient. Finally, it should be sufficiently modular to permit cost-effectiveness experiments to be undertaken. We would then finally have a vehicle for exploring the often-asked but never-answered question about privacy controls, "How much does technique X cost?"

We now present such a method.

CHAPTER III

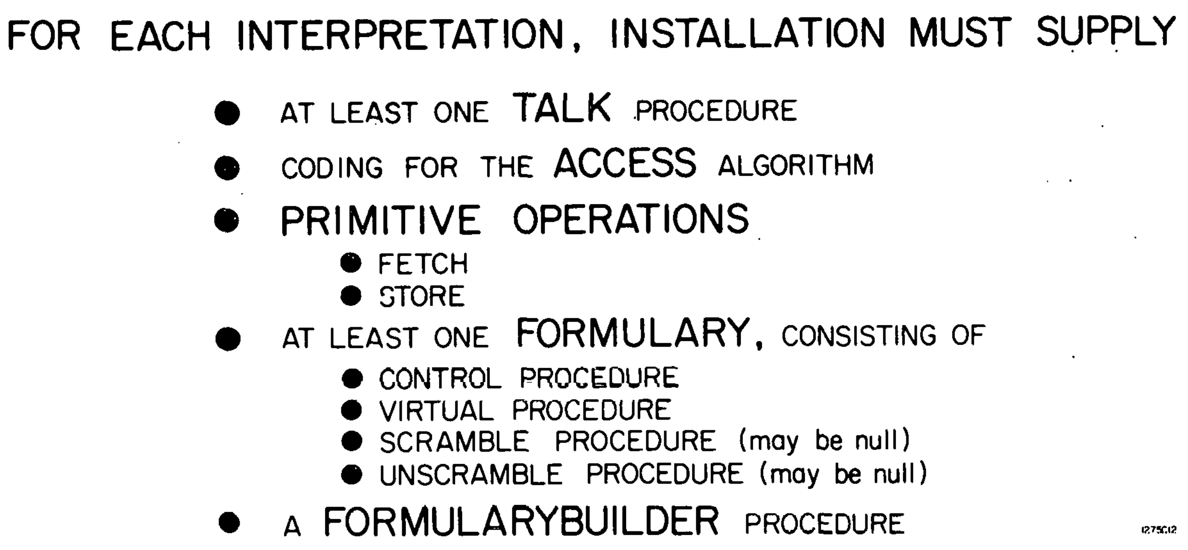
THE FORMULARY METHOD OF ACCESS CONTROL

We now describe the "formulary" method of access control. Its salient features have been mentioned in Chapter I. The decision to grant or deny access is made at data access time, rather than at file creation time, as has generally been the case in previous systems. This, together with the fact that the decision is made by a program (not by a scan of bits or a table), allows more flexible control of access. Data-dependent, terminal-dependent, time-dependent, and user response-dependent decisions can now be made dynamically at data request time, in contrast to the predetermined decisions made in previous systems, which are, in fact, subsumed by the formulary method. Access to individual related data items which may have logical addresses very close to each other can be controlled individually. For example, a salary figure might be released without any identification of an employee or any other data.

For any particular interpretation, the installation must supply the procedures listed in Table I. These procedures can all be considered a part of the general accessing mechanism, each performing a specific function. By clearly delimiting these functions, a degree of modularity is gained which enables the installation to experiment with various access control methods to arrive at the modules which best suit its needs for efficiency, economy, flexibility, etc. This modularity also results in access control becoming independent of the remainder of the operating system, a desirable but elusive goal (Weissman [1969]). While the formulary model and its central ACCESS procedure remain unchanged, each installation can supply and easily change the procedures of Table I as desirable. They are all specified in the body of this paper, and examples are given in Appendix A.

TABLE I

Procedures Supplied by the Installation



FOR EACH INTERPRETATION, INSTALLATION MUST SUPPLY

- AT LEAST ONE TALK PROCEDURE
- CODING FOR THE ACCESS ALGORITHM
- PRIMITIVE OPERATIONS
  - FETCH
  - STORE
- AT LEAST ONE FORMULARY, CONSISTING OF
  - CONTROL PROCEDURE
  - VIRTUAL PROCEDURE
  - SCRAMBLE PROCEDURE (may be null)
  - UNSCRAMBLE PROCEDURE (may be null)
- A FORMULARYBUILDER PROCEDURE

1275C12

The basic idea behind the formulary method is that a user, a terminal, and a previously built formulary (defined below) must be linked together, or attached, in order for a user to perform information storage, retrieval, and/or manipulative operations. At the time the user requests use of the data base system, this linkage is effected, but only if the combination of user, terminal, and formulary is allowed. The general linking process is described in Section G of this chapter.

Virtual memory mapping hardware is not required to implement the model, but the model does handle systems equipped with such hardware. It is assumed that enough virtual addressing capacity is available to handle the entire data base. Virtual addresses are mapped into the physical core memory locations, disc tracks, low-usage magnetic tapes, etc., by hardware and/or by the FETCH and STORE primitive operations (see Section L of this chapter) for a particular implementation.

A. Definitions and Notation

The internal name of a datum is its logical address (with respect to the structure of the data base). The internal name of a datum does not change during continuous system operation.

Examples:

1) A "tree name" such as 5.7.3.2 which denotes field 2 of branch 3 of branch 7 of branch 5 in the data base

2) "Associative memory identifiers" such as (14, 273, 34) where 14 represents the 14th attribute, 273 represents the 273rd object, and 34 represents the 34th value, in a memory similar to the one described in (Rovner and Feldman [1968]).

A User Control Block, or UCB, is space in primary (core) storage allocated during the attachment process (described in Section G). It contains the user identification, terminal identification, and information about the VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures of the formulary the user is linked to.

Usually this information is just the virtual address of each of these procedures. The virtual addresses are kept in primary storage in the UCB since a formulary, once linked to a user and terminal, will probably be (oft-) used very shortly. The first reference to any of these addresses (indirectly through the UCB) will trigger an appropriate action (e.g., a page fault on some computers) to move the proper program into primary storage (if it is not there already). It will then presumably stay there as long as it is useful enough to merit keeping in high-speed memory. The virtual addresses of procedures of a formulary cannot change while they are contained in any UCB. This constraint is easy to enforce using the CONTROL procedure described below which controls operations on any datums, including formularies. Each UCB always is in high-speed primary storage in the data area of the ACCESS procedure.

B. The ACCESS Procedure

All control mechanisms in the formulary model are invoked by a central ACCESS procedure. This ACCESS procedure is the only procedure which directly calls the primitive FETCH and STORE operations and which performs locking and unlocking operations on data items in the data base. All requests for operations on the data base must go through the ACCESS procedure.

The ACCESS procedure is a very important element of the formulary model. It is described in full detail in Section K, and its algorithm is supplied there.

The user communicates only indirectly with ACCESS. The bridge (see Fig. 2) between the system-oriented ACCESS procedure and the application-oriented user is provided by the (batch or conversational) storage and retrieval program, TALK.

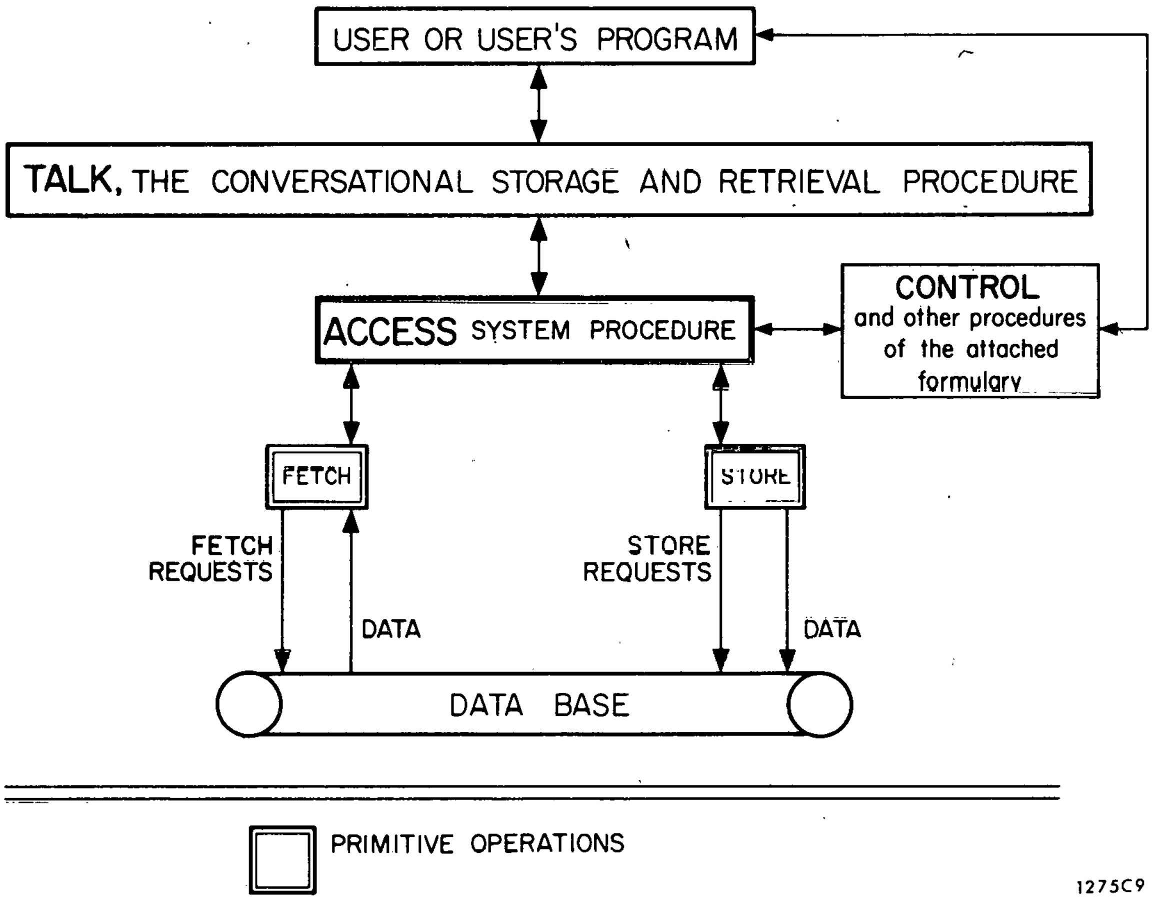


FIG. 2--User/data base interface

C. TALK, The Application-Oriented Storage and Retrieval Procedure

To access a datum, the user must call upon TALK, the (nonsystem) application-oriented storage and retrieval procedure. TALK converses with the user (or the user's program) to obtain, along with other information, (1) a datum description in a user-oriented language, and (2) the operation the user wishes to perform on that datum. TALK translates the datum description in the user-oriented language into an internal name, thus providing a bridge between the user's conception of the data base and the system's conception of the data base. The TALK procedure is described in more detail in Section J.

D. Formularies — What They Are

A formulary is a set of procedures which controls access to information in a data base. These procedures are invoked whenever access to data is requested. They perform various functions in the storage, retrieval, and manipulation of information. The set of procedures and their associated functions are the essential elements of the formulary model of access control.

Different users will want different algorithms to carry out these functions. For example, some users will be using data which is inaccessible to others; the name of a particular data element may be specified in different ways by different users; some users will manipulate data structures — such as trees, lists, sparse files, ring structures, arrays, etc., — which are accessed by algorithms specifically designed for these structures. Depending on how he wishes to name, access, and control access to elements of the data base, each user will be attached to a formulary appropriate to his own needs.

1. Procedures of a Formulary

In this subsection, we describe the procedures of a formulary. These procedures determine the accessibility, addressing, structure and interrelationships of data in the data base dynamically, at data request time. They can be arbitrarily complex. In contrast, earlier systems usually made only table-driven static determinations, prespecified at file makeup time. By use of the formulary method, these advantages are gained:

1) flexibility and changeability of data base organization to reflect current needs
2) capability to perform access control at request time as well as at file creation time
3) more efficient use of storage

Each procedure of a formulary should, if possible, run from execute-only memory, which is alterable only under administrative control. The integrity of the system depends on the integrity of the formularies and therefore the procedures of all formularies should be written by "system" programmers who are assumed honest. These procedures should be audited for program errors, hidden "trap doors," etc., before being inserted into the (effective) execute-only memory under administrative control. Failure to do this may result in the compromising of sensitive data, since an unscrupulous programmer of a formulary could cause the formulary to "leak" sensitive information to himself or to his agents.

A formulary has four procedures: VIRTUAL, SCRAMBLE, UNSCRAMBLE, and CONTROL. The first three are relevant but not central to access control; the decision on whether to grant the type of access desired is made solely by the CONTROL procedure. The first three procedures are explicitly included in each formulary for three reasons:

1) to centralize in one place all functions dealing with addressing and access control;
2) to give the model the generally necessary to model existing and proposed systems; and

3) to provide well-delimited modules for cost/effectiveness studies and for experimentation with different addressing schemes and access control schemes.

a. <u>The VIRTUAL procedure</u>. VIRTUAL translates an internal name into the virtual address of the corresponding datum. VIRTUAL is a procedure with two input parameters:

1) the internal name to be translated

2) a cell which will sometimes be used to hold "other information" as described in Section D1d below.

VIRTUAL returns

1) the resulting virtual address

2) a completion code (1 if normal completion)

Recall that enough virtual addressing capacity is assumed available to handle the entire data base. Virtual addresses are mapped into the physical core memory locations, disc tracks, low-usage magnetic tapes, etc., by hardware and/or by the FETCH and STORE primitive operations for a particular implementation.

b. <u>The SCRAMBLE procedure</u>. SCRAMBLE is a procedure which transforms raw data into encrypted form. (In some specific systems, SCRAMBLE may be null.) SCRAMBLE has two input parameters:

1) the virtual address of the datum to be scrambled

2) the length of the datum to be scrambled

SCRAMBLE has three output parameters:

1) a completion code (1 if normal completion)

2) the virtual address of the scrambled datum

3) the length of the scrambled datum

Note that if an auto-key cipher (one which must access the start of the cipher-text, whether or not the information desired is at the start) is used, <u>all</u> of the information encrypted using that cipher, be it as small as a single field or as large as an entire "file," must be governed by the same access control privileges. Therefore, some applications may choose to use several (or many) auto-key ciphers within the same "file." It is inefficient and usually undesirable to scramble data items at other than the internal name level, e.g., scrambling as a block (to effectively increase key length) the data represented by several internal names. In cases where internal names represent data which fits into very small areas of storage, greater security may be obtained by other methods (e.g., use of nulls).

We do not discuss encrypting schemes in this paper. The interested reader is referred to (Shannon [1949]), (Kahn [1967]), and (Skatrud [1969]).

c. The UNSCRAMBLE procedure. UNSCRAMBLE is an unscrambling procedure which transforms encrypted data into raw form. (In some specific systems, UNSCRAMBLE may be null.) UNSCRAMBLE has two input parameters:

1) the virtual address of the datum to be unscrambled
2) the length of the datum to be unscrambled

UNSCRAMBLE has three output parameters:

1) a completion code (1 if normal completion)
2) the virtual address of the unscrambled datum
3) the length of the unscrambled datum

d. The CONTROL procedure. CONTROL is a procedure which decides whether a user is allowed to perform the operation he requests (FETCH, STORE, FETCHLOCK, etc.) on the particular datum he has specified. CONTROL may consider the identification of the user and/or the source of the request (e.g., the terminal identification) in order to arrive at a decision. CONTROL may also converse with the requesting user before making the decision.

CONTROL has two input parameters and two output parameters. The two input parameters are:

1) the internal name of the datum

2) the operation the user desires to perform

The two output parameters are:

1) 1 if access is allowed; otherwise an integer greater than 1

2) "other information" (explained below).

In some specific systems, data elements may themselves contain access control information. Consider three examples:

Example 1.

DATUM	R	W	30 bits of actual data

If bit R is on, DATUM is readable.

If bit W is on, DATUM is writeable.

Example 2.

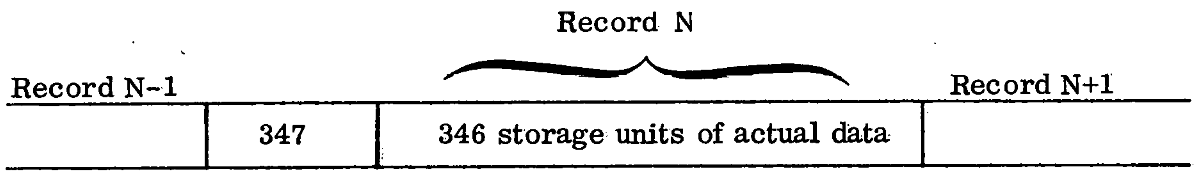
SALARY	$25,000

Reading or writing of salaries of $25,000 or over requires special checking. CONTROL must inspect the SALARY cell before it can do further capability checking and eventually return 1 or some greater integer as its first output parameter (see Fig. 5). Note that return of an integer greater than 1 actually transmits some information to the user; if he knows that he will not be allowed to alter salaries which are $25,000 or over, a denial of access actually tells him that the salary in question is at least $25,000. In the formulary model, CONTROL can only make a yes or no decision about access to a particular datum. Any more complex decisions, such as one involving release of a count which is possibly low enough to allow unwanted identification of individual data (e.g., "Tell me how many people the Health Physics Group treated for radiation sicknesses last year"), can only be made by a suitably sophisticated TALK procedure. More on pitfalls involved in using counts while protecting sensitive data is given in (Miller and Hoffman [1969]).

In order to not give out any information to the unauthorized user, the installation must decide to give up the capability provided by the formulary model to make decisions which depend on values of sensitive data.

The use of threat monitoring (Hoffman [1969]) in conjunction with the CONTROL procedure will help the installation pinpoint rapidly unauthorized attempts to access data.

Example 3.



The record contains its own length (and, therefore, also points to its successor). This type of record would appear, for example, in variable length sequential records on magnetic tape and in some list-processing applications.

In systems of this type, CONTROL might often duplicate VIRTUAL's function of transforming the internal name of a datum into that datum's virtual address. To achieve greater efficiency, CONTROL can (when appropriate) return the datum's virtual address as "other information." VIRTUAL, which is called <u>after</u> CONTROL (see the ACCESS algorithm in Section K), can then examine this "other information." If a virtual address has been put there by CONTROL, VIRTUAL will not duplicate the possibly laborious determination of the datum's virtual address, since this has already been done. VIRTUAL will merely pluck the address out of the "other information" and pass it back.

Note that CONTROL can be as sophisticated a procedure as desired; it need not be merely a table-searching algorithm. Because of this, CONTROL can consider many heretofore ignored factors in making its decision (see Fig. 3). For example, it can make decisions which are data-dependent and time-dependent. It can require two keys (or N keys) to open a lock. Also it can carry on a lengthy dialogue with the user before allowing (or denying) the access requested.

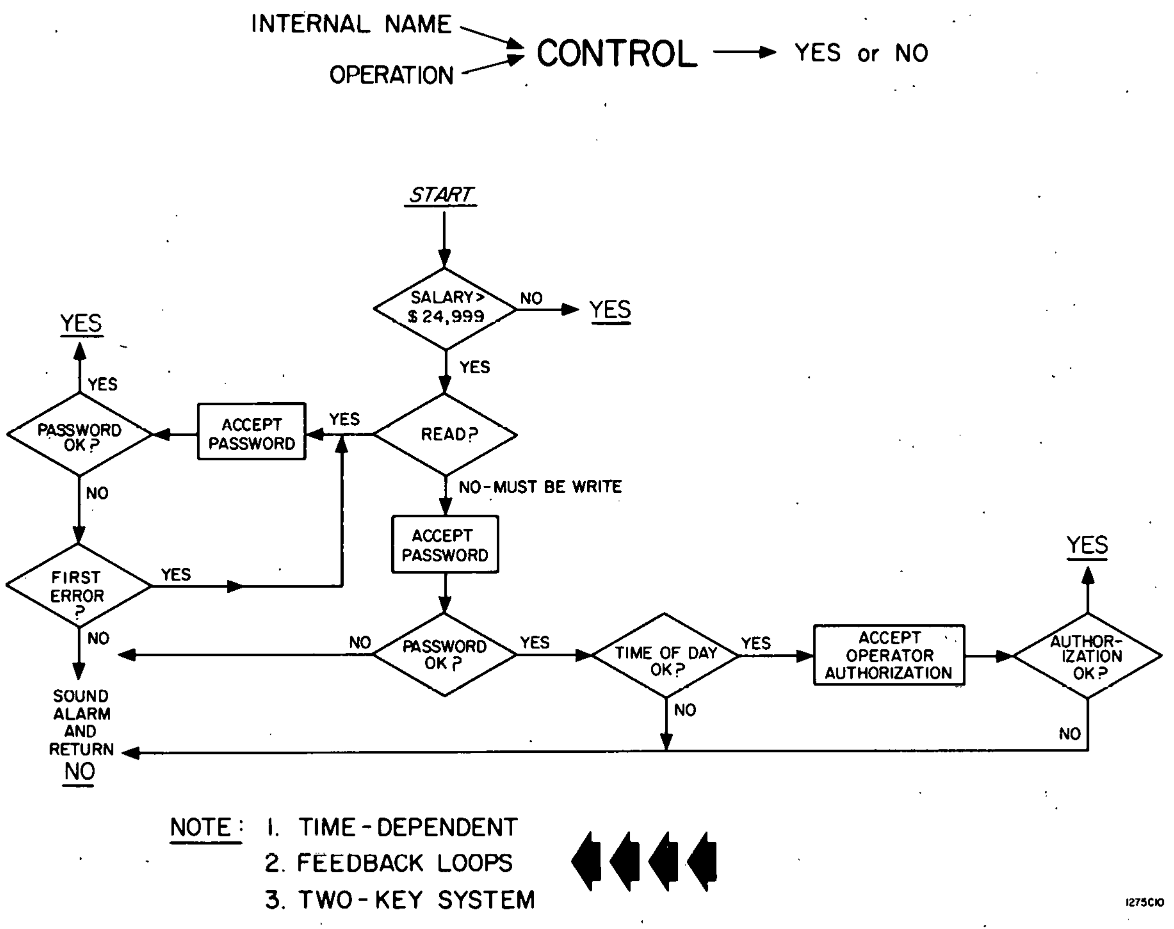


FIG. 3--A sample CONTROL procedure

CONTROL is not limited to use at data request time. In addition to being used to monitor the interactive storage, retrieval, and manipulation of data, it can also be used at initial data base makeup time for data edit picture format checking, data value validity checking, etc. Of, alternatively, one could have two procedures CONTROL1 and CONTROL2, in two different formularies, F1 and F2. F1 could be attached at data input time and F2 at on-line storage, retrieval, manipulation, and modification time.

E. Simultaneous Use of One Formulary by Multiple Users

Note that the same formulary can be used simultaneously by several different users with different access permissions. This is possible because access control is determined by the CONTROL procedure of the attached formulary. This procedure can grant different privileges to different users.

F. Building a Formulary

Before a formulary can be attached to a user and a terminal, the procedures it contains must be specified. This is done using the system program FORMULARYBUILDER. FORMULARYBUILDER converses with the systems programmer who is building a formulary to learn what these procedures are, and then retrieves them from the system library and enters them as a set into a formulary which the user names. The specifics of FORMULARYBUILDER depend on the particular system.*

G. The Attachment Process — The Method of Linking a Formulary to a User and Terminal

In order to allow information storage and retrieval operations on the data base to take place, a user, a terminal, and a formulary which has been previously built using FORMULARYBUILDER must be linked together. This linking process is done in the following manner.

*An extension to FORMULARYBUILDER which would allow a user to grant capabilities to other users, and then allow these users to grant capabilities to still other users, etc., has been proposed by Victor Lesser and will be investigated further in the future.

At the first time ACCESS is called (by TALK) for a given user and terminal, it will only permit attachment of a formulary to the user and terminal (i.e., it will not honor a request to fetch, store, etc.). The attachment is permitted only if the CONTROL program of the default formulary allows. The default formulary, like all other formularies, contains VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures. For the default formulary, they act as follows:

CONTROL	CONTROL takes the internal name representing the formulary and decides whether user U at terminal T is allowed to attach the formulary represented by the internal name. U and T are maintained in the UCB and passed to CONTROL by ACCESS.
VIRTUAL	VIRTUAL takes the internal name representing the formulary and returns the virtual address of the formulary.
SCRAMBLE	No operation.
UNSCRAMBLE	No operation.

The ATTACH attempt, if successful, causes information about the formulary specified by the user to be read into the UCB (which is located in the data area of the ACCESS procedure). ACCESS then uses this information (when it is subsequently called on behalf of this user/terminal combination) to determine which CONTROL, VIRTUAL, SCRAMBLE, and UNSCRAMBLE procedures to invoke.

1. Independence of Addressing and Access Control

After the attachment process, the User Control Block (UCB) contains the user identification U, terminal identification T, and information about (usually pointers to) the VIRTUAL, CONTROL, SCRAMBLE, and UNSCRAMBLE procedures of a formulary. Whether the user can perform certain operations on a given datum is controlled by the CONTROL program. The addressing of each datum is controlled by the VIRTUAL program. Addressing of data items is now completely independent of the access control for the data items.

2. Breaking an Attachment

An existing attachment is broken whenever

1) the user indicates that he is finished using the information storage and retrieval system (either by explicitly declaring so or implicitly by logging out, removing a physical terminal key, reaching the end-of-job indicator in his input card deck, etc.), or

2) the user, via his TALK program, explicitly detaches himself from a formulary.

H. Subdivision of Data Base into Files Not Required

Note that while the concept of a data set (or a "file") MAY be used, the formulary method does not require this. This represents a significant departure from previous large-scale data base systems which were nearly all organized with files (data sets) as their major subdivisions. Under the formulary scheme, access to information in a data set is not governed by the data set name. Rather, it is governed by the CONTROL procedure of the attached formulary. Similarly, addressing of data in a data set is governed by the VIRTUAL procedure and not by the data set name. Subdividing a data base into data sets, while certainly permitted and often desirable, is not required by the formulary model.

I. Concurrent Requests to Access Data — The LOCKLIST

The problem of two or more concurrent requests for exclusive data access necessitates a mechanism to control these conflicts among competing users. This problem has been discussed, and solutions proposed, in (Dijkstra [1965]), (Hsiao [1968]), and (Shoshani and Bernstein [1969]). In the formulary model, data can be set aside (locked) dynamically for the sole use of one user/terminal combination in a manner similar to Hsiao's "blocking" (Hsiao [1968]), using a mechanism known as the LOCKLIST.

The locking and unlocking of data to control simultaneous updating is an entirely separate function from the access control function. Access control takes into account privacy considerations only. Locking and unlocking are handled by a separate mechanism, the LOCKLIST. The LOCKLIST is a list of triplets maintained by the ACCESS program and manipulated by the FETCHLOCK, STORELOCK, UNLOCKFETCH, and UNLOCKSTORE operations. Each triplet contains (1) the internal name of a current item, (2) the identification of the user/terminal combination which caused it to be locked, and (3) the type of lock (fetch or store). Any datum represented by a triplet on the LOCKLIST can be accessed only by the user/terminal combination which caused it to be locked.

Data items which can be locked are atomic, i.e., subparts of these data items can not be locked. This implies, for example, that if a user wishes to lock a tree structure and then manipulate the tree without fear of some other user changing a subnode of the tree, either

1) The tree must be atomic in the sense that its subnodes do not have internal names in the data base system, or
2) each subnode must be explicitly locked by the user and only after all of these are locked can he proceed without fear of another user changing the tree.*

*A more general and elegant method of handling concurrent requests to access data is being developed by R. D. Russell as part of a general resource allocation method. Much of the housekeeping work currently done in the formulary model can be handled by his method.

J. The TALK Procedure — Details

To access a datum, the user must effectively call upon TALK, the (nonsystem) application-oriented storage and retrieval procedure. TALK converses with the interactive user and/or the user's program and/or the operating system to obtain

(1) a datum description in a user-oriented language

(2) the operation the user wishes to perform on that datum

(3) user identification and other information about the user and/or the terminal where the user is located.

Depending on the particular system, the user explicitly gives TALK zero, one, two, or all three of the above parameters. TALK supplies the missing parameters (if any), converts (1) to an internal name, and then passes the user identification, the terminal identification, the internal name of the datum, and the desired operation to the ACCESS procedure, which actually attempts to perform the operation.

Note that one system may have available many TALK procedures. A user requests invocation of any of them in the same way he initiates any (nonsystem) program. Sophisticated users will require only "bare-bones" TALK procedures, while novices may require quite complex tutorial TALK procedures. They may both be using the same data base while availing themselves of different datum descriptions. As an example, one TALK procedure might translate English "field names" into internal names, while another TALK procedure translates French "field names" into internal names. This ability to use multiple and user-dependent descriptions of the same item is not available with such generality in any system the author is aware of, though some systems allow lesser degrees of this (Jones [1968], Giering [1967]).

Different TALK procedures also allow concealment of the fact that certain information is even in a data base, as illustrated in Fig. 4.

USER 1

```
WHAT PROGRAM? talk1
TALK1 HAS BEGUN EXECUTION.
WHAT DATA WOULD YOU LIKE TO SEE?
        salary of robert d. jones
YOU ARE NOT PERMITTED READ ACCESS
   TO THE SALARY FIELD.
```

CONTROL determined that the user was not permitted read access, causing this reply to be given by TALK1.

USER 2

```
WHAT PROGRAM? talk2
TALK2 HAS BEGUN EXECUTION.
WHAT DATA WOULD YOU LIKE TO SEE?
        salary of robert d. jones
NO FIELD NAMED SALARY.
```

TALK2 intentionally returned this reply to the user.

1465AII

FIG. 4--Concealment of the fact that a data base contains certain information

The above remarks about using different TALK procedures also apply if a system uses only one relatively sophisticated TALK procedure which takes actions dependent on the person or terminal using it at a given time.

K. The ACCESS Procedure — Details

ACCESS uses the VIRTUAL, CONTROL, UNSCRAMBLE, and SCRAMBLE procedures specified in the UCB to carry out information storage and retrieval functions. Its input parameters are:

(1) information about the user, terminal, etc., defined by the installation. This information is passed by the procedure that calls ACCESS;

(2) internal name of datum;

(3) an area which either contains or will contain the value of the datum specified by (2);

(4) the length of (3);

(5) operation to perform — FETCH, FETCHLOCK, STORE, STORELOCK, UNLOCKFETCH, UNLOCKSTORE, ATTACH, or DETACH. FETCHLOCK and STORELOCK lock datums to further fetch or store accesses respectively (except by the user/terminal combination for which the lock was put on). UNLOCKFETCH and UNLOCKSTORE unlock these locks. ATTACH and DETACH respectively create and destroy user/terminal/formulary attachments.

(6) a variable in which a completion code is returned by ACCESS.

ACCESS itself handles all operations of (5) except FETCH and STORE. For FETCH and STORE operations on the data base, it invokes the FETCH and STORE primitives specified in Section L.

An ALGOL algorithm for the ACCESS procedure follows. This procedure is quite important and should be examined carefully. The comments in the algorithm should not be skipped, as they often suggest alternate methods for accomplishing the same goals. An example of the actual coding in use at one particular installation is given as Exhibit 1 of Appendix A. Note that some means must be provided to determine which formulary is attached so that the CONTROL, SCRAMBLE, UNSCRAMBLE, and VIRTUAL procedures of that particular formulary can be invoked. The program in Exhibit 1 transfers this responsibility to those procedures themselves, which determine which formulary is attached by examining common data set up previously by the ACCESS procedure. An alternative method, if ACCESS were written in a more powerful language or in assembly language, would be to use a transfer vector.

Note that two procedures and their corresponding calls can be removed from ACCESS if no user will ever have to lock out access to a datum which ordinarily can be accessed by several users at the same time or if the installation wishes to use another method to control conflicts among users competing for exclusive access to datums; this makes the procedure considerably shorter. Such a "no parallelism" version of the ACCESS algorithm is given in Appendix C.

The ACCESS Algorithm

```
procedure access (info, intname, val, length, opn, compcode);
integer array info, val; integer, length, opn, compcode;
begin comment If OPN = FETCH, VAL is set to the value of the datum
                represented by INTNAME.
              If OPN = STORE, the value of the datum represented by
                INTNAME is replaced by the value in the VAL array.
              If OPN = FETCHLOCK or STORELOCK, the datum is locked to
                subsequent FETCH or STORE operations by other users or from
                other terminals until an UNLOCKFETCH or UNLOCKSTORE operation,
                whichever is appropriate, is performed.
              If OPN = UNLOCKFETCH or UNLOCKSTORE, the fetch lock or store
                lock previously inserted by a FETCHLOCK or STORELOCK opera-
                tion is removed.
              If OPN = ATTACH, the formulary represented by internal name
                INTNAME is attached to the user and terminal described in the
                INFO array.
              If OPN = DETACH, the formulary represented by internal name
                INTNAME is detached from the user and terminal described
                in the INFO array.
              VAL is LENGTH storage elements long.
              Note that a FETCH (STORE) operation will actually attempt
                to fetch (store) LENGTH storage elements of information.
              It is the responsibility of the TALK procedure to handle
                scrambling or unscrambling algorithms that return outputs
                of a different length than their inputs.
```

ACCESS returns the following integer completion codes in COMPCODE:

1	normal exit, no error
2	unlock operation requested by user or terminal who/which did not set lock
3	operation permitted but gave error when attempted
4	attempt to unlock datum which is not locked in given manner
5	cannot handle any more User Control Blocks (would cause table overflow)
6	attempt to detach nonexistent user/terminal/formulary combination
7	operation permitted for this user and terminal but could not be carried out since datum was locked (by another user/terminal) to prevent such an operation
8	cannot put lock on as requested since LOCKLIST is full
9	datum already locked by this user and terminal
10	error return from VIRTUAL procedure
11	operation on the datum represented by INTNAME not permitted by CONTROL procedure of the attached formulary
12	end of data set encountered by FETCH operation.

Note that by the time the user has left the ACCESS routine, the data may have been changed by another user (if the original user did not lock it). Note that ACCESS could be altered to allow scrambling and unscrambling to take place at external devices rather than in the central processor.

Important: ACCESS expects the following to be available to it. The installation supplies these in some way other than as parameters to ACCESS (for example, as global variables in ALGOL or COMMON variables in FORTRAN) —

(1)	ISTDUCB	the default User Control Block. Its length is NUCB storage units.
(2)	NUCB	see (1).
(3)	UCB	a list of User Control Blocks (UCB's) initialized outside ACCESS to ucb (1, 1) = -2, ucb (i, j) = anything when $\sim(i = j = 1)$ UCB is declared as <u>integer</u> <u>array</u> (1:maxusers, 1:nucb).
(4)	MAXUSERS	the maximum number of users which can be actively connected to the system at any point in time.
(5)	ITALK	the length of the INFO array (which is the first parameter of ACCESS) — INFO contains information about the user and terminal which is used by ACCESS and also passed by ACCESS to procedures of the attached formulary. INFO(1) contains user identification.
(6)	LOCKLIST	a list of locks (each element of the LOCKLIST array should be initialized outside ACCESS to -1). LOCKLIST is declared as <u>integer</u> <u>array</u> (1:4, 1:maxllist).
(7)	MAXLLIST	the maximum length of the LOCKLIST
(8)	CS1	a semaphore to govern simultaneous access to the critical section of the ACCESS procedure (initialized to 1 outside ACCESS).

ACCESS assumes that the variables FETCH, STORE, FETCHLOCK, STORELOCK, UNLOCKFETCH, UNLOCKSTORE, ATTACH, and DETACH have been initialized globally and are never changed by the installation;

```
integer array iucb [1:nucb], reslt [1:length];
integer i, ii, islot, j, yesno, other, n, datum;

integer procedure testandset (semaphore); integer semaphore;
begin comment TESTANDSET is an integer function designator. It returns -1
if SEMAPHORE was in the state LOCKED on entry to TESTANDSET. Otherwise,
TESTANDSET returns something other than -1. In all cases, SEMAPHORE is in
state LOCKED after the execution of the TESTANDSET procedure, and must be
explicitly unlocked in order for it to be used again.
    TESTANDSET is used to implement a controlling mechanism to prevent
conflicts among users competing for the same resource, as discussed in
(Dijkstra [1965]). It will not prevent "deadly embraces" (Habermann [1969]). No
explicit code is given here, since the function is machine-dependent. The manner
in which TESTANDSET is implemented for a particular machine, the IBM 360/67,
is shown in the listing of the TESTSE procedure in Exhibit 1 of Appendix A.
    This procedure can be removed if no user will ever have to lock out access
to a datum which ordinarily can be accessed by several users at the same time
or if the installation wishes to use another method to control conflicts among users
competing for exclusive access to datums;
                                  < code >
end testandset;

integer procedure idxll (intname, opn); integer intname, opn;
begin comment IDXLL, given an internal name INTNAME, returns the relative
position of INTNAME on the LOCKLIST if the datum represented by INTNAME is
locked in a manner affecting the operation OPN. Otherwise, IDXLL returns
```

the negation of the relative location of the first empty slot on the LOCKLIST. If the LOCKLIST is full and the INTNAME/OPN combination is not found on it, IDXLL returns 0.

This procedure can be removed if no user will ever have to lock out access to a datum which ordinarily can be accessed by several users at the same time or if the installation wishes to use another method to control conflicts among users competing for exclusive access to datums;

```
integer firstempty;
j := if opn = FETCH or opn = UNLOCKFETCH or opn = FETCHLOCK then 1 else 2;
idxll := firstempty := 0;
for i := 1 step 1 until maxllist do
      begin ii := -i;
      if locklist [1, i] = -1 then firstempty := i
else if locklist [1, i] = intname and locklist [2, i] = j then begin idxll := i;
                                                                  go to RET
                                                                  end;
      end;
if firstempty ≠ 0 then idxll := -firstempty;
RET:
end idxll;

procedure ret (i); integer i;
begin comment RET sets the completion code compcode to i and then causes
exit from the ACCESS procedure;
compcode := i; go to FIN
end ret;
```

```
compcode := 1;
comment first let's see if we recognize the user/terminal combination
in INFO;
islot := 0;
for i := 1 step 1 until maxusers do
    begin ii := i;
        if ucb [i, 1] = -2 then begin comment end of list of ucb's;
                    if islot=0 then begin if ii ≠ maxusers then ucb [ii+1, 1]:= - 2;
                                go to XFER;
                                end
                            else go to PRESETUP;
                            end
else if ucb [i, 1] =-1 then islot := ii
    comment remember this slot if vacant;
else begin for j := 1 step 1 until italk do
        if ucb [i, j]≠info[j] then go to ILOOPND;
    go to SETUPPTRS
    end;
ILOOPND:
    end i loop;
if islot = 0 then ret (5); comment cannot handle any more UCBs;
PRESETUP:
ii := islot;
XFER:
for k := 1 step 1 until italk do ucb[ii, k] := info[k];
for k := italk + 1 step 1 until nucb do ucb[ii, k] := istducb[k];
```

```
SETUPPTRS:
for i := 1 step 1 until nucb do iucb[i] := ucb[ii, i];
comment set up pointers to appropriate user control block for particular
implementation.  Note well:  Setting up pointers to appropriate user control blocks
is quite dependent on the particular system.  For an example of one implementation,
see Exhibit 1 of Appendix A;
comment  We have now associated user and terminal with the user control block
(representing a formulary) in relative position i of the UCB table;
if iucb[nucb] ≠ intname and opn = DETACH then ret (6);
comment attempt to detach user/terminal/formulary combination not currently
attached;
control (intname, opn, yesno, other);
if yesno  > 1 then ret (11);
comment  return 11 if CONTROL does not permit operation;
if opn = ATTACH then begin ucb[ii, nucb] := intname; go to FIN
                          end;
comment  Note well:  In many implementations, pointers to each procedure of
the formulary (obtained by having VIRTUAL transform intname into a virtual
address) might be put into the UCB upon attachment.  In others, the philosophy
used here of only putting one pointer — to the formulary — into the UCB will be
followed.  The decision should take into account design parameters such as
implementation language, storage available, etc.;
if opn = DETACH then begin comment detach formulary (this leaves an open
                          slot in the ucb array); ucb[ii, 1] := -1; go to FIN
                          end;
```

```
if opn = UNLOCKFETCH or opn = UNLOCKSTORE then
     begin i := idxll(intname, opn); comment find internal name on LOCKLIST;
     if i ≤ 0 then ret(4); comment cannot find it;
     for j := 1 step 1 until italk do
          if locklist [2+j, i] ≠ iucb[j] then ret(2);
     locklist[1, i] := -1; comment undo the lock and mark slot in UCB array empty;
     go to FIN
     end unlock operation;
TRY:
if testandset(cs1) = -1 then go to TRY;
comment loop until no other user is executing the critical section below;
comment ACCESS should ask to be put to sleep if embedding system permits;
comment ----------------- enter critical section for locking out datums --------;
i := idxll(intname, opn);
comment get relative location of locked datum in locklist;
if i > 0 then begin comment datum found on locklist so see if it was locked by
               this user and terminal;
               for j := 1 step 1 until italk do
                   if locklist [2+j, i] ≠ iucb[j] then ret(7);
                   comment data already locked by another user or terminal;
                   if opn = FETCHLOCK or opn = STORELOCK then ret(9);
                   comment datum already locked by this user and terminal,
                   so return completion code of 9;
               end;
```

```
i :=-i;
if opn = FETCHLOCK or opn = STORELOCK then
     begin comment this is a lock operation;
     if i = 0 then ret(8); comment connot set lock since locklist is full;
     locklist[2, i] :=if opn = FETCHLOCK then 1 else 2;
     comment set appropriate lock;
     for j :=1 step 1 until italk do locklist[2+j, i]:=iucb[j];
     comment place user and terminal identification into LOCKLIST;
     locklist[1, i]:= intname; comment place internal name on LOCKLIST;
     go to FIN;
     end lock operation;
virtual (intname, datum, other, compcode);
comment VIRTUAL returns in datum the virtual address of the datum specified;
if compcode > 1 then ret(10); comment error return from VIRTUAL;
if opn = STORE then
     begin comment store operation;
     scramble (val, length, compcode, reslt, n);
     if compcode > 1 then ret(3);
     comment operation permitted but gave error when attempted;
     comment now perform a physical write of n storage units to the block
     starting at reslt;
     store (datum, reslt, n, compcode);
     if compcode > 1 then ret(3)
     end
else
     begin comment fetch operation;
     fetch (datum, reslt, length, compcode);
```

```
    if compcode = 2 then ret(12); comment end of data set encountered;
    if compcode > 1 then ret(3);
    unscramble (reslt, length, compcode, val, n);
    if compcode >1 then ret(3);
    end fetch operation;
FIN:
comment --------------- Leave critical section for locking out datums ------------;
cs1 := 1;
end access;
```

L. FETCH and STORE Primitive Operations

The two primitive operations FETCH and STORE are supplied by the installation. These primitives actually perform the physical reads and writes which cause information transfer between the media the data base resides on and the primary storage medium (usually, magnetic core storage). They are invoked only by the ACCESS procedure. Examples of FETCH and STORE primitives for a particular implementation are given in Exhibit 2 of Appendix A.

The primitive operations cannot be expressed in machine-independent form, but rather depend on the specific system and machine used. They are defined functionally below.

FETCH(ADDR, VALUE, LENGTH, COMP)

This primitive fetches the value which is contained in the storage locations starting at virtual address ADDR and returns it in VALUE. This value may be scrambled, but if so unscrambling will be done later by UNSCRAMBLE (called from ACCESS), and LENGTH is the length of the scrambled data. The value comprises LENGTH storage elements. Upon completion, the completion code COMP is set to:

1 if normal exit

2 if end of data set encountered when physical read attempted

3 if length too big (installation-determined)

4 if illegal virtual address given to fetch from

5 if error occurred upon attempt to do physical read

STORE(ADDR, VALUE, LENGTH, COMP)

This primitive stores LENGTH storage elements starting at virtual address VALUE into LENGTH storage elements starting at virtual address ADDR. The information stored may be scrambled, but if so the scrambling has already been done by SCRAMBLE (called from ACCESS), and LENGTH is the length of the scrambled data. Upon completion, the completion code COMP is set to:

1 if normal exit

3 if length too big (installation-determined)

4 if illegal virtual address given to store into

5 if error occurred upon attempt to do physical write.

IV. USE OF FORMULARIES IN A WORKING MEDICAL SYSTEM

This section describes a particular implementation of the formulary model of access control and privacy. This implementation was used to insure privacy for the computer-based records of individual patient visits at the Cowell Student Health Service of Stanford University.

The Cowell Student Health Service (hereafter referred to as SHS) maintains short records of each individual patient visit (in addition to the more detailed medical histories which each physician affiliated with the SHS keeps). These short records contain information which is used to review and make more efficient use of physician services, nursing services, and office resources. They are also used to spot short and long term trends in causes for visiting the SHS, so that specific trends can be planned for and/or arrested. Each record contains an SHS-assigned number which identifies the particular patient.

The data which is kept for these short records was kept under fairly tight control even before SHS adopted the formulary model. No "leaks" had ever been detected. But as a result of a general review of privacy control in the SHS computer-based files, additional safeguards were implemented, including protection of privacy via the formulary scheme. All of these additional safeguards could have been implemented without making use of the formulary model. One result of the use of the formulary model, however, has been the compartmentalization and separation of scrambling, unscrambling, and access-granting decision functions. These functions can now be easily changed or "tuned" to fit future requirements. The SHS system is an example of a particular implementation of the general formulary model. The system as described here is nearly 100% operational at this time (though general use will be phased in as funds become available).

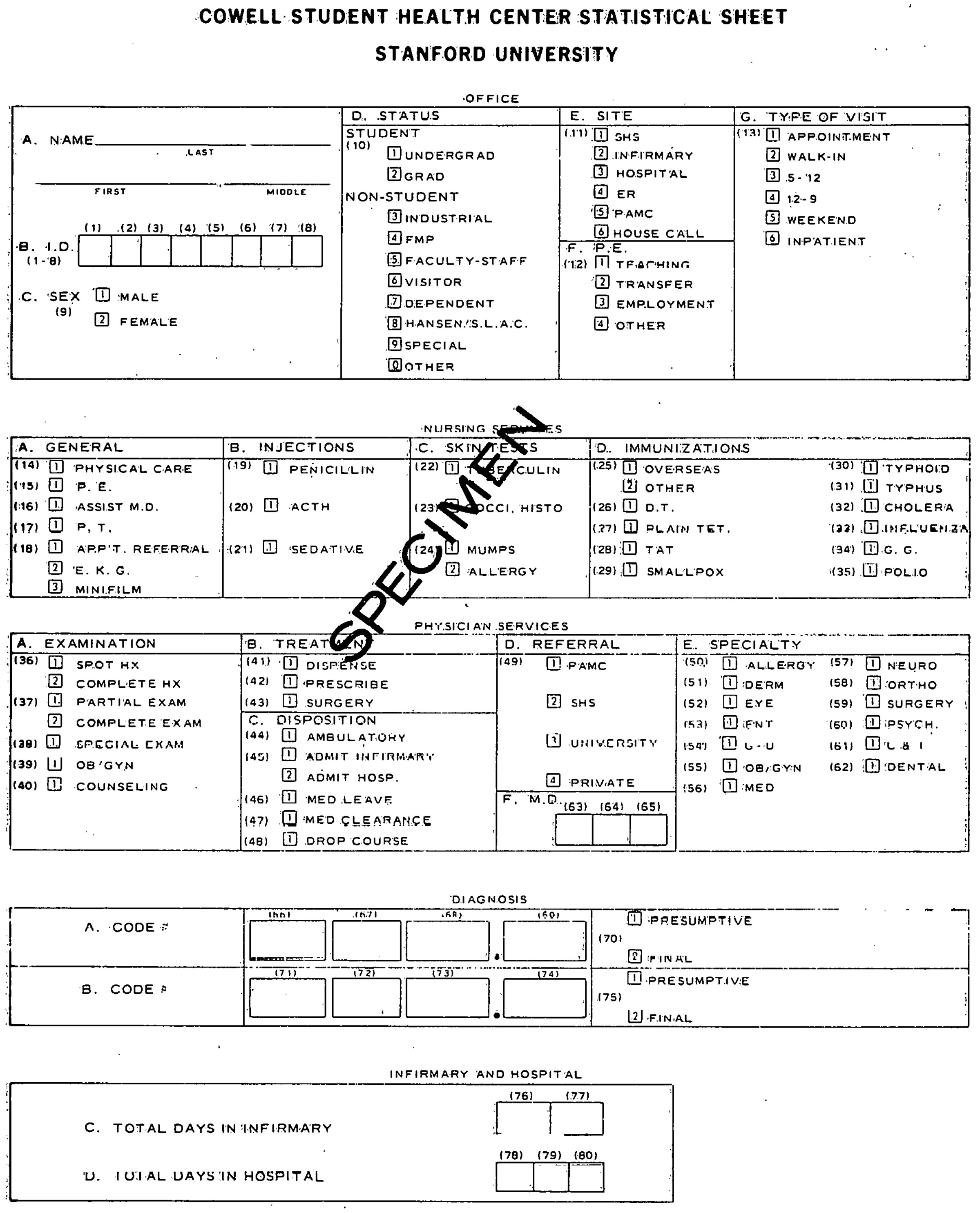
A. Storing and Retrieving Information in the Current SHS System

Information on each patient visit is typed into the on-line computer file system, WYLBUR (Riddle [1968]) by an employee of the SHS. The input terminal, commonly referred to as a WYLBUR terminal, is physically located in a secure area at the SHS offices and access to it is controlled there. Knowledge of the SHS account number, its corresponding WYLBUR keyword, a valid user name, and its password (assigned and maintained by SHS — not the WYLBUR keyword) are all necessary to input data to or output data from the system.

Periodically (every academic quarter or so), a statistical summary is requested from the terminal located at SHS. The program which prints the summary will do so only after it verifies that an authorized user is using his authorized password. In addition, this program requires the user to give the operating system both the SHS account number and its associated keyword. The summary (which includes no patient names or patient identification numbers of any sort) is printed out only at the WYLBUR terminal located at SHS.

Notice that the patient visit data (and associated patient identification number) exists in only three places:

1) on the Cowell Student Health Service statistical sheet (Fig. 5), which is made up for each patient visit to the Health Service, and is kept in a physically secure area at the Student Health Service.
2) on the paper in the WYLBUR terminal (an IBM 2741 Communications Terminal) which is located in a controlled-access area at SHS. The paper is kept under controlled access until it is no longer useful and then is destroyed.
3) on the tape at the campus facility of the Stanford University Computation Center. The information is scrambled on the tape; it is kept "in the clear" only while the SHS statistical programs are actually being executed.



COWELL STUDENT HEALTH CENTER STATISTICAL SHEET
STANFORD UNIVERSITY

OFFICE

A. NAME / B. I.D. / C. SEX	D. STATUS	E. SITE / F. P.E.	G. TYPE OF VISIT
A. NAME ______ LAST ______ FIRST ______ MIDDLE	STUDENT (10) 1 UNDERGRAD, 2 GRAD	(11) 1 SHS, 2 INFIRMARY, 3 HOSPITAL, 4 ER, 5 PAMC, 6 HOUSE CALL	(13) 1 APPOINTMENT, 2 WALK-IN, 3 5-12, 4 12-9, 5 WEEKEND, 6 INPATIENT
B. I.D. (1-8) (1) (2) (3) (4) (5) (6) (7) (8)	NON-STUDENT 3 INDUSTRIAL, 4 FMP, 5 FACULTY-STAFF, 6 VISITOR, 7 DEPENDENT, 8 HANSEN/S.L.A.C., 9 SPECIAL, 0 OTHER	F. P.E. (12) 1 TEACHING, 2 TRANSFER, 3 EMPLOYMENT, 4 OTHER	
C. SEX (9) 1 MALE, 2 FEMALE			

NURSING SERVICES

A. GENERAL	B. INJECTIONS	C. SKIN TESTS	D. IMMUNIZATIONS	
(14) 1 PHYSICAL CARE	(19) 1 PENICILLIN	(22) 1 TUBERCULIN	(25) 1 OVERSEAS, 2 OTHER	(30) 1 TYPHOID
(15) 1 P.E.	(20) 1 ACTH	(23) 1 COCCI, HISTO	(26) 1 D.T.	(31) 1 TYPHUS
(16) 1 ASSIST M.D.	(21) 1 SEDATIVE	(24) 1 MUMPS, 2 ALLERGY	(27) 1 PLAIN TET.	(32) 1 CHOLERA
(17) 1 P.T.			(28) 1 TAT	(33) 1 INFLUENZA
(18) 1 APP'T. REFERRAL, 2 E. K. G., 3 MINIFILM			(29) 1 SMALLPOX	(34) 1 G. G.
				(35) 1 POLIO

SPECIMEN

PHYSICIAN SERVICES

A. EXAMINATION	B. TREATMENT / C. DISPOSITION	D. REFERRAL / F. M.D.	E. SPECIALTY	
(36) 1 SPOT HX, 2 COMPLETE HX	(41) 1 DISPENSE	(49) 1 PAMC	(50) 1 ALLERGY	(57) 1 NEURO
(37) 1 PARTIAL EXAM, 2 COMPLETE EXAM	(42) 1 PRESCRIBE	2 SHS	(51) 1 DERM	(58) 1 ORTHO
(38) 1 SPECIAL EXAM	(43) 1 SURGERY	3 UNIVERSITY	(52) 1 EYE	(59) 1 SURGERY
(39) 1 OB/GYN	C. DISPOSITION (44) 1 AMBULATORY	4 PRIVATE	(53) 1 ENT	(60) 1 PSYCH.
(40) 1 COUNSELING	(45) 1 ADMIT INFIRMARY, 2 ADMIT HOSP.	F. M.D. (63) (64) (65)	(54) 1 G-U	(61) 1 L & I
	(46) 1 MED LEAVE		(55) 1 OB/GYN	(62) 1 DENTAL
	(47) 1 MED CLEARANCE		(56) 1 MED	
	(48) 1 DROP COURSE			

DIAGNOSIS

A. CODE #	(66) (67) (68) . (69)	(70) 1 PRESUMPTIVE, 2 FINAL
B. CODE #	(71) (72) (73) . (74)	(75) 1 PRESUMPTIVE, 2 FINAL

INFIRMARY AND HOSPITAL

C. TOTAL DAYS IN INFIRMARY	(76) (77)
D. TOTAL DAYS IN HOSPITAL	(78) (79) (80)

1468

FIG. 5--Cowell Student Health Service statistical sheet

No card decks are keypunched by non-SHS personnel and then left in unsecured bins to be picked up by couriers and transported to SHS. Human-readable input and output is generated only at the WYLBUR terminal in the secure area at SHS.

We believe that only persons with exceptional knowledge of the operating system at Stanford and the SHS programs themselves and the operating procedures of the SHS can "break" this system to the extent that they obtain meaningful data related to an identificable patient. While the system does not represent the "ultimate" in security, we feel the records are just as secure as those in physical file cabinets at SHS and that the cost paid to maintain this degree of security is not prohibitive.

B. Attaching to the SHS Formulary

Since WYLBUR provides password protection, only those persons knowing both the charge number and the keyword of the Student Health Service are permitted to log in and attempt to use the SHS system. These people are limited to a few SHS personnel and two programmers responsible for maintaining the system. So, in effect, only these people can be attached to the SHS formulary.* Attaching, in this implementation, consists of two stages: (1) logging in successfully to the WYLBUR system, and (2) successfully fetching and starting up an SHS TALK procedure.

C. Formulary Building

Since there are only three formularies in the system described, we decided that it was not worthwhile to write a FORMULARYBUILDER program; the formularies were built manually, and their procedures were linked to the already

*This excludes possible compromise of the system by wiretapping, personnel problems, etc., which are explicitly not handled by the formulary scheme (see Chapter I).

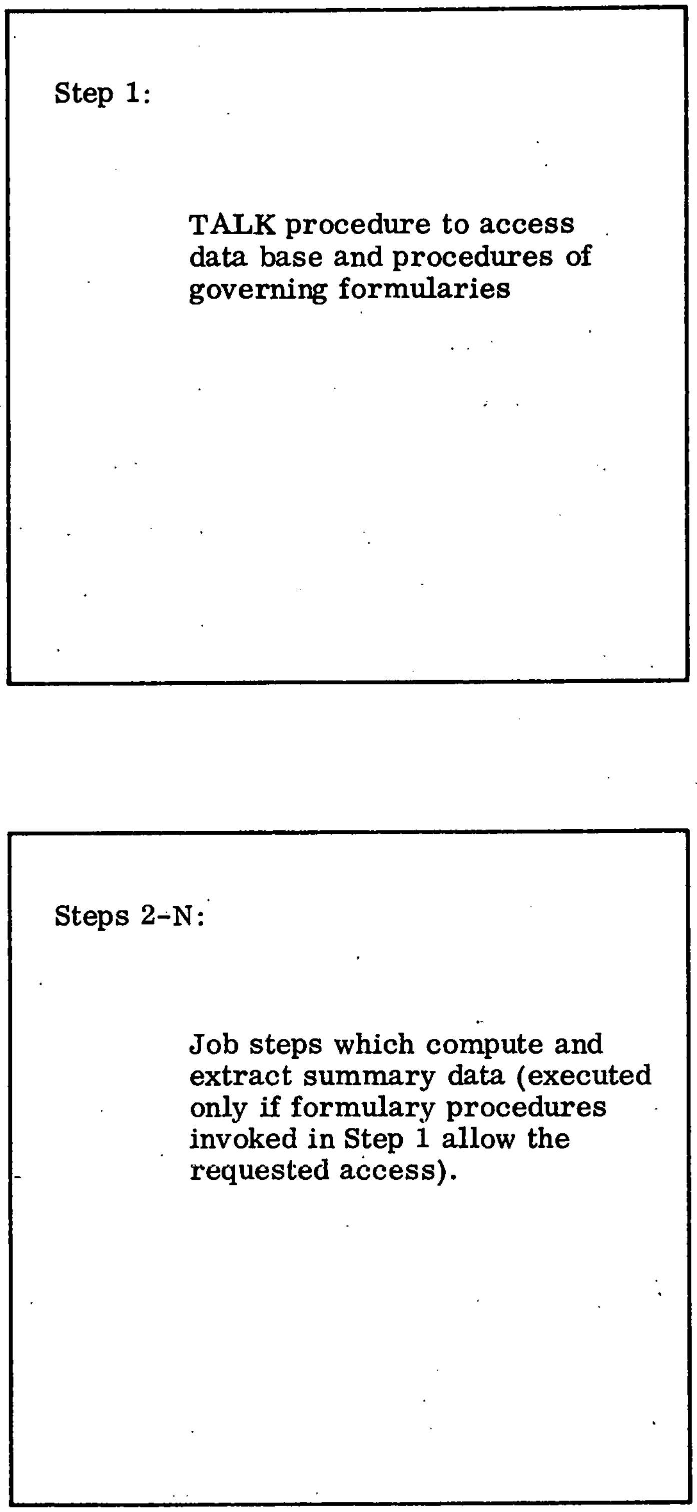
existing SHS statistical procedures. Clearly, such a FORMULARYBUILDER procedure could be written.

D. The TALK Procedure

We shall now limit our discussion to the part of the SHS system which handles requests for the computation and extraction of statistical summary data, though the other parts operate in a similar manner. In particular, only the formulary and TALK procedure relevant to that part of the SHS system will be discussed here, though in fact other TALK procedures and formularies exist in that system. The TALK procedure we shall discuss obtains user and terminal identification from the operating system and user password and authentication sequences interactively from the user. Due to the system characteristics of the Campus Facility of the Stanford Computation Center at the time the procedures were coded,* it was decided to handle the authentication of users in both the TALK procedure and in the CONTROL procedure. The TALK procedure handles interactive authentication, and the CONTROL procedure, running in the batch, performs a final user and password authentication. Note that all of the code necessary to make up a formulary is broken out separately from the code that does the actual data manipulation and statistics gathering (see Fig. 6). This makes it easy for systems programmers to replace or modify formularies without any modification of the actual application program. In fact the code for the formularies was added to previously existing SHS application programs with no change in these application programs.

The TALK procedure was written in CYVYL, an on-line interactive language designed for user mainly in computer-assisted instruction applications. It engages

*At the time these procedures were coded, it was relatively difficult for procedures written in certain languages to communicate with procedures written in certain other languages and with user terminals. This situation should disappear soon, and then the authentication will be easily handled entirely by the CONTROL procedure.



Step 1:

TALK procedure to access data base and procedures of governing formularies

Steps 2-N:

Job steps which compute and extract summary data (executed only if formulary procedures invoked in Step 1 allow the requested access).

FIG. 6--Skeleton of terminal-initiated job to compute and extract summary data the user in a dialogue to verify an authentication sequence (see Fig. 7). TALK also obtains printing parameters and an output heading from the user. It then initiates the program which will fetch information from the data base and compute statistical summary data.

E. Procedures of the SHS Formulary

Once someone is attached to the SHS formulary (i.e., has logged in successfully to WYLBUR), he still must provide a valid user identification and the corresponding password (not the one used to log in to WYLBUR) in order to put any information into or get any information out of the system. This information is obtained and checked by the Fortran subroutine CONTROL (Exhibit 3 of Appendix A) which is invoked by the system ACCESS procedure and which serves as the CONTROL procedure of the SHS formulary. This function is invoked at both data input and at data output times.

The Fortran subroutine SCRAMBLE (Exhibit 4 of Appendix A) scrambles the data at data input time. It serves as the SCRAMBLE procedure of the SHS formulary.

The Fortran subroutine UNSCRAMBLE (Exhibit 5 of Appendix A) unscrambles the data when it is read in for use in statistical computations or for outputting purposes. It serves as the UNSCRAMBLE procedure of the SHS formulary.

VIRTUAL is generally used to map an internal name into a virtual address. In the SHS system however, there is only one internal name associated with raw patient data (as opposed to formularies). This internal name, NEXTRECORD, is the only one which is ever mapped into a virtual address by VIRTUAL. In the SHS system, the virtual address is the same as the internal name. So VIRTUAL is very simple in this system; it is in fact, effectively, the identity function (Exhibit 6 of Appendix A).

```
WHAT COURSE DO YOU WANT? rsubmit
QUEUED
QUEUED
 SUBMISSION PROCEDURE HAS BEEN INITIATED.  WAIT!
 USER ID OBTAINED FROM OPERATING SYSTEM-- WILL BE VALIDATED LATER BY THE CONTROL PROCEDURE.
 TERMINAL ID OBTAINED FROM OPERATING SYSTEM AND APPROVED.
 ACCOUNT NUMBER OBTAINED FROM OPERATING SYSTEM AND APPROVED.
 YOU ARE PERMITTEDTO USE THE SHS PROGRAM AT THIS TIME OF DAY.

 WHAT IS YOUR HEALTH SERVICE PASSWORD?   ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
 PASSWORD WILL BE VALIDATED LATER BY THE CONTROL PROCEDURE.

 PLEASE RESPOND TO AUTHENTICATION SEQUENCE:
     8213   -- ?   ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
     4932   -- ?   ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
DO YOU WISH TO PULL ID NUMBERS?  yes
PLEASE GIVE THE ADDITIONAL ID NUMBER VALIDATION KEY:  ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
 AUTHENTICATION SEQUENCE VALID.
TALK PROCEDURE APPROVES THIS USER AND TERMINAL.

WHEN REQUESTED, PLEASE TYPE EACH DISEASE NUMBER YOU WISH TO PULL THE IDS FOR.
WHEN DONE, JUST HIT "CARRIAGE RETURN".
   DISEASE NUMBER = ?  0020
   DISEASE NUMBER = ?  y020
   DISEASE NUMBER = ?
HOW MANY COPIESOF THE PRINTOUT DO YOU WANT?  1
WHAT QUARTER WILL THIS RUN COVER?  fall 1984
937 IS YOUR JOB NUMBER.
YOUR JOB HAS BEEN SUBMITTED AND WILL BE READY TOMORROW.
WE"LL LOG YOU OFF FOR NOW.  THANK YOU AND GOODBYE.
COMPUTE TIME = 64.36 SECONDS
MEMORY USAGE = 988.50 PAGE-SECONDS
I/O ACTIVITY = 0 UNITS
EDITING TIME = 22.74 SECONDS
ELAPSED TIME = 00:23:55
END OF SESSION
```

1557A7

FIG. 7--User dialogue with TALK procedure

The Fortran subroutine CONTROL (Exhibit 3 of Appendix A) serves as the CONTROL procedure of the SHS formulary. The CONTROL procedure in this implementation verifies the password of each user. It allows unlimited access to certain users, provided that they give the proper password. Other users are restricted as to the data they receive. (Note in Fig. 7, for example, the additional authorization required to pull student identification numbers. This additional authorization is currently checked by TALK but will eventually be handled by CONTROL.)

F. Primitive Operations

The FETCH and STORE operations in the SHS system merely read and write the next record on a sequential data set. Only the internal name NEXTRECORD is acceptable to FETCH or STORE. FETCH and STORE in the SHS system are shown in Exhibit 2 of Appendix A.

G. Realization of the ACCESS Procedure

The Fortran subroutine ACCESS (Exhibit 1 of Appendix A) is merely the FORTRAN implementation of the ACCESS algorithm for this particular system.

CHAPTER V

A NOTE ON THE COST OF SOME PRIVACY SAFEGUARDS

As mentioned in Chapter II, a desirable property for an access control model is that it be sufficiently modular to permit cost-effectiveness experiments to be undertaken. In this way the model would serve as a vehicle for exploring questions of cost with respect to various privacy safeguards.

Using the formulary model, an experiment was run on the IBM 360/91 computer system at the SLAC Facility of Stanford University Computation Center. This experiment was designed to obtain figures on the additional overhead due to using the formulary method and on the costs of encoding data (and conversely the costs of decoding data).

A tape containing 10,001 80-character card images in clear (unscrambled) format was first generated. Then 10,000 of the 80-character records were sequentially read in, scrambled, and the (encoded) card images written out onto a new output tape. Appendix B shows the FORTRAN program used to do this job, and also the printout of the timing results.

Three different scrambling algorithms were used: algorithm 0 — no scrambling at all; algorithm 1 — simple exclusive-or operations with predetermined random numbers which did not vary from one record to the next; and algorithm 2 — exclusive-or operations with the concatenation of four small pseudo-random numbers which did vary over records. Each of these scrambling algorithms was timed twice: first without going through the central ACCESS procedure of the formulary model (and therefore without invoking the procedures of the attached formulary which it calls), and then using the central ACCESS procedure and the formulary model.

Ten trials were run of the experiment. The timing results are shown in Table II, and the averages summarized in Table III.

TABLE II

Timing Results of Cost Experiment

N = Formulary Method Not Used

F = Formularies Used

Scrambling Method \ Elapsed Wall Clock Time (sec.)	N	F	N	F	N	F	N	F	N	F	N	F	N	F	N	F	N	F	N	F
0	19.39	19.42	19.88	19.44	19.69	19.42	19.44	19.44	20.58	19.42	Note A	19.40	20.12	19.87	20.55	19.40	19.87	19.42	19.40	19.87
1	19.39	19.42	19.45	19.90	19.44	19.44	19.44	19.42	19.42	19.42	19.42	19.42	Note B	19.87	19.40	19.42	20.57	19.42	19.40	19.87
2	21.02	19.44	19.42	19.42	19.87	19.42	19.40	19.88	19.40	19.88	19.40	19.42	20.28	19.66	19.40	20.78	19.40	19.40	19.40	19.40

Note A: No time available, since timer overflowed at this point.

Note B: Time of 28.60 seconds is not meaningful, since a tape write error occurred and recovery procedures for this were also invoked during this trial.

TABLE III

Average Timing Results of Cost Experiment

MEAN VALUES

Scrambling Method	No Formularies	Formularies Used
0	19.88 sec (9 trials)	19.51 sec (10 trials)
1	19.55 sec (9 trials)	19.56 sec (10 trials)
2	19.67 sec (10 trials)	19.58 sec (10 trials)

MEAN VALUE (58 trials) = 19.64 sec

We see from those tables that there was no significant difference in the wall-clock times needed to encode 10,000 records. That is, the times used were about the same regardless of which of the three scrambling algorithms were used and regardless of whether the formulary method was used. Additional overhead caused by use of the formulary method was all taken up by the input/output wait time. We conjecture that this will be the case in general. All times are total wall-clock times used from the time the first clear record was read in until the time the last encoded record was written out onto the output tape. <u>All waits for input/output, etc., are included in these times.</u> The times are not directly related to central processor cycles. They are <u>wall-clock time</u> on a system where this was the only job running in addition to the operating system (OS/360), spooling subsystem (HASP), and remote file management/job entry subsystem (CRBE). The experiment was carried out in this manner in order to get a <u>high</u> estimate of the incremental cost involved in scrambling a large number of cards. In a multiprogramming system the actual time used in encoding could be overlapped with input/output tasks from other jobs and therefore would not be nearly so costly. On the other hand, if CPU cycles are a major cost factor, another experiment should be carried out to determine this incremental cost.

In this worst case we see that 10,000 cards were scrambled in an average of 19.64 seconds. We can put it another way; the incremental cost of encoding (or decoding) <u>one card image</u> on this system is 0.001964 seconds. Under the existing rate structure at the Stanford Computation Center, it then costs approximately one-twentieth of a cent to encode (or decode) each card image. Therefore, encoding one card image (80 bytes of information) for each of the 20,000,000 residents of the State of California would take only 39,280 seconds (less than 11 hours) and would cost under $11,000. These results seem to indicate that the incremental cost of scrambling information in a large computer data base where fetch accesses (and hence unscrambling operations) are infrequent is infinitesimal.

Clearly, it will be easy to use the formulary model to carry out various other experiments as well, to ascertain the relative costs of diverse encoding methods and data accessing schemes. We expect to do more of this in the future.

CHAPTER VI

CONCLUSIONS

A. Summary

We have defined and demonstrated a model of access control which allows real-time decisions to be made about privileges granted to users of a data base. Raw data need appear only once in the data base and arbitrarily complex access control programs can be associated with arbitrarily small fragments of this data.

The desirable characteristics for an access control method laid out in Chapter II are all present (though we have not yet run enough experiments to make general statements about efficiency):

1) No arbitrary constraint (such as segmentation or sensitivity levels) is imposed on data or programs.

2) The method allows control of individual data elements. Its efficiency depends on the specific system involved and the particular controls used. As seen in Chapter V, very little performance degradation due to increased overhead was added by the introduction of formularies to the tape-based system in the example there.

3) No extra storage or time is required to describe data which the user does not desire to protect.

4) The method is machine-independent and also independent of file structure. The efficiency of each implementation depends mainly on the adequacy of the formulary method for the particular data structures and application involved.

5) Chapters IV and V certainly demonstrate the modularity of the formulary model and its ability to support cost-effectiveness experiments.

B. Future Work

More experiments should be carried out to determine the amount of additional system overhead introduced by user formularies. This will vary over data structures and over data base systems. In particular, actual costs in additional central processor cycles should be determined for various hardware systems.

Criteria of system efficiency, degree of control required, etc., should be developed to determine the extent of usefulness of the formulary method. Some preliminary work has already been done in this area (Wortman and Hoffman [1969]).

Using the formulary method, cost measures for scrambling and unscrambling techniques and for threat monitoring (Hoffman [1969]) subsystems can be developed in the same manner that the cost measures of Section V were developed.

To observe the full capabilities of the method and its potential for storage efficiency, a system should be developed where quite a number of users share several formularies. Also, the problem of users granting limited capabilities to other users, these new users granting even more limited capabilities to still other users, etc., and all this being done while access control decisions are being made in real time by procedures, should be investigated in more detail. Once this problem of granting limited privileges is solved, we will see much more controlled sharing of mutually useful programs and data. The implications here for proprietary software and for application-oriented data banks are very great.

A most promising area for future work is the development of a generalized resource allocation system which incorporates the formulary model as a first stage and a sophisticated scheduler as a second stage. Such a system is currently being investigated by R. D. Russell at SLAC.

Finally, since the central ACCESS procedure is fixed, hardware or micro-programmed implementations of it could be built which would greatly decrease the overhead in central processor cycles involved in using the formulary method.

REFERENCES

Arvas, Christer [1968]. Joint Use of Databanks. Statistiska Centralbyran, Stockholms Universitet, Ukas P5, Sweden, Report No. 6.

Babcock, J. D. [1967]. A brief description of privacy measures in the RUSH time-sharing system. Proc. AFIPS 1967 Spring Joint Comput. Conf., Vol. 30, Thompson Book Co., Washington, D. C., 301-302.

Bingham, Harvey W. [1965]. Security techniques for EDP of multilevel classified information. Document RADC-TR-65-415, Rome Air Development Center, Griffiss Air Force Base, New York, Dec. 1965. (Unclassified)

Castleman, P. A. [1967]. "User-defined syntax in a general information storage and retrieval system," in Information Retrieval, The User's Viewpoint, An Aid to Design, International Information, Inc.

Crisman, P. S. (ed.) [1965]. The Compatible Time-Sharing System — A Programmer's Guide (Second ed.). MIT Press, Cambridge, Massachusetts.

Dennis, J. B. and Van Horn, E. C. [1966]. Programming semantics for multiprogrammed computation. Comm. ACM 9, 3(March 1966), 143-155.

Dijkstra, E. W. [1965]. Cooperating sequential processes. Department of Mathematics, Technological University, Eindhoven, The Netherlands.

Evan, D. C. and Le Clerc, J. Y. [1967]. Address mapping and control of access in an interactive computer. Proc. AFIPS 1967 Spring Joint Computer Conf., Vol, 30, Thompson Book Co., Washington, D. C., 23-30.

Ewing, R. G. and Davies, P. M. [1964]. An associative processor. Proc. IFIPS 1964 Fall Joint Computer Conference.

Feldman, J. A. [1965]. Aspects of Associative Processing, Technical Note 1965-13. Lincoln Laboratory, MIT, Cambridge, Massachusetts.

Gall, R. G. [1964]. A hardware-integrated GPC/search memory. Proc. IFIPS 1964 Fall Joint Computer Conference.

Giering, R. H. [1967]. Information Processing and the Data Spectrum. Technical Note DTN-68-2, Data Corporation, Arlington, Virginia.

Graham, R. M. [1968]. Protection in an information processing utility. Comm. ACM 11, 5 (May 1968), 365-369.

Habermann, A. N. [1969]. Prevention of system deadlocks. Comm. ACM 12, 7 (July 1969), 373.

Hoffman, Lance J. [1969]. Computers and privacy: A survey. Computing Surveys 1, 2 (June 1969).

Hsiao, D. K. [1968]. A File System for a Problem Solving Facility. Ph.D. Dissertation in Electrical Engineering, Univ. of Pennsylvania, Philadelphia.

Iliffe, J. K. [1968]. Basic Machine Principles, MacDonald and Co. (London).

Jones, R. S. [1968]. DATA FILE TWO — A data storage and retrieval system. Proc. SJCC 1968, 171-181.

Kahn, D. [1967]. The Codebreakers. MacMillan, New York.

Kellogg, C. H. [1968]. A natural language compiler for on-line data management. Proc. FJCC 1968, 473-492.

Lampson, B. W. [1969]. Dynamic Protection Structures. Proc. AFIPS 1969 Fall Joint Computer Conference, pp. 27-38.

Lesser, V. R. [1968]. A multi-level computer organization designed to separate data-accessing from the computation. Technical Report No. CS90. Computer Science Department, Stanford University, Stanford, California, March 1968.

McAteer, J. et al. [1968]. Associative Memory System Implementation and Characteristics. Proc. IFIPS 1964 Fall Joint Computer Conference.

Miller, W. F. and Hoffman, L. J. [1969]. A method of extracting record-specific information from "statistical" data banks. CGTM-67, Stanford Linear Accelerator Center, Computation Group, Stanford, California.

Raffel, J. I. and Crowther, T. S. [1964]. A proposal for an associative memory using magnetic films. IEEE Trans. on Electronic Computers, EC-13, No. 5.

Riddle, William E. [1968]. WYLBUR, Stanford University Computation Center Text Editor, Appendix E to Users Manual, Stanford Computation Center Campus Facility, Stanford, California.

Rovner, P. D. and Feldman, J. A. [1968]. The Leap language and data structure. Proc. IFIP Congress 1968, C73-C77.

Shannon, C. E. [1949]. Communication theory of secrecy systems. Bell System Tech. J. 28, 656-715.

Shoshani, A. and Bernstein, A. J. [1969]. Synchronization in a parallel-accessed data base. Comm. ACM 12, 11 (November 1969), 604-607.

Skatrud, R. O. [1969]. The application of cryptographic techniques to data processing. Proc. AFIPS 1969 Fall Joint Computer Conference, 111-117.

Stone, M. G. [1968]. TERPS-file independent enquiries. Computer Bulletin 11, 4 (March 1968), 286-289.

Weissman, Clark [1969]. Security Controls in the ADEPT-50 Time-Sharing System. Proc. AFIPS 1969 Fall Joint Computer Conference, 119-133.

APPENDIX A

EXAMPLES OF PROCEDURES USED BY A PARTICULAR INSTALLATION

This appendix contains listings of procedures which are used in the Cowell Student Health Service system which operates under the OS/360 Operating System on the IBM 360/67 at the Stanford University Computation Center Campus Facility. Except for the ACCESS procedure, all of the algorithms and coding were supplied by the Student Health Service. They supplied the coding for the ACCESS procedure, but its algorithm was fixed, of course; its ALGOL version is given in Section K of Chapter III.

Some data (including key privacy data) in named common areas are intialized in a BLOCK DATA subprogram (not shown) which is similar to the BLOCK DATA subprogram in Appendix B. The subprogram shown there, however, does <u>not</u> contain key privacy data for the SHS system or for any other system.

```
      SUBROUTINE ACCESS(INFO,INTNAME,VALUE,LENGTH,OPN,COMPCODE)        00524500
C                                                                      00524600
C THIS PROCEDURE TAKES AS INPUT THE INTERNAL NAME INTNAME AND          00524700
C DOES THE FOLLOWING:                                                  00524800
C                                                                      00524900
C IF IOPN=FETCHP, VALUE IS SET TO THE VALUE OF THE                     00525000
C DATUM REPRESENTED BY INTNAME.                                        00525100
C                                                                      00525200
C IF IOPN=STOREP, THE VALUE OF THE DATUM REPRESENTED                   00525300
C BY INTNAME BECOMES VALUE.                                            00525400
C                                                                      00525500
C IF IOPN=FLOCKP, SLOCKP, UNLFEP, OR UNLSTP, THE DATUM                 00525600
C REPRESENTED BY INTNAME IS RESPECTIVELY LOCKED TO FUTURE              00525700
C FETCHES, LOCKED TO FUTURE STORES, UNLOCKED TO FUTURE                 00525800
C FETCHES, OR UNLOCKED TO FUTURE STORES.                               00525900
C (LOCKING A DATUM LOCKS OUT ALL USER/TERMINAL COMBINATIONS            00526000
C  EXCEPT THE ONE THAT SET THE LOCK.)                                  00526100
C                                                                      00526200
C THE LENGTH OF VALUE IS LENGTH.                                       00526300
C                                                                      00526400
C ACCESS RETURNS IN COMPCODE THE FOLLOWING COMPLETION CODES:           00526500
C                                                                      00526600
C            1 NORMAL EXIT, NO ERROR                                   00526700
C            2 UNLOCK OPERATION REQUESTED BY USER/TERMINAL             00526800
C              WHO/WHICH DID NOT SET LOCK                              00526900
C            3 IOPN OPERATION PERMITTED BUT GAVE ERROR WHEN ATTEMPTED   00527000
C            4 ATTEMPT TO UNLOCK DATA WHICH IS NOT LOCKED IN GIVEN MANNER00527100
C            5 CANNOT HANDLE ANY MORE USER CONTROL BLOCKS              00527200
C            6 ATTEMPT TO DETACH NONEXISTENT USER/TERMINAL/FORMULARY   00527300
C              COMBINATION                                             00527400
C            7 IOPN OPERATION PERMITTED BUT WAS UNABLE TO BE CARRIED OUT 00527500
C              SINCE THE DATUM WAS LOCKED TO PREVENT SUCH AN OPERATION 00527600
C            8 CANNOT PUT ON LOCK AS REQUESTED SINCE LOCKLIST IS FULL  00527700
C            9 DATUM ALREADY LOCKED BY THIS USER AND TERMINAL          00527800
C           10 VIRTUAL PROCEDURE CANNOT TRANSLATE INTERNAL NAME INTO   00527900
C              VIRTUAL ADDRESS                                         00528000
C           11 IOPN OPERATION NOT PERMITTED ON DATUM REPRESENTED       00528100
C              BY INTNAME; DETECTION CARRIED OUT BY THE CONTROL        00528200
C              PROGRAM OF THE ATTACHED FORMULARY                       00528300
C           12 END OF DATA SET ENCOUNTERED ON FETCH ATTEMPT            00528400
C                                                                      00528500
C                                                                      00528600
C                                                                      00528700
C FORMAT OF LOCKLIST (LLIST) IS:                                       00528800
C                                                                      00528900
C    ENTRY 1    ENTRY 2    ...    ENTRY N    ENTRY N+1 ... ENTRY 100   00529000
C INAME                                                                00529100
C OPN                                                                  00529200
C USER/TERMINAL INFORMATION                                            00529300
C                                                                      00529400
C OPERATIONS --                                                        00529500
C  1 FETCH, 2 STORE, 3 BOTH FETCH AND STORE                            00529600
C INAME=-1 IMPLIES THAT SLOT ON LOCKLIST IS EMPTY                      00529700
C                                                                      00529800
C                                                                      00529900
C                                                                      00530000
      IMPLICIT INTEGER(A-Z)                                            00530100
      COMMON/CURUCB/IUCB                                               00530200
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,             00530300
     1   FORM1,FORM2,FORM3,                                            00530400
     2   NEXTALL,SAMEALL,                                              00530500
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FLOCKP,SLOCKP,ATTACHP,DETACHP     00530600
      COMMON/UCB/ISTDUCB                                               00530700
      COMMON/OWN1/UCB1,LLIST,LSI                                       00530800
      INTEGER LLIST(4,100)                                             00530900
      INTEGER UCB1(100,3)                                              00531000
C ************* 100 IS MAXUSERS, NUCB IS 3                             00531100
      INTEGER ISTDUCB(3)                                               00531200
      INTEGER INFO(ITALK)                                              00531300
      INTEGER VALUE(20)                                                00531400
C **** DIMENSION IS LENGTH STORAGE ELEMENTS, IN THIS CASE 80           00531500
C STORAGE ELEMENTS.  THIS MUST BE SPECIFIED AS 20 FORTRAN ELEMENTS DUE 00531600
C TO REQUIREMENTS OF THE FORTRAN LANGUAGE.                             00531700
      INTEGER INTNAME,LENGTH,OPN,COMPCODE                              00531800
C                                                                      00531900
      INTEGER IUCB(3)                                                  00532000
C DIMENSION SHOULD BE NUCB BUT FORTRAN DOES NOT ALLOW THAT CONSTRUCTION 00532100
      INTEGER RESLT(20)                                                00532200
C **** DIMENSION IS LENGTH STORAGE ELEMENTS, IN THIS CASE 80           00532300
C STORAGE ELEMENTS.  THIS MUST BE SPECIFIED AS 20 FORTRAN ELEMENTS DUE 00532400
C TO REQUIREMENTS OF THE FORTRAN LANGUAGE.                             00532500
C                                                                      00532600
```

Exhibit 1--FORTRAN Version of ACCESS Procedure

The ACCESS procedure has the following characteristics:

a. only procedure which directly calls FETCH and STORE primitives.
b. only procedure which performs locking and unlocking operations.
c. all requests for operations on data base must go through it.

Lines 5247-5284 above describe the operation of the ACCESS procedure.

```
      COMPCODE=1
      ISLOT=0
C FIRST TRY TO RECOGNIZE USER/TERMINAL COMBINATION IN INFO ARRAY
      DO 1 I=1,MAXUSERS
      II=I
      IF (UCB1(I,1) .EQ. -2) GO TO 2
C END LIST OF UCBS
      IF (UCB1(I,1) .EQ. -1) GO TO 3
      DO 4 J=1,ITALK
      IF (UCB1(I,J) .NE. INFO(J)) GO TO 1
    4 CONTINUE
      GO TO 6
    2 IF (ISLOT .NE. 0) GO TO 7
      IF (II .NE. MAXUSERS) UCB1(II+1,1)=-2
      GO TO 16
    3 ISLOT=II
C REMEMBER THIS SLOT IF VACANT
    1 CONTINUE
      IF ( ISLOT .EQ. 0) GO TO 805
C CANNOT HANDLE ANY MORE UCBS
    7 II=ISLOT
   16 DO 5 K=1,ITALK
    5 UCB1(II,K)=INFO(K)
      K1=ITALK+1
      DO 8 K=K1,NUCB
    8 UCB1(II,K)=ISTDUCB(K)
    6 DO 9 I=1,NUCB
    9 IUCB(I)=UCB1(II,I)
C SET UP POINTERS TO APPROPRIATE USER CONTROL BLOCK
C USER AND TERMINAL NOW ASSOCIATED WITH POSITION II OF UCB TABLE.
      IF((IUCB(NUCB) .NE. INTNAME).AND. (OPN .EQ. DETACHP)) GO TO 806
C ATTEMPT TO DETACH USER/TERMINAL/FORMULARY COMBINATION NOT CURRENTLY
C ATTACHED
      CALL CONTROL(INTNAME,OPN,YESNO,OTHER)
      IF (YESNO .GT. 1) GO TO 811
C RETURN 11 IF CONTROL DOES NOT PERMIT OPERATION
      IF (OPN .EQ. ATTACHP) GO TO 10
      IF (OPN .EQ. DETACHP) GO TO 11
      IF((OPN .NE. UNLFEP) .AND. (OPN .NE. UNLSTP)) GO TO 12
      I=IDXLL(INTNAME,OPN)
C FIND INTERNAL NAME ON LOCKLIST
      IF (I .LE. 0) GO TO 804
C CANNOT FIND IT IF I .LE. 0
      DO 13 J=1,ITALK
      IF (LLIST(2+J,I) .NE. IUCB(J)) GO TO 802
C JUMP IF UNLOCK REQUESTED BY USER/TERMINAL WHO/WHICH DID NOT SET LOCK
   13 CONTINUE
      LLIST(1,I)=-1
C UNDO THE LOCK AND MARK SLOT IN UCB ARRAY EMPTY
      GO TO 801
   12 IF (TESTSE(CS1) .EQ. -1) GO TO 12
C-------------------------------------------------
C  ENTER CRITICAL SECTION FOR LOCKING OUT DATUMS
C-------------------------------------------------
      I=IDXLL(INTNAME,OPN)
C GET RELATIVE LOCATION OF LOCKED DATUM IN LOCKLIST
      IF(I .LE. 0) GO TO 14
C IF DATUM NOT LOCKED TO THIS OPN, GO TO 14
```

1557[illegible]

Exhibit 1--FORTRAN Version of ACCESS Procedure (cont'd.)

```
C NOW SEE IF DATUM FOUND ON LOCKLIST LOCKED BY THIS USER AND TERMINAL
      DO 15 J=1,ITALK
      IF (LLIST(2+J,I) .NE. IUCB(J)) GO TO 807
   15 CONTINUE
      IF((OPN .EQ. FLOCKP) .OR. (OPN .EQ. SLOCKP)) GO TO 809
   14 I=-I
      IF ((OPN .NE. FLOCKP) .AND. (OPN .NE. SLOCKP)) GO TO 18
C JUMP IF NOT A LOCK OPERATION
      IF (I .EQ. 0) GO TO 808
      K1=2
      IF (OPN .EQ. FLOCKP) K1=1
      LLIST(2,I)=K1
C SET APPROPRIATE LOCK
      DO 20 J=1,ITALK
   20 LLIST(2+J,I)=IUCB(J)
C PLACE USER AND TERMINAL ID INTO LOCKLIST
      LLIST(1,I)=INTNAME
C PLACE INTERNAL NAME ON LOCKLIST
      GO TO 801
C
   18 CALL VIRTUAL(INTNAME,DATUM,OTHER,COMP)
C VIRTUAL RETURNS IN DATUM THE VIRTUAL ADDRESS OF THE DATUM SPECIFIED
      IF (COMP .GT. 1) GO TO 810
C JUMP IF ERROR RETURN FROM VIRTUAL
      IF (OPN .EQ. STOREP) GO TO 21
      CALL FETCH(DATUM,RESLT,LENGTH,COMP)
      IF (COMP .EQ. 2) GO TO 812
C JUMP TO 812 IF END OF DATA SET ENCOUNTERED
      IF (COMP .GT. 1) GO TO 803
      CALL UNSCRAMBLE(RESLT,LENGTH,COMP,VALUE,N)
      IF (COMP .GT. 1) GO TO 803
      GO TO 801
   21 CALL SCRAMBLE(VALUE,LENGTH,COMP,RESLT,N)
      IF (COMP .GT. 1) GO TO 803
C OPERATION PERMITTED BUT GAVE ERROR WHEN ATTEMPTED
C
C NOW PERFORM A PHYSICAL WRITE OF N STORAGE UNITS TO THE BLOCK STARTING
C AT RESLT
      CALL STORE(DATUM,RESLT,N,COMP)
      IF (COMP .GT. 1) GO TO 803
      GO TO 801
   10 UCB1(II,NUCB)=INTNAME
      GO TO 801
   11 UCB1(II,1)=-1
C DETACH FORMULARY
C (THIS LEAVES AN OPEN SLOT IN THE UCB TABLE)
      GO TO 801
C
  812 COMPCODE=COMPCODE+1
  811 COMPCODE=COMPCODE+1
  810 COMPCODE=COMPCODE+1
  809 COMPCODE=COMPCODE+1
  808 COMPCODE=COMPCODE+1
  807 COMPCODE=COMPCODE+1
  806 COMPCODE=COMPCODE+1
  805 COMPCODE=COMPCODE+1
  804 COMPCODE=COMPCODE+1
  803 COMPCODE=COMPCODE+1
  802 COMPCODE=COMPCODE+1
  801 CS1=1
C ------------------------------------------------------------
C LEAVE CRITICAL SECTION FOR LOCKING OUT DATUMS
C ------------------------------------------------------------
      RETURN
      END
```

1557[illegible]

Exhibit 1--FORTRAN Version of ACCESS Procedure (cont'd.)

```
      INTEGER FUNCTION IDXLL(INTNAME, OPN)
      IMPLICIT INTEGER(A-Z)
      INTEGER INTNAME,OPN
C IDXLL, GIVEN AN INTERNAL NAME INTNAM AND AN OPERATION  OPN,
C RETURNS THE RELATIVE P(SITION OF INTNAM ON THE LOCKLIST IF
C IT IS LOCKED IN A MANNER AFFECTING OPERATION  OPN.  OTHERWISE,
C IDXLL RETURNS THE NEGATION OF THE FIRST EMPTY RELATIVE LOCATION
C ON THE LOCKLIST.  IF THE LOCKLIST IS FULL AND THE INTNAM/ OPN
C COMBINATION IS NOT FOUND, IDXLL RETURNS 0.
C
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,
     1  FORM1,FORM2,FORM3,
     2  NEXTALL,SAMEALL,
     3  FETCHP,STOREP,UNLFEP,UNLSTP,FLOCKP,SLOCKP,ATTACHP,DETACHP
C
      COMMON/OWN1/UCB1,LLIST,CS1
      INTEGER LLIST(4,100)
      INTEGER UCB1(100,3)
      J=2
      IF((OPN .EQ. FETCHP) .OR. (OPN .EQ. UNLFEP) .OR. (OPN .EQ. FLOCKP)
     1    )  J=1
      FIRSTEMPTY=0
      IDXLL=0
      DO 1 I=1,MAXLLIST
      II=I
      K=LLIST(1,I)
      IF (K .EQ.-1) FIRSTEMPTY=I
      IF((K .EQ. INTNAME) .AND.(LLIST(2,I) .EQ. J)) GO TO 4
    1 CONTINUE
    2 IF (FIRSTEMPTY .NE. 0) IDXLL=-FIRSTEMPTY
      RETURN
    4 IDXLL=II
    5 RETURN
      END
```

```
TESTSE START 0
* TESTSE IS AN INTEGER FUNCTION DESIGNATOR CALLABLE FROM FORTRAN
* VIA THE CALL
*                       J=TESTSE(I)
* I IS A VARIABLE OF TYPE INTEGER*4.  J CONTAINS, ON RETURN,
* -1 ONLY IF THE CONDITION CODE WAS 1 AFTER EXECUTING THE TS OPERATION
* ON I.  THE LEFTMOST BYTE OF I IS SET TO ALL ONES ON
* RETURN FROM TESTSE.
*
* THANKS TO JOHN EHRMAN FOR THE CODING OF THIS.
*
       L      1,0(0,1)
       TS     0(1)
       BALR   0,0
       SLL    0,3
       SRA    0,31
       BR     14
        END
```

1468A16

```
      SUBROUTINE FETCH(IADOR, IVALUE, LENGTH, ICOMPL)
C
C --  FETCH PRIMITIVE --
C THIS PRIMITIVE FETCHES THE VALUE WHICH IS CONTAINED IN THE
C STORAGE LOCATIONS STARTING AT VIRTUAL ADDRESS IADOR AND RETURNS
C THE LENGTH STORAGE ELEMENTS (BYTES) THIS VALUE TAKES IN VALUE.
C UPON COMPLETION. THE COMPLETION CODE ICOMPL IS SET TO:
C
C           1 IF NORMAL EXIT
C           2 END OF DATA SET ENCOUNTERED WHEN PHYSICAL READ ATTEMPTED
C           3 IF LENGTH TOO BIG (>80 BYTES FOR THIS IMPLEMENTATION)
C           4 ILLEGAL VIRTUAL ADDRESS TO FETCH FROM
C           5 ERROR WHEN ATTEMPTING TO DO PHYSICAL READ
C
C ******* CERTIFIED 20 MAY 1969
C
      IMPLICIT INTEGER(A-Z)
      COMMON/CONSTANTS/NUCB,NFORM, MAXUSERS,MAXLLIST,ITALK,
     1   FORM1.FORM2.FORM3,
     2   NEXTALL,SAMEALL,
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FANCLP,SANDLP,ATTACHP,DETACHP
C
      INTEGER IVALUE(LENGTH)
      IF ((LENGTH .GT. 80) .OR. (LENGTH .LT. 0)) GO TO 3
      ICOMPL=1
      IF (IADOR .NE. NEXTALL) GO TO 4
      IF (LENGTH .EQ. 0) RETURN
C NEXT RECORD IS DESIRED SO PHYSICALLY READ IT FROM DATA BASE (UNIT 8)
      READ(8,16,END=2,ERR=5) IVALUE
   16 FORMAT(20A4)
      RETURN
    2 ICOMPL=2
      RETURN
    3 ICOMPL=3
      RETURN
    4 ICOMPL=4
      RETURN
    5 ICOMPL=5
      RETURN
      END



      SUBROUTINE STORE(IADOR, IVALUE, LENGTH, ICOMPL)
C
C --  STORE PRIMITIVE --
C THIS PRIMITIVE STORES LENGTH STORAGE ELEMENTS (BYTES) STARTING AT
C VIRTUAL ADDRESS IVALUE INTO LENGTH STORAGE ELEMENTS STARTING AT
C VIRTUAL ADDRESS IADOR.  UPON COMPLETION, THE COMPLETION CODE ICOMPL
C IS SET TO:
C
C           1 IF NORMAL EXIT
C           3 IF LENGTH TOO BIG (>80 BYTES FOR THIS IMPLEMENTATION)
C           4 ILLEGAL VIRTUAL ADDRESS TO STORE INTO
C           5 ERROR WHEN ATTEMPTING TO DO PHYSICAL WRITE
C             (IMPOSSIBLE TO DETECT USING FORTRAN)
C
C ******* CERTIFIED 20 MAY 1969
C
      IMPLICIT INTEGER(A-Z)
C
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,
     1   FORM1,FORM2,FORM3,
     2   NEXTALL,SAMEALL,
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FANDLP,SANDLP,ATTACHP,DETACHP
      INTEGER IVALUE(LENGTH)
      IF ((LENGTH .GT. 80) .OR. (LENGTH .LT. 0)) GO TO 3
      ICOMPL=1
      IF (IADOR .NE. NEXTALL) GO TO 4
      IF (LENGTH .EQ. 0) RETURN
C NOW PHYSICALLY WRITE OUT RECORD TO DATA BASE (UNIT 8)
      WRITE(8,16) IVALUE
   16 FORMAT(20A4)
      RETURN
    3 ICOMPL=3
      RETURN
    4 ICOMPL=4
      RETURN
      END
```

1557810

Exhibit 2--FETCH and STORE Primitive Operations in the SHS System

The FETCH and STORE primitive operations actually perform the physical reads and writes which cause information transfer between the media the data base resides on and the primary storage medium (usually, magnetic core storage).

```
      SUBROUTINE CONTROL(INAME,IOPN,IYESNO,IOTHER)
C
C CONTROL IS CALLED TO DETERMINE WHETHER
C THE USER IS PERMITTED TO PERFORM OPERATION IOPN ON THE DATUM
C SPECIFIED BY INTERNAL NAME INAME.
C IYESNO IS SET TO 1 BY CONTROL IF THE OPERATION IS
C PERMITTED AND 2 OTHERWISE.  IN THIS IMPLEMENTATION,
C "OTHER INFORMATION" IS MEANINGLESS.
C
      IMPLICIT INTEGER(A-Z)
      DATA BLANK/'    '/
      DATA DLM1/';   '/
      COMMON/CURUCB/IUCB
      COMMON/ADDL1/IRAND,IRPT,PASSWD,USER,CARDA,PWTBL,IPWTBL,UTBL
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,
     1   FORM1,FORM2,FORM3,
     2   NEXTALL,SAMEALL,
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FANDLP,SANDLP,ATTACHP,DETACHP
      INTEGER PASSWD(10),USER(10),PWTBL(10,10),UTBL(10,10)
      INTEGER IRPT(4),CARDA(80),IRAND(20)
      INTEGER CARD(80)
      INTEGER IUCB(3)

C
C ------------------------------------------
C -------------- FORMULARY SELECTOR -------------
C ------ ------ ------ ------- --------- ----
C
      III=IUCB(3)
      GO TO (600,601,602), III
C
C -------- ------- --------- --------- -------
C ------------------------------------------------
C
C ******************************************************************
  602 CONTINUE
C CONTROL PROCEDURE FOR FORMULARY 3
C
C THIS PROCEDURE CURRENTLY ALLOWS ONLY FETCHES OF THE NEXT RECORD OR
C DETACHING OF FORMULARIES.  NO STORE OPERATIONS ARE PERMITTED.
C ******************************************************************
      IF ((IOPN .EQ. FETCHP).AND.(INAME .EQ. NEXTALL)) GO TO 20
      IF (IOPN .EQ. DETACHP) GO TO 20
C ALLOW DETACHMENT OF FORMULARY
    2 IYESNO=2
C OPERATION NOT ALLOWED
      RETURN
   20 IYESNO=1
C OPERATION IS ALLOWED
      RETURN

C
C ******************************************************************
  601 CONTINUE
C CONTROL PROCEDURE FOR FORMULARY 2
C
C THIS PROCEDURE CURRENTLY ALLOWS ONLY STORE OPERATIONS OF THE NEXT
C RECORD AND DETACHING OF FORMULARIES.  NO FETCH OPERATIONS ARE
C PERMITTED.
C ******************************************************************
      IF ((IOPN .EQ. STOREP) .AND. (INAME .EQ. NEXTALL)) GO TO 20
      IF (IOPN .EQ. DETACHP) GO TO 20
C ALLOW DETACHMENT OF FORMULARY
      GO TO 2
```

1557811

Exhibit 3--A CONTROL Procedure in the SHS System

The CONTROL procedure decides whether a user is allowed to perform the operation he requests on the particular datum he has specified. The subroutine illustrated here actually contains the CONTROL procedures for formularies 1, 2, and 3 in the SHS system. Formulary 3 allows only fetches of the next record in the data set or detaching of formularies; no STORE operations are permitted. Formulary 2 allows only STORE operations of the next record and detaching of formularies; no FETCH operations are permitted. Formulary 1 is the system formulary; it allows only detaching of formularies or attachment of formulary 1, 2, or 3. Before any attachment is made, a user identification and password check is carried out.

Exhibit 3--A CONTROL Procedure in the SHS System (cont'd.)

```
C
C *********************************************************************
C CONTROL PROCEDURE FOR SYSTEM FORMULARY (FORMULARY 1)
C
C THIS PROCEDURE CURRENTLY ALLOWS ONLY DETACHING OF A FORMULARY OR
C ATTACHMENT TO FORMULARY 1, FORMULARY 2, OR FORMULARY 3.
C *********************************************************************
  600 IF ((IOPN .NE. ATTACHP) .AND. (IOPN .NE. DETACHP)) GO TO 2
      IF ((INAME .NE. FORM1) .AND. (INAME .NE. FORM2) .AND.
     1   (INAME .NE. FORM3)) GO TO 2
C ONLY ALLOW ATTACH OPERATION ON THE DESIRED DATA (A FORMULARY)
      IF (IOPN .EQ. DETACHP) GO TO 20
C ALLOW DETACHMENT OF FORMULARY
      READ(5,18,END=2,ERR=2) CARD
C READ IN CARD WITH ACCESS CONTROL INFORMATION ON IT
   18 FORMAT(80A1)
C ... THIS CODE READS A CARD IMAGE AND
C CHECKS THE USER ID AND PASSWORD IT FINDS THERE AGAINST
C PRESTORED INFORMATION.  IF THE USER ID AND PASSWORD MATCH
C THOSE IN THE PROGRAM, CONTROL SETS IYESNO TO 1, SIGNIFYING
C THAT THE USER HAS PASSED A PRIVACY CHECK AND IS ALLOWED TO USE
C THE SYSTEM.  OTHERWISE,
C CONTROL SETS IYESNO TO 2, SIGNIFYING THAT HE HAS NOT.
C
      I=ISCAN(CARD,1,BLANK,80,0)
C HUNT FOR FIRST NON-BLANK
      IF (I .GT. 80) GO TO 990
C GO TO 990 IF SCAN RAN OFF END OF CARD
      LAST=I
      I=ISCAN(CARD,I,DLM1,80,1)
C HUNT FOR SEMICOLON (LEFT TO RIGHT-SCAN)
      IF (I .GT. 80) GO TO 990
C GO TO 990 IF SCAN RAN OFF END OF CARD
      CALL CLRTOHASH(CARD,LAST-1,I-LAST,USER)
C WE SCRAMBLE THE (CLEAR) USER ID BEFORE TESTING FOR A MATCH, SINCE
C THE MATCHING TEST IS MADE USING SCRAMBLED PRESTORED INFORMATION
C (NEEDHAM'S DEVICE)
      I=ISCAN(CARD,I+1,BLANK,80,0)
C HUNT FOR FIRST NON-BLANK
      IF (I .GT. 80) GO TO 990
C GO TO 990 IF SCAN RAN OFF END OF CARD
      LAST=I
      I=ISCAN(CARD,I,DLM1,80,1)
C HUNT FOR SEMICOLON (LEFT TO RIGHT-SCAN)
      IF (I .GT. 80) GO TO 990
C GO TO 990 IF SCAN RAN OFF END OF CARD
      CALL CLRTOHASH(CARD,LAST-1,I-LAST,PASSWD)
C WE SCRAMBLE THE (CLEAR) PASSWORD ID BEFORE TESTING FOR A MATCH, SINCE
C THE MATCHING TEST IS MADE USING SCRAMBLED PRESTORED INFORMATION
C (NEEDHAM'S DEVICE)
      DO 950 J=1,IPWTBL
      DO 951 I=1,10
      IF (PWTBL(I,J) .NE. PASSWD(I)) GO TO 950
  951 CONTINUE
C IF WE GET HERE, A MATCH ON PASSWORD HAS BEEN FOUND
C BUT NOT NECCESSARILY FOR THE CORRECT USER
      DO 952 I=1,10
      IF (UTBL(I,J) .NE. USER(I)) GO TO 950
  952 CONTINUE
C     MATCH EXISTS FOR USER/PASSWORD COMBINATION
      GO TO 20
C PERMIT FETCHING OF FORMULARY
  950 CONTINUE
      GO TO 2
  990 WRITE(6,793)
  793 FORMAT(' *** ACCESS CONTROL ERROR - SCAN RAN OFF CARD')
      GO TO 2
      END
```

1557811

Exhibit 3--A CONTROL Procedure in the SHS System (cont'd.)

```
      INTEGER FUNCTION ISCAN(BUF,N,DLM,MAX,K)
C ISCAN SCANS THE BUFFER BUF, WHICH CONTAINS ONE CHARACTER PER WORD,
C STARTING AT RELATIVE LOCATION N OF IT.  IT SCANS OVER TO THE NEXT
C CHARACTER = OR ¬= TO DLM, AND RETURNS AS ITS VALUE THE INDEX OF THE
C BUF BUFFER AT THAT PLACE.  THE SCAN IS TERMINATED AT RELATIVE
C LOCATION MAX OF BUFFER BUF IF NO MATCH (OR NON-MATCH) HAS BEEN
C FOUND UP TO OR INCLUDING THAT POINT; IN THIS CASE, AN INTEGER > MAX
C IS RETURNED.
C K = 1 IF THE SCAN SHOULD STOP WHEN A CHARACTER EQUAL TO DLM IS FOUND.
C K = 0 IF THE SCAN SHOULD STOP WHEN A CHARACTER UNEQUAL TO DLM IS
C FOUND.
      INTEGER BUF(80)
      INTEGER DLM
C
      I=N
  902 IF (I .GT. MAX) GO TO 901
      IF (((BUF(I) .EQ. DLM) .AND. (K .EQ. 1)) .OR.
     1    ((BUF(I) .NE. DLM) .AND. (K .EQ. 0))     )
     2          GO TO 901
      I=I+1
      GO TO 902
  901 ISCAN=I
      RETURN
      END
```

1465A15

```
      SUBROUTINE SCRAMBLE(CLRBUF,ICLRLEN,ICOMPL,SCRBUF,ISCRLEN)
C THIS SUBROUTINE SCRAMBLES THE UNSCRAMBLED DATUM WHICH
C IS ICLRLEN CHARACTERS LONG STARTING IN CLRBUF(1), AND IS
C STORED FOUR CHARACTERS PER WORD.  IT LEAVES THE
C SCRAMBLED DATUM IN THE FIRST ISCRLEN BYTES OF THE SCRBUF
C ARRAY (AND RETURNS ISCRLEN TO THE CALLING ROUTINE).
C THIS SUBROUTINE STORES A COMPLETION CODE IN ICOMPL.
C 0 < ISCRLEN < 81 AND 0 < ICLRLEN < 81.
C
C COMPLETION CODES STORED IN ICOMPL:
C      1     NORMAL EXIT
C      2     SCRAMBLE OPERATION NOT PERMITTED BY THIS FORMULARY
C      3     ILLEGAL LENGTH OF  DATUM TO SCRAMBLE
C
C ******* CERTIFIED 8 MAY 1969 *****
C
C
      COMMON/CURUCB/IUCB
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,
     1   FORM1,FORM2,FORM3,
     2   NEXTALL,SAMEALL,
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FANDLP,SANDLP,ATTACHP,DETACHP
      INTEGER SCRBUF(20),CLRBUF(20),IUCB(3)
      COMMON/ADDL1/IRAND,IRPT,PASSWD,USER,CARDA,PWTBL,IPWTBL,UTBL
      INTEGER PASSWD(10),USER(10),PWTBL(10,10), UTBL(10,10)
      INTEGER IRPT(4),CARDA(80),IRAND(20)
C ------------------------------------------
C -------------- FORMULARY SELECTOR -------------
C ------  ------  ------  -------  --------  ----
C
      III=IUCB(3)
      GO TO (3,1,3), III
C
C --------  -------  ---------  ---------  -------
C ---------------------------------------------------
    1 IF ((ICLRLEN .GT. 80) .OR. (ICLRLEN .LT. 1)) GO TO 4
      ISCRLEN=(ICLRLEN-1)/4+1
      DO 2 I=1,ISCRLEN
    2 SCRBUF(I)=LG01XR(CLRBUF(I),IRAND(I))
      ISCRLEN=ICLRLEN
      ICOMPL=1
      RETURN
    3 ICOMPL=2
      RETURN
    4 ICOMPL=3
      RETURN
      END                                                    1557A13
```

Exhibit 4--A SCRAMBLE Procedure in the SHS System

SCRAMBLE transforms raw data into encrypted form.

```
      SUBROUTINE UNSCRAMBLE(SCRBUF,ISCRLEN,ICOMPL,CLRBUF,ICLRLEN)
C THIS SUBROUTINE UNSCRAMBLES THE SCRAMBLED DATUM WHICH
C IS ISCRLEN CHARACTERS LONG STARTING IN SCRBUF(1), AND IS
C STORED FOUR CHARACTERS PER WORD.  IT LEAVES THE UNSCRAMBLED
C DATUM FOUR CHARACTERS PER WORD IN THE FIRST ICLRLEN
C BYTES OF THE CLRBUF ARRAY (AND RETURNS ICLRLEN TO THE
C CALLING ROUTINE).
C THIS SUBROUTINE STORES A COMPLETION CODE IN ICOMPL.
C 0 < ICLRLEN < 81 AND 0 < ISCRLEN < 81.
C
C COMPLETION CODES STORED IN ICOMPL:
C       1    NORMAL EXIT
C       2    UNSCRAMBLE OPERATION NOT PERMITTED BY THIS FORMULARY
C       3    ILLEGAL LENGTH OF DATUM TO UNSCRAMBLE
C
C
C ******* CERTIFIED 8 MAY 1969 *****
C
C
      COMMON/CURUCB/IUCB
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,
     1   FORM1,FORM2,FORM3,
     2   NEXTALL,SAMEALL,
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FANDLP,SANDLP,ATTACHP,DETACHP
      INTEGER SCRBUF(20),CLRBUF(20),IUCB(3)
      COMMON/ADDL1/IRAND,IRPT,PASSWD,USER,CARDA,PWTBL,IPWTBL,UTBL
      INTEGER PASSWD(10),USER(10),PWTBL(10,10), UTBL(10,10)
      INTEGER IRPT(4),CARDA(80),IRAND(20)
C -------------------------------------------
C -------------- FORMULARY SELECTOR -------------
C ------  ------  ------  -------  --------  ----
C
      III=IUCB(3)
      GO TO (1,1,2), III
C
C --------  -------  ---------  ---------  -------
C ---------------------------------------------
    1 ICOMPL=2
      RETURN
    2 IF ((ISCRLEN .GT. 80) .OR. (ISCRLEN .LT. 1)) GO TO 4
      ICLRLEN=(ISCRLEN-1)/4+1
      DO 3 I=1,ICLRLEN
    3 CLRBUF(I)=LG01XR(SCRBUF(I),IRAND(I))
      ICLRLEN=ISCRLEN
      ICOMPL=1
      RETURN
    4 ICOMPL=3
      RETURN                                                     1557A14
      END
```

Exhibit 5--An UNSCRAMBLE Procedure in the SHS System

UNSCRAMBLE tranforms encrypted data into raw form.

```
      SUBROUTINE VIRTUAL(INAME,IADDR,IOTHER,ICOMPL)
C
C
C
      COMMON/CURUCB/IUCB
      COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK,
     1   FORM1,FORM2,FORM3,
     2   NEXTALL,SAMEALL,
     3   FETCHP,STOREP,UNLFEP,UNLSTP,FANDLP,SANDLP,ATTACHP,DETACHP
      INTEGER IUCB(3)
C
C ********************** CERTIFIED 6 JUNE 1969
C -------------------------------------------
C -------------- FORMULARY SELECTOR -------------
C ------  ------  ------  -------  --------  ----
C
      III=IUCB(3)
      GO TO (1,1,1),III
C
C --------  -------  ---------  ---------  -------
C ---------------------------------------------
    1 IADDR=INAME
      ICOMPL=1
      RETURN
      END                                                    1557A15
```

Exhibit 6--A VIRTUAL Procedure in the SHS System

VIRTUAL transforms an internal name into the virtual address of the corresponding datum. In the SHS system, VIRTUAL is the identity transformation.

APPENDIX B

A COST EXPERIMENT

This appendix contains the source code and output relevant to the cost experiment described in Chapter V. The UNSCRAMBLE, VIRTUAL, and CONTROL procedures were essentially null and the ACCESS procedure of Exhibit 1, Appendix A was used.

```
OMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=58,SOURCE,EBCDIC,NOLIST,NODECK,LOAD,
          IMPLICIT INTEGER(A-Z)
          COMMON/COM1/NCARDS,ONE,TWO,ZERO,BLANK,IRAND,
         1 NEXTREC,FETCHP,STOREP,FLOCKP,SLOCKP,UNLFEP,UNLSTP,ATTACHP,DETACHP
          INTEGER IRAND(20)
          INTEGER CARD(20),SCARD(20),IUCB1(2),TIM1
          REAL TIME
    C NOTE PUN
          DATA NO/' NO '/
    C
          C1=2**8
          C2=2**16
          C3=2**24
          READ(5,910)NCARDS,ITRIES
      910 FORMAT(2I10)
    C FIRST, CREATE A TAPE WITH NCARDS 80-CHARACTER RECORDS
             REWIND 8
          DO 1 I=1,NCARDS
        1 WRITE(8) CARD
          NCARDS=NCARDS-1
          DO 200 NLOOP=1,ITRIES
          REWIND 8
             REWIND 9
    C
    C GET TIME JUST TO READ INPUT TAPE AND WRITE OUTPUT TAPE
    C    (NO FORMULARIES, NO SCRAMBLING)
          WRITE(9) CARD
    C OPEN DATA SET (USED TO SUBDUE JITTER IN TIMING TESTS)
          READ(8)CARD
    C NECESSARY TO INSURE REWIND IS DONE BEFORE INITIATING TIMING TEST
          TIM2=CLOCK1(4)
          DO 9 I=1,NCARDS
          READ(8)CARD
        9 WRITE(9)SCARD
          TIME=(CLOCK1(4)-TIM2)*26/1000000.0
          WRITE(6,901) NCARDS,ZERO,NO,TIME
          REWIND 8
          REWIND 9
    C
    C NEXT, SCRAMBLE THE TAPE USING ALGORITHM 1 AND NOT USING THE FORMULARY
    C METHOD.
          WRITE(9) CARD
    C OPEN DATA SET (USED TO SUBDUE JITTER IN TIMING TESTS)
          READ(8) CARD
    C NECESSARY TO INSURE REWIND IS DONE BEFORE INITIATING TIMING TEST
          TIM2=CLOCK1(4)
          DO 2 I=1,NCARDS
          READ(8)CARD
          DO 3 J=1,20
        3 SCARD(J)=LG01XR(CARD(J),IRAND(J))
        2 WRITE(9)SCARD
          TIME=(CLOCK1(4)-TIM2)*26/1000000.0
          WRITE(6,901) NCARDS,ONE,NO,TIME
      901 FORMAT(' TIME USED FOR ',I6,' CARDS WITH ALGORITHM ',I1,A4,
         1  ' FORMULARY METHOD WAS ',F9.5,' SECONDS.')
    C
    C NEXT, SCRAMBLE THE TAPE USING ALGORITHM 2 AND NOT USING THE FORMULARY
```

```
C METHOD.
      REWIND 8
      REWIND 9
      WRITE(9) CARD
C OPEN DATA SET (USED TO SUBDUE JITTER IN TIMING TESTS)
      READ(8) CARD
C NECESSARY TO INSURE REWIND IS DONE BEFORE INITIATING TIMING TEST
      TIM2=CLOCK1(4)
      CALL RAN2A(21474835)
      DO 4 I=1,NCARDS
      READ(8)CARD
      DO 5 J=1,20
      K1=MOD(RAN2(0),256)
      K2=MOD(RAN2(0),256)
      K3=MOD(RAN2(0),256)
      K4=MOD(RAN2(0),128)
C GET FOUR SMALL NON-NEGATIVE PSEUDO-RANDOM NUMBERS
      RAND=C3*K4+C2*K3+C1*K2+K1
C USE THEM TO MAKE ONE BIG PSEUDO-RANDOM NUMBER
    5 SCARD(J)=LGO1XR(CARD(J),RAND)
    4 WRITE(9)SCARD
      TIME=(CLOCK1(4)-TIM2)*26/1000000.0
      WRITE(6,901) NCARDS,TWO,NO,TIME
C
C NOW RUN TIMINGS USING THE FORMULARY METHOD
C
      DO 13 I=1,3
   13 CALL SCRTIM(I)
  200 CONTINUE
      RETURN
      END
```

```
COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=58,SOURCE,EBCDIC,NOLIST,NODECK,LOAD,M
             SUBROUTINE SCRTIM(FORMK)
      C         SCRAMBLE THE TAPE USING ALGORITHM K AND USING THE FORMULARY
      C METHOD. PRINT OUT THE TIME THIS TAKES.
             IMPLICIT INTEGER(A-Z)
            COMMON/COM1/NCARDS,ONE,TWO,ZERO,BLANK,IRAND,
           1 NEXTREC,FETCHP,STOREP,FLOCKP,SLOCKP,UNLFEP,UNLSTP,ATTACHP,DETACHP
            INTEGER CARD(20),SCARD(20),IUCB1(2),TIM1
            INTEGER IRAND(20)
            INTEGER FORMK
            REAL TIME
            REWIND 8
            REWIND 9
            WRITE(9) CARD
      C OPEN DATA SET (USED TO SUBDUE JITTER IN TIMING TESTS)
            READ(8) CARD
      C NECESSARY TO INSURE REWIND IS DONE BEFORE INITIATING TIMING TEST
            TIM2=CLOCK1(4)
            CALL ACCESS(IUCB1,FORMK,CARD,80,ATTACHP,COMPCODE)
      C ATTACH TO APPROPRIATE FORMULARY FOR SCRAMBLING ALGORITHM K
            DO 6 I=1,NCARDS
            READ(8)CARD
          6 CALL ACCESS(IUCB1,NEXTREC,CARD,80,STOREP,COMPCODE)
      C STORE DATA (SCRAMBLED)INTO DATA BASE(I.E., ONTO THE TAPE)
            TIME=(CLOCK1(4)-TIM2)*26/1000000.0
            FORMKM=FORMK-1
            WRITE(6,901) NCARDS,FORMKM,BLANK,TIME
        901 FORMAT(' TIME USED FOR ',I6,' CARDS WITH ALGORITHM ',I1,A4,
           1  ' FORMULARY METHOD WAS ',F9.5,' SECONDS.')
            RETURN
            END
```

```
COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=58,SOURCE,EBCDIC,NOLIST,NODECK,LOAD,
            SUBROUTINE SCRAMBLE(CLRBUF,ICLRLEN,ICOMPL,SCRBUF,ISCRLEN)
            IMPLICIT INTEGER(A-Z)
            COMMON/CURUCB/IUCB
            COMMON/COM1/NCARDS,ONE,TWO,ZERO,BLANK,IRAND,
           1 NEXTREC,FETCHP,STOREP,FLOCKP,SLOCKP,UNLFEP,UNLSTP,ATTACHP,DETACHP
            INTEGER SCRBUF(20),CLRBUF(20),IUCB(3),IRAND(20)
      C ------------------------------------------------------------------
      C ----------                   FORMULARY SELECTOR           ----------
            III=IUCB(3)
            GO TO (1,2,3), III
      C ------------------------------------------------------------------
          1 ICOMPL=1
            RETURN
          2 DO 5 J=1,20
          5 SCRBUF(J)=LG01XR(CLRBUF(J),IRAND(J))
            ICOMPL=1
            RETURN
          3 C1=2**8
            C2=2**16
            C3=2**24
            DO 6 J=1,20
            K1=MOD(RAN2(0),256)
            K2=MOD(RAN2(0),256)
            K3=MOD(RAN2(0),256)
            K4=MOD(RAN2(0),128)
      C GET FOUR SMALL NON-NEGATIVE PSEUDO-RANDOM NUMBERS
            RAND=C3*K4+C2*K3+C1*K2+K1
          6 SCRBUF(J)=LG01XR(CLRBUF(J),RAND)
            ICOMPL=1
            RETURN
            END
```

```
COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=58,SOURCE,EBCDIC,NOLIST,NODECK,LOAD,M/
            BLOCK DATA                                                             345.
            IMPLICIT INTEGER(A-Z)                                                  346.
            COMMON/COM1/NCARDS,ONE,TWO,ZERO,BLANK,IRAND,
           1 NEXTREC,FETCHP,STOREP,FLOCKP,SLOCKP,UNLFEP,UNLSTP,ATTACHP,DETACHP
            COMMON/CONSTANTS/NUCB,NFORM,MAXUSERS,MAXLLIST,ITALK
            COMMON/OWN1/UCB1,LLIST,CS1,ISTDUCB
            INTEGER IRAND(20)
            INTEGER UCB1(100,3)
            DATA BLANK/'    '/,ONE/1/,TWO/2/,ZERO/0/
            DATA FETCHP/1/,STOREP/2/,UNLFEP/3/,UNLSTP/4/,                          409.
           1  FLOCKP/5/,SLOCKP/6/,ATTACHP/7/,DETACHP/8/                            410.
            DATA NEXTREC/1000/
            DATA IRAND/-143295037, 12498331, -99905473, 107015948,                 359.
           1           -85881432, 13737389, -254817906, 227051690,                 360.
           2           267059188,-305496183,132598180,-133310762,
           3           -124696699,-143295037,243176905,-240111797,
           4           199832006,-178963561, -219961227,-174003653/
            DATA NFORM/3/                                                          404.
        C NUMBER OF FORMULARIES CURRENTLY IN THE SYSTEM                            405.
            DATA MAXUSERS/100/                                                     397.
        C MAXUSERS = MAX. NO. OF USER/TERMINAL COMBINATIONS                        398.
        C POSSIBLE AT ANY GIVEN TIME                                               399.
            INTEGER LLIST(4,100)/400*-1/                                           366.
        C THE LOCKLIST
            INTEGER ISTDUCB(3)/0,0,1/                                              364.
        C STANDARD USER CONTROL BLOCK (TEMPLATE)                                   365.
            DATA NUCB/3/                                                           395.
        C NUCB = NO. OF WORDS IN EACH USER CONTROL BLOCK                           396.
            DATA CS1/1/
        C INITIALIZE TO CRITICAL SECTION OF ACCESS PROC. NOT CURRENTLY IN USE
            DATA UCB1(1,1)/-2/                                                     393.
        C INITIALIZE TO NO ACTIVE USER CONTROL BLOCKS                              394.
            DATA MAXLLIST/5/
        C MAXIMUM LENGTH OF LIST OF LOCKED DATUMS MAINTAINED BY ACCESS PGM         392.
            DATA ITALK/2/                                                          389.
        C LENGTH OF ARRAY PASSED BY TALK PROGRAM TO ACCESS PROGRAM                 390.
            END                                                                    411.
```

```
1E USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS  19.38559 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS  19.38559 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS  21.01631 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS  19.41887 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS  19.43552 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS  19.43552 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS  19.88480 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS  19.45215 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS  19.41887 SECONDS.
1E USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS  19.43552 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS  19.90143 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS  19.68512 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS  19.43552 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS  19.86815 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS  19.43552 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS  19.43552 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS  19.43552 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS  19.40224 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS  19.43552 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS  19.88480 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS  20.58368 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS  19.40224 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS  19.88480 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS ********* SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS  19.40224 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS  19.40224 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS  19.41887 SECONDS.
 E USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS  20.11775 SECONDS.

218I FIOCS - I/O ERROR  LJHCOSTS,GO       ,OC1,TA,FT09F001,WRITE ,DATA CHECK     ,000

        X YO#    F Y         ,L   Q ;     K N X    O   NF7V   =N 4&5.   Q )  5  , 0

   O   NF7V   =N 4&5.   Q )  5  , 0          X YO#    F Y         ,L   Q ;     K N X

#    F Y         ,L   Q ;     K N X    O   NF7V   =N 4&5.   Q )  5  , 0          X YO

  =N 4&5.   Q )  5  , C          X YO#    F Y         ,L   Q ;     K N X    O   NF7V

    ,L   Q ;     K N X    O   NF7V   =N 4&5.   Q )  5  , 0          X YO#    F Y

Q )  5  , 0          X YO#    F Y         ,L   Q ;     K N X    O   NF7V   =N 4&5.   Q

;     K N X    O   NF7V   =N 4&5.   Q )  5  , 0          X YO#    F Y         ,L   Q ;

        X YO#    F Y         ,L   Q ;     K N X    O   NF7V   =N 4&5.   Q )  5  , 0

      NF7V   =N 4&5.   Q )  5  , 0          X YO#    F Y         ,L   Q ;     K N X

#    F Y         ,L   Q ;     K N X    O   NF7V   =N 4&5.   Q )  5  , 0          X YO
```

```
7V    =N 4&5.    Q )   5   , C          X YO#     F Y           ,L    Q ;      K N X     O   NF7V
        ,L    Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#     F Y
   Q )   5   , 0          X YO#     F Y           ,L    Q ;      K N X     O   NF7V    =N 4&5.   (
  Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#     F Y           ,L    Q ;
  O          X YO#     F Y           ,L    Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0
  X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#     F Y           ,L    Q ;      K N X
  YO#     F Y           ,L    Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#
7V    =N 4&5.    Q )   5   , C          X YO#     F Y           ,L    Q ;      K N X     O   NF7V
        ,L    Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#     F Y
   Q )   5   , 0          X YO#     F Y           ,L    Q ;      K N X     O   NF7V    =N 4&5.   0
  Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#     F Y           ,L    Q ;
  O          X YO#     F Y           ,L    Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0
  X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#     F Y           ,L    Q ;      K N X
  YO#     F Y           ,L    Q ;      K N X     O    NF7V    =N 4&5.    Q )   5   , 0          X YO#
7V    =N 4&5.    Q )   5   , 0  &          0 & 0 0         -                                 &J   /
TRACEBACK FOLLOWS-   ROUTINE    ISN   REG.  14    REG.  15    REG.   0    REG.   1
                     IBCOM            A218AD00    0018C430    64800A0C    0018A9D4
                        MAIN          401677CC    0018A808    FF000018    001817F8

ENTRY POINT=  0018A808

STANDARD FIXUP TAKEN , EXECUTION CONTINUING
TIME USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS   28.60416 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS   20.28415 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS   19.86815 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS   19.86815 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS   19.55199 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS   20.55040 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS   19.41887 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS   20.78336 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS   19.86815 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS   20.56703 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS   19.41887 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS   19.41887 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 0 NO  FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 1 NO  FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2 NO  FORMULARY METHOD WAS   19.40224 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 0     FORMULARY METHOD WAS   19.86815 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 1     FORMULARY METHOD WAS   19.86815 SECONDS.
TIME USED FOR  10000 CARDS WITH ALGORITHM 2     FORMULARY METHOD WAS   19.40224 SECONDS.
```

APPENDIX C

THE ACCESS PROCEDURE — "NO PARALLELISM" VERSION

This appendix presents a version of the ACCESS algorithm which can be used when no user will ever have to lock out access to a datum which ordinarily can be accessed by several users at the same time or if the installation wishes to use a method other than the one given in Section K of Chapter III to control conflicts among users competing for exclusive access to datums.

```
procedure access (info, intname, val, length, opn, compcode);
integer array info, val; integer intname, length, opn, compcode;
begin comment  If OPN = FETCH, VAL is set to the value of the datum
               represented by INTNAME.
               If OPN = STORE, the value of the datum represented by INTNAME
               is replaced by the value in the VAL array.
               If OPN = ATTACH, the formulary represented by internal name
               INTNAME is attached to the user and terminal described
               in the INFO array.
               In OPN = DETACH, the formulary represented by internal name
               INTNAME is detached from the user and terminal described
               in the INFO array.
               VAL is LENGTH storage elements long.
               Note that a FETCH (STORE) operation will actually attempt
               to fetch (store) LENGTH storage elements of information.
               It is the responsibility of the TALK procedure to handle
               scrambling or unscrambling algorithms that return outputs
               of a different length than their inputs.
               ACCESS returns the following integer completion codes in
               COMPCODE:
               1  normal exit, no error
               3  operation permitted by CONTROL procedure gave error
                  when attempted
               5  cannot handle any more User Control Blocks (would cause
                  table overflow)
               6  attempt to detach nonexistent user/terminal/formulary
                  combination
```

10 error return from VIRTUAL procedure

11 operation on the datum represented by INTNAME not permitted by CONTROL procedure of the attached formulary

12 end of data set encountered by FETCH operation.

Note that by the time the user has left the ACCESS routine, the data may have been changed by another user. Note that ACCESS could be altered to allow scrambling and unscrambling to take place at external devices rather than in the central processor.

Important: ACCESS expects the following to be available to it. The installation supplies these in some way other than parameters to ACCESS (for example, as global variables in ALGOL or COMMON variables in FORTRAN) —

(1)	ISTDUCB	the default User Control Block. Its length is NUCB storage units.
(2)	NUCB	see (1).
(3)	UCB	a list of User Control Blocks (UCBs) initialized outside ACCESS to ucb(1, 1) = -2, ucb(i, j) = anything when ~(i=j=1) UCB is declared as <u>integer</u> <u>array</u> (1: maxusers, 1: nucb).
(4)	MAXUSERS	the maximum number of users which can be actively connected to the system at any point in time.
(5)	ITALK	the length of the INFO array (which is the first parameter of ACCESS) — INFO contains information about the user and terminal which is used by ACCESS and also passed by ACCESS to procedures of the attached formulary. INFO(1) contains user identification.

ACCESS assumes that the variables FETCH, STORE, FETCHLOCK, STORELOCK,

```
UNLOCKFETCH, UNLOCKSTORE, ATTACH, and DETACH have been initialized
globally and are never changed by the installation;
integer array iucb (1:nucb), reslt (1:length);
integer i, ii, islot, j, yesno, other, n, datum;

procedure ret (i); integer i;
begin comment RET sets the completion code compcode to i and then causes
exit from the ACCESS procedure;
compcode := i; go to FIN
end ret;

compcode := 1;
comment first let's see if we recognize the user/terminal combination
in INFO;
islot := 0;
for i := 1 step 1 until maxusers do
     begin ii := i;
     if ucb [i, 1] = -2 then begin comment end of list of ucb's;
                              if islot = 0 then begin if ii ≠ maxusers then
                              ucb [ii + 1, 1] := -2; go to XFER
                                                 end
                                          else go to PRESETUP;
                              end
     else if ucb [i, 1] = -1 then islot := ii
          comment remember this islot if vacant;
     else begin for j := 1 step 1 until italk do
```

```
                if ucb [i, j] ≠ info [j] then go to ILOOPND;
            go to SETUPPTRS
        end;
ILOOPND:
    end i loop;
if islot = 0 then ret (5); comment cannot handle any more UCBs;
PRESETUP:
ii := islot;
XFER:
for k := 1 step 1 until italk do ucb [ii, k] := info[k];
for k :== italk + 1 step 1 until nucb do ucb[ii, k] := istducb[k];
SETUPPTRS:
for i := 1 step 1 until nucb do iucb[i] := ucb[ii, i];
comment set up pointers to appropriate user control block for particular
implementation. Note well: Setting up pointers to appropriate user control
blocks is quite dependent on the particular system. For an example of one
implementation, see Exhibit 1 of Appendix A;
comment We have now associated user and terminal with user control block
(representing formulary) in relative position ii of the ucb table;
if iucb[nucb] ≠ intname and opn = DETACH then ret (6);
comment attempt to detach user/terminal/formulary combination not currently
attached;
control (intname, opn, yesno, other);
if yesno > 1 then ret (11);
comment return 11 if CONTROL does not permit operation;
if opn = ATTACH then begin ucb[ii, nucb] := intname; go to FIN
                        end;
```

```
comment Note well: In many implementations, pointers to each procedure of the
formulary (obtained by having VIRTUAL transform intname into a virtual address)
might be put into the UCB upon attachment. In others, the philosophy used here
of only putting one pointer — to the formulary — into the UCB will be followed.
The decision should take into account design parameters such as implementation
language, storage available, etc.;
if opn = DETACH then begin comment detach formulary (this leaves an open
                          slot in the ucb array); ucb(ii, 1) :=-1; go to FIN
                          end;
virtual (intname, datum, other, compcode);
comment VIRTUAL returns in datum the virtual address of the datum specified;
if compcode >1 then ret (10); comment error return from VIRTUAL;
if opn = STORE then
   begin comment store operation;
   scramble (val, length, compcode, reslt, n);
   if compcode >1 then ret (3);
   comment operation permitted but gave error when attempted;
   comment now perform a physical write of n storage units to the block
   starting at reslt;
   store (datum, reslt, n, compcode);
   if compcode >1 then ret (3)
   end
else
   begin comment fetch operation;
   fetch (datum, reslt, length, compcode);
   if compcode = 2 then ret (12); comment end of data set encountered;
   if compcode >1 then ret (3);
```

```
        unscramble (reslt, length,  compcode,  val,  n);
        if compcode > 1 then ret (3);
        end fetch operation;
    FIN:
    end access;
```