The problem of sharing information while protecting proprietary data in large computer files is reviewed.

The author suggests certain guidelines for data protection in general-purpose, time-sharing systems, and develops a model of a secured shared file. Operation of the system based on these guidelines is discussed.

The authorization problem in shared files

by T. D. Friedman

Time-sharing computer systems permit large numbers of users to operate on common sets of data and programs. Since certain of this material may be sensitive or proprietary, protective measures are required to prevent disclosures to persons lacking proper authority. In current time-sharing systems, protection is often inadequate. A central concern in protection is the authorization problem[1-3] which involves determining whether a user who is recognized by the system should be allowed access to information he desires. An authorization system must include a mechanism for withholding or releasing the information after the determination is made. This mechanism is closely associated with access mechanisms and operating systems.

This paper considers authorization, as far as possible, apart from specific access mechanisms or operating systems, and suggests directions for study and research. Observations are made about information protection in general-purpose, time-sharing systems. Authorization is then considered in detail, and guidelines for dealing with this problem are discussed. A hypothetical model of an authorization system is described.

information protection

Authorization is part of the more general problem of information protection in computers,[4,5] which consists of both protecting data and programs from destruction and securing them against improper

disclosure. In providing comprehensive protection, the following four objectives should be considered:

- *System integrity.* Hardware, operating system, channels, and switching centers must operate reliably and resist penetration. Areas of particular vulnerability are storage protection, privileged instruction control, page roll-in/roll-out, management of dumps, system failure and recovery procedures, and the handling of removable files. Integrity also depends on the physical isolation of the machine room, communication lines, and remote devices. Personnel selection and computer center policies require attention as well.
- *User identification.* Users at remote terminals must be correctly identified when they log on, and their identities must continue to be assured throughout the execution of their jobs. Various identification mechanisms have been proposed, including secret passwords, locks and keys, badge readers, and automatic voice recognition. If user identities are not adequately verified, all other protection measures are rendered ineffectual because an intruder may penetrate the system by masquerading as a known user.
- *Audit trails.* Accesses to protected data should on occasion—or perhaps at all times—be recorded. In this way, infiltration attempts may be detected, and the threat of discovery should discourage mischief.
- *Authorization.* Methods to share data selectively among users is the subject of this paper.

The problem

It would be desirable to provide selective sharing of information in main storage. However, the required authorization may hamper performance or increase costs since elaborate checking and disabling processes may have to be introduced into every storage access. Current time sharing systems do in fact achieve some degree of protection in main storage, but sharing is permitted only at the level of large storage segments rather than individual fields.[6]

This paper addresses the more tractable problem of providing authorization for accesses to secondary, or auxiliary, storage media. Small delays in auxiliary storage accesses may be tolerable because of the slower operation of these media as well as their lower frequency of access. In this discussion, authorization refers only to the protection of access to shared auxiliary files. The separate objectives of system integrity, user identification, and audit trails are not treated in detail, nor are the problems of operating system design, access methods, data management, and field handling. These topics, however, affect authorization critically and require attention.

Authorization involves the comparison of restrictions placed on information—data and programs—in files with privileges of users requesting access to this information. In the past, this requirement did not arise because files were not shared; users exchanged information only off line. Proprietary files were protected by being stored on removable tapes or disks, which could be held under lock and key when not in use. During runs, the entire computing center could be made secure.

The true authorization problem arises when several users must access a common file, subject to the constraint that not all users are permitted access to all data. Such a requirement may arise in police networks, government data centers, banking data centers, management information systems, or credit bureaus. Conceivably, some future computer center may simultaneously serve several of these applications.

efficiency

Authorization is not so much a theoretical problem as one of efficiency.[7] The problem increases when there are large numbers of data sets, protection categories, and users. Authorization may then be considered as a mapping function of users to data. But in some future computer systems, such a mapping function might well become unmanageable. Imagine, for example, a system supporting 20,000 users, 2,000 of whom may be on line simultaneously. They may be processing data from a common bank containing perhaps trillions of records. It would seem to be a formidable task to screen each file access according to all restrictions that may have been placed on the data versus all privileges that the user may hold. Possibly the process could be shortened by consolidating the diverse privileges and restrictions or assigning them to a hierarchy. Nevertheless such a scheme creates interdependencies of categories that make later modifications cumbersome. A file that cannot be easily modified is likely to suffer early obsolescence.

specialized applications

There may be little need to make major modifications, however, in files dedicated to various specialized applications. Many management information systems are of this type in which all data is maintained by a central administrator. Formats and data organization in these systems are well-defined and relatively fixed, and the classes of users are also restricted and easily described. All types of protection and privileges can be determined in advance. The authorization process may be organized in a straightforward manner because only a small fixed number of protection categories are needed for information in the file, and a fixed number of user classes with their associated privileges are required.

A variety of techniques could serve to simplify authorization in such systems. Because the format of data fields is fixed, the individual fields need not be tagged. Protection could be provided by reference to a single format descriptor. Sets of protections and privileges

could be combined, and specific combinations could be represented in terms of such logical connectives as AND, OR, and NOT. Greater convenience and efficiency could thereby be provided than is possible with exhaustive lists of protections or privileges. Protection may also be defined in terms of the value of certain fields, such as "Salaries over $10,000," which could prove quite useful. Systems of this type are gaining wide usage, and substantial activity is currently devoted to providing authorization for these applications.

general systems

This paper, however, is concerned with large systems of a less centralized type intended to serve a variety of applications, such as the MIT Compatible Time-Sharing System (CTSS)[8] and the IBM System/360 Time-Sharing System (TSS/360).[9] A large number of dissimilar data sets may be maintained within storage, and the formats and organization of these data may not be known in advance. Indeed, new data sets in entirely unpredictable formats may be introduced by users at any time. In such systems, a more flexible authorization mechanism is required, and the specialized techniques previously discussed may, in these general systems, cause difficulties and inefficiencies. For example, protection according to value would require the authorization system to retrieve information about the formatting of records involved, which could cause large delays. Instead, the techniques considered here are intended for systems in which format, data organization, user categories, and applications are not fixed in advance.

In a general shared file system, complex and unsystematic relationships may develop between user privileges and data restrictions. In management applications, for example, managers may possess the right to see their employees' pay records, but not, say, their health records. Corporate officers should have access to information within their provinces of authority. Since executive responsibilities may in part be duplicated, the privileges may be neither disjoint nor fall into a simple tree hierarchy, but instead may be overlapping. Moreover, privileges may change more or less continuously as users acquire or lose responsibilities in their assignments.

enabling and disabling access

Thus, it may be shortsighted to base an authorization system on anticipated user relationships. Authorization simply requires that users be separated from information they lack privilege to access. A test for protection violations must be performed; if a user does not hold privilege to records he requests, access must be prevented.

To consider how access may be disabled, let us distinguish the following four functional steps in a read operation. (Analogous steps exist in write or update operations.)

1. User establishes connection to the system and is logged on.
2. User requests information.
3. System responds by selecting the information.

4. Information is transmitted to the user.

Methods for disabling (or enabling) access can thus be classified according to the step that is affected.

The highest degree of protection is provided by disabling the first step; i.e., a user is prevented from logging on altogether if he lacks privilege to any data on the system. This level of protection implies that sets of data having distinct protection requirements must be maintained on separate machines for different users. Such a scheme eliminates a major part of the protection problem, but it also eliminates the capability for time sharing. The result is unattractive not only because the processor and file system must be duplicated for each user group, but also because common updating operations are prohibited. Even so, this scheme may be the only acceptable approach to authorization where military security standards apply.

Alternatively, the user may be allowed to log on, but he is prevented from issuing unauthorized requests. This can be accomplished by restricting knowledge of the data names to persons privileged to those data. By this scheme, if someone knows what to ask for, it implies he has a right to see it.[10] This is a simple and easily realized approach. However, anyone who, through accident or design, acquires a secret name also acquires the ability to violate the protection. A refinement to this approach is not to conceal data names but to require the requester to supply an additional secret password for protected sets of data. The advantage of this scheme is that passwords can be made longer and more difficult to discover than would be convenient for data names. However, it burdens the user with remembering and communicating more terms. Moreover, the scheme still remains vulnerable to penetration by accidental discovery of the secret terms.

In a third approach, the user is allowed to log on and to request access. If he does not hold appropriate privilege, however, the selection mechanism is inhibited. The data protection techniques used in CTSS and TSS/360 fall into this category. Sensitive files outside the user's storage area can be selected only if he possesses special internal pointers in his directory to those records. The distinction between the preceding method and the method suggested here is that in this case the "passwords" (i.e., pointers) are supplied and maintained by the system rather than by the user. Because the user cannot change his own pointers, this system is more resistant to mischief than the previous one. Nevertheless, even here the test for violation is conducted on information specified in the request rather than on the information that is actually selected. If a machine error occurs after the test and during selection, then sensitive data may be made available to unauthorized persons. This protection method may thus fail in the event of a single error such as the alteration of an address.

Stronger protection is provided if the data transmission step rather than the selection step is disabled on detection of a violation. Then the test for violations may be conducted using the data that are actually selected from storage, but before they are made available to the user. In this case, protection failures would require at least two independent but coincidental mishaps: (1) unauthorized data are accessed in storage, and (2) these data are not recognized as unauthorized for the requester during the subsequent testing phase.

Rather than disabling the actual transmission of data, they could always be made available. If a violation occurs, however, the data would be presented in an unusable form. For example, sensitive records could be stored in crytographically enciphered form, and be automatically deciphered during output using a key assigned to the user. Cryptographic techniques provide impressive levels of protection, [2,11–13] although the security of any cipher is never certain. Cryptography seems especially promising for protecting communication lines to remote terminals. Nevertheless, this approach may prove unacceptable for routine file processing because of the resulting delay in channel access.

In contrast to full-scale cryptography, an elementary scrambling process[14] has attracted interest because of its economy and speed. Scrambling consists of the replacement of characters in a record according to a simple, fixed substitution rule. As scrambled records are read from a file, they are simultaneously unscrambled according to a user's key. Improperly retrieved records are scrambled according to an unknown key and so are unreadable. Scrambling thus prevents one user from directly browsing through another user's data. However, it is a relatively trivial (as well as an entertaining) challenge to decipher such a code. Since scrambling does not offer significant protection, it may actually constitute a danger by providing an illusion of protection that does not exist.

Although these six methods do not exhaust the means of disabling access, they illustrate the wide choice of techniques possible. A combination of techniques may be stronger than any one alone, but this would be more costly and is not considered further here. Of the alternatives, protection by disabling the data transmission appears most promising because failure would require the concurrence of two independent mishaps.

test for violations

Consider now a test to detect protection violations. The test determines whether the requested access falls within the allowed privileges of the user. A straightforward approach is to scan a list of codes delegated to the user until a code is found that authorizes the requested access. More sophisticated approaches could involve the derivation of protection and privilege values indirectly by some algorithm.[6] Protection codes may be stored with the data, or alternatively, they may be held in separate attribute tables. Protection

might even be defined as a function of the value of the data. According to one view, individual entries need no protection. Rather, the function of a protection system is to ensure that a user does not obtain certain combinations of entries that would enable him to discover sensitive facts. This last threat may be averted by requiring a user to hold privilege for all data in a category before he can see any of these data.

It is not possible to settle such application- and system-related issues in this general discussion. However, considerations such as simplicity, efficiency, and flexibility appear to recommend the less sophisticated method, namely, the systematic comparison of protection codes with privilege codes.

privilege

Privilege may be described as the right of users to access sensitive information. However, it is often advantageous for a program to possess privileges different from those of the person who invoked it. The program's privileges may exceed those of the person. For example, a statistician may be prohibited from seeing individual salary records, but he could be permitted to execute a program that has the privilege of reading salaries in order to determine salary averages. Likewise, it would be desirable to deny a program certain access privileges held by the person using the program. In that way, a user could ensure that a program does not disturb certain of his data sets.

In the case of a program that possesses privileges not held by the person using it, such a program must have a kind of miniature security system built in to censor sensitive information from data returned to the user. Consequently, the user must be prevented from modifying the program, and perhaps prevented even from looking at it.

Furthermore, it would be useful if privileges and restrictions could be passed along selectively as programs invoke other programs. But then rather elaborate controls would be needed to keep track of the levels of privilege and restriction that apply at any moment. Before any access can be obtained, the authorization system would have to determine the result of all accumulated privileges and restrictions. Such capabilities would burden the authorization system with chores involving scoping rules and stacking. Because all file accesses would be affected, system performance might deteriorate.

The notion of privileged programs warrants further attention. Although they are not considered further here, privileged programs are discussed in Reference 10. In the remainder of this paper, privilege is referred to as an attribute of users, but of course, observations made regarding user privilege could also apply to program privilege.[15]

Planning for authorization should begin during the preliminary development of a system. It may not be sufficient to "patch up" an existing system by the addition of checking and monitoring features. Checking and monitoring, even when used extensively, do not necessarily make safe a vulnerable system. For example, if there are many access paths to certain sensitive data, a checking process could be introduced into each path. Nevertheless, unforeseen combinations of paths may provide "trap door" entrances, allowing resourceful human infiltrators to circumvent the protection.[2] Instead, it seems preferable to redesign such a system so that only one fully protected access path exists to the sensitive data.

Although it is clear that protection is mandatory for shared-file systems, it is not evident how extensive it should be. Commercial users, for example, do tolerate occasional disclosures under batch processing conditions. Occasionally, a user discovers that he has received someone else's output or that someone else has received his. On the other hand, extreme military levels of security[16] are more than appropriate for most non-military applications.[17] Indeed, if a reasonably secure system could be realized, it might in time be accepted for certain military applications.

But what is reasonable? Application studies and user surveys could help to indicate the degrees of protection that customers will demand, the costs they will pay, and the penalties in processing speed and convenience they will tolerate. In the absence of this information, the discussion must be speculative.

An ideal system

At this point, let us consider features that would characterize an ideal authorization mechanism for a general-purpose, time-sharing system.

- Users should be assured that protected data, programs, or messages will not be disclosed to unauthorized parties, even in case of major hardware failure or loss of the operating system. On the other hand, users should never be denied access to information for which they are authorized.
- It should not be possible for any user to "break" the protection mechansim so as to discover secret data, even if he understands how the mechanism operates.
- Users should be able to enter data freely into a protected file, and they should be able to specify the individuals who are allowed access to those data, and the type of access permitted.
- All common types of file updating and processing should be permitted. Users should be able to create, modify, and delete data within their areas of responsibility.
- Response time for processes in the secure shared file should

not be appreciably greater than the time required for any other processing.

- The authorization system should impose as few restrictions as possible on the operating system, file structure, time-sharing system, and hardware.
- It would be undesirable for users who have already been logged on and identified to be expected to remember long lists of passwords, secret keys, or special commands. People are inclined to write lists down, thereby compromising secrecy. Forgetful users may at length attempt to disable the protection mechanisms altogether in order to continue using the machine.
- The system should not depend upon continuous attention of a human security officer for its normal operation, since a human is likely to be overwhelmed during periods of high activity. However, a human authority should be notified in the event of irregularities, and he should be able to suspend any job on command.

Although we have indicated that a protection mechanism should, where possible, be independent of the hardware, operating system, and environment, a practical protection mechanism must reflect characteristics of the entire system including the file organization, channel switching, and record identification methods. Whereas some general recommendations may be advanced on the basis of the ideal system, those recommendations are suggested only if they can be provided compatibly with the specific system architecture. Possible difficulties resulting from these recommendations are pointed out later in this paper.

To discuss authorization, a basic unit of protected information is necessary. Accordingly, we define a *protected field* as a section of data or program in storage that is subjected to a uniform degree of protection, i.e., all bits of the field receive exactly the same protection. This unit should be distinguished from a *physical record*, which is defined as a separately retrievable unit of information from a given storage device. Protected fields in certain cases should also be distinguished from a *logical record*, which is considered a unit in terms of its content, function, or use.

This concept implies the existence of field handling within the data management system. However, it should be noted that field handling, which consists of identifying and extracting fields from larger physical or logical records, involves formidable design problems that may exceed the difficulties of the authorization system itself.

System guidelines

On the basis of the preceding considerations, the following guidelines are suggested by the author:

- Isolation of the authorization mechanism
- Access limitation
- Adjacent tagging
- Single-tag rule
- Compartmentalization

isolation of the authorization system

Although functions of authorization and the operating system often overlap, the authorization system should be organized as an isolated program module distinct from the remainder of the operating system. Protection would then be distinguished from other data management functions. The authorization system would be invoked as an independent task whenever access to data in the secured shared file is requested. This would allow the authorization system to be programmed as a limited, self-contained package so that it could be subjected to unusually thorough debugging and check-out.

The package should reside in a separate protected region of storage in order to eliminate "trap door" entrances into the routines. Attempts to branch into those programs, even when made by the operating system, would then be rejected automatically as violations of the storage bounds. Instead, the routines should be invocable only in response to a limited set of explicit requests.

The designer, however, should not disregard possible difficulties that isolation may impose. Separate "packaging" might increase the program size, and delays could also occur. Housekeeping and other utilities might have to be duplicated in such a package. Extra programming may also be required for routines to be made invocable functions, i.e., prologues, epilogues, or argument-passing mechanisms might be needed. Interruption handling and failure-recovery routines may present particular problems for such a package.

access limitation

The shared file itself must be isolated so that it will be impossible to access it except by means of the authorization system. One way to meet this requirement is to dedicate certain channels to the authorization system. That is, channels would be assigned permanently to the authorization programs. No way should be provided to reassign these channels by program control. It is possible, however, that a satisfactory level of security can be obtained as well by programmed access methods, without dedicated channels.

adjacent tagging

Some form of tagging is needed to designate data as "protected." The tags ought to be kept adjacent to the data themselves, provided this is consistent with the file organization. It is often convenient to carry data attributes in lists separated from the data. However, every interval separating a protected field from its protection presents a slight but real opportunity for errors of reference to arise. Since the goal is to minimize risk, physical separation should be avoided. A possible disadvantage is that the arrangement may make the

process of checking the tag less efficient, and thereby impede performance.

single-tag rule

If protection tags are carried along with data, the tags will consume file space. Such space is likely to be costly because of the exceptional treatment given this file. Also, since each tag increases the quantity of information contained in the file, additional time may be required to locate a given protected field.

To conserve file space no more than a single protection tag should be attached to any such field. Thus if further protection is desired for a field that is already protected, instead of adding a second tag to the field, a single new tag must replace the old one. The new tag will signify the combination of the old and the new protections. This procedure prevents long lists of tags from being attached to data in the file.

The single-tag rule may, however, prove to be impracticable if the length of the average protected field does not greatly exceed the the length of the tag. A major part of the file space would then be allotted to the protection tags, a requirement that is undesirable. If this is the case, it may be necessary to limit protected fields to larger units, or alternatively, to impose restrictions on format. As a third possibility, the number of protected categories could be restricted.

compartmentalization

However, if the single-tag rule is followed, each distinctly protected segment of data will possess one tag, and that tag alone must serve to identify all specific restrictions placed upon the segment. It is convenient to make use of the tagging system to organize the protection classification scheme, so that each tag itself constitutes a protection category. All data that are similarly restricted to certain users are assigned a common protection tag. Therefore, those data are assigned to a common protection category.

For convenience, such a category of data is called a *group*, which we define as the most elementary, atomic protection category. There are no subcategories within groups with respect to protection. Privilege to any information within a group implies privilege to all information in that group.

Protection groups are discrete and compartmentalized. Any item of protected data is assigned to one and only one group. Every group is an independently existing entity, and is not affected by changes in other groups. This strategy is similar to the scheme devised by Hsiao,[18] except that in our case protection applies to individual records rather than to files as a whole.

The compartmentalized scheme contrasts with the concept of a stratified or multilevel security classification scheme.[16] There can

be no levels or structures in the compartmentalized system. Every protected record is as absolutely restricted to its delegated users as any other. The notion of levels of security, such as CONFIDENTIAL, SECRET, or TOP SECRET, does not apply in the compartmentalized system. In a compartmentalized scheme, such levels simply constitute nonfunctional information to be carried in addition to the protection categories.

Compartmentalized protection groups correspond to the military notion of NEED TO KNOW. It is undesirable to automatically release data with, say, CONFIDENTIAL protection to anyone who has access to SECRET data. There may be an occasion when certain SECRET data has to be disclosed to someone who is not authorized to see certain CONFIDENTIAL data. However, a NEED TO KNOW for one category of data does not imply a NEED TO KNOW for another category. In other words, protections should not have interdependencies or hierarchy. Weissman[6] and Graham[7] have described schemes combining NEED TO KNOW and protection levels. It is not evident, however, whether these approaches are more useful than compartmentalization alone.

The compartmentalized, single-tag approach requires a less complex test for violations than some alternative methods. For example, if a field has more than one associated protection tag, a non-trivial test may be required to determine whether a request for access is permissible. In contrast, if each field has only one tag, the test could consist only of comparison of that tag with the explicit privileges of the requester, which may reduce the processing time to approve the request.

The compartmentalized system also provides a high degree of flexibility. As previously noted, users' privileges to data may be complex, changing, and without a well-defined hierarchy. If any associations of protection groups and users are built into the basic file organization, small changes in relationships could require reconstructing much of the file. A compartmentalized system, on the other hand, makes possible the association of any record with any clique of users. A distinct user *clique* is definable for each group of records.

The term clique refers to a set of users such as "Department Managers" or "Corporation Attorneys," in contrast to a *group*, which is defined as a set of data requiring a common type of protection. The protection of a group always consists of granting to some user clique the exclusive privilege to that data group; thus there may be a one-to-one correspondence of groups and cliques. Two terms are used here because it is sometimes desirable to make the same data available in different ways to different users. A certain group of data, for example annual salaries, may be released to the clique of payroll department members on a read-only basis, whereas

those same data may be released to the clique of payroll clerks on a read-write basis. Incidentally, that second clique might consist of only one individual.

Compartmentalized data would be partitioned into a multitude of disjoint groups, thereby providing complete freedom to associate individuals with protected data. Changes could be readily made. An individual would be delegated privilege to a specific set of data by adding the explicit protection code for that data group to his security profile. Privilege is revoked by divesting his profile of that protection code. Such operations do not affect the individual's privileges to any data except those explicitly delegated or revoked. If a subset of some previously defined group of data is to be altered in respect to its restrictions to users, the original protection group must be redefined as two groups according to this distinction.

The compartmentalized protection scheme has the disadvantage that it might lead to the proliferation of protection groups, which may impose an administrative burden on the authorization system.[4] However, this burden may be tolerable, as will be described later in this paper. To aid processing, an auxiliary table of the interrelations of protection groups may be maintained in addition to the primary classification system. Such a table would serve as a convenience and would be distinct from the primary system.

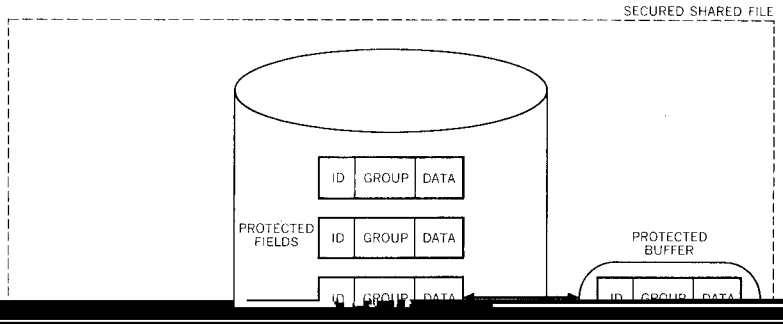
The primary directory of the authorization system is a matrix of profiles, in which data group privileges are listed with respect to each user. Of course, such a matrix could be represented as well in transposed form, whereby all privileged users would be listed with respect to each data group. In the latter case, when access is requested, the list of privileged users for the requested data group would be searched for the user who issued the request. If there are many more data groups than users, searching a row of the transposed matrix would usually be shorter than searching the untransposed matrix. However, there would be more rows in the transposed matrix, so that it might take longer to locate the appropriate row.

A system model

The foregoing guidelines are now expanded into an illustrative model of an authorization system. This system is necessarily hypothetical, because any implementation must take account of specific hardware, software, applications, load considerations, and file organization. The organization of the system is shown in Figure 1.

We assume that the computer runs under an operating system that supports several remote-terminal users who carry on separate jobs simultaneously. For work space, each user is allocated a private bounded region of the computer's main storage. Within his region,

Figure 1 Model authorization system



protection scheme must be effective in preventing intrusions into the authorization program region.

The secured shared file resides in an auxiliary storage medium, or possibly a set of auxiliary media, which are accessible only to the authorization system. These media (indicated in Figure 1 as a disk storage) and their associated channels should be protected against invasion, bugging, or physical removal. It must not be possible for a user to switch the secured shared file to normal channels so as to circumvent the authorization system.

group tags and ID

There may be normal, freely accessible channels for files that are neither secured nor shared. These files are not under control of the authorization system. To avoid overburdening secured shared files, users could be directed to use them only for data they expect to share selectively. However, in an actual system it may be more economical to use the secured shared-file mechanism for all secondary files even though the mechanism is not required in each case. According to the suggested guidelines, protection information is carried adjacent to the data in the file. This protection information, shown in the figure as GROUP and GR, is termed the "group tags." All data that are to be similarly restricted to a common set of users should be assigned the same unique group tag. As the the system is used, it may happen that more than one group of data may be defined that are similarly restricted to the same users. The authorization system searches the files periodically to discover such equivalent groups, and then adjusts those group tags to be identical.

As suggested, each protected field possesses a single group tag. Again these fields need not correspond to physical records of the storage medium. They may be of different length from physical records, nor need the protected fields be of fixed length. Means must be provided, however, to identify these fields. The identification, ID, serves to distinguish protected fields within a single group or within distinct groups. Precise identification conventions are more properly a topic of file organization and accessing, and are not considered in this discussion. The ID could be carried along with the field, as shown in Figure 1, or alternatively, the ID could be held in a separate directory. It is, however, desirable to keep the group tag distinct from the ID, since the ID provides the name of a protected field in the files, whereas the group tag serves to protect the field. By means of this distinction, a protected field that had been improperly selected through an error of identification can, nevertheless, still be detected, and then withheld during the testing period.

public status

Testing may sometimes be omitted by the use of special *public status* categories of the tag. A public status tag indicates that normal protection is not required. The categories could be indicated when a tag value falls within certain numeric ranges, as shown by the following table:

Tag	*Status*
0	Default tag for a completely unprotected field. Any user may read, alter, or write.
1 to 20,000	Publicly available for reading or updating, but may not be deleted except by users with appropriate privilege.
20,001 to 40,000	Publicly available for reading only.
40,001 to 60,000	Publicly available as an executable program only.
Over 60,000	Fully private.

profiles

In addition, there could be a category permitting public availability for updating but not reading, although it is questionable whether such a category is desirable. The actual tag ranges should be determined by statistical studies of the usage of public categories. The authorization system also must provide a security profile for each user. Such a profile consists of a complete list of data protection groups for which the user has access privileges.

When a user logs on and is properly identified, the authorization system copies that user's security profile into a control block located in a protected main storage region, as shown by the heavy arrow at the bottom of the figure. By holding this copy in main storage, the authorization system can complete the confirmation of requests more quickly than otherwise. Whenever a user requests access to the secured shared file, the authorization system consults his profile to determine whether he has the required authority.

privilege levels

Associated with each group tag in the profile there is also a field to specify the level of privilege that the user has been granted for that group. (Privilege is indicated by "PR" in the profile area of the figure). Four levels of privilege may be sufficient, which could be represented with a two bit code, as follows:

Code	*Privilege Level*
00	Execute only. Fields are available only as programs for execution; they cannot be displayed or dumped.
01	Read only. User may execute and read, but not modify these fields.
10	Update. User may execute, read, and modify fields

If fields are available only as executable routines, the authorization system loads the fields into an inaccessible main storage region in preparation for execution. In other cases, fields are copied into the user's area.

Additional privilege levels may be desired as, for example, "create but not read," "update but not read," or "create but not delegate." Such categories, however, may lead users to enter material that they are unable to review or control. Consequently, the file may become cluttered with data of doubtful accuracy, and which cannot readily be verified, corrected, or removed. In contrast, the four privilege levels originally suggested can achieve the same objectives without causing these problems. The purpose of "create but not read," for example, is to prevent a user from browsing through data in a category even though he is authorized to enter data into that category. The same ends, however, would be served if each such user establishes a private group for recording data and then delegates privilege to an official (or to a privileged program) to copy his group into a special master table. The individual user would lack privilege to read that table, but he would be able to review and correct the original entries in his private file. Thus, users can prepare data for entry in the master table, although they are not able to read the table. Similarly, "update but not read" and "create but not delegate" could be realized by application of the four previously suggested privilege levels.

System operation

file operation mode

The model authorization system operates in two modes. In the file operation mode the system executes on demand when a user requests a service relating to the secured shared file. The background mode is used for inspecting and revising files for consistency. We first consider the file operation mode, which includes the following commands:

LOG ON/LOG OFF
DELEGATE (REVOKE) PRIVILEGE
READ
EXECUTE
READ FOR UPDATING
UPDATE
CREATE
DELETE

LOG ON and LOG OFF simply involve copying the user profile into his control block after he has been properly identified, and removing it after the user signs off. The DELEGATE (REVOKE) PRIVILEGE command operates on the user security profiles. When this command is given, the authorization system confirms that the

requester has full privilege for the data group referred to; then the system attaches or removes the group tag from the profile of the specified user. Privilege at any level can be delegated. Although operations on user profiles should be executed rapidly, they need not be accomplished as quickly as operations on the secured shared file.

The READ, EXECUTE, READ FOR UPDATING, UPDATE, CREATE, and DELETE commands involve operations on the file that are each composed of the four steps: *request*, *search*, *confirmation*, and *release*.

READ

Request. The user calls the authorization system (heavy arrows between user and main storage on the left side of the figure), and he, or a program he calls, issues a read request that identifies a record in the secured file. Record identification is an aspect of file management, and could be accomplished by such methods as ID field, displacement from base, address, indirect reference, or structure. Optionally, the requester may supply the group name, so that a search of his profile for that group is initiated at the same time.

Search. The file accessing system initiates a search for the record. If group has been specified in the request, a simultaneous search for group privilege is initiated in the user control block.

Confirmation. When the record has been located in the secured file, the authorization system holds it in a protected buffer for confirmation, as indicated at the top of the figure. There may be several such buffers. If a record has public-read status, confirmation is omitted. Otherwise, the group name in the file entry is compared with group privileges listed for the user in the user control block (light arrow to the right of the figure). If confirmation is not achieved, the request may constitute an attempted violation, and should be recorded for audit.

Release. If confirmation is achieved, the portion of the buffer containing the field is copied into the user's space.

EXECUTE is similar to READ, except for the following factors. Confirmation may be omitted if the protected field has public-execute-only status. The authorization system loads the field into a special protected region of main storage rather than into the user's region. When an entire program—linkages, arguments, and pointers to data—has been loaded, the program is executed. If the user wants the program to process his protected files, he delegates privilege for those files to the program when he calls it.

The READ FOR UPDATING command is similar to READ except for the following provisions. An interlock must be provided so that no other user may update the field.[3] The original record is retained after release in the authorization system buffer to await the updated version. User confirmation is waived only if the field has public-updatable status. Otherwise, the user must have at least update privilege for this group. Finally, if the user logs off before completing an update, the field must have its interlock removed.

UPDATE

Request. This command is executable only if a READ FOR UPDATING command had previously been executed for the same field by the user during the current run—otherwise the request is rejected. The same ID should be used as had been used previously in the READ FOR UPDATING. The authorization system compares this ID with the copy of the original record held in its protected buffer. If a match occurs, the user's updated data are copied into the buffer in place of the original data.

Search. As an added precaution, the original protected record must be read from the file again just prior to updating. Therefore, for certain storage media, a search may be initiated.

Confirmation. The same confirmation is required when rereading the entry from the file as was required in the READ FOR UPDATING; namely, the field should be public updatable or the user must have privilege for this group of at least update level. Also, the protected field's ID, status, and group must match those of the updated version held in the authorization system buffer.

Release. When confirmation is complete, the authorization system writes the updated entry into the file. If record sizes are fixed, writing could be accomplished by overwriting the space of the original record. Otherwise, the original record is deleted, and the updated record is added. The interlock is removed.

CREATE

Request. Any user may create an entry in the file. If a user issues a CREATE command, but does not specify group and status, a default group is automatically assigned to the field with private status. The default group name is taken directly from the user's own identification, thereby defining a group unique to that user. Of course, the user may also request the creation of a protected field with another group name or status. At request time, the desired entry is copied from the user space into the system buffer.

Confirmation. If the user has indicated his own name for his group, either explicitly or by default, confirmation is directly provided.

If any other group is indicated, the authorization system first determines whether the user has full privilege for that group.

Release. After confirmation, the entry is copied into the secured shared file from the protected buffer.

The DELETE command is similar to CREATE in that identifications must match, and the user must have full privilege.

background mode

We now outline the background mode which includes the DEFINE GROUP and IDENTIFY CLIQUE commands. The DEFINE GROUP establishes a category of data, and the IDENTIFY CLIQUE command establishes a category of users. The background mode also includes regular inspections to determine that each group has an owner with full privilege, that equivalent groups are merged, and that privileges delegated to cliques of users are granted to each member of the clique.

The DEFINE GROUP command affects both the secured shared file and user profiles. By means of this command, an existing group can be fragmented into several groups, or several groups can be merged into one. A group may also be defined as being "public," in which case the tag value is assigned within the range of one of the special public categories.

Protected fields that have been filed under a user's private default group tag can be shared if the user explicitly delegates privilege for that group to other users. Alternatively, the sharing of protected fields may also be accomplished by designating certain fields to be in a group for which other users already possess privilege.

The DEFINE GROUP command is relatively slow in execution because it requires that the secured shared file be thoroughly searched for all fields tagged with the old group categories. These categories must be redesignated with the new categories. Also, all user profiles that include privilege to the obsolete group categories must be revised before regular processing proceeds. The DEFINE GROUP command, therefore, halts file processing, and should be preformed only in a low priority mode. When issuing this command, the user must hold full privilege for the old groups and, if the new group categories already exist, he must hold full privilege for these groups as well.

Besides the secured shared file and user profile list, a table of user cliques may be provided. Privilege for a data group could thereby be delegated to a clique as a whole. Then when the IDENTIFY CLIQUE command is issued, the authorization system, in the background mode, goes through the list of clique members and attaches the privilege to each member's profile. Incidentally, the existing protection system can be used to protect this clique membership

list. The list may simply be maintained as a secured shared file group. As a consequence, the list will then have explicit sets of users authorized to refer to it and change it.

quantitative projections

The system model is used for illustrative purposes and is not intended for inclusion in an existing or future shared computer without change. Performance of an authorization system can only be evaluated if it is a component of an actual time-sharing system. Although studies and experiments may in time provide empirical information on comparative authorization systems, it may be instructive to consider some projections based on our model.

Assume that the hypothetical system supports 20,000 recognized users from various independent organizations and that 2,000 terminals may be simultaneously connected on-line. If such a large general-purpose, secured shared file system becomes operational, a demand for an extensive exchanging and trading of data and programs may result. For our projections, let us assume that the average user holds privileges for 200 groups of protected data. (There may, of course, be a few users with privileges to all or almost all data groups.) The average user will originate perhaps ten groups of secured data. From these assumptions, it follows that there are 10 times 20,000 (or 200,000) data groups. To distinguish that number of groups, an 18-bit tag is required for each protected field. If we assume that an average entry is 50 bytes (400 bits), it follows that the proportion of information in the secured shared file devoted to protection is 18 bits divided by 400 bits, or 4.5 percent of the total file. (Parity bits or error correction bits, of course, are not included in these figures.) However, if the average protected field is considerably smaller than 50 bytes, the proportion of storage required for protection will grow, which may make the system unacceptable.

Each user's security profile consists of his personal identification field and a list of group privileges and privilege levels. Assume that each user's identification field consists of 15 bytes (120 bits). Of these, 18 bits are needed for each group code, and 2 bits for the privilege level indicator. An 18-bit field is also required to report the number of groups in the profile because the maximum is 200,000. Since the average user is assumed to hold privileges for 200 groups, storage for an average profile is $120 + 200\,(18 + 2) + 18$ (or 4,138) bits. Since 20,000 users are recognized, 82,760,000 bits or 10,345,000 bytes of storage are required for a complete set of profiles.

The authorization system will impose a delay, which is expected to be small in comparison with the file search delay. The most commonly used file commands are expected to be READ, READ FOR UPDATING, and UPDATE. In execution, delays will result from the following authorization system operations:

- Calling the authorization system.
- Copying the field into the buffer.
- Comparing the group tag with the user profile.

It is hoped that the authorization system can be programmed so that system calls involve only a few instructions. Copying a field into the buffer may involve only a single start-input/output instruction. Comparison of the group with the user profile involves executing a short program loop to scan the profile. Since the average user is assumed to hold privilege for 200 groups, an average of 100 loop iterations is expected. The average delay could be reduced by listing the more frequently used groups at the beginning of the profile. Iteration delay could be avoided by the user's supplying the group tag in his request, so that his profile is scanned while the file is searched.

The vulnerability of the hypothetical system should be considered. Only users who hold full privilege for groups can alter security profiles, and they may alter only references to those groups for which their full privilege applies. Otherwise people cannot affect their own or anyone else's profile. Users should not be able to interfere with the authorization system except by a rather unlikely combination of accidents. Even catastrophic system failures do not appear to provide opportunities for such lapses in protection.

Concluding remarks

Selective sharing of information has been considered in this paper only with regard to auxiliary storage in a certain general type of time-sharing system. Further study is needed to provide an efficient authorization system within the central processor's main storage. Effort is also required to enable programs to pass privileges selectively when calling other programs.

ACKNOWLEDGMENT

The author wishes to thank P. S. Dauber for suggesting this study. He also acknowledges the aid of P. R. Schneider, C. J. Stephenson, M. A. Auslander, and M. E. Hopkins; of R. Courtney and L. Moss, who established the conceptual groundwork on which this paper is based; and of A. M. Pfaff for much constructive criticism.

CITED REFERENCES

1. *The Consideration of Data Security in a Computer Environment,* 520–2169, International Business Machines Corporation, Data Processing Division, White Plains, New York.
2. H. E. Petersen and R. Turn, "System implications of information

privacy," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **30,** Thompson Book Company, Washington, D. C., 291–300 (1967).
3. J. B. Dennis and E. C. van Horn, "Programming semantics for multiprogrammed computation," *Communications of the Association for Computing Machinery* **9,** No. 3, 143–155 (March 1966).
4. L. J. Hoffman, "Computers and privacy: a survey," *Computing Surveys* **1,** No. 2, 85–103 (1969).
5. W. H. Ware, "Security and privacy in computer systems," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **30,** Thompson Book Company, Washington, D. C., 279–282 (1967).
6. C. Weissman, "Security controls in the ADEPT-50," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **35,** 119–133, AFIPS Press, Montvale, New Jersey (1969).
7. R. M. Graham, "Protection in an information processing utility," *Communications of the Association for Computing Machinery* **11,** No. 5, 365–369 (May 1968).
8. P. A. Crisman, Editor, *The Compatible Time-Sharing System—A Programmer's Guide,* MIT Press, Cambridge, Massachusetts (1965).
9. System/360 *Time-Sharing System, Concepts and Facilities,* C28–2003, International Business Machines Corporation, Data Processing Division, White Plains, New York.
10. B. W. Lampson, "Dynamic protection structures," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **35,** 27–38, AFIPS Press, Montvale, New Jersey (1969).
11. W. F. Friedman, "Cryptology," *Encyclopaedia Britannica* **6,** 844–851, Chicago, Illinois (1967).
12. C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal* **28,** No. 4, 656–715 (October 1949).
13. R. O. Skatrud, "A consideration of the application of cryptographic techniques to data processing," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **35,** 111–117, AFIPS Press, Montvale, New Jersey (1969).
14. P. Baran, "Communications, computers and people," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part 2, Thompson Book Company, Washington, D. C., 45–49 (1965).
15. R. C. Daley and P. G. Neumann, "A general-purpose file system for secondary storage," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part 1, Spartan Books, New York, New York, 213–229 (1965).
16. B. Peters, "Security considerations in a multiprogrammed computer system," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **30,** Thompson Book Company, Washington, D. C., 283–286 (1967).
17. W. H. Ware, "Security and privacy: simularities and differences," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **30,** Thompson Book Company, Washington, D. C., 287–290 (1967).
18. D. K. Hsiao, "A file system for a problem solving facility," Ph.D. Dissertation (Electrical Engineering), University of Pennsylvania, Philadelphia, Pennsylvania (1968).