Data Security*

DOROTHY E. DENNING AND PETER J. DENNING

Computer Science Department, Purdue University, West Lafayette, Indiana 47907

The rising abuse of computers and increasing threat to personal privacy through data banks have stimulated much interest in the technical safeguards for data. There are four kinds of safeguards, each related to but distinct from the others. Access controls regulate which users may enter the system and subsequently which data sets an active user may read or write. Flow controls regulate the dissemination of values among the data sets accessible to a user. Inference controls protect statistical databases by preventing questioners from deducing confidential information by posing carefully designed sequences of statistical queries and correlating the responses. Statistical data banks are much less secure than most people believe. Data encryption attempts to prevent unauthorized disclosure of confidential information in transit or in storage. This paper describes the general nature of controls of each type, the kinds of problems they can and cannot solve, and their inherent limitations and weaknesses. The paper is intended for a general audience with little background in the area.

Keywords and Phrases: security, data security, protection, access controls, information flow, confidentiality, statistical database, statistical inference, cryptography, capabilities, public-key encryption, Data Encryption Standard, security classes

CR Categories. 4.35, 4.33, 1.3

INTRODUCTION

The white-collar criminal, the old adage goes, is a man who has learned to steal with a pencil. In the last decade or two, a few of the more perceptive of these entrepreneurs have discovered that the rewards are greater and the risks lower if they steal with a computer. There have already been some spectacular thefts but, so far, computer criminals have been handicapped by a lack of technical know-how and by a certain inability to think big. The sums involved in typical cases of computer abuse would have been front-page news had they been stolen by armed desperados but have generally been smaller than the haul that might have been made by someone with more expertise and boldness.

The records of hundreds of cases of computer abuse have been analyzed by Parker [PARK76]. Parker believes many more cases probably remain undetected or unreported. Banks in particular are not eager to acknowledge that they have been embezzled. The median loss in reported cases was almost $500,000, and the total known loss from all computer crime has been about $100 million annually. These figures are destined to rise unless effective countermeasures are taken against the more expert attacks of the second generation of computer criminals, who are now learning their trade.

About 40 percent of reported abuses were data entry problems. Most of the rest were thefts or embezzlements by a trusted employee who misused his access to the computer. A few were malicious pranks or sabotage. Nearly all the known cases involve breaches of external security. So far, very few computer crimes have involved breaches of internal security: design flaws

* This work was supported in part by National Science Foundation Grant MCS77-04835.

CONTENTS

within the computer system itself. But the rapid proliferation of computers and the increasing sophistication of users make businesses and individuals increasingly vulnerable to abuse by computer experts. As the potential rewards increase, so will the sophistication of attacks on computer systems.

An expert criminal, for example, might intercept electronic-funds-transfer messages between two banks; within a few hours he could steal, without a trace, several millions of dollars. An investigative reporter might deduce from questions answered by a medical information system that a senatorial candidate once took drugs for depression; if published, this information might force the candidate to withdraw from the election even though he had been cured. An employee of a government agency might blackmail citizens using information purloined from a confidential data bank accessible over a federal computer network.

These three speculations represent breaches of internal security. Internal safeguards for data security have been actively studied since the early 1960s, and in anticipation of future security threats this work has been intensified in the last few years. Systems designers and engineers are developing hardware and software safeguards, and theoreticians are studying the inherent complexity of security problems. Although we have made considerable progress, there is still a wide gap between the safeguards that can be implemented in the laboratory—safeguards well within the reach of current technology—and those available in most commercial systems. Some of the safeguards that users want are theoretically impossible or would be prohibitively expensive.

This last point is probably the most important. Absolute security is no more possible in computer systems than it is in bank vaults. The goal is cost-effective internal safeguards, sufficiently strong that computer hardware and software are not the weakest links in the security chain.

In this paper we summarize current research in *internal security mechanisms*, how they work, and their inherent limitations. Internal security controls regulate the operation of the computer system in four areas: access to stored objects such as files, flow of information from one stored object to another, inference of confidential values stored in statistical databases, and encryption of confidential data stored in files or transmitted on communications lines. The problems these four types of mechanism attempt to control are illustrated in Figure 1.

We have not attempted to treat *external security controls*, which affect operations outside the main computing system; examples are personnel screening, limiting access to the computer room and to certain terminals, fire protection, and protection of removable media against destruction or theft. Some security mechanisms lie at the interface between users and the system; examples are user authentication, password management, security monitoring, and auditing. These are discussed only in relation to internal security mechanisms. Neither have we attempted a treatment of privacy and the law. Our objective is simply an overview of four areas of security research.

Other papers and books that treat internal controls are ANDE72, GRAH72, HOFF77, HSIA78, MADN79, POPE74, SALT75, and SHAN77. The book DEMI78 is a collection of recent papers on security research.

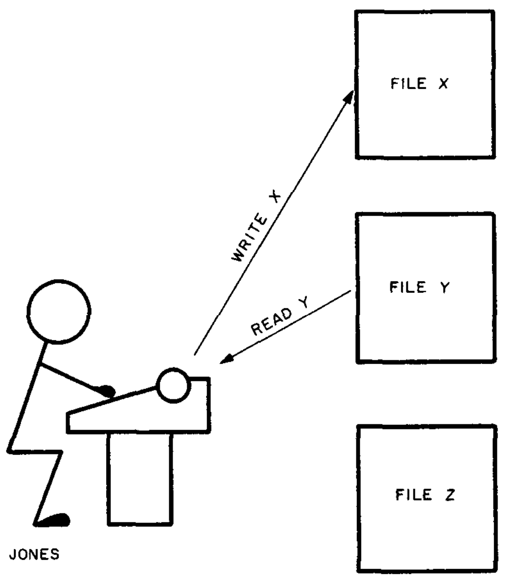


FIGURE 1a. ACCESS. Jones can be permitted to read file *Y* and write in file *X*; he has no access to file *Z*.

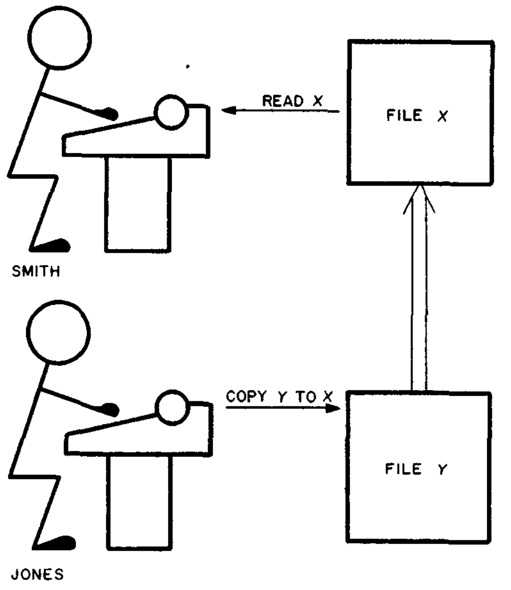


FIGURE 1b. FLOW. Denied access to file *Y*, Smith gets confederate Jones to make a copy; flow controls could prevent this.

FIGURE 1. Four kinds of security controls.

FIGURE 1c. INFERENCE. A questioner used prior knowledge to deduce confidential information from a statistical summary; inference controls could prevent this.

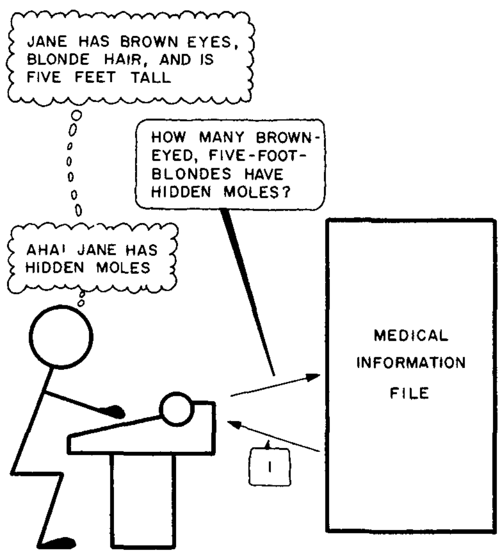
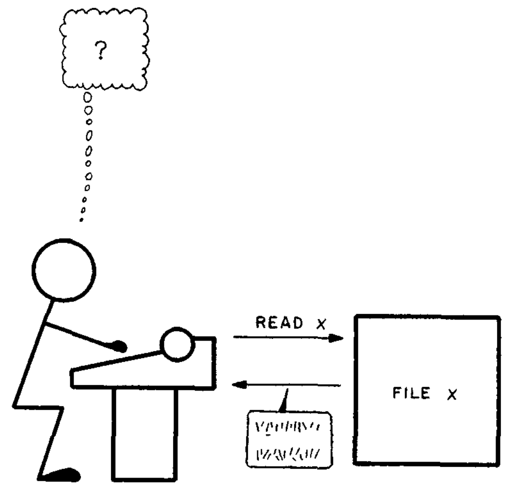

JANE HAS BROWN EYES, BLONDE HAIR, AND IS FIVE FEET TALL
HOW MANY BROWN-EYED, FIVE-FOOT-BLONDES HAVE HIDDEN MOLES?
AHA! JANE HAS HIDDEN MOLES
MEDICAL INFORMATION FILE
1

FIGURE 1d. ENCRYPTION. Jones illicitly obtains a copy of file *X*; but its encrypted contents are meaningless to him.



Overviews of external controls are given in MADN79, NIEL76, PARK76, SALT75, and SHAN77. The general concepts of all controls are reviewed in GAIN78. The issues of privacy and the law and their technological implications are discussed in PARK76, RPP77, TURN76, and WEST72.

1. ACCESS CONTROLS

Access controls regulate the reading, changing, and deletion of data and programs.

These controls prevent the accidental or malicious disclosure, modification, or destruction of records, data sets, and program segments. They prevent malfunctioning programs from overwriting segments of memory belonging to other programs. They prevent the copying of proprietary software or confidential data.

Many access control systems incorporate a concept of *ownership*—that is, a user may dispense and revoke privileges for objects he owns. This is common in file systems intended for the long-term storage of one's own data sets and program modules. Not all systems include this concept; for example, the patient does not own his record in a medical information system. Access control systems for owned objects must efficiently enforce privileges that are added, changed, or revoked.

The effectiveness of access controls rests on three assumptions. The first is *proper user identification*; no one should be able to fool the system into giving him the capabilities of another. Authentication schemes based on passwords are common and simple, but they need safeguards to thwart systematic penetration [GAIN78, MORR78, PARK76, SALT75, SALT78]. Schemes based on identifying personal characteristics such as voiceprints or dynamic signatures are more reliable, but more expensive. The second assumption is that *unanticipated observers do not gain access* by stealing tapes or disk packs or by wiretapping. The usual safeguard is encryption, which is discussed later—information that could be monitored by strangers is scrambled. The third assumption is that *privilege-information is heavily protected*; this is all the information that specifies the access each program has to objects in the system. No user's program can write into the segment containing its own privilege specifiers. Privilege-information is accessible only to authorized programs of the supervisor, and the privilege to call these programs is itself controlled.

The following subsections consider two important classes of access control mechanisms for transaction-processing systems and for general purpose programming systems. We intend our treatment as a guide to the literature, not a detailed study of the many trade-offs that must be faced in practice.

Controls for Transaction-Processing Systems

The commands issued by the user of a transaction-processing system are calls on a small library of "transaction programs" that perform specific operations, such as querying and updating, on a database [DENN71]. The user is not allowed to write, compile, and run arbitrary programs. In such systems the only programs allowed to run are the certified transaction programs. Therefore it is possible to enforce the rules of access at the interface between man and machine.

A database management system is an example. A user can identify a set of records by a "characteristic formula" C, which is a logical expression using the relational operators ($=$, $\neq$, $<$, etc.) and the Boolean operators (AND, OR, NOT); these operators join terms which are indicators of values or compositions of relations. An example is

$$C = \text{"FEMALE AND PROFESSOR OR (SALARY} \geq \$20\text{K)."}$$

The transaction program looks up a formula R specifying restrictions that apply to the given user; it then proceeds as if the user had actually presented a formula C AND R. The concept of adding to the requests of a user constraints that depend on that user is common in data management systems [BONC77, STON74].

This form of access control is potentially very powerful. The restrictions R may include *data-dependent restrictions*, which are also functions of the current values of the data [CONW72], or *history-dependent restrictions*, which are functions of the records previously accessed [HART76]. Implementing these kinds of restrictions can be very difficult. We refer the reader to HSIA78 for details.

When the system allows owners of records to revoke privileges that may have been passed around among users, it must be designed to revoke also any privileges that emanated from the revoked privilege. Griffiths, Wade, and Fagin have studied a revocation method that stamps each privilege-specifier with the time of its creation [GRIF76, FAGI78].

Controls for General Purpose Systems

General purpose systems permit users to write, compile, and run arbitrary programs.

It is not possible to certify a priori that arbitrary programs will forever meet the (changing) access rules of the system, or that there will never be program failures or equipment malfunctions. Therefore, these systems provide access control mechanisms as part of the run-time environment, often with hardware support for performing access checks in parallel with the main computation. These mechanisms are typically based on *object-dependent controls* (as opposed to data-dependent controls), which regulate access to an object irrespective of the values stored in that object.

Object-dependent controls are also needed in transaction-processing systems to protect against faulty transaction programs, equipment malfunctions, and intruders—problems that cannot be prevented simply by "certifying" the transaction programs.

Object-dependent controls can be enforced by type-checking in compilers for languages with abstract data types, or by run-time checking in the hardware. A proposal for compiler-based enforcement is given by Jones and Liskov [JONE76]. An illustration of hardware-based enforcement is given in the next section.

Example of an Object-Dependent Design

This section illustrates the architecture of object-dependent access controls. The central concept, *capability addressing*, has been the subject of considerable research. Detailed treatments of this design and its trade-offs can be found in DENN76b, FABR74, LIND76, ORGA72, ORGA73, and SALT75.

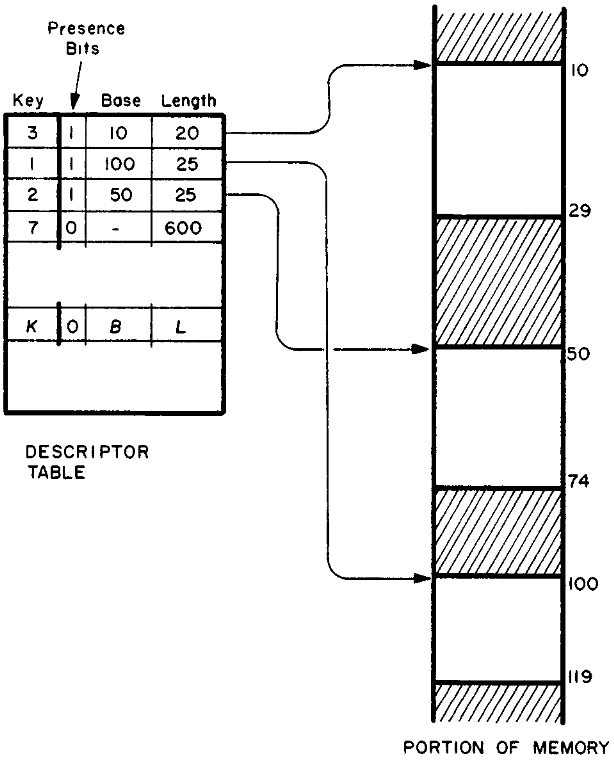


FIGURE 3. Multiple-segment storage.

Most systems have a large number of segments (data sets and programs) which are kept in the slow-speed bulk store under the auspices of a file system. When an active program requires access to a segment, the operating system provides that program with a "capability" for the segment. All of a program's capabilities are stored in a "capability list," which is used by the addressing hardware when interpreting the program's virtual addresses. We first describe the operation of the addressing hardware. Then we describe how a program acquires capabilities.

Figures 2–4 summarize the mechanism for verifying attempted accesses to segments stored in the main memory. Figure 2 shows a copy of a 20-word segment stored in memory at the beginning (base) address 10. A descriptor of the form (B, L) records the base address B and length L of the segment. Programs refer to words in seg-

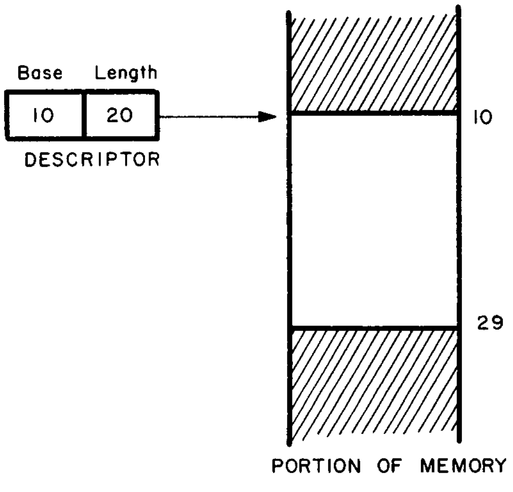


FIGURE 2 A descriptor addressing a segment of memory.

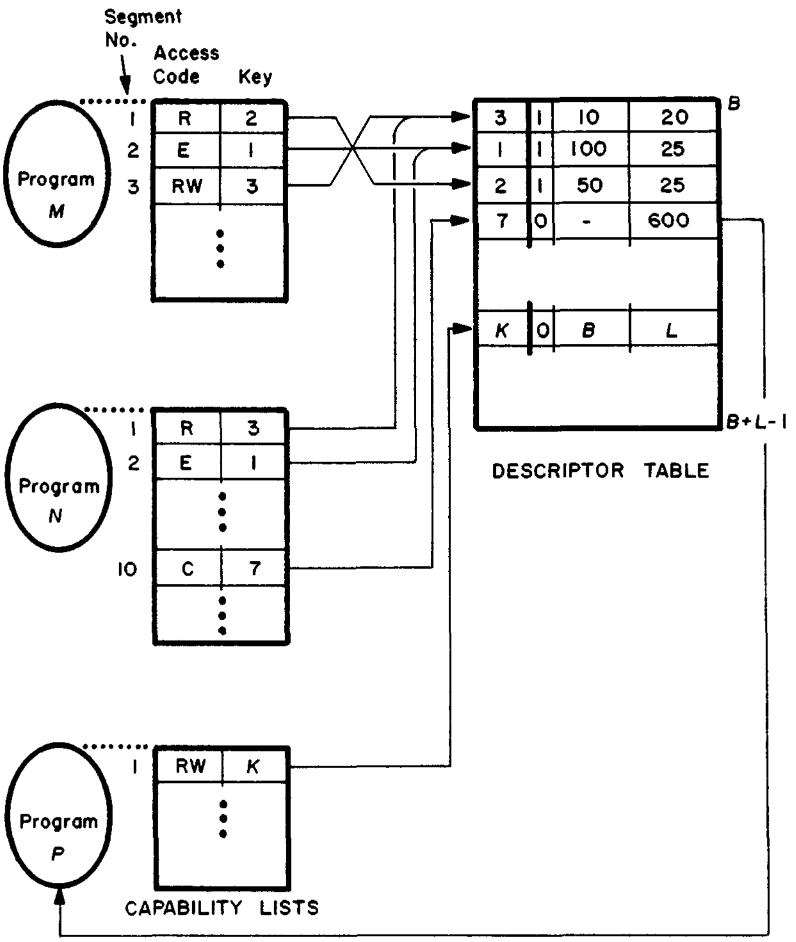


FIGURE 4. Capability lists for accessing segments.

ments by displacements (line numbers). The command "Read D" refers to the Dth line of the segment—that is, memory address $B + D$. A reference is valid only if the displacement is in range—that is, if $0 \leq D < L$.

Figure 3 shows that descriptors of all memory segments may be kept in a descriptor table. Each descriptor is identified by a unique key K. Now a program refers to a segment by specifying the displacement and the key—thus "Read K, D" refers to the Dth word in the segment whose key is K. Each descriptor is stored with a "presence bit" that tells whether the associated segment is in the main store; if it is not, as for key 7 in Figure 3, an attempted reference will trigger a "missing segment fault" that will cause the operating system to suspend the program and load the segment. This scheme confers considerable flexibility because the operating system can freely move segments between main and secondary memory merely by updating the descriptors. Because the keys do not change, no program is affected by relocations of segments in the memory hierarchy.

Figure 4 shows the final step in the scheme: the capability lists are associated with programs. The access code of a capability specifies one or more kinds of access: read (R), write (W), execute (E), or call (C). Read access permits a program to copy a value out of a memory segment; write access permits a program to store a value in a memory segment; execute access permits a processor to fetch instructions from a segment; and call access permits a program to execute a procedure call instruction with that segment as the target. Execute access is valuable for restricting access to privileged operations. A program actually refers to a segment by a segment number S and a displacement D. Thus "Read S, D" refers to the *S*th segment in the capability list of the program. The reference is valid only if the *S*th capability specifies R-access and contains key *K*, with *D* being within the range specified in the *K*th descriptor.

This strategy has special advantages when segments are shared. Each program can be given its own capability, with its own access code, for the common segment; the common segment can be used by different programs whose authors require no prior agreement on the local segment numbers. In Figure 4, programs *M* and *N* share the segment with key 3.

With this mechanism each program module can have its own capability list. In Figure 4, program *N*'s call access for program *P* is stored in the tenth segment of *N*. If program *N* executes the command "Call 10," control will pass to program *P*. This resembles a standard subroutine call, but with the important difference that the called program has a capability list different from that of the caller. Program *P*, whose first capability designates the memory segment containing the descriptor table, would be a certified program of the operating system; it would screen all requests to update descriptors. When program *P* executes a "Return" command, program *N* resumes execution after the call and its capability list is again in control.

The concept of giving each program its own set of capabilities supports the *principle of least privilege* [DENN76b, LIND76, SALT75]. Each capability list need contain entries only for the segments required for the associated program to carry out its task. Damage is confined in case the program contains an error. Untrusted programs can be encapsulated and cannot endanger unrelated programs. Critical data, such as the descriptor table of Figure 4, can be hidden away, tamperproof, in a domain accessible only to the program certified to manipulate it.

The foregoing discusses how capabilities are used to limit access. We turn now to the question of how programs obtain their capabilities in the first place.

Figure 5 illustrates how a file system attaches privilege-specifiers to permanent files. All users are registered in a master directory. Each user lists all his files in a personal directory, of which each entry specifies the file's name (*N*), length (*L*), address in the bulk store (*BA*), a list of authorized users (*AL*), and the file's unique identifier (*K*). Each entry in an authorization list specifies the name and type of access permitted of some individual designated by the owner. The figure shows that Jones has granted read (R) permission for his file *Y* to Smith and himself, write (W) permission to himself alone, and no access to Cox. If a program owned by Smith attempts access to Jones's file *Y*, the operating system will intervene and insert a capability (R, *K*) at some position *S* in the capability list of Smith's program; thereafter, Smith's program can access the file by issuing read commands with the segment number *S*. This is the essence of the scheme used in MULTICS [ORGA72, SALT75, SCHR72].

It is also possible to create a program's capability list during compilation. This is natural in an environment where the program modules correspond to managers of extended-type objects, and the capabilities point to the components of a particular object. This view, proposed by Dennis and Van Horn [DENV66], is used in the Hydra system [COHE75], the CAP system [NEED77], and the Plessey System 250 [ENGL74]. (See GEHR79 for a review.)

Some systems permit owners to revoke privileges. If all privilege-specifiers are stored in a central table, it is a relatively simple matter to purge them [GRIF76, FAGI78]. But if they are scattered throughout the system in capability lists, revocation becomes considerably harder: the descriptor table must contain chains of descriptors that can be broken by an owner; this renders revoked capabilities useless but does not purge them [NEED77, REDE74, SALT75].

Limitations of Access Controls

Most security flaws in existing systems are the consequences of design shortcuts taken to increase the efficiency of the operating system. Hardware designed to support access control efficiently would go a long way toward removing these flaws. An example is the high overhead in managing small memory segments. Users are forced to pack data and subprograms into large segments, which means that small program blocks cannot be individually protected. This

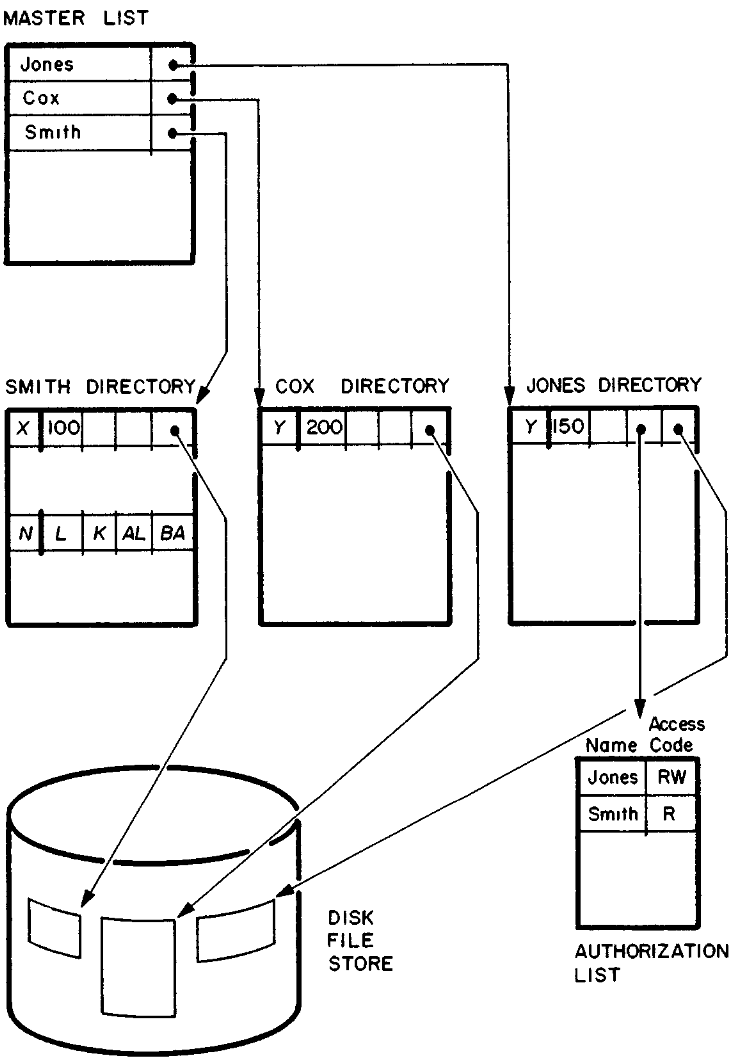


FIGURE 5. Access controls for permanent files.

makes the ideal of a separate capability list for every program very difficult to achieve. Radical changes in computer architecture, designed to bridge the "semantic gap" between concepts in the programming language and concepts in the machine language, may be needed to overcome this difficulty. Myers's SWARD machine [MYER78] and Gehringer's "typed memory" [GEHR79] point in the right direction.

Another serious problem with existing systems is excessive privilege vested in the operating system. A supervisor mode of operation takes over when the user's program calls any operating system program. The supervisor mode overrides all or most of the storage protection mechanisms. The supposedly correct and trustworthy supervisor programs can manipulate capabilities and segments without restriction [WILK68]. This difficulty is ameliorated somewhat in MULTICS, which has a linear hierarchy of supervisor states, called rings, that confer successively greater privilege; the user can operate some untrusted subprograms in rings of low privilege [SCHR72]. Contrary to the principle of least privilege, systems based on supervisor states permit programs in higher rings to run with much more privilege than they require for their tasks. There is no efficient way for two cooperating subprograms to have nonoverlapping sets of privileges.

The supervisor mode problem is an instance of exposure to the general problem of "trojan horses" [LIND76]. It arises when a subprogram written by an outsider is brought into the domain of a given user. With all the user's privileges and possibly more, the subprogram is free to wreak havoc—for example, by erasing files or entering erroneous data. A system capable of running each subprogram with its own set of capabilities offers the best practical defense against trojan horses, because the outsider's program can be confined to a domain having the least privilege required for the agreed task.

Access mechanisms as outlined here are feasible today at reasonable cost. The increasing importance of sharing information among different users of a common database, of encapsulating programs, and of limiting damage in case of error or malfunction all contribute to a growing pressure on manufacturers to build better machines.

There are additional limitations that are much more difficult to overcome: proving that a computer system continually meets its access specifications, proving that authorizations are continually consistent with owners' intentions, and proving that information stored in files and segments remains among authorized users. The possibility of hardware malfunction, which can alter information stored in the memory, makes rigorous proofs impossible because it subverts the necessary assumption that all possible changes in system state are controllable. Arbitrarily low risks of damage can be achieved only at correspondingly high investments in error-checking equipment.

Proving that a computer system continually meets its access specifications is straightforward in principle: the prover must show that all programs and hardware for enforcing the current authorizations and for permitting changes in authorizations work as specified. In practice this is easier to say than do because the correctness of many complex programs must be established [GAIN78] and because automatic "program proving" is a distant goal. Much effort has been devoted to developing formal access control models which can be elaborated into the design of a system. At SRI International, the PSOS (Provably Secure Operating System) is structured as a linear hierarchy of 14 nested abstract machines, each of which can be proved correct if the machines below are correct [NEUM77]. This system is expressed in a special language, called SPECIAL, that incorporates specifications explicitly into the programs. Several other research groups have adopted a less ambitious approach; rather than try to prove that the entire system meets all its specifications, they centralize all the operations affecting security into a system nucleus called the "security kernel." The correct operation of the kernel implies that the entire system is secure. (See MILL76, POPE78b, POPE78c, and SCHR77.)

Proving that extant authorizations are continually consistent with owner's intentions is fraught with difficulties. Many systems permit users to grant others subsets of their own privileges. In such systems an owner might well be interested in the *safety problem*, which seeks answers to questions of the form "Can the programs of user X ever gain read access to file Y?" Safety problems are easily answered in the special case of systems conforming to the "take-grant model" [SNYD77, LIPT78]. However, Harrison, Ruzzo, and Ullman have shown that the primitive operations of practical access control systems are sufficiently powerful to encode the state of an arbitrary Turing machine into the extant access control privileges; the halting problem is thereby reducible to the safety problem, which means that the safety problem is undecidable [HARR76]. This result is mainly of theoretical interest, for it is usually possible to answer specific safety questions. However, this result explains why it is impossible to devise a single approach for all safety questions; each one must be analyzed separately.

Proving that stored information remains among authorized users is also difficult because a user who may read a file may also make a copy, perhaps in code, which he can then pass along to someone who is denied access to the original. However, this is not a genuine defect of access controls, which are intended to regulate access to stored objects, but not what happens to the information contained in these objects. Many leaks based on copying can be eliminated by augmenting an access control mechanism with controls on information flow. This is studied next.

2. FLOW CONTROLS

A flow occurs from object X to object Y when a sequence of instructions that reads from X writes a value into Y. Copying file X into file Y is an example of a simple flow. Much more subtle flows are possible, as we note shortly.

Active flow control research began in the early 1970s. Most flow controls employ some concept of security class; the transfer of information from a sender to a receiver is allowed only if the receiver's security class is at least as privileged as the sender's [DENN76a]. A flow policy specifies the channels along which information is allowed to move. Flow controls can prevent a service program from leaking a customer's confidential data. They can block the transmission of secret military data to an unclassified user.

The most general flow controls monitor the detailed data flows in programs. However such controls are often complex and hard to use efficiently. Controls based on security classes are usually efficient, though often exasperatingly conservative.

Flow Policies

The simplest flow policy specifies just two classes of information: confidential (C) and nonconfidential (N), and allows all flows except those from class C to class N. This policy can solve the confinement problem that arises when a service program handles customer data, some of which are confidential [FENT74, LAMP73, LIPN75]. The service program may retain some or all of the customer's nonconfidential data, but it must be prevented from retaining or releasing to its owner any of the confidential data. An income-tax-computing service, for example, might be allowed to retain a customer's address and the bill for services rendered, but not the customer's income or deductions.

Government and military computer systems have a more complex flow policy for classified data. Each security class is represented by two parts (i, x) where i denotes an authority level and x a category. There are usually three authority levels: 1) confidential, 2) secret, and 3) top secret. There are 2^m categories, comprising all possible combinations of m compartments; typical compartments are U (unrestricted), R (restricted), S (sensitive), and C (crypto). Information is permitted to flow from an object with security class (i, x) to one with class (j, y) only if $i \leq j$ and only if the compartments of x are also compartments of y. Transmissions from (2, RS) to (3, RS) or to (2, RSC) are allowed, for example, but those from (2, RS) to (1, RS) or to (3, R) are not.

Mechanisms

Simple flow controls can be enforced by an extended access control mechanism, which involves assigning a security class (usually called the clearance) to each running program. The program is allowed to read a particular memory segment only if its security class is as high as that of the segment. It is allowed to write in a segment only if its class is as low as that of the segment. This automatically ensures that no information transmitted by the program can move from a higher to a lower class. A military program with a secret clearance, for example, can read only from objects which are unclassified, confidential, or secret; it can write only into objects which are secret or top secret. It is forbidden to write into unclassified or confidential objects, or to read from top secret objects. Systems designed at SDC [WEIS69], MITRE [MILL76], Case Western Reserve University [WALT75], and SRI International [NEUM77] are of this type.

The extended access control mechanism has a tendency to overclassify data. Information flows upward but never downward. This problem can be mitigated by letting the security class of a program rise while running, so that it is only as high as the highest security information that the program has read. In this case the security class of the program is a "high-water mark" rather than a clearance. This reduces but does not eliminate overclassification because the high-water mark cannot be lowered.

An important limitation of the extended access control mechanism is its lack of generality. For example, if the income tax program mentioned earlier is confidential, it will be forbidden to process confidential customer data; if it is confidential, it will be forbidden to write nonconfidential information into any of its files. A usable operating system could not be secured by this mechanism—all its outputs would have to be classified at least as high as every class of information stored within. This limitation results from the implicit assumption that any input of a program can flow to any output, which forces the designer to assume that the confidentiality of each output is as high or higher than the confidentiality of every input.

To be free of this limitation, flow controls must be able to examine the way information flows through the statements and variables of a program, determining precisely how inputs flow to each output. This is not straightforward. Suppose that x is 0 or 1 when this statement is executed:

if $x = 0$ **then** $y := 0$ **else** $y := 1$.

This statement copies x to y *implicitly* by encoding the value of x into the control flow. (Note that this program still transmits some information from x to y even if the initial value of x is unknown.) Implicit flows of this type can be easily detected by associating with the program counter a dynamic security class corresponding to the Boolean expressions which influence it.

Here is a program fragment specifying a flow that is more difficult to detect:

while $x \neq 0$ **do** skip; **print** ('done'); **stop**.

This program will print 'done' only if $x = 0$; otherwise it will enter an "infinite" loop, which actually means it will eventually be terminated abnormally. In this case the output produced by the program reveals the position of the program counter when the program stops, thereby revealing the information encoded therein. A partial solution results if one can prove that the loops of one's program all terminate [REIT78]. However, all types of abnormal program termination present problems for flow detectors [DENN76a, DENN77].

Several techniques are available for detecting and verifying internal flows of programs. The most common employ traditional methods of program verification. The allowable input/output dependencies of program modules are stated as formal assertions, which are then proved by tracing data flows within the program. This approach was used for the systems at MITRE [MILL76], UCLA [POPE78c], and SRI [NEUM77]. A simpler approach results when security classes can be declared for the variables in the program; using a technique similar to type-checking, the compiler can verify that each program statement specifies flows that are consistent with the given flow policy [DENN77]. Although highly efficient, this method is also highly conservative: it will reject programs that would be certified under a traditional flow analysis. Cohen's information-theoretic scheme relieves this difficulty by certifying all flows that will occur during some execution of the program [COHE77, COHE78]. Furtek and Millen have proposed a theory of information flow using analogs of prime implicants of switching theory [FURT78, MILL78]; it too can lead to more precise certification. Reitman and Andrews have incorporated the flow semantics of DENN77 into the formalism of program verification for more precise certification [REIT78].

The foregoing methods are based on static analysis; they certify a program prior to execution and require no run-time support. However we do not know how to certify programs that use variables whose security classes can change during execution or whose inputs can have different security classes on different invocations. These cases require a combination of static analysis and run-time checking. A preliminary study of such a system was made by Fenton, whose "data mark machine" tagged every memory cell (and the program counter) with a security class; the processor would inhibit any instruction whose execution would violate the flow policy [FENT74].

Limitations of Flow Controls

We suggested earlier that all mechanisms based on security classes tend to overclassify information, since only upward flow is allowed. This problem can be mitigated by permitting "downgrading"—the manual lowering of security classes by an authorized person. It is also possible to permit downward flows through certain information-losing programs. Such programs are supposed to filter out enough information about their inputs that their results are of lower confidentiality. Not much is known about such programs except that, as we shall observe in the discussion of inference control, many programs believed to filter out information actually do not.

One type of flow cannot be controlled easily, if at all. A program can convey information to an observer by encoding it into some physical phenomenon without storing it into the memory of the computer. These are called flows on covert channels [LAMP73, LIPN75]. A simple covert channel is the running time of a program. A program might read a confidential value, then enter a loop that repeatedly subtracts 1 from the value until it reaches zero, whereupon it stops. The owner of the program can determine the confidential value simply by observing the running time. More complex channels exploit other resource usage patterns such as the electric power consumed while running a program.

The only known technical solution to the problem of covert channels requires that the owner of a job state in advance what resources his job will use and how much time it will take. The requested resources are dedicated to the job, and the results, even if incomplete, are returned to the user at precisely the time specified. This strategy allows the user to deduce nothing from running time or resource usage that he did not know beforehand; but even then he can deduce something from whether his program was successfully completed. This scheme can be prohibitively expensive. Cost-effective methods of closing all covert channels completely probably do not exist.

3. INFERENCE CONTROLS

When information derived from confidential data must be declassified for wider distribution, the rules of flow control must be suspended. This is true of statistical databases (data banks) which contain sensitive information about individuals and must provide various kinds of statistical summaries about the population. The Bureau of the Census, for example, is charged by law to collect information on all citizens and to report summaries of this information without revealing any particulars. Similarly, medical information systems are supposed to produce health statistics but not to release health data about any one patient.

The problem is that summaries contain vestiges of the original information; a snooper might be able to reconstruct this information by processing enough summaries. This is called *deduction of confidential information by inference.* When the information pertains to an individual, the deduction compromises his privacy. The objective of inference controls is to make the cost of obtaining confidential information unacceptably high.

When invoking a query program, a questioner supplies a characteristic formula C, which is a logical formula whose terms are joined by the Boolean operators (AND, OR, NOT). The set of records whose contents satisfy formula C is called the *query set* for C. The query program's response is computed from the query set; it may be the *count* of the records, the *sum* of values in the records, or the *selection* of a value, such as a maximum or median.

A record is compromised if a questioner can deduce its confidential values by correlating responses to his queries using any prior information he might have. Most compromises are based on isolating a desired record at the intersection of a set of interlocking queries. Defenses include controls that withhold response for improper query set sizes and overlaps, controls that distort the responses by rounding or falsifying data, and controls that apply queries to random samples of the database. Examples are given in the next subsections.

Controls on Query Set Sizes and Overlaps

When the questioner has complete control over the query set and when responses are undistorted, compromise is easy. This is illustrated by a dialogue between a questioner and a medical database:

Q: How many patients have these characteristics?
    Male
    Age 45–50
    Married
    Two children
    Harvard law degree
    Bank vice-president
A: 1.

Suppose the questioner knows that Fenwick has these characteristics; he now attempts to discover something confidential about Fenwick from this query:

Q: How many patients have these characteristics:
  Male
  Age 45-50
  Married
  Two children
  Harvard law degree
  Bank vice-president
  Took drugs for depression

This query will respond with "1" if Fenwick has taken drugs for depression and "0" otherwise.

The principle of this compromise is simple. The questioner finds a formula C whose query set count is 1. He can then discover whether the individual thus isolated has any other characteristic X by asking, "How many individuals satisfy C AND X?" (The response "1" indicates that X is characteristic of the individual and "0" indicates not.) This attack was first reported by Hoffman and Miller [HOFF70].

It might seem that this compromise could be prevented by a *minimum query set control:*

> Do not respond to queries for which there are fewer than k or more than $n - k$ records in the query set, where n is the total number of records in the database.

The positive integer k in this control is a design parameter specifying the smallest allowable size of a query set. If the query language permits complementation, a maximum size $n - k$ of the query set must also be enforced, for otherwise the questioner could pose his queries relative to the complement (NOT C) of the desired characteristics (C).

Unfortunately, this control is ineffective. Schlorer showed that compromises may be possible even for k near $n/2$ by the technique of a "tracker" [SCHL75, SCHL79]. The basic idea is to pad small query sets with enough extra records to make them answerable, then subtract out the effect of the extra records. Schlörer called the formula identifying the extra records the *tracker* because the questioner can use it to "track down" additional characteristics of an individual.

Suppose that a questioner, who knows from external sources that an individual I is characterized by the logical formula C, is able to express C in the form C = (A AND B) such that queries for the formulas A and (A AND NOT B) are both answerable. Schlörer called the formula T = (A AND NOT B) the tracker of I. Table 1 shows a database recording $n = 8$ secret political contributions. Suppose that $k = 2$; then responses are given only to queries applying to at least two but not more than six individuals. Suppose further that the questioner believes that C = (JOURNALIST AND FEMALE) uniquely identifies Darwin. The minimum query set control would prevent direct questioning about Darwin. The dialogue below shows how Darwin's contribution can be deduced by using as tracker the formula T = (JOURNALIST AND NOT FEMALE) = (JOURNALIST AND MALE).

TABLE 1. POLITICAL CONTRIBUTION DATABASE

Name	*Sex*	*Occupation*	*Contribution* ($)
Abel	M	Journalist	3000
Baker	M	Journalist	500
Charles	M	Entrepreneur	1
Darwin	F	Journalist	5000
Engel	F	Scientist	1000
Fenwick	M	Scientist	20000
Gary	F	Doctor	2000
Hart	M	Lawyer	10000

Q: How many persons are JOURNALIST?
A: 3
Q: How many persons are JOURNALIST AND MALE?
A: 2

By subtracting the second response from the first, the questioner verifies that (JOURNALIST AND FEMALE) identifies only one individual (Darwin). The questioner continues with

Q: What was the total of contributions by all persons who are JOURNALIST?
A: $8500
Q: What was the total of contributions by all persons who are JOURNALIST AND MALE?
A: $3500

Since Darwin is the only female journalist, her contribution is the difference between the response of the second query and the response of the first ($5000).

It might seem that the effort to compromise the entire database is very high because the questioner would have to know identifying characteristics of each individual in order to construct a tracker for that individual. However, if a questioner can find *any* formula whose query set contains at least $2k$ but not more than $n - 2k$ records, he can use that formula as a "general tracker" to determine the answer to any (unanswerable) query of the database [DENN79a]. Schlörer has shown that often more than 99 percent of the possible formulas will be general trackers, and that the effort to retrieve data using trackers is usually quite low [SCHL79]. It is possible to find a tracker with at most $\log_2 S$ queries, where S is the number of possible distinct configurations of characteristics [DENN79b].

Tracker-based compromises employ groups of records having high overlaps. The compromise illustrated above works precisely because Darwin is the only JOURNALIST excluded from the group (JOURNALIST AND MALE). To protect against trackers, one might consider a minimum overlap control:

> Do not respond to a query that has more than a predetermined number of records in common with every prior query.

Such a control is obviously infeasible: before responding, the query program would have to compare the latest query group against every previous one. But even if feasible, this control can be subverted in databases where each confidential value appears just once [DOBK79]. This dialogue illustrates such compromise using queries that overlap by one record:

> Q: What was the largest of the contributions by persons who are JOURNALIST?
> A: $5000.
> Q: What was the largest of the contributions by persons who are FEMALE?
> A: $5000.

Because each contribution is unique, there can be only one person who is JOURNALIST, FEMALE, and contributed $5000 (Darwin). Indeed, the same compromise works even if the query program occasionally returns the contribution of the wrong person [DEMI77]:

> Q: What was the smallest of the contributions by persons who are JOURNALIST?
> A: $5000 (lying).
> Q: What was the largest of the contributions by persons who are FEMALE?
> A: $5000 (truthfully).

A minimum overlap control may also be subverted by solving a linear system of equations for an unknown data value [DOBK79, SCHW79].

These examples illustrate compromises that use the combinatorial principle of inclusion and exclusion to isolate a record. A design that can prevent this is a *partitioned database* [YU77]:

> Store the records in groups, each containing at least some predetermined number of records. Queries may apply to any set of groups, but never to subsets of records within any group.

With this control, attacks based on inclusion and exclusion can, at best, isolate one of the groups—but queries for single groups are allowed. "Microaggregation" is a form of partitioning: groups of individuals are aggregated into synthetic "average individuals" and statistics are computed for the synthetic individuals rather than the real ones [FEIG70]. The partitioned database has two practical limitations that can be quite severe. First, the legitimate free flow of statistical summaries can be inhibited by excessively large groups or by ill-considered groupings. Second, forming and reforming groups as records are inserted, updated, and deleted from the database leads to excessive bookkeeping.

Rounding and Error Inoculation

The second class of inference controls is based on distorting the responses. These are usually called *rounding controls* because the exact answer to a query is perturbed by a small amount before it is reported to the questioner.

Rounding by adding a zero-mean random value is insecure, since the correct answer can be deduced by averaging a sufficient number of responses to the same query.

Rounding by adding a pseudorandom value that depends on the data is preferable, since a given query always returns the same response. Although reasonably effective, this method can sometimes be subverted with trackers [SCHL77], by adding dummy records to the database [KARP70], or simply by comparing the responses to several queries [ACHU78].

A perturbation can also be achieved with *error inoculation*: the value in a record is randomly perturbed or replaced by another value before it is used to compute a statistic [BECK79, BORU71, CAMP77]. Data could be modified before being stored in a record, in which case the original data are discarded. This can have serious adverse consequences if the correct data are supposed to be available for authorized users of the database. Alternatively, a "perturbation factor" can be stored permanently in the record along with the original data; it is applied when the data are used in a query.

Like rounding, error inoculation reduces the quality of the statistics released. To prevent compromise, large errors may have to be introduced into the data.

A variation of this approach, which may yield more accurate statistics, is *data swapping*: the values of fields of records are exchanged so that all i-order statistics (which involve i attributes) are preserved for $i \le d$ and some d [DALE78, SCHL78]. Even if a questioner succeeds in isolating a value, he has no way of knowing with which individual it is actually associated. The problem with the approach is the difficulty of finding sets of records whose values can be swapped. It remains to be seen whether this control can be cost effective.

Random Samples

The third group of controls is based on applying queries not to the entire database but to a "random sample"—a subset of records chosen at random. This group is potentially the most effective because it deprives the questioner of control over the contents of query sets. The 1960 U.S. Census, for example, was distributed on tape as a random sample of one record in one thousand; each sample record contained no name, and it specified location only by size of city in one of nine geographic regions [HANS71]. The cleverest snooper would have at best 1/1000 chance of associating a given sample record with the right individual. These odds are too poor for compromise of the sample to be worthwhile.

Commercial data management systems now permit the construction of small- to medium-scale databases which change constantly through insertion, deletion, and modification of records. A small random sample would be useless because it would not be statistically significant and because it would not represent the current state of the database. For these reasons random samples have been ignored as an inference control in such databases.

However, when combined with a minimum query set control, random sample queries can be an effective defense [DENN79c]. The scheme works as follows. As it locates each record satisfying a given formula C, the query program determines whether or not to keep that record for the "sampled query set." This determination should ideally be pseudorandom so that the same sampled query set is computed any time the given formula C is presented. Each queried record is selected independently for the sampled query set with a given, fixed probability. Statistics then are computed from the sampled query set. A minimum query set control inhibits the response if the true query set's size is too small or too large.

With a simulated database and $p = 0.9375$, this method estimated counts (and sums) of answerable query sets to within 1 percent of their true values [DENN79c]. However, the estimates of counts (and sums) estimated with trackers contained errors of several hundred percent; this is because the questioner must estimate small counts (or sums) by subtracting large counts (and sums). (An illustration: The questioner tries to find the count for C by subtracting the response 100 from the response 101, both of which have error ± 1; the difference, 1, has error ± 2.)

Limitations of Inference Controls

Because queries inevitably carry some information out of the database, one cannot reasonably hope to design a system that is impossible to compromise. The objective of research on inference controls is to discover how much computational work, measured

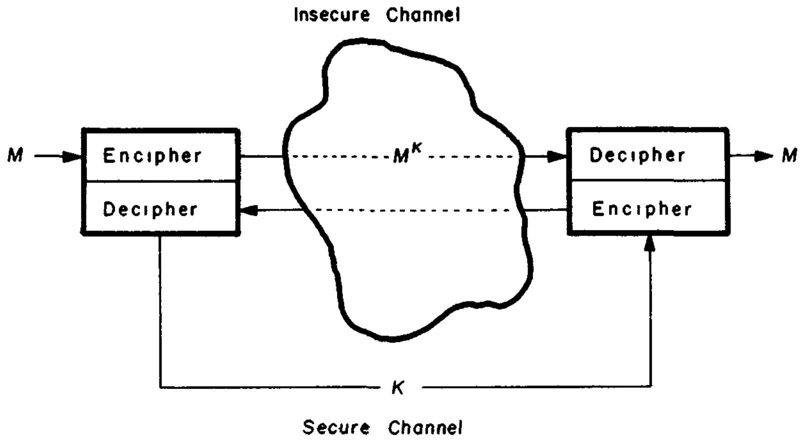


FIGURE 6. Encryption using one key (traditional view).

by computer time or by dollars spent, would be required to compromise a particular system. This would enable designers either to raise the cost of compromise beyond a specific threshold, or to declare that the desired level of security could not be implemented with the given cost constraints.

The query functions of many commercial database systems seem to release much more information about confidential records than many people have suspected. Without proper controls, databases subject to simple compromises will be the rule rather than the exception. When combined with minimum query set restrictions, random sample queries appear to offer the best defense.

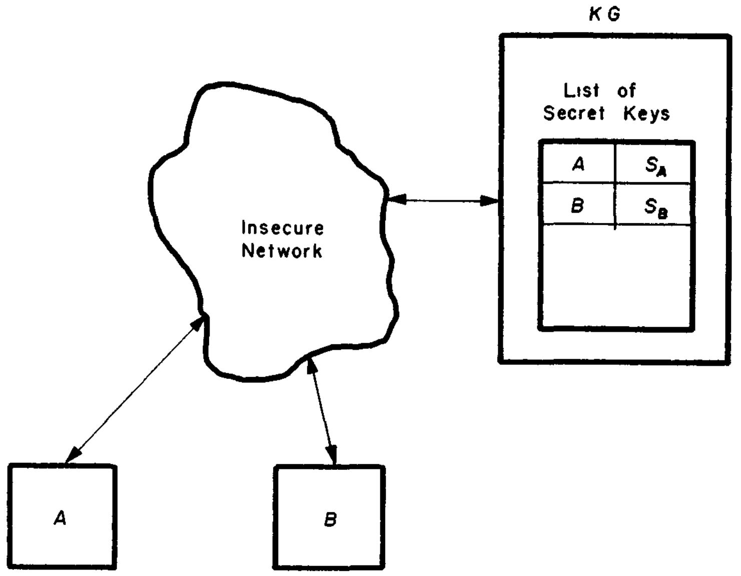
A final defense against snooping is *threat monitoring*—inspection of logs or audit trails for unusual patterns of queries, especially many queries for the same records [HOFF70]. Although it does not attempt to control the flow of information through query programs, monitoring threatens exposure of illegal activity.

4. CRYPTOGRAPHIC CONTROLS

Access, flow, and inference controls may not prevent accidental or malicious disclosures of sensitive data. None of these controls helps if an operator leaves a listing of the password file on a desk, if confidential data are moved off-line during backup or maintenance, if transmissions are tapped or played back, or if hardware and software are faulty. Encryption is a common safeguard for data stored in, or in transit through, media whose security cannot be guaranteed by these other controls. With the help of a secret key (K) the sensitive plaintext (M) is scrambled into unintelligible ciphertext (M^K) before being put in the insecure medium.

In a traditional cryptosystem, illustrated in Figure 6, there is a slow-speed secure channel by which the sender can inform the receiver of the key K used to encode the message. The message itself, transmitted at high speed through the insecure medium, is secure as long as the key is secret. Simmons calls this *symmetric encryption* because the same key is used at both ends of the channel [SIMM79]. The code is broken if an intruder can deduce the key by analyzing the ciphertexts. Keys are changed regularly, usually more often than the time in which the cleverest intruder is believed capable of locating the key systematically.

The code will be unbreakable if the key is a sequence of random bits as long as the message (pseudorandom generation will not do!); each key bit specifies whether the corresponding message bit is complemented or not. With such a key, called a *one-time pad*, each bit of ciphertext is purely random and uncorrelated with all other bits of ciphertext. Practical cryptosystems are based on keys that are much shorter than the messages; because the intruder may know the enciphering and deciphering algorithms, the security of these cryptosystems depends on the secrecy of the keys and the computational difficulty



A to KG: $(A, (I, B)^{S_A})$

KG to A: $(I, K, (K, A)^{S_B})^{S_A}$

A to B: $(K, A)^{S_B}$

FIGURE 7. Protocol for allocating A and B a common key K for exchanging messages.

of inverting the enciphering algorithms. Overviews of cryptosystems are given by Diffie and Hellman [DIFF76], Gardner [GARD77], Hoffman [HOFF77], Konheim [KONH78], Lempel [LEMP79], and Simmons [SIMM79]. Fascinating accounts of codes and their breakings have been written by Kahn [KAHN67].

It is reasonable to suppose that military and diplomatic experts have secure channels for exchanging encryption keys—for example, secret couriers. Therefore the security of the traditional cryptosystem is properly measured as the work required for an intruder to invert the code through cryptanalysis.

With computer networks it is no longer reasonable to suppose that individual senders and receivers have secure means of exchanging secret keys. Needham and Schroeder [NEED78] and Popek [Pope78a] have outlined methods, called "protocols," for simulating a secure key-exchange channel. Figure 7 gives the central idea of the protocol suggested by Needham and Schroeder. A sending computer A and a receiving computer B seek a secret key K from a secure key-generating facility (KG). The computer KG contains a list of special secret keys, one assigned to each computer of the network; thus A and KG can exchange secret messages encrypted with key S_A known only to the two. Having decided to send a message to B, A first transmits the message $(A, (I, B)^{S_A})$ to KG, wherein I is message identifier chosen arbitrarily by A. Since A's name is a plaintext prefix of this message, KG can locate A's secret key, S_A, and decipher the component $(I, B)^{S_A}$. Then KG generates a key K and responds to A with the message $(I, K, (K, A)^{S_B})^{S_A}$. Only A can decode this message and obtain the newly generated key K and the embedded ciphertext $T = (K, A)^{S_B}$; A can also check that this message is a unique response from KG by verifying that I is one of his own (recent) message identifiers. Then A forwards the ciphertext T to B, who is the only one capable of decoding it. After these exchanges, both A and B have the key K and can begin transmitting directly to each other in code.

The security of this system clearly depends on the security of the key-generating facility. Both *A* and *B* must trust *KG* to generate a unique key *K*, to keep all keys secret, and to not monitor any of their private transmissions.

This example illustrates why *key management* is essential to secure cryptographic control in computer networks. The security of these cryptosystems is often less dependent on the indecipherability of the code than it is on the ability to secure the keys. Saltzer has argued that our demonstrated inability to protect passwords in file systems suggests that many cryptosystems will be easily broken not by cryptanalysis but by elementary attacks on the key manager. (See SALT78 and also DENN79d, EHRS78, GAIN78, MATY78, MORR78, and POPE78a.)

The Data Encryption Standard (DES)

In 1977 the National Bureau of Standards announced a standard encryption algorithm (DES) to be used in unclassified U.S. Government communications [NBS77]. The algorithm was developed by IBM, which offers products that use DES [EHRS78, LENN78, MATY78]. Each 64-bit block of plaintext undergoes a complex transformation comprising 16 levels of substitutions and permutations, all controlled by a 56-bit key. The algorithm can be implemented cheaply as an LSI chip, which would allow it to operate at a high data rate.

The DES can be used as in Figure 6, providing "end-to-end encryption" on the channel between the sender *A* and receiver *B*. A user can also use DES to encipher files for storage in removable or insecure media. However, the data must usually be deciphered for processing; other mechanisms, such as access and flow controls, are needed to protect the data while they are plaintext. Rivest has studied cryptosystems which allow a limited number of operations which can be performed directly on the ciphertext; however, these systems must exclude compare operations if they are to be highly secure [RIVE78b].

The DES can also be used as a *one-way cipher* to secure files containing passwords [EVAN74, MORR78, WILK68]. Each password *X* is used as the key to encipher a predetermined plaintext *C*; the resulting ciphertext C^X is placed in the password file along with the name of the password's owner. When a user *N* presents a password *X*, the password is accepted only if C^X is already in the password file with name *N*. Because no password is actually stored as plaintext in the system, passwords are protected even if the file is disclosed.

The DES is not universally regarded as a highly secure cryptosystem. Diffie and Hellman argue that it is possible for about $20 million to build a highly parallel computer that will locate a key by exhaustive search in about 12 hours [DIFF77]. At the 1978 National Computer Conference, Hellman showed how to use about $4 million of conventional equipment with a heuristic search to find a key within 24 hours. Diffie and Hellman maintain that a 128-bit key (not the 56 bits in the standard) would make the DES virtually unbreakable. IBM maintains that two DES chips in series can, with double encryption, simulate a single DES chip with a 128-bit key [HOFF77]. IBM also maintains that even with 56-bit keys DES is not likely to be the weak link in the security chain.

Public-Key Encryption

In 1976 Diffie and Hellman proposed a new kind of cryptosystem, which they called *public-key encryption* [DIFF76]. Each user has a matched pair of keys, the "public key" *P* and the "secret key" *S*. Each user publishes his public key to everyone through a directory but reveals his secret key to no one. As with the DES, the encryption algorithm need not be secret.

Enciphering a message *M* with the public key *P* gives a ciphertext M^P which can be sent to the owner of the public key. The recipient deciphers the ciphertext using his secret key *S* to retrieve the message: $(M^P)^S = M$. Since *P* and *S* are a matched pair, no one but the owner of the public key can decipher M^P. The operations of *P* and *S* are commutative; that is, $(M^P)^S = (M^S)^P = M$. It is infeasible to compute one key given the other. Figure 8 illustrates communication between two computers by this scheme. Simmons calls this *asymmetric encryption* because different keys are used at the ends of the channel [SIMM79]. Examples of specific algorithms are in DIFF76, HELL78, KONH78, LEMP79, MERK78, and RIVE78a.

Public-key cryptosystems also present an easy solution to "digital signatures," the problem of proving that a particular message was in fact transmitted by the person claiming to have transmitted it. To create a signature, one enciphers a predetermined message M with his secret key S, offering M^S as the signature. Anyone challenging the signature need only apply the public key P of the purported signer, for only then will $(M^S)^P = M$. (See DIFF76, RIVE78a, KONH78, and LEMP79.) Digital signatures can also be implemented with single-key cryptosystems, but the solution is much less elegant [NEED78, RABI77].

A result by Shamir, Rivest, and Adleman suggests that a modification of public-key encryption could also be an approximate one-time pad [SHAM79]. Users A and B each select a matched pair of keys but keep both secret. Then A sends the enciphered message M^{S_A} to B, who then enciphers it and returns $(M^{S_B})^{S_A} = (M^{S_A})^{S_B}$ to A. Then A applies his second key, P_A, to obtain $M^{S_B} = ((M^{S_B})^{S_A})^{P_A}$, which he returns to B. Finally, B obtains the message M by applying his P_B, since $M = (M^{S_B})^{P_B}$. This is not a true one-time pad because three messages are actually sent using the four keys.

Its proponents argue that public-key encryption is more secure than DES because no user need rely on the security of the computer network to safeguard his secret key. Indeed, a user's local, personal computer can interface with the network through the encryption device as in Figure 8, and his secret key can be engraved electronically into a memory chip that can be plugged into the encryption device; he could then guard his encryption key to the same extent as any other key in his possession [DENN79d].

It is true that the public keys can be distributed by a public directory without endangering the secret keys. However, a user still needs assurances that the key received from the (purported) public directory is in fact the public key of the requested individual. Confidence in correct distribution of public keys can be increased if the public directory signs its responses [KONF78, NEED78].

5. SUMMARY

The four kinds of internal security controls—access, flow, inference, and cryptographic—complement each other. No one of them can solve the problems handled by the other three.

Access controls operate at the external interface, verifying that an individual attempting to use the system is authentic, and internally, verifying that each running program generates references only to authorized segments of memory. The ideal access control operates each program in a domain having the minimum privilege required for the immediate task. The principle of minimum privilege contributes greatly to security by protecting against "trojan horses," and to reliability by minimizing the extent of possible damage by a malfunctioning program.

Flow controls regulate the dissemination or copying of information by prohibiting derived data from having lower confiden-

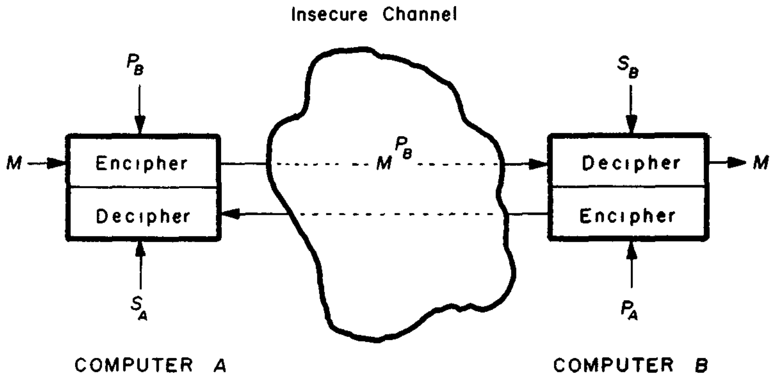


FIGURE 8. Public-key encryption.

tiality than the original. The higher the data's confidentiality, the stricter the rules on their dissemination. When applied to program input-to-output flow, these controls offer a partial solution to the "confinement problem."

Inference controls prevent "leakage" through programs that produce summaries of groups of confidential records. They reduce the risk that by correlating the responses from many summaries, a user can deduce the confidential values denied him by access and flow controls.

Cryptographic controls protect information stored or transmitted in insecure media. The data encryption standard (DES) is efficient and economical, though it has been criticized as being breakable and overly dependent on secure key management. Public-key encryption does not rely on any central manager for safeguarding secret keys, though it requires secure distribution of public keys.

All these controls are subject to practical (and sometimes theoretical) limitations which keep them from achieving their objectives under all conditions. No mechanism is perfectly secure. A good mechanism reduces the risk of compromise to an acceptable level.

ACKNOWLEDGMENTS

A preliminary version of this paper appeared under the title "The Limits of Data Security," in the pilot issue of *ABACUS*, the *Scientific-American*-style magazine for the computing community undertaken by the American Federation of Information Processing Societies (AFIPS). We are grateful to AFIPS for permission to use parts of the pilot article in this one.

We are grateful to William E Boggs, whose journalistic genius considerably improved most of the draft We are also grateful to Adele Goldberg and the referees for their comments and suggestions.

REFERENCES

ACHU78 ACHUGBUE, J.O., AND CHIN, F.Y. *Output perturbation for protection of statistical data bases*, Dept. Computing Science, Univ. Alberta, Edmonton, Canada, Jan. 1978.

ANDE72 ANDERSON, J.P. "Information security in a multi-user computer environment," in *Advances in computers*, Vol. 12, Morris Rubinoff (Ed.), Academic Press, New York, 1972.

BECK79 BECK, L.L. *A security mechanism for statistical databases*, Dept. Computer Science and Engineering, Southern Methodist Univ., Dallas, Tex., Jan. 1979.

BONC77 BONCZEK, R.H., CASH, J.I., AND WHINSTON, A.B. "A transformational grammar-based query processor for access control in a planning system," *ACM Trans. Database Syst.* **2**, 4 (Dec. 1977), 326-338.

BORU71 BORUCH, R.F. "Maintaining confidentiality in educational research: A systematic analysis," *Am. Psychol.* **26** (1971), 413-430.

CAMP77 CAMPBELL, D.T., ET AL. "Confidentiality-preserving modes of access to files and to interfile exchange for useful statistical analysis," *Eval. Q.* **1**, 2 (May 1977), 269-299.

COHE75 COHEN, E., AND JEFFERSON, D. "Protection in the HYDRA operating system," in *Proc. 5th Symp. Operating Systems Principles*, (special issue) *Oper. Syst. Rev.* (ACM) **9**, 5 (Nov. 1975), 141-160.

COHE77 COHEN, E. "Information transmission in computational systems," *Proc. 6th Symp. Operating Systems Principles*, (special issue) *Oper. Syst. Rev.* (ACM) **11**, 5 (Nov. 1977), 133-139.

COHE78 COHEN, E. "Information transmission in sequential programs," in *Foundations of secure computation*, R.A. DeMillo et al. (Eds.), Academic Press, New York, 1978, pp. 297-335.

CONW72 CONWAY, R.W., MAXWELL, W.L., AND MORGAN, H.L. "On the implementation of security measures in information systems," *Commun. ACM* **15**, 4 (April 1972), 211-220.

DALE78 DALENIUS, T., AND REISS, S.P. *Data-swapping—A technique for disclosure control*, Computer Science, Brown Univ., Providence, R.I., (1978).

DEMI77 DEMILLO, R.A., DOBKIN, D., AND LIPTON, R.J. "Even data bases that lie can be compromised," *IEEE Trans. Software Eng.* **SE-4**, 1 (Jan. 1977) 73-75.

DEMI78 DEMILLO, R.A., DOBKIN, D.P., JONES, A.K., AND LIPTON, R.J. (Eds.) *Foundations of secure computation*, Academic Press, New York, 1978.

DENN71 DENNING, P.J. "Third generation computer systems," *Comput. Surv.* **3**, 4 (Dec. 1971), 175-216.

DENN76a DENNING, D.E. "A lattice model of secure information flow," *Commun. ACM* **19**, 5 (May 1976), 236-243.

DENN76b DENNING, P.J. "Fault tolerant operating systems," *Comput. Surv.* **8**, 4 (Dec. 1976), 359-389.

DENN77 DENNING, D.E., AND DENNING, P.J. "Certification of programs for secure information flow," *Commun. ACM* **20**, 7 (July 1977), 504-513.

DENN79a DENNING, D.E., DENNING, P.J., AND SCHWARTZ, M.D. "The tracker: A threat to statistical database security," *ACM Trans. Database Syst.* **4**, 1 (March 1979), 76-96.

DENN79b DENNING, D.E., AND SCHLORER, J. *A fast procedure for finding a tracker in a statistical database*, Computer Science Dept., Purdue Univ., W. Lafayette, Ind. and Inst. Medizinische Statistik und Dokumentation, Univ Giessen, W. Germany, Feb. 1979.

DENN79c DENNING, D.E. *Securing databases under random sample queries*, Computer Science Dept., Purdue Univ., W. Lafayette, Ind., April 1979.

DENN79d DENNING, D.E "Secure personal computing in an insecure network," *Commun. ACM* **22,** 8 (Aug 1979), 476-482.

DENV66 DENNIS, J.B., AND VAN HORN, E.C. "Programming semantics for multiprogrammed computations," *Commun. ACM* **9,** 3 (March 1966), 143-155.

DIFF76 DIFFIE, W., AND HELLMAN, M. "New directions in cryptography," *IEEE Trans. Inf Theory* **IT-22,** 6 (Nov. 1976), 644-654.

DIFF77 DIFFIE, W., AND HELLMAN, M E "Exhaustive cryptanalysis of the NBS Data Encryption Standard," *Computer* **10,** 6 (June 1977), 74-84.

DOBK79 DOBKIN, D., JONES, A.K., AND LIPTON, R.J. "Secure databases· Protection against user inference," *ACM Trans. Database Syst.* **4,** 1 (March 1979), 97-106.

EHRS78 EHRSAM, W.F , MATYAS, S.M., MEYER, C.H., AND TUCHMAN, W.L. "A cryptographic key management scheme for implementing the Data Encryption Standard," *IBM Syst. J.* **17,** 2 (1978), 106-125.

ENGL74 ENGLAND, D.M. "Capability concept mechanism and structure in system 250," in *Proc. Int. Workshop on Protection in Operating Systems,* Inst. Recherche d'Informatique et d'Automatique, Rocquencourt, Le Chesnay, France, Aug. 1974, pp. 63-82.

EVAN74 EVANS, A. JR., KANTROWITZ, W., AND WEISS, E. "A user authentication scheme not requiring secrecy in the computer," *Commun. ACM* **17,** 8 (Aug. 1974), 437-442.

FABR74 FABRY, R.S. "Capability-based addressing," *Commun. ACM* **17,** 7 (July 1974), 403-412.

FAGI78 FAGIN, R. "On an authorization mechanism," *ACM Trans. Database Syst.* **3,** 3 (Sept. 1978), 310-319.

FEIG70 FEIGE, E.L., AND WATTS, H.W. "Protection of privacy through microaggregation," in *Data bases, computer, and the social sciences,* R.L. Bisco (Ed.), Wiley-Interscience, New York, 1970.

FENT74 FENTON, J.S. "Memoryless subsystems," *Comput. J* **17,** 2 (May 1974). 143-147.

FURT78 FURTEK, F. "Constraints and compromise," in *Foundations of secure computation,* R.A. DeMillo et al. (Eds.), Academic Press, New York, 1978, pp. 189-204.

GAIN78 GAINES, R.S., AND SHAPIRO, N.Z. "Some security principles and their application to computer security," in *Foundations of secure computation,* R.A. DeMillo et al. (Eds.), Academic Press, New York, 1978, pp 223-236; also in *Oper. Syst. Rev.* **12,** 3 (July 1978), 19-28.

GARD77 GARDNER, M. "Mathematical games," *Sci. Am.* **237,** 2 (Aug. 1977), 120-124.

GEHR79 GEHRINGER, E. "Functionality and performance in capability-based operating systems," Ph.D. Thesis, Computer Science Dept., Purdue Univ., W. Lafayette, Ind , May 1979.

GRAH72 GRAHAM, G S , AND DENNING, P.J. "Protection—Principles and practice," in *Proc. 1972 AFIPS Spring Jt. Computer Conf.,* Vol. 40, AFIPS Press, Montvale, N.J., pp. 417-429.

GRIF76 GRIFFITHS, P.P., AND WADE, B.W. "An authorization mechanism for a relational database system," *ACM Trans. Database Syst* **1,** 3 (Sept. 1976), 242-255.

HANS71 HANSEN, M.H. "Insuring confidentiality of individual records in data storage and retrieval for statistical purposes," in *Proc. 1971 AFIPS Fall Jt. Computer Conf.,* Vol. 39, AFIPS Press, Montvale, N.J., pp. 579-585.

HARR76 HARRISON, M.A., RUZZO, W.L., AND ULMAN, J.D. "Protection in operating systems," *Commun. ACM* **19,** 8 (Aug. 1976), 461-471.

HART76 HARTSON, H R., AND HSIAO, D.K. "Full protection specifications in the semantic model for database protection languages," in *Proc. 1976 ACM Annual Conf.,* Oct. 1976, pp. 90-95.

HELL78 HELLMAN, M.E. "Security in communications networks," in *Proc. AFIPS 1978 Nat. Computer Conf.,* Vol. 47, AFIPS Press, Montvale, N.J., 1978, pp. 1131-1134.

HOFF70 HOFFMAN, L.J., AND MILLER, W.F. "Getting a personal dossier from a statistical data bank," *Datamation* **16,** 5 (May 1970), 74-75.

HOFF77 HOFFMAN, L.J. *Modern methods for computer security and privacy,* Prentice-Hall, Englewood Cliffs, N.J., 1977.

HSIA78 HSIAO, D.K., KERR, D.S., AND MADNICK, S.E. "Privacy and security of data communications and data bases," in *Proc. Int Conf. Very Large Data Bases,* Sept. 1978.

JONE76 JONES, A.K., AND LISKOV, B.H. "A language extension mechanism for controlling access to shared data," in *Proc. 2nd Int. Conf. Software Engineering,* 1976, pp. 62-68.

KAHN67 KAHN, D. *The codebreakers,* Macmillan Co., New York, 1967.

KARP70 KARPINSKI, R.H. "Reply to Hoffman and Shaw," *Datamation* **16,** 10 (Oct. 1970), 11.

KONF78 KONFELDER, L.M. "A method for certification," Lab. Computer Science, MIT, Cambridge, Mass., May 1978.

KONH78 KONHEIM, A.G. "Cryptographic methods for data protection," Res. Rep. RC 7026 (#30100), IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., March 1978.

LAMP73 LAMPSON, B.W. "A note on the confinement problem," *Commun. ACM* **16,** 10 (Oct 1973), 613-615.

LEMP79 LEMPEL, A. "Cryptography in transition," to appear in *Comput. Surv.* **11,** 4 (Dec. 1979).

LENN78 LENNON, R.E. "Cryptography architecture for information security," *IBM Syst. J.* **17,** 2 (1978), 138-150.

LIND76 LINDEN, T.A. "Operating system structures to support security and reliable software," *Comput. Surv.* **8,** 4 (Dec. 1976), 409-445.

LIPN75 LIPNER, S.B. "A comment on the confinement problem," in *Proc 5th Symp Operating Systems Principles,* (special issue) *Oper. Syst. Rev.* (ACM) **9,** 5 (Nov. 1975) 192-196.

LIPT78 LIPTON, R.J., AND BUDD, T.A. "On classes of protection systems," in *Foundations of secure computation,* R.A. DeMillo et al. (Eds.), Academic Press, New York, 1978, pp. 281-296.

MADN79 MADNICK, S.E. *Computer security,* Academic Press, New York, 1979.

MATY78 MATYAS, S.M., AND MEYER, C.H. "Generation, distribution, and installation of cryptographic keys," *IBM Syst. J.* **17,** 2 (1978), 126–137.

Merk78 Merkle, R.C., and Hellman, M.E. "Hiding information and signatures in trap door knapsacks," *IEEE Trans. Inf. Theory,* **It-24,** 5 (Sept. 1978), 525–530.

Mill76 Millen, J.K. "Security kernel validation in practice," *Commun. ACM* **19,** 5 (May 1976), 243–250.

Mill78 Millen, J.K. "Constraints and multilevel security," in *Foundations of secure computation,* R.A. DeMillo et al. (Eds.), Academic Press, New York, 1978, pp 205–222.

Morr78 Morris, R., and Thompson, K. *Password security: A case history,* CS-TR-71, Bell Labs, Murray Hill, N.J., April 1978.

Myer78 Myers, G. *Advances in computer architecture,* Wiley, New York, 1978.

NBS77 National Bureau of Standards, Data Encryption Standard, FIPS PUB 46, Jan. 1977.

Need77 Needham, R.M., and Walker, R.D.H. "The Cambridge CAP computer and its protection system," in *Proc. 6th Symp. Operating Systems Principles,* (special issue) *Oper. Syst. Rev.* (ACM) **11,** 5 (Nov. 1977), 1–10.

Need78 Needham, R.M., and Schroeder, M.D. "Using encryption for authentication in large networks of computers," *Commun. ACM* **21,** 12 (Dec. 1978), 993–999.

Neum77 Neumann, P.G., et al. *"A probably secure operating system: The system, its applications, and proofs,"* Project 4332 Final Rep., SRI International, Menlo Park, Calif., Feb. 1977.

Niel76 Nielsen, N.R., Ruder, B., and Brandin, D.H. "Effective safeguards for computer system integrity," in *Proc. AFIPS Nat. Computer Conf. Vol. 45,* AFIPS Press, Montvale, N.J., 1976, pp. 75–84.

Orga72 Organick, E.I. *The multics system: An examination of its structure,* MIT Press, Cambridge, Mass., 1972.

Orga73 Organick, E.I. *Computer system organization: The B5700/B6700 series,* Academic Press, New York, 1973.

Park76 Parker, D.B. *Crime by computer,* Scribner's, New York, 1976.

Pope74 Popek, G.J. "Protection structures," *Computer* **7,** 6 (June 1974), 22–31.

Pope78a Popek, G.J., and Kline, C.S. "Design issues for secure computer networks," in *Operating systems, an advanced course,* R. Bayer, R.M. Graham, and G. Seegmuller (Eds.), Springer-Verlag, New York, 1978.

Pope78b Popek, G.J., and Kline, C.S. "Issues in kernel design, *Proc. AFIPS Nat. Computer Conf.* Vol. 47, AFIPS Press, Montvale, N.J., 1978, pp. 1079–1086.

Pope78c Popek, G.J., and Farber, D.A. "A model for verification of data security in operating systems," *Commun. ACM* **21,** 9 (Sept. 1978), 737–749.

Rabi78 Rabin, M. "Digital signatures using conventional encryption algorithms," in *Foundations of secure computing,* R.A. DeMillo et al. (Eds.), Academic Press, New York, 1978, pp. 155–166.

Rede74 Redell, D.R., and Fabry, R.S. "Selective revocation of capabilities," *Proc. Int. Workshop Protection in Operating Systems,* Inst. Recherche d'Informatique et d'Automatique, Rocquencourt, Le Chesnay, France, Aug. 1974

Reit78 Reitman, R.P., and Andrews, G.R. *Certifying information flow properties of programs: An axiomatic approach,* Syracuse Univ., Syracuse, N.Y., and Cornell Univ., Ithaca, N.Y., 1978.

Rive78a Rivest, R.L., Shamir, A., and Adleman, L. "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM* **21,** 2 (Feb. 1978), 120–126.

Rive78b Rivest, R.L., Adleman, L., and Dertouzos, M.L. "On data banks and privacy homomorphisms," in *Foundations of secure computing,* R.A. DeMillo et al. (Eds.), Academic Press, New York, 1978, pp. 169–179.

RPP77 "The report of the privacy protection study commission," Appendix 5, in *Technology and privacy,* U.S. Gov. Printing Office, Washington, D.C. July 1977.

Salt75 Saltzer, J.H., and Schroeder, M.D. "The Protection of information in computer systems," *Proc IEEE* **63,** 9 (Sept. 1975), 1278–1308.

Salt78 Saltzer, J. "On digital signatures," *Oper. Syst. Rev.* **12,** 2 (April 1978), 12–14.

Schl75 Schlörer, J. "Identification and retrieval of personal records from a statistical data bank," *Methods Inf. Med* **14,** 1 (Jan. 1975), 7–13.

Schl77 Schlörer, J. "Confidentiality and security in statistical data banks," *Proc. Workshop on Data Documentation,* Verlag Dokumentation, 1977, pp. 101–123.

Schl78 Schlörer, J. *Security of statistical databases: Multidimensional transformation,* TB-IMSD 2/78, Inst. Medizinische Statistik und Documentation, Univ. Giessen. W. Germany, 1978.

Schl79 Schlorer, J. *Disclosure from statistical databases: Quantitative aspects of trackers,* Inst. Medizinische Statistik und Dokumentation, Univ. Giessen, W. Germany, March 1979; to appear in *ACM Trans. Database Syst.*

Schr72 Schroeder, M.D., and Saltzer, J.H. "A hardware architecture for implementing protection rings," *Commun ACM* **15,** 3 (March 1972), 157–170.

Schr77 Schroeder, M.D., Clark, D.D., and Saltzer, J.H. "The MULTICS kernel design project," in *Proc. 6th Symp. Operating Systems Principles,* (special issue) *Oper. Syst. Rev.* (ACM) **11,** 5 (Nov. 1977), 43–56.

Schw79 Schwartz, M.D., Denning, D.E., and Denning, P.J. "Linear queries in statistical databases," *ACM Trans Database Syst.* **4,** 2 (June 1979), pp. 156–167

Sham79 Shamir, A., Rivest, R.L., and Adleman, L.M. "Mental poker," Lab. Computer Science, MIT, Cambridge, Mass., Jan. 1979.

Shan77 Shankar, K.S. "The total computer security problem: An overview," *Computer* **10,** 6 (June 1977), 50–73.

Simm79 Simmons, G.J., ""Symmetric and asymmetric encryption," to appear in *Comput. Surv.* **11,** 4 (Dec. 1979).

Snyd77 Snyder, L. "On the synthesis and analysis of protection systems," *Proc. 6th Symp. Operating Systems Principles,* (special issue) *Oper. Syst. Rev.* (ACM) **11,** 5 (Nov. 1977), 141–150.

Ston74 Stonebraker, M., and Wong, E.

"Access control in a relational data base management system by query modification," in *Proc. 1974 ACM Annual Conf.*, pp. 180–186.

TURN76 TURN, R., AND WARE, W.H. "Privacy and security issues in information systems," *IEEE Trans. Comput.* **C-25,** 12 (Dec 1976), 1353–1361.

WALT75 WALTER, K.G., ET AL. "Structured specification of a security kernel," *Proc. Int. Conf. Reliable Software,* (special issue) *SIGPLAN Notices* (ACM) **10,** 6 (June 1975), 285–293.

WEIS69 WEISSMAN, C. "Security controls in the ADEPT-50 time-sharing system," *Proc. 1969 AFIPS* Fall Jt. Computer Conf., Vol. 35, AFIPS Press, Montvale, N.J., pp. 119–133.

WEST72 WESTIN, A.F., AND BAKER, M.A. *Databanks in a free society,* Quadrangle Books, New York, 1972.

WILK68 WILKES, M.V. *Time sharing computing systems,* Elsevier/MacDonald, New York, 1968; 3rd ed., 1975.

YU78 YU, C.T., AND CHIN, F.Y. *A study on the protection of statistical data bases, Proc ACM SIGMOD Int. Conf. Management of Data*, 1977, pp. 169–181.

RECEIVED OCTOBER 1978; FINAL REVISION ACCEPTED MAY 1979