DATA SECURITY

The term *data security* refers to the protection of information against possible violations that can compromise its *secrecy* (or confidentiality), *integrity,* or *availability.* Secrecy is compromised if information is disclosed to users not authorized to access it. Integrity is compromised if information is improperly modified, deleted, or tampered with. Availability is compromised if users are prevented from accessing data for which they have the necessary permissions. This last problem is also known as *denial-of-service.*

The increasing development of information technology in the past few years has led to the widespread use of computer systems that store and manipulate information and greatly increased the availability and the processing and storage power of information systems. The problem of protecting information exists because this information has to be managed. However, as technology advances and information management systems become even more powerful, the problem of enforcing information security becomes more critical. There are serious new security threats, and the potential damage caused by violations rises. Organizations more than ever today depend on the information they manage. A violation to the security of the information may jeopardize the whole working system and cause serious damage. Hospitals, banks, public administrations, private organizations, all depend on the accuracy, availability, and confidentiality of the information they manage. Just imagine what could happen, for instance, if a patient's data were improperly modified, were not available to the doctors because of a violation blocking access to the resources, or were disclosed to the public domain.

The threats to security to which information is exposed are many. Threats can be *nonfraudulent* or *fraudulent.* The first category comprises of all the threats resulting in nonintentional violations, such as natural disasters, errors or bugs in hardware or software, and human errors. The second category comprises all of such threats that can be attributed to authorized users (*insiders*) who misuse their privileges and authority, or external users (*intruders*) who improperly get access to a system and its resources. Ensuring protection against these threats requires the application of different protection measures. This article focuses mainly on the protection of information against possible violations by users, insiders, or intruders. The following services are crucial to the protection of data within this context (12):

1. *Identification and Authentication.* It provides the system with the ability of identifying its users and confirming their identity.

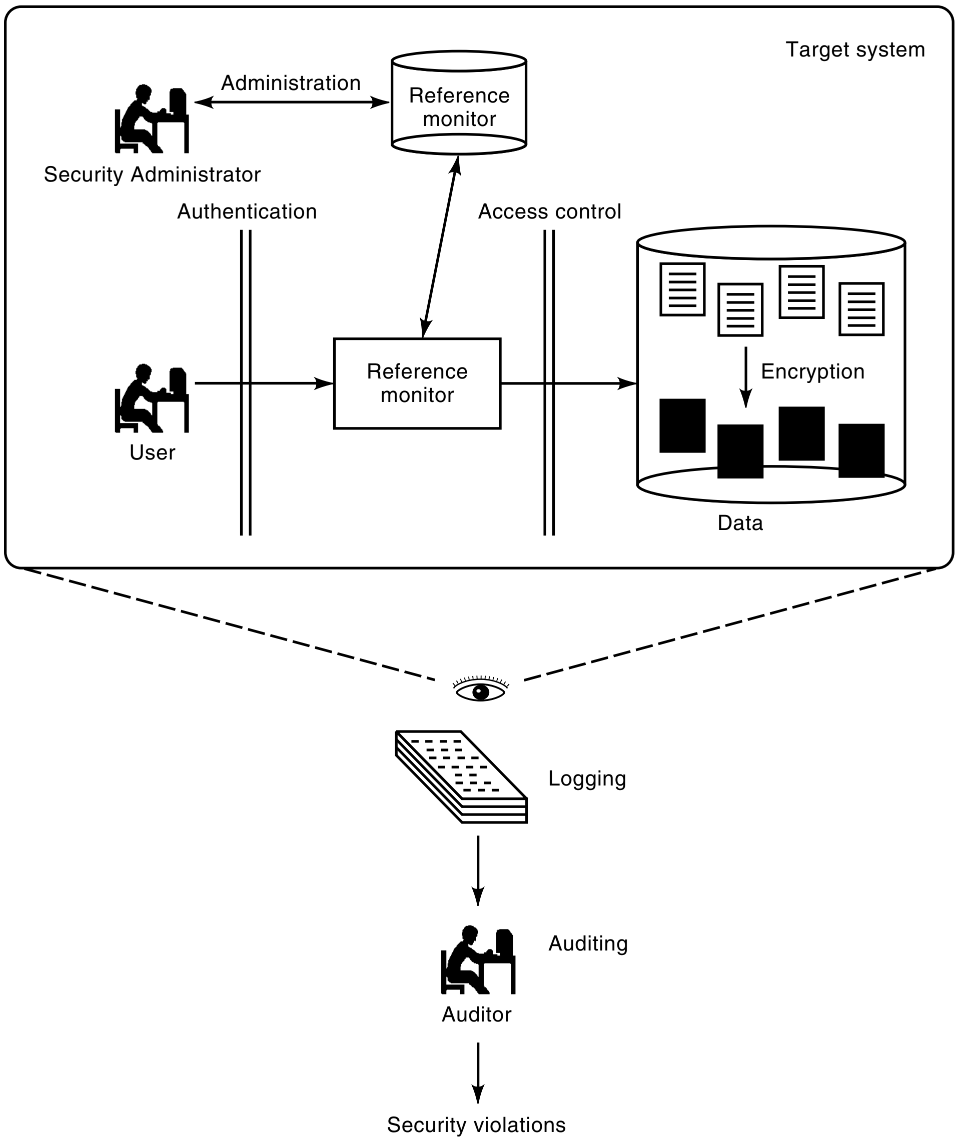

Figure 1. Authentication, access control, audit, and encryption.

2. *Access Control.* It evaluates access requests to the resources by the authenticated users, and based on some access rules, it determines whether they must be granted or denied.
3. *Audit.* It provides a post facto evaluation of the requests and the accesses occurred to determine whether violations have occurred or have been attempted.
4. *Encryption.* It ensures that any data stored in the system or sent over the network can be deciphered only by the intended recipient. In network communication, encryption can also be used to ensure the authenticity of the information transmitted and of the parties involved in the communication.

Figure 1 illustrates the position of these services within the system working. Their treatment is the focus of this chapter.

IDENTIFICATION AND AUTHENTICATION

Authentication is the process of certifying the identity of one party to another. In the most basic form, authentication certifies the identity of a human user to the computer system. Authentication is a prerequisite for a correct access control, since the correctness of the access control relies on the correctness of the identity of the subject requesting access. Good authentication is also important for *accountability,* whereby users can be retained accountable for the actions accomplished when connected to the system. In the authentication process we can generally distinguish an *identification* phase, where users declare their identity to the computer and submit a proof for it, and an actual *authentication* phase, where the declared identity and the submitted proof are evaluated. Authentication of a user to a computer can be based on

- *something the user knows,* such as a password
- *something the user possesses,* such as a magnetic card

- *something the user is or does,* such as his/her physical characteristics

or a combination of the above.

Authentication Based on Knowledge

The most common technique based on user's knowledge uses secret keywords, named *passwords.* A password, known only to the user and the system, proves the identity of the user to the system. Users wishing to log into the computer enter their identity (*login*) and submit a secret keyword (*password*) as a proof of their identity. Passwords are the most commonly used authentication technique for controlling accesses to computers. The wide use of this technique is due to the fact that is very simple, cheap, and easily enforceable. A drawback is that this technique is quite vulnerable. Passwords can often be easily *guessed, snooped* by people observing the legitimate user keying it in, *sniffed* during transmission, or *spoofed* by attackers impersonating login interfaces. By getting a user's password an attacker can then "impersonate" this user and enter the system. An important aspect necessary to limit the vulnerability of passwords is a good password management. Often passwords are vulnerable because users do not put enough care in their management: They do not change their passwords for a long time; share their passwords with friends or colleagues; choose weak passwords that can be easily guessed, such as common words, the name or birthdate of a relative, or the name of their pet, simply because they are easy to remember; use the same password on different machines; or write passwords down over pieces of papers to make sure they do not forget them. A good password management requires users to change their password regularly, choose passwords that are not easy to guess, and keep the password private. Unfortunately, these practices are not always followed.

Having to remember passwords can become a burden for a user, especially when multiple passwords, necessary to access different accounts, need to be remembered. To make this task easier, users often end up falling in some of the bad habits listed above, thus making the attackers task easier as well. To avoid this problem, many systems enforce automatic controls regulating the specification and use of passwords. For instance, it is possible to enforce restrictions on the minimum number of digits a password must have, possibly requiring the use of both alphanumeric and nonalphanumeric characters. Also often systems check passwords against language dictionaries and reject passwords corresponding to words of the language (which would be easily retrieved by attackers enforcing dictionary attacks). It is also possible to associate a maximum lifetime to passwords, and require users to change their password when it expires. Passwords that remain unchanged for a long time are more vulnerable and, if guessed and never changed, would allow attackers to freely access the system impersonating the legitimate users. A history log can also be kept to make sure users do not just pretend to change password while reusing instead the same one. Sometimes a minimum lifetime can also be associated with passwords. The reason for this is to avoid users to actually reuse the same password over and over again despite the presence of lifetime and history controls. Without a minimum lifetime a user required to change password but unwilling to do so could simply change it and then change it back right away to the old value. A minimum lifetime restriction would forbid this kind of operation.

Authentication Based on Possession

In this category, also called *token*-based, there are all the techniques that require users to present a *token* as a proof of their identity. A token is a creditcard-sized device storing some information establishing and proving the token's identity. The simplest form of token is a memory card containing magnetically recorded information, which can be read by an appropriate card reader. Essentially this technique authenticates the validity of the token, not of the user: Possession of the token establishes identity for the user. The main weakness of such an approach is that tokens can be forged, lost, or stolen. To limit the risk of security breaches due to such occurrences, often memory cards are used together with a *personal identification number* (PIN), generally composed of four numeric digits, that works like a password. To enter the system, a user needs both to present the token and to enter the PIN. Like passwords, PINs can be guessed or spoofed, thus possibly compromising authentication, since an attacker possessing the token and knowing the PIN will be able to impersonate the legitimate user and enter the system. To limit the vulnerability from attackers possessing a token and trying to guess the corresponding PIN to enter the system, often the authentication server terminates the authentication process, and possibly seizes the card, upon submission of few bad tries for a PIN. Like passwords, tokens can be shared among users, thus compromising accountability. Unlike with passwords, however, since possession of the token is necessary to enter the system, only one user at the time is able to enter the system. Memory cards are very simple and do not have any processing power. They cannot therefore perform any check on the PIN or encrypt it for transmission. This requires sending the PIN to the authentication server in the clear, exposing the PIN to sniffing attacks and requiring trust in the authentication server. ATM (automatic teller machine) cards are provided with processing power that allows the checking and encrypting of the PIN before its transmission to the authentication server.

In token devices provided with processing capabilities, authentication is generally based on a challenge-response handshake. The authentication server generates a challenge that is keyed into the token by the user. The token computes a response by applying a cryptographic algorithm to the secret key, the PIN, and the challenge and returns it to the user, who enters this response into the workstation interfacing the authentication server. In some cases the workstation can directly interface the token, thus eliminating the need for the user to type in the challenge and the response. Smart cards are sophisticated token devices that have both processing power and direct connection to the system. Each smart card has a unique private key stored within. To authenticate the user to the system, the smart card verifies the PIN. It then enciphers the user's identifier, the PIN, and additional information like date and time, and sends the resulting ciphertext to the authentication server. Authentication succeeds if the authentication server can decipher the message properly.

Authentication Based on Personal Characteristics

Authentication techniques in this category establish the identity of users on the basis of their biometric characteristics.

Biometric techniques can use physical or behavioral characteristics, or a combination of them. Physical characteristics are, for example, the retina, the fingerprint, and the palmprint. Behavioral characteristics include handwriting, voiceprint, and keystroke dynamics (37). Biometric techniques require a first phase in which the characteristic is measured. This phase, also called *enrollment,* generally comprises of several measurements of the characteristic. On the basis of the different measurements, a template is computed and stored at the authentication server. Users' identity is established by comparing their characteristics with the stored templates. It is important to note that, unlike passwords, biometric methods are not exact. A password entered by a user either matches the one stored at the authentication server or it does not. A biometric characteristic instead cannot be required to exactly match the stored template. The authentication result is therefore based on how closely the characteristic matches the stored template. The acceptable difference must be determined in such a way that the method provides a high rate of successes (i.e., it correctly authenticates legitimate users and rejects attackers) and a low rate of unsuccesses. Unsuccesses can either deny access to legitimate users or allow accesses that should be rejected. Biometric techniques, being based on personal characteristics of the users, do not suffer of the weaknesses discusses above for password or token-based authentication. However, they require high-level and expensive technology, and they may be less accurate. Moreover techniques based on physical characteristics are often not well accepted by users because of their intrusive nature. For instance, retinal scanners, which are one of the most accurate biometric method of authentication, have raised concerns about possible harms that the infrared beams sent to the eye by the scanner can cause. Measurements of other characteristics, such as fingerprint or keystroke dynamics, have instead raised concerns about the privacy of the users.

ACCESS CONTROL

Once users are connected to the system, they can require access to its resources and stored data. The enforcement of an access control allows the evaluation of such requests and the determination of whether each request should be granted or denied. In discussing access control, it is generally useful to distinguish between *policies* and *mechanisms.* Policies are high-level guidelines that determine how accesses are controlled and access decisions determined. Mechanisms are low-level software and hardware functions implementing the policies. There are several advantages in abstracting policies from their implementation. First, it is possible to compare different policies and evaluate their properties without worrying about how they are actually implemented. Second, it is possible to devise mechanisms that enforce different policies so that a change of policy does not necessarily require changing the whole implementation. Third, it is possible to devise mechanisms that can enforce multiple policies at the same time, thus allowing users to choose the policy that best suits their needs when stating protection requirements on their data (22,28,29,46,50). The definition and formalization of a set of policies specifying the working of the access control system, providing thus an abstraction of the control mechanism, is called a *model.* A main classification of access control policies distinguishes between discretionary and mandatory policies (and models).

Discretionary Access Control Policies

Discretionary access control policies govern the access of users to the system on the basis of the user's identity and of rules, called *authorizations,* that specify for each user (or group of users) the types of accesses the user can/cannot exercise on each object. The objects to which access can be requested, and on which authorizations can be specified, may depend on the specific data model considered and on the desired granularity of access control. For instance, in operating systems, objects can be files, directories, programs. In relational databases, objects can be databases, relations, views, and, possibly tuples or attributes within a relations. In object-oriented databases objects include classes, instances, and methods. Accesses executable on the objects, or on which authorizations can be specified, may correspond to primitive operations like read, write, and execute, or to higher level operations or applications. For instance, in a bank organization, operations like debit, credit, inquiry, and extinguish can be defined on objects of types accounts.

Policies in this class are called discretionary because they allow users to specify authorizations. Hence the accesses to be or not to be allowed are at the discretion of the users. An authorization in its basic form is a triple ⟨`user`, `object`, `mode`⟩ stating that the `user` can exercise the access `mode` on the `object`. Authorizations of this form represent permission of accesses. Each request is controlled against the authorizations and allowed only if a triple authorizing it exists. This kind of policy is also called *closed* policy, since only accesses for which an explicit authorization is given are allowed, while the default decision is to deny access. In an *open* policy, instead, (negative) authorizations specify the accesses that should not be allowed. All access requests for which no negative authorizations are specified are allowed by default. Most system supports the closed policy. The open policy can be applied in systems with limited protection requirements, where most accesses are to be allowed and the specification of negative authorizations results therefore more convenient.

Specification of authorizations for each single user, each single access mode, and each single object can become quite an administrative burden. By grouping users, modes, and objects, it is possible to specify authorizations holding for a group of users, a collection of access modes, and/or a set of objects (3,33,28,48). This grouping can be user defined or derived from the data definition or organization. For instance, object grouping can be based on the type of objects (e.g., files, directories, executable programs), on the application/activity in which they are used (e.g., ps-files, tex-files, dvi-files, ascii), on data model concepts (e.g., in object-oriented systems a group can be defined corresponding to a class and grouping all its instances), or on other classifications defined by users. Groups of users generally reflect the structure of the organization. For instance, example of groups can be `employee`, `staff`, `researchers`, or `consultants`. Most models considering user groups allow groups to be nested and nondisjoint. This means that users can belong to different groups and groups themselves can be members of other groups provided that there are no cycles in the membership relation (i.e., a

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Figure 2. Example of an access matrix.

group cannot be a member of itself). Moreover, a basic group, called `public`, generally collects all users of the system.

Most recent authorization models support grouping of users and objects, and both positive and negative authorizations (6,7,28,33,40,50). These features, toward the development of mechanisms able to enforce different policies, allow the support of both the closed and the open policy within the same system. Moreover they represent a convenient means to support exceptions to authorizations. For instance, it is possible to specify that a group of users, with the exception of one of its members, can execute a particular access by granting a positive authorization for the access to the group and a negative authorization for the same access to the user. As a drawback for this added expressiveness and flexibility, support of both positive and negative authorizations complicates authorization management. In particular, conflicts may arise. To illustrate, consider the case of a user belonging to two groups. One of the groups has a positive authorization for an access; the other has a negative authorization for the same access. Conflict control policies should then be devised that determine whether the access should in this case be allowed or denied. Different solutions can be taken. For instance, deciding on the *safest* side, the negative authorizations can be considered to hold (*denials take precedence*). Alternatively, conflicts may be resolved on the basis of possible relationships between the involved groups. For instance, if one of the groups is a member of the other one, then the authorization specified for the first group may be considered to hold (*most specific authorization takes precedence*). Another possible solution consists in assigning explicit priorities to authorizations; in case of conflicts the authorization with greater priority is considered to hold.

Authorization Representation and Enforcement. A common way to think of authorizations at a conceptual level is by means of an *access matrix.* Each row corresponds to a user (or group), and each column corresponds to an object. The entry crossing a user with an object reports the access modes that the user can exercise on the object. Figure 2 reports an example of access matrix. Although the matrix represents a good conceptualization of authorizations, it is not appropriate for implementation. The access matrix may be very large and sparse. Storing authorizations as an access matrix may therefore prove inefficient. Three possible approaches can be used to represent the matrix:

Access Control List (ACL). The matrix is stored by column. Each object is associated a list, indicating for each user the access modes the user can exercise on the object.

Capability. The matrix is stored by row. Each user has associated a list, called capability list, indicating for each object in the system the accesses the user is allowed to exercise on the object.

Authorization Table. Nonempty entries of the matrix are reported in a three-column table whose attributes are users, objects, and access modes, respectively. Each tuple in the table corresponds to an authorization.

Figures 3, 4 and 5 illustrate the ACLs, capabilities, and authorization table, respectively, corresponding to the access matrix of Fig. 2.

Capabilities and ACLs present advantages and disadvantages with respect to authorization control and management. In particular, with ACLs it is immediate to check the authorizations holding on an object, while retrieving all the authorizations of a user requires the examination of the ACLs for all the objects. Analogously, with capabilities, it is immediate to determine the privileges of a user, while retrieving all the accesses executable on an object requires the examination of all the different capabilities. These aspects affect the efficiency of authorization revocation upon deletion of either users or objects.

In a system supporting capabilities, it is sufficient for a user to present the appropriate capability to gain access to an object. This represents an advantage in distributed systems, since it permits to avoid multiple authentication of a subject. A user can be authenticated at a host, acquire the appropriate capabilities, and present them to obtain accesses at the various servers of the system. Capabilities suffers, however, from

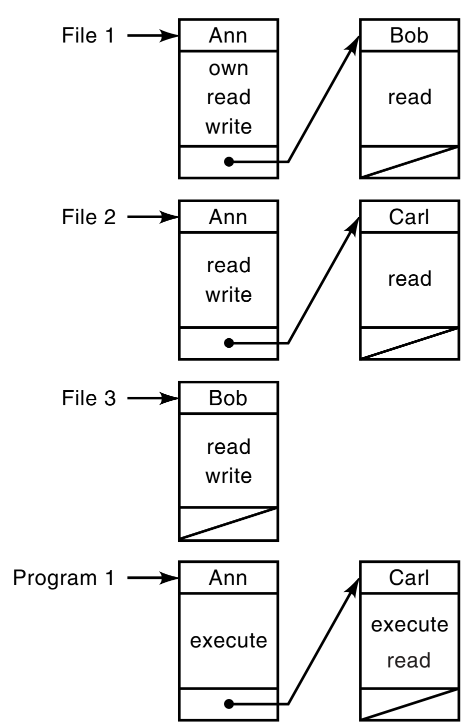

Figure 3. Access control lists for the matrix in Fig. 2.

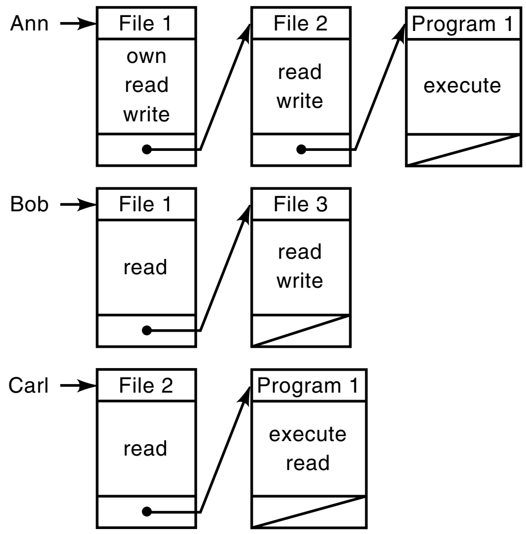

Figure 4. Capabilities for the matrix in Fig. 2.

a serious weakness. Unlike tickets, capabilities can be copied. This exposes capabilities to the risk of forgery, whereby an attacker gain access to the system by copying capabilities. For these reasons capability are not generally used. Most commercial systems use ACLs. The popular Unix operating system uses a primitive form of authorizations and ACLs. Each user in the system belongs to exactly one group, and each file has an owner (generally the user who created it). Authorizations for each file can be specified for the owner, the group to which s/he belongs, and for "the rest of the world." No explicit reference to users or groups is allowed. Each object is associated with an access control list of 9 bits indicating the read, write, and execute privileges of the user (first three bits), the group (second three bits), and the rest of the world (last three bits) on the file. For instance, the ACL `rwxr-x--x` associated with a file indicates that the file can be read, written, and executed by its owner; read and executed by the group to which the owner belongs; and executed by all the other users.

User	Access mode	Object
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 2
Bob	write	File 2
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

Figure 5. Authorization table for the matrix in Fig. 2.

Administration of Authorizations. Discretionary protection policies generally allow users to grant other users authorizations to access the objects. An administrative policy regulates the specification and deletion of the authorizations. Some administrative policies that can be applied are as follows:

Centralized. A privileged user or group of users is reserved the privilege of granting and revoking authorizations.

Ownership. Each object is associated with an owner, who generally coincides with the user who created the object. Users can grant and revoke authorizations on the objects they own.

Decentralized. Extending the previous two approaches, the owner of an object (or its administrators) can delegate other users the privilege of specifying authorizations, possibly with the ability of further delegating it.

Decentralized administration is convenient, since it allows users to delegate administrative privileges to others. Delegation, however, complicates the authorization management. In particular, it becomes more difficult for users to keep track of who can access their objects. Furthermore revocation of authorizations becomes more complex. In decentralized policies, generally authorizations can be revoked only by the user who granted them (or possibly by the object's owner). Upon revocation of an administrative authorization, the problem arises of dealing with the authorizations specified by the users from whom the administrative privilege is being revoked. For instance, suppose that Ann gives Bob the authorization to read File1 and allows him the privilege of granting this authorization to others [in some systems such capability of delegation is called *grant option* (26)]. Consequently Bob grants such authorization to Chris. Suppose now that Ann revokes the authorization from Bob. The question now becomes what should happen to the authorization that Chris has received. Different approaches can be applied in this case. For instance, the authorization of Chris can remain unaltered, and the ability of revoking it given to Ann (8), it can be revoked as well [*recursive* revocation (26)], or the deletion of the Bob's authorization may be refused because of the authorization that would remain pending. Each approach has some pros and cons and can be considered appropriate in different circumstances.

Limitation of Discretionary Policies: The Trojan Horse Problem

In discussing discretionary policies we have referred to users and to access requests on objects submitted by users. Although it is true that each request is originated because of some user's actions, a more precise examination of the access control problem shows the utility of separating *users* from *subjects*. Users are passive entities for whom authorizations can be specified and who can connect to the system. Once connected to the system, users originate processes (subjects) that execute on their behalf and, accordingly, submit requests to the system. Discretionary policies ignore this distinction and evaluate all requests submitted by a process running on behalf of some user against the authorizations of the user. This aspect makes discretionary policies vulnerable from processes executing malicious programs exploiting the authorizations of the user on behalf of whom they are executing. In particular, the access control system can be bypassed by Trojan Horses embedded in programs. A Trojan Horse is a computer program with an apparently or actually useful function that contains additional *hidden* functions to surreptitiously exploit the legitimate authorizations of the invoking process. A Trojan Horse can improperly use any authorizations of the invoking user, for instance, it could even delete all files of the user (this destructive behavior is not uncommon in the case of viruses). This vulnerability of Trojan Horses, together with the fact discretionary policies do not enforce any control on the flow of information once this information is acquired by a process, makes it possible for processes to leak information to users not allowed to read it. All this can happen without the cognizance of the data administrator/owner, and despite the fact that each single access request is controlled against the authorizations. To understand how a Trojan Horse can leak information to unauthorized users despite the discretionary access control, consider the following example. Assume that within an organization, Vicky, a top-level manager, creates a file Market containing important information about releases of new products. This information is very sensitive for the organization and, according to the organization's policy, should not be disclosed to anybody besides Vicky. Consider now John, one of Vicky's subordinates, who wants to acquire this sensitive information to sell it to a competitor organization. To achieve this, John creates a file, let's call it Stolen, and gives Vicky the authorization to write the file. Note that Vicky may not even know about the existence of Stolen or about the fact that she has the write authorization on it. Moreover John modifies an application generally used by Vicky, to include two hidden operations, a read operation on file Market and a write operation on file Stolen [Fig. 6(a)]. Then he gives the new application to his manager. Suppose now that Vicky executes the application. Since the application executes on behalf of Vicky, every access is checked against Vicky's authorizations, and the read and write operations above will be allowed. As a result, during execution, sensitive information in Market is transferred to Stolen and thus made readable to the dishonest employee John, who can then sell it to the competitor [Fig. 6(b)].

The reader may object that there is little point in defending against Trojan Horses leaking information flow: Such an information flow could have happened anyway, by having Vicky explicitly tell this information to John, possibly even off-line, without the use of the computer system. Here is where the distinction between users and subjects operating on their behalf comes in. While users are trusted to obey the access restrictions, subjects operating on their behalf are not. With reference to our example, Vicky is trusted not to release to John the sensitive information she knows, since, according to the authorizations, John cannot read it. However, the processes operating on behalf of Vicky cannot be given the same trust. Processes run programs which, unless properly certified, cannot be trusted for the operations they execute, as illustrated by the example above. For this reason restrictions should be enforced on the operations that processes themselves can execute. In particular, protection against Trojan Horses leaking information to unauthorized users requires controlling the flows of information within process execution and possibly restricting them (5,15,25,30,35,36). Mandatory policies provide a way to enforce information flow control through the use of labels.

Mandatory Policies

Mandatory security policies enforce access control on the basis of classifications of *subjects* and *objects* in the system. Objects are the passive entities storing information such as files, records, and records' fields in operating systems; databases, tables, attributes, and tuples in relational database systems. Subjects are active entities that request access to the objects. An *access class* is defined as consisting of two components: a *security level* and a *set of categories*. The security level is an element of a hierarchically ordered set. The levels generally considered are Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U), where TS > S > C > U. The set of categories is a subset of an unordered set, whose elements reflect functional or competence areas (e.g., NATO, Nuclear, Army for military systems; Financial, Administration, Research for commercial systems). Access classes are partially ordered as follows: an access class c_1 *dominates* ($\geq$) an access class c_2 iff the security level of c_1 is greater than or equal to that of c_2 and the categories of c_1 include those of c_2. Two classes c_1 and c_2 are said to be incomparable if neither $c_1 \geq c_2$ nor $c_2 \geq c_1$ holds. Access classes together with the dominance relationship between them form a lattice. Figure 7 illustrates the security lattice for the security levels TS and S and the categories Nuclear and Army. Each object and each user in the system is assigned an access class. The security level of the access class associated with an object reflects the sensitivity of the information contained in the object, that is, the potential damage that could result from the unauthorized disclosure of the information. The security level of the access class associated with a user, also called *clearance,* reflects the user's trustworthiness not to disclose sensitive information to users not cleared to see it. Categories are used to provide finer-grained security classifications of subjects and objects than classifications provided by security levels alone, and are the basis for enforcing *need-to-know* restrictions. Users can connect to their system at any access class dominated by their clearance. A user connecting to the system at a given access class originates a subject at that access class. For instance, a user cleared (`Secret`, $\emptyset$) can connect to the system as a (`Secret`, $\emptyset$), (`Confidential`, $\emptyset$), or (`Unclassified`, $\emptyset$) subject. Requests by a subject to access an object are controlled with respect to the access class of the subject and the object and granted only if some relationship, depending on the requested access, is satisfied. In particular, two principles, first formulated by Bell and LaPadula (4), must be satisfied to protect information confidentiality:

No Read Up. A subject is allowed a read access to an object only if the access class of the subject dominates the access class of the object.

No Write Down. A subject is allowed a write access to an object only if the access class of the subject is dominated by the access of the object. (In most applications, subjects are further restricted to write only at their own level so that no overwriting of sensitive information can take place by low subjects.)

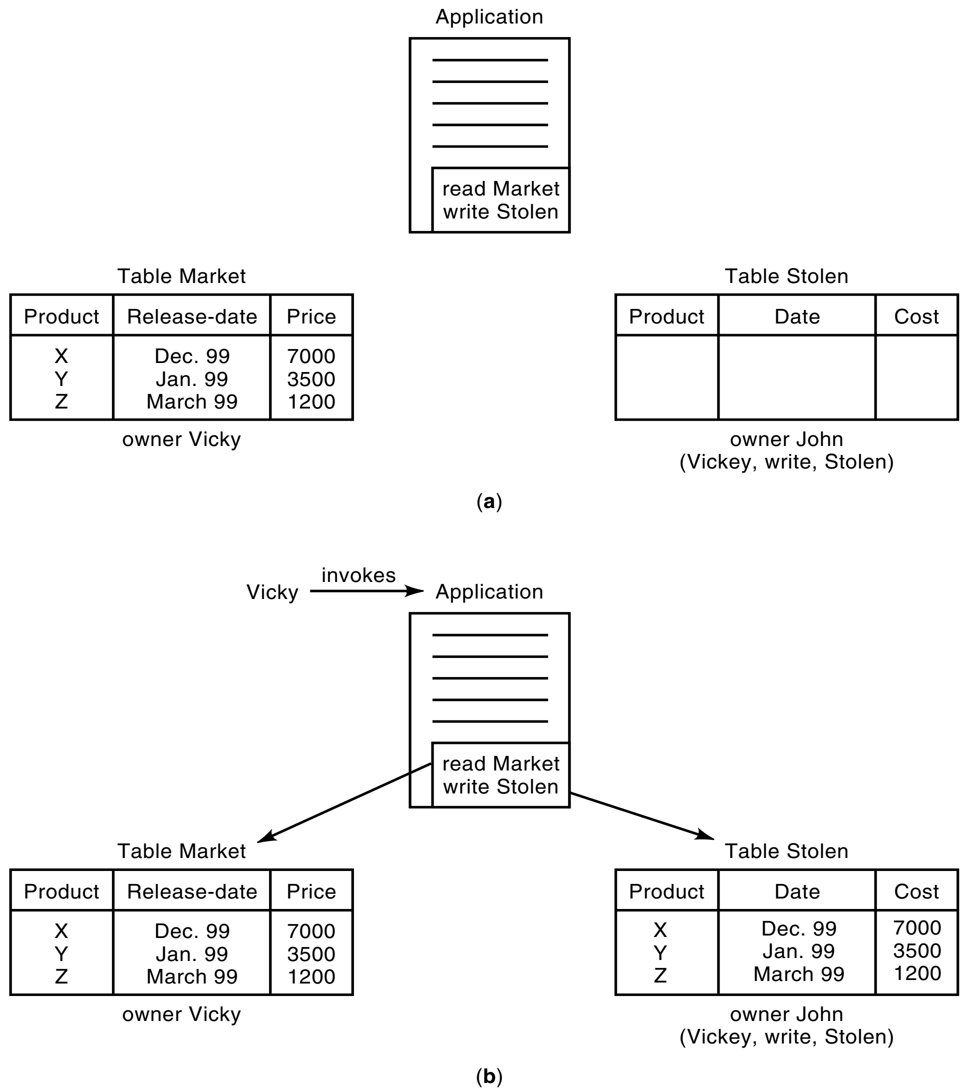

Figure 6. Example of a Trojan Horse.

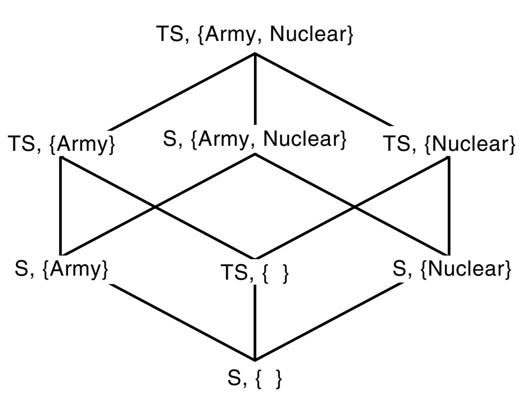

Figure 7. Example of a classification lattice.

Satisfaction of these two principles prevents information flow from high-level subjects/objects to subjects/objects at lower levels, thereby ensuring the satisfaction of the protection requirements (i.e., no process will be able to make sensitive information available to users not cleared for it). This is illustrated in Fig. 8. Note the importance of controlling both read and write operations, since both can be improperly used to leak information. Consider the example of the Trojan Horse illustrated before. Possible classifications reflecting the specified access restrictions could be: Secret for Vicky and Market, and Unclassified for John and Stolen. In the respect of the no-read-up and no-write-down principles, the Trojan Horse will never be able to complete successfully. If Vicky connects to the system as a Secret (or Confidential) subject, and thus the application runs with a Secret (or Confidential) access class, the write operation would be blocked. If Vicky invokes the application as an Unclassified subject, the read operation will be blocked instead.

Given the no-write-down principle, it is clear now why users are allowed to connect to the system at different access

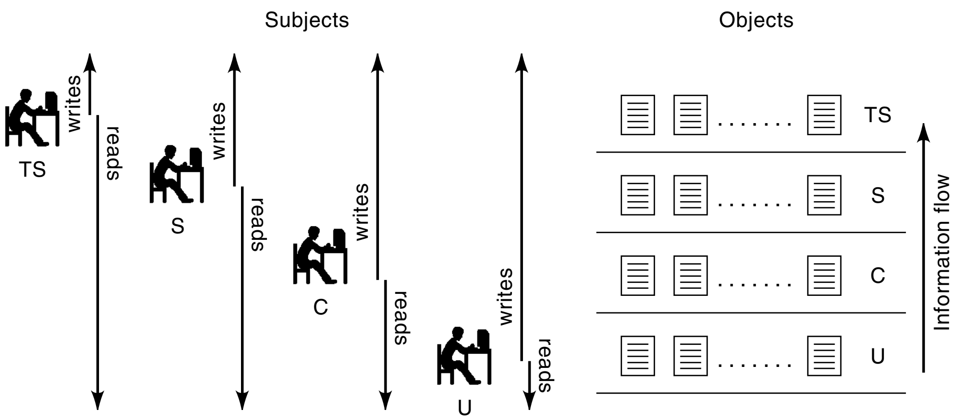

Figure 8. Controlling information flow for secrecy.

classes so that they are able to access information at different levels (provided that they are cleared for it). For instance, Vicky has to connect to the system at a level below her clearance if she wants to write some Unclassified information, such as working instructions for John. Note also that a lower class does not mean "less" privileges in absolute terms, but only less reading privileges, as it is clear from the example above.

The mandatory policy that we have discussed above protects the confidentiality of the information. An analogous policy can be applied for the protection of the integrity of the information, to keep untrusted subjects from modifying information they cannot write and compromising its integrity. With reference to our organization example, for instance, integrity could be compromised if the Trojan Horse implanted by John in the application would write data in file Market. Access classes for integrity comprise of an integrity level and a set of categories. The set of categories is as seen for secrecy. The integrity level associated with a user reflects the user's trustworthiness for inserting, modifying, or deleting information. The integrity level associated with an object reflects both the degree of trust that can be placed on the information stored in the object and the potential damage that could result from unauthorized modification of the information. Example of integrity levels include Crucial (C), Important (I), and Unknown (U). Access control is enforced according to the following two principles:

No Read Down. A subject is allowed a read access to an object only if the access class of the object dominates the access class of the subject.

No Write Up. A subject is allowed a write access to an object only if the access class of the subject is dominated by the access of the object.

Satisfaction of these principles safeguard integrity by preventing information stored in low objects (and therefore less reliable) to flow to high objects. This is illustrated in Fig. 9.

As it is visible from Figs. 8 and 9, secrecy policies allow the flow of information only from lower to higher (security) levels, while integrity policies allow the flow of information only from higher to lower security levels. If both secrecy and integrity have to be controlled objects and subjects have to be assigned two access classes, one for secrecy control and one for integrity control.

The main drawback of mandatory protection policies is the rigidity of the control. They require the definition and application of classifications to subjects and objects. This may not always be feasible. Moreover accesses to be allowed are determined only on the basis of the classifications of subjects and objects in the system. No possibility is given to the users for granting and revoking authorizations to other users. Some approaches have been proposed that complement discretionary access control with flow control similar to that enforced by mandatory policies (5,25,35).

Role-Based Policies

A class of access control policies that has been receiving considerable attention recently is represented by role-based policies (20,21,44,49). Role-based policies govern the access of users to the information on the basis of their organizational role. A role can be defined as a set of actions and responsibilities associated with a particular working activity. Intuitively a role identifies a task, and corresponding privileges, that users need to execute to perform organizational activities. Example of roles can be `secretary`, `dept-chair`, `programmer`, `payroll-officer`, and so on. Authorizations to access objects are not specified directly for users to access objects: Users are given authorizations to activate roles, and roles are given authorizations to access objects. By activating a given role (set of roles), a user is able to execute the accesses for which the role is (set of roles are) authorized. Like groups, roles can also be organized in a hierarchy, along which authorizations can be propagated.

Note the different semantics that groups, and roles carry (see section entitled Discretionary access control policies). Roles can be "activated" and "deactivated" by users at their discretion, while group membership always applies; that is, users cannot enable and disable group memberships (and corresponding authorizations) at their will. Note, however, that a same "concept" can be seen both as a group and as a role. To understand the difference between groups and roles, consider the following example: We could define a group, called `G_programmer`, consisting all users who are programmers. Any authorizations specified for `G_programmer` are propagated to its members. Thus, if an authorization to read `tech-`

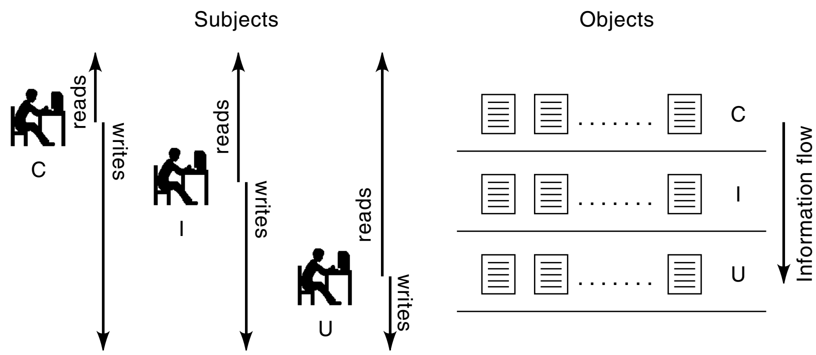

Figure 9. Controlling information flow for integrity.

reports is given to G_programmer, its members can exercise this right. We could also define a role, called R_programmer, and associate to it those privileges that are related to the programming activity and necessary for the programmers to perform their jobs (compiling, debugging, writing reports, etc.). These privileges can be exercised by authorized users only when they choose to assume the role R_programmer. It is important to note that roles and groups are two complementary concepts; they are not mutually exclusive.

The enforcement of role-based policies present several advantages. Authorization management results simplified by the separation the users's identity from the authorizations they need to execute tasks. Several users can be given the same set of authorizations simply by assigning them the same role. Also, if a user's responsibilities change (e.g., because of a promotion), it is sufficient to disable the user for the previous roles and enable him/her for a new set of roles, instead of deleting and inserting the many access authorizations that this responsibility change implies. A major advantage of role-based policies is represented by the fact that authorizations of a role are enabled only when the role is active for a user. This allows the enforcement of the least privilege principle, whereby a process is given only the authorizations it needs to complete successfully. This confinement of the process in a defined workspace is an important defense against attacks aiming at exploiting authorizations (as the Trojan Horse previously illustrated). Moreover, the definition of roles and related authorizations fits with the information system organization and allows to support related constraints, such as separation of duties (11,38,43). Separation of duties requires that no user should be given enough privileges to be able to misuse the system. For example, the person authorizing a paycheck should not be the same person who prepares them. Separation of duties can be enforced statically by controlling the specification of roles associated with each user and authorizations associated with each role, or dynamically by controlling the actions actually executed by users when playing particular roles (28,38).

AUDITING

Authentication and access controls are enforced prior to the users' access to the system or its resources, and more precisely, they determine whether such accesses should be allowed or denied. Auditing controls complement the previous two form of controls by providing a post facto evaluation of the accesses (or of their requests) to determine whether security violations have been attempted or have occurred. Despite the fact that each request is controlled and allowed only if the authenticated user is authorized (or has the appropriate clearance) for it, violations are still possible: Attackers can gain access to the system masquerading as legitimate users, software or security mechanisms may contain bugs or be bypassed, Trojan Horses or viruses may have been implanted in programs, legitimate users can misuse their privileges [most security experts believe that insiders are responsible for a vast majority of computer crimes, comprising about 80% according to a US Air Force study (10)]. An off-line examination of the events occurred in the system may help pinpoint these situations. Auditing controls can also work as a deterrent, since users are less likely to attempt violations or behave improperly if they know their activities are being monitored.

Events Registration and Analysis

The enforcement of an audit control requires the registration (*logging*) of all the events occurring in the system for later examination. Such registration is called *audit trail* or *log*. The audit trail must be complete and detailed in order to allow a full examination of all the events. However, it is important that only relevant events be recorded to avoid the proliferation of useless data. All actions requested by privileged users, such as the system and the security administrator, should be logged. Registration of these actions is important to ensure that privileged users did not abuse their privileges (the "who guards the guardian" problem) and to determine possible areas where attackers gained superuser privileges. Events to be recorded may vary depending on the desired granularity and control to be enforced. For instance, high-level commands requested by users and/or the elementary read and write operations into which they translate could be recorded. The first option gives a picture of the events at a higher level, and therefore may be more understandable to a human auditor, but it might hide some details (e.g., operations actually executed on the underlying data) that may be evidence of anomalous behavior or violations. The alternative solution provides the desired detail and in such extensive form that the data cannot be easily analyzed by a human auditor. Information to be recorded for each event includes the subject requesting access, the location and the time of the request, the operation requested and the object on which it was requested, the response (grant or deny) of the access control system, and general information on the execution (CPU, I/0, memory usage, success or abort execution, etc.).

A big problem with audit controls is that they are difficult to enforce. The amount of data recorded reaches massive proportions very quickly. Analyzing these data to determine which violations have been attempted or have occurred is often an impossible task. A security violation may occur through the execution of several different operations and leave a number of records in the audit trail. Attackers have been known to spread their activities over a long period of time so that their operations could be concealed among many others. Because of these data problems, audit analysis is often executed only if a violation is suspected (e.g., because the system shows an anomalous or erroneous behavior) and by examining only the audit data that may be connected with the suspected violation. In other words, analysis is executed with some knowledge of "what to look for" in the audit trail. This may in fact happen some time after the violation occurred. Clearly this enforcement of audit control is insufficient. Recent research has proposed the use of automated tools to help the security officer in the enforcement of audit controls. These tools can examine the audit log and produce reports and summaries regarding the events occurred, which can then be examined by the auditor. More sophisticated tools, also called *intrusion detection systems,* can also perform audit analysis and automatically, or semiautomatically, pinpoint possible violations or anomalies (34).

Intrusion Detection

The basic assumption of intrusion detection systems is that each violation, or attempt of violation, translates in some observable on the events occurring in the system. Some approaches that can be used to define what constitutes a violation in terms of the events that occurred in the system are as follows.

Threshold Based. Since violations involve abnormal use of the system, acceptable fixed thresholds defined by the security officer could control the occurrences of specific events over a given period of time and raise an alarm if the thresholds are passed. For instance, more than three failed connection attempts in a row for a given login may be considered suspicious (symptomatic of an attacker trying to gain access to the system by guessing a legitimate user's password).

Anomaly Based. Again, these are violations that involve abnormal use of the system. Normal behavior, however, is not defined with respect to predefined fixed thresholds, but rather as "behavior significantly different from what is normally observed." The security officer specifies profiles against which normal behavior must then be evaluated. Possible profiles could be the number of daily connections for a given user (or set of users), login duration, or number of browsing commands per session. Moreover the security officer would define the acceptable deviation from the normal behavior, possibly as a function of it. The audit controls observe the system working and define, based on the observations, the normal behavior for the different users (or groups of users), actions, objects, and, more generally types of events for each specified profile. An alarm is raised if an observation would change a given profile of an amount greater than the acceptable threshold of deviation.

Rule Based. Rules, defined by the security officer, would describe violations on the basis of known intrusion patterns or system vulnerabilities. A rule could, for instance, indicate whether a certain sequence of actions, or actions satisfying particular conditions, are symptomatic of a violation. For instance, opening an account late at night and transferring to it in small amounts taken from different accounts may be considered suspicious.

All the approaches described above have some advantages, in terms of kind of violations they pinpoint, and some shortcomings. The threshold-based approach could be used only to determine violations that imply an improper use of the system or its resources, and for which such an implication is known. The anomaly-based approach could overcome the limitation of requiring prior acceptable thresholds, thus making it possible, for instance, to discover Trojan Horses and viruses, whose execution generally changes a program's usual behavior. However, it also can only detect violation that involve anomalous use. Moreover it is limited by the fact that it is not always possible to define abnormal behavior for users: Some users may habitually exhibit erratic behavior (e.g., logging on off-hours and from remote location); other users may be "bad" from the beginning, or change their behavior so slowly as to not pass the acceptable threshold. The rule-based approach complements the previous two approaches by providing a way to define violations that do not involve abnormal use of resources. However, it can control only violations for which there exists prior knowledge describing how a violation maps into recordings in the audit logs. For these reasons none of the approaches can be considered alone. Rather, they complement one another, since each can be applied to determine a different type of violation.

Other approaches to intrusion detection and audit controls are possible. For instance, neural network (14,23), state-based (27), or model-based (24) approaches have been proposed as a way to describe violations in terms of events or observables in the system. Some other approaches proposed use specific techniques as a protection against specific attacks. For instance, the keystroke latency property of a user, which we mentioned earlier as a possible method of authentication, can be applied to pinpoint attackers who gain access to the system by masquerading as legitimate users.

In our discussion we have assumed that the intrusion detection system raises an alarm whenever a violation is suspected. More sophisticated systems, called *active,* react to violations automatically and undertake appropriate defense measures. For instance, just as a masquerading attack is suspected, the system will automatically terminate the connection and disable the login.

DATA ENCRYPTION

Another measure for protecting information is provided by cryptography. Cryptographic techniques allow users to store, or transmit, encoded information instead of the actual data. An *encryption* process transforms the *plaintext* to be protected into an encoded *ciphertext,* which can then be stored or trans-

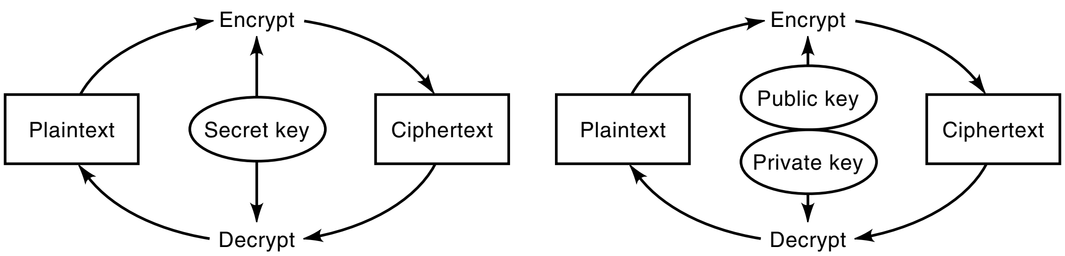

Figure 10. Secret key compared with public key cryptography.

mitted. A *decryption* process is used to retrieve the *plaintext* from the *ciphertext.* The encryption and decryption functions take a *key* as a parameter. A user with access to data, or able to sniff the network, but who lacks the appropriate decryption key will not be able to understand the text. Also tampering of data results is prevented by users without the appropriate encryption key.

Cryptographic techniques must be proved resistant to attacks by cryptoanalysts trying to break the system to recover the plaintext or the key, or to forge data (generally messages transmitted over the network). Cryptoanalysis attacks can be classified according to how much information the cryptoanalyst has available. In particular, with respect to secrecy, attacks can be classified as *ciphertext-only, known-plaintext,* and *chosen-plaintext.* In ciphertext-only attacks the cryptoanalyst only knows the ciphertext, although he/she may know the encryption algorithm, the plaintext language, and possibly some words used in the plaintext. In known-plaintext attacks the cryptoanalyst also knows some plaintext and corresponding ciphertext. In chosen-plaintext attacks the cryptoanalyst is able to acquire the ciphertext corresponding to a selected plaintext. Most cryptographic techniques are designed to withstand chosen-plaintext attacks. The robustness of cryptographic algorithms relies on the amount of work and time that would be necessary for a cryptoanalyst to break the system using the best available techniques. With respect to protecting authenticity of the information, there are two main classes of attacks: *impersonation attacks,* in which the cryptoanalyst creates a fraudulent ciphertext without knowledge of the authentic ciphertext, and *substitution attacks,* in which the cryptoanalyst intercept the authentic ciphertext and improperly modifies it.

Encryption algorithms can be divided into two main classes: *symmetric,* or *secret key,* and *asymmetric,* or *public key.* Symmetric algorithms encrypt and decrypt text using the same key or a pair of keys easily derivable one from the other. Public key algorithms use, instead, two different keys. A *public* key is used to encrypt, and a *private* key, which cannot be guessed by knowing the public key, is used to decrypt. This is illustrated in Fig. 10. Symmetric algorithms rely on the secrecy of the key. Public key algorithms rely on the secrecy of the private key.

Symmetric (Secret Key) Algorithms

Symmetric algorithms use substitution techniques, transposition techniques, or a combination of both techniques.

Substitution Algorithms. Substitution algorithms define a mapping, based on the key, between characters in the plaintext and characters in the ciphertext. Some substitution techniques are as follows:

Simple Substitution. Simple substitution algorithms are based on a one-to-one mapping between the plaintext alphabet and the ciphertext alphabet. Each character in the plaintext alphabet is therefore replaced with a fixed substitute in the ciphertext alphabet. An example of simple substitution is represented by the algorithms based on *shifted* alphabets, in which each letter of the plaintext is mapped onto the letter at a given fixed distance from it in the alphabet (wrapping the last letter with the first). An example of such algorithm is the Caesar cipher in which each letter is mapped to the letter 3 positions after it in the alphabet. Thus A is mapped to D, B to E, and Z to C. For instance, `thistext` would be encrypted as `wklvwhaw`. Simple substitution techniques can be broken by analyzing single-letter frequency distribution (16).

Homophonic Substitution. Homophonic substitution algorithms map each character of the plaintext alphabet onto a set of characters, called its *homophones,* in the ciphertext alphabet. There is therefore a one-to-many mapping between a plaintext character and the corresponding character in the ciphertext. (Obviously a vice-versa operation cannot occur, since decrypting cannot be ambiguous.) In this way different occurrences of a same character in the plaintext are mapped to different characters in the ciphertext. This characteristic allows the flattening of the letter frequency distribution in the ciphertext and provides a defense against attacks exploiting it. A simple example of homophonic substitution (although not used for ciphering) can be seen in the use of characters for phone numbers. Here the alphabet of the plaintext are numbers, the alphabet of the ciphertext are the letters of the alphabet but for Q and Z which are not used and numbers 0 and 1 (which are not mapped to any letter). Number 2 maps to the first three letters of the alphabet, number 3 to the second three letters, and so on. For instance, number 6974663 can be enciphered as `myphone`, where the three occurrences of character `6` have been mapped to three different letters.

Polyalphabetic Substitution. Polyalphabetic substitution algorithms overcome the weakness of simple substitution through the use of multiple substitution algorithms. An example of definition of multiple substitutions is represented by the cipher disk of Alberti, illustrated in Fig. 11. The disk is composed of 2 circles. The outer circle reports 20 letters of the plaintext (H, K, and Y were not included, while J, U, and W were not part of the considered alphabet) plus the numbers 1, 2, 3, 4. The movable inner circle reports the 23 letters of the alphabet plus the character &. By moving the inner circle, it is possible to define 24 different substitutions. Most polyalphabetic algorithms use periodic sequences of alphabets. For instance, the Vigenère cipher uses a word as a key. The position in the alphabet of the *i*th character of the key gives the number of right shifts to be enforced on each *i*th element (modulo the key length) of the plaintext. For instance, if key CRYPT is used, then the first, sixth, eleventh, . . ., characters of the plaintext will be shifted by 3 (the position of C in the alphabet), the second, seventh, twelfth, . . ., characters will be shifted by 17 (the position of R in the alphabet), and so on.

Polygram Substitution. While the previous algorithms encrypt a letter at the time, polygram algorithms encrypt blocks of letters. The plaintext is divided into blocks of letters. The mapping of each character of a block depends on the other characters appearing in the block. For example, the Playfair cipher uses as key a 5×5 matrix where the 25 letters of the alphabet (J is not considered) are inserted in some order. The plaintext is divided into blocks of length two. Each pair of characters is mapped onto a pair of characters in the ciphertext, where the mapping depends on the position of the two plaintext characters in the matrix (e.g., whether they are in the same column and/or row). Polygram substitution destroys single-letter frequency distribution, thus making cryptoanalysis harder.

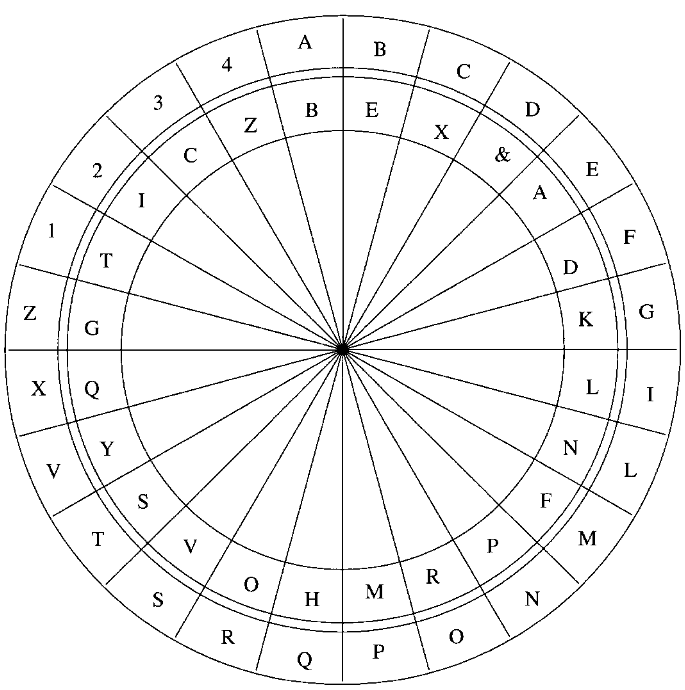

Figure 11. Cipher disk.

Transposition Algorithms. Transposition algorithms determine the ciphertext by permuting the plaintext characters according to some scheme. The ciphertext therefore contains exactly the same characters as the plaintext but in different order. Often the permutation scheme is determined by writing the plaintext in some geometric figure and then reading it by traversing the figure in a specified order. Some transposition algorithms, based on the use of matrixes, are as follows:

Columnary Transposition. The plaintext is written in a matrix by rows and re-read by columns according to an order specified by the key. Often the key is a word: The number of characters in the key determines the number of columns, and the position of the characters considered in alphabetical order determines the order to be considered in the reading process. For instance, the key CRYPT would imply the use of a five-column matrix, where the order of the columns to be read is 14253 (the position in the key of the key characters considered in alphabetical order, i.e., CPRTY).

Periodic Transposition. This is a variation of the previous technique, where the text is also read by rows (instead of by columns) according to a specified column order. More precisely, instead of indicating the columns to be read, the key indicates the order in which the characters in each row must be read, and the matrix is read row by row. For instance, by using key CRYPT, the ciphertext is obtained by reading the first, fourth, second, fifth, and third character of the first row; then the second row is read in the same order, then the third row, and so on. This process is equivalent to breaking up the text into blocks with the same length as the key, and permuting the characters in each block according to the order specified by the key.

Pure transposition and substitution techniques have proved very vulnerable. Transposition algorithms can be broken through anagramming techniques, since the characters in the ciphered text correspond exactly to the characters in the plaintext. The fact that a transposition method has been used to encrypt can be determined by the fact that the ciphertext respects the frequency letter distribution of the considered alphabet. Simple substitution algorithms are vulnerable from attacks exploiting single-letter frequency distribution. Among them, shifted alphabet ciphers are easier to break, given that the mapping function applies the same transformation to all the characters. Stronger algorithms can be obtained by combining the two techniques (47).

Product Algorithms: The Data Encryption Standard (DES)

Product algorithms combine transposition and substitution techniques. The most well-known example of a product algorithm is the Data Encryption Standard (DES), which was adopted in 1977 by the National Bureau of Standards (39). DES considers text blocks of 64 bits, and a key of 56 bits. The key is actually composed of 64 bits, but one of the bit in each of the 8 bytes is used for integrity control. The algorithm, sketched in Fig. 12, works as follows: First, the 64-bit block goes under a fixed permutation specified as an 8×8 matrix IP. The permutation transposes the 64 bits according to the order specified by the entries of the matrix. Then the 64-bit block goes through 16 iterations as follows: Let $L_i = t_1, \ldots, t_{32}$ and $R_i = t_{33}, \ldots, t_{64}$ denote the first and last half, respectively, of block T_i. The ith iteration produces block T_i, with $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, where $\oplus$ is the exclusive-or operator, f is a function that combines substitution and transposition, and K_i is a subset of 48 bits of the considered 56-bit key. Each K_i is obtained by permutation, transposition, and shifting over the original key. At the end of the sixteenth round, the output is subjected to another permutation IP^{-1}, defined as the inverse of the original one. This last permutation is necessary to make the algorithm applicable for both encrypting and decrypting. The decrypting process uses the same algorithm but uses the keys in reversed order (the first iteration uses K_{16} and the last K_1), and decrypts messages by computing $R_{i-1} = L_i$ and $L_{i-1} = R_i \oplus f(L_i, K_i)$.

DES has been implemented both in software and in hardware. The hardware implementation proves faster and more secure (software can be modified by intruders, whereas hardware can be tamper resistant). The software method is cheaper and generally easier to integrate with the system. Since the time it was adopted as a standard, researchers have raised several concerns about possible weaknesses of DES. The main objections are the use of 56 bits for the key, which is considered too small, and possible hidden trapdoors in the implementation of function f (in particular the S-box, enforcing substitution, whose design was secret at the time the algorithm was adopted). However, DES has been reviewed every five years since it became a standard, and it has been reaffirmed until 1998.

Asymmetric (Public Key) Algorithms

Public key algorithms use two different keys for encryption and decryption. They are based on the application of one-way functions. A one-way function is a function that satisfies the property that it is computationally infeasible to compute the input from the result. Public key algorithms are therefore based on hard to solve mathematical problems, such as computing logarithms, as in the proposals by Diffie and Hellman (18), who are the proponents of public key cryptography, and by ElGamal (19), or factoring, as in the RSA algorithm illustrated next.

RSA Algorithm. The best-known public key algorithm is the RSA algorithm, whose name is derived from the initials of its inventors: Rivest, Shamir, and Adleman (41). It is based on the idea that it is easy to multiply two large prime numbers, but it is extremely difficult to factor a large number. The establishment of the pair of keys works as follows: The users wishing to establish a pair of keys chooses two large primes p and q (which are to remain secret) and computes $n = pq$ and $\phi(n) = (p - 1)(q - 1)$, where $\phi(n)$ is the number of elements between 0 and $n - 1$ that are relatively prime to n. Then the user chooses an integer e between 1 and $\phi(n) - 1$, that is, relatively prime to $\phi(n)$, and computes its inverse d such that $ed \equiv 1 \bmod \phi(n)$. The d can be easily computed by knowing $\phi(n)$. The encryption function E raises the plaintext M to the power e, modulo n. The decryption function D raises

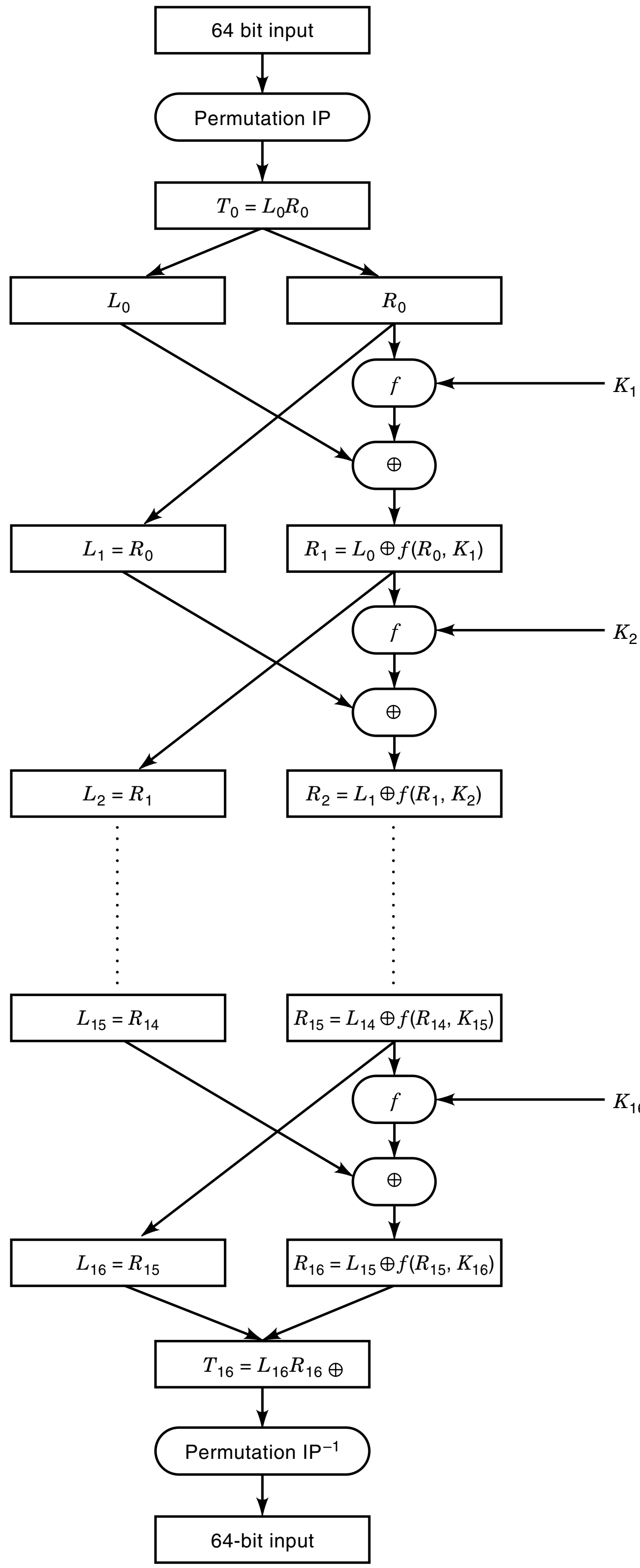

Figure 12. DES enciphering algorithm.

the ciphertext C to the power d, modulo n. That is, $E(M) = M^e \bmod n$, and $D(C) = C^d \bmod n$. Here the public key is represented by the pair (e, n) and the private key by d. Because $\phi(n)$ cannot be determined without knowing the prime factors p and q, it is possible to keep d secret even if e and n are made public. The security of the algorithm depends therefore on the difficulty of factoring n into p and q. Usually a key with n of 512 bits is used, whose factorization would take a half million MIPS-years with the best techniques known today. The algorithm itself, however, does not constraint the key length. The key length is variable. A longer key provides more protection, while a shorter key proves more efficient. The authors of the algorithm suggested using a 100-digit number for p and q, which would imply a 200-digit number for n. In this scenario factoring n would take several billion years. The block size is also variable, but it must be smaller than the length of the key. The ciphertext block is the same length as the key.

Application of Cryptography

Cryptographic techniques can be used to protect the secrecy of information stored in the system by making it not understandable to intruders who bypass access controls. For instance, password files are generally encrypted. Cryptography proves particularly useful in the protection of information transmitted over a communication network (31). Information transmitted over a network is vulnerable from *passive* attacks in which intruders sniff the information, thus compromising its secrecy, and from *active* attacks in which intruders improperly modify the information, thus compromising its integrity. Protecting against passive attacks means safeguarding the confidentiality of the message being transmitted. Protecting against active attacks requires to be able to ensure the *authenticity of the message, its sender, and its receiver.* Authentication of the receiver means that the sender must be able to verify that the message is received by the recipient for which it was intended. Authentication of the sender means that the recipient of a message must be able to verify the identity of the sender. Authentication of the message means that sender and recipient must be able to verify that the message has not been improperly modified during transmission.

Both secret and public key techniques can be used to provide protection against both passive and active attacks. The use of secret keys in the communication requires the sender and the receiver to share the secret key. The sender encrypts the information to be transmitted by using the secret key and then sends it. Upon reception, the receiver decrypts the information with the same key and recovers the plaintext. Secret key techniques can be used if there is confidence in the fact that the key is only known to the sender and recipient and no disputes arise (e.g., a dispute can arise if the sender of a message denies to have ever sent it). Public keys, like secret keys, can provide authenticity of the sender, the recipient, and the message as follows: Each user establishes a pair of keys; the private key is known only to him/her, and the public key can be known to everybody. A user wishing to send a message to another user encrypts the message by using the public key of the receiver and then sends it. Upon reception, the receiver decrypts the message with his/her private key. Public keys can also be used to provide *nonrepudiation,* meaning the sender of a message cannot deny having sent it. The use of public keys to provide nonrepudiation is based on the concept of *digital signatures* which, like handwritten signatures, provides a way for a sender to sign the information being transmitted. Digital signatures are essentially encoded information, function of the message and the key, which are appended to a message. Digital signatures can be enforced through public key technology by having the sender of a message encrypting the message with his private key before transmission. The recipient will retrieve the message by decrypting it with the public key of the sender. Nonrepudiation is provided, since only the sender knows his/her public key and therefore only the sender could have produced the message in question. In the application of secret keys, instead, the sender can claim that the message was forged by the recipient him/herself, who also knows the key. The two uses of public keys can be combined, thus providing sender, message, and recipient authentication together with nonrepudiation.

Public key algorithms can do everything that secret key algorithms can do. However, all the known public key algorithms are orders of magnitude slower than secret key algorithms. For this reason often public key techniques are used for things that secret key techniques cannot do. In particular, they may be used at the beginning of a communication for authentication and to establish a secret key with which to encrypt information to be transmitted.

CONCLUSIONS

Ensuring protection to information stored in a computer system means safeguarding the information against possible violations to its secrecy, integrity, or availability. This is a requirement that any information system must satisfy and that involves the enforcement of different protection methods and related tools. Authentication, access control, auditing, and encryption are all necessary to this task. As it should be clear from this article, these different measures are not independent but rather strongly dependent on each other. Access control relies on good authentication, since accesses allowed or denied depend on the identity of the user requesting them. Strong authentication supports good auditing, since users can be held accountable for their actions. Cryptographic techniques are necessary to ensure strong authentication, such as to securely store or transmit passwords. A weakness in any of these measures may compromise the security of the whole system (a chain is as strong as its weakest link). Their correct and coordinated enforcement is therefore crucial to the protection of the information.

ACKNOWLEDGMENTS

The work of Pierangela Samarati was supported by DARPA/Rome Laboratory under contract F30602-96-C-0337 and by the National Science Foundation under grant ECS-94-22688.

BIBLIOGRAPHY

1. M. Abrams, S. Jajodia, and H. Podell (eds.), *Information Security: An Integrated Collection of Essays,* Los Alamitos, CA: IEEE Computer Society Press, 1994.

2. L. Badger, A model for specifying multi-granularity integrity policies, *Proc. IEEE Comput. Soc. Symp. Security Privacy,* Oakland, CA, 1989, pp. 269–277.
3. R. W. Baldwin, Naming and grouping privileges to simplify security management in large databases, *Proc. IEEE Symp. Security Privacy,* Oakland, CA, 1990, pp. 61–70.
4. D. E. Bell and L. J. LaPadula, *Secure computer systems: Unified exposition and Multics interpretation,* Technical Report, Mitre Corp., Bedford, MA, 1976.
5. E. Bertino et al., Exception-based information flow control in object-oriented systems, *ACM Trans. Inf. Syst. Security,* June 1998.
6. E. Bertino, S. Jajodia, and P. Samarati, Supporting multiple access control policies in database systems, *Proc. IEEE Symp. Security Privacy,* Oakland, CA, 1996, pp. 94–107.
7. E. Bertino, S. Jajodia, and P. Samarati, A flexible authorization mechanism for relational data management systems, *ACM Trans. Inf. Syst.,* 1998, to appear.
8. E. Bertino, P. Samarati, and S. Jajodia, An extended authorization model for relational databases, *IEEE Trans. Knowl. Data Eng.,* **9**: 85–101, 1997.
9. K. J. Biba, *Integrity considerations for secure computer systems,* Technical Report TR-3153, Mitre Corp., Bedford, MA, 1977.
10. P. Boedges, Quoted in "Air Force mounts offensive against computer crime," *Govt. Comput. News,* **8**: 51, 1988.
11. D. F. C. Brewer and M. J. Nash, The Chinese wall security policy, *Proc. IEEE Comput. Soc. Symp. Security Privacy,* Oakland, CA, 1989, pp. 215–228.
12. S. Castano et al., *Database Security,* Reading, MA: Addison-Wesley, 1995.
13. D. D. Clark and D. R. Wilson, A comparison of commercial and military computer security policies, *Proc. IEEE Comput. Soc. Symp. Security Privacy,* Oakland, CA, 1987, p. 184–194.
14. H. Debar, M. Becker, and D. Siboni, A neural network component for an intrusion detection system, *Proc. IEEE Symp. Security Privacy,* Oakland, CA, 1992, pp. 240–250.
15. D. E. Denning, A lattice model of secure information flow, *Commun. ACM,* **19** (5): 236–243, 1976.
16. D. E. Denning, *Cryptography and Data Security,* Reading, MA: Addison-Wesley, 1982.
17. US Department of Defense, National Computer Security Center, *Department of Defense Trusted Computer Systems Evaluation Criteria,* December 1985, DoD 5200.28-STD.
18. W. Diffie and M. Hellman, New directions in cryptography, *IEEE Trans. Inf. Theor.,* **22**: 644–654, 1976.
19. T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inf. Theory,* **31**: 469–472, 1985.
20. D. Ferraiolo and R. Kuhn, Role-based access controls, *15th NIST-NCSC Natl. Comput. Security Conf.,* Baltimore, MD, 1992, pp. 554–563.
21. D. F. Ferraiolo and R. Kuhn, Role-based access controls, *Proc. NIST-NCSC Natl. Comput. Security Conf.,* Baltimore, MD, 1993, pp. 554–563.
22. T. Fine and S. E. Minear, Assuring distributed trusted Mach, *Proc. IEEE Symp. Security Privacy,* Oakland, CA, 1993, pp. 206–218.
23. K. L. Fox et al., A neural network approach towards intrusion detection, Technical Report, Harris Corp., Government Info. Systems Division, July 1990.
24. T. D. Garvey and T. Lunt, Model-based intrusion detection, *Proc. 14th Natl. Comput. Security Conf.,* Washington, DC, 1991, pp. 372–385.
25. R. Graubart, On the need for a third form of access control, *NIST-NCSC Natl. Comput. Security Conf.,* 1989, pp. 296–303.
26. P. P. Griffiths and B. W. Wade, An authorization mechanism for a relational database system, *ACM Trans. Database Syst.,* **1** (3): 242–255, 1976.
27. K. Ilgun, R. A. Kemmerer, and P. A. Porras, State transition analysis: A rule-based intrusion detection approach, *IEEE Trans. Softw. Eng.,* **21**: 222–232, 1995.
28. S. Jajodia, P. Samarati, and V. S. Subrahmanian, A logical language for expressing authorizations, *Proc. IEEE Symp. Security Privacy,* Oakland, CA, 1997, pp. 31–42.
29. S. Jajodia et al., A unified framework for enforcing multiple access control policies, *Proc. ACM SIGMOD Conf. Manage. Data,* Tucson, AZ, 1997, pp. 474–485.
30. P. A. Karger, Limiting the damage potential of discretionary trojan horses, *Proc. IEEE Symp. Security Privacy,* Oakland, CA, 1987, pp. 32–37.
31. C. Kaufman, R. Perlman, and M. Speciner, *Network Security,* Upper Saddle River, NJ: Prentice-Hall, 1995.
32. C. E. Landwehr, Formal models for computer security, *ACM Comput. Surveys,* **13** (3): 247–278, 1981.
33. T. Lunt, Access control policies: Some unanswered questions, *IEEE Comput. Security Foundations Workshop II,* Franconia, NH, 1988, pp. 227–245.
34. T. F. Lunt, A survey of intrusion detection techniques, *Comput. Security,* **12** (4): 405–418, 1993.
35. C. J. McCollum, J. R. Messing, and L. Notargiacomo, Beyond the pale of MAC and DAC—Defining new forms of access control, *Proc. IEEE Comput. Soc. Symp. Security Privacy,* Oakland, CA, 1990, pp. 190–200.
36. J. McLean, Security models and information flow, *Proc. IEEE Comput. Soc. Symp. Res. Security Privacy,* Oakland, CA, 1990, pp. 180–187.
37. F. Monrose and A. Rubin, Authentication via keystroke dynamics, *Proc. ACM Conf. Comput. Commun. Security,* Zurich, Switzerland, 1997.
38. M. N. Nash and K. R. Poland, Some conundrums concerning separation of duty, *Proc. IEEE Comput. Soc. Symp. Security Privacy,* Oakland, CA, 1982, pp. 201–207.
39. National Bureau of Standard, Washington, DC, *Data Encryption Standard,* January 1977. FIPS PUB 46.
40. F. Rabitti et al., A model of authorization for next-generation database systems, *ACM Trans. Database Syst.,* **16** (1): 88–131, 1991.
41. R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Commun. ACM,* **21**: 120–126, 1978.
42. R. Sandhu and P. Samarati, Authentication, access control and intrusion detection, in A. Tucker (ed.), *Database Security VII: Status and Prospects,* Boca Raton, FL: CRC Press, 1997, pp. 1929–1948.
43. R. S. Sandhu, Transaction control expressions for separation of duties, *4th Annu. Comput. Security Appl. Conf.,* Orlando, FL, 1988, pp. 282–286.
44. R. S. Sandhu et al., Role-based access control models, *IEEE Comput.,* **29** (2): 38–47, 1996.
45. R. S. Sandhu and P. Samarati, Access control: Principles and practice, *IEEE Commun.,* **32** (9): 40–48, 1994.
46. O. S. Saydjari et al., Synergy: A distributed, microkernel-based security architecture. Technical Report, National Security Agency, Ft. Meade, MD, November 1993.
47. C. E. Shannon, Communication theory of secrecy systems, *Bell Syst. Tech J.,* **28**: 656–715, 1949.

48. H. Shen and P. Dewan, Access control for collaborative environments, *Proc. Int. Conf. Comput. Supported Cooperative Work,* 1992, pp. 51–58.

49. D. J. Thomsen, Role-based application design and enforcement, in S. Jajodia and C. E. Landwehr (eds.), *Database Security IV: Status and Prospects,* Amsterdam: North-Holland, 1991, pp. 151–168.

50. T. Y. C. Woo and S. S. Lam, Authorizations in distributed systems: A new approach, *J. Comput. Security,* **2** (2,3): 107–136, 1993.

PIERANGELA SAMARATI
SRI International

SUSHIL JAJODIA
George Mason University

DATA STORAGE. See DATA RECORDING.

DATA STRUCTURES. See ALGORITHM THEORY.