MIT/LCS/TM-135

TIMESTAMPS AND CAPABILITY-BASED PROTECTION IN A DISTRIBUTED COMPUTER FACILITY

Rosanne H. Wyleczuk

June 1979

TIMESTAMPS AND CAPABILITY-BASED PROTECTION IN A DISTRIBUTED COMPUTER FACILITY

ROSANNE HAZEL WYLECZUK

March 1979

This research was supported in part under NUSC No. A70240, "Computer Technology Review," Principle Investigator: Dr. J.C. Lamb (code 314), project element No. 62766N, subproject/task No. ZF61-112-001, "Independent Exploratory Development - Target Surveillance," Project Manager: J.H. Probus, Naval Material Command (code MAT08T1).

Massachusetts Institute of Technology

Laboratory for Computer Science

Cambridge Massachusetts 02139

TIMESTAMPS AND CAPABILITY-BASED PROTECTION IN A DISTRIBUTED COMPUTER FACILITY

by

ROSANNE HAZEL WYLECZUK

Submitted to the Department of Electrical Engineering and Computer Science
on February 26, 1979 in partial fulfillment of the requirements
for the Degree of Bachelor of Science and
for the Degree of Master of Science

Abstract

This thesis investigates the problems of supporting security requirements and providing protection mechanisms in a distributed computer facility. The nature of the environment necessitates examination of operating systems, data base systems, and computer networks. The capability approach to providing protection in a centralized system is chosen as the foundation for the protection mechanism of the distributed system.

The thesis also relies on an interesting approach to the representation of objects in a computer system. An object is represented by a sequence of immutable versions that represent the state of the object over time; each version is the result of an update on the object. This approach to describing objects provides the basis for a flexible definition of the world in which timestamps are naturally associated with every object in the system.

The development of a DCF capability mechanism resulted in the following discoveries: Capabilities need not become immediately effective upon their generation. It is not necessary that the object to which access is being authorized exist at the time the capability is generated. And, the revocation of access privileges and the control of capability propagation are not insurmountable problems even in a distributed environment.

Keywords: distributed computer systems, timestamps, capabilities, protection

Thesis Supervisor: Liba Svobodova
Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

I have been greatly influenced by many individuals during the course of this research. Lynn DeNoia, at NUSC, helped me through the adolescent period of this thesis and as they say, it is the most difficult. And thanks to Carol Scaletti, Jerry Lamb, and Bob Bernecky, a few of those who made the lab a pleasant place at which to work.

Of all the people who shared the experiences of this research, I must single out Professor L. Svobodova, my advisor, for the greatest patience and understanding, and the most constructive criticisms and useful contributions to the completed research. How she does it, I don't know.

I must also express my warm feelings for all of the members of the Computer Systems Research Group who made my short stay with them an experience -- a very pleasant experience. I am especially grateful for the friendship and patience of my office-mates, may they proceed to a speedy completion of their chosen tasks. And, thanks Allen, for everything.

A special note of appreciation is due to all those, and especially John and Lydia, who work to maintain and strengthen the Cooperative Program in Electrical Engineering and Computer Science. The program provided me with an excellent opportunity to continue my education at M.I.T. while also obtaining invaluable exposure to the non-academic aspects of my field.

Finally, and with the deepest feelings, I must recognize all my friends who made my M.I.T. experience as great as it was. Most especially WAl-NEL, RilyR. And, my mother, great as she is, one who would support me through another five years, thanks -- but, that's all folks. TGID.

CONTENTS

FIGURES

Chapter One

INTRODUCTION

As an organization grows and as its members and responsibilities increase in number, its control and proper functioning become more difficult and complex. Man was able to care for himself until his membership in society caused the establishment of many inter-dependencies and relationships. The complexity of control necessitated a distribution of tasks and resulted in the variety of organizations that the citizenry depend on for the satisfaction of their basic needs today.

Similarly, computer technology has grown in sophistication and now is a vital component in government, military, business, and educational organizations. The tasks that have been designated for these computer facilities have grown in number and have such diverse characteristics that the computer facility no longer functions efficiently in a centralized manner.

A large amount of research on computer networks and their use in facilitating resource sharing has greatly advanced data communications technology and control. Computer system designers can now consider building distributed operating systems and distributed data bases. This progression has been a natural one. First, the user at a site realized

that he need not physically possess all possible computing tools at that site in order to make use of them. Now it is evident that the computer, similar to organizations, need not generate, store, and later make use of its data (bases) in one centralized location. The combined technologies resulting from network and communications research enable the consideration of a realizable computer information utility or distributed computer facility (DCF).

To realistically consider the myriad of possible uses and users of such a utility one must consider the need for support of security requirements and protection mechanisms. Much work has been published and many mechanisms implemented to provide protection in a single computer system. Also, there have been extensive investigations of encryption schemes to provide protection of information flow in computer networks. Little consideration, however, of protection mechanisms dealing with access authorization in distributed systems exists. Yet, a means for providing protection in the distributed environment will be a necessary attribute of the DCF. Otherwise, few users of current computer systems would be willing to integrate their systems, or convert their systems, into those that could be connected to the computer information utility.

1.1 Overview

I will, throughout, use the terms distributed computing facility (DCF) and computer (information) utility interchangeably. However, there is a conceptual distinction that needs to be made at the start. I mean the DCF to refer to the technical design and features that are important when one connects many different computer systems to many other different computer systems. Each of these systems provides a variety of hardware and software tools; the remote resources are connected by a communication subnet.

The computer utility is meant to include all the users of these dispersed facilities, and all the different policies that control their use. Similar to the utilities that exist in society, the term implies an operation with a reasonable guarantee of dependable service. It also implies that different services are offered with fees being charged for those services. It is the human requirements for use and the human policy specifications that actually distinguish the utility from the conglomerate of machinery that makes up the facility.

It is also necessary to point out that the term "distributed" as used in this thesis does not simply mean the distribution of physical computing hardware but rather the distribution of processing logic, function, data, control, users or a combination of them. Basically, a distributed system may be considered to be an organization "of highly autonomous information processing modules, called nodes, which cooperate

in a manner that produces an image of a coherent system on a certain defined level" <LCS_78>. Base level operations and resources exist in each node to facilitate the processing and communication functions that each is designed to offer.

Before further discussion, it must be agreed that a primary goal of the distributed computing facility is to make itself available to the users who might like to take advantage of its capabilities and for whom its design was intended. In conjunction with satisfying the primary goals of the DCF, some means of satisfying the privacy and security requirements of its users must exist. Without the trust of the user the information utility will not be used to the full advantage of its capabilities.

The problem to be attacked in this thesis is that of providing for these privacy and security requirements in a distributed system. In an environment where the free exchange and the sharing of information is desirable, how does one flexibly control access to the system's resources and information? After appropriate access authorizations have been decided, how does one guarantee that access will only be permitted according to those decisions?

The protection mechanisms developed in this thesis draw mainly upon work in two areas: the work on protection mechanisms as used in centralized systems, and the work on the design of distributed systems, in particular, the notion of timestamps. Timestamps are used in

distributed systems to maintain the consistency of duplicated or distributed information. They are employed in many protocols designed to reliably synchronize the application of updates on a data base so that the data base, and all copies of it, are considered to be in a consistent state. In this thesis, timestamps will be used in conjunction with the capability-based protection mechanism to arrive at a protection scheme that is viable in the distributed environment. A capability, as defined for a centralized computer system, is an unforgeable ticket (name) which, when provided to a controlling entity, "can be taken as uncontestable proof that the presenter is authorized to have access to the object named by the ticket" <Salt75>.

A hypothesis maintained by many investigators of distributed systems is: A distributed but coherent (to some degree) computing facility yields 1)modularity and improved structural implementation that increase flexibility, 2)increased reliability and availability, 3)improved throughput and response time, 4)geographical distribution that facilitates the maintenance of local autonomy, 5)resource sharing and reduced material investments that reduce overall cost, 6)enhanced application oriented systems, and 7)expandibility and maintainability. Any particular investigator may be driven by all, or a subset, of the above.

The design of the protection mechanism proposed herein must remain within the guideposts described above. To maintain the desired local

autonomy and reliability, it is not possible to use a central security node that would be crucial to the correct operation of the overall protection mechanism. Similarly, the protection mechanism should not require repeated verification of a user's authorization otherwise the response time may be considered unreasonable. Essentially, the operation of the protection mechanism in the distributed environment must be equivalent, in terms of efficiency, flexibility, and understandability, to the operation of similar mechanisms in centralized systems.

1.2 Protection

Protection is crucial to operating in any environment, but is essential to operating in a distributed environment. It must be assumed that the environment holds many threats to the continued operation of the computing facility; it is full of unknowns. In general, although any node may support a set of cooperating and mutually trusting users, it cannot be assumed that any site to which data is sent will enforce the same security policies as the site that normally holds the data.

At any node there might exist uneducated users, miseducated users, ignorant users, and/or malicious users. The threats to the system may be direct threats to its physical existence, or indirect threats via the clandestine introduction of security flaws into any of the system's supporting elements. The threats may be overt or covert; they may be

actively disruptive, or they may take the form of a passive absorption of information.

Each node of the system must provide for the correct identification and authentication of its users. The work on public encryption schemes and signatures <Kent76, Rive78> has been extensive. This research provides a means of reliable identification of communicating entities despite their remote locations, and the physical exposure of the communications medium.

Many of the other controls that exist to provide security in a centralized facility need to be provided in the DCF also. Physical security, personnel security, and administrative security need to be seriously implemented at each of the participating computer systems (nodes). Some physical security may be inherent in the structure of the DCF. For instance, at any one site a back-end computer for data base (DB) processing could screen data base access requests. In such a situation, no application programs could execute on such a dedicated back-end machine; no malevolent program monitoring of DB activity could take place.

In general, there are three potential ways in which the security requirements of a computer system may be violated. Violations may occur by

1) unauthorized information release,
2) unauthorized information modification (or manipulation), or
3) unauthorized denial of use.

The protection mechanisms, whether directed by the person who owns the information or by the system, must prevent these violations.

It should be realized that, for the distributed system, a modification to the approach taken in the design of the protection mechanisms internal to a tightly coupled computer system is necessary. The difficulty is apparent if one considers the crucial passing of protection information from one processor to another, thereby exposing the information to interference from other network members or complete outsiders. Also, in a centralized computer system, the user accessing an object could be identified, at the very least, by an exhaustive search of all those permitted to access the particular object. In a distributed system, however, assuming the capability approach, this is not feasible; one cannot ascertain the extent of the propagation of a capability. Therefore, if such an approach as the capability scheme is to be considered, in which the identification of the subject accessing an object is not known at the moment of access, a flexible revocation scheme is necessary.

As well as satisfying discretionary user security requirements <Salt75>, membership in the network community must guarantee safety against the mischief and mistakes of other users. The network itself must specify and guarantee certain levels of protection. These requirements may establish what is known as non-discretionary access controls <Salt75>.

The distinction between discretionary and non-discretionary controls is critical to an understanding of protection issues. A discretionary control scheme provides the mechanism for an individual to freely decide and flexibly manipulate the access rights he has granted (or will grant) to other users, for objects under his control. The critical notion is the ability to alter the authorizations, dynamically, if necessary.

A non-discretionary scheme prevents the alteration of predefined and specific access controls. In the extreme case, the military security scheme, sensitivity levels are embedded in the authorization mechanisms to restrict the flow of information according to the assigned classifications of both users and objects. Karger <Karg77> has examined the issues relating to non-discretionary access controls in decentralized computing systems. He developed new mechanisms to deal with the decentralized administration of the lattice model (see chapter 2) and examined the related problems of decentralized computing.

There are two major approaches to providing discretionary controls in centralized systems. Access control lists (a list of users authorized for access to some object) and capabilities have been implemented to various degrees of sophistication, in several systems <Cohe75, Cook78, Engl74, Need77, Salt74, Schr75, Wulf74>. In attempting to adapt either of these schemes to the solution of protection problems in a distributed environment, several new problems arise. The capability scheme is chosen for investigation in this thesis because it is believed that, incorporated with timestamps, it provides a viable solution to at least some of these problems.

1.3 The Use Of Timestamps in the Security Context

Two notions underlie the mechanisms that will be introduced within. One, information can change in value depending on when it is released and two, information changes in value as it ages, that is, depending on its currency or time of last update (which is determinable via the timestamp). Therefore, security policies based on the progression of time can be useful. An underlying assumption is that, in a global sense and on a local level, everybody lives according to a schedule. This is guaranteed to some degree because the government enforces laws, such as income tax filing times, and the reporting of accidents; and one's job requires scheduling to accomodate the company's needs. In a practical sense, an ordering is required for the maintenance of organization. At

many different levels, one is always dealing with someone else's organization, either of their life or their business.

With this understanding, the other concept integral to this investigation, the timestamping of objects, can be introduced. A timestamp will serve as a unique identifier for an object while its semantic value lies in its representation of a moment in time. This moment in time may be the time at which the object was last accessed, the time it was last updated, or the time it was created. The scheme employed also provides one with the ability to distinguish between versions of an object and thereby grant and selectively revoke access to individual versions, without any global knowledge of the propagation of capabilities.

The implications of timestamping to this thesis are based on the association of timestamps with objects and the contents of the objects. The timestamping of objects provides an indication of the currency (age) of the values contained in those objects. It will be shown that the degree of security that is required for objects may sometimes depend on the age of their values. Independent of the time of object creation or update, the access predicates upon which authorization decisions can be made may employ other time values, e.g. the time, in a global sense, at which the request is being made may be critical.

1.4 The Thesis Plan

This thesis will integrate the timestamp concept with a capability-based protection mechanism to yield a viable discretionary protection scheme for a distributed computing facility that performs as a computer information utility. Part of the mechanism proposed is actually a further development of ideas found in several existing or proposed systems. It will be shown that the role of the timestamp in a capability scheme facilitates easy revocation of access privileges in a distributed system. In addition, the use of a "time" entity will enhance the choice of authorization criteria and provide an elegant means of limiting the period of usefulness of capabilities, if desired.

The next chapter considers the problems involved with providing protection in operating systems, networks, and data base systems. The coherence and cooperation between the different mechanisms is crucial. Chapter 3 will proceed to introduce the concept of timestamps as a useful protection mechanism. The dynamic nature of protection as a function of time is crucial to an understanding of the conceptual foundation for this investigation.

The characteristic operation and function, with respect to capability generation and use, of a typical node of the facility are described in Chapter 4. The prototype node employs the protection mechanism using capabilities and timestamps. Finally, Chapter 5 completes the thesis with a summary of the work presented and a consideration of those issues that deserve further thought.

Chapter Two

THE PROTECTION OF COMPUTING SYSTEMS

Before proceeding with an investigation of the problem of providing protection in a distributed computer facility, we must examine the elements of the facility and their attendant manifestations of the protection problem. An understanding of the total security problem is necessary before defining the security goals in computer systems. A protection model is described to provide the basis upon which comparison of different approaches to solving protection problems in computing systems can be made. The rules that are an important component of such models are described in depth. And, the notion of a public domain is introduced to facilitate whatever unrestricted sharing may be desired across a large and complex system.

After looking at those aspects of the DCF that are common to centralized operating systems this chapter includes a discussion of the motivations for providing security amongst nodes that include data base systems. The nature of the problems unique to data base systems are presented along with an analysis of the criteria that may be applicable to their solutions. It is only with the understanding of the issues presented in this chapter that a workable mechanism can be sought for operation in a distributed environment.

The question of whether protection mechanisms are really necessary in information systems can be answered in two ways. An examination of the legislation that has been enacted in the past few years, to protect an individual's privacy, provides one answer; the public concern is constantly growing. The other answer requires introspection; the value that individuals assign to information is what gives rise to different security goals. In the financial environment, for example, the value of information is very high and therefore the security requirements are very high. Substantial monetary gains could be realized upon penetration of an electronic payments system. The primary security concern is to prevent losses, especially through embezzlement or fraud.

Similarly, the concern for security varies in magnitude in medical information processing (a matter of life and death, not dollars), in the educational community (where the users are the most devious and ingenious), in service bureaus, and in the military. The degree of concern for security and privacy issues is an indication of the effort that will (has been) put into the design and implementation of controls. Also, the more rigorous the security requirements are, the less likely is an implementation of a discretionary control scheme. A discretionary scheme is a more flexible approach but, it is prone to more errors, or rather, misguided judgements.

2.1 The Total Security Problem

The provisions needed for the total computer security of a computing system cannot be determined until a careful examination of the operating (user's) environment has been made. Such a study would have to provide enough information to formulate a general set of security requirements. Input in four areas is necessary: the possible threats to computer services, the degree of sharing among users that is desirable, the complexity of the services that are to be provided to the user community, and a consideration of the value placed on the information and its integrity. Since the results of such a study are uniquely determined by the situation, the problem of providing protection should at first, and at least, be attacked at the most primitive and obvious levels. One need not have decided upon the policies that will control the facility before protection mechanisms are designed, provided that the mechanisms are sufficiently flexible to support implementation of a large class of policies. Primitive mechanisms can be employed to control access to the physical representation of information while more sophisticated policies control access at a higher level outside of the computer system.

A first, and minimum, attempt to provide for a particular system's security requirements should include administrative controls and some degree of physical security. Of the many breaches in security that have been recounted in Computerworld, the majority could have been prevented

had adequate controls existed in these two areas. Such controls are generally referred to as external protection mechanisms (external with respect to the computer system). The mechanisms that are dependent and integrated with the computer system are referred to as internal mechanisms. This chapter concentrates on the details of these internal protection mechanisms most of which have been provided in operating systems software and hardware.

An authentication mechanism prohibits the use of the system by unauthorized users by verifying the identity of the subject attempting to access the computing facility. The methods that can be applied to check a person's identity are numerous, but can be classified into three categories, those based on something the person knows (passwords), those based on something the person has (magnetic cards), and those based on something the person is (signature).

Threat monitoring and security auditing are possibilities for surveillance mechanisms. These mechanisms are provided because it is realized that whatever other protection mechanisms exist, they often have not been certified correct.[1] These other regularly used mechanisms may be incomplete or even ill-defined. It is presumed that the knowledge that the surveillance of users is customary may provide an additional deterrent to attack.

---

1. If one could certify all protection mechanisms fool-proof and with a mean time between failures (errors) approximating infinity then such surveillance mechanisms would be unnecessary.

The need to protect information transmitted in computer networks introduces a communications aspect to the security problem. The use of data encryption and decryption techniques is one of the most highly researched areas of the security field. In a computer network the communication lines are one of the most exposed elements of the system; they are the most susceptible to attack. In the first generation of computer networks, it was obvious that all protective efforts would initially be directed to the communications element. As data base services become a more common facility provided in computer networks, to facilitate the establishment of decentralized information utilities, the protection of information flow becomes even more crucial.

The communications medium represents a fruitful ground for an attempted solution of the total security problem. First, the defense against attack on this subsystem must be guaranteed to work correctly over all time. Second, encryption mechanisms do not interfer with the routine of ordinary users and remain transparent; the encryption process consists of transforming the sensitive information into another form and using that for transmission. In existing computer networks encryption and authentication may be the only forms of control that are provided. However, in the design of the second generation of computer networks, the feasibility of sophisticated, versatile, and flexible security policies lies in the implementation of access controls.

2.2 A Protection Model

The requirements of the protection mechanisms in a single computer system of a network node can be basically described in the same manner as most other protection schemes, that is, in terms of subjects, objects, and the access rights a subject has for any object. Lampson's access matrix is the most widely used approach to describing the access rights to objects <Lamp71>. Subjects are represented in the rows of the matrix while the columns provide the representations of the objects. The intersection of a column and a row indicates the access rights of the subject for that object. Permissible access rights may include READ, WRITE, EXECUTE, MODIFY CAPability, PASS CAPability, and/or ENTER permission into a protected procedure, among others.

The mechanisms used to provide protection in various existing systems include a non-discretionary lattice model approach <Denn76>, and/or the use of access control lists <Salt75> or capabilities <Fabr74, Rede74> to enable discretionary control (both of which can easily be described in terms of the access matrix). The lattice model defines sensitivity level attributes of the subjects and objects in the system. Access is only permitted if the subject has the appropriate clearance or level of security for the object in question.

Access control lists are associated with objects. They indicate those subjects (users) that may access a particular object and the specific access rights of those subjects. The capability mechanism

provides, logically, a capability list (C-list) which is associated with a subject and that describes the objects which that subject may access. However, capabilities do not have to exist in lists, rather they can be dispersed throughout the domain of the subject, e.g. embedded in other objects.

All of the above mechanisms have been clearly described and examined in the literature for the centralized case. To date, however, little research has been published on the problem of providing similar controls on access to resources in the distributed environment. The main reason for this is the lack of any realized or fully designed distributed computing facility. Much of the research on distributed data bases and operating systems is still centered around basic design issues such as consistency, organization, availability, and control.

The protection mechanism chosen for the individual node sites in the DCF is a capability-based scheme for discretionary controls. No performance statistics, in terms of the number of authorization checks or the degree of flexibility, or the response time to a request versus the storage costs incurred, can be presented in support of this choice, however, there are a few substantial reasons for the choice.

The first reason for choosing a capability scheme stems from the question of who is to bear the burden of the work in an authorization check. In an ACL scheme, for each access attempt on an object, the controller or guardian of the object must search for the existence of

the principal's[1] identifier on the ACL. In a distributed system, this list may be very long, and there may be many accesses to the same object, thus demanding much work from the guardian entity. A capability scheme forces a user to choose from amongst his capabilities, the one he wishes to present to the object's guardian. Now the guardian need only perform a simple comparison to decide authorization.

The notion of "ticket" passing affords easy propagation of capabilities in a network scheme. Tickets to control access are conceptually pleasing, and in the decentralized case, the passing of such objects appears natural. If the protection scheme can be made easy to use and understand, it will automatically be many times more effective than any other, more complex scheme.

Capabilities force access privileges to be associated with the users rather than with the objects of the system. Thereby, one can devise one's own security policies and apply them to other users by granting groups of users, all of whose members are similarly trusted, identical capabilities. An action taken to alter the privileges or effectiveness of the granted capabilities will affect the group uniformly, regardless of an individual's location or status.

---

1. A principal is an entity in a computer system to which authorizations are granted; the unit of accountability in a computer system <Salt75>.

2.2.1 The Rules

In addition to designing the representation of subjects and objects in the protection model, one must consider the specification of a set of rules that govern their interactions. In centralized computing systems, these rules may vary from one model to another since the choice of rules is at the discretion of the system designer, as is the choice of subjects and objects. Despite the fact that the prototype node system is designed to be included in a distributed computing facility, and that it must operate efficiently in that mode, I will grant that the choice of subjects and objects may still be made at each individual site. Any misunderstanding of subject or object definitions must be resigned between cooperating or communicating sites. However, some base-level set of rules that act as a protection mechanism protocol must exist.

One cannot, however, count on identical or even similar security policies being implemented at different sites (except between private nodes or in private networks). This introduces a distinction between security and protection. This distinction is emphasized throughout this thesis by the use of protection 'mechanism' and security 'policy'.[1] It is introduced here because one could argue that the notion of rules is more a policy than a mechanism issue and that therefore, one cannot assume uniformity throughout a decentralized system. I maintain that

---

1. A clear discussion of the policy/mechanism distinction can be found in the literature on the HYDRA system <Wulf74, Cohe75>.

the set of (protection) rules, or protocols, is necessary for the proper functioning of any (protection) mechanism between two distant and distinct communicators.

Lampson <Lamp74> includes, in his description of a protection model, four rules that govern the operations of one domain with regard to those actions that affect the access attributes (rights) of another domain for an object.

> a) -A- can remove access attributes from -B- if it has 'control' access to -B- (or -B-'s domain).
> b) -A- can copy to -B- any access attributes it has for -X- which have the copy flag set, <*>, and can say whether the copied attribute shall have the copy flag set or not.
> c) -A- can add any access attributes for -X-, to -B-, with or without the copy flag, if it has 'owner' access to -X-.
> d) -A- can remove access attributes for -X-, from -B-, if -A- has 'owner' <*> access to -X-, provided -B- does not have 'protected' access to -X-.

<*>: the use of 'control', 'owner', and 'copy flag' is not important to the following discussion and clarification may be found in the reference sited.

I propose a modification to rule (b) that embodies an explicit statement of an assumption that often is, implicitly, relied upon in designing protection mechanisms.

> b') -A- can copy to -B- any access attribute it has for -X- which have the copy flag set, and can say whether the copied attribute shall have the copy flag set or not, only if it has received from -B- a request for that particular access attribute to -X-.

The modification of rule (b), the explicit request requirement, assures that the possessor of a capability with particular access attributes for an object, does not, indiscriminately and without provocation, grant, copy, or pass a capability, specifying some subset of those attributes, to another subject. A request for the access attribute must be identical to the attribute actually granted.

An implementation of the above rule (b') would distribute some of the burden of authorization procedures onto the subjects; they must make explicit requests for capabilities to particular objects. Of course, some means of advertising the existence of a tool or feature (program, optional utility modification, etc.) to possible users could exist, but even then, one should wait until an entity makes an indication of its desire to use the object before a capability is granted.

The particular request from -B- may be implicitly represented by the existence of a distribution list. One can assume that each member on the distribution list had, at one time, made a request for the particular object to which access is being granted. For instance, all members of a committee may be on a distribution list for the minutes of the meetings and reports; indirectly, the members have requested these objects. And, if a subject is actually a component or element of another higher-level subject then it may or may not, at the discretion

of the higher-level subject, be automatically included on a "second-order" distribution list associated with the higher-level subject.

Second, to the control of distribution, a distribution list is an administrative technique of informing all the receivers of an object of the other subjects who have received that same object. And the distribution list procedure formalizes the process of record-keeping by the sender of an object, whether it be a capability or a data object. It is a means by which the sender can keep track of those users to whom he has directly passed access rights. Even if the capabilities are freely copyable, the distribution list provides at least a hint as to where in the DCF the capability was "seeded".

2.3 The Public Domain

Every computer system has certain tools available to all users. There are certain utilities such as a file editor, a directory manager, a debugger, a file transfer program, information directories, query programs, etc., that make it easier for a new user to begin functioning within the system. Similarly, large-scale software projects provide libraries for use by all project team members. These libraries consist of routines and data bases that are particular to a project and whose availability eases the tasks of all members. And, as in the case of

the system utilities, common availability of these objects reduces the storage requirements of all those involved.

This concept of providing unlimited and uncontrolled (except for modify (WRITE) privileges, perhaps) access to certain objects must be more clearly defined and broadened in the information utility. We will rely on the common usage of "public domain" to assume the existence of a "public domain entity" that is accessible to all. The idea here is to provide a means for unrestricted sharing of information across node boundaries that is easy and that does not require the attention of the "creator" of the information. In an environment that is as highly uncertain and variable as the one that is assumed for the DCF, the public domain relieves the burden of continual review of access requests for some subset of the objects in the system (the subset that has been defined to be in the public domain) from the controlling entities.

The DCF will, more likely than not, also maintain a number of copies of what might be considered a large-scale facility library. This library will be maintained by the facility personnel. The public domain, however, must be maintained as a separate (for administrative reasons) and distinct entity that is as easily accessible and usable as the system's library. It is, in essence, a manifestation of the DCF "users group". Those objects that a user feels would truly be valuable to a large community can become part of the public domain after they have been clearly annotated. The creation of all objects in the public

domain are credited to the donator with the understanding that he has absolved himself from any further control over the object, and/or responsibility for its actions. The public domain is distinct from all other repositories in that anyone in the DCF community may donate objects to it and then absolve themselves from any associations with or responsibilities for the objects.

The fundamental point to be made by the inclusion of the public domain in this discussion is the notion of a repository for objects that are freely accessible and freely copyable, without the need to grant privileges separately to each interested individual. The existence of an entity of this nature prevents a user from becoming too burdened by the explicit request requirement; he may donate a copy of his object for use by the community at large. Once donated to the public, privileges to the object are no longer revocable, nor is there any longer a need for the original owner to keep tabs on those subjects accessing it. Effectively, all privacy rights to the object have been sacrificed. Of course, there can be no modify privileges granted to anyone for any of the objects resident in the public domain. And the DCF administration should periodically clean up the public domain if some objects do not appear to be useful to the community.

2.4 Inter-Node Vs. Intra-Node Perspectives

It has been decided that the mechanism internal to each computer node of the DCF will be capability based. We will refer to this as the intra-node mechanism. In a distributed system, one must also consider whether the protection mechanisms that guide the nodes internally will guide their interactions with each other. This component of the overall protection design will be referred to as the inter-node mechanism <DSG Progress Report78>.

This is similar to the notion of partitioning an operating system into levels <Schr75>. Schroeder had previously suggested, for instance, that a bottom level implement non-discretionary controls while a top level implement discretionary controls, constrained, of course, by the bottom level. This means, in the current context, that the following possibilities exist:

	intra-node controls	inter-node controls
1.	Non-discretionary	Non-discretionary
2.	Non-discretionary	Discretionary
3.	Discretionary	Non-discretionary
4.	Discretionary	Discretionary

Since we have decided upon a capability-based mechanism for the intra-node mechanics, we have limited ourselves to either approach (3) or (4).

The choice of control for inter-node communications is entirely environment dependent. The world in which the system is to operate, the

functionality demanded of it, and the required interfaces with other systems will dictate the nature of the inter-node controls. For instance, there are two regulations that would guide a typical electronic payments system to dictate that alternative (3) be implemented. First, banks are regulated by what information they can pass between them, so the network communications routine knows what to expect; it is a highly regulated business. And secondly, considering a broader system boundary, countries carefully control what financial data is allowed outside of their borders.

It must be clear that no definite decision can be made here. This decision is an even more application-dependent one than that which decided on discretionary controls for intra-node control. My hypothesis is that the incorporation of timestamps and capabilities will make the capability scheme useful throughout the DCF. In the next chapter the integration of the two concepts is initiated and it is shown that the resulting mechanisms have increased in power over the original capability scheme. Before such a claim can be supported, however, we must look at several protection problems that arise in data base systems the solutions of which may be facilitated by the mechanisms to be proposed.

2.5 Data Base Systems

Data base (DB) systems are an indispensible ingredient of information utilities. Presently, much of the computer processing in organizations consists of acquiring, processing, and storing data for future reference, organizing and generating summary reports, and processing data in response to queries brought into the computer system via the data base management system (DBMS). Notice that the computer is no longer considered a tool solely for its computational powers ("number crunching"). The DBMS controls and manages the DB of the computer system; the DBMS therefore, controls the critical information resources of an organization. It is in the DBMS then, that one is likely to find more of the mechanisms that implement the required security policies.

It is also likely that many of the security requirements that will be specified will be closely tied with the data base processing function of the computer. Access rights dependent on values being accessed, access rights dependent on possible virtual information[1] accessed, access rights dependent on previous accesses, and access rights dependent on possible future accesses are all possible specifications for a systems' security procedures and therefore, some subset at least, should be realizable through the protection mechanism of the DCF.

---

1. Virtual information is information that is not physically stored in the DB but that can be derived from other stored data.

Broader criteria, used for deciding authorization, are essential to the successful modeling of an application. The variety of access requests will be richer, e.g. along with the simply DATA$READ, and PROCEDURE$EXECUTE, there will be NUMBERS$MAX, EMPLOYEES$COUNT_OF, BOSSES$GIVE_RAISE_TO, etc. Yet, for all the enhancements to existing mechanisms sought to better model the application environment, the DBMS will probably still be accessed according to constrained interfaces: query languages, special procedural languages, special protocols, etc. If more protection functionality were embedded in these interfaces they would not appear to be simply overhead.

It might even be desired that access rights be effective dependent on the current clock time and the time of creation of the accessed object, i.e. depending upon the age of the object. Such a policy for controlling access to objects directly models the value (importance) of information that is initially secured, as a function of time. The importance of certain data naturally diminishes in magnitude as time goes on, thereby appearing to behave according to an inverse relationship between the degree of protection required for a data item and the currency of its value. This relationship results in dynamically changing protection requirements.

The investigation of data bases includes such objects as the subschema (or views) of the data base, that is, that part of the entire data base that is logically associated to form any particular user's

perception of his subset of the entire data base. The DCF is now confronted with providing protection for such (abstract type) objects. The solution may be a simple one, if one considers the defining mechanism to be a protected procedure, or a secured extended type representation both of which would be accessible via a capability <Rede74>.

It is not intended that this section present all the protection problems inherent in data base systems, or all the problems that can be handled by the mechanisms found in this thesis. Yet, the mechanisms are extensible and flexible enough to provide for the initiation of a number of security policies that are rooted in the desire to protect large data base systems. Also, an information utility, by its very nature, embodies the problems inherent in DB systems and therefore, it is crucial for one to clearly understand the intricacies and subtleties of information protection, which is highlighted in data base systems, as well as the problems of data protection described earlier.

2.5.1 The Nature of the Data Base Problem

We are assuming a DBMS that runs as a subsystem along with the operating (sub)system, and therefore, it is not intended that the DBMS bear the full burden of implementing security controls independent of the controls traditionally provided in operating systems. The protection mechanisms provided by the DBMS could include those that take

advantage of the other functions performed by the DBMS. For instance, access controls that depend on the values of the accessed data items are more efficiently implemented in the DBMS than in the OS. Nevertheless, from the previous chapter and our knowledge of data base systems, some differences between the manifestation of the protection problem in operating systems and in data base systems are perceived.

One of the most obvious differences in the problem manifestation is that the operating system is concerned with the name and address spaces of the data (the actual memory locations), while the DBMS is concerned with the semantics of the data. Each user will have a different conceptual view of the data objects and their relationships. Both this and the heterogeneity of data representations across computing systems, require that the specifications for protection be in terms of logical entities; these logical entities will have to reflect the semantics of the application. This is very different from an operating systems orientation to physical data representations in memory (segments of bits). Content-dependent and context-dependent access controls (described in one of the next sections) could directly relate the desire to provide semantic meaning to authorization decisions.

In the same vein, operating system data items correspond to real objects or resources, i.e., segments, files, devices, routines, etc. However, in a DBMS there may be frequent definitions of new aggregate-type objects that need protection. These aggregates do not

correspond to physical entities but rather, they are dynamically constructed in response to particular queries. That is, the processing of a data base transaction[1] may result in the creation of a number of temporary aggregates several of which may be required to provide a complete response to the query; these aggregates may subsequently become permanent entities in the DB. For instance, in a relational data base, a query may require the execution of a number of primitive operations (JOIN, PROJECT, UNION, etc.) each yielding a temporary relation that can be named and thereby become permanent, if desired.

In an operating system one checks authorization at each access of a data item (file, segment, memory location). However, in data base systems, one has a choice between checking the subject's authorization to perform the desired query or its authorization to access each primitive data item in response to the query. The former approach, in a sense, treats the transaction as an object; the only privilege on such an object is an execute privilege. This approach implicitly relies on the data base schema (view) structure to limit, in a non-discretionary manner, the scope of a subject's influence. Alternatively, a transaction involves many accesses to distinct data-items to satisfy the query. Each of these accesses may have to be individually authorized

---

1. A data base transaction may be considered to be a mapping of one set of data base values, themselves mapped to the data items contained in the DB, to another set of data values. The transaction consists of a set of primitive operations on or accesses to the individual data items that make up the DB.

for the particular data items accessed. The granularity of the object, in the terms of the protection model, is critical to such a decision.

An even more difficult security problem particular to data base systems is the protection of derived or virtual information. This type of information is not stored anywhere in the data base. For instance, given the value of a person's date_of_birth, a user of the data base, given sufficient processing abilities, can derive the value of a data item called age using the accessed data item data_of_birth and the wall (system) clock. This is the most often cited example of virtual (derived) information; others (not so obvious) do exist. For instance, given the internal financial reports of a company, one can determine information that ordinarily would only appear in an obscured form in yearly financial statements made public due to Securities and Exchange Commission regulations. This information, if obtained earlier in the year and delivered to the company's competitors, could be very damaging to the company from which the information was taken, and even to the economy as a whole. The protection of such information is an interesting problem but one which will require the integration of a large knowledge base that is particular to the data stored in the data base and its area of expertise. Even then one cannot be sure that one's representation of the specific area of expertise is complete or adequate enough to detect when such inferences may be drawn.

The above problems depict situations in which some form of "context-dependent" control is needed. This form of control allows one to cite the presence, or lack, of specific combinations of data in the pattern of access requests, as a condition for access. The pattern of accesses of only the current session may be involved or all previous access by the given user may be involved. The later is similar to history-dependent controls where the history of a subject's interactions with the DB is maintained to prevent the derivation of virtual information over time (besides, the history can also act as a security log). It must be realized, however, that the degree of sophistication of the context-dependent controls can have a great affect on the performance of the overall DBMS.

2.5.2 Approaches to the Data Base Problem

Earlier we discussed discretionary access controls in which users have arbitrary rights defined with respect to the objects of a system. These rights are granted according to the personal criteria of the controller of the desired object. Four approaches in the implementation of this type of control exist and they follow:

a. Control by object: Access to an object depends only on the object requested, not on the requester, e.g. an object resident in the public domain is accessible to all.

b. Control by object and subject: Access to an object depends upon the object and the requesting subject, e.g. the

password file may only be accessible to the data base administrator.

c. Control by object and type of access: Access to any object makes sense only if the requested access is of some limited variety, e.g. one can not print to a magnetic tape drive.

d. Control by subject, object, and type of access: This is similar to the access matrix devised by Lampson, as discussed earlier. Here a subject can have the right to access an object in certain precise ways.

Now, having introduced data base systems, we must consider any additional criteria upon which access predicates may be devised and upon which the authorization for access will be decided. Data base systems, by their very nature, permit "data-dependent" controls. In such a mechanism access is conditioned by a predicate whose truth depends on the contents of some data item. The data item may play any one of the following roles in the system:

i. Event-sensitive: Access would rely on the proper value of some system variable, e.g. access could only be possible during office hours, whatever they may be.

ii. Value-sensitive: The access decision is based on the current value of the data being accessed (also referred to as content-dependent control).

iii. State-sensitive: Here the access decision is based on the dynamic state of the DBMS. For instance, a user may access certain files when the DBMS is not heavily loaded.

Incorporated into capabilities, timestamps will facilitate discretionary controls that depend on the occurrence of some event. This event is never fully defined but rather is identified from amongst an infinite series of such events; i.e. one must identify discrete points in time, from the continual passage (flow) of time, at which access is possible. When an access is attempted the current value of the system clock will provide a value that defines a discrete event. An access decision can now be made using the value in the evaluation of the criteria relevant to the particular request. Alternatively, as will be shown, the above description, given a different model of the world, could lead one to conclude that the access was value sensitive.

2.5.3 The Granularity of Data Items

A large subset of the possible access control mechanisms that one could implement in a data base management system have been considered. Yet, we have avoided the issue of the actual implementation

possibilities for subjects, objects, and access rights, in the traditional sense. A brief discussion of the possibilities for the implementation of objects will be presented here for completeness. However, a recommendation for a particular implementation is not attempted because of the many factors involved, such as the requirements of the application, the desired efficiency of the system, the desired performance, etc. (The terminology associated with relational data models will be used simply because it is easy to use.)

A Data Item = A Relation: In this case, authorization for access to an entire relation is required to access any individual data item in the data base. If a subject can access the relation he can access any row or field of the relation. This scheme appears to lead to an overclassification of data due to the large size of the classified items. The large object size and the existence of irrelevant data in the object granule can also lead to decreased performance, depending upon the scheme used to maintain data base consistency. Also, the approach permits access to data that was not specifically requested by the subject.

A Data Item = A Row: In this case, a subject can be authorized to access each row of the DB individually. A subject is either granted or not granted access to the

entire row, and all of its field entries. This scheme is subject to the same type of problem as the previous one only to a lesser extent. It also introduces the possibility that users may infer the contents of the data base if certain rows have volatile access predicates. For instance, if a ship alternates between carrying highly classified cargo and unclassified cargo, then subjects that are not trusted enough to have free access to this data may infer information about the ship's cargo on a particular day by the existence or nonexistence of the relevant row in the set of rows accessible to these subjects.

It is conceivable that the DBMS might have to supply lies to the users who do not have free access to the entire relation. The procedure used to accomplish this would have to tell the truth to the more trusted subjects. This, however, moves away from the notion that the data model is actually modeling the real world. One is changing the operating semantics without telling the participants. But, this is a peculiar exception, and in general, the decision to use rows of relations, as objects, is reasonable provided that the data values are not very volatile and that authorization decisions do not depend on those values that are.

A Data Item = A Field: In this instance, access conditions to be met are defined on each field in the data base. It is now possible to grant to a subject a partial view of each row of a relation. This approach clearly provides the greatest flexibility. If the granularity of an object is a data field then the implementation of any of the previously mentioned authorization schemes is more straight-forward, in the sense that any data value may be used in the access control predicates.

In general, as the granularity of the unit of data for which access controls are specified grows smaller, the degree of permitted sharing becomes greater, i.e. by decreasing the granularity of the item to which access is prohibited (making the object size smaller), one has provided for a potentially greater degree of sharing. Rather than prohibit access to an entire relation to prevent the access of one row, in a system in which the granularity of items is as fine as single rows, one need prohibit access to the particular row only. Essentially, needless prohibition is decreased. Also, as the items become smaller, there is more room for concurrency of operations, i.e. given a relation where the individual rows are independently controllable, any number of the rows may be accessed simultaneously. From a different perspective, however, if the granularity of the accessible item were as large as the entire relation, and one were authorized access to it for the particular

purposes of accessing a single row, one would also have gained access to the rest of the relation.

On the other hand, as the granularity of data items becomes smaller, the complexity of the implemented control mechanisms grows. The increase in complexity results in an increased likelihood that the control mechanisms are implemented with logical errors. If one is not assured of operating in an environment where reliable controls exist, one would tend to operate in isolation. We have now come full-circle and have reached a dead-end since isolation is not conducive to sharing and it is to increase the potential for sharing that one originally forced the granularity of the data items to be small.

2.6 Summary

In this chapter we have chosen the capability approach to providing protection in centralized computing systems as the basis of the mechanisms to be used in the DCF. We have introduced the use of distribution lists, an explicit request requirement, and a public domain entity to facilitate sharing and ease the coordination and cooperation of subjects that may coexist in a highly dispersed and variable environment. And we have made a preliminary examination of the protection mechanisms that should exist amongst nodes as opposed to the mechanisms that operate internally within each node.

Finally, we examined several issues associated with the presence of data base systems in the DCF. The most interesting result of this investigation is the identification of three classifications of criteria that may be used in deciding access authorizations: event-sensitive, value-sensitive, and state-sensitive. After consideration of the mechanisms that will be described in the next chapter we may see that the use of timestamps provides a mechanism that can be considered event-sensitive or value-sensitive.

Chapter Three

TIMESTAMPS AND CAPABILITIES

In the last chapter we touched upon several aspects of the protection problem in operating and data base systems that could be merged into a coherent mechanism for operation in a dispersed and distributed environment. Before actually describing the DCF mechanisms a new approach to object management is discussed. This approach, first used by Reed <Reed78>, provides a view of the world that lends itself to an interesting technique for the solution of protection problems. The chapter proceeds with the definition of capabilities in which timestamps provide the values for several fields. And finally, unique mechanisms are introduced for granting access privileges and for revoking access privileges.

3.1 A Perception of the World

The nature of the representation of the individually and independently identifiable entities of the system is critical to a user's perception of the operating world. An object is recognized as a named entity that stores information. The information, by being identifiable as an object, can be more easily manipulated by programs and people than if it did not have a name. A broad classification of

computer system objects, based upon the possible ways of manipulating the objects, might yield the following types: data objects (physical files, logical files (views)), communication objects (logical ports, message queues), transaction objects (encapsulated or parameterized programs which perform operations on data and communication objects).

The existence of objects is usually assumed to have resulted from explicit creation operations. An alternate view is taken in the dissertation of Reed <Reed78>. This thesis uses Reed's approach although the motivation is different. It is assumed that each update of an object's value warrants the resulting value be attributed to a new object (a descendant). That new object has a kinship with the object upon which the update was applied, yet it is different. The value of the new object is different, and it is that value that is the essence of the object's state. Amongst all objects of a given type (types more refined than those cited above), their value or contents is that which requires unique identification or naming. An update then, changes the state of an object to yield a new object logically related to the first but implicitly created (born) at the time of the update operation. An explicit CREATE operation supplies the primeval ancestor for the group of related objects.

What I have been describing is a relationship between objects that exists because each represents a different "version" of one logical concept. Whether this logical concept is itself an object, is an

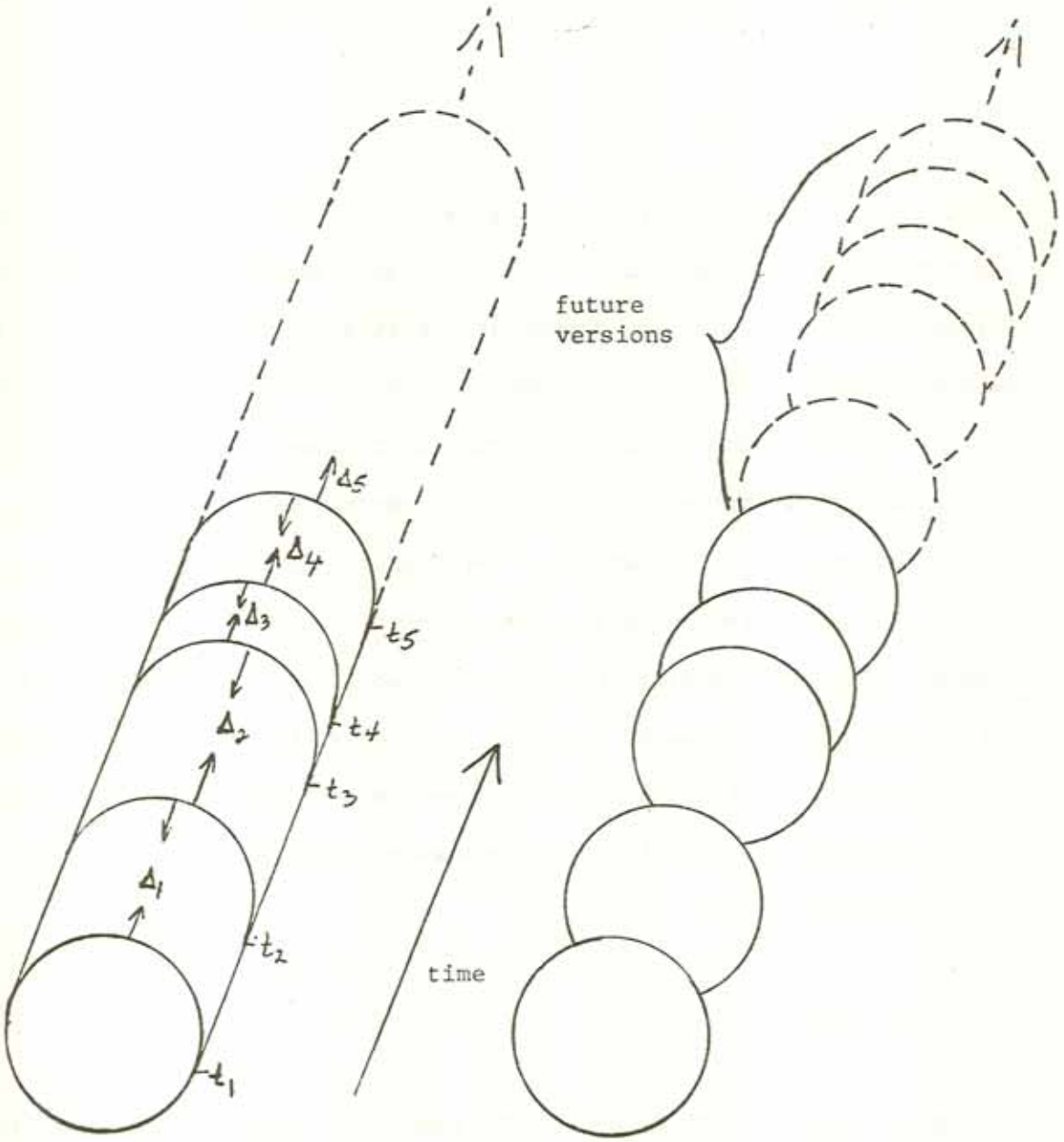
interesting question. The logical concept provides a complete description of some information but does not supply values for that information. It is an important consideration in the protection mechanism design because it is possible one might like to grant access to a logical concept while limiting that access to particular values that represented that concept's state at certain points in time, i.e. to particular versions of an object. For example, one could permit access to only certain versions of a document. For now, let the single logical concept refer to a family of physical objects, all of whose members are related because they are of the same type, exist for the same purposes, and are referenced with the same object name. Each object represents the value that the logical concept posessed at a particular point in time, as in Figure 1.

3.1.1 The Naming of Objects

Reed suggests that two part names be employed to refer to a particular member of the group of objects that form an object history.[1] The first part consists of an object identifier (called an object

---

1. An object history consists of a sequence of versions of an object distinguishable because any two consecutive versions differ in value as a result of update operations performed at distinct points in time. In Reed's work, the object history is actually the logical concept described above rather than an object itself. The object history cannot be an object simply because its value is not defined. More precisely, its value is not bounded until the system ceases execution. Until all system operations have halted, update (or create) operations on objects will result in additions to the group of objects that form the object history.



A representation of
the logical concept
≡ An Object History

The objects that represent
an object history
(the manifestations of
the concept over time)

t_n = time at which version is created
Δ_n = length of time for which version "n" is the most current version
t_5 = current system clock value (TE = time of environment)

Figure 1: An Object History

reference) and the second part consists of the information necessary to select the desired member of the object history i.e. a particular version. Reed introduces the concept of "pseudo-times" for the second component of the name which, when combined with the "object reference" (identifier), yields the "version reference" that identifies the precise version of the object desired.

With respect to the DCF protection mechanism, a logical name (object reference) will be sufficient to obtain an object history. A version number may be included at this (user-interface) level to identify, for the user, the particular version of the object accessible to the user, but it need not exist for actually accessing the object. At a lower level of naming, a timestamp (a time value read from the local site clock) will suffice to uniquely identify a version of any object in physical space; a site identification number will be appended to the timestamp to provide unique identification in the distributed system. The timestamp will be the value of the local clock at the time the object is created, either explicitly by the issuance of a CREATE operation or implicitly by an update operation applied to an already existing object. When an object is accessed, it is this timestamp that will be used to locate the object.

The timestamp may be considered an extension of the object reference, required by the user, to uniquely select an individual version from the object history, although the timestamp alone is

sufficient for unique identification. It is possible to consider the logical name as identifying the concept and the timestamp as identifying manifestations of that concept over time, that is,

object reference (ORef) = logical name

while,

object reference + version information, at one level

= unique object identifier (OID)

= timestamp, at a different (lower) level

For example, an object reference might be,

ORef = A.B.federal-tax-routine

(where the (.) period separates different parts of a path-name).

Now, to select the proper routine from the number that have existed over the years, the version information must be appended. Given the value for ORef above and a user supplied version number, the tax routine for a particular year may be uniquely identified from all those routines that exist in the object history for ORef. A unique object identifier might then be,

A.B.federal-tax-routine.9,

where the "9" represents a version number and the entire name refers to

the federal-tax-routine for 1976.[1]

The routine for 1976 may have been selected from the following object history. The sequence that forms the object history might consist of those objects that are the routines for computing taxes for those years in which the tax forms or regulations were changed from previous years. The object history of the logical concept "A.B.federal-tax-routine" consists of six objects named, as below, with a version number:

Object History (A.B.federal-tax-routine) :=

user-level names

A.B.federal-tax-routine.1	(for 1956)
A.B.federal-tax-routine.3	(for 1959)
A.B.federal-tax-routine.4	(for 1959)
A.B.federal-tax routine.9	(for 1976)
A.B.federal-tax-routine.12	(for 1978)
A.B.federal-tax-routine.13	(for 1979)

In the above, an object history is depicted in a simplified way. It is intended that, at the current value of the system clock (TE = time of the environment), the object history specified by the object

---

1. The date information is not actually part of the object identifier here, but it could easily become part of the identifier, as will be shown.

reference, ORef=A.B.federal-tax-routine, consists of versions 1,3,4,9,12, and 13. For any number of reasons, versions 2,5,6,7,8,10, and 11 have been eliminated.

At a lower level in the system naming hierarchy, the above unique OID might be equivalent to

1946 060 1976,[1]

a timestamp representing the time at which the ninth version of the federal-tax-routine object was created.

The same object history is repeated below; this time timestamps are used to name the different versions of the object in the object history named by ORef, as shown in Figure 2. It should be realized that the above listing represents the user-oriented higher level names; each of the names uniquely identifies an object for the user. The lower level, machine oriented, unique identifier consists of the timestamp whose value is the creation time of the object.

---

1. For now, let the first four digits represent an hour and the minutes of a day, let the next three digits represent the day of the year, and let the last four digits represent the year; the spaces exist for readability only.

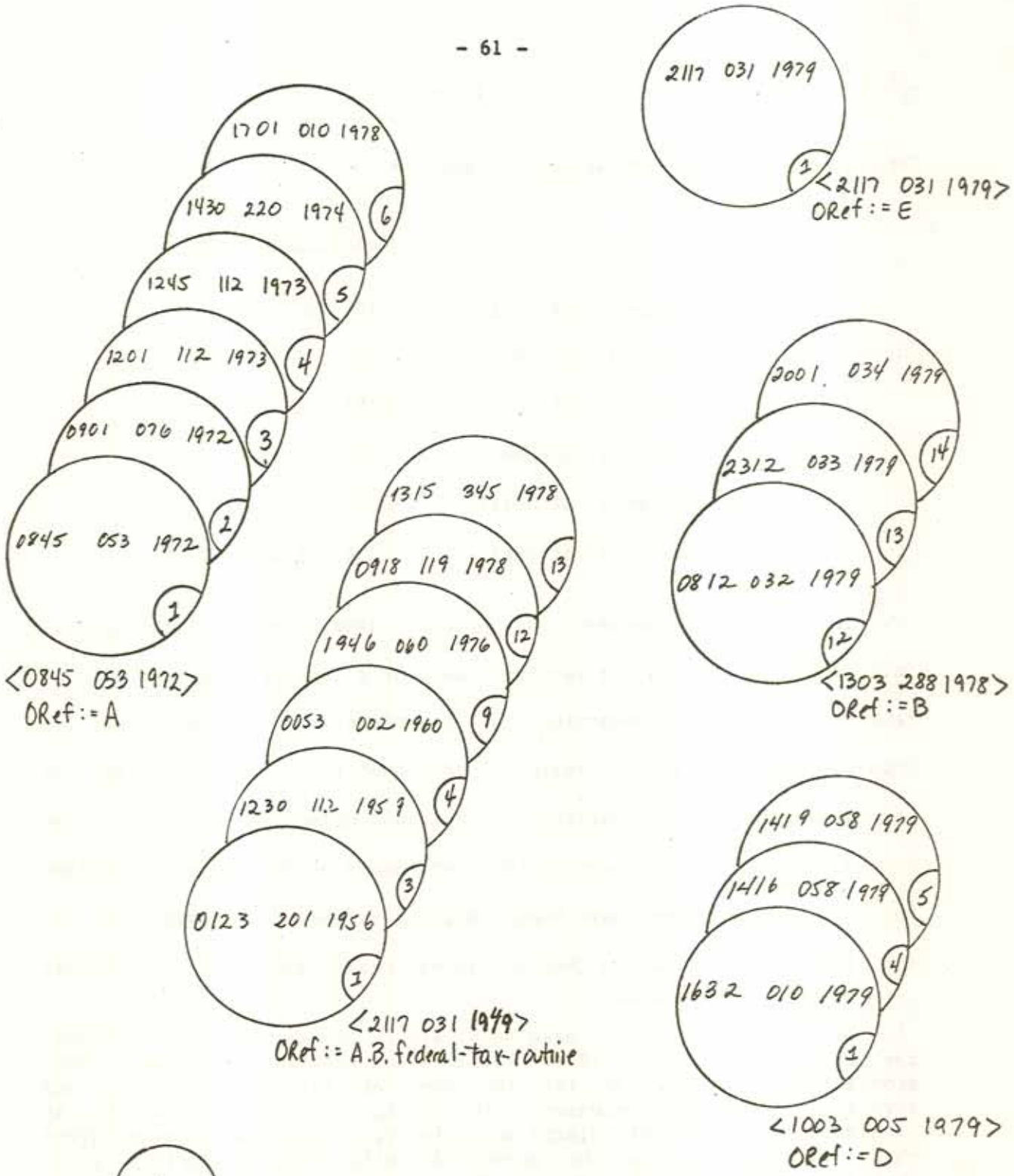


Figure 2: An Example of Several Object Histories

Object History (A.B.federal-tax-routine) :=

	machine-level names
(version 1 of ORef)	0123 201 1956
(version 3 of ORef)	1230 112 1959
(version 4 of ORef)	0053 002 1960
(version 9 of ORef)	1946 060 1976
(version 12 of ORef)	0918 119 1978
(version 13 of ORef)	1315 345 1978[1]

In this last sequence of object identifiers, we lost explicit information concerning the version number of a particular object but we gained information concerning the object's time of creation, that is, from the timestamp we can abstract a date that is the creation time. It should be realized that version 3 (the second object in the list) and version 4 (the third object in the list) could be objects with the same value. Version 3 might have been a working version (see footnote 1) of the 1959 tax regulations that was automatically generated by an update

---

1. This timestamp may not make sense since, in the original listing, the last version was for 1979. The 1979 tax regulations, however, have probably not been finalized (at the time of this publication), and surely have not been released to the public. It will be shown that by timestamping versions of objects accordingly, access by persons other than the creator, may be prevented until the appropriate time. Therefore, this last version of the federal-tax-routine object may remain a working copy by maintaining an identifier for which an effective capability, other than the creator's, does not yet exist. When the regulations are completed, the object may be released as a new object (see later sections of this chapter), according to a schedule, some time in 1979.

on an earlier working version. Version 4 might have been explicitly created, according to a fixed schedule, as a new object when the 1959 regulations were finalized, and released at the time[1] indicated by the timestamp on the third object in the sequence. (The above example attempts to model the real world procedure, i.e. tax forms for year -X- are not available until the beginning of the next year, -X+1-.)

In addition, as a new object, version 4 (0053 002 1960) can be accessed via a capability with different privileges than those that were specified in the capabilities used to access all earlier versions. The notion that the update of a version of an object results in an entirely new object, that must be uniquely identified, is critical to the revocation schemes that will be discussed shortly. New capabilities for the new (versions of the) objects must be generated, and these new capabilities need not, and often will not, specify the same privileges for access to the more recent versions.

Finally, one might consider the time of creation of the logical concept, the object history, with some significance. For all practical purposes, it is the time at which the first version of the object was created. All subsequent versions of the object could be represented by time displacements from that creation time, however, this is not carried any further in this thesis. More importantly, the conception of the

---

1. An exact specification of the release time may not be possible due to the performance of operations at a higher priority, by the processor, overriding the object update request.

logical entity must be considered because one might like an object identifier to always reference the most recent version of the logical concept. This requires being able to reference an object history without citing a specific version.

The creator of the primeval object in an object history is special, in the sense that he first conceived of the logical concept that is later manifest in several versions of an object. That is, any one who updates an already existing object may be considered the creator of the new version, however, the creator of the object with version-number = ONE, performed a more powerful operation. The title of creator will only be applied to those entities that perform an explicit CREATE operation which generates an (object, version = ONE), thereby initiating a new object history.

3.1.2 The Naming and Accessing of Objects Via Capabilities

As mentioned, the timestamp[1] alone actually provides a unique object identifier without the logical object reference (ORef). We have assumed that the timestamp value is the time on the node clock at which the version of the object was created. If the precision of the

---

1. For the purposes of this thesis, timestamps will only be precise to the nearest minute. For example, 1630 120 1978 specifies 4:30 p.m. on the 120th day of 1978. This is clearly not sufficient for any type of real operating environment as transactions may update objects at a much more rapid rate, thereby requiring greater precision of timestamps to maintain the unique identification facility.

timestamp is adequate enough, the value for the creation time will uniquely identify all objects generated at a site. The addition of a site ID uniquely identifies all objects in the DCF but, for simplicity, this site ID will be ignored in future discussions.

Now, if we hypothesize a capability-based system, the lowest level names of objects, one level above the object's physical address, are capabilities. It is therefore, the capability that actually contains the timestamp for the creation of the object it names. The full specification of capabilities and their contents appears in the next section.

In concluding this section it is necessary to consider the levels at which the different names and their components will be used. A reasonable scheme is that used in the directory management subsystems of several operating systems. The user is required to handle the equivalent of an (object reference + version number) name. Information equivalent to that represented by a timestamp may appear in the user's local directory listing. The value of the timestamp may be obtained at the time of creation of the object, if the object is private to the user. Alternatively, the value of the timestamp may represent a limit on the user's privileges to access the object, if the object exists outside of his sphere of control; the choice will be more clearly described later in this chapter. This information, however, is

superfluous to the object names that are included in the directory; this information appears for the user's convenience only.

In the DCF system, the user may handle an (object reference + version number) higher level name. The directory, however, will actually contain a capability for each object named, rather than simply extraneous information. It is from the object's ID, contained within the capability, that the physical address of the object is derived. The information about the timeliness of the object and any restrictions on the usage of the capability may be abstracted from the cited capability, if the appropriate privileges for such manipulations were granted in the capability. Such abstraction procedures may exist as system utilities depending upon policy decisions.

This is a most primitive scheme and optimizations can easily be included, for instance, for objects that are repeatedly referenced. An implementation feature might include the use of a '*' version that would result in the most recent version of an object being referenced at each access. The point to be made, however, is that the user should not be allowed to manipulate raw timestamp values; logical names should serve as an interface to the unique identifiers.

3.2 The Mating of Timestamps and Capabilities

Suppose a user has been granted a capability for an object. Despite the identification and authentication of the requesting user that precedes the granting of the capability, the granting user can never fully trust the requesting user. The controller (granting user) may make a mistake or simply decide differently (change of mind or heart) subsequent to the granting of the capability. It is for these two reasons, error and fickleness, that revocation procedures must exist. In the succeeding sections, the usefulness of timestamps will be shown with respect to the granting of capabilities and the revocation of the access rights granted therein.

First, let us agree upon when an object first becomes accessible. In a capability scheme accessibility of an object comes with the possession of a capability that names the object. There is a relationship established between the capability and the object, that of naming. When one cannot name an object one cannot use the object, and if one refers to the object with a dated or invalid name, a similar fault occurs.

Redell, in his dissertation, describes a Typical Capability System (TCS) in which a capability for an object contains,

the unique identifier (UID) of the object,

the type of the object,

a set of privileges to access the object.

A capability in this system will contain additional information. In particular, a capability consists of (Figure 3 and Figure 4):

a) the time at which the capability becomes valid or effective (TCef), (time of creation of the relationship between the capability and the object);

b) the length of time for which the capability is valid, or the time at which the it will expire (TCep);

c) the object type [1] (if objects are typed);

d) a set of privileges that control access to the object; and

e) the timestamp that serves as the unique identifier for the object (OID) named by this capability.

---

1. Luniewski <Luni79> is currently designing the architecture of an object based personal computer in which capabilities name objects but do not contain type information because every object is marked, at the most basic level of the system, with its type.

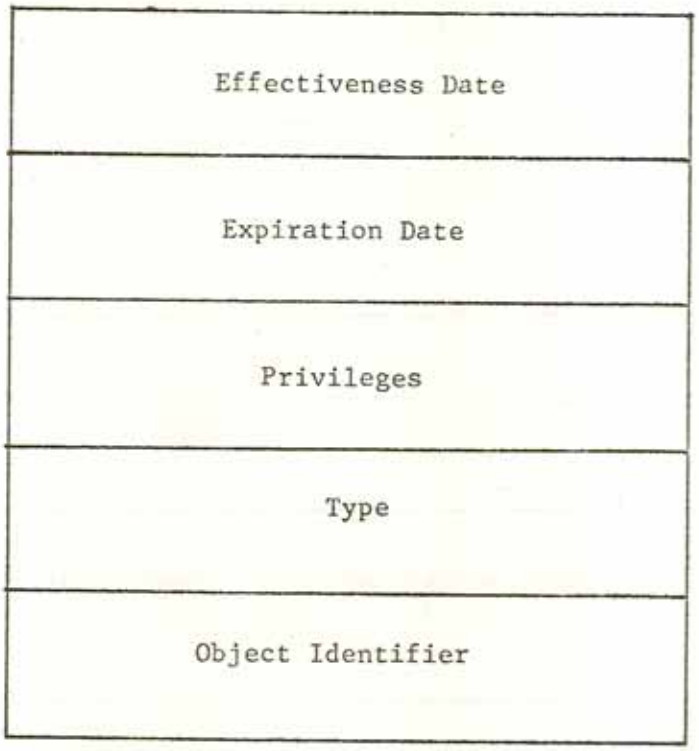


Figure 3: Format of Capabilities

timestamp:	time after which the capability can be used to access the specified object
timestamp:	time at which the capability's usefulness expires
	privileges
type:	object history type and object identifier type
timestamp:	unique identifier consisting of time of creation of object

Figure 4: Contents of the Capability Fields

Further, at the implementation level, part of the capability will consist of a type bit to distinguish a fixed number of machine bits as comprising a capability. The setting of this bit is under special control and thereby, forgery of capabilities (the creation of phony capabilities) is impossible.

The values for the timestamps that are components of a capability will be provided by a hardware register that operates as a clock and is large enough to never overflow in the life of the system. A timestamp value may be the value of the clock (1) at the moment the object was created, or the time at which (2) the capability becomes effective, or (3) the capability expires. This requires that the precision of the clock must be such that no two operations in the machine occur at the same moment in time. If one is dealing with some hypothetical inherently parallel machine, then one must further constrain the timestamp fields in all capabilities to include not only site-IDs for uniqueness, but processor-IDs as well since all processors in such a parallel machine may share the same clock.

3.2.1 The Effectiveness Time (Date) Component

Ordinarily, the generation of a capability would represent a binding of a name to an object. The binding relationship, however, is not something that exists over all time. Names, as basic entities, exist without being tied to existing objects, i.e. they exist in a name

space. It is suggested therefore, that a capability can be created for an object before accesses to the object are permitted.

The DCF capability scheme then, maintains the information (the timestamp) that identifies the moment at which the binding of the capability to the object becomes usable or effective. Any attempt to use the capability for accessing the object before the specified time will result in an exception or error.

3.2.2 The Expiration Time (Date) Component

The motivation for time limits on the usefulness of a capability, that is, the need to specify an expiration date, stems mainly from the desire to provide for an automatic revocation of access privileges. The procedure for revocation of a capability by means of capability expiration is discussed in a later section. The choice, however, between specifying this expiration time as a time value that will eventually be reached on the system clock or as a displacement from the time at which the capability becomes effective, is considered here.

The question is, should a capability that is to be effective, for instance, from 9:00a.m. to 5:00p.m. (09:00 to 17:00) be represented by (A) or (B).

(A)

TCef := 0900 000 0000
TCep := 1700 000 0000

-----

or

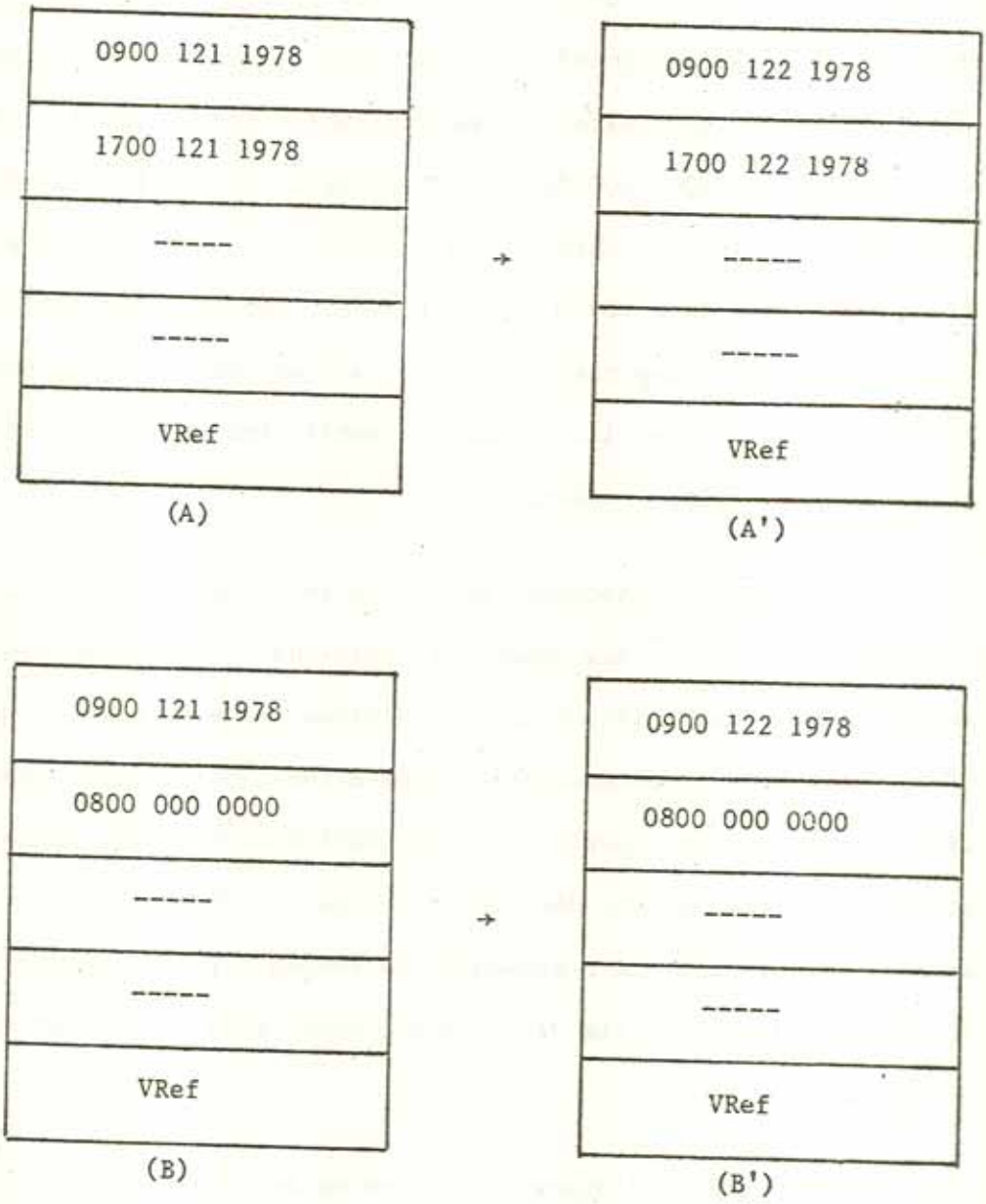
(B)

TCef := 0900 000 0000
TCep := 0800 000 0000

The existence of a timestamp in the TCep field results in a simplification of the authorization check that is made each time a capability is presented for access to an object. The check is made to verify that the capability is still valid (see Chapter 4), i.e., some trusted system component must check that the capability has not yet expired. If a timestamp value actually exists in the appropriate field,

a simple bit comparison, with the current clock time (TE = environment time), is adequate. However, if a displacement exists in that field, the arbiter must first add that displacement to the effectiveness date and then proceed with a comparison. A similar argument for using a time value exists in the process of copying a capability but this will be discussed in the next chapter. As an additional feature, any of the components of the timestamp (minutes, hours, day, year) may be masked out to indicate that they are of no importance in the authorization check, i.e. they may be considered to be "wild".

On the other hand, a determination of the lifetime of a capability by inspection, by someone with the appropriate authority, seems desirable. In a later section, the utility of a procedure of the following form will be shown: when presented with a capability that is to expire, the procedure automatically generates a similar capability, differing from the first only in its effectiveness and expiration dates. Now if the expiration date is implemented by specifying the lifetime of the capability, as in (B) above, then the procedure need only modify the effectiveness date to generate the desired capability. For instance, if the capability is only useful for 8 hours, then the expiration date field will contain (0800 000 0000). This value will be copied into the new capability; if the pertinent fields of the input capability appear as in (B), then the appropriate output should be (B′) in Figure 5.



[VRef ≡ version reference]
where the type and privilege fields are of no concern here.

Figure 5: The Representation of the Expiration Date

Alternatively, if a timestamp value appeared in the expiration date field, as in (A), the output capability would be (A'). In this instance, the generating procedure is required to compute the difference between the effectiveness date and the expiration date (1700 121 1978 - 0900 121 1978 = 0800 000 0000) of the input capability and add this value to the effectiveness date of the capability being generated, to yield its expiration date (0900 122 1978 + 0800 000 00 = 1700 122 1978). This procedure is independent of the policy that is guiding the generation procedure, that is, whether capabilities are automatically generated daily, weekly, or yearly is of no concern.

In the two schemes discussed above, it is necessary to specify the effectiveness date of the new capability when the generating procedure is invoked. This last point deserves additional consideration. We have not discussed specific procedures for generating capabilities under such policies. One could imagine a CAP_IDENTICAL$GEN(TCef, capability) procedure that performed the desired function. The procedure would depend, in part, on the representation chosen for the TCep, but will require the effectiveness time for the new capability regardless of the generating policy.

One could take a different approach by associating an individual generation procedure with any policy that controls regular access of limited duration. The procedures would examine that portion of the timestamp that is critical to the policy and alter the particular bits

of concern. For instance, if a capability is to be altered daily then the procedure simply increments the day bits by one, to represent the next calendar day, without concerning itself with the minutes, year, etc. This requires many generating procedures, (CAP_DAILY$GEN(TCef, capability), CAP_WEEKLY$GEN(TCef, capability), CAP_YEARLY$GEN(TCef, capability), etc.) or one more complex generating procedure that requires the appropriate information upon invocation, to increment the proper time component of the timestamp (minutes, hours, days, year). Nevertheless, it is clear that a timestamp value complicates the automatic generation procedures. But, since there are no strong positive arguments for a displacement value to be used in the capability, explicit timestamp values for the effectiveness date and the expiration date are used, for reasons that will become evident.

3.2.3 The Object Identifier Component

The OID is the timestamp that uniquely identifies the object by the time and place of its birth. The OID value can either be a UID of a specific version, that is, a version reference, or it can be an object reference (that is, a reference to the object as a logical concept). A bit in the type field, the reference-type bit, specifies whether the OID field contains an object reference or a version reference.

The OID value could also provide the user with information as to the currency of the contents (value) of the object he has access to.

Such an abstraction manipulation is privileged, and it must be granted in the capability.

3.2.4 The Object Type Information

This thesis has avoided any discussion of extended objects and abstract data types. If such is the nature of the world, the object type information is needed by the entity performing the authorization check to verify the appropriateness of the action requested upon presentation of the capability. Whether mechanisms such as representation capabilities, domain capabilities, or sealed capabilities <Rede74> provide for the extended object nature of the system is of no concern to the main topic of this thesis.

This system presents a new problem, however, that can be resolved, in some form, by use of this object type field. Earlier we hypothesized a desire to always provide access to the latest version of the object history; now we need to provide the mechanism for this. The problem is that at any point in time one does not know the precise identifier for the most current version, i.e. what is the value of the OID field. The most useful information that could appear in the capability would be identification of the object history with which the desired latest version is associated.

The timestamp specified as the OID, in such a capability, will be recognized as referring to the object history (or logical concept) by

means of the reference-type bit in the type field. The reference-type bit indicates whether the OID field contains an object reference or a version reference. When this bit is set, indicating the presence of an object reference, the timestamp may be used to identify the latest version, by whatever means exists for logically connecting the objects in one object history.[1] That is, the value of the timestamp that appears in the OID field represents the time at which the logical concept was conceived. For all practical purposes, the conception of the logical concept may be considered to be simultaneous with the creation of the first version of the history; yet, the object header which represents the object history is a distinct object from the first version of the object history.

The characterization of "latest version" may refer to any number of objects through the course of the existence of the logical concept; the object named is dynamically changing. To facilitate further accesses of the object, a capability for the most current version may be created and temporarily bound to the user's directory for the remainder of his work session. Any further accessing of the object would bypass the original

---

1. We will assume a mechanism similar to that described in <Reed78> by which the relationships between the versions of one logical concept are maintained. A data structure is maintained at the home of the object consisting of several entries one of which is a pointer to the representation of the value of a particular version. These structures are then threaded together. The entire object is represented by an "object header" that contains a pointer to the list formed as above.

capability. However, the original capability must be invoked at subsequent sessions as the "latest" version may no longer be the same.[1]

It should be realized, however, that the accessing of the most current version of an object history is only one of a number of manipulations that can be performed given a capability that contains an object reference. Therefore, for now we will simply state that one of the privileges that can be granted in a capability is that which results in the eventual access of the most recent version of a logical concept. The ways in which the object reference can be used in any particular capability are indicated in the privileges field.

3.2.5 The Privileges

A discussion of access privileges, given our view of objects, takes on new meaning. Since each update of an object results in the creation of a new version of the object, i.e. a new object itself, certain operations on objects no longer make any sense. In effect, the WRITE privilege, in the traditional sense: to be able to read an object's value and then write a different value in its place, no longer exists. One cannot alter the value of an object and refer to it with the same name, where (name = object reference + version information). The state

---

1. The "latest" version may, of course, change during a session, but it is impractical, and perhaps undesirable from the user's point of view too, to keep track of these changes.

of the object is different and in the world that has been defined in this thesis, that means a new object has come into existence; versions are immutable.

Instead of the conventional "write", an operation such as the "VERSION_REF$DEFINE(vr, value)" operation described by Reed <Reed78> has to be used. This operation takes two arguments: the version reference to be defined and the value to be associated with that reference. This operation can be applied once to any version. An error may be signalled if a valid version already exists that is specified by a version reference identical to the parameter supplied.

However, there are now a number of new privileges that may be considered. One might ask for a summary of the object's state, for the results of a statistical operation applied to the object, for the number of versions that exist after the version specified in the capability, for the amount of time lapsed between the version specified in the capability and the most recent version (currency data), for the value of new state variables such as the Severity-of-Change (SOC) indicator <DeNo79>, etc.

The SOC specifies the degree to which the value of the object has changed from the version specified in the capability to that of the latest version. Such information could be useful in determining whether the latest version is really necessary for user operations as well as for other optimizations. The node at which the most current version

resides may not be operating as part of the network at the desired time of access; the time delay suffered upon retrieval of that version may be unreasonable in the current transaction while a less recent version may be usable, as long as it is readily obtainable, etc.

It was also mentioned, in the last section, that the time of creation of the object need no longer explicitly appear in the directory because that information is inherent in the capability. This information may be obtained by abstracting the TCef and TCep fields of the capability and translating the information into a more usable form. Every user need not be permitted the information on the period of validity, effectiveness and expiration dates, of a particular capability.

If the facilities for abstracting information from capabilities are deemed privileged, permission for their use will have to be granted in the capability. If such information can be abstracted by the particular user, it may then be stored in the directory for future reference. In addition, the fact that this information can be extracted from a capability avoids the inefficiencies that would arise if, at the granting or copying of a capability, an additional piece of information, the user interpretable date, were also generated and passed along.

Also, as described in the previous section, the privileges field will indicate the permissible manipulations given the nature of the reference appearing in the OID field. Given an object reference, we

have discussed the possibility of accessing the most recent versions of object history's; other manipulations necessitating the object reference value will continue to appear. In addition, the creator of the logical concept must have a fully privileged capability to the object header (a capability specifying an object reference in the OID field) so that he can perform any of the manipulations on the object header that may have been implemented as part of the basic mechanism, e.g. the creator may desire a complete list of all currently existing versions in the object history, he may destroy the entire history, etc.

3.3 The Granting of Access Privileges

Once a request for access to an object has been favorably decided upon there is yet another important decision to be made. One has the opportunity, when specifying the capability to be passed to the requesting user, to delay the effectiveness of the privileges being granted. Also, one must specify a particular version to which these privileges may be applied; one could specify the version as the most recent.

3.3.1 Granting Capabilities That Are Not Yet Effective

The facilities for the implementation of the concepts of reservations or appointments for access to objects can be provided by the DCF protection mechanism. A capability can act as a ticket for

access to an object at a future date. This moment in the future, at which time the effectiveness of the capability may be realized, is specified in the TCef field of the capability by a timestamp whose value is greater then the current value of the system clock, TE = environment time. The TCef holds the time at which the binding between the capability and the object is valid, as described earlier. This scheme relies upon the understanding that the capability is defined by a finite period throughout which it may be used for valid access to the named object. The capability owned by the creator (of the logical concept) of an object is an exception to this rule; the creator's capability has an infinite lifetime. It is important to note that the value of the OID field is a version reference; the precise object to which access is authorized exists at the time the capability is generated.

This scheme is, in some sense, a scheduling aid as well as a feasible security mechanism. By scheduling access to objects one indirectly facilitates accountability for actions, especially if one provides for mutually exclusive access periods. That is, by granting capabilities that become effective at different times, one has made appointments for the recipients to access the desired object at the given time. Of course, there are no guarantees made by this mechanism, at that same time, on the object's integrity and/or availability.

More importantly, however, by specifying a future time at which the capability becomes effective, one can make immediate decisions on the

requests of users to access objects, and grant those users privileges that may be used for accessing the particular objects after the specified future time. The future access is dependent on the TCef of each individual capability. When the controlling entity decides on the timestamp value for the effectiveness date it is making a decision that affects only one user, ignoring, for now, those to whom this user may pass the capability.

Underlying these applications is the dynamically changing importance of information; dynamically changing with the flow of time. Information can be valued by the time of its release, i.e. exam answers are more highly valued by a stuient if discovered before an exam rather than after; knowledge of profits ten years ago may be more valuable, for a given application, than knowledge of last years profits; news of a merger is more valuable, even in real dollars, before a public announcement is made (because stock can be bought at the lower, pre-merger, price), etc.

It is this approach that provides one with the facility for automatic granting procedures for (automatic generation of) some class of capabilities. For instance, a set of capabilities that becomes invalid at the end of each working day may be invoked by users who are authorized for use of the named resources on a daily basis. Ideally, one would then like a new set of capabilities to be generated for the next working day. The expiration date of these capabilities need not be

as limiting as a day: bank safes have controlled hourly access while payroll processing is only authorized weekly, but there is a regular usage pattern.

The generation of a new capability depends on the user having already possessed a capability that was identical in every way except for its effectiveness date. Bank tellers should be allowed access to account debiting and crediting procedures on a daily basis and with EXECUTE privileges. Yet, the bank requires that each teller be prohibited from access to these procedures at the end of each working day and on weekends and holidays. (Presumably, a teller will be prevented from access to the entire system, on such days, by physical controls on the bank itself.) From day to day, these procedures will not vary, and so the notion that they be destroyed daily with new versions subsequently created and new capabilities granted to those versions, is not pleasing.

One would, therefore, like a teller's capability for accounting procedures to expire at the end of the day with a new capability generated that will not be effective until the next working day. The correct and desired version of the object to be accessed exists at time TE, while the capability cannot be used until some future time, TCef. This scheme is most appropriate when there is no meaningful way to justify the creation of a new version of the object, i.e. there is no state change from one version to the next, nor is there a change in the

state of some system variables such that the older version now has a different meaning or role.

3.3.2 Granting Capabilities for Future Objects

Alternatively, access for a time in the future can be granted by creating a capability that identifies a future entity or event (a version of an object, e.g. the final report to be released on a certain date, or the occurrence of a special event, or the automatic declassification of government documents which, by law, are maintained secure for forty years after their classification).[1] Access to the object depends on events that are in no way related to the requestor; the existence or the elimination of the object may affect a larger group of users. If this approach was applied in the example cited above, the versions of the accounting procedures used on one day would be different, in name, from those used on the next day. The procedures used on a particular day would be destroyed at the end of that day, with new versions of the procedures (and new capabilities) generated for the next day at that time.

---

1. The declassification of an object requires that certain changes be made to the physical object resulting in an entirely different physical representation of the same object. This new object possesses the same value as the original but the "packaging" is different, e.g. a document no longer bears the words "TOP SECRET". The classification information is an integral part of the classified document and not of the document that is finally made available to the public.

This scheme and the one described above are similar yet quite different. In both, at some point in time -T-, access to object -O- is prohibited but, access at time -T+t- is known to be possible. In the former case, this is possible because authorization until time -T+t- is invalid. The field of the capability that is to contain the capability's effectiveness date, contains a future date, i.e. a time value greater then that of the current system clock value. In the scheme being introduced here, access is prohibited because at time -T- the object -O- does not yet exist, i.e. although it can be named it does not exist in a physical or logical sense.[1] Granting a capability for a future version is similar to granting a capability for the most current version. In both cases, the OID of that version is not known at the time its capability is generated.

All capabilities granted to a future version would take effect at the same time and could, but need not, expire at the same time. Of course, if the version was explicitly destroyed, all capabilities to it would become ineffective simultaneously. For instance, one might consider a season ticket to the symphony as being composed of a set of capabilities for future objects. Once the symphony has given a particular performance, the object (the performance) accessible with one of the capabilities from the set no longer exists and therefore, the capability is worthless. If an attempt were made to get into the

---

1. The future existence of that version of the object is not in question.

concert hall on the following evening it would fail. The mechanism for detecting when a future object is being referenced is exhibited in the next chapter.

Another motivation for granting access to objects that do not yet exist is an accounting one as well as being a better model of some real world authorization procedures. If it is expected that many users will make use of an object then it may be more efficient if access authorization is decided as the requests are made rather than at the release date. The efficiency is realizable at both the controlling user and the requesting user ends. The controller will not be consumed with processing requests for capabilities that queue up subsequent to the creation of the object. And, the controller may perform more extensive checking procedures on the requesting entity. From the requesting subject's perspective, he does not have to delay his processing by queueing and waiting for the arrival of a capability to a newly released object.

3.4 The Revocation of Access Privileges

After an entity has granted a capability to a subject for access to objects in its own domain, that entity is still faced with the possibility that a mistake was made or that it might change its "mind". Revocation of access privileges can be a problem. If capabilities can be freely copied in the DCF the problem is further complicated. Some

schemes have constrained capabilities to include a "copy" bit to indicate whether or not they can be freely copied or have forced them to reside in designated capability-holding segments. Redell <Rede74> extended the capability scheme to provide for the accessing of target objects via an indirect object that could be independently destroyed, thereby revoking access to the target object. Below, the DCF capability scheme is shown to be versatile at revocation of access privileges; granted privileges can be revoked without knowledge of where in the system the capability exists and to whom it was originally granted.

3.4.1 Revocation by Expiration

We have hypothesized the existence of objects that are repeatedly and regularly used in certain environments. This may be dictated by the operating hours of the system, e.g. 9:00 a.m. to 5:00 p.m. for business organizations. Similarly, library books may be borrowed for two weeks, apartments may be rented on a yearly basis, etc. One might like to model this control by an automatic and automated revocation procedure that is referred to as revocation by expiration. A capability that is granted to name an object under the auspices of such a policy would require the facilities to specify that the relationship between the name and the object have a limited lifetime. The relationship between the name and the object is that which may expire, not the capability nor the object itself.

This is the motivation for the existence of the expiration date in the capability. By specifying this date at the time the original capability is generated, one need not worry about the propagation of that capability nor the accumulation of a multitude of worthless capabilities across the DCF. If a capability is copied, the expiration date will remain the same; any limits on the use of the object by the original recipient of the capability will be carried over to those who possess copies of the capability. Of course, we are assuming the maintenance of the integrity of the capability by means of the tag associated with the machine entity that represents the capability and that is not to be tampered with.

3.4.2 Revocation by Elimination

In the first section it was suggested that it be possible to grant access to an object specifying the version as the most recent version of the object. Such a capability would be granted to the most trusted users. However, it seems more desirable to only grant access to particular versions of an object. This would allow for selective revocation of the access rights specified in the capability by elimination of that version of the object from the sequence of objects that is the object history.

Clearly, the creator of an object will not need, for all time, a record of all the states his object has assumed throughout its lifetime.

Selected versions will be eliminated as soon as they are useless and/or obsolete. This will result, in effect, in a selective revocation of the effectiveness of those capabilities that name the eliminated version as the object for which access privileges are contained.

Revocation by elimination also provides information to a user concerning the state of the object he wished to access. In the Typical Capability System <Rede74>, if a capability is invoked by a user and results in an exception then the user is aware that his rights, and/or the resources he may manipulate, have been reduced. This is true of the current scheme, however, now if a user invokes a capability for a particular version and receives an exception he is not at a total loss. He could initiate a new request for the same object, and thereby receive a new capability that names a different, and perhaps more recent, version of the object. The user could be provided with some information about the resource he was using; the object's value has been updated and he now possesses access to a more current version of the object.

Alternatively, the grantor of the capability may have discovered an error on his part, in granting a capability to a particular version. He may create a new object that will have the same value and then eliminate the version. By eliminating the version he has made all outstanding capabilities to that version ineffective; the object they name no longer exists. Subsequent requests for access to the particular object may be granted to older versions or more recent versions, or none.

3.5 Summary

In this chapter we have defined a world in which timestamps are naturally associated with every object. Using newly defined capabilities as the basis for a protection mechanism we have shown how access decisions may be event-sensitive or value-sensitive. Event-sensitive criteria include those decisions that are made based upon the appropriate values of the system clock, the effectiveness date and the expiration date. Value-sensitive criteria include those decisions that are made based upon the version of the object that is being accessed, i.e. decisions based upon the current value of the data being accessed, as described in the preceding chapter.

Chapter Four

THE MECHANISMS AT WORK

The protection mechanism employed in a prototype node will be described in order to assure that the intra-node protection mechanisms have been adequately considered. One cannot guarantee that any distributed facility will provide protection among its members unless each component of the facility can provide its own internal protection. Of course, it could be decreed that those nodes that do not meet a minimum specification for internal controls remain isolated from some class of users. For the purposes of this thesis, however, we will only consider the general case, i.e. whether or not there exists a transparent classification of users is irrelevant.

This chapter contains a discussion of the arbiter entity that is required to assure the integrity of capabilities and that performs the appropriate authorization check when a capability is invoked. The capability generation procedures are defined as well as a distinction between the creator of the logical concept or object header and the creators of object versions. Also, the generation of capabilities for access to future objects is considered. This chapter proceeds with

a close look at the mechanics of the authorization check procedures and then concludes with a discussion of the copying of capabilities and the passing of capabilities among the nodes in the DCF.

It should be noted again that the discretionary access controls may operate under the constraints of a transparent non-discretionary control mechanism. That non-discretionary scheme may be implemented by the classification of nodes (users) at the time of their admittance to the network community. The entire DCF will have an administration node combining characteristics of the Network Information Center that supports the ARPANET, and the Network Security Center. This administration will control and screen network membership. New nodes will have to be included in the network directory and valid site-IDs will have to be established for them. Further, it will be the administration's job to provide special classification for each node that requires something other than public membership in the DCF.

The implementation of the DCFA is of no concern here. I simply wish to assume that there is at least one source of the most current and accurate information on the nodes in the network, their access capabilities (to other nodes), and their sensitivity level, if any.

The operating system at each site will provide for the identification and authentication of users, allowing only the data base management (sub)system access to the data base, and prohibiting any

alteration of DBMS or operating system code. The attempt made here is not to modify existing capability systems, but rather to extend the designs, in conjunction with the use of timestamps, to provide new and enhanced functionality for the usage of capabilities in a distributed environment. The changes required of capability generation schemes and of procedures used for the revocation of access privileges, will be discussed.

4.1 The Arbiter

It is assumed that the users at each node trust, at the very least, one process resident at that node. Mutually trusting users may exist at any particular node but when one expands the system boundaries to include the communications medium one cannot assume mutual trust overall. The trusted process in each node, henceforth referred to as the arbiter, is relied upon to handle the generation of capabilities, the copying of capabilities, and the authorization checks at each invocation of a capability.

The arbiter is a highly privileged and protected locus of control whose function is to check each user's request for access and to grant or deny access, as appropriate. This trusted process will be the only entity capable of setting the type bit of a capability to distinguish it from data objects. This means that it is this trusted component of the node's operating system that will generate capabilities, that is, it is

the only process that can use the clock value (time), read from the local node clock, for the OID field of the capability.

The use of the expression "locus of control" is very important. It should be realized that as I have added (and will add) functionality to the overall protection mechanism I have paid no attention to where exactly the implementation of that functionality will be placed. The arbiter manifests a trusted process that may actually be implemented as several separate procedures for the generation, the copying, and the verification of validity of a capability.

4.2 Capability Generation

If one has a need for an object under the control of another user, an explicit request for a capability to the desired object must be directed to the controlling user. The controlling user may be the creator of the object or he may be the owner of a capability previously granted by the creator of the object. In any case, the controlling user makes whatever checks he deems appropriate and then decides whether or not to authorize access, by means of granting a capability, to the requesting user.

The examination of the requesting user will vary in thoroughness depending upon the value of the object desired and depending upon the identity of the controlling user. The creator of the object is more

likely to perform an extensive investigation of the requesting user than is any other possessor of a capability to the object. On the other hand, if the object's security is of little importance, even to the creator, he may perform a cursory check of the user and then grant him a capability for the desired object.

Suppose a user has been authorized for access to a particular object. As was noted, the user who has made that decision may, or may not be, the creator, in the traditional sense, of the object being accessed. Regardless, he must make several decisions affecting the generation of the capability that will actually be given to the requesting user: can the user access the object now or sometime in the future?, in what particular ways may the user access (manipulate) the object?,[1] for how long should he be able to take advantage of these rights?, etc. The answers to all of these questions, in effect, determine the degree of trust that the granting user has for the requesting user.

---

1. The creator of the object may grant any subset of all possible privileges whereas an ordinary user can only grant the requesting user some subset of the privileges he was previously granted in his capability.

4.2.1 A Capability for the Creator of an Object

It should be noted that the capability of the creator is special in that it has no expiration date (infinity),[1] and its effectiveness date is identical to the object identifier (OID), as depicted in Figure 6. Further, the OID does not consist of a timestamp serving as a version reference, but rather, it is a value that serves to identify an object reference. As was discussed earlier, the object reference serves to identify an entire object history; its value is the timestamp that represents the moment at which the logical concept was conceived.

The creator of the logical concept should be allowed access to all existing and future versions of the object. Therefore, by possessing a capability that names the logical concept, and with knowledge of the appropriate procedures for manipulating the information about the object history (derivable from the object header as suggested in the previous chapter), the creator has access to the entire object history (all versions, current and future, of the logical concept). The fact that the creator's capability has an OID value that represents an object reference indicates that the capability's privileges field represents the permissible manipulations on the object header and not on a particular version of the object itself. One such manipulation that has been discussed and which may be authorized for ordinary subjects also,

---

1. The time of expiration is represented as infinity (oo) for the reader but, in effect, it will be implemented as the largest possible time value representable in the TCep field of a capability.

TCef	:=	1003 005 1979
TCep	:=	∞
type	:=	reference-type: 1
privileges	:=	-------
OID	:=	1003 005 1979

where,

object reference := "D" and timestamp of creation := 1003 005 1979
and the privileges field is of no concern here.

Figure 6: A Creator's Capability for An Object

is that of obtaining the most current version information from the object header information.

Now that we have drawn a distinction between the creator of the logical concept and the creator of subsequent versions of the object that the logical concept represents, we must consider the capability of this other (version) "creator". Most obvious is the fact that the creator of a new version of an object must possess a capability specifying some sort of update privilege on an already existing version of an object. Such a privilege might provide for access to the VERSION_REF$DEFINE(vr, value) operation described in the last chapter. We may therefore consider the creator of a version to actually be the "definer" of the version.

The execution of the VERSION_REF$DEFINE(vr,value) operation will need to involve the generation of another capability for the executing entity. This new capability will have to specify the newly created version in the OID field and the time at which the "define" operation is applied in the TCef (effectiveness time) field.[1] The entity executing such an operation should be allowed to apply the operation to each of the resulting versions, unless the capability is revoked. It must be

---

1. The semantics of the mechanism warrant the updating of the TCef value, however, the information retained in the original TCef value may be useful. That is, the original TCef value indicates when the subject began to manipulate the objects in the relevant object history. This may serve, in some sense, as an official alibi mechanism, i.e. if a malicious act was committed certain subjects can prove themselves innocent. Regardless, maintaining the original TCef does no harm.

remembered that, by definition, the VERSION_REF$DEFINE(vr, value) can only be applied once to any particular version. In essence, we must provide the same functionality as an ordinary WRITE operation.

4.2.2 A Capability for a Future Object

Recall also that we may need to generate capabilities that refer to future objects as suggested in the previous chapter. In such future capabilities, the OID cannot specify precisely the future version, since the UID of a version is the time of the version's creation, and the exact moment at which creation will take place cannot be determined a priori. Thus in a future capability, the OID field will contain an object reference with the reference-type bit set. However, in this case, as opposed to the capability held by the creator, or a capability for the most current version, the TCef is some time in the future. At authorization check time, if the TCef has come to pass, and the OID is an object reference, the future capability is converted into a capability for the last version created before the time value in the TCef field.

We have now made the authorization check procedure more difficult; when it is discovered that the OID field contains an object reference and the TCef field contains a valid effectiveness date (a value smaller than the current system clock value), the nature of the capability is still not fully defined; 1) it may be a capability that was granted to

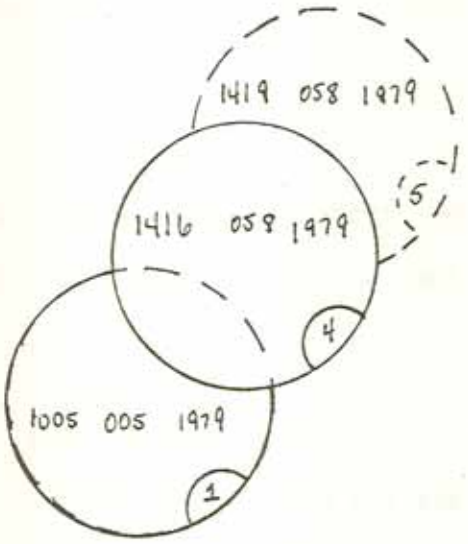
an object that was non-existent at the time the capability was generated, e.g. it is a future capability, or 2) it may be a capability that provides access to the most current version of the logical concept.

Since the distinction affects the manner in which the two capabilities may be manipulated by the arbiter to arrive at a capability that contains a precise version reference, we will assume that the pertinent information can be found in the privileges field. The information as to the character of the capability (whether the target object is the most current version or was a future version) is represented by privileges for manipulation of the invoked capability. Once the capability has been determined to contain an object reference, we know that the semantics of the privileges granted in the capability are different from those associated with version references.

The version reference that results from the authorization check rests on a subtle distinction between the two capabilities: The most current version capability is one that provides the possessor of the capability with access to the very latest version <u>after</u> the time value in the TCef field has passed. When a capability is invoked there is only one "latest" version, however, the physical version that possesses that attribute changes depending upon when the capability is invoked. The future capability is granted to authorize access to a non-existent version; the version to be accessed is that which was created immediately <u>before</u> the TCef value. At any point in time there will be

only one version that can satisfy the criterion for access; the satisfaction of the criterion depends on the particular value of the TCef field.

To summarize, Figure 7 depicts a number of capabilities to illustrate the use of the reference-type bit and the privileges field. Capability (1) permits ordinary access of the target object, capability (2) is one that might be in the possession of the creator of the object history, capability (3) authorizes delayed access of the target object, capability (4) grants access to a future version, and capability (5) enables access of the most current version.



Assume TE = 1113 073 1979
and a target object from Figure 2.
(from Figure 2 we know the logical
concept (object header) was created
at 1003 005 1979))

ORef := "D"

VRef := 1416 058 1979

TCef := 1113 073 1979
TCep := 2400 160 1979
type := -----
priv := -----
OID := 1416 058 1979

(capability 1)

A capability for a
particular version.
The capability was generated
at the current time.

Figure 7: Variations on a Capability

TCef := 1003 005 1979
TCep := ∞
type := reference-type: 1
priv := -----
OID := 1003 005 1979

(capability 2)

The creator of the object history "D" might possess, at the very least, a capability similar to (2) for the object header.

TCef := 2400 082 1979
TCep := 2400 160 1979
type := -----
priv := -----
OID := 1416 058 1979

(capability 3)

Capability (3) is one which delays access to the target object until some future time TCef.

Figure 7: Variations on a Capability (continued)

TCef := 0000 150 1979
TCep := 2400 160 1979
type := reference-type: 1
priv := future/latest: 0
OID := 1003 005 1979

(capability 4)

Capability (4) provides access to a future version of the logical concept "D". This is indicated by the combination reference-type = 1 and future/latest privilege = 0.

TCef := 0815 035 1979
TCep := 2400 160 1979
type := reference-type: 1
priv := future/latest: 1
OID := 1003 005 1979

(capability 5)

Capability (5) provides access to the most current version in the object history. This is indicated by the combination reference-type = 1 and future/latest privilege = 1.

Figure 7: Variations on a Capability (concluded)

4.3 Using Capabilities for Protection

Finally, the authorization check procedure, that is, the procedure that is followed to verify that the capability presented to the arbiter is valid for access to the object, at the current time, TE = environment time, can be described. It is initiated by a subject presenting a capability for an object to the arbiter in anticipation of being granted access to the object. When the capability is exercised, the following functions are performed:

0) the type field is checked to determine whether the OID field contains an object reference or a version reference. If the reference-type bit is set the OID field contains an object reference. Now the nature of the capability must be determined: is it a capability for the latest version of the object referenced?, is it for a future version of the object?, or is it the capability of the creator of the object concept and therefore, just a pointer to the object header? This determination can be made by examining the bit set aside in the privileges field for the possible manipulations of an object reference. Once the object to which the capability refers is determined (a precise version reference is arrived at) the authorization check may proceed through the following steps. (At this time, a capability specifying the version reference may be

substituted for the original capability for increased efficiency, if desired). If the reference-type bit was not set then the capability contains the exact version reference for the object desired.

1) the timestamp representing the OID is checked to assure that the object exists and then a verification of the map entry[1] is made,

2) the timestamp representing the effectiveness date is checked to assure that the capability is effective

3) the timestamp representing the expiration date is checked to assure that the capability has not become obsolete,

4) the appropriateness of the requested action is checked against the privileges and type information specified in the capability,

5) the map entry is checked for the presence of the object in primary memory, and finally

6) the address appearing in the map is used to perform the access to the object.

---

1. The map refers to whatever implementation scheme is chosen to arrive at the physical address of an object from the object identifier that appears in the capability presented to the arbiter.

Any of the above steps, except step 5, may lead to an exception whereupon the authorization check will be considered to have resulted in a negative response to the request for access. If the object is not found to exist in primary memory, it must be retrieved from the supporting memory levels.

4.4 Using Timestamps for Protection

The granting or denial of access permission would be decided by the arbiter after a 3-way comparison is made. There are four clock times involved:

1) the current value of the node clock, or environment time = TE, at the instant an authorization check is made,

2) the timestamp (time at which the capability becomes effective) of the capability, TCef,

3) the timestamp specifying the time at which the capability expires, TCep, and

4) the timestamp (time of creation = unique identifier) of the data object named by the capability, TO.

The time values representing (2), (3), and (4) are actually contained in the capability held by the person requesting access.

The possibilities for the ordering of TCef, TE, and TO at the time (TE) a capability is invoked follow:

(1a)	TE	<	TCef	<	TO
(1b)	TE	<	TO	<	TCef
(2)	TCef	<	TE	<	TO
(3)	TO	<	TE	<	TCef
(4a)	TCef	<	TO	$\leq$	TE
(4b)	TO	$\leq$	TCef	$\leq$	TE

< := comes before relationship

$\leq$:= comes before or simultaneous with

> := comes after relationship

$\geq$:= comes after or simultaneous with

(1a), (1b), (2), and (3) all yield negative responses to the request for access: (1a), (1b), and (2) are identical, for all intents and purposes, because the object that is requested does not yet exist. (3) yields a negative result because the capability has not yet become effective (this is true of the situations depicted in (1a) and (1b) as well). Access to the object is permitted in (4a) and (4b).

The above is sited from the point of view of a user other than the creator of the object TO. A creator of the object will always have TCef = TO with TCep = ∞.

After we take a closer look at the above relationships we notice that there are only two possibilities for the relationship between the effectiveness date of the capability (TCef) possessed by the user requesting access to the object and the timestamp of the object (TO). Either, (a) TO < TCef, or (b) TCef < TO. Each of the situations listed represents a possible outcome (result) determined by the arbiter, at the time an authorization check is made.

Cases (1b) and (4b), TCef < TO, exhibit a certain determinacy in the system that can be perceived by the fact that a capability was granted to access a particular object before the object had even been created (scheduled access). Cases (1a) and (4a), TO < TCef, provide us with the 'usual' sequence of events.

We observe that some of the above are not of interest. For instance, at authorization decision time, a capability must be effective when presented to the arbiter to qualify for access to the object. If the capability is not effective as of that time, TE, the decision is automatically made. Relationships (1a), (1b), and (3) fall into the above category; if a user does not have an effective capability to access an object, the authorization check need not proceed any further, the result is negative.

4.5 Copying Capabilities

An entity that possesses a capability is allowed to manipulate it in constrained ways as well as being able to access the object named within. One of the most important manipulations on a capability is the copying of the capability to yield two capabilities for access to the object cited in the original. In conjunction with copying one can also reduce the privileges in a capability by performing the equivalent of CAP$REDUCE(capability, privileges) where "privileges" represents a subset of the privileges that are to be retained from those that appeared in the original capability, i.e. one can reduce the privileges that appear in the copied capability from those in the original.

The possessor of a capability may now not only copy the capability elsewhere in his domain but he can also pass it on to other subjects without communicating with the entity that granted the original capability. In the typical capability scheme, the copying of capabilities and their propagation throughout the system without control greatly complicated the revocation problem. In the DCF capability scheme the solution is quite straightforward; when a capability is copied the expiration time is limited by the TCep of the original capability.

Essentially, the possessor of a capability can produce another capability, by copying the first, that is, at most, as powerful as the one in its possession. The privileges of the copied capability can be

restricted but they can not be expanded; the possessor of a capability does not also possess all the powers of the creator of the object from whom the original capability was granted. Therefore, by applying the same rule to the expiration time, we are saying that it too, in some sense, represents a privileged length of time throughout which the capability is useful.

This approach, as well as assuring a revocation mechanism, prevents the propagation problem from becoming unmanageable and unreasonable. A copied capability will only be effective for, at most, the same amount of time as the original. Regardless of where in the DCF it may finally reside, it is limited by the TCep. Also, the possessor of the capability from which the copy was generated can use its discretion to further limit the use of the target object and better control its resources by further restricting the lifetime of the capability.

4.6 Capability Passing Among the Nodes in the DCF

In the DCF, capabilities can be propagated to unknown subjects, making the knowledge of who the potential users of an object are very difficult to obtain. The question of who can get to a particular object is easily answered in an ACL system. The technique that will be used in the DCF is the following: Capabilities that have been granted for access to non-local objects will be stored in a specific area (segment) of the storage system. Each user will have to be granted a capability into

this segment as soon as they are granted a capability to a non-local object. More specifically, the user will be given a capability for the particular entry in this segment that corresponds to the appropriate non-local object. The privileges in this capability will only provide for access of the non-local object named in the capability placed in the storage segment.[1] The manipulations on the object are restricted according to the privileges specified in the capability appearing in the table. The entire procedure will maintain transparency as far as an end-user is concerned.

Since the user is required to access the non-local object via a local capability one can synthesize other uses for this non-local capability segment. For instance, the node itself may have been granted a "long-term" capability for a non-local object; the use of this capability by the local node users may be controlled via these local capabilities. Thereby users may be prevented from using remote resources during those hours when the rates are high. An even more subtle facility is achieved if the node was granted a capability that always accessed the latest version of an object. The node may now grant capabilities that have a limited lifetime while maintaining its own capability for access to the most recent version.

---

1. If we had not made the assumption that the arbiter at each node was trusted, this segment could contain all non-local capabilities in an encrypted form.

In summary, the scheme described above achieves two things: 1) One need only search these particular areas at each node, rather than all stored information in the system, to determine exhaustively all potential users of a given object. It is true that the local capability may propagate amongst other users at the node, but one has, at least, isolated a group of potential users of an object. Almost as important may be the knowledge that the users at a particular node cannot access the object at all, simply because the capability for the object does not appear in the non-local capability table. And, 2) it allows the system to keep tabs on the use of non-local objects and dynamically (if desired) maintain multiple copies of objects if it appears necessary, to maintain the efficiency and performance of the system. This possibility of dynamic reorganization is very interesting but it also requires a proficient scheme of update synchronization.

A capability can be passed to another node only through the trusted arbiter. Further, it will be assumed that data items and capability items will not be passed in the same message. That is, a capability must be passed in a special message; the type field of the message will indicate that it is a capability message. This message is created in the trusted process and the receiving side will thus believe that it is truly a capability message. This, of course, necessitates some mechanism that prevents the message from being modified while it is being transferred between the two physical nodes. Specifically, it is assumed that these messages are encrypted.

End-to-end encryption can provide adequate protection of information transmitted over computer networks <Kent76>. The encryption process consists of transforming the sensitive information into another form and using that for transmission purposes. An intruder would be unable to determine the sensitive information unless he knows or can find the transformation <Sha77a>. Essentially, the integrity of the capability must be maintained regardless of where in the distributed environment it comes to reside.

4.7 Summary

In this chapter we have more closely examined the semantics of the protection mechanisms defined in Chapter 3. The critical distinction, resulting from the new approach to object management, between the creators of object histories and the creators of object versions was discussed. A discussion of the generation of capabilities for the creators of the object history was included as well as a discussion of the generation of capabilities for future objects.

Subsequently, a summary of the capabilities that may exist for access to objects in an object history was presented followed by an analysis of the authorization check procedures and the criteria used in making the accompanying decisions. Finally, the procedures for the copying of capabilities and the passing of capabilities in the distributed system were briefly examined.

Chapter Five

Summary

We have considered a small portion of the myriad of issues that arise in investigating the problems of providing protection in computer systems. In this thesis, protection mechanisms were designed for a distributed computing facility. This facility ideally would exist as a system that is distributed both physically and logically but, most importantly, a system that involves many different kinds of users and applications that rely upon very large and diverse data bases.

One cannot make any assumptions about the applications for which the system may be used nor about those who will make use of the systems facilities. The users cannot be expected to cooperate nor will a simple classification scheme for making authorization decisions be applicable to a large body of users. And, just as users cannot be expected to cooperate the nodes of the DCF cannot be expected to cooperate; autonomy of all nodes must be maintained. The requirements of such a distributed system make the direct application of known techniques impossible; techniques that successfully provide protection in centralized systems have to be modified to operate in a distributed environment.

An examination at the nature of the protection problem in centralized computer systems was necessary to direct the thesis to those

problems that are the most critical and that become even more important in the distributed environment. The environment of a distributed computer facility required examination of operating system characteristics, data base systems, and computer network interconnection. An attempt was made to highlight those aspects of the solutions to the protection problems that have been applied to these elements individually in order to identify any techniques that could be easily merged into a mechanism for the DCF. It was considered crucial that the chosen approach lead to a mechanism that is understandable and justifiable. A consideration of data base issues lent insight into access decisions that might be event-sensitive, value-sensitive, or state-sensitive. The proposed mechanisms are aimed most directly at handling criteria that can be characterized as event or value sensitive.

Having looked at some of the interesting features of each of the components, the capability scheme was chosen as the approach with the most potential for an interesting and versatile solution to the protection problems in the DCF. The solution to the protection problems was sought in a world defined by a new approach to the handling of objects in the system. This approach had been developed by Reed as a means for implementing atomic actions in distributed systems <Reed78>. Reed's description of an object as a sequence of immutable versions that represent the state of the object over time provided the flexibility to define a world in which timestamps were naturally associated with every object in the system. The new perception of the world was fully

described and then the resulting effects on the typical capability system were investigated.

We discovered that capabilities need not become immediately effective upon their generation. Further, it was not necessary for the object to which access was being authorized to exist at the time the capability was generated. The integration of timestamps, capabilities, and the new world of objects resulted in mechanisms that are unique in their versatility and flexibility. In addition, the mechanism provides an elegant solution to the revocation problem. Despite the distributed nature of the DCF, the revocation of access privileges and the control of propagation do not require further extension of the basic DCF capability mechanism.

The thesis includes a detailed consideration of the mechanisms for certain of the features defined. A subtle distinction between the creator of an object, in the traditional sense, and the "updater" of the object is recognized and the attendant implications on the mechanism are highlighted. The distinction between versions of an object and the logical concept with which all versions are associated is crucial to providing for capabilities to future objects, for capabilities to the most current versions, and for capabilities for the creators of objects. Finally, the passing of capabilities among the nodes in the DCF is briefly examined to assure that the integrity of the capability will be

maintained regardless of the nature of the environment in which it must exist.

The suggested mechanisms, of course, are not without their problems. Most obvious is the large size of the capabilities as defined in the DCF. Three of the five fields of a capability require timestamp values that need to be accurate to the degree that every operation and entity in the system is uniquely identifiable through time. As the mechanisms stand, they are clearly too inefficient to be used for authorization checks on every memory access. After further examination and evaluation, the mechanisms may prove to be most useful for inter-domain accesses, where the domains correspond to logical subsystems such as the guardians described in <LCS_78>. A more efficient form of capability, used primarily as protection against errors, can be employed internally within these domains.

This criticism also manifests another instance of the object granularity problem; in general, the DCF capabilities will not protect data items of the size of individual integers but rather higher-level data base objects, for instance. However, this may lead to recursively more complex problems, i.e. if a higher-level data base object consists of a number of components all of which can be updated independently of the others then how is access decided if the criterion used for the particular higher-level object is version dependent?

The other critical problem with the current design is the assumption that the system clocks at each of the DCF nodes are guaranteed against subversion, i.e. the clock cannot be set forward or back. The occurrence of such an act would render ineffective the entire mechanism in which the access criteria are time value dependent.

Finally, we must consider a few ideas that arose peripheral to the primary design motivation but that may stand on their own as feasible research areas. For instance, the use of capabilities that become effective at some future date and that will be effective for only a limited time interval provides a reasonable scheme for resource scheduling. One could conceive of systems that operated under a policy of resource reservations and which guaranteed the particular resource or service but only up to the expiration date.

Also, the existence and usefulness of the remote capability segment should be further investigated. In defining such an object we, in effect, granted capabilities to a different type of subject, i.e. we granted the capability for the non-local object to a node rather than to an individual user or program entity. Further, the remote capability segment imposed an additional level of indirection through which access of the target object must progress. It is not clear that this is the most appropriate means of handling capabilities across several nodes nor is it clear that the proposed mechanism was exploited to the fullest, i.e. a means by which a subject can do certain operations from only

certain specific nodes may be desirable and may be feasible via use of the remote capability segment.

This thesis essentially provides a consideration of the issues raised in designing protection mechanisms for operation in a diverse and distributed environment. The ingredients used in the design had all been individually employed in a number of solutions to a variety of other problems that arise in designing distributed systems. The contribution of this thesis is in utilizing them in solving the problems of protection. When combined to form a unified approach to the protection problem, the sum total of all the elements yielded a protection mechanism that may reasonably support a myriad of security policies in a distributed computer facility.

References

[Astr76] Astrahan, M. M. et al, "System R: Relational Approach to Database Management," ACM Transactions on Database Systems, vol. 1 no. 2, June 1976, pp.97-137.

[Banc77] Bancilhon, Francoism, and Spyratos, Nicolas, "Protection of Information in Relational Data Bases," Proceedings of the Third International Conference on Very Large Data Bases, Japan, October 6-8 1977, pp.494-500.

[Beck78] Becker, H.B., "Let's Put Information Networks in Perspective," Datamation, vol.24 no.3, March 1978, pp.81-86.

[Bens72] Bensoussan, A., et al, "The Multics Virtual Memory: Concepts and Design," Communications of the ACM, vol.15 no.5, May 1972, pp.308-318.

[Boot72] Booth, G., "The Use of Distributed Data Bases in Information Networks," Computer Communication Impacts and Implications, S. Winkler (ed.), Proceedings of the First International Conference on Computer Communication, Washington, D.C., October 1972, pp.371-376.

[Bran73] Branstad, D.K., "Security Aspects of Computer Networks," AIAA Computer Network Systems Conference, April 1973, paper 73-427.

[Bran75] Branstad, D.K., "Encryption Protection in Computer Data Communications," Fourth Data Communications Symposium, Quebec, October 1975.

[Cana74] Canady, R.H., et al, "A Back-end Computer for Data Base Management," Communications of the ACM, vol.17 no.10, October 1974, pp.575-582.

[Cham75] Chamberlin, D. D., Gray, J. N., and Traiger, I. L., "Views, Authorization, and Locking in a Relational Data Base System," AFIPS Conference Proceedings, 1975, pp.425-430.

[CISR76] Center for Information Systems Research, DRAFT:"Proposal for Research on the Design of a Family of Database Systems," December 1976.

[Cohe75] Cohen, E., and Jefferson,D., "Protection in the HYDRA Operating System," Proceedings of the 5th Symposium on Operating System Principles, ACM Operating Systems Review vol9 no.5, November 1975, pp.141-160.

[Comb75] Comba, Paul G., "Needed: Distributed Control," Proceedings of the International Conference on Very Large Data Bases, D.S. Kerr (ed.), ACM, September 1975, pp.364-375.

[Conw72] Conway, R.W., et al, "On the Implementation of Security Measures in Information Systems," Communications of the ACM, vol.15 no.4, April 1972, pp.211-220.

[Cook78] Cook, Douglas, "The Cost of Using the CAP Computer's Protection Facilities," ACM Operating Systems Review, vol.12 no.2, April 1978.

[Cott77] Cotton, Ira W., "Computer Network Interconnection," Proceedings of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, pp.3-18.

[Date77] Date, C.J.An Introduction to Data Base Systems ed. 2, Addison-Wesley Pub. Co., 1977, pp.536.

[DenD76] Denning, D.E., "A Lattice Model of Secure Information Flow," Communications of the ACM, vol.19 no.5, May 1976, pp.236-243.

[Denn66] Dennis, Jack B., and Van Horn, Earl C., "Programming Semantics for Multiprogrammed Computations," Communications of the ACM, vol.9 no.3, March 1966, pp.143-155.

[DeNo77] Denoia, Lynn A., "Distributed Data Base Issues," Naval Underwater Systems Center, New London, Connecticut, Technical Memorandum #771248, December 1977.

[DeNo78] DeNoia, Lynn A., "Performance and Timeliness in a Distributed Database," Ph.D. Thesis, Brown University, in progress.

[Depp76] Deppe, Mark, and Fry, James, "Distributed Data Bases: A Summary of Research," Computer Networks vol.1 no.2, September 1976, pp.130-138.

[Dono75] Donovan, J.J., and Madnick,S.E., "Hierarchical Approach to Computer System Integrity," IBM Systems Journal vol.14 no.2, 1975, pp.188-202.

[Down77] Downs, Deborah, and Popek,Gerald J., "A Kernel Design for a Secure Data Base Management System," Proceedings of the 3rd International Conference on Very Large Data Bases, October 1977.

[Engl74] England, D., "Capability Concept Mechanisms and Structure in System 250," in the Proceedings of the IRIA International Workshop on Protection in Operating Systems, Institut de Recherche d'Informatique et de Automatique, France, 1974, pp.63-82.

[Fabr74] Fabry, R.S., "Capability-Based Addressing," Communications of the ACM, vol. 17 no. 7, July 1974, pp.403-412.

[Fagi78] Fagin, Ronald, "On an Authorization Mechanism," ACM Transactions on Database Systems, vol. 3 no. 3, September 1978, pp.310-319.

[Farr76] Farrell, J., "The Datacomputer-A Network Data Utility," Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks, May 1976, pp.352-364.

[Fer75a] Fernandez, E. B., Summers, R. C., and Coleman, C. D., "An Authorization Model for a Shared Data Base," Proceedings of the 1975 SIGMOD International Conference on the Management of Data, pp.24-31.

[Fer75b] Fernandez, Eduardo, B., Summers, Rita C., and Lang, Tomas, "Definition and Evaluation of Access Rules in Data Management Systems," Proceedings of the International Conference on Very Large Data Bases, Framingham, Mass., 1975, pp.268-285.

[Feus73] Feustel, E.A., "On the Advantages of Tagged Architecture," IEEE Transactions on Computers, vol. C-22 no. 7, July 1973, pp.644-656.

[Gain78] Gaines, R. Stockon, and Shapiro, Norman Z., "Some Security Principles and Their Application to Computer Security," ACM Operating Systems Review vol.12 no.3, July 1978, pp.19-28.

[Gord77] Gordon, Robert, Notes on the Panal Session on Capability Systems, 6th Symposium on Operating System Principles, November 1977, Prime Computer, Framingham, Mass.

[Grah72] Graham, G.Scott, and Denning,Peter J., "Protection-Principles and Practice," AFIPS Spring Joint Computer Conference, vol.40, 1972, pp.417-424.

[Grif76] Griffiths, Patricia P., and Wade,Bradford W., "An Authorization Mechanism for a Relational Database System," ACM Transactions on Database Systems vol.1 no.3, September 1976.

[Habe76] Haberman, A., et al, "Modularization and Hierarchy in a Family of Operating Systems, Communications of the ACM, vol.19 no.5, May 1976.

[Harr76] Harrison, M.A., and Ruzzo, W.L., "Protection in Operating Systems," Communications of the ACM, vol.19 no.8, August 1976, pp.461-471.

[Herb78] Herbert, A.J., "A New Protection Architecture for the Cambridge Capability Computer," ACM Operating Systems Review vol.12 no.1, January 1978.

[Hort78] Horton, Forest Woody, "Reducing the Federal Paperwork Burden," Datamation vol.24 no.4, April 1978.

[Hsi75a] Hsiao, David K., "Recent Advances in Information Secure Systems Research," Proceedings of the 4th Texas Conference on Computing Systems, November 1975, pp.2A-2.1 to 2A-2.9.

[Hsia76] Hsiao, David K., and Baum, Richard I., "Information Secure Systems," from Advances in Computers, vol.14, Academic Press Inc., 1976, pp.231-272.

[Hsia77] Hsiao, D.K., and Madnick, S.E., "Database Machine Architecture in the Context of Information Technology Evolution," Proceedings of the 3rd International Conference on Very Large Data Bases, October 1977.

[Ilif72] Iliffe, J.K., "Basic Machine Principles," American Elsevier Inc., New York, 1972.

[Jone75] Jones, A.K., and Lipton, Richard, "The Enforcement of Security Policies for Computation," Proceedings of the 5th Symposium on Operating Systems Principles, ACM Operating Systems Review vol.9 no.5, November 1975, pp.197-206.

[Jone78] Jones, A.K., and Liskov, B.H., "A Language Extension for Expressing Constraints on Data Access," Communications of the ACM, vol.21 no.5, May 1978, pp.358-367.

[Karg77] Karger, Paul A., "Non-Discretionary Access Control for Decentralized Computing Systems," Laboratory for Computer Science, Massachusetts Institute of Technology, MIT/LCS/TR-179, May 1977.

[Kent76] Kent, Stephen T., "Encryption-Based Protection Protocols for Interactive User-Computer Communication," M.I.T. Laboratory for Computer Science Technical Report TR-162, May 1976.

[LamL78] Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System," CACM vol.27 no.7, July 1978, pp.558-565.

[Lamp71] Lampson, B.W., "Protection," in Proceedings of the Fifth Annual Princeton Conference on Information Sciences and Systems, 1971, pp.437-443. Reprinted in ACM Operating Systems Review, January 1974.

[Lamp73] Lampson, B.W., "A Note on the Confinement Problem," Communications of the ACM, vol.16 no.10, October 1973, pp.613-615.

[Lamp76] Lampson, Butler W., and Sturgis, Howard E., "Reflections on an Operating System Design," Communications of the ACM, vol.19 no.5, May 1976, pp. 251-266.

[LeLa77] LeLann, Gerard, "Distributed Systems--Towards a More Formal Approach," IFIP Congress 1977, Toronto, Canada, August 1977.

[Levi73] Levin, Eugene, "The Future Shock of Information Networks," Proceedings of the AIAA Computer Network Systems Conference, April 1973, paper 73-439.

[LevP77] Levine, Paul H., "Facilitating Interprocess Communication in a Heterogeneous Network Environment," M.I.T. Laboratory for Computer Science Technical Report TR-184, July 1977.

[LCS_78] Laboratory for Computer Science, Distributed Systems Group Progress Report 1978, "Semantics of Distributed Computing," Massachusetts Institute of Technology, Cambridge, Mass. 02139.

[Lind74] Linden, Theodore A., "Capability-Based Addressing to Support Software Engineering and System Security," Proceedings of the 3rd Texas Conference on Computing Systems, November 1974,

pp.8-5-1 to 8-5-6.

[Lind76] Linden, T.A., "Operating Systems Structures to Support Security and Reliable Software," ACM Computing Surveys vol.8 no.4, December 1976, pp.409-445.

[Lind78] Lindsay, B. and Gligor, V., "Migration and Authentication of Protected Objects," IBM Research Report RJ-2298, August 1978.

[Luni79] Luniewski, A.W., "The Architecture of an Object Based Personal Computer," Ph.D. Thesis, Massachusetts Institute of Technology, in progress.

[Madn77] Madnick, S.E., "Trends in Computers and Computing:The Information Utility," Science vol.195 no.4283, 18 March 1977.

[Mart77] Martin, James, Computer Data Base Organization, ed.2, Prentice-Hall Inc., 1977, pp.713.

[Mary78] Maryanski, Fred J., "A Survey of Developments in Distributed Data Base Management Systems," Computer vol.11 no.2, February 1978, pp.28-38.

[Merk78] Merkle, Ralph C., "Secure Communications Over Insecure Channels," Communications of the ACM, vol.21 no.4, April 1978, pp.294-299.

[Mins76] Minsky, Naftaly, "Intentional Resolution of Privacy Protection in Database Systems," Communications of the ACM, vol.19 no.3, March 1976, pp.148-159.

[Moha78] Mohan, G., "Survey of Operating Systems Research, Designs and Implementation," ACM Operating Systems Review vol.12 no.1, January 1978.

[Need77] Needham, R., and Walker, R.D.H., "The Cambridge CAP Computer and it's Protection System," Proceedings of the 6th Symposium on Operating System Principles, ACM Operating Systems Review

vol.11 no.5, November 1977, pp.1-10.

[Padl78] Padlipsky, M.A., "An Architecture for Secure Packet-Switched Networks," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, August 29-31 1978, pp.54-68.

[Peeb78] Peebles, Richard, and Manning, Eric, "System Architecture for Distributed Data Management," Computer vol.11 no.1, January 1978.

[Pope74] Popek, Gerald J., "Protection Structures," Computer vol.7 no.6, June 1974, pp.22-33.

[Pope75] Popek, Gerald J., "On Data Secure Computer Networks," Proceedings of the ACM SIGCOMM/SIGOPS Interprocess Communication Workshop, ACM Operating Systems Review vol.9 no.3, July 1975.

[Rede74] Redell, David D., "Naming and Protection in Extendible Operating Systems," M.I.T. Laboratory for Computer Science Technical Report TR-140, November 1974.

[Reed78] Reed, D. P., "Naming and Synchronization in a Decentralized Computer System," M.I.T. Laboratory for Computer Science Technical Report TR-205, September 1978.

[Rive78] Rivest, R. L., Shanii, A, and Adleman, L., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," Communications of the ACM, vol.21 no.2, February 1978, pp.120-126.

[Roth77] Rothnie, James B., and Goodman, Nathan, "A Survey of Research and Development in Distributed Database Management," Proceedings of the 3rd International Conference on Very Large Data Bases, October 1977.

[Salt74] Saltzer, J.H., "Protection and the Control of Information in Multics," Communications of the ACM, vol.17 no.7, July 1974, pp.388-402.

[Salt75] Saltzer, Jerome H., and Schroeder, Michael D., "The Protection of Information in Computer Systems," Proceedings of the IEEE vol.63 no.9, September 1975, pp.1278-1308.

[Sal78a] Saltzer, J., "On Digital Signatures," ACM Operating Systems Review vol.12 no.2, April 1978.

[Sal78b] Saltzer, J., "Research Problems of Decentralized Systems with Largely Autonomous Nodes," ACM Operating Systems Review vol.12 no.1, January 1978, pp.43-52.

[Schr75] Schroeder, M.D., "Engineering a Security Kernel for Multics," Proceedings of the 5th Symposium on Operating System Principles, November 1975.

[Schr77] Schroeder, Michael D., Clark, David D., and Saltzer, Jerome H. "The Multics Kernel Design Project," The Proceedings of the Sixth Symposium on Operating System Principles, November 1977, ACM Operating Systems Review vol.11 no.5, pp.43-56.

[Senk73] Senko, M.E., "Data Structures and Data Accessing in Data Base Systems: Past, Present, Future," IBM Systems Journal vol. 12 no.1, 1973.

[Sha77a] Shankar, K.S., "The Total Computer Security Problem: An Overview," Computer vol.10 no.6, June 1977, pp.50-73.

[Sha77b] Shankar, K.S., and Chandersekaran, C.S., "The Impact of Security on Network Requirements," Proceedings of Trends and Applications 1977: Computer Security and Integrity, May 1977.

[Shos72] Shoshani, Arie, "Data Sharing in Computer Networks," WESCON Conference, September 1972.

[Ston76] Stonebraker, Michael, and Rubinstein, Peter, "The INGRES Protection System," Proceedings of the ACM Annual Conference, October 1976, pp.80-84.

[Tsic77] Tsichritzis, D., "Research Directions in Data Base Management," ACM SIGMOD Record vol.9 no.3, 1977.

[Wink74] Winkler, Stanley, and Danner, Lee, "Data Security in the Computer Communication Environment," Computer vol.7 no.2, February 1974, pp.23-31.

[Wior78] Wiorkowski, Gabrielle, and Wiorkowski, John J., "Does A Database Management System Pay Off," Datamation vol.24 no.4, April 1978.

[Wulf74] Wulf, W., "Hydra: The Kernel of a Multiprocessor Operating System," CACM vol.17 no.6, June 1974, pp.337-345.