IMPLEMENTING CAPABILITY-BASED PROTECTION USING ENCRYPTION

by

D.L. Chaum and R.S. Fabry

Memorandum No. UCB/ERL M78/46

17 July 1978

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley, CA 94720

IMPLEMENTING CAPABILITY-BASED PROTECTION USING ENCRYPTION

by

D.L. Chaum and R.S. Fabry

Memorandum No. UCB/ERL M78/46

17 July 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Implementing Capability-Based Protection Using Encryption*

D.L. Chaum and R.S. Fabry

University of California, Berkeley

ABSTRACT

Use of encryption allows a system to provide self-authenticating capabilities. Such capabilities can be mixed freely with other information without compromising system security. They can also be used to construct a network-based protection system in which secure nodes are not endangered by less secure nodes. The approach allows capabilities to be used on systems which are not capability based and allows capabilities to be kept by individuals much as passwords are. A number of constraints of earlier implementations are relaxed. A particularly simple implementation of type extension becomes possible.

Key Words and Phrases: Capabilities, Protection, Encryption, Type Extension, Networks, Access Control.

CR Categories: 4.35

* This work was partially supported by the National Science Foundation under NSF Grant MCS75-23739.

Implementing Capability-Based Protection Using Encryption

D.L. Chaum and R.S. Fabry

Introduction

It is generally agreed that strong encryption algorithms either exist today or will exist within a few years. Such algorithms allow new solutions to a number of old problems in computer systems. Solutions based on encryption have somewhat different properties than conventional solutions. In this paper we contrast conventional capability-based protection schemes with those implemented using encryption.

We consider the application of cryptography to protecting capabilities themselves; we do not consider the use of encryption for protecting data [8,1].

Encryption is a technique with which information is mapped into a form which reveals the original information when an inverse mapping is applied, but otherwise reveals nothing about the original information [3]. The mapping is assumed to be drawn from some large class of mappings. The information specifying which member of the class has been used to encrypt some piece of information is called the key of the encrypted information because it is needed to perform the mapping or the inverse mapping. We assume that the size and nature of the class of mappings is such that there is a

negligible probability of determining the key (or of performing a mapping without it) by carrying out some feasible computation.

A capability is a bit pattern which functions both as the address of some object (such as a file) within a computer system, and as authorization for its possessor to access the addressed object [2,5,11]. Typically, a capability is implemented as an object identifier concatenated with some access bits that specify which of the accesses defined for the object (such as reading and writing) are allowed to the possessor of the capability. Figure 1 shows such a capability representation.

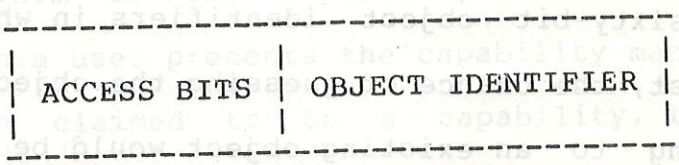

Figure 1. Typical Capability Representation.

Authentication of Capabilities

The traditional implementation of capability-based protection enforces a distinction between capabilities and all other forms of information in the system [5]. In such systems, only the capability manager routines which create and interpret capabilities can establish a particular bit pattern as a capability. Even if a user knows the bit pattern

for a desired capability, there is no way the user can cause the bit pattern to be interpreted as a capability. In such systems capabilities are said to be externally authenticated. Two approaches have been used for external authentication. In the tagged approach, each word of information in the system is tagged to indicate whether it contains a capability or not. In the partition approach, capabilities appear only in certain segments or tables which are known to contain only capabilities.

Instead of being externally authenticated, capabilities may be self-authenticating. If capabilities did not include access bits, self-authenticating capabilities could be fabricated by choosing the object identifier at random from a sparsely filled identifier space. For example, in a system with sixty-bit object identifiers in which a million objects exist, the chance of guessing the object identifier corresponding to an existing object would be one in a million million. If it were possible to test a bit pattern every millisecond, it would take over thirty years, on the average, to find one valid object identifier. The probability of a failure in a protection mechanism due to such an attack could be made negligible compared to other possible failures by adjusting the fraction of the identifier space which is valid. The risk could also be decrease by techniques like logging attempted use of invalid object identifiers.

If one were to merely concatenate an access bit field with such a self-authenticating capability, a user who understood the structure of the capability representation could modify the access bits to authorize any desired access. Such a possibility would eliminate much of the usefulness of the access bits.

Encryption can be used to make it very difficult for a user to change the access bits in a self-authenticating capability. The capability manager requires a secret encryption key. It first constructs an unencrypted capability for a new object by concatenating the access bits and an unused object identifier. The capability manager then encrypts the capability using its secret key. Only the encrypted form of the capability is distributed to its users. When a user presents the capability manager with a bit pattern claimed to be a capability, the capability manager can decrypt the bit pattern, and thereby obtain the corresponding unencrypted capability. The redundancy in the identifier space not only authenticates the object identifier but also authenticates the access bits.

This solution makes the common assumption that a block encryption scheme can provide authentication under a known plaintext attack [3]. In particular, even a user who has examined many capabilities and their unencrypted representation must not have gained insight into how to form new encrypted capabilities.

There are two costs to the self-authenticating capability scheme described above. First, the identifier space must be sparsely filled. Fortunately, the size of the object identifier grows logarithmically as a function of the number of potential object identifiers. As hinted above, even a sixty-four bit capability may allow enough redundancy for many applications. Second, encryption and decryption steps are introduced. There may be a delay due to encryption and decryption even with the use of hardware encryption devices. If capabilities are cached [5], the average delay may be diminished.

Networks

In a network which uses protection based on externally authenticated capabilities which can be passed between nodes, all of the nodes of the network must work together to ensure that the separation between capabilities and other information is maintained. In this respect, each node is no more secure than the weakest node. In such a situation, a user might have to be concerned that a node the user does not trust has either accidentally or maliciously fabricated an improper capability for one of the user's objects.

Self-authenticating capabilities behave similarly to passwords. It is possible for a user to obtain the bit pattern corresponding to such a capability from a computer system, to retain the bit pattern outside the computer system, and to re-enter the bit pattern at a later time.

The use of self-authenticating capabilities allows multiple capability managers, each with its own key. With multiple capability managers, a capability can be used only by presenting it to the appropriate manager. The holder of a capability must remember, explicitly or implicitly, which capability manager created the capability.

One capability manager cannot be compromised by the actions of a second capability manager which uses a different key unless the first explicitly relies on the second for some service. In a network, a capability manager at each node can have its own key. A user can then obtain a capability for an object and pass that capability around to users and nodes which are trusted without being concerned that untrustworthy nodes of the network will somehow subvert the protection mechanism.

Type extension

The definition of a complex system as a hierarchy of levels has been found to be a valuable technique for structuring systems [4,7]. In object-oriented systems, such as those using capabilities, this implies that many types of objects will be unknown to the base-level of a system and that type extension facilities will be used to implement the object types of one level in terms of those of lower levels [6,11]. The set of programs which implement a new type of object is called the type manager for that type of object.

In systems based on externally authenticated capabilities, a type manager cannot issue capabilities for new objects directly. The type manager might, for example, have to call the system capability manager, passing as a parameter a capability which authorizes the bearer to ask for capabilities for objects of the new type [6,11].

With self-authenticating capabilities, a type manager can choose a key of its own and use that key to generate its own capabilities. That is, it can manage its own capabilities. There is no need for capabilities for different types of objects to be the same length. The size of the access bit field can vary depending on the object type. The object identifier part of the new capability can contain one or more capabilities for the object used to implement the new object. When the object identifier includes capabilities, the redundancy in the included capabilities may be used to authenticate the including capability.

An advantage of multiple capability managers, each with its own key, is that the decision concerning the sparseness of the identifier space is separate for each capability manager. In particular, the length of the object identifier need not grow as the size of the network or the number of types of objects grows.

Mapping

With externally authenticated capabilities, a map is

used to implement a mapping from the object identifier in a capability to the implementation of the object for two reasons. First, it is almost always necessary to have more information about the implementation than can be put into a relatively small, fixed-length object identifier. Second, it may be necessary to change implementation objects as when files are relocated on a disk.

With self-authenticating capabilities, the size of an object identifier can vary with the type of the object. (Variable-size capabilities for objects of a single type are even possible.) Because changing the implementation is usually relevant only for objects implemented directly in terms of physical resources, one may expect that most higher-level type managers using self-authenticating capabilities will not need a map.

A map can perform a number of functions in addition to those mentioned above, however. For example, revocation can be implemented using a map [9]. If a system defines a notion of the identity of an accessor, a map can also be used to implement access control lists [10] or logging. If such features are required of a particular object manager, the object manager will still need some of the services of a map. The object manager may use an internal map or rely on an external map. In either case the map need be no more complex than that required with externally-authenticated capabilities.

Auditing

Auditing is a concern in some capability-based systems. With externally authenticated capabilities, all of the storage of a system which might contain capabilities can be scanned during an audit (after stopping the system, for example, so as to obtain consistent results.) In this way, it can be determined whether the protection state has certain properties or a snapshot of the current protection state can be obtained. With self-authenticating capabilities, it is not, in general, even possible to locate all capabilities.

The utility of auditing is limited by the fact that capabilities are often entrusted to programs whose behavior cannot be easily discovered. Independent of how capabilities are authenticated, systems for which auditing is important must rely primarily on techniques like revocation, access control lists and logging.

Conclusion

Capabilities can be made self-authenticating by the use of encryption. The major advantage of such a scheme is that capabilities need not be kept exclusively under the control of universally trusted authorities. Such capabilities can be used in networks where nodes do not necessarily trust one another and can be kept outside computer systems. Type extension works well with self-authenticating capabilities.

References

1. Bayer, R. and Metzger, J.K. On the encipherment of search trees and random access files. ACM Trans. Database Syst. 1, 1 (March 1976), 37-52.

2. Dennis , J.B and Van Horn, E. Programming semantics for multiprogrammed computations. Comm. ACM 16, 3 (March 1973), 143-155.

3. Diffie, W. and Hellman, M. New directions in cryptography. IEEE Trans. Inform. Theory 22, 6 (Nov. 1976), 644-654.

4. Dijkstra, E.W. The structure of THE multiprogramming system. Comm. ACM 11, 5 (May 1968), 341-346.

5. Fabry, R.S. Capability-based addressing. Comm. ACM 17, 7 (July 1974), 403-412.

6. Lampson, B.W. and Sturgis, H.E. Reflections on an operating system design. Comm. ACM 19, 5 (May 1976), 236-243.

7. Parnas, K.L. A technique for software module specification with examples. Comm. ACM 15, 5 (May 1972), 330-336.

8. Petersen, H.E. and Turn, R. System implications of information privacy. Proc. AFIPS 1967, SJCC, Vol. 30, AFIPS Press, Montvale, N.J. pp. 291-300.

9. Redell, D. Naming and protection in extensible operating systems. Ph.D. Dis., Comptr. Sci. Div., Elect. Eng. and Comptr. Scis. Dept., U. of Calif., Berkeley, 1974. Also as: Tech. Rep. No. MAC-TR-140, Project MAC, MIT, Cambridge, Mass. 1974.

10. Saltzer, J.H. Protection and control of information sharing in Multics. Comm. ACM 17, 7 (July 1974), 388-402.

11. Wulf, W. et al. Hydra: The kernel of a multiprocessor operating system. Comm. ACM 17, 6 (June 1974), 337-345.