Proc. of the 14th NIST-NCSC National Computer Security Conference, Washington D.C., October 1991, pages 2-11

A DISTRIBUTED IMPLEMENTATION OF THE TRANSFORM MODEL

Ravi S. Sandhu and Gurpreet S. Suri

Center for Secure Information Systems
and
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

ABSTRACT The Transform access-control model is based on the concept of transformation of access rights. It has previously been shown that Transform unifies a number of diverse access control mechanisms such as amplification, copy flags, separation of duties and synergistic authorization. It has also been shown that Transform has an efficient algorithm for safety analysis of the propagation of access rights (i.e., the determination of whether or not a given subject can ever acquire access to a given object). In this paper we propose a distributed implementation of Transform. Our design is based on capabilities with identities of subjects buried in them. This ensures unforgeability of capabilities as well as enables enforcement of "mandatory" controls on propagation of capabilities from one subject to another. The design provides for immediate, selective, partial and complete revocation on a temporary as well as permanent basis.

Keywords: Distributed Systems, Secure Architectures, Capabilities

1 INTRODUCTION

The need for access controls arises in any computer system that provides for controlled sharing of information and other resources among multiple users. Access control models (or protection models) provide a framework for specifying, analyzing and implementing security policies in multi-user systems. These models are typically defined in terms of the well-known abstractions of subjects, objects and access rights with which we assume the reader is familiar. A wide variety of access-control models have been described in the literature [3, 4, 10, 12, 16, for instance]. Unfortunately very few have been implemented or have even influenced implementations of actual systems.*

In this paper we take a step towards closing this gap between theory and practise. Our principal contribution is the outline of a distributed implementation of the recently proposed Transform model [17]. Transform derives its name from its central concept of transformation of access rights. The idea is that access rights get transformed as they are propagated from one subject to another, e.g., a security-officer who has the review right for a document may propagate the release right for the document to the document's author. It has previously been shown [17] that Transform elegantly unifies a number of seemingly different access control mechanisms such as amplification [5], copy flags [12], separation of duties [4] and synergistic authorization [14]. It has also been shown [17] that there are efficient algorithms for the safety problem in Transform (i.e., the determination of whether or not a given subject can ever acquire access to a given object).

*The notable exception is the Bell-LaPadula model [3] whose strong influence on military systems has been formally incorporated in evaluation criteria [8].

Thus Transform incorporates practically useful expressive power while allowing for safety analysis. Transform is actually a special case of the Schematic Protection Model (SPM) [16]. Like Transform, SPM also exhibits strong safety properties. This is in contrast to the weak safety properties of the access-matrix model commonly known as HRU [10]. Both HRU and SPM have undecidable safety in general [10, 18]. In HRU safety becomes undecidable under very weak assumptions, notably the bi-conditional monotonic case of [11]. On the other hand safety in SPM remains decidable under very strong assumptions, notably the acyclic attenuating case of [16]. In particular Transform falls outside the known decidable cases for HRU but well within the known decidable cases for SPM [17].

Our implementation proposal for Transform is strongly influenced by the identity-based capability architecture proposed by Gong [9]. The concept of embedding the identity of a subject in a capability in distributed systems has been known for some time [6]. It ensures that capabilities cannot be forged or propagated from one subject to another without intervention of trusted software. Gong's architecture is based on the familiar client-server model of services in a distributed system and includes mechanisms for revocation which were missing in earlier proposals such as [6]. We have extended Gong's proposal to accommodate Transform. In particular the concept of strongly typed subjects and objects, which is essential to Transform, has been incorporated.

The paper is organized as follows. Section 2 reviews the Transform model to the extent required for our objectives in this paper. Section 3 discusses distributed capability-based architectures in general and motivates our choice of building on Gong's approach. Section 4 describes our proposed implementation for Transform. The protocols involved in creation, propagation and revocation are presented. An example of the implementation is presented in section 5. The paper is concluded in section 6 with a discussion and proposals for future research.

2 THE TRANSFORM MODEL

The Transform model [17] was obtained by identifying the common foundation underlying a variety of different access-control mechanisms proposed in the literature. These include amplification [5], copy flags [12], separation of duties [4] and synergistic authorization [14]. Considered in isolation these mechanisms are diverse and were largely proposed independently of each other. They all appear to be desirable and should be supported by any system which claims generality. However simply lumping them together results in a complex system with many unrelated mechanisms.

Transform introduces the unifying concept of transformation of rights which can occur in two different ways.

1. *Self transformation* or *internal transformation* allows a subject who possesses certain rights for an object to obtain additional rights for that object.

2. *Grant transformation* or *external transformation* occurs in the granting of access rights by one subject to another. The general idea is that possession of a right for an object by a subject allows that subject to give some other right for that object to another subject.

In addition Transform is based on the *strong typing* of subjects and objects, i.e., subjects and objects are classified into types when they are created and their type cannot change thereafter. Much of the power of transformation derives from predicating the ability to transform on the types of subjects and objects involved.

A security policy is stated in Transform by specifying the following (finite) components.

1. Disjoint sets of subject types TS object types TO and rights R.

2. A can-create function $cc : \mathrm{TS} \rightarrow 2^{\mathrm{TO}}$.

3. Create-rules $cr : \mathrm{TS} \times \mathrm{TO} \rightarrow 2^{\mathrm{R}}$.

4. An internal transformation function $itrans : \mathrm{TS} \times \mathrm{TO} \times 2^{\mathrm{R}} \rightarrow 2^{\mathrm{R}}$.

5. A grant transformation function $grant : \mathrm{TS} \times \mathrm{TS} \times \mathrm{TO} \times 2^{\mathrm{R}} \rightarrow 2^{\mathrm{R}}$.

The notation 2^{X} denotes the power set of X, i.e., the set of all subsets of X. These components of a Transform specification are explained in turn below.

The sets TS and TO define the subject types and object types respectively. For example subject types might be faculty, student, guest, etc., and object types might be file, mail-message, bulletin-board, etc. R defines the set of rights or privileges in the system, e.g., read, write, execute, etc.

There are two issues involved in object creation.† Firstly subjects need authorization to create objects. Secondly the rights obtained as a result of creation must also be specified. Transform authorizes creation by means of the can-create function *cc*. The interpretation of

$$cc(\mathrm{u}) = \{\mathrm{o}_1, \mathrm{o}_2, \ldots, \mathrm{o}_\mathrm{k}\}$$

is that a subject of type u is authorized to create objects of type o_1 and objects of type o_2, etc. The effect of creation is defined by create-rules. The interpretation of

$$cr(\mathrm{u}, \mathrm{o}) = \{\mathrm{r}_1, \mathrm{r}_2, \ldots, \mathrm{r}_\mathrm{p}\}$$

is that when a subject U of type u creates an object O of type o the creator U obtains the rights r_1, r_2, ..., r_p for O. For example if *cc*(user) = {file} and *cr*(user,file) = {own} the creator of a file gets the own right for it. For readability we will usually drop the set parenthesis around singleton sets, for instance by writing *cc*(user) = file and *cr*(user,file) = own.

Authorization for internal transformation is specified by the internal transformation function *itrans*. The interpretation of

$$itrans(\mathrm{u}, \mathrm{o}, \{\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}\}) = \{\mathrm{y}_1, \ldots, \mathrm{y}_\mathrm{m}\}$$

is that a subject of type u who has *all* the x_i rights specified on the left hand side for an object of type o can obtain the rights y_1, ..., y_m for that object by internal transformation. For example, the policy that possession of the w (write) privilege for a file implies possession of the r (read) privilege is easily stated as follows.‡

$$itrans(\mathrm{user}, \mathrm{file}, \mathrm{w}) = \mathrm{r}$$

Another example of internal transformation occurs in situations described as synergistic authorization in [14]. For instance consider a situation where a scientist (abbreviated as sci) needs approvals from a security officer and a patent officer before he can release a document (abbreviated as doc) for publication. Say these two approvals are respectively signified by possession of the a_s and a_p rights. We can express this policy as follows.

$$itrans(\mathrm{sci}, \mathrm{doc}, \{\mathrm{own}, \mathrm{a}_\mathrm{s}, \mathrm{a}_\mathrm{p}\}) = \mathrm{release}$$

That is, a scientist who owns a document and possesses the two approvals can acquire the release right for that document.

Grant transformations are authorized by the grant transformation function *grant*. The interpretation of

† There must be provision for creation of subjects in any realistic system. In practise creation of subjects is often strictly controlled by some distinguished system administrator or security officer. Such creation can be considered as occurring outside the normal scope of the system.

‡ In multilevel systems this policy would amount to prohibiting write-up.

$$grant(\mathrm{u}, \mathrm{v}, \mathrm{o}, \{\mathrm{x}_1,\ldots,\mathrm{x}_n\}) = \{\mathrm{y}_1,\ldots,\mathrm{y}_m\}$$

is that a subject of type u who has *all* the x_i rights specified on the left hand side for an object of type o can grant *one or more* of the rights $y_1, \ldots, y_m$ for that object to a subject of type v. A common example of grant transformation occurs with the copy flag c which controls whether the granted privilege can itself be further granted or not. For instance the following

$$\begin{aligned} grant(\text{user, user, file, xc}) &= \{\text{xc, x}\} \\ grant(\text{user, user, file, x}) &= \phi \end{aligned}$$

defines the (unlimited) copy flag. Here a user who has the xc privilege for a file can grant the xc privilege or the x privilege to another user, whereas a user with the x privilege for the file cannot grant x any further. Other variations of the copy flag, such as 1-step or n-step copy flags can be similarly defined [17].

The expressive power of Transform is illustrated by the following policy specification.

$$\begin{aligned} cc(\text{sci}) &= \text{doc} \\ cr(\text{sci, doc}) &= \{\text{own, read}\} \\[4pt] grant(\text{sci, security-officer, doc, own}) &= \text{review} \\ grant(\text{sci, patent-officer, doc, own}) &= \text{review} \\[4pt] grant(\text{security-officer, sci, doc, review}) &= \mathrm{a_s} \\ grant(\text{patent-officer, sci, doc, review}) &= \mathrm{a_p} \\[4pt] itrans(\text{sci, doc}, \{\text{own}, \mathrm{a_s}, \mathrm{a_p}\}) &= \text{release} \end{aligned}$$

The first two equations specify that (i) a scientist can create documents, and (ii) the scientist who creates a document obtains the own and read privileges for it.[§] The next two equations specify that a scientist who owns a document can ask for it to be reviewed by a security-officer and by a patent-officer. These officers can respectively return the a_s and a_p rights to the scientist signifying the respective approvals. The scientist can then release the document. This example is further elaborated in section 5.

This completes our description of the Transform model. Further motivation for Transform and additional examples of policies are given in [17].

3 DISTRIBUTED CAPABILITY SYSTEMS

Capability-based architectures have had a strong appeal ever since the concept was first proposed [7]. They are viewed as providing a sound and common basis for providing both reliability and security. In the context of conventional centralized systems a number of such machines have been built [13]. Some have even achieved moderate commercial success. Nevertheless today's popular CPUs are not capability based. In retrospect one can argue that using capabilities to solve the memory protection problem is an overkill. The marginal advantages of capabilities over memory segmentation and protection rings (which are available in the latest generation of microprocessors such as the Intel 80386) do not justify the extra costs and performance penalties. In other words the initial application of capabilities was at too low a level.

It is expected by many researchers [15, for instance] that in the 1990s distributed operating systems will dominate the computing environment. These systems will appear to users as a single

[§] Once a document has been created it can no longer be written. This is necessary in order to freeze the contents of the document. If revisions are required a new version of the document needs to be created.

centralized system with complete location transparency. To achieve this, reliability and security must be addressed as part of the basic design of these systems. Attempts to graft security features later in the design cycle will surely fail, much as they are failing in conventional centralized systems. The capability-based framework continues to offer an attractive approach to these problems. In a distributed operating system capabilities are introduced at a much higher level than memory addressing. Capabilities need to be incorporated into the remote procedure call mechanism rather than the memory addressing mechanism. This offers the hope that the additional overhead will not severely degrade performance. Capabilities can moreover be integrated into the basic client-server structure of distributed systems to provide transparency.

There are three basic issues which must be confronted by the designer of a distributed capability-based system. These issues are complicated relative to conventional centralized capability-based systems because capabilities are dispersed in individual workstations and can no longer be assumed to be under tight control of a security kernel.

1. *Unforgeability.* It must be guaranteed that capabilities cannot be modified or manufactured by subjects. This requires some form of cryptographic sealing.
2. *Propagation.* It must be guaranteed that capabilities cannot be copied from one user to another. This requires some means of embedding the identity of a subject in a capability.
3. *Revocation.* It must be guaranteed that capabilities which have been granted can be withdrawn or revoked in a timely manner. This requires some means of invalidating existing capabilities and accounting for cascaded revocation.

Various solutions to one or more of these problems have been proposed in the literature. For instance Amoeba [15] uses "sparse capabilities" with cryptographic protection to ensure unforgeability. Unfortunately Amoeba does not address capability propagation or revocation. Davies [6] discusses mechanisms to embed the identity of a subject in a capability. This ensures that capabilities cannot be forged or propagated from one subject to another without intervention of trusted software. Davies, however, does not address the revocation issue. Gong's proposed architecture [9] is the first attempt to address all three issues in a distributed context. It is based on the familiar client-server model of services in distributed systems and therefore is a suitable foundation for us to build upon. However, Gong does not incorporate the notion of types which is basic to Transform. His architecture therefore needs to be extended for this purpose.

4 IMPLEMENTATION OF TRANSFORM

We now describe a distributed capability-based implementation of the Transform model. We assume that objects are encapsulated within object servers. The basic computation model is that of remote procedure calls involving the following sequence of events: (i) a client sends a request to a server to manipulate one or more objects, (ii) the server accepts and services the request, and (iii) the server sends back a reply. The object server runs on a trusted host which guarantees that the server cannot be bypassed. For ease of exposition we visualize each object server as running on a separate host. However, we allow multiple object servers on the same trusted host provided the security kernel on the host can enforce separation among these servers. If we have sufficient confidence in the security kernel we can also allow untrusted clients to coexist with object servers on a single trusted host.

Each object server acts as the reference monitor (or access mediator) for the set of objects it manages. In other words the object server is part of the trusted computing base (TCB). The object server is responsible not only for access mediation but also for ensuring semantic correctness of the objects with respect to the abstract operations exported from the server. The object server itself has the ability to access all objects within its control. We emphasize that the object server is not a subject in the system but is rather a part of the TCB.

For simplicity, we require that each object server manage exactly one type of object. In practise this rule would probably be relaxed to allow a single server to manage multiple object types, particularly if they are closely related. On the other hand the same type of object may be managed by multiple object servers. For instance a given system may have numerous file servers. An individual file server manages some subset of the total collection of files in the system. We assume there is no replication of files, i.e., each file resides at exactly one file server.

Finally we assume there is an access decision facility (ADF) which can be consulted by object servers to determine the security policy. In the context of Transform the ADF will be consulted by object servers for finding out appropriate values of cc, cr, *grant* and *itrans*. Pieces of the ADF may actually reside at each object server while other pieces are remotely accessed. The reason for this is to allow quick local access to well-established and relatively static aspects of the policy while at the same time allowing for new types etc. to be introduced.

4.1 Identity and Type

Each subject or object in the system has a globally unique identifier. Each subject or object also has a unique type which is determined when that subject or object is created. Thereafter the type cannot change. We assume the type of a subject or object is embedded in its identifier. Henceforth we refer to a subject identifier by *sid* and a object identifier by *oid*. These identifiers have the following structure.

type	identifier

The type field denotes the type of the object while the identifier field uniquely identifies each subject or object among instances of the same type. Note that sid's and oid's can be generated at will by users.

4.2 Capability Seeds

A capability seed is a secret random number associated with each oid. The seed is known only to the object server which manages the object identified by oid. We can visualize this association by the following pair.¶

oid	seed

The purpose of the seed is to facilitate revocation and prevent against replay of revoked capabilities, as will be discussed later.

4.3 Capabilities

A capability has the following structure.

oid	rights	seal

where the seal is computed using a publicly known one-way function f as follows.

¶Gong [9] calls this pair an "internal capability." We feel the name "internal capability" is a misnomer and prefer to call the secret random number a capability seed because its principal use is in cryptographically sealing capabilities exported from the object server.

$$\text{seal} = f(\text{sid, oid, rights, seed})$$

The oid and rights components of a capability are exactly as one would expect even in a conventional centralized system. The seal cryptographically embeds the subject identifier (sid) in the capability using the capability seed for that purpose.

4.4 Access Mediation

Access mediation must be incorporated into the RPC (Remote Procedure Call) mechanism of the client-server architecture. The object server must authenticate the source of every RPC request. For this purpose, we assume that each subject has the means to place its digital signature on every RPC communication to a object server. The RPC also carries within it the relevant capabilities for the operation being requested. The object server first verifies that the sid on each capability is authenticated by the digital signature, otherwise the RPC is immediately rejected. Then the object server looks up the capability seed for oid, computes the seal using the above formula and compares the computed seal with the seal submitted by the subject. If these match the capability is known to be authentic and the operation is performed provided the rights are sufficient to authorize it.

Digital signatures for the reverse communication from object servers to subjects can also be incorporated. The details of these protocols are beyond the scope of this paper and can readily be found in the standard literature [1, for instance]. We envisage a implementation similar to the interface function box of Amoeba [15] which are placed between each processor module and the network.

4.5 Creation

For object creation the object server consults the access decision facility (ADF) to determine whether or not such creation is authorized by *cc*(sid.type). If the creation is authorized a new object is created with a new oid and a new capability seed. The rights to be entered on the capability are determined from *cr*(sid.type,oid.type). Finally the capability is sealed and returned to the subject.

4.6 Internal Transformation

Let subject sid request the following internal transformation for object oid.

$$\mathit{itrans}(\text{u, o}, \{x_1,\ldots,x_n\}) = \{y_1,\ldots,y_m\}$$

The object server must, of course, be a manager for objects of type o. The server checks that sid.type=u and oid.type=o. It also checks that the RPC request includes a capability (or capability list) for object oid with the rights $x_1, \ldots, x_n$. This check is performed by comparing the computed seal with the seal on the capability as discussed in section 4.4. Finally the object server creates a new capability sealed for sid with rights $x_1, \ldots, x_n$ $y_1, \ldots, y_m$. This capability is returned to the subject sid. Note that the original capability, with rights $x_1, \ldots, x_n$ continues to be valid. It is however redundant and can be discarded by the subject.

4.7 Grant Transformation

Let subject sid1 request the following grant transformation for object oid to subject sid2.

$$\mathit{grant}(\text{u, v, o}, \{x_1,\ldots,x_n\}) = \{y_1,\ldots,y_m\}$$

The object server should again be a manager for objects of type o. The server checks that sid1.type=u, sid2.type=v and oid.type=o. It also checks that the RPC request includes a capability (or capability list) for object oid with the rights $x_1, \ldots, x_n$. If the check is successful the object server creates a new capability sealed for sid2 with rights $y_1, \ldots, y_m$. This capability is returned to the subject sid1 who can then pass it on to subject sid2.

4.8 Revocation

Revocation has always been a problem in capability-based systems. In distributed systems the problem is further compounded, since the subjects are completely autonomous with no centralized authorities enforcing security. There are various issues against which the implementation of revocation can be compared [19].

1. Partial or Complete: Whether it is possible to revoke a specific right or whether all rights in a capability have to be revoked to get any sort of denial of access in the system?
2. Immediate or Delayed: If the implementation executes revocation immediately or it comes into force only the next time the subject tries to access the object?
3. Selective or General: Does the revocation process affect all users or a select group of users having access over the object?
4. Temporary or Permanent: Is access is to be denied permanently or if once it is revoked, is it retrievable?

We provide revocation by a revocation list and a count field appended to the seed as shown below.

oid	seed	count	revocation list

The revocation list contains entries of sids for whom the rights for that particular oid have been revoked. The list specifies for each sid which of its rights have been revoked. When the validity of the capability is checked during access mediation, the revocation lists are checked in parallel as well. Since access mediation is performed on every operation revocation is immediate. The owner of an oid always has the option to revoke partially or completely the capability of a sid for that oid. Partial or complete revocation of a sid in no way interferes with the access rights of other sids.

The count is a measure that determines the number of valid capabilities for that seed. The count is incremented during creation and propagation, but decremented during complete revocation (i.e. when all the rights of a subject for that object are revoked). Temporary or permanent revocation is carried out, depending on the value of the count. If the size of the revocation list becomes a significant fraction of the count the object server goes ahead with permanent revocation. The server deletes the seed associated with that oid, computes a new one and sends new recomputed capabilities to other associated sids. This of course requires that the object server keep a log of propagation of capabilities. However if the size of the revocation list is small in comparison to the count, the object server goes ahead with temporary revocation. In this case the object server appends the revocation information onto the revocation list associated with that oid.

5 EXAMPLE

The scientist and the security-officer example discussed earlier in section 2 is illustrated here using the protocols described above. A scientist (say Joe) creates a document (say SDI) on his workstation, but before he can release it he needs to have approval from a security-officer (say Sam) and a patent-officer (say Pat). The following is the sequence of protocols needed to complete the task.

1. Joe asks the server to create a document called SDI. This RPC is made by the kernel of Joe's workstation to the appropriate daemon responsible for the server's actions. The RPC contains the action requested, the sid, oid, the types of sid and oid involved, and the actual data to be stored in the created document; all signed under Joe's digital signature. In this case the sid=sci.Joe and the oid=doc.SDI. Joe and SDI are respectively of type sci and doc. On receiving the request, server checks the digital signature to authenticate Joe. The server then checks the *cc* policy, taking into account the sid, oid and their types provided. If it is in the affirmative it checks the *cr* policy, by which it determines what rights Joe gets for the document he is creating. The server then pulls out the seed say seed1 for that document and stores it in its internal tables with the following association:

doc.SDI	seed1

   Then the object server manufactures the following capability and sends it to Joe (strictly speaking to the kernel of Joe's workstation):

doc.SDI	own, read	seal1

   where seal1 = f(sci.Joe, doc.SDI, {own, read, write}, seed1)

2. Now Joe is ready to release the document. His workstation sends the propagation requests to the server on his behalf. The RPC looks like this:

   grant(Sam, review)

doc.SDI	own, read	seal1

   The host when framing the RPC, appends to it the capability it possesses for SDI and signs the request under Joe's digital signature. The server on receiving the request verifies the digital signature and authenticates Joe. Then the server checks the validity of the capability by retrieving the seed of SDI, i.e. seed1, from its internal tables, and computing the seal using the one way function f. Then it computes seal1 from the capability provided by Joe and if the two seals match the validity of the capability is confirmed. The request is then checked against the *grant* policy of Transform. When the server determines Joe has sufficient rights, i.e. own, for SDI, it authorizes the grant. The server then computes the capability for the security-officer Sam to have the review right for SDI. The capability

doc.SDI	review	seal2

   where seal2 = f(security-officer.Sam, doc.SDI, review, seed1)

   is sent to Joe. Joe then forwards this capability to Sam. Sam now has the capability for oid=doc.SDI with the review right. With this capability he can only access the document to review it. If he tries to get additional rights by internal transformation, the server will turn down his request because when it will check the set of rights he possesses, namely review, which is insufficient set for it to grant him additional rights. Sam now reviews the document, and if he approves of the action to release SDI he requests the server to grant Joe the approval (a_s) right.

   grant(sci.Joe, a_s)

doc.SDI	review	seal2

The server computes the following capability and sends it back to Sam who in turn sends it to Joe.

doc.SDI	a_s	seal3

where seal3 = f(sci.Joe, doc.SDI, a_s, seed1)

3. Exact similar protocol steps are executed to get the approval (a_p) from the patent-officer Pat. At the end of this session Joe possesses the following capability.

doc.SDI	a_p	seal4

where seal4 = f(sci.Joe, doc.SDI, a_p, seed1)

4. Now the scientist Joe possesses the capabilities giving him the approval to get the release right by internal transformation. Joe presents these capabilities to the server with the following request:

	doc.SDI	own, read	seal1
itrans(release)	doc.SDI	a_s	seal3
	doc.SDI	a_p	seal4

Like before, the server carries out the authentication and the validity tests on the capabilities presented to it by Joe. Then the server checks that Joe has the rights own, a_s and a_p for SDI which are required to get the additional release right. The server sends him a new capability:

doc.SDI	own, read, a_s, a_p, release	seal5

where seal5 = f(sci.Joe, doc.SDI, {own, read, a_s, a_p, release}, seed1)

This completes the example.

6 CONCLUSION

In this paper we have proposed a distributed capability-based implementation for the Transform model. The system is based on object servers who act as access-mediators on any attempt by a subject to create, use, acquire, grant or revoke capabilities. We assume a digital signature facility which authenticates the originating subject on each remote procedure call. The capabilities are cryptographically sealed to tie together the identity of the subject, the identity of the object, the rights and a secret cryptographic seed. Strong typing of subjects and objects has also been incorporated.

Our long term goal is to arrive at a practical distributed implementation for SPM (and its recent extension called ESPM [2]). Our first step towards this goal is the implementation of Transform described here. Transform is a sufficiently interesting and non-trivial special case of SPM. At the same time Transform is a sufficiently simplified version of SPM for which a realistic near-term implementation can be contemplated.

Acknowledgment

We are indebted to Howard Stainer and Sylvan Pinsky for their support and encouragement, making this work possible. The opinions expressed in this paper are of course our own and should not be taken to represent the views of these individuals.

References

[1] Akl, S.G. "Digital Signatures: A Tutorial Survey." *Computer* 16(2):15-24 (1983).

[2] Ammann, P. and Sandhu, R.S. "Extending the Creation Operation in the Schematic Protection Model." *Proc. Sixth Annual Computer Security Applications Conference*, 340-348 (1990).

[3] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, Mitre, Bedford, Massachusetts (1975).

[4] Clark, D.D. and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies." *IEEE Symposium on Security and Privacy* 184-194 (1987).

[5] Cohen, E. and Jefferson, D. "Protection in the Hydra Operating System." *5th ACM Symposium on Operating Systems Principles*, 141-160 (1975).

[6] Davies, D.W. "Protection." In Lampson, B.W., Paul, M. and Siegert, H.J. (Editors). *Distributed Systems: An Advanced Course.* Springer-Verlag, 211-245 (1981).

[7] Dennis, J.B. and Van Horn, F.C. "Programming Semantics for Multiprogrammed Computations." *Communications of ACM* 9(3):143-155 (1966).

[8] *Department of Defense Trusted Computer Systems Evaluation Criteria.* DoD 5200.28-STD, Department of Defense National Computer Security Center (1985).

[9] Gong, L. "A Secure Identity-Based Capability System." *IEEE Symposium on Security and Privacy*, 56-63 (1989).

[10] Harrison, M.H., Russo, W.L. and Ullman, J.D. "Protection in Operating Systems." *Communications of ACM* 19(8):461-471 (1976).

[11] Harrison, M.H. and Russo, W.L. "Monotonic Protection Systems." In DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J. (Editors). *Foundations of Secure Computations.* Academic Press, 337-365 (1978).

[12] Lampson, B.W. "Protection." *5th Princeton Symposium on Information Science and Systems*, 437-443 (1971). Reprinted in *ACM Operating Systems Review* 8(1):18-24 (1974).

[13] Levy, H.M. *Capability-Based Computer Systems.* Digital Press (1984).

[14] Minsky, N. "Synergistic Authorization in Database Systems." *7th International Conference on Very Large Data Bases*, 543-552 (1981).

[15] Mullender, S.J., van Rossum, G., Tanenbaum, A.S., van Renesse, R. and van Staveren, H. "Amoeba: A Distributed Operating System for the 1990s." *IEEE Computer*, 23(5):44-53 (1990).

[16] Sandhu, R.S. "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes." *Journal of ACM* 35(2):404-432 (1988).

[17] Sandhu, R.S "Transformation of Access Rights" *IEEE Symposium on Security and Privacy*, 259-268 (1989).

[18] Sandhu, R.S. "Undecidability of Safety for the Schematic Protection Model with Cyclic Creates." *Journal of Computer and System Sciences*, to appear.

[19] Siberschatz, A., Peterson, J., and Galvin, P. *Operating System Concepts.* Addison Wesley(1991).