An Architecture for Practical Delegation in a Distributed System

Morrie Gasser, Ellen McDermott*

Digital Equipment Corporation
BXB1-2/D04
85 Swanson Road
Boxborough, Massachusetts 01719-1326

Abstract

Delegation is the process whereby a user in a distributed environment authorizes a system to access remote resources on his behalf. In today's distributed systems where all the resources required to carry out an operation are rarely local to the system to which the user is logged in, delegation is more often the rule than the exception. Yet, even with the use of state-of-the-art authentication techniques, delegation is typically implicit and transparent to the remote system controlling the resources, making it difficult for that system to determine whether delegation was authorized by the user. This paper describes a practical technique for delegation that provides both cryptographic assurance that a delegation was authorized, and authentication of the delegated systems, thereby allowing reliable access control as well as precise auditing of the systems involved in every access. It goes further than other approaches for delegation in that it also provides termination of a delegation on demand (as when the user logs out) with the assurance that the delegated systems, if subsequently compromised, cannot continue to act on the user's behalf. Delegation and revocation are provided by a simple mechanism that does not rely on online trusted servers.

1 Background

On any standalone computer system with resources to protect, users authenticate themselves through a trusted path [DOD85] to some form of reference monitor [Ames83] in control of the objects in the system. The reference monitor typically associates an active entity, such as a process, with the authenticated user. Access requests made by the process are enforced by the reference monitor as if the associated user had made them directly. The access control scheme may be identity-based, using mechanisms such as access control lists (ACLs), rule-based, according to a mandatory security policy, or a combination of both.

Despite the possible presence of Trojan horses in the user's process, the reference monitor has no choice but to believe that the access request made by a process reflects the user's wishes. The fact that the user has authorized the process to act on his behalf (by logging in and authenticating himself) is sufficient grounds on which to base this belief. Only in exceptional cases, where the user makes a request directly to the reference monitor through a trusted path, can the reference monitor know the user's intent with certainty.

The user's authorization of his process to act on his behalf is a form of *delegation* of rights from the user to the process. In some cases the user may delegate the rights of one of several permissible roles or identities (e.g., by logging in using different names and/or passwords), in order to limit the actions of the process to some subset that the user is authorized. Limited delegation also occurs routinely in multilevel secure systems where the user selects a single classification for his process that is a subset of the access classes for which the user is authorized.

In a distributed system the reference monitor in a computer system also enforces access requests from its local processes, but these processes are not necessarily associated with local users that the reference monitor has directly authenticated. Most likely the local process has been activated by another process on some remote system to which the user is logged in, where the login system's reference monitor has authenticated the user. How should the local reference monitor determine the identity of the user to associate with its local process?

The most straightforward approach is for the local reference monitor to ask the remote login system to supply the user's identity. In this case the reference monitor trusts the login system to properly authenticate the user. But this level of trust is too great to bear in the world of mutually suspicious systems.

To limit the need to trust the login system the local reference monitor can ask the login system to supply a password for the user. The login system, in turn, obtains the password from the user and forwards it to the local system. Receipt of the password is taken as evidence that the user is currently at the login system. Asking for a password requires a bit less trust than simply believing the login system's claim, but is still not very secure because the login system, if malicious, has a chance to capture and replay the password at a later

*This paper presents the opinion of its authors, which are not necessarily those of the Digital Equipment Corporation. Opinions expressed in this paper must not be construed to imply any product commitment on the part of the Digital Equipment Corporation.

CH2884-5/90/0000/0020$01.00 © 1990 IEEE

time. Any system to which the user has ever logged in (since he last changed his password) can masquerade as the user to the local reference monitor.

A more secure technique for authenticating the remote user employs a cryptographic algorithm and a hardware device operated by the user, allowing the reference monitor to determine that the user is in fact currently logged into (or communicating through) the login system in a manner that cannot be spoofed by the login system. A number of "see-through" authentication devices are available that will accomplish this [NCSC87a, Racal]. (A see-through device is one that permits the authentication dialog between the user and the remote reference monitor to take place without special hardware physically attached to a terminal.) More convenient for the user, but requiring special hardware, is some type of smart card that carries out the cryptographic challenge/response protocol for the user.

Once the user properly authenticates himself through his login system to the reference monitor, the reference monitor associates the user's identity with a local process and sets up some form of "secure channel" between the local process and the login system.[1] As in the case of the standalone system, the reference monitor explicitly trusts its local process to act on behalf of the user, but in addition, by setting up the secure channel, the reference monitor implicitly trusts the login system to be faithful to the user's wishes. The fact that the user has authenticated himself through that login system is sufficient grounds on which to base this additional trust. By authenticating himself through the login system the user has delegated to that system, and by establishing a connection to the reference monitor's local process the login system in turn has delegated to the process.

By directly authenticating the remote user in an unspoofable manner the reference monitor can reliably enforce access controls without trusting any components of the distributed system to which the user did not delegate. But interactive authentication of the remote user by the reference monitor is not always practical. Except for examples of "terminal emulation" as in the common use of TCP/IP's telnet or Unix rlogin, it is not always convenient, or possible, for the reference monitor to freely interact with the user to carry out the authentication dialog. Consider a file server behind a database server that is accessed by a query processor running on the user's login system (figure 1). It is not practical to

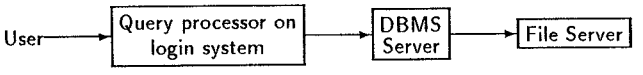


Figure 1: Multiple levels of indirection in an access. The reference monitor in the file server cannot directly authenticate the user on behalf of whom the request was made.

expect the file server to directly authenticate the user at the time it receives the access request from the DBMS server on behalf of the user, as the user may be operating in an entirely different context and may not even be logged in at the time the query is processed. Similar problems occur with transaction processing applications. Any delegation mechanism suitable for general-purpose distributed computing must not require real-time interaction between the remote user and the reference monitor.

[1] Ideally this secure channel should be an integrity-protected communications path using some form of encryption.

2 Overview of the Architecture

The delegation architecture described here permits the reference monitor to know whether the remote user has delegated to a login system without communicating directly with the user or with any online trusted third party. The delegation feature is very similar to that implemented by the Kerberos V4 [Miller87] authentication system, except that Kerberos requires an online authentication server to initiate the delegation. Our use of public key cryptography eliminates Kerberos's need for an online trusted server at delegation time.

This architecture goes further than Kerberos V4 in that it permits a "chain" or cascade of delegations through multiple systems (such a feature, also called "authentication forwarding" or "proxy", is proposed for Kerberos V5 [Kohl89] and was also discussed by Karger [Karger86] and Girling [Girling83]). In our chain of delegations the reference monitor knows that all systems in the chain are authorized delegates. In addition, the reference monitor can determine the identity of those systems (as well as that of the user) for auditing and access control purposes.

One additional feature, not known by us to be present in other authentication forwarding schemes, permits the delegation to be explicitly terminated by the user so that the intermediate systems, if compromised subsequent to the termination, cannot make use of the rights attained through delegation. This instant revocation of delegation on demand provides more security than preset timeouts used in other systems. Other systems that implement instant revocation do so with the help of online authentication servers [Girling82] and indirection [Redell74].

In our scheme delegations are specified by public key certificates that are dynamically created at the user's login system. Because they are unforgeable tokens serving as proof of authorized access, delegation certificates are very much like traditional capabilities, except that possession of the certificate is not sufficient to permit access. Access is permitted by only those who possess the certificate and who can prove they possess a cryptographic key named in the certificate. Our use of digital signatures protects the integrity of the certificate, and the key in the certificate prevents use by other than the authorized holder.

Redell [Redell74] first proposed the concept of capability sealing to prevent forgery using a tagging mechanism, and a number of tagged architectures have been proposed and implemented. Mullender suggested, in the Amoeba system [Mullender85], that encryption and digital signatures be used if forgery and misuse of capabilities by unauthorized users is a concern. Girling's capabilities contain one-time keywords

to prevent forging, but the capabilities must be kept secret because keywords in capabilities communicated over the wire can be captured and re-used. In a distributed system where delegation credentials must be transferred to potentially untrusted locations (a topic on which we elaborate later), secrecy of a certificate cannot be assumed. Our delegation credentials are similar to the privilege attribute certificates (PACs) in ECMA's "Security in Open Systems" framework [ECMA89], except that ECMA's certificates are created by a trusted online authentication service rather than by the user's login system.

3 Context and Definitions

This delegation architecture is specified within the framework of Digital's Distributed System Security Architecture (DSSA) [Gasser89], an outgrowth of work originated by Birrell, et al. [Birrell86]. While DSSA incorporates the notion of a clearly defined, protected "reference monitor" normally used to describe security-kernels, DSSA does not concern itself with degrees of assurance and is equally applicable to systems built using standard commercial practices.

Under DSSA users log into local systems using smart cards that carry out cryptographic authentication based on RSA public key cryptography [Rivest78]. Online directory services implementing a hierarchical name space based on Digital's naming service (DNS) [Lampson86], similar to X.500 [CCITT88a], are available throughout the distributed system, and concepts similar to those discussed in X.509 [CCITT88b] are used for certification and authentication. For the most part, directory services are not trusted for security. The user authentication mechanism for DSSA is covered in detail by Linn [Linn90].

Any entity that can authenticate itself or can make an access request on its own or another's behalf is called a "principal." We generally do not have to distinguish between different types of principals, although for clarity we often use more descriptive terms such as user, workstation, process, system, or server. Principals are identified by a global name. All principals possess a relatively permanent cryptographic secret consisting of the private component of an RSA key, and have the ability to authenticate themselves to other systems or to digitally sign certificates using their private keys. The names and public keys of principals are usually bound together in a certificate signed by a certifying authority (CA) and stored in the directory service.

The distributed system consists of a collection of autonomous, mutually suspicious computer systems, each with its own reference monitor. Except for compliance with a common set of protocols and naming conventions for interoperability, the reference monitors are not centrally managed. Each reference monitor implements discretionary and mandatory access controls and enforces its own security policy for access to the objects under its control. DSSA specifies the use of ACLs for each object, where the ACL lists the global names of principals and their access rights. The names on an ACL may identify users, systems, groups of users and systems, or groups of groups, and the ACL can specify that access by a user is permitted only from a given set of delegated systems.

4 Simplified View of Delegation

Our simplified model of computing involves a user logged into a workstation that issues requests to access remote resources on behalf of that user. In general, however, the workstation can be any multi-user computer system, and the user need not be physically present or "logged in" at the time the access request is made.

Local uses of a workstation require no explicit delegation. If the user logs into a standalone workstation, access to local files within the workstation is controlled by the workstation's reference monitor (figure 2).

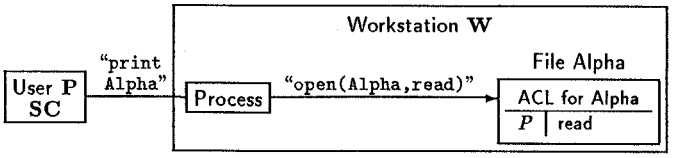


Figure 2: Access to local resources. No delegation is required when the user, authenticated to the workstation **W** by his smart card, accesses files on that directly attached system.

Because the reference monitor has authenticated the user, and because the reference monitor maintains a secure physical channel between the user and the process, the process cannot forge the identity of the user on behalf of whom it is operating. This form of implicit delegation for access to local resources is a common occurrence in today's systems and does not require special delegation mechanisms.

4.1 Single-level Delegation

The simplest example of distributed delegation is from a user to his workstation, so that the workstation can access the user's files resident on a remote server. In figure 3, if the

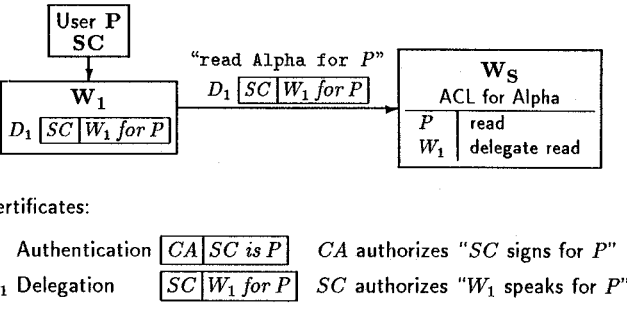


Figure 3: Delegation to workstation. SC delegates to W_1 for access to files on W_S. The delegation certificate D_1 is passed to the W_S along with the access request.

user (P), through his local workstation (W_1), needs to access

remote files on a server (W_S), the user must delegate to the workstation the ability to access those files on his behalf. This delegation delegation must result in a set of credentials that permit W_1 to prove it was an authorized delegate.

The delegation from the user to the workstation is represented by a delegation certificate, signed by the user's smart card at login time (certificate D_1 in figure 3), authorizing the specific workstation to speak for the specific user. The delegation certificate format we show, $\boxed{SC}\boxed{W_1\ for\ P}$ means that SC signed a certificate authorizing W_1 to speak for P. In other words, SC has authorized W_1 to access anything that P can. The workstation passes this certificate, as proof of delegation, to any server to which the workstation submits a request on behalf of P.

For this scheme to be robust there must be a secure channel (encrypted or physically protected link) between the W_S and W_1. The secure channel provides (1) authentication to verify the identity of W_1, so that the server knows it is receiving the request from the system named in the delegation certificate, and (2) integrity so that the server can be sure the request is the same as the one W_1 sent. Mechanisms to provide secure communications channels are well known and are outside the scope of this delegation architecture. While we require secure communications for a completely foolproof system, significant improvement in security is attained by the use of this architecture even without secure communications. Integrity of the delegation mechanism discussed here does not depend on integrity of this secure channel.

Shown for completeness in figure 3 is an authentication certificate authorizing the smart card to represent the user: $\boxed{CA}\boxed{SC\ is\ P}$. This certificate is signed in advance by some certifying authority that associates the smart card with the user. The details of the certification and authentication process, covered elsewhere [Linn90], are not important to this discussion, but it is worth noting that the authentication certificate A is static information probably stored in the directory service, as compared to the delegation certificates that are dynamically created. Even with secure channels between all the systems, the authentication certificate is needed by the server to verify that the user named in the access request (P) is the owner of the smart card (SC) that signed the delegation certificate. Since authentication certificates are available to systems from the directory service, we don't show A being passed from W_1 to W_S.

The server W_S enforces an ACL that contains the name of the user and also may list the name of the workstation(s) to which the user may delegate. Both user and workstation(s) must have access permission in order for the request to be granted. Before granting access, the server scans the ACL, looking for a user name and system name that matches those identified in the delegation certificate. Of course, the reference monitor must insure that the user named in the request, and the authenticated identity of the system making the request, are also be the same as those named in the delegation certificate.

4.2 Chained Delegations

In many situations there is more than one system between the user and the server. In figure 4 the user on workstation W_1 accesses a file on the server W_S, but that request is in fact made by W_2, acting on behalf of the user. In this case, the user delegates to W_1 which in turn delegates to W_2 to act on behalf of the user.

The first delegation from the user to the workstation happens as before, where W_1 is authorized to speak for the user (certificate D_1 in the figure). The second delegation $\boxed{W_1}\boxed{W_2\ for\ P}$ says that W_1 permits W_2 to speak for the user. Before making any access requests to the server, W_2 forwards to the server a copy of both its own and W_1's delegation certificate (D_1 and D_2). Through a chain of reasoning using D_1, D_2 and A the server can conclude that W_2 is authorized to speak for the user. Both W_1 and W_2, as well as P, must be named on the ACL.

Note that the server has no proof, and does not care, whether W_1 or the user actually made any access request. The server only knows that the delegations have been authorized and, as in the single workstation case, must trust W_2 to act in good faith on behalf of both W_1 and the user.

5 Details of the Delegation Process

The simplified overview of delegation above, where delegation certificates mention the names of the delegated systems, has some security weaknesses and is also insufficiently precise to be implemented directly. This section provides a more thorough description of the delegation process with additional motivation for the mechanisms.

5.1 Creating and Using Delegations

Once a delegation has been received, the delegated principal may make requests to access an object on behalf of the delegating principal, and may further delegate by authorizing a third principal to act on behalf of the original delegator. As we indicated earlier this delegation can be "chained" to an arbitrary number of other principals. The requests to access an object on a target system made by any of the delegated principals, when explicitly indicated as requests that are being made on behalf of the first principal, are mediated by the reference monitor of the target system as if the requests had been made directly by the first principal. An additional constraint is that all the delegated principals in the chain must be listed on the ACL of the object as permitted delegates.

Mandatory access controls, if employed by the target system, may further limit access to the object. While the details of the mandatory security policy and its mechanisms are not relevant to this discussion of delegation, it is worth noting that a reference monitor that enforces a label-based policy need only know the security label or "access class range" associated with the system from which it directly receives the access request, usually the last system in the delegation chain, and not the label of every system involved in the delegation.

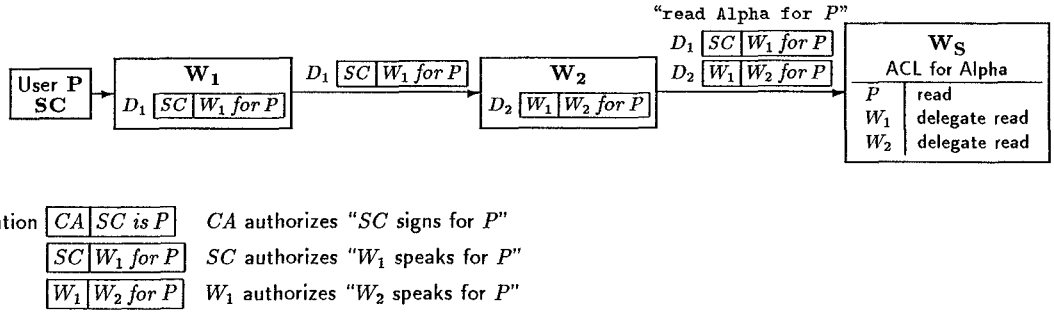


Figure 4: Delegation two levels deep. SC delegates to W_1 which delegates to W_2.

This is because no system can be permitted to communicate outside its access class range, and within that range the system must be trusted to specify the proper access class of every request. Delegation at a specific access class does not provide any security benefit over simple assertion of the access class by the last system in the chain.

Delegation is authorized solely by signed delegation certificates. In a chain of delegations each principal retains a delegation certificate for itself signed by the previous principal in the chain, and signs a delegation to the next principal if it chooses.

Any principal in the chain wishing to access an object on behalf of the first principal must obtain a copy of all the delegation certificates for the previous principals in the chain and must forward those copies to the reference monitor along with the access request. The reference monitor must validate the chain of certificates, being sure the certificates are linked by authorized delegates, in addition to enforcing the ACL of the object. (Once the reference monitor has verified the validity of a delegation chain the information can be cached and the certificates need not be forwarded on subsequent requests.)

5.2 Delegation Timeout

Typically each system in the chain will keep copies of all the delegation certificates for the systems prior to it so it can forward those copies to the next principal or to the reference monitor as required. When the delegations are no longer needed, as when the original user logs out, the systems in the chain should not continue to represent the user. As we stressed earlier, good systems are expected to act in the best interests of the user, and the user must assume that no system to which he delegates will delegate to another system that the user doesn't trust. But, in case one of the systems in a chain is in fact corrupt and doesn't act according to the rules, each certificate contains a timeout that limits the period during which the corrupt system could continue to operate on the user's behalf. If any of the certificates in the chain have timed out, the reference monitor will grant no access.

Timeouts are expected to be reasonably long, on the order of a day or more, so that a typical interactive session will not be interrupted by a timeout. For very long sessions where expiration is imminent a delegation renewal is initiated by the user's local workstation and propagated to all the systems that have received soon-to-expire delegation certificates. This renewal requires that the workstation reauthenticate the user, requiring that the user re-enter his smart card or password, because the first certificate in the chain must be signed by the user's private key which the workstation does not possess. In case the user is no longer physically present at the original workstation, as is the case for batch or background jobs when the user may have logged out long ago, restricted delegations are used. These topics are discussed in section 7.3.

Timeouts are more difficult to implement if the certificate cannot be cryptographically integrity-protected. Girling's capabilities are timed out with the help of an online authentication server that is queried at appropriate intervals, and which must signal a timeout to all systems that have made the query.

5.3 Delegation Keys

Whereas a system must be assumed to behave properly while a delegation to it is in effect, it would be nice if a subsequent compromise of a previously trusted system did not place previous users of the trusted system at risk. In other words, a system, while it is still operating on behalf of the user, should be able to prevent its own future compromise from affecting that user. Eventually delegations time out, so a compromise of a system can only affect users who recently delegated to the system, but for the 24 hours or so that the delegation is still in effect a user might rightly be concerned about a possible compromise of his system after he has logged out. This scenario is a serious threat in an environment of public workstations where anyone can use the machine after the user departs.

Immediate revocation of a delegation could be implemented by deleting copies of all delegation certificates in a chain from the user to all servers. In schemes such as Girling's, using one-time passwords for proof of delegation, each system need only delete its password. But delegation certificates are "public" and their whereabouts cannot be constrained. Even though the legitimate systems in the delegation chain could be trusted to delete their own copies of the delegation certificates, there are additional copies of the delegation certificates

not under control of any of the trusted systems in the chain.

In particular, the server for an object need not be one of the delegates, yet it must receive a copy of the delegation certificate for each of the systems in the chain. In order to access files on a server, a good system in the chain will have a legitimate need to forward all delegation certificates to the server whether or not the good system trusts the server enough to delegate to it. (In a distributed system where the user has access to data on many different servers, there are cases where a user authenticates himself in order to access data on one server does but not trust that server for access to data on a different server.) A malicious server desiring to make use of the user's rights to access information resident on another server could retain copies of the received certificates even after all copies have been deleted by the delegated systems. These copies are of no direct use to the malicious server, because the server is not named in any of the delegation certificates and would therefore not be able to represent the user when making a request to another system. But if, at some time after logout but before expiration, one of the previously good systems in the chain is compromised by an accomplice of the malicious server, the compromised system can retrieve the certificates from the server and can then act on behalf of the original user for access to any file on any other good server, until the delegation certificates expire.

The above scenario may seem a bit far-fetched, but it points out that a delegation certificate may have to be disclosed to a principal not trusted as much as the named delegate. In any public key scheme it is generally unsafe to assume that a digitally signed certificate is confidential. As a minimum, certificates in transit can be captured by wiretappers.

To avoid this type of reuse of a delegation after the "session" ends but before the timeout, the delegation is based on a "delegation key." A delegation key is a public/private key pair generated by the delegated system for each session. The public component of the key is named in the delegation certificate signed by the delegating system, and the private component is held by the delegated system, which it uses to authenticate itself to the server and to sign the next delegation certificate (if any). Figure 5 shows a more precise version of the two-level delegation in figure 4 illustrating the keys used in the certificates. In this and future diagrams we show certificates more precisely by replacing the name of the signer of the certificate with the key used to sign it.

Certificate A shows that the public key of the smart card, K_0, is certified by the CA's private key, K_{CA}. K_1 is the public delegation key generated for the current session, the private portion of which is held by W_1, and certificate D_1, signed by the smart card's private key, specifies that the private component of K_1 is permitted to speak for the user. Similarly, K_2 is authorized to speak for the user through the certificate, D_2, signed by the private component of K_1. The delegation certificate format here is expanded to include the timeout plus a new field for the delegation key in addition to the names of the principal and workstations:[2]

[2]We don't explicitly bother to indicate in these figures the private vs. public components of the keys; this should be obvious from context.

$$\boxed{K_0}\boxed{K_1 \text{ as } W_1 \text{ for } P \text{ until } T}$$

The certificate continues to include the workstation name W_1, because that name is needed for lookup on the ACL, but the delegation is to the key K_1.

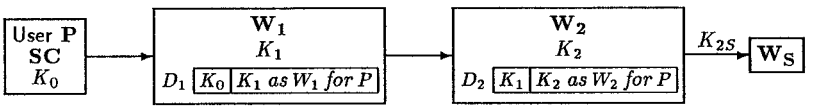
Using certificate D_2 in figure 5 the server (W_S) will only accept a request from W_2 on behalf of the user if W_2 can prove, at the time it makes the request, that it possesses the private component of K_2. To provide this proof W_2 could digitally sign its request with the private component of K_2, but more likely it will send the request on a secure communications channel K_{2S}, between W_2 and W_S. W_2 associates this secure channel with K_2 through a challenge/response protocol with W_2, where W_2 proves it possesses the private component K_2 named in certificate D_2.

Now, when the delegations are no longer needed, the delegated systems W_1 and W_2 erase the private components of their delegation keys K_1 and K_2, rendering useless any copies of the delegation certificates held by W_S. If W_2 is subsequently compromised, W_2 can attempt to obtain from W_S and reuse the delegation certificates, but since the private component of K_2 was not saved, W_2 can no longer successfully respond to any challenges from other servers to verify its possession of K_2's private component as required by certificate D_2.

If W_1, rather than W_2, is subsequently compromised, there are two things W_1 can attempt to do. W_1 can attempt to reuse the old delegation certificate D_1 as proof that it is still delegated or it can attempt to sign a new delegation certificate D_2' (for another system W_2', perhaps) using the private component of the old delegation key K_1. But the old delegation certificate D_1 is useless because neither W_2 nor any other system possesses the private component of the delegation key K_2 mentioned in that certificate. A new delegation to W_2' will require a new certificate with a new delegation key pair K_2', which W_1 will not be able to sign with K_1's private component. Hence, W_1 has no choice but to generate a new delegation key pair for itself K_1', but K_1' is useless unless signed by the smart card delegating to this key. The smart card will only sign such a certificate if the user is willing to log into, and delegate to, the compromised system.

In practice it is unlikely that a server will challenge a workstation for possession of the delegation key on every access attempt, since that would seriously impact performance. Instead, W_S will assume that any message on the secure channel K_{2S} is on behalf of P (if so claimed by W_2) once proof of delegation and possession of K_2 has already been established. Therefore, in order to completely terminate the delegation, W_2, in addition to deleting its private component of K_2, must inform W_S to tear down the secure channel. If W_2 then tries to reopen a channel to W_S, a new certificate and proof of possession will be required—something W_2 cannot do if it has erased its key.

It should be understood that signatures are always done using private components, and only public components are transmitted between principals as "data" in a certificate. Private components are never disclosed to other principals.



Certificates:

A Authentication	K_{CA} K_0 is P	K_{CA} signs "K_0 signs for P"
D_1 Delegation	K_0 K_1 as W_1 for P	K_0 signs "K_1 as W_1 speaks for P"
D_2 Delegation	K_1 K_2 as W_2 for P	K_1 signs "K_2 as W_2 speaks for P"

Figure 5: Delegation keys. Delegation is expressed in terms of the ability of a system W_i to speak on the user's behalf if it possesses the delegation key K_i. W_i uses K_i to authenticate itself and to sign further delegation certificates.

Successful revocation depends on the reliability of communications. Since an attacker could disrupt communications and defeat the revocation, it may be desirable for the server and each intermediate delegate to periodically reauthenticate delegates (by challenging them for possession of K_2) at some interval (say, an hour) that is much smaller than the timeout but not so often that performance is hampered.

A reasonable question may be why we require each system W_i in the chain to have a separate delegation key K_i, rather than simply forwarding to each W_i copy of K_1 used by the first system. The primary reason is to limit the scope of trust. If all systems share the same delegation key then W_1 cannot, at the end of the session, prevent a future attacker of W_1 from starting new delegation chains from W_1 if subsequent systems in the existing chain are corrupt and working in collusion with the attacker. This is a minor problem (since the corrupt systems can do considerable damage anyway through the existing delegation chain) but is easily avoidable by using different keys. Also, securely forwarding the key to the next system requires encryption, a service that we do not expect to be universally available for a number of reasons. Nothing in our delegation architecture depends on the use of encryption for confidentiality.

6 Practical Refinements

The structure of a delegation certificate used in figure 5,

$$\boxed{K_d \mid K_t \text{ as } W_t \text{ for } P \text{ until } T}$$

has the public delegation key K_t, the name of the delegated (target) system W_t, and the name of the original delegating principal in the chain P, all signed by the private component of the delegation key, K_d, of the previous delegator in the chain. Though adequate in principal, practical considerations dictate some further refinements.

6.1 Roles

So far we assume that the user has one identity whose full access rights he delegates when signing the first delegation certificate in the chain. We can provide more flexibility in the use of "roles" where users may have a number of aliases, each identified by a role name. Role names may appear on ACLs as if they were user names, and it is possible to set up both very restrictive roles (that don't have access to much of interest) as well as privileged roles (that have access to more than usual), by appropriate use of role names on ACLs. Within DSSA, roles are simply (small) groups, and they work very much like the groups in Unix, "`*.project`" names in Multics [Organick72], or rights identifiers in VMS.

By placing the role name in the delegation certificate, a user can specify that access available to only the role is delegated. This use of a role is a restriction on the delegation, and is accommodated by substituting the role name R for the the user's name P:

$$\boxed{K_d \mid K_t \text{ as } W_t \text{ for } R \text{ until } T}$$

The reference monitor will grant access only if R is named on the ACL of the object.

Since the reference monitor now uses the role name to enforce access, the reference monitor must ascertain that the role is authorized for the user whose smart card signed the first delegation certificate in the chain. Also, the real user's name P, in addition to the role name R, must be available to the reference monitor for auditing purposes. But the name P does not need to be bound into the certificate (it is simply communicated to the server as a "hint") because denial of service is the only effect of giving a false name. A hint is adequate because the reference monitor must fetch the user's authentication certificate from the directory service anyway in order verify the signature on the first delegation certificate, and the authentication certificate is what certifies P and its public key. The reference monitor determines whether R is an authorized role for P by similar lookup of certificates in the directory. The precise description of the mechanisms by which the user asserts a role at login time, and the validation of authorized roles, is a topic for a future paper.

6.2 UIDs and Public Keys as Identifiers in Place of Names

In any implementation, human-readable names for users and systems, as specified on ACLs, do not uniquely identify a user or system. Even in the case of a structured name

space (as in DNS or X.500), there is the need for soft links to permit transparent name space reorganizations (and other conveniences) as described in DSSA. Because of these links there may be more than one global name for a given principal, and it is possible that the name of a user or system listed on an ACL (created long ago) is not the same as the name of that user or system identified in the delegation certificate. It is not practical to change all ACLs every time any portion of the name space is altered because there is no way to find all the ACLs that contain a name. If we don't accommodate name changes in some way, the reference monitor has only two choices: (1) deny access if a name in a delegation cannot be found on the ACL; or (2) on every access, look up each name on the ACL in the directory service and fetch that directory entry's attributes to determine whether those attributes match those of the user identified by the delegation certificates. Choice (1) is undesirable because many name changes do not reflect a change in organizational affiliation, and choice (2), involving a number of directory lookups proportional to ACL size, could be a performance disaster.

To permit the reference monitor to quickly determine whether a name on the ACL refers to the same principal as a different name in the certificate, it is better to put unique identifying information (which does not change when names change) in the certificates and in the ACLs, along with the names. The reference monitor then matches up unique identifiers. If it finds a match, it looks up both names in the directory service to see if they refer to the same principal (by comparing certified information stored in the directory entry for that principal).

In place of names in certificates we could use public keys of the principals as unique identifiers, since public keys are not expected to change as often as names.[3] A public key is a reasonably good unique identifier for a delegated system, but is not good enough for a user. This is because a user may have multiple roles with different names, all sharing the same public key contained in his smart card. Public keys do not differentiate between roles.

A convenient source of unique names is a UID (unique-identifier) that is maintained by the directory service when a principal or role is created. Directory services are able to maintain UIDs fairly well, and UIDs can be used for other directory functions. Note that the directory service need not be trusted for uniqueness of these UIDs, since the reference monitor still verifies each ACL match with the actual credentials of the principal stored in the directory service. The worst that can happen if role UIDs aren't unique is that a user will delegate a right he doesn't have (e.g., because the reference monitor has interpreted the role's UID as belonging to someone else) and all accesses will fail. The UID is needed solely as an aid to search the ACL.

The UID of the principal or role, U_R, is inserted into the first delegation certificate signed by the smart card and copied into subsequent certificates. U_R replaces the role name R in the certificate. For delegated systems, which are not qualified by roles, we can use the public key K_{W_t} as the unique name in place of the system name W_t.

$$\boxed{K_d}\boxed{K_t \text{ as } K_{W_t} \text{ for } U_R \text{ until } T}$$

7 Restricted Delegations

A delegation certificate is very much like a capability that grants broad powers, permitting access to any object accessible to the identified role. There are a number of cases where a user might want to delegate only a subset of his rights rather than all of his rights: the ability to access a restricted set of files, or the ability to read, but not write his files. Adding such mechanisms to this architecture is not hard (we would add more fields to the delegation certificate), but there seems to be little reason to do so. It appears that almost all needs for restricted delegations can be better addressed using roles and other mechanisms.

7.1 Delegate Access to Subset of Files

A principal delegating to a system may want to delegate access to only a subset of its accessible objects. With some advance planning this type of restriction is readily implemented through the role mechanism as discussed in section 6.1. The user creates a restricted role for access to a limited set of objects, and places the restricted role name on the ACLs of only those objects. When authenticating to a system that is to have this limited access the user specifies the limited role and then does the usual full delegation. Only objects with the limited role on their ACLs will be accessible to the system. This concept can be extended to a role per individual object, but too many roles are likely to get out of hand since the user has to manage and be aware of them. Another problem with exclusively using roles to restrict access is that the user may not have the ability to alter the ACLs of all the objects to which he has access.

The current architecture does not support a generalized ability for a user to make an on-the-spot ad hoc decision to restrict access to specific objects on a per-delegation basis. Mechanisms such as "constraints" [Sollins88] and capabilities in the Monash system [Anderson86] offer means to dynamically restrict delegated access, and are candidates for inclusion in the delegation architecture if the need arises.

7.2 Delegate a Subset of Access Rights

DSSA does not constrain semantics of access requests or types of access rights that may be requested or recorded in ACLs because these details depend on the types of objects and operations implemented by each reference monitor. If a user is to restrict delegated access to some subset of possible rights, the syntax and semantics describing that that subset (specified in the delegation certificate) must be understood by all reference monitors likely to make use of the delegation.

[3] We recognize that uniqueness of independently generated public keys cannot be guaranteed, but the odds of duplication are sufficiently remote for our purposes. Intentional duplication of a public key is not possible unless the corresponding private key is compromised.

Since a universal standard for representation of access rights is not available, delegation of arbitrary subsets is difficult.

One might be able to define a common set of rights like "read" and "write" that all reference monitors are likely to implement, but restriction to a subset of these is unlikely to be useful except in special cases. Carrying out any activity on a remote system is likely to require rights such as "login" or "connect" to the remote system, rights defined by the system manager of the remote system (or owner of the remote object) which the user may know nothing about. In general, the user will not have enough information to select the proper subset of possible rights needed to accomplish a given task.

7.3 Delegation to Background Jobs

There are cases where a system must continue to operate on behalf of a user after the user logs out. If success of an operation depends on delegations, then the delegations must remain valid for the duration of the operation. In many cases the activity will complete within the expiration time of the original delegation certificate so no change need be made to the architecture described so far: the delegated system simply retains its certificates even after the user logs out until all requested operations are completed or the delegations expire.

But there are several cases where the system might have to operate well beyond the normal expiration time of delegation certificates. Deferred background or batch jobs (those that are scheduled to run at a specific time in the future, including jobs that rerun themselves on a periodic basis) are the most common examples, although any batch job could remain pending in queues for an arbitrary length of time before it runs. If such batch jobs require delegations, then the delegations must be in effect when the job runs. A delegation to such a job might have to have a very long timeout, and long timeouts reduce the security offered by the timeout mechanism. Nonetheless, we believe that extending the timeouts for such jobs is an acceptable risk in most cases. After all, if the user expects a system to operate on his behalf at some time in the future, he should be willing to delegate to it for that expected amount of time.

For batch jobs that run far in the future the role mechanism is a suitable way to restrict access. Setting up such batch jobs, especially those that run on a daily basis, normally requires some advance planning by the user, and such jobs usually carry out for very specific functions (e.g., check for mail). It is not unreasonable to expect users to set up special roles for these selected jobs.

7.4 Limited Delegation to Servers

There is at least one specific case where limited delegation to a server might make sense. This involves print servers.

When submitting a print request to a server the user might like to limit the server's access to only the file to be printed, only for a limited time, and only for a single access.

Possibly the most reasonable approach is to simply copy the file to the print server as part of the print request, rather than having the print server access the file directly at time of printing. The only major drawback from this scheme is that the user may have to wait for his file to be queued for printing, as the print server will only be able to store a finite number of files.

In today's printing architectures a print job may consist of multiple included files and fonts fetched by the print server from places that the user knows nothing about. Under these sophisticated schemes simple delegation of rights to a single named file will not generally permit the file to be printed.

7.5 Delegation to Users

Within DSSA we do not intend users to be the targets of delegation. Delegation requires the delegator to exchange information with the delegated principal. While we could devise a scheme whereby two smart cards communicate with one another in a trusted manner, doing so in practice would be cumbersome. Also, a smart card that is the target of a delegation would have to generate and remember a separate delegation key for each delegation for which it is a target, and would have to know which key to use in a given session. Delegating to a user is better handled with some advance planning in the use of roles.

8 Improving Performance

It is well known that calculations using RSA public key techniques are computationally intensive. The primary operations with which this architecture needs to be concerned are digital signatures (using the private key), verifying signatures (using the public key) and key generation (for delegation keys). Key generation is by far the most time consuming operation. By choosing short public exponents and long private exponents, we can arrange for signature verification, which is done relatively frequently, to be quite fast, while the less frequent operation, digital signing, is much slower.

The need for a signature for each delegation is not expected to add much delay to the establishment of a session. This is because protocols can be structured so that signature takes place in parallel with network communication. While we envision situations where a system may need to check large numbers of signatures (e.g., a server that receives access requests from large numbers of systems), we do not foresee a case where a system has to sign large numbers of certificates.

Key generation appears to be the most serious performance concern, and this architecture requires a new key for each delegation in each chain. To address this problem we have investigated several optimizations:

- **Generate keys in advance.** During periods of idle processing, systems can generate keys in anticipation of delegation and can keep a cache of ready keys. These keys can safely be kept in online storage as long as they are erased from disk when used for a delegation and not used if the disk has been removed.

- **Reuse unused keys.** Depending on the implementation of the protocols, many delegations will occur but will never be used. That is, a system will delegate to another system at the beginning of a session in case remote access is required, but the resources might be local to the delegated system and the delegation may never be required. In this case the same delegation key can be reused for the next delegation. This technique is only secure if the delegation certificate is destroyed by both delegator and delegate and never disclosed to a third party.

- **Use short keys.** Time to generate a key is a function of key length. Unlike principals' authentication keys that are relatively permanent and stored in authentication certificates in the directory service, delegation keys are only needed for the duration of a session. As long as the public component of the delegation key is never used for confidentiality (it is not in our architecture), no harm is done if the key is compromised after the delegation expires. We can therefore probably get by with much shorter (less secure) keys for delegation than would be needed for long term use.

- **Avoid unnecessary keys.** The architecture is described in terms of principals rather than users and computer systems, where a principal may be a process or application on a system. It is possible, then, for many principals to exist on the same computer system. In the case where the delegator and delegate are principals under control of the same reference monitor (i.e., are two processes on the same system), it is not necessary to use cryptographic techniques to certify the portion of the delegation chain involving the two principals. Instead, the reference monitor maintains a single delegation key for all processes under its control that are part of the same chain. The reference monitor keeps track of the keys that processes may use. Only when the delegation chain "enters" and "leaves" the system is cryptography required.

- **Copy delegation keys.** Although we stated earlier that ideally we would like each system to have its own delegation key, significant security is not lost in most cases by simply forwarding the key from one system to the next. In the event that key generation becomes a problem in certain applications, the architecture permits any two systems to resort to key copying without affecting any of the other aspects of the architecture.

While key generation performance is not something to ignore, we think that there are enough ways to minimize the problem that feasibility of implementing the delegation architecture is not in danger.

9 Conclusion

The delegation scheme presented in this paper is an important component of Digital's Distributed System Security Architecture. Items yet to be done include a formal definition and, of course, prototype implementations. A formal definition and justification for all of DSSA is being written using notation and rules to express statements of belief and trust. The logic we are using is an extension of that defined by Burrows, et al. [Burrows88]. Rangan [Rangan88] has developed a more complete method for formalizing reasoning and trust in a distributed system, but it appears that a simpler logic will be adequate to express the basic concepts of DSSA.

The goal of DSSA is to elevate the level of security of a heterogeneous distributed system to that routinely available with today's state-of-the-art standalone secure operating systems. The architecture is not a theoretical exercise, but was designed with numerous practical implementation issues in mind. It is intended to be implemented, at reasonable cost and performance, using technology that we expect to be available in the very near future. Initial installments of this architecture will be deployable without a wholesale replacement of existing hardware, operating systems, and applications, and we are currently developing prototypes to demonstrate this feasibility.

At the same time, DSSA does not commit us to undesirable tradeoffs that may be due to limitations of today's technology. Since architectures, once implemented, do not die easily, we have set our sights on a long term goal to provide the maximum level of security for the widest possible variety of computing styles and applications.

References

[Ames83] S. R. Ames, Jr., M. Gasser, and R. R. Schell, "Security Kernel Design and Implementation: An Introduction," *Computer*, 16(7):14–22, July 1983.

[Anderson86] M. Anderson, R. D. Pose, and C. S. Wallace, "A Password-Capability System," *The Computer Journal*, 29(1):1 8, 1986.

[Birrell86] A. D. Birrell, B. W. Lampson, R. M. Needham, and M. D. Schroeder, "A Global Authentication Service without Global Trust," *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1986, pp. 223 230.

[Burrows88] M. Burrows, M. Abadi, and R. M. Needham, "Authentication: A Practical Study in Belief and Action," *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Vardi, ed., 1987.

[CCITT88a] International Telegraph and Telephone Consultative Committee (CCITT), X.500, *The Directory – Overview of Concepts, Models and Services* (same as ISO 9594-1).

[CCITT88b] CCITT X.509 *The Directory - Authentication Framework* (same as ISO 9594-8).

[DOD85] Department of Defense, *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985.

[ECMA89] ECMA/TR46, "Security in Open Systems – A Security Framework" and ECMA/TRxx, "Security in Open Systems – Data Elements and Service Definitions."

[Gasser89] M. Gasser, A. Goldstein, C. Kaufman and B. Lampson, "The Digital Distributed System Security Architecture," *Proceedings of the 1989 National Computer Security Conference.*

[Girling82] C. G. Girling, "Object Representation on a Heterogeneous Network," *Operating Systems Review* 16:49–59, October 1982.

[Girling83] C. G. Girling, "Representation and Authentication in Computer Networks," Ph.D. Thesis, University of Cambridge, Queens' College, April 1983.

[ISO88b] International Standards Organization, ISO 7498-2, *Security Architecture.*

[Karger86] P. A. Karger, "Authentication and Discretionary Access Control in Computer Networks," *Computers and Security* 5:314-24, 1986, and *Computer Networks and ISDN Systems* 10(1):27-37, 1985.

[Kohl89] J. Kohl, B. C. Neuman, J. Steiner, "Kerberos Version 5 draft RFC", Project Athena, Massachusetts Institute of Technology, 3 August 1989.

[Lampson86] "Designing a Global Name Service," *Proc. 4th ACM Symposium on Principles of Distributed Computing*, pp. 1-10, Minaki, Ontario, 1986.

[Linn90] J. Linn, "Practical Authentication for Distributed Computing," *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1990.

[Miller87] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Kerberos Authentication and Authorization System," Project Athena Technical Plan, Section E.2.1, Massachusetts Institute of Technology, 10 April 1987.

[Mullender85] S. J. Mullender, "Principles of Distributed Operating System Design," Ph.D. Thesis, Vrije University, Amsterdam, The Netherlands, 1985.

[Organick72] E. I. Organick, *The Multics System: An Examination of Its Structure*, MIT Press, Cambridge, Massachusetts, 1972.

[NCSC87a] "Final Evaluation Report of Security Dynamics Access Control Encryption System," National Computer Security Center, 9800 Savage Road, Fort George G. Meade, MD 20755-6000, CSC-EPL-87/001 Library No. S228,455.

[Racal] "**WATCH**WORD Generator User's Manual," Racal-Guardata Limited, Richmond Court, 309 Fleet Road, Hampshire, England GU138BU.

[Rangan88] P. V. Rangan, "An Axiomatic Basis of Trust in Distributed Systems," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1988, pp. 204–211.

[Redell74] D. D. Redell, "Naming and Protection in Extendible Operating Systems," Ph.D. Thesis, University of California, Berkeley, CA, published as Project MAC TR-140, Massachusetts Institute of Technology, Cambridge, MA, November 1974.

[Rivest78] R. L. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM*, 21(2):120 126, 1978.

[Sollins88] Karen R. Sollins, "Cascaded Authentication" *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1988, pp. 156-163.