The CRISIS Wide Area Security Architecture[*]

Eshwar Belani[†] Amin Vahdat[†] Thomas Anderson[‡] Michael Dahlin[§]

Abstract

This paper presents the design and implementation of a new authentication and access control system, called CRISIS. A goal of CRISIS is to explore the systematic application of a number of design principles to building highly secure systems, including: redundancy to eliminate single points of attack, caching to improve performance and availability over slow and unreliable wide area networks, fine-grained capabilities and roles to enable lightweight control of privilege, and complete local logging of all evidence used to make each access control decision. Measurements of a prototype CRISIS-enabled wide area file system show that in the common case CRISIS adds only marginal overhead relative to unprotected wide area accesses.

1 Introduction

One of the promises of the Internet is to enable a new class of distributed applications that benefit from a seamless interface to global data and computational resources. A major obstacle to enabling such applications is the lack of a general, coherent, scalable, wide area security architecture. In this paper, we describe the architecture and implementation status of CRISIS, a wide area authentication and access control system. CRISIS forms the security subsystem of WebOS, a system that extends operating system abstractions such as security, remote process execution, resource management, and named persistent storage, to support wide area distributed applications.

Today, many wide area applications are limited by the lack of a general wide area security system. As one example, one of the goals of the WebOS project is to build a scalable SchoolNet service, to provide a safe place for the tens of millions of K-12 students in our states (California, Washington, and Texas) to learn and play. One challenge to making SchoolNet a reality is building scalable network services, for example, to provide a highly available email account to every student, without requiring a system administrator at every location. This paper focuses on an equally difficult challenge – maintaining the confidentiality and integrity of data and resources, for example, so that unauthorized people cannot obtain information about school children. As another example, we (as a geographically distributed development team) would like to use WebOS to allow us to seamlessly access any file or computational resource at any of our sites; once CRISIS is fully operational, we plan to rely on it to protect our development environment from external attacks. As a final example, we have built Rent-A-Server [Vahdat et al. 1997], a system to dynamically replicate and migrate Internet services, to gracefully handle bursty request patterns, and to exploit geographic locality to reduce latency and congestion. To be practical, however, RentAServer requires the ability to securely access and control remote data and computational resources (e.g., CPUs and disks).

An initial approach for supporting secure access to remote resources is to simply employ an authenticated login protocol. Unfortunately, this approach is inadequate because many wide area applications require more fine-grained control over access to remote resources. Further, the administrative overhead of creating and maintaining separate accounts in all domains where users wish to run jobs can be prohibitive. For example, it would be difficult to use authenticated login to support a user job running on an anonymous compute server in a remote administrative domain that needs access to a single file on the user's home file system.

Another approach for supporting secure wide area applications is to add fine-grained rights transfer to an existing authentication system, such as Kerberos [Steiner et al. 1988]. However, while Kerberos has proven quite successful for local area networks in a single adminis-

[*]This work was supported in part by the Defense Advanced Research Projects Agency (N00600-93-C-2481, F30602-95-C-0014), the National Science Foundation (CDA 9401156), Sun Microsystems, California MICRO, Novell, Hewlett Packard, Intel, Microsoft, and Mitsubishi. Anderson was supported by a National Science Foundation Presidential Faculty Fellowship. For more information, please see `http://now.cs.berkeley.edu/WebOS`, or send email to `vahdat@cs.berkeley.edu`.

[†]Computer Science Division, University of California, Berkeley

[‡]Department of Computer Science and Engineering, University of Washington, Seattle

[§]Computer Science Department, University of Texas, Austin

trative domain, it faces a number of challenges when extended to the wide area. First, Kerberos has no redundancy; security is undermined if even a single authentication server or ticket granting server is compromised, allowing an adversary to impersonate any principal that shares a secret with the compromised authentication server. In the wide area, the number of such single points of failure scales with the size of the Internet. Further, Kerberos requires synchronous communication with the ticket granting server in order to set up communication between a client and server; in the wide area, synchronous communication with a hierarchy of ticket granting servers is required. Given that the Internet today is both slow and unreliable, this can have a significant effect on availability and performance as perceived by the end-user. Although Kerberos servers could conceivably be replicated to improve availability, the servers would need to be geographically distributed to hide Internet partitions, providing an intruder even more points of attack.

Public key cryptography seems to hold out the promise of improving availability and security in the wide area, by eliminating the need for synchronous communication with a trusted third party. The public key of every principal (user or machine) can be freely distributed; provided the public keys are known, two principals can always communicate if they are connected, regardless of the state of the rest of the Internet. Unfortunately, this also comes at a cost; any compromise of a private key requires that every entity on the Internet be informed of the compromise. This is analogous to Kerberos, in that the number of single points of failure (in this case, the number of private keys) scales with the size of the Internet.

In this paper, we present the design and implementation of CRISIS, a system for secure, authenticated access to wide area resources. To avoid an ad hoc design where features are thrown together in an attempt to prevent all known types of security attacks, our approach is the systematic application of a set of design principles. These principles are inspired by analogy with other areas of distributed systems, where scalability, performance and availability can be achieved through redundancy, caching, lightweight flexibility, and localized operations. A goal of the CRISIS architecture is to demonstrate that these principles can also be applied to increasing the security of wide area distributed systems.

Specifically, the principles underlying the design of CRISIS include:

- Redundancy: There should be no single point of failure in the security system; any attack must compromise two different systems in two different ways. For example, every certificate (such as that identifying a user's public key) is revocable within a few minutes; thus, an attacker must not only steal a private key, but must do so without detection or must also corrupt the revocation authority. The notion of using redundancy to improve security is an old one, but it has not been systematically applied. For example, Internet firewalls are used to protect organizations from security attacks from the outside world, to hide the fact that most local operating systems are notoriously insecure. Unfortunately, this has reduced the pressure on local operating systems to improve their security, so that once inside a firewall, an attacker usually has nearly free reign. Similarly, Internet browsers purport to be able to safely execute Java applets, supposedly rendering traditional operating system security irrelevant. The ongoing discovery of security holes in Java verifier implementations [Dean et al. 1996, Sirer et al. 1997], however, has led us to run remotely executing programs in a restricted environment [Goldberg et al. 1996], *in addition to* using a verifier.

- Caching: CRISIS caches certificates to improve both performance and availability in the presence of wide area communication failures; while all certificates are revocable, they are given a revalidation timeout to hide short-term unavailability of the revocation authority due to reboots or Internet partitions. This principle was inspired by research in mobile file systems which argued that caching can improve availability, e.g., for disconnected operation [Kistler & Satyanarayanan 1992].

- Least Privilege: Users should have fine-grained control over the rights they delegate to programs running on their behalf, whether local or remote; to be useful, this control must be easy to specify. CRISIS provides two mechanisms to support least privilege: *transfer certificates*, limited unforgeable capabilities which allow for confinement and can be immediately revoked, and lightweight roles that can created by users without the intervention of a system administrator and without requiring changes to access control lists.

- Complete accountability: CRISIS logs all evidence used to determine if a request satisfies access control restrictions, locally at the machine guarding access. Most existing systems log only coarse-grained authentication information, making accountability in the face of rights transfer difficult (e.g., some systems may want to grant a request only if every member along a chain of delegation is trusted). Our design differs from previous efforts to add capability-like certificates to Kerberos [Neuman 1993], which require distributed logging by all ticket granting servers involved in propagating a request in the wide area.

- Local Autonomy: Each user identity is associated with a single administrative domain, and that domain is solely responsible for determining how and when user privileges are distributed beyond the local administrative domain. Each local domain is also responsible for determining the level of trust placed in any given remote domain. A design relying on deference to a global, centralized authority is not only less flexible, but less likely to be widely adopted [Birrell et al. 1986].
- Simplicity: Simple designs are easier to understand and implement. In the context of security, simplicity is especially important to minimize the probability of an security hole resulting from an implementation error. A provably secure but highly complex system architecture is unlikely to be either properly implemented by system designers or properly understood by end users (for example, leading to errors in setting up access control lists).

CRISIS is loosely based on the DEC SRC security model [Lampson et al. 1991]. Relative to their work, one of our contributions is to simplify the model by using *transfer certificates* as the basis of fine-grained rights transfer across the wide area. Transfer certificates provide an intuitive model for both rights transfer and accountability, as they allow a complete description of the chain of reasoning associated with a transfer of rights. In addition, *revocation* is a first class CRISIS operation; even privileges described by transfer certificates (which are typically valid only for a limited period of time) can be revoked immediately. CRISIS also provides for explicit reasoning about the state of loosely synchronized clocks, an important consideration for wide area applications. Further, CRISIS supports user-defined lightweight roles, to capture persistent collections of transferred rights (e.g., “Tom running a job on remote supercomputer”). Finally, in contrast to the DEC SRC work which was implemented in the kernel of a platform that is no longer available, CRISIS is designed to run portably across multiple platforms, a requirement for a wide area security system to be useful in practice.

The rest of this paper describes CRISIS in more detail. We first provide some motivating examples for CRISIS along with a quick review of relevant technology in Sections 2 and 3. We then outline the CRISIS architecture in Section 4, followed by a detailed example of how CRISIS is used in Section 5. We evaluate the performance of our implementation in Section 6, and discuss related work in Section 7. We summarize our results in Section 8.

2 Motivation

CRISIS is the security subsystem for WebOS [Vahdat et al. 1997]. The goal of WebOS is to provide operating system primitives for wide area applications now available only for a single machine or on a local area network. Such abstractions include authentication, authorization, a global file system, naming, resource allocation, and an architecture for remote process execution. To date, wide area network applications have been forced to re-implement these services on a case by case basis. WebOS aims to ease and support network application development by providing a substrate of common OS services.

The focus of this paper is the architecture of the WebOS security subsystem which cuts across all other aspects of the system. Below, we briefly describe a number of scenarios we have used to drive the CRISIS design:

- SchoolNet: One motivating example is to provide Internet services such as email, Web page hosting, and chat rooms for very large numbers of school children. One desirable feature of such a system is to allow geographically distributed children to be able to interact with one another, while keeping both the interactions and the identities of those involved private. Further, to be useful to school children, the security system must work with only limited direction from end users (e.g., you cannot trust a fifth grader to correctly set up access control lists).
- Wide Area Collaboration: Users in separate administrative domains should be able to collaborate on a common project. For example, a project's source code repository should be globally accessible to authorized principals for check in/check out; in addition, unique hardware (such as supercomputers) should be seamlessly accessible independent of geographic location.

- Geographically Distributed Internet Services: If it were easy to geographically replicate and migrate Internet services, end-users would see better availability, reduced network congestion, and better performance. Today, only the most popular sites can afford to be geographically distributed; for example, Alta Vista [Dig 1995] has mirror sites on every major continent, but these mirrors are physically administered by DEC, manually kept up to date, and visible to the end user. One of our goals is to make all this transparent, to make it feasible for third party system administrators to offer computational resources strategically located on the Internet for rent to content providers; in the limit, content providers could become completely virtual, with the degree and location of replicas dynamically scaled based on access patterns.

- Mobile Login: Users should be able to login and to access resources from any machine that they trust. Secure login requires mutual authentication. Thus, users will only log into machines certified to have been booted properly by a trusted system administrator. Likewise, local system administrators enforce which users are allowed login access (e.g. login to Berkeley by Stanford users would be disallowed outright). Finally, users should be allowed to adopt restricted roles representing the amount of trust they have for the machine being logged into.

- Encrypted Intermediate Caches: To improve application performance, untrusted third party servers may be utilized to cache encrypted private data. A special key would be created to encrypt the data rather than using the key of a particular principal; this key would then only be distributed to authorized users. One path to implementing such an application would be the use of Active Networks [Tennenhouse & Wetherall 1996] where intelligent routers can be utilized to perform the caching.

- Large Scale Remote Execution: Principals should be able to exploit global resources to run large scale computations. For example, NASA is placing petabytes of satellite image data on-line for use by earth scientists in predicting global warming. It is impractical to access this information using the current Internet "pull" model; scientists need to be able to run filters remotely at the data storage site to determine which data is useful for download. These filters should have access to necessary input (e.g., the filter executables) and output files (e.g., files into which the results are to be stored) on the scientist's machine, but to no other potentially sensitive data. Further, the remote computation environment should be protected from any bugs in the filters written by the scientists.

3 Background

One of the first steps in providing a secure Internet information system is to allow for encrypted, authenticated communication between arbitrary endpoints over an inherently insecure wide area network. Traditionally, the two choices for encryption and authentication are using secret key or public key cryptography. Encryption ensures an eavesdropping third party cannot alter the integrity or determine the content of the communication. Authentication allows for the identity of the principal at the opposite end of a communication link to be securely identified.

We choose public key over secret key (though one can be simulated with the other [Lampson et al. 1991]) because of the synchronous communication usually required by secret key systems. Secret key systems require a trusted third party that shares a secret with every potential communication endpoint. Although this requirement impacts system performance and availability by imposing an extra step in initiating communication, it is reasonable in the local area because the number of communication endpoints are limited and the network is more reliable. In the wide area, such a requirement strains system scalability because synchronous communication with a hierarchy of trusted third parties is required. Public key systems also require trusted third parties to produce certificates identifying principals with their public keys. However, these certificates can be cached (with a timeout), removing the need for synchronous communication with a third party to set up a communication channel. Allowing for direct communication in this fashion offers two advantages. First system availability is improved because an unavailable third party does not necessarily prevent communication. Second, system performance is improved by removing a communication step to a third party behind a potentially slow link.

In addition to public key encryption, we employ a number of other technologies to assist in development and to reduce the chance of introducing security flaws. We use Janus [Goldberg et al. 1996] to "sandbox" locally running applications that are not fully trusted. Janus runs at user-level, employing the UNIX System V `proc` file system to intercept potentially dangerous system calls and to disallow accesses outside of each process's defined sandbox. The implementation has negligible performance overhead and can sandbox unmodified applications. CRISIS also employs the SSL [Hickman & Elgamal 1995] protocol to provide transport network layer privacy and integrity of data, using encryption and message authentication codes. SSL supports a wide variety of cryptographic algorithms and is being deployed into wide area applications. Finally, as will be described in the next section, we use the X.509 syntax [Con 1989] to encode all certificates in CRISIS. The ITU-T Recommendation X.509 specifies the authentication service for X.500 directories, as well as the X.509 certificate syntax. The X.509 certificate syntax is supported by a number of protocols including PEM, S-HTTP, and SSL.

4 System Architecture

The goals of the CRISIS architecture can be described in two parts. First, users should be allowed secure access to global resources such as files, CPU cycles, or storage from anywhere in the world. Next, resource providers need mechanisms for authenticating those requesting their services and for authorizing those with the proper credentials. In this section, we provide a high-level view of the system architecture before detailing example usage in the next section.

In the following discussion, we assume the presence of three basic entities, adapted from the SRC logic [Lampson et al. 1991]:

- *Principals*: Principals are sources for requests. Examples of principals include users and machines. Principals make statements (requests, assertions etc.), have names, and can be associated with privileges.
- *Objects*: Objects are global system resources such as files, processors, printers, memory, etc.
- *Reference Monitors*: Once an access request from a principal to an object is authenticated, the reference monitor determines whether or not to grant the principal access to the object.

Consider the scenario where a user in California wishes to run a job at Texas which requires access to two input files. In turn, the job at Texas decides to subcontract a portion of its work to a machine in Washington. This sub-contracted work only needs access to the second input file. More formally, P_1 is a user in California, while

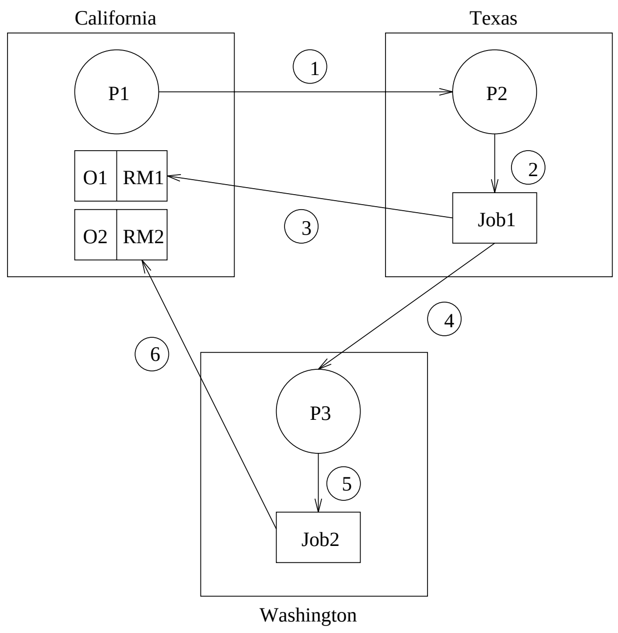

Figure 1: This figure describes a sample scenario where a user, P_1 requests a machine P_2 to run a job on its behalf. In turn P_2 sub-contracts a portion of the job to another machine P_3 in a separate administrative domain.

P_2 and P_3 are machines willing to run jobs located at Texas and Washington respectively. O_1 and O_2 are objects (e.g. input files) located in California, with RM_1 and RM_2 their associated reference monitors. Assuming that P_1 (and only P_1) possesses access privileges to O_1 and O_2, consider the following sequence of events (summarized in Figure 1):

P_1 `states that` P_2 `can access` O_1 `and` O_2 `until time` T_1`.`
P_1 `requests that` P_2 `execute a job on its behalf (steps 1 and 2).`
P_2 `requests access to` O_1 `from` RM_1 `(step 3).`
P_2 `states that` P_3 `can access` O_2 `until time` T_2`.`
P_2 `requests that` P_3 `execute a job on its behalf (steps 4 and 5).`
P_3 `requests access to` O_2 `from` RM_2 `(step 6).`

To carry out the above scenario in WebOS, the security system must support:

- Statements: Statements may be requests, declarations of privileges, or transfer of privileges. The identity of the principal making a statement must be verified, and all statements must be revocable.
- Associating privileges with processes: While a particular machine may possess a large set of privileges, individual processes only have access to a subset of these privileges. Similarly, users may only wish to grant a subset of their available privileges to each of their programs.
- Distributing statements across the wide area: A protocol for trust between different administrative domains must be established to allow for validation of privileges and identities across administrative boundaries.
- Time: The transfer of privileges can only be valid for a limited period of time. CRISIS requires a clear protocol for reasoning about time, and a methodology for isolating failures in cases where clock skews lead to a security breach.
- Authorization: When a reference monitor receives a request to access an object, it must determine the identity of the requester, ascertain the principal's privileges, and finally decide whether the request is authorized.

Our solutions to each of the above are described in the following subsections.

4.1 Validating and Revoking Statements

All statements in CRISIS, including statements of identity, statements of privilege, and transfer of privilege, are encoded in *certificates*. CRISIS certificates are signed by the principal making the statement and then counter-signed by a principal of the signer's choosing. Each signature uses a separate timeout: the principal's signature is issued with a long timeout, while the counter-signature is issued with a short timeout. The counter-signer (i) checks if the statement has been revoked and (ii) refreshes its *endorsement* (by applying a new counter-signature with a new timeout to an expired certificate) of certificates, indicating that the rights are are still valid. Since we are building a public key system, the certificate's author need not be aware of the certificate's destination when it is created. Any principal with access to the certificate can determine the statement's author. Our certificates use the X.509 [Con 1989] standard format. CRISIS employs two basic types of certificates:

- *Identity Certificates*: An identity certificate associates a public key with a principal for a certain period of time. Depending on the type of principal (person, machine, process, etc.), an identity certificate can also specify a number of the principal's properties, such as name or organization.
- *Transfer Certificates*: Transfer certificates transfer a subset of a principal's privileges to another principal. A principal P_1 can use a transfer certificate to transfer to P_2 access rights to any objects it owns (e.g. O_1 and O_2). These transfers are expressed as a list of capabilities, resulting in arbitrary length certificates. Individual transfer certificates can be chained or they can disallow further transfers. Each successive link in a chain (e.g., from P_2 to P_3) can only refine, and never expand, the rights transferred. Transfer certificates are presented to reference monitors as proof of access rights. The reference monitor is able to verify the sequence of statements and the identity of each principal in the chain, ensuring complete accountability.

Of course, identity certificates must be signed by an authority trusted by both endpoints of a communication channel (see section 4.3 for the case where no single authority is so trusted by both parties). This trusted third party, called the Certification Authority (CA), maps public keys to principals and maintains a Certificate Revocation List enumerating all public keys that have changed or that have been knowingly compromised. In CRISIS, CA's sign all identity certificates with a long timeout (usually weeks) and identify a locally trusted on-line agent (OLA) responsible for counter-signing the identity certificate with a relatively short timeout (usually hours).

The redundancy of a split CA/OLA approach offers a number of advantages. First, to successfully steal keys, either both the OLA and CA must be subverted or the CA must be subverted undetected. By making key revocation a simple operation (as described below), we are able to pro-actively revoke keys when a CA comes under attack. Further, the CA is usually left off-line since certificates are signed with long timeouts, increasing system security since an off-line entity is more difficult to attack. Another advantage of the split CA/OLA approach is that a malicious CA is unable to revoke a user's key, issue a new identity certificate, and masquerade as the user without colluding with the OLA [Crispo & Lomas 1996]. Also, while a malicious OLA can mount a denial of service attack, the CA is still able to reissue new certificates employing a different OLA. Finally, this approach improves system performance because certificates can be cached for the timeout of the counter-signature, removing the need for synchronous three-way communication in the common case.

We generalize the OLA to make revocation a first class operation in CRISIS. All certificates are revocable modulo a timeout. To revoke a particular privilege, the OLA which endorses the certificate must be informed that the certificate should no longer be endorsed. Once the timeout period for the endorsed certificate expires, the rights described by the certificate are effectively revoked because the OLA will refuse re-endorsement for that certificate. Revocation is used not only for exceptional events such as stolen keys. For example, the rights of a remote job are revoked upon its completion or when a user decides to kill the job. As another example, privileges associated with a login session are revoked on user logout.

The use of transfer certificates in CRISIS also simplifies both the implementation of and reasoning about delegation, which allows one principal to act on behalf of a second principal. Such delegation is useful in many contexts. For example, a database server will receive requests from many users, with individual operations executed in the context of the rights of a single user. It is important that a user's privileges are not amplified by employing the rights of the server. Such delegation is difficult to properly design. For example, early versions of UNIX `sendmail` were *setuid root* to allow the program to write to any user's mailspool. However, a bug allowed users to write any system file to a mail message addressed to themselves.

In CRISIS, users sign transfer certificates allowing servers to act on their behalf for accessing files, running jobs, etc. Servers provide these certificates to reference monitors when making requests on behalf of a user (as opposed to certificates describing their own rights), reducing the chance of the server being granted access on its own behalf when acting on a user's behalf. Relative to the SRC system [Lampson et al. 1991], where reference monitors use a pull model to search for proof that a principal should be granted access, CRISIS transfer certificates reduce complexity and hence the chance that an implementation error will lead to unauthorized accesses.

4.2 Processes and Roles

4.2.1 Security Domains

Given the abilities to authenticate principals, CRISIS also requires a mechanism for associating privileges with running processes. Each CRISIS node runs a security manager responsible for mediating access to all local resources and for mapping credentials to *security domains*. In CRISIS, all programs execute in the context of a security domain. For example, a login session creates a new security domain possessing the privileges of the principal who successfully requested login. As will be described in Section 5.1, a security domain, at minimum, is associated with a transfer certificate from a principal to the local node allowing the node to act on the principal's behalf for some subset of the principal's privileges.

Processes are able to access wide area resources through resource providers responsible for managing each remote resource, such as processor cycles or disk space. In conjunction with security managers, resource providers determine the access privileges of processes requesting resources. CRISIS nodes currently run the following resource providers, each with their own set of reference monitors:

- *Process Managers*- A Process Manager is responsible for executing jobs on requested nodes. The Process Manager identifies the security domain associated with a request, obtains the credentials associated with the domain from the security manager, and then attempts to satisfy the request.
- *WebFS* - A WebFS server implements a cache coherent global file system. Similar to the Process Manager, upon receiving a file access request, the WebFS server determines the security domain from the security manager. Using this information, the WebFS server determines whether the access should be granted or denied.
- *Certification Authorities* - As described above, CA's take requests for creating identity certificates. The CA maintains a reference monitor with the list of principals authorized to create, modify, or invalidate identity certificates.

The interaction between resource providers, security domains, and security managers are described through the CRISIS protocols for login, file access, and remote process execution in Section 5.

4.2.2 Roles

In the wide area, it is vital for principals to restrict the rights they cede to their jobs. For example, when log-

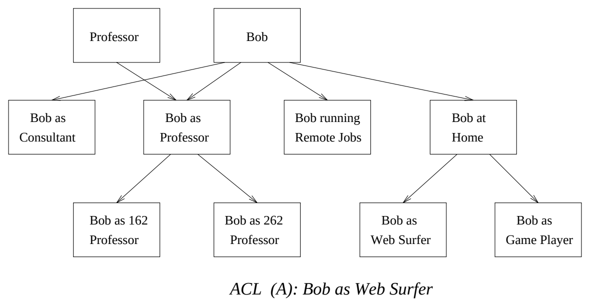

Figure 2: This figure describes how users may arrange their roles in a hierarchical fashion where each node in the tree possesses all the privileges of all of its direct descendants.

ging into a machine, a principal implicitly authorizes the machine and the local OS to speak for the principal for the duration of the login session. Whereas with private workstations, users generally have faith that the local operating system has not been compromised, confidence is more limited when using anonymous machines across the wide area, for example, to run large scale simulations. Roles associate a subset of a user's privileges with a name, allowing users a convenient mechanism for choosing the privileges transferred to particular jobs.

A principal (user) creates a new role by generating an identity certificate containing a new public/private key pair and a transfer certificate that describes a subset of the principal's rights that are transferred to that role; an OLA chosen by the principal is responsible for endorsing the certificates. Thus, in creating new roles, principals act as their own certification authority. The principal stores the role identity certificate and role transfer certificate in a *purse* of certificates that contains all roles associated with the principal. The purse is stored in the principal's home domain. While it is protected from unauthorized access by standard OS mechanisms, the contents of the purse are not highly sensitive since each entry in the purse simply contains a transfer certificate naming a role and potentially describing the rights associated with that role. The principal also stores each role's private key —encrypted by a password unique to the role —in the file system.

CRISIS roles are more lightweight than the roles described in other security systems (e.g., [Lampson et al. 1991]). First, they can be created by the user without requiring intervention from a centralized authority, allowing the CA to remain off-line. Next, while ACLs can be modified to describe a particular role's privileges, roles can also act as persistent lightweight capabilities. The transfer certificate used to create the role can describe the exact access rights possessed by the role (e.g., read access to files A, B, and C).

Further, transfer certificates can be used to arrange roles in a hierarchy, with the principal's most privileged role serving as the hierarchy's root. Such a hierarchy can be used in two ways. In a *single inheritance* model, each role possesses a strict subset of its parent's privileges. Thus, ACLs can be used to describe the “minimum” privilege level required to access a given object (a role is given access to an object only if it is a direct ancestor of a role in the object's ACL). With a *multiple inheritance* model, roles can draw upon the privileges of multiple ancestors. Figure 2 presents an example of such a hierarchy and applications of the two models. The object, A, can only be accessed by *Bob as Web Surfer* or one of its direct ancestors (*Bob at Home* or *Bob*). The Figure also illustrates that *Bob as Professor* (and its descendants) inherits privileges from both *Bob* and the generic, *Professor*. This may be useful, for example, to express that professors are able to read student accounts but to allow such access to Bob only when he is acting as a professor.

In CRISIS, creating a new group is similar to creating a new role. A principal creates a new group by acting as a CA to create an identity certificate naming the new group. The creating principal then signs transfer certificates to all group members, specifying both membership and any update privileges associated with the group, for example, whether the member has the ability to add or remove other group members. The newly created group name can then appear on ACLs like any other principal name.

4.3 Hierarchical Trust

We assume the presence of multiple, autonomous administrative domains, and that each domain has at least one trusted CA/OLA pair. CA's in different administrative domains are not equally trusted. Thus, CA's are arranged hierarchically, with individual CA's determining which parents, siblings, or children are trusted (and to what extent). The hierarchical arrangement of CA's builds on our model of implementing roles, where principals act as CA's in creating roles with the locally trusted CA acting as the principal's parent in a global hierarchy.

The manner in which the hierarchy is traversed is based on the theory presented in [Birrell et al. 1986]. In this model, a CA cannot speak for a principal who belongs to a descendant's domain, allowing separate administrative

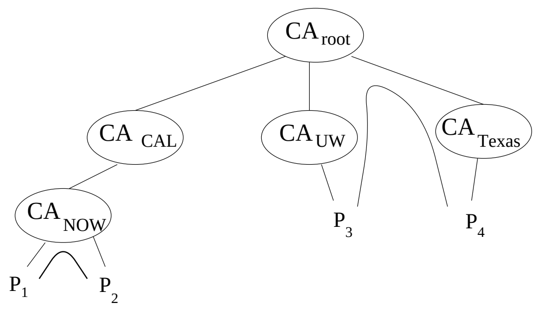

Figure 3: This figure describes how principals in different administrative domains can mutually authenticate. A path of trust is established through the principals' least common ancestor.

domains to maintain local autonomy. Thus, a principal receiving a certificate endorsed by a CA in a foreign administrative domain believes the certificate valid only if a *path of trust* is present from the local domain to the remote domain. The presence of such a path is determined by traversing the least common ancestor of the two domains in the CA hierarchy. Principals trust their local CA more than any of the CA's ancestors in the CA hierarchy. Thus, if an ancestor of a CA is compromised, transactions among local principals are not affected, increasing system availability and keeping trust as local as possible.

Figure 3 depicts an example of the arrangement of principals in multiple administrative domains. In this example, Principals $P3$ and $P4$ must establish a path of trust through the root CA to successfully authenticate one another. Demonstrating the principle of locality of trust, Principals $P1$ and $P2$ need only establish a path of trust through their common ancestor one level up to mutually authenticate.

4.4 Time

Since all CRISIS certificates contain timeouts and since these certificates are distributed across the wide area, the system must make assumptions about clock synchronization. Further, CRISIS must protect against security attacks exploiting a node's notion of time. If time on a machine is corrupted, statements can be used beyond their period of validity.

Today, most workstations possess fairly accurate clocks that are periodically synchronized with any of a number of external sources. However, time-sensitive applications (and hence, reference monitors) may require guarantees above and beyond such loose synchronization, for example, that the local clock is periodically synchronized with a trusted external source. Other applications will require an invoice of all assumptions made during a computation in the case where data is corrupted or leaked to determine the exact cause of the corruption, assuring complete accountability.

For CRISIS, we assume the presence of replicated, trusted time servers. Principals producing certificates with timeouts (e.g., CA's and OLA's) contact these servers periodically to obtain signed certificates containing the current time to validate the principal's notion of time. If the principal's time differs by more than a few seconds (i.e., within network delay bounds) from the time supplied by the server, the principal assumes that either the time server or the local operating system/hardware has been compromised (to determine which, a second server might be contacted). Such communication with time servers need not be synchronous, since the time certificates can be cached to prove recent synchronization.

In CRISIS, time certificates are provided to resource managers to prove that a node's notion of time closely matches the value reported by a trusted time server at some recent point in the past. CRISIS identity and transfer certificates report time values (such as expiration time) relative to the value contained in a chained time certificate. While use of time certificates does not guarantee that time-based attacks can be avoided or prevented, it can aid in determining the cause of certain security violations *post-mortem*. Thus, if a security breach is detected, analysis of certificates used to gain unauthorized access can be used to determine the cause of the attack. For example, examination of the certificates may show that a node attempted to use an expired time certificate or that a time server was compromised and reported faulty values of time.

4.5 Authorization

Once a request has been securely transmitted across the wide area, and properly authenticated, the remaining task is *authorization*, determining whether the principal making the request should be granted access. Traditionally, both Access Control Lists (ACLs) and capabilities have been used to describe the set of principals authorized to access a particular object. Since both ACLs and capabilities have advantages in different situations, we use a hybrid model similar to that proposed in [Neuman 1993].

The targets of CRISIS ACLs are service-specific. Currently, file ACLs contain lists of principal's authorized for read, write, or execute access to a particular file. Process execution ACLs are a simple list describing all principals permitted to run jobs on a given node. CA ACLs contains the list of principals authorized to update, modify, or revoke identity certificates.

A process requests access to an object by contacting the object's reference monitor. In CRISIS, reference monitors are implemented on a service-by-service (e.g., file service) basis and form separate modules in the security manager. For example, the WebFS reference monitor is a separate module in the CRISIS security manager.

CRISIS takes a push-based approach to providing credentials for authorization: requesters are responsible for proving they are authorized to access to an object. Thus, principals transmit to reference monitors their request in conjunction with a list of certificates describing their credentials. This list of certificates may simply contain the requester's identity certificate or may contain a more elaborate set of transfer certificates. The alternative to push, a pull-based mechanism where the reference monitor requests necessary credentials from principals, can provide more flexibility; however, it also complicates system design and can reduce performance.

To determine whether a request for a particular operation should be authorized, the reference monitor first verifies that all certificates are signed by a public key with a current endorsement from a trusted CA and OLA. In doing so, the reference monitor checks for a path of trust between its home domain and the domains of all signing principals (as described in Section 4.3). In the common case, the existence of such paths is cached. The reference monitor then checks that none of the timeouts have expired and that time is reported relative to a value stated by a trusted time server (again by checking for a path of trust to the time server).

Once the above steps are taken, the reference monitor is ensured that all certificates are well-formed and valid. Given this knowledge, the reference monitor then reduces all certificates to the identity of single principals. For transfer certificates, this is accomplished by working back through a chain of transfers to the original granting principal. The requesting principal is able to act on the behalf of the reduced list of principals. Finally, the reference monitor checks the reduced list of principals against the contents of the object's ACL, granting authorization if a match is found.

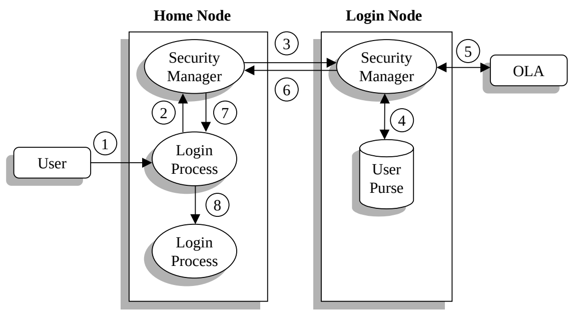

Figure 4: This figure details the steps used in CRISIS to authenticate a principal and authorize the principal for login.

5 CRISIS Protocols

Given the above high level description of the CRISIS architecture, we will now describe how the various system components interact to allow secure execution of routine tasks, including login, file access, and job execution (operations that potentially cross machine and/or administrative boundaries).

5.1 Login

The goal of login is to authenticate a principal to a node and to create a shell process with the principal's privileges. We achieve this by associating a security domain on the login node with a transfer certificate granting the node the privileges of the login role. We assume that each role is associated with a *home domain* and that users wishing to log in must authenticate their identity to their home domain. By minimizing the trust placed in the login node and by choosing a role with an appropriately small set of privileges, we enhance security and reduce the danger of key compromise (private keys never leave the home node). Further, the home domain possesses autonomy in determining the set of sites where principals are allowed to log in. The disadvantage of this approach is that all attempts to authenticate the user must involve the home domain, potentially decreasing system availability. In the future, we plan to investigate using smart cards in place of home domain machines to address this issue; we outline how this scheme would be integrated in the CRISIS architecture at the end of this subsection.

Initially, we consider the following login scenario: a principal accesses a shared workstation by entering a globally unique role name (e.g., *remote_tom@cs.washington.edu*). This role corresponds to the level of trust the principal places in the login node, and to the amount of rights required to successfully complete the desired tasks, for example, reading mail. Once the role is chosen, the principal trusts the OS of the login node with all the privileges associated with that role, since the OS is free to masquerade as the role (at least for the duration of the granted transfer certificate). The login sequence is described in Figure 4 and is summarized below:

1. The principal types in a suitable role name to the login process and enters the password for that role.

2. The login process sends the role name to the local security manager.

3. The security manager at the login node determines the home domain of the specified role (currently explicitly described in the role name) and contacts the security manager at the role's home domain. The two security managers mutually authenticate using SSL and a trusted hierarchy of CA's and online agents.

   As part of this mutual authentication, the login node transmits a certificate signed by a local system administrator stating that the administrator believed that the login node had not been tampered with at boot time. The home node uses this information to aid in the login authorization decision.

4. The home domain uses the password to decrypt the locally stored private key for the specified role name. If the key is successfully decrypted, the home security manager looks up the credentials associated with the specified role in the principal's certificate purse.

5. The certificates are presented to the home domain OLA for endorsement. The OLA sends back the endorsed certificates. The home domain's security manager can optionally update the principal's purse with the endorsements.

6. The home domain signs transfer certificates (on the principal's behalf), transferring all the privileges associated with the specified role to the security manager on the login node.

7. The result of the login request is returned to the login process.

8. If the login is successful, the login process creates a login shell for the user. The security manager creates a new security domain, associating the login shell with the set of certificates transmitted by the home node.

For a successful login, the result of the above sequence of steps is to allow the login node to act on behalf of the role for a time period determined by the home domain's security manager. Any processes spawned by the login shell are by default assigned to the same security domain. The protocols employed to access resources through this security domain are detailed in the next two subsections.

One limitation of the above scheme is that the login node is trusted with the role's password (though not its private key). A well-behaved machine will erase the password from memory as soon as it is transmitted to the home domain. Similarly, the local file and memory cache should be flushed upon logout to ensure that private state is not leaked even if the machine is compromised at a later time. Another limitation with the protocol is that the principal's home domain must be available at the time of login (i.e. no network failures/partitions), or authentication becomes impossible. We believe that both of these limitations are inherent given the current state of hardware/software systems. However, our design also supports the use of specialized, trusted hardware (such as smart cards or a portable computer) to enhance security (keep password from local machine) or higher availability (no need to contact home domain for login) or both.

A trusted hardware proxy, such as a smart card or a portable computer, can also be used to separate the tasks of authentication (principals proving their identity) and authorization (determining that the principal is privileged to login to the remote machine with the specified role)[1]. The proxy can store both a role's private key and the associated password to implement a challenge/response protocol at login as follows[2]. When the home domain is notified of a login attempt, it encrypts a random number in the role's public key and transmits the result to the login machine's security manager. The proxy prompts the user for a password needed to de-

[1]Even if a smart card is used for authentication, it may still be desirable to require joint endorsement of a login session from both the target login machine and the user's home domain. Thus, if remote login is locally authorized, the home domain may disallow the login as a matter of policy. For example, login to a competitor's machine may be disallowed to prevent spoofing attacks.

[2]We present one simple scheme; other zero-knowledge algorithms such as Fiat-Shamir [Fiat & Shamir 1987] could also be utilized.

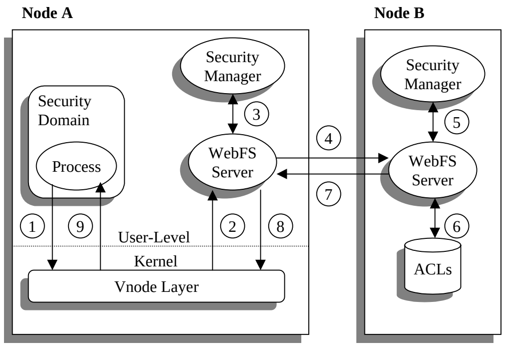

Figure 5: This figure describes the sequence of operations in accessing a remotely stored file through WebFS, a global file system.

crypt a locally stored (but encrypted) private key file for the role. The private key is used to decrypt the number, which is then transmitted back to the home domain. If the correct number is returned, the process of producing proper transfer certificates is followed as enumerated above. As a separate optimization, the task of authorization can be co-located with the hardware proxy to improve both the performance and availability of the login process (while trading off centralized autonomy in authorizing the locations from where particular roles are allowed login).

5.2 Accessing a Remote File

In this section, we demonstrate how the privileges associated with a CRISIS security domain are used to access a remotely stored file. While our techniques are general, we restrict our discussion to our specific implementation environment. We have built a global file system, WebFS [Vahdat et al. 1997] that allows read/write access to files stored across the wide area. WebFS is implemented at the vnode level [Kleiman 1986], similar to other distributed file systems such as NFS [Walsh et al. 1985] or AFS [Howard et al. 1988].

To illustrate the protocol for secure file access, we consider the scenario where a process running on Node A attempts access to a file located on Node B. The example is described in Figure 5, with the individuals steps detailed below:

1. A user process performs an `open` system call on a WebFS file stored on node B (currently WebFS employs a URL-like hierarchy for naming, e.g., /http/B/foo specifies a file foo stored on node B). The kernel translates this call into a `NodeAccess` operation in the Vnode layer.

2. The Vnode layer makes an upcall to a user-level WebFS server to carry out the access request (mode of WebFS functionality is implemented at user-level for ease of debugging and implementation).

3. The WebFS server contacts the local security manager with the requesting `uid/pid` pair to ascertain the privileges associated with the process attempting access to the remote file. For UNIX, The security manager maintains mappings between `uid/pid` pairs and security domains which in turn map to a set of transfer certificates describing the process's privileges.

4. The WebFS server on node A establishes an SSL connection with the WebFS server on node B, transmitting its own credentials and the credentials of the process requesting file access.

5. The WebFS server on node B contacts its local security manager to validate the transmitted certificates and to establish any necessary paths of trust to the potentially remote administrative domain containing node A.

6. Local file ACLs are consulted to determine if the requesting process possesses the request access privileges (e.g. read, write, or execute).

In steps 7-9, the result of the ACL check is returned through the WebFS server on node A, the vnode layer, and finally as the return value to the original `open` system call.

One concern with any system that allows file access from potentially untrusted machines is that local operating systems must be trusted with the contents of the file. That is, a corrupted operating system (or the local CRISIS security manager for that matter) could allow access to unauthorized users on the same host. Worse, if a machine is compromised after a user logs out, sensitive data could still be lost by inspecting the file/virtual memory cache. CRISIS employs two techniques to address these concerns. First, the CRISIS log out process discards the cache of any user accessed files through a WebFS system call. Next, for remote access to highly sensitive data, CRISIS allows the use of trusted portable computers running CRISIS software supporting mobile login. Using this technique, files are transmitted encrypted end-to-end until they reach the portable, at which point they can be decrypted and cached locally with a higher degree of security.

5.3 Running a Remote Job

Conceptually, the process of authenticating and authorizing execution of jobs on remote machines is similar to the process of remote file access. Currently, WebOS uses a UNIX command line program to request remote execution. This request results in a CRISIS library call, which contacts the local process manager with the identity of the principal. The protocol for process execution then proceeds similarly to the file access example described in the previous subsection.

Currently, the ACLs for remote process execution simply include the names of principals which have access to execute programs on a remote node. In the future, we plan to use transfer certificates and ACLs to contain information which specify the portion of the resources a certain role can consume. Another avenue for future work is building an interface to allow principals to reason about the set of privileges required by remote jobs. Clearly, remote jobs should run with the minimum set of privileges necessary to complete their task. However, determining this minimal set can be difficult. We plan to build an interface that allows users to run jobs locally to identify the minimal set of privileges that should be transferred to the job when it is run remotely.

Once a job execution request is authorized, CRISIS uses Janus [Goldberg et al. 1996] to set up a virtual machine to execute the process on the target machine, reducing the risk of violating system integrity. The Janus profile file describing the level of restriction imposed by the virtual machine is generated on the fly based on the identity of the requesting principal and the requirements of the job to be executed. Once set up, the virtual machine is associated with a CRISIS security domain, associating the virtual machine with the set of privileges specified by the principal requesting process execution. By both restricting jobs to originate from authorized users and placing running jobs in a sandbox, the local machine is protected from malicious or buggy programs even if the program's execution is requested from an authorized principal.

6 Performance

To quantify the performance impact introduced by CRISIS, we measured our global file system, WebFS, both with and without CRISIS enhancements. We measure the time required to read and write both 1 byte and 10 MB to a remote file. Measurements were taken between two Sun Ultrasparc 1's connected by a 10 Mb/s switched Ethernet.

Operation	*NFS*	*WebFS*	*WebFS w/CRISIS*
Read 1 byte	3 ms	47 ms	55 ms
Write 1 byte	100 ms	289 ms	340 ms
Read 10 MB	9.8 s	11.0 s	12.2 s
Write 10 MB	9.2 s	12.8 s	14.0 s

Table 1: This table describes the overhead introduced by adding CRISIS security to WebFS, a global file system. CRISIS file transfers are encrypted through SSL.

Table 1 summarizes our results. The first column describes performance for accessing uncached NFS files. The second column describes access to uncached files through a version of WebFS without CRISIS modifications. The added overhead of WebFS relative to NFS is caused by kernel to user-level crossings for cache misses (WebFS network communication code is implemented at the user-level for ease of implementation and debugging). The third column describes performance of WebFS with CRISIS security enhancements. We believe the 10-20% slowdown relative to the baseline WebFS to be acceptable given the added functionality of access control checks and encrypted file transfer.

The measurements in the third column reflect the case where user credentials are cached on the remote node. An additional 175 ms overhead is introduced to establish an SSL connection and 230 ms are required to transfer and cache an identity plus a single transfer certificate if user credentials are not cached remotely. Once again, this total 400 ms overhead is a one-time cost incurred the first time a user makes any access to a remote site (WebFS maintains a "cache" of active SSL connections between machines to avoid the cost of re-establishing an SSL connection for each access). Finally, read access to a cached 1 byte file through WebFS with CRISIS enhancements takes 720 μs, and reading a cached 10 MB file takes 170 ms, values comparable to cached access through NFS. In summary, our security enhancements introduce significant overhead for initial and uncached access because of switching to a user-level process for communication and the overhead of establishing an SSL connection for transmission of certificates. However, the common case read access to a cached file stays entirely in the kernel and provides performance comparable to a file system such as NFS.

7 Related Work

The conceptual framework of our security architecture is largely based on the theory presented in [Lampson et al. 1991]. In the introduction, we discussed the relationship between the DEC security work and our own. In this section, we describe a number of other efforts related to CRISIS.

SDSI [Rivest & Lampson 1996] is a distributed security infrastructure based on public keys with goals similar to our own. Their emphasis is on defining a standard format for certificates, rights transfer, and name spaces to provide a general security framework for Internet applications. With minimal extensions, SDSI could support CRISIS transfer certificates and remote execution of programs. Our work, however, is the largely orthogonal task of defining how such a framework can be used to provide redundant, high performance, and available security mechanisms for applications requiring secure remote control of wide area resources.

Neuman [Neuman 1993] discusses distributed mechanisms for authorization and accounting. Neuman's work has much the same vision as our own, namely limited capabilities in addition to ACL's. His work proposes a more general capability model. However, the capabilities are not auditable because proxies do not carry a chain of transfers. Further, Neuman's work is secret key as opposed to public key, meaning that synchronous communication is required for each transfer of rights. The trusted third party is responsible for recording transfers and transferring the end result. For example, if P_1 transfers rights to P_2, and P_2 further transfers rights to P_3, the trusted third party only passes on P_1 transferring rights to P_3 to any end reference monitors.

Jaeger and Prakash [Jaeger & Prakash 1995] present a model for discretionary access control in a wide area environment. In their work, principals specify the subset of their privileges that are to be transferred to a script written by a potentially untrusted third party. The actual rights transferred are negotiated between the application writer and the user. In their system implementation in Taos [Wobber et al. 1993] (a secure OS based on [Lampson et al. 1991]), they add dynamic principals for running programs with some subset of a principal's privileges, observing the difficulty of creating temporary principals and updating all necessary ACLs with the new principal name. Their dynamic principals are similar to one of the applications of CRISIS transfer certificates.

The goals of the Legion [Wulf et al. 1995] project are similar to our own in WebOS. In Legion, distributed computation takes place in the context of a distributed object system. Their approach to security is orthogonal to our own, with their primary goal being flexibility. Each legion object is able to implement its own security policy. Presumably, a number of base policies will be implemented which will suit the needs of a vast majority of applications. We believe that flexibility in the security system is a desirable feature; our approach in CRISIS can be viewed as one implementation of security for Legion objects.

8 Conclusions

In this paper, we have described the architecture of CRISIS, a security system for wide area applications. In designing CRISIS, we have endeavored to systematically apply principles from related fields to increase system security, availability, and performance across the wide area. These principles include: redundancy, caching, local autonomy, least privilege, and complete accountability. This paper describes how these principles have influenced our design and details the specific protocols used to carry out common operations across the wide area. Relative to earlier efforts, CRISIS uses transfer certificates as a simple mechanism for lightweight creation of roles and capabilities. While the current implementation runs only on Solaris, we expect to port the system to other platforms in the near future.

Acknowledgments

We would like to thank Neal Cardwell, Adam Costello, Terence Spies, and Chad Yoshikawa for participating in the many discussions which lead to our system architecture. We would also like to thank Brian Bershad, Neal Cardwell, Brian Dewey, Robert Grimm, and the anonymous referees for their valuable feedback and suggestions on earlier drafts of this paper.

References

[Birrell et al. 1986] A. Birrell, B. Lampson, R. Needham, and M.Schroeder. “Global authentication without global trust”. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, May 1986.

[Con 1989] Consultation Committee, International Telephone and Telegraph, International Telecommunications Union. *The Directory–Authentication Framework*, 1989. CCITT Recommendation X.509.

[Crispo & Lomas 1996] B. Crispo and M. Lomas. "A Certification Scheme for Electronic Commerce". In *Security Protocols International Workshop*, pp. 19–32, Cambridge UK, April 1996. Springer-Verlag LNCS series vol. 1189.

[Dean et al. 1996] D. Dean, E. Felten, and D. Wallach. "Java Security: From HotJava to Netscape and Beyond". In *Proceedings of the IEEE Symposium on Security and Privacy*, 1996.

[Dig 1995] Digital Equipment Corporation. *Alta Vista*, 1995. `http://www.altavista.digital.com/`.

[Fiat & Shamir 1987] A. Fiat and A. Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In *Advances in Cryptology - Crypto '86*, pp. 186–194. Springer-Verlag, 1987.

[Goldberg et al. 1996] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer. "A Secure Environment for Untrusted Helper Applications". In *Proceedings of the Sixth USENIX Security Symposium*, July 1996.

[Hickman & Elgamal 1995] K. Hickman and T. Elgamal. "The SSL Protocol". In *Internet RFC Draft*, 1995.

[Howard et al. 1988] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. "Scale and Performance in a Distributed File System". *ACM Transactions on Computer Systems*, 6(1):51–82, February 1988.

[Jaeger & Prakash 1995] T. Jaeger and A. Prakash. "Implementation of a Discretionary Access Control Model for Script-based Systems". In *Proc. of the 8th IEEE Computer Security Foundations Workshop*, pp. 70–84, June 1995.

[Kistler & Satyanarayanan 1992] J. J. Kistler and M. Satyanarayanan. "Disconnected Operation in the Coda File System". *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[Kleiman 1986] S. R. Kleiman. "Vnodes: An Architecture For Multiple File System Types in SUN UNIX". In *Proceedings of the 1986 USENIX Summer Technical Conference*, pp. 238–247, 1986.

[Lampson et al. 1991] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. "Authentication in Distributed Systems: Theory and Practice". In *The 13th ACM Symposium on Operating Systems Principles*, pp. 165–182, October 1991.

[Neuman 1993] B. C. Neuman. "Proxy-Based Authorization and Accounting for Distributed Systems". In *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

[Rivest & Lampson 1996] R. L. Rivest and B. Lampson. "SDSI–A Simple Distributed Security Infrastructure". `http://theory.lcs.mit.edu/cis/sdsi.html`, 1996.

[Sirer et al. 1997] E. G. Sirer, S. McDirmid, B. Pandey, and B. N. Bershad. "Kimera: A Java System Architecture". `http://kimera.cs.washington.edu/`, 1997.

[Steiner et al. 1988] J. G. Steiner, B. C. Neuman, and J. I. Schiller. "Kerberos: an authentication service for open network systems". In *Usenix Conference Proceedings*, Dallas, Texas, February 1988.

[Tennenhouse & Wetherall 1996] D. Tennenhouse and D. Wetherall. "Towards an Active Network Architecture". In *ACM SIGCOMM Computer Communication Review*, pp. 5–18, April 1996.

[Vahdat et al. 1997] A. Vahdat, P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin. "WebOS: Operating System Services For Wide Area Applications". UCB Technical Report CSD-97-938, December 1997.

[Walsh et al. 1985] D. Walsh, B. Lyon, G. Sager, J. M. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, and P. Weiss. "Overview of the Sun Network File System". In *Proceedings of the 1985 USENIX Winter Conference*, pp. 117–124, January 1985.

[Wobber et al. 1993] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. "Authentication in the Taos Operating System". In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pp. 256–269, December 1993.

[Wulf et al. 1995] W. A. Wulf, C. Wang, and D. Kienzle. "A New Model of Security for Distributed Systems". University of Virginia CS Technical Report CS-95-34, August 1995.