Decentralized Access Control in Distributed File Systems

STEFAN MILTCHEV and JONATHAN M. SMITH

University of Pennsylvania

VASSILIS PREVELAKIS

Drexel University

ANGELOS KEROMYTIS

Columbia University

and

SOTIRIS IOANNIDIS

Institute of Computer Science (ICS), Foundation for Research and Technology, Hellas (FORTH)

10

The Internet enables global sharing of data across organizational boundaries. Distributed file systems facilitate data sharing in the form of remote file access. However, traditional access control mechanisms used in distributed file systems are intended for machines under common administrative control, and rely on maintaining a centralized database of user identities. They fail to scale to a large user base distributed across multiple organizations. We provide a survey of decentralized access control mechanisms in distributed file systems intended for large scale, in both administrative domains and users. We identify essential properties of such access control mechanisms. We analyze both popular production and experimental distributed file systems in the context of our survey.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Management, Security

Additional Key Words and Phrases: Authentication, authorization, certificates, credentials, decentralized access control, networked file systems, trust management

ACM Reference Format:

Miltchev, S., Smith, J. M., Prevelakis, V., Keromytis, A., and Ioannidis, S. 2008. Decentralized access control in distributed file systems. ACM Comput. Surv. 40, 3, Article 10 (August 2008), 30 pages DOI = 10.1145/1380584. 1380588 http://doi.acm.org/10.1145/1380584.1380588

This work was supported by DARPA and NSF under Contracts F39502-99-1-0512-MOD P0001, CCR-TC-0208972, and CISE-EIA-02-02063.

Corresponding author's address: S. Miltchev, Department of Computer & Information Science, University of Pennsylvania, Levine Hall, 3330 Walnut Street, Philadelphia, PA, 19104-6389; email: miltchev@dsl.cis.upenn.edu.

1. INTRODUCTION

The Internet offers the possibility of global data sharing and collaboration. One class of mechanisms commonly used by organizations is shared data access via file sharing, using remote file access in distributed/networked filesystems. However, most existing systems do not offer secure, scalable, and dynamic cooperation *across organizational boundaries*. When users in distinct administrative domains try to share files, either inefficient and cumbersome exchange of information or compromises in security result.

For example, consider users Alice and Bob, employees of two different companies, who wish to collaborate on a project (see Figure 1). Alice and Bob have at least four approaches to sharing project files:

(1) *ask their system administrators to create accounts in their own administrative domain for each remote user*. This has several problems. First, it imposes an additional administrative burden, which is not scalable with increased users and projects. Often the latency of opening an account for a new user is unacceptable. Second, creating an account for an external user raises escalation of privilege issues. Ideally the user should only be able to use the account for the intended purpose, that is, working on the project files. However, an account could enable an external user to snoop, search for local system vulnerabilities, use up CPU cycles, disk space, *etc.* Because of these problems, company policy typically limits or prohibits the creation of accounts for external users.

(2) *share account passwords*. This approach has serious security implications as it causes lack of accountability and enables escalation of privileges.

(3) *avoid employing an access control mechanism and put the files on the web or anonymous ftp*. This is an unacceptable solution if the content of the files is at all confidential or sensitive.

(4) *exchange files via email or another out of band mechanism*. This is an inefficient way of working as it does not take advantage of any of the safeguards and conveniences that a file system has to offer. In the event that the emails are sent in the clear, there are obvious security concerns. While still not as convenient as a file system, sites like www.filesdirect.com act as a broker between users in different administrative domains and offer better security than unencrypted email. However, such solutions require trust to be placed in a third party.

While more approaches can be imagined, the four listed illustrate the challenges of file sharing across organizational boundaries. This survey examines how access-control mechanisms of different distributed file systems handle file sharing across distinct administrative domains. This survey is restricted to the topic of *access control* in distributed file systems, and largely ignores other design features and tradeoffs, except where they impact access control. It is clear that well engineered systems must pay attention to *many* diverse goals, and the system designer must decide how to weigh different axes of interest during the design phase. As a result, a system that evaluates well here may appear weaker when examined along other important axes. The survey should be interpreted for what it is: an attempt to understand how the choices made by different system designers affect the ability of end users to share information, and control the sharing of that information, using a distributed file system.

The rest of this survey is organized as follows. We establish a framework for comparison in Section 2. Section 3 presents a survey of distributed file systems in our framework. We discuss the results in Section 4 and conclude with Section 5.

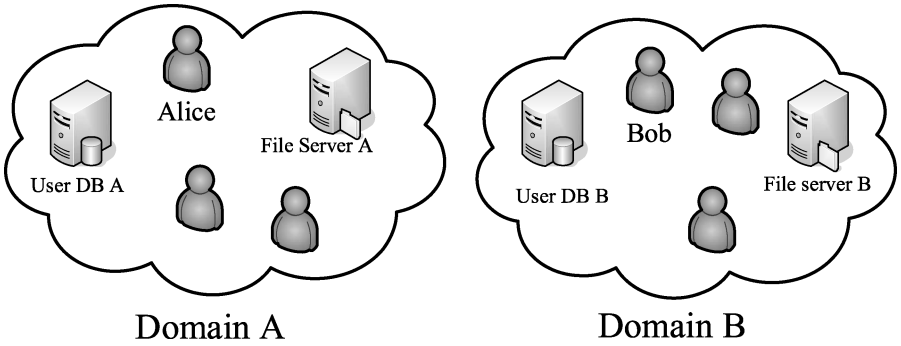

Fig. 1. File sharing across distinct administrative domains. Each administrative domain keeps track of its users in a user account database. Alice cannot grant Bob access to files on file server A because Bob is not listed in domain A's user database.

	File 1	File 2
User X	read	read, write
User Y		read

Fig. 2. An access control matrix.

2. COMPARISON FRAMEWORK

We survey selected distributed file systems to determine their suitability for file sharing across organizational boundaries. To classify the surveyed systems we use the following necessary features as axes of a comparison framework.

(1) *Authentication.* Authentication determines and verifies the identity of a user in the system, that is, providing an answer to the question: "Who is the user?" Traditional authentication mechanisms rely on maintaining a centralized database of user identities, making it difficult to authenticate users in a different administrative domain as depicted in Figure 1. Systems aiming to provide decentralized access control cannot rely on local identification and must employ a decentralized authentication mechanism, or rely on indirect authentication.

(2) *Authorization.* Authorization determines the access rights of a user, that is, it provides an answer to the question: "Is user X allowed to access resource R?" The common way of performing authorization is to look up a user's rights in an access control matrix [Lampson 1971], for example, such as the one depicted in Figure 2. The access control matrix is usually implemented either in the form of access control lists (ACLs) or capabilities.

ACLs correspond to columns of the access control matrix. An ACL is associated with every resource, that is, every object in the file system, and lists all users authorized to access the object along with their access rights. The identity of a user must be known before access rights can be looked up in the ACL. Thus, authorization depends on prior authentication, that is, systems that rely on ACLs for authorization must use a decentralized authentication mechanism to work across administrative boundaries.

Capabilities [Dennis and Van Horn 1966; Levy 1984] correspond to rows of the access control matrix. A capability is an unforgeable token that identifies one or more resources and the access rights granted to the holder of the capability. A user that possesses a capability can access the resources listed in the capability with the specified rights. In contrast to ACLs, capabilities do not require explicit authentication. Capabilities can

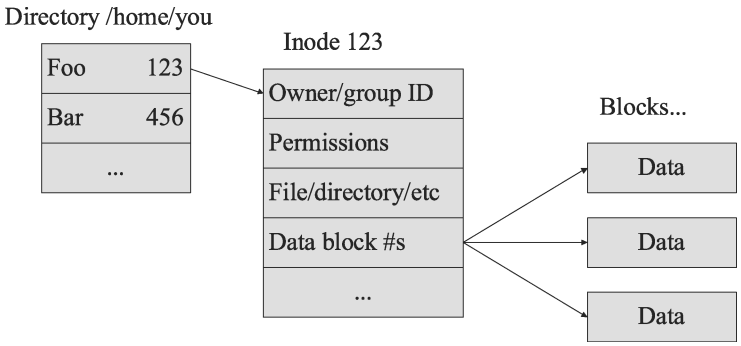

Fig. 3. Simplified structure of the UNIX file system (from [Farmer and Venema 2004]).

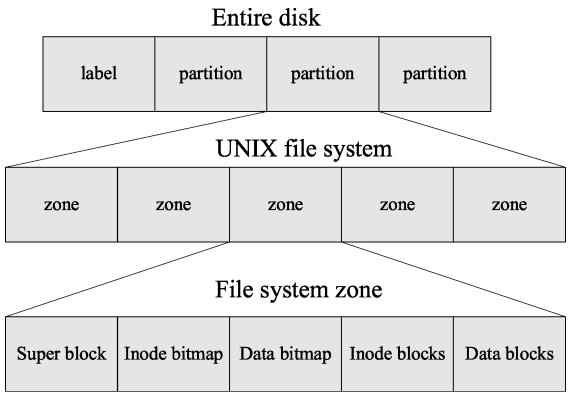

Fig. 4. On-disk layout of a typical UNIX file system (from [Farmer and Venema 2004]).

be transferred among users, which makes them suitable for authorization across organizational boundaries. Because capabilities explicitly list privileges over a resource granted to the holder, they naturally support the property of *least privilege*, an intuitively desirable goal in a system design. However, because possession of a capability conveys access rights, capabilities must be carefully protected from theft, which in a distributed system requires that they be transferred over secure and authenticated channels [Tanenbaum et al. 1986]. In addition, capabilities may make it more difficult to perform later auditing or forensic analysis. Especially for large-scale decentralized systems where the logs themselves or the meaning of the information contained in the capabilities is spread across several system components, collecting all the necessary information involves considerable effort.

(3) *Granularity.* Granularity is the extent to which a system contains discrete components of ever-smaller size. For example, UNIX file systems are organized within a single tree structure underneath one root directory, internal nodes of the tree recursively represent sub-directories of the root, and leaves of the tree can be either files or directories. At a lower layer of abstraction, the same file system consists of inodes and data blocks (Figure 3), and yet another layer lower one can find zones, labels, and partitions (Figure 4).

A distributed file system must strike a balance between extremely coarse-grained and extremely fine-grained authorization. Some systems work at a coarser granularity of higher-level container objects, for example, directories or volumes. While coarser

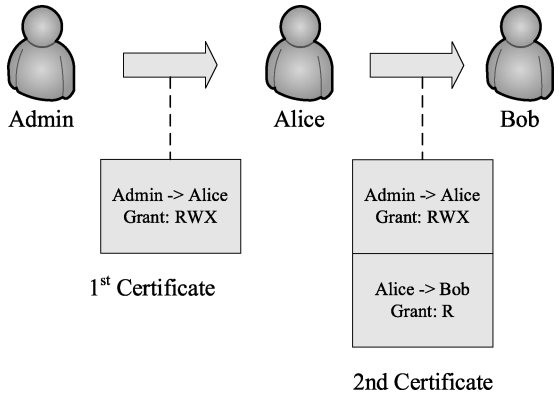

Fig. 5. Delegation of privileges, from an administrator to Alice, and from Alice to Bob. The administrator grants Alice full access by issuing her the first certificate. Alice can then delegate read access to Bob by issuing him the second certificate. To be granted access Bob must present a certificate chain consisting of both certificates.

granularity decreases the amount of access control metadata and the number of access control decisions required, it can make sharing of individual files cumbersome for users. In turn, systems that employ only fine-granularity access control can become difficult to manage, for example, specifying block-level access control when only file-level control is desired. Ideally, the system should allow a flexible level of access control granularity.

(4) *Autonomous delegation.* We evaluate the suitability of file systems for file sharing across organizational boundaries with minimal administrative overhead. A user should be able to delegate access rights to another user, subject to administrative policy. Figure 5 illustrates delegation using authorization certificates. We identify the following requirements for delegation:

—*Autonomy.* To facilitate ease of file sharing, and lower administrative overhead, the delegation mechanism should be user-to-user, that is, no administrator involvement should be required. If delegation is not allowed by default, the administrator will need to be involved in each permission change, becoming a significant bottleneck in large-scale systems. Of course, this need not be a binary condition: for example, unlimited delegation may be allowed between users of the same organization, but explicit administrator approval may be required to delegate to external entities.

—*Accountability.* It should always be possible to determine *who* delegated access to a particular user, at least as part of an auditing (forensics) process.

—*Organizational independence.* A user should be able to delegate his access rights to a user in a different administrative domain, if this is allowed by organizational policy. Furthermore, this should be done while preserving accountability.

—*Low Latency.* A user should be able to access a resource as soon after a delegation as possible.

—*Transitivity.* Delegation chaining should be possible, for example, if Alice delegates access to Bob, Bob should be able to further delegate to Charlie (creating a chain from Alice to Charlie). A mechanism to restrict the right to further delegate and thus limit the length of the delegation chain is also desirable. This allows the system to scale arbitrarily, by pushing administrative responsibility to end users.

—*Fine granularity.* A user should be able to delegate a subset of his access rights, for example, if Alice has read and write access to a file, she should be able to delegate read only access to Bob.

(5) *Revocation.* While the ability to grant access to users in different administrative domains is very desirable, a distributed file system should also have provisions for revoking access. Revocation in systems that base authorization on ACLs is conceptually simpler: a user's access to an object can be revoked by updating the object's ACL to remove access. Capability based systems must rely on timeouts encoded in the capabilities or centralized revocation mechanisms, for example, revocation lists or trusted on-line agents that determine if a capability is still valid. An in-depth evaluation of revocation techniques for a capability based system is presented in Keromytis [2001] and Keromytis and Smith [2007]. There is also a fundamental tension between the requirement for revocation and caching. Once a file has been cached by a temporarily trusted client, the client might allow future accesses even after access to the file has been revoked by the server. The same tension applies also to auditing as the client might allow access to the cached copy without informing the server.

We survey a number of distributed file systems in this comparison framework in the next section, Section 3, and summarize the results in Table I and Table II.

3. DISTRIBUTED FILE SYSTEMS

It is useful to divide systems into *production* and *experimental*, with the split centered on the scale and persistence of deployment, use and experience. A reasonable rule of thumb to designate a system as production would be one which has found wide-spread acceptance with (at least) many thousands of users.

3.1. Production Systems

The initial analysis is an examination of how the access control mechanisms of *production* systems handle file sharing across administrative boundaries. The need to be robust in the face of mission-critical use often forces these systems to be conservative in their design choices. Thus, fundamental considerations like performance, portability, robustness are likely to take precedence over the features that are the focus of this paper. We anticipate that readers of this survey will have used at least some of the file systems presented in this section. Thus, our review of production systems is biased towards the user experience. We review the systems in chronological order.

3.1.1. NFS. The Network File System (NFS) [Sandberg et al. 1985] developed at Sun Microsystems remains one of the most widely used network-attached file systems. Security in NFS appears to have been an afterthought, and global file sharing was not part of the original design. However we choose to review NFS in our framework due to its familiarity and widespread use; it makes an excellent baseline.

The NFS protocol uses the Sun Remote Procedure Call (RPC) [Lyon 1984] mechanism as illustrated in Figure 6. The RPC protocol allows several styles of user authentication, referred to as *authentication flavors*. The original NFS release used weak UNIX-style authentication (user ID and group ID) allowing a user's credentials to be forged (see Figure 7). Support for Diffie-Hellman and Kerberos version 4 authentication flavors was added later, but UNIX style authentication (AUTH_SYS) was the only mandatory flavor, and thus the most commonly implemented. Host authentication is also weak, because it relies on spoofable IP addresses or DNS names.

Authorization in NFS follows UNIX semantics [Thompson 1978]. Thus, access to every file is controlled by the standard UNIX mode bits associated with the file. The permission bits can be viewed as a simple ACL, that lists three principals: the owner of the file, the group associated with the file, and the group consisting of all other users. Thus, we refer to UNIX mode bits as UNIX ACLs throughout the rest of the discussion.

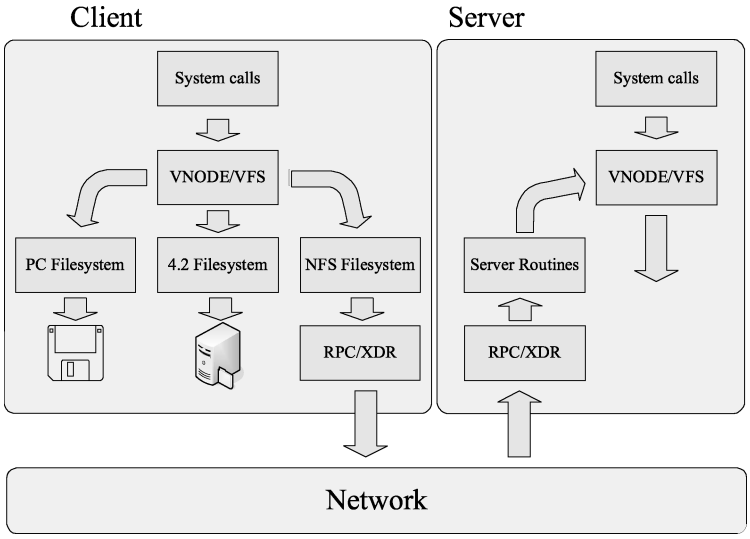

Fig. 6. NFS architecture (from [Sandberg et al. 1985]).

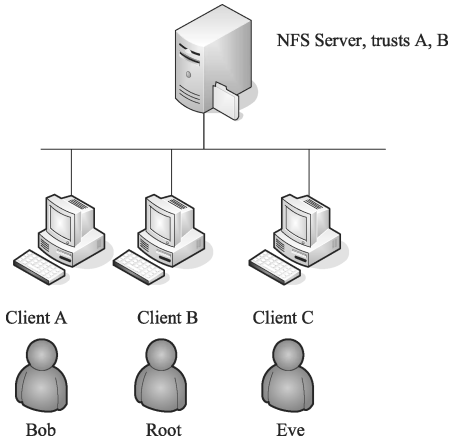

Fig. 7. NFS trust model when using the AUTH_SYS authentication flavor (adopted from Callaghan [2000]). The NFS server trusts client hosts A and B. Access control is enforced by inspecting the source IP address of RPC requests. User Bob can legitimately access his files after authenticating to client A. However, a privileged user on client B (Root) can easily assume the credential of Bob without knowledge of his password. Finally, user Eve on client C can spoof the IP address of client A. Thus, RPC requests from C appear to come from A, and client C is trusted, though it is *not* in the server's access list!

The rights that can be given to each principal are Read, Write and Execute. Before users can access a remote file, privileged administrators must mount the file system where the remote file is located. This is done through the mount protocol [Callaghan et al. 1995], through which file system names are mapped to directory identifiers (handles). The remote server's administrator controls access by listing exported file systems and hosts allowed to mount them. A handle for the top-level directory of an exported file system will be provided to hosts that are allowed to mount that file system. Once that handle is acquired, no further use of the mount protocol is needed. This is another

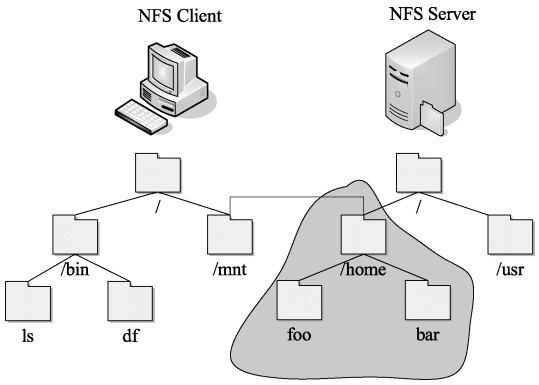

Fig. 8. NFS access control granularity with the (remote) mount protocol. The server exports a file system (e.g., /home) to the client. An administrator on the client mounts the exported file system (e.g., under /mnt). Because the server trusts the client to enforce file-access rights, object-access granularity in NFS is at the file system level.

weakness of the NFS security model: since directory handles do not change often (or at all), revocation of mount privileges cannot be assured.

While initially it appears that the object access granularity in NFS is at the file level, the server actually trusts the client workstation that mounts an exported file system to check file-access rights (see Figures 7 and 8). This security problem was addressed with the introduction of the ACCESS procedure in NFSv3. Because no strong host authentication mechanism is used, security is based merely on matching the IP or DNS name of the client workstation. Because a file cannot be shared without a file system being exported on the server and mounted on the client, object-access granularity in NFS is at the file system level. Once an exported file system has been mounted, the user perceives object access granularity to be at the individual file level.

Significant administrative involvement is required for Alice to share a file with Bob if he resides in a different administrative domain. The administrator of Alice's server must trust Bob's server and export a part of the local file system to it. The administrator of Bob's workstation must trust Alice's server and mount the exported file system. Finally, since access control is performed using UNIX permission bits, Bob must obtain an account in Alice's domain to have a meaningful UNIX user identifier (UID). Thus, autonomous delegation between users in different administrative domains is *not* supported in NFS.

Revocation in NFS is conceptually simple. A server administrator can edit the export list and remove directories or hosts. Administrators can also disable user accounts or edit group definitions in the centrally administered user database. Finally, access to individual files or directories can be revoked by changing the UNIX bit masks associated with them.

In summary, authentication and authorization in early versions of NFS were designed assuming a tightly administered domain (e.g., a single campus LAN or extended LAN), making it unsuitable for global file sharing. This view is reflected in some earlier literature. The creators of the Athena system [Rosenstein et al. 1988; Dyer 1988], which relies on NFS and Kerberos, recognize some of the barriers to access control scalability, and indicate the numerous ACLs in the system were difficult to administer. Further, additional intermediate levels of access between administrators and users were desirable. The authors of the Bones system [Schönwälder and Langendörfer 1993] point out similar problems.

3.1.2. AFS. The Andrew file system (AFS) [Howard et al. 1988; Howard 1988; Satyanarayanan 1989, 1990, 1992] was developed at Carnegie Mellon University as a secure distributed file system with centralized user authentication. The earliest version of AFS was developed concurrently and independently of NFS, but its design was strongly influenced by the need to scale to many thousands of users. This need for scalability drove many aspects of its design, especially those pertaining to performance and security. AFS introduced several improvements in access control mechanisms, for example, richer ACLs and user editable groups. The resulting reduced administrative overhead and improved scalability of access control management, though limited to the local administrative domain, make AFS very relevant to this survey.

Authentication in early versions of AFS was based on a variant of the Needham-Schroeder authentication protocol [Needham and Schroeder 1978]. Users could only share files with other users in the same *cell* (i.e., AFS administrative domain). Cross-cell authentication required users to have an account in each foreign cell where they wished to access files. Later versions of AFS have adopted the Kerberos authentication system [Miller et al. 1987] for purposes of standardization.

Kerberos version 5 [Kohl and Neuman 1993; Linn 1996] is a centralized authentication system based on symmetric-key cryptography. Administrative domains in Kerberos are called realms. An administrator maintains the user database for each realm. A Key Distribution Center (KDC) and Ticket Granting Service (TGS) grant users tickets that allow them to access services on specific hosts in a realm. Because Kerberos relies on a trusted third party and symmetric key cryptography, accessing services across administrative boundaries is not straightforward. Administrators have to set up trust relationships and exchange keys for users to access services in a different realm. While cross-realm authentication has been studied [Trostle et al. 2001; Westerlund and Danielsson 2001], Kerberos does not currently allow for autonomous delegation between users in different administrative domains. A more extensive evaluation of Kerberos for decentralized access control scenarios is presented in Keromytis and Smith [2007].

An AFS cell defined along administrative boundaries corresponds to a Kerberos realm. Cross-realm authentication allows users to share information between their respective cells, without possessing accounts in each cell. However, cross-realm authentication requires administrator involvement, because a local administrator must configure in advance which remote cells should be available to users in the local cell. Thus, AFS does not support autonomous delegation between users in different administrative domains.

Authorization in AFS is based on ACLs associated with directories rather than individual files. Thus, object access granularity is at the directory level. The authors argue that the reduction in state and conceptual simplicity coming from a coarser granularity facilitate scalability. AFS ACLs specify the operations that principals (users or groups) can perform on directories, namely:

—read any file in the directory
—write any file in the directory
—list directory contents
—insert new files in the directory
—delete files from the directory
—lock files in the directory
—administer the directory, that is, modify the ACL

If there is no ACL entry allowing a particular operation, access is denied. AFS ACLs can also specify *negative rights*, that is, explicitly state that a user is *not* allowed to perform

one or more of the operations listed above. When a request for access is evaluated, the entries in the normal rights section of the ACL are examined first. Any permission associated with the user on the negative rights section of the ACL are then subtracted. Thus, in the case of conflicts, negative rights override positive rights. This mechanism facilitates rapid and selective revocation, for example, in cases where a user is a direct or indirect member of groups with access to the object. Using negative rights, the user can be explicitly denied access to the object while the user's group membership information is being updated and propagated, a process that may sometimes take significant time in a large distributed system. AFS also retains the standard UNIX mode bits on files; however, these are not used to enforce access on the server and only have local significance on the user's workstation.

Group names are used in AFS ACLs to identify lists of users with particular access permissions. Users can create and maintain their own protection groups—as opposed to UNIX, where only system administrators can manage */etc/group*. Nesting of protection groups is not allowed, that is, a protection group cannot be a member of another protection group. While user-configurable groups improve the ease of file sharing between users in the same cell they do not address the problem of granting access to users in different administrative domains.

More recent versions of AFS allow users external to the current cell to appear on an ACL, for example, an ACL on a server that is in the "cs.cmu.edu" cell can have an entry giving "bob@cs.ucla.edu" rights on a directory. However, configuring the respective cells to support cross-realm authentication requires administrator involvement.

Revocation in AFS is conceptually simple. Because user accounts are centrally managed, any account can easily be disabled. Any user's access to a directory can be revoked by editing the corresponding ACL. Using groups simplifies revocation considerably—whenever there is a change of membership of a group, the change needs only to be made in the definition of the group and not on each ACL concerned. In addition, negative rights allow for rapid revocation if resolving and updating the user's group membership is expected to take significant time.

The Coda file system [Satyanarayanan et al. 1990; Kistler and Satyanarayanan 1991; Satyanarayanan 2002] is a descendant of AFS developed with the goal of being more resilient to failures. Coda provides high availability through the use of two distinct but complementary mechanisms, *server replication* and *disconnected operation*. However, because the access control model of Coda is based largely on AFS, it faces similar limitations in regard to supporting collaboration between users in different administrative domains.

3.1.3. CIFS. The Common Internet File System (CIFS) [Leach and Perry 1996; SNIA CIFS Technical Work Group 2002; Hertel 2003] is the distributed file system native to the Microsoft Windows family of operating systems, and, due to its ubiquitous nature, of particular interest to this survey. CIFS is not limited to the Windows platform as the Samba project (http://www.samba.org.) offers open source implementations of a server and client for UNIX based platforms. CIFS is based on the Server Message Block (SMB) protocol [Microsoft Corporation 1996] originally developed at IBM in the mid-1980s [IBM Corp. 1984]. In CIFS every server offers a set of resources (directory tree, named pipe, printer) to clients over the network. Whenever a resource is made available (shared) via SMB it is given a share name. Before a user on a client can access a share they must authenticate to the server holding the corresponding resource.

CIFS permits a number of different authentication methods. The SMB protocol defines two security levels: share-level and user-level.

Share-level mode is a form of SMB authentication from the days of early corporate LANs when security was not considered a top priority and PC operating systems (e.g.,

DOS) did not support user-based authentication. Thus, passwords, if used at all, are assigned to shares, not users, and are transmitted in plaintext over the network. Users that know the name of a server and a share, along with the potential password, can gain access to that share. A single share may have multiple passwords assigned, each granting different access rights' for example, one password may grant read-only and another read/write access.

Share-level mode, while still used, is considered deprecated and has been replaced with user-level mode. A server employing user-level security makes use of username/password pairs instead of sharename/password pairs. With user-level security, a user must first authenticate and get a valid user identifier (UID), and then present the UID to gain access to any shares. User-level security can be implemented using a plethora of authentication protocols. It is possible to use anonymous or guest login, plaintext passwords, several challenge-response variations (LanManager (LM), NTLM, NTLMv2), and, in more recent versions, Microsoft's implementation of Kerberos [Swift et al. 2002] or other mechanisms based on the Generic Security Services API (GSS-API) [Linn 1997] and the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) [Baize and Pinkas 1998]. The GSS-API enables source-level portability of applications to different environments by providing callers with a common interface to security services in a generic fashion. Thus, security services can be supported with a range of different underlying mechanisms and technologies.

Authorization in CIFS depends on the authentication level and the underlying file system access control mechanism. In share-level mode authorization is combined with authentication: knowledge of a password grants access to a share. In user-level mode the server could in the best case use ACLs to control file accesses. However, ACLs may not be available on all systems. Because CIFS was designed to work with DOS, OS/2, and Windows systems, the underlying file system on the server could be FAT, FAT32, HPFS or NTFS. While FAT has no concept of file ownership and only supports 6 attribute bits (e.g., the archive, hidden, read-only, and system bits), NTFS offers support for ACLs. Thus, depending on version and the mechanisms supported by the underlying file system, authorization in CIFS can exhibit varying degrees of sophistication: none (when anonymous access is allowed), rudimentary (read-only or read-write access), or more fine grained access control (when ACLs are supported).

Object access granularity in CIFS is at the share level. In a file system context a share is a directory.

Like NFS, CIFS was designed for tightly administered domains and thus does not support all the requirements for autonomous delegation across organizational boundaries. As expected, anonymous and guest access or share-level passwords do not provide accountability or fine granularity of delegation. If user-level security with stronger authentication is used, delegation of access control cannot take place without administrative intervention. Administrators must either create accounts for users outside of the local domain, or deal with establishing complex trust relationships between different domains.

Revocation in CIFS can be accomplished in a number of ways. Sharing of a resource can be turned off. Administrators can disable user accounts. If supported, ACLs on any files may be edited to revoke access at a finer level of granularity.

3.1.4. NFSv4. In an effort to address requirements mandated by the widespread use of the Internet, NFS version 4 [Shepler et al. 2003] proposes many improvements over earlier versions. Stronger security and better suitability to deployment on the Internet are the main design goals. A good overview of NFSv4 and a comparison with older versions is presented in [Pawlowski et al. 2000]. We review the relevant changes in the context of our framework.

NFS is based on, and relies on, the underlying security of ONC (Open Network Computing) RPC [Srinivasan 1995], a remote procedure call framework developed by Sun Microsystems. NFSv4 mandates the use of strong RPC security flavors for authentication (older methods, for example, AUTH_SYS can optionally still be supported). This is achieved by adding a new security flavor based on GSS-API [Linn 1993a; Wray 1993] called RPCSEC_GSS [Eisler et al. 1997]. RPCSEC_GSS encapsulates the GSS-API messaging tokens and acts as a transport for conforming security flavors. Examples of GSS-API implementations include:

—Kerberos version 5

—The Low Infrastructure Public Key (LIPKEY) system [Eisler 2000]. LIPKEY provides an authentication model resembling the Secure Sockets Layer (SSL), that makes it more suitable for use on the Internet. Authentication with LIPKEY is similar to using an HTTPS server with *htaccess*, that is, the server is authenticated with a public key certificate, while the clients authenticate using usernames and passwords. Communication is encrypted with a session key. This scheme relies on passwords being centrally managed at the server, that is, a user cannot delegate access to another user not listed in the centralized password database without administrator involvement. Thus, LIPKEY is not suitable for autonomous delegation between users in different administrative domains.

—The Simple Public-Key GSS-API Mechanism (SPKM) [Adams 1996]. In contrast to Kerberos, SPKM is based on an asymmetric-key infrastructure. SPKM allows both unilateral and mutual authentication to be accomplished without the use of secure timestamps. Thus, out of the existing GSS-API mechanisms, SPKM with both client and server authentication using public keys is most suitable for global file sharing across administrative boundaries. However, the GSS-API decouples authentication and authorization, thus limiting the support for autonomous delegation across administrative domains (see discussion in Section 3.2.9).

The implementation of user and group identifiers also influences the suitability of an authentication mechanism for deployment across the Internet. Earlier NFS versions represented users and groups via 32-bit integers. This is unsuitable for global file sharing, because user and group identifier assignments in different administrative domains are unlikely to agree. NFSv4 uses character strings instead of integers to represent user and group identifiers. Uniqueness can be guaranteed by using a format of *user@domain* or *group@domain* and leveraging the global domain name registry.

Authorization in NFSv4 is enhanced over the UNIX mode bits used by earlier versions with the introduction of support for ACL attributes. NFSv4 ACL support is similar to the Windows NT model [Microsoft Corporation 2005; Swift et al. 2002]. The NFSv4 ACL attribute is an array of access control entries. Access control entries can be one of four types: ALLOW, DENY, AUDIT, or ALARM. The ability to explicitly grant access to users who are not the owner or in the group of a file improves flexibility over standard UNIX ACLs. The ability to explicitly deny access facilitates rapid revocation.

NFSv4 eliminates the mount protocol by using initialized file handles like the public file handle in WebNFS [Callaghan 1996a; Callaghan 1996b] (A WebNFS client uses the special reserved public filehandle as an initial filehandle rather than using the mount protocol). File-access rights as specified in ACLs are checked on the server, not the client. Thus, while the server administrator still exports file systems rather than individual files, object access granularity is at the file level.

While NFSv4 introduces changes that facilitate global file sharing (elimination of the mount protocol, introduction of public file handles, a global user identifier name space), autonomous delegation between users in different administrative domains is still not

possible with the currently supported authentication mechanisms. Kerberos requires administrator involvement for establishing trust relationships between realms, while LIPKEY requires administrator involvement in account creation for the non-local user.

Revocation mechanisms in NFSv4 remain mostly unchanged and involve editing ACLs. Support for more feature-rich ACLs and negative rights in ACLs are the major changes over previous versions.

3.2. Experimental Systems

Our review suggests that widely adopted production systems are evolving from supporting file sharing within a single administrative domain to supporting file sharing between different organizations with a preestablished administrative relationship, often referred to as *federation*. However, production systems fail to address the problem of file sharing between distinct domains with no preexisting administrative trust relationship. In the following section we examine a number of experimental systems and evaluate their support for autonomous ad-hoc sharing between users in different administrative domains. Experimental systems are not as widely adopted as production systems and their maturity can range from simple proof of concept implementations to prototypes tested by a limited user base within a university's computer science department. Thus, our review of the following systems is based on what authors claim can be done, rather than user experience, which can put production systems at a disadvantage. We review the experimental systems in chronological order.

3.2.1. Truffles. Truffles [Reiher et al. 1993] was one of the early systems to recognize and address the need for file sharing between users in different administrative domains. Truffles was built on the replication services provided by the Ficus file system [Guy et al. 1990] and added a mechanism for setting up secure file sharing without administrator intervention. Sharing was at the granularity of a volume, that is, a subset of a local file system.

Truffles used Privacy Enhanced Mail (TIS/PEM) [Linn 1993b; Kent 1993; Balenson 1993; Kaliski 1993] to authenticate users and provide a secure transport channel. Users were identified by public keys bound to X.500 distinguished names in X.509 certificates [CCITT 1989]. Truffles authentication thus relied on a hierarchy of certification authorities (CAs). This limited autonomous delegation, because users from different administrative domains still had to have a common root CA.

Authorization in Truffles relied on standard UNIX and Ficus access control mechanisms, where each file has a standard UNIX ACL associated with it.

Truffles did not address revocation.

3.2.2. Bayou. Bayou [Terry et al. 1995; Petersen et al. 1996] was a replicated, weakly consistent storage system designed for the mobile computing environment. To maximize availability, users could read and write any available replica. The Bayou system used a *primary commit* scheme to resolve conflicts, that is, one server designated as the primary took responsibility for commiting updates. Bayou is relevant to this survey because it was one of the early systems trying to address the problem of enabling autonomous delegation by using an authorization mechanism based on access control certificates instead of ACLs.

Authentication in Bayou was based on public-key cryptography. Every user possessed a public/private key pair and was authenticated by the server using a challenge/response protocol.

Authorization in Bayou was based on digitally signed access control certificates. Three types of certificates were supported:

—*access granting certificates* granted a user access (one of read, write, or server) to a data collection, the unit of replication. In the context of a file system the unit of replication was a directory. The "server" privilege enabled a user to maintain a replica of the data on his workstation or portable computer, i.e., to run a server for the data collection. Access granting certificates were signed by a single trusted signing authority.

—*delegation certificates* delegated a user's privileges from an access control certificate to another user. Delegation certificates had to be signed by the delegating user.

—*revocation certificates* allowed the original signer of a certificate to revoke it. Thus, access-granting certificates were revoked by the signing authority (administrator), while delegation certificates could be revoked by the user that issued them.

As a side note, Bayou required separate certificates for read and write access.

All access-granting certificates in Bayou were signed by a single trusted signing authority. This approach limits autonomous delegation across organizational boundaries, because a user in a different administrative domain might be unknown to the signing authority. The access control model in Bayou provided authorization at the granularity of a whole data collection.

Revocation in Bayou was accomplished using revocation certificates. Revocation certificates were stored by write operations and propagated with the data collections to which they apply. Thus, revocations of write privileges were applied at the primary server, and there was no need to ensure that every other server be notified of the revocation.

3.2.3. xFS. xFS [Anderson et al. 1995], a serverless distributed file system, was developed as part of the UC Berkeley Network of Workstations (NOW) project. Any node in the system could act as both server and client to provide all file system services in a peer-to-peer fashion. The primary concerns of the xFS architects were better performance, scalability, and higher availability than traditional file systems. However, the decentralized architecture of xFS did not carry over to its access control mechanisms. Unfortunately we were not able to find a detailed description of the xFS access control mechanisms in the literature.

The xFS architects describe the system as appropriate for a restricted environment, where machines trust one another's kernels to enforce security; that is, the system was designed to operate within a given admnistrative domain. xFS nodes were split in two categories: trusted core nodes within the admnistrative domain and less trusted client nodes. Trusted nodes ran the standard xFS file sharing protocol and acted as NFS servers to the less trusted client nodes. Because communication with clients outside of the trusted admnistrative domain followed NFS security semantics, xFS was functionally equivalent to NFS for file sharing across organizational boundaries and consequently suffered from the same limitations.

3.2.4. WebFS. WebFS was part of the WebOS [Vahdat 1998] project at UC Berkeley. The system's authorization mechanism was based on a combination of ACLs and authorization certificates. Thus, it is of interest to examine whether such a hybrid approach can exploit the advantages of both mechanisms while minimizing the disadvantages.

WebFS was a global file system layered on top of the HTTP protocol. This approach allows access to files through the file system using existing URLs as file names. The security architecture for WebOS was called CRISIS [Belani et al. 1998]. Authentication in CRISIS was based on X.509 certificates [CCITT 1989; Polk et al. 2002; Housley et al. 2002].

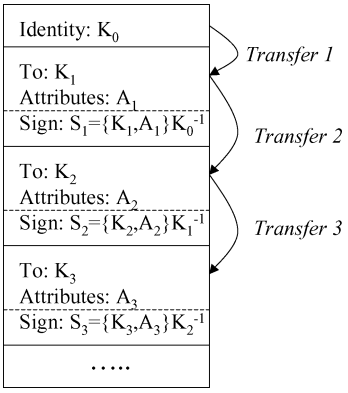

Fig. 9. Structure of a CRISIS transfer certificate (from Vahdat [1998]). The transfer certificate is a chain of X.509 certificates. The first certificate is an identity certificate identifying the principal wishing to make the transfer by his public key, K_0. In each subsequent certificate the issuer transfers a subset of his available privileges to another principal. For example, in the first transfer K_0 delegates privileges described by A_1 to K_1 and signs the certificate with his private key, K_0^1. Certificates can be arbitrarily chained, for example, in this example K_1 transfers privileges to K_2, who in turn transfers privileges to K_3.

Authorization in CRISIS used a hybrid model to best exploit the tradeoffs between ACLs and capabilities. Principals that should have long-term access to an object were listed on the ACL for that object. In the case of WebFS, each file had an associated list of users authorized to read, write or execute. The principals listed on an ACL could then further delegate a subset of their rights to an object by creating *transfer certificates*, short-lived and revocable capabilities. Transfer certificates were encoded in X.509 format, digitally signed and could be chained. Figure 9 shows the structure of a CRISIS transfer certificate.

Object access granularity in WebFS was at the file level. Autonomous delegation in WebFS was limited since users could only delegate to users who had a certificate from a CA trusted by the local domain. Because WebFS relied on a hierarchy of certification authorities, users in different administrative domains still had to have a common root CA to share files.

CRISIS had good support for revocation. If a principal was listed on an object's ACL his access could be revoked simply by modifying the ACL. When access was granted with certificates, revocation relied on timeouts. Each certificate was first signed by the principal making a statement with a longer timeout. The certificate was then countersigned by a principal of the signer's choosing. The countersignature was issued with a shorter timeout. The countersigner acted as a locally trusted online agent (OLA). The OLA checked if a certificate had been revoked before refreshing its countersignature with a new short timeout. While the CRISIS approach allows for shorter timeouts, it also introduces the need for trusted online agents.

3.2.5. Self-Certifying File System (SFS). SFS [Mazieres et al. 1999; Mazieres 2000; Fu et al. 2002] was a global decentralized file system. SFS is relevant to our survey as its major stated goal was to free SFS clients from any notion of administrative realm, making inter-realm file sharing trivial. To accomplish this goal SFS introduced the notion of *self-certifying pathnames*—file names that effectively contain the appropriate remote server's public key (see Figure 10). Thus, SFS needed no separate key management machinery to communicate securely with file servers. By convention, SFS files could be accessed under **/sfs/Location/HostID/Path**, where **Location** is the DNS name or IP address of the server, **HostID** specifies the server's public key, and **Path** is the path to

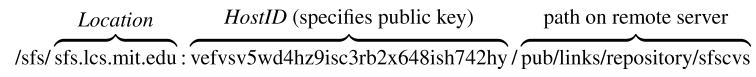

Fig. 10. SFS self-certifying pathname (from [Mazieres et al. 1999]).

the file on the server. The resulting file names were difficult to remember due to the embedded cryptographic information, so symbolic links had to be used as a mnemonic aid.

SFS separated user authentication from the file system by removing key management from the file system. Users in SFS were authenticated using public key cryptography. On the client an agent program with access to the user's private keys was used to authenticate the user to a separate authentication server on the remote server. The authentication server maintained a database mapping public keys to UNIX credentials (a user ID and a list of group IDs). If a user did not have an account on a file server, the server defaulted to anonymous access.

Object access granularity in SFS was at the file level. Object access control in SFS was similar to NFS. Authorization was performed by matching the UNIX credentials returned by the authentication server with standard UNIX ACLs associated with each file.

Autonomous delegation in SFS was not supported because users must have an account on the authentication server trusted by the file server. This would not necessarily be the case for users in different administrative domains. GSFS, a further development of SFS, tried to overcome this limitation and is covered later in this survey.

Revocation of a user's access in SFS was simple. Because the authentication server hosts a centralized user database, the user's entry in the database could be easily removed/disabled. A user could also be removed from groups that appear on ACLs for files he was no longer supposed to access. The authors also describe mechanisms for revoking self-certifying pathnames using revocation certificates, should a server's private key be compromised. As an alternative, a user's agent could also request HostID blocking from the client. The second approach could be useful when no signed revocation certificate is found, but access restriction is still desirable, for example, due to system policy.

3.2.6. OceanStore. OceanStore [Kubiatowicz et al. 2000] was a proposed architecture for global-scale persistent storage. Pond [Rhea et al. 2003] was the OceanStore prototype containing many of the features of a complete system. The primary design goals of the architecture were high reliability and scalability to billions of users. The system relied upon an overlay network named Tapestry [Zhao et al. 2001; Hildrum et al. 2002] for decentralized object location and routing. This allowed the Oceanstore designers to defer many access-control decisions to the overlay. While cryptographic mechanisms were used to deal with Byzantine failures (which would affect reliability) some of the access control issues that would be addressed by a conventional file system were addressed by participation or non-participation in the overlay (e.g., authentication) while other issues were addressed more conventionally (e.g., storage and access of blocks and files by the file system itself).

Authentication of clients in OceanStore was based on public key cryptography.

The access control model of the Bayou system inspired the designers of OceanStore to adopt an asymmetric authorization model with regard to *reader* and *writer* restrictions: reads were restricted at clients via key distribution, while writes were restricted at servers by ignoring unauthorized updates. Files were encrypted and the encryption key was distributed to users with read permission. A file was located in the system using its *globally unique identifier* (GUID). The GUID is computed as the secure hash of the owner's public key and some human-readable name. The owner could choose an

ACL for the object. Write access was enforced at servers by verifying all write requests against the respective object's ACL.

The granularity of sharing in OceanStore was at the file level.

The overview of the OceanStore architecture and the description of the Pond prototype give no indication how delegation would be accomplished in the system. As described, delegation granularity is coarse, that is, limited to distinguishing between read and write access. For read access, delegation would be accomplished by communicating the key used to encrypt the file. While this provides autonomy, there are difficulties with accountability and the need to rekey and redistribute the key to all legitimate users if revocation becomes necessary. Write access is controlled with ACLs, and delegation in this case is subject to the same limitations as other ACL based systems. The authors briefly hint at the possibility of using a trust-management system such as PolicyMaker [Blaze et al. 1996] for expressing richer access control policies, but no details are given.

Revocation in OceanStore would be handled differently, depending on whether it is read or write access that needs to be revoked. To revoke read permission, the owner must request that replicas be deleted or reencrypted with a new key. However, old data from cached copies could still be available to revoked readers. To revoke write access, the owner of an object could modify the ACL for the object. Because all writes must be signed, servers can verify requests against the ACL. While the access control mechanism of Tapestry is not specified, revocation could possibly also be accomplished by blacklisting users so that they can no longer participate in the overlay network.

3.2.7. CapaFS. CapaFS [Regan and Jensen 2001] used self-certifying file names as sparse capabilities to control access to files by users in different administrative domains. CapaFS dispensed with user identifiers altogether, thus eliminating the need to resolve the identities of remote users locally. By relying solely on knowledge of the capability file name for access control, CapaFS aimed to provide autonomous delegation across organizational boundaries.

A capability file name consisted of two parts: a client part used by the client to locate the remote server and a server part used by the server to find the file in local storage. The client part contained the hostname and port of the server. The server part contained the local path name and access rights on the server and was encrypted to protect it from tampering. However, the resulting capability file names were long and meaningless to users, and necessitated the use of symbolic links to assign meaningful names to remote files.

There was no explicit user authentication in CapaFS: knowledge of the filename was sufficient to obtain access to a file. Authorization was based on the access rights encoded in the server part of the capability file name. Object access granularity was at the file level.

Because there was no local user identification in CapaFS, autonomous delegation was easily achieved. To share a file, a user needed only to communicate the file name to another user. Thus, no system administrator involvement was required. However, there were a number of problems with the original CapaFS. Because knowledge of the file name provided access to the file, communicating file names to other users had to be done over a secure and authenticated channel (however, no infrastructure for that was developed as part of the system). The original CapaFS was also vulnerable to a man-in-the-middle attack because there was no server authentication. The authors suggested implementing server authentication by adding the server's public key to the capability filename. Because no client authentication was performed, there was no accountability in the original CapaFS; that is, there was no way of telling which particular user accessed a file. This made auditing impossible in CapaFS. The authors describe a way of adding client authentication by adding a client's public key to the server part of the

capability file names. The proposed approach allowed for delegation to specific users by including their public keys as an extension of the capability file name. However, there was no way for a user to delegate only a subset of his access rights to another user, for example, a user possessing a read/write capability file name could not delegate read-only access to another user.

Revocation in CapaFS could be achieved by having the server keep a capability revocation list (CRL) of all capability file names that have been revoked. This approach is unlikely to scale well as the list grows with time. Because the user's public key was not included in the capability filename, the original CapaFS design did not support revoking access on a per-user basis. Another approach to revocation suggested by the authors was to limit the lifetime of a capability file name by including a timeout in it.

3.2.8. Fileteller. FILETELLER [Ioannidis et al. 2002] was a credential-based distributed file storage system with provisions for paying for file storage and getting paid when others access files. FILETELLER was developed by a subset of the authors of this paper. Users used a micropayments system to pay for both the initial creation of files and any subsequent accesses to network-based storage. FILETELLER illustrates the use of *trust management credentials* for *both* access control and payment resulting in an elegant and scalable architecture that works across organizational boundaries. Trust management [Blaze et al. 1996; Blaze et al. 1999a] eliminates the need for ACLs by incorporating access control in a new kind of certificate, namely an *authorization certificate* or *credential*. Such a credential directly authorizes an action rather than dividing the authorization task into authentication and access control. Unlike traditional credentials, which bind keys to principals, trust-management credentials bind keys to the authorization to perform certain tasks.

Authentication in FILETELLER was based on public keys. There were three participants in the system: *Network Users* (NUs), *Network Storage Providers* (NSPs), and *Check Guarantors* (CGs). All participants were identified by their public keys. A network user had to to authenticate with the storage provider before any file operation could take place. The authentication protocol provided strong authentication and, optionally, let the user piggy-back credential delivery to the NSP. Security protocols such as IPsec [Kent and Atkinson 1998] or TLS [Dierks and Allen 1999] could be configured to meet these requirements.

Authorization in FILETELLER was based on KeyNote [Blaze et al. 1999b] trust management credentials. A network user held one or more credentials issued by a check guarantor indicating the user's credit line with the CG, as shown in Figure 11. CGs played a role similar to that of PKI CAs, sharing many of the deployment and operational limitations. There were four kinds of credentials used in different parts of the system:

(1) Check Guarantor credentials, which specified a user's line of credit.
(2) Microchecks, which authorized a payment from a network user to an NSP, or to another NU.
(3) Server credentials, issued by the check guarantors, that identified complying storage providers the network users can use.
(4) File-access credentials, initially issued by NSPs when a file is created, authorizing subsequent access to that file by the owner. File owners could then issue further file-access credentials, delegating access to other NUs.

Granularity of access in FILETELLER was at the file level; that is, users were able to create, read, delete, append to, or replace whole files. Whole files were prefered to individual blocks for two reasons: to amortize the cost of a check verification over the

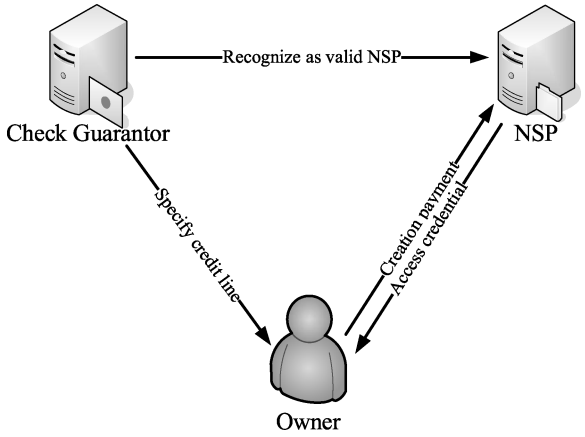

Fig. 11. Network Storage Providers (NSPs) issue a KeyNote credential to each Check Guarantor (CG) authorizing them to act as introducers of users, by in turn issuing them credentials. A file owner needs to convince a CG to provide them with a credit line, also expressed as a KeyNote credential. The file owner needs to provide these two credentials to the NSP, along with a microcheck conveying payment to the storage provider. In response, the NSP returns to the file owner a KeyNote access credential, granting her full privileges in accessing the file.

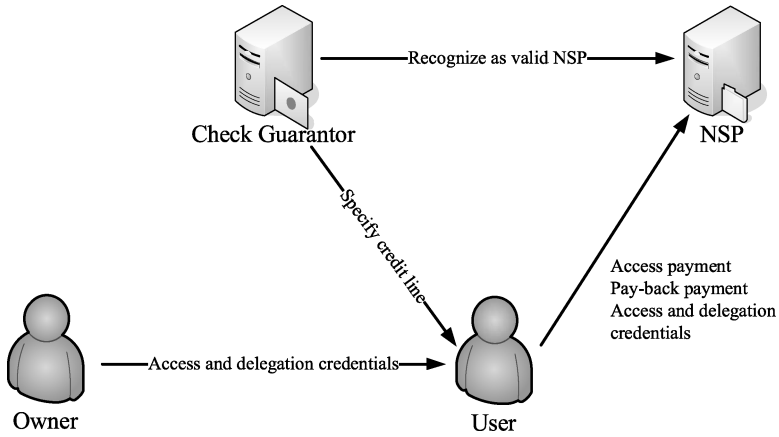

Fig. 12. A user wishing to access another user's file needs to have their own line of credit with a Check Guarantor (CG), as well as a credential from the file owner granting them access to that file. When accessing the file, the user needs to provide the credit-backing credential from the CG, a microcheck to the NSP, and the access credential(s) to the file. If the owner has set a "pay-back" disposition for the file, an additional microcheck to the owner may also be needed to gain access.

transfer of an entire file, and to avoid choosing some arbitrary block size and defining block-level operations, which would tie FILETELLER to a particular file system philosophy rather than make it a general file-storage service.

Autonomous delegation across organizational boundaries was supported in FILETELLER as shown in Figure 12. A network user with access to a file could delegate a subset of their access rights to another NU by issuing a file-access credential. This delegation mechanism is transitive and does not require administrator involvement. Users did not have to reside in the same administrative domain; however, a user wishing to access a file served by a given storage provider had to establish a line of credit with a CG that recognized the NSP as valid. Because users were vouched for by

```
KeyNote-Version: 2
authorizer: "<Administrator's Public Key>"
licensees: "<Alice's Public Key>"
conditions: (app_domain == "DisCFS") &&
            (HANDLE == "discfs://discfs.cis.upenn.edu/Makefile.stefgjxg")
             -> "RWX";
signature:
          "<Signature by Administrator>"
```

Fig. 13. Credential granting user *Alice* (as identified by her public key, in the *Licensees* field) access to file *Makefile.stefgjxg* on host *discfs.cis.upenn.edu*. The 1024-bit keys and signatures in hex encoding have been omitted in the interest of readability.

a CG and uniquely identified by their public keys, accountability was preserved. File attributes were used in file-access credentials to allow fine-granularity delegation. These attributes were metadata associated with the file by the owner, and could be used to implement easy file grouping, associate security labels with files, or for any other similar scheme. For example, a user could associate arbitrary textual tags with each file, similar to the way popular Websites allow the tagging of digital photos and video clips; access control credentials could then use such tags as part of the access control decision.

Revocation in FILETELLER was time-based and relied on credential expiration. As with previous work on which FILETELLER was based [Blaze et al. 2001], CG credentials issued to users were relatively short-lived, avoiding the need for credential revocation lists. Other revocation mechanisms could also be used with FILETELLER, as specified on a per-credential basis.

3.2.9. DisCFS. The Distributed Credential File System (DisCFS) [Miltchev et al. 2003] was developed by the authors of this paper with the explicit goal of allowing access to remote users not known in advance to the file server. Thus, DisCFS directly addressed the problem focused upon by this survey. DisCFS, like FILETELLER used KeyNote trust management credentials [Blaze et al. 1999b] to identify: (*1*) files being stored; (*2*) users; and (*3*) conditions under which their file access is allowed. An example credential is shown in Figure 13.

Users in DisCFS were identified by their corresponding public keys. Authorization in DisCFS was based on trust-management credentials. Trust-management credentials contain the identity (i.e., public key) of the user authorizing an action, and the identity of the user authorized to perform the action (respectively, the *authorizer* and *licensee* fields in Figure 13).

When a user wished to access a remote file, the software on the client's workstation sent the relevant credentials with a request to access the file on behalf of the user. The file server passed the credentials along with a query to the KeyNote system. KeyNote checked the signatures on all credentials, evaluated whether the conditions specified in the credentials were met and returned an answer to the query. If the query was successful, the file server granted the user access to the file. As part of this exchange, the server had to verify that a user was the legitimate owner of the public key present in the *licensee* field of the credential(s) she presented, that is, that the user had knowledge of the corresponding private key. In DisCFS, this was accomplished by establishing an IPsec connection between the client workstation and the file server, using the Internet Key Exchange (IKE) [Harkins and Carrel 1998] protocol. File sharing then took place over this IPsec association.

DisCFS controled access at the file level, however trust-management credentials could also be applied at a coarser granularity if system requirements favored a minimization of state over fine-grained control.

DisCFS had full support for autonomous delegation between users in different administrative domains. If Alice has been granted access to a file, she possesses a credential specifying her access rights (e.g., the one depicted in Figure 13). If she wishes to delegate a subset of these access rights to Bob, Alice can create a new credential identifying her as the *authorizer*, Bob as the *licensee*, and specifying Bob's access rights in the *conditions* field. Alice must then sign the new credential and send it to Bob along with her original credential. When Bob requests access to the file, he must present the *credential chain* consisting of both credentials. This mechanism provides autonomy and organizational independence: no administrator involvement is necessary, and Bob does not have to be a member of the same administrative domain as Alice. Because each user could act as a CA in DisCFS, the need for higher-level certification authorities was eliminated. Credentials were signed to prevent tampering and could be sent in the clear or posted on the web (of course, this is not a good idea in environments where privacy of file-access rights is desirable). DisCFS provided good delegation latency: users could begin accessing files as soon as they were issued a credential.

DisCFS supported multilevel delegation. That is, if Alice delegates access to Bob, he can then further delegate to Charlie by creating a new credential. It was also possible to limit delegation to one hop. Trust-management credentials allow for fine granularity of delegation: users can delegate any subset of their rights. The trust management engine ensures that there is no rights amplification, that is, if Alice is granted read access to a file and issues Bob a credential granting read/write access, Bob will not be able to write to the file using the credential.

Delegation in DisCFS preserved accountability, because the public keys corresponding to each authorizer and licensee were included in the credentials.

Revocation in DisCFS was not as straightforward as in ACL-based systems, because it was not always evident who had access to a resource. In a multilevel delegation chain, a user is only aware of the next "hop," for example, if Alice delegates access to Bob, and Bob delegates access to Charlie, Alice has no knowledge of Charlie, and thus no way to revoke his access. Thus, DisCFS relied on timeouts in credentials to limit their useful life. As a more user-centric system, DisCFS made a tradeoff and avoided the administrative overhead of running on-line agents for revocation checks at the expense of having to use longer timeouts.

3.2.10. WebDAVA. WebDAVA [Levine et al. 2003] was a web file sharing service designed specifically for users in distinct administrative domains. WebDAVA was developed by a subset of the authors of this paper. The system provided *file transfer* rather than *file-access* services, that is, files had to be transferred in their entirety between server and client, rather than being manipulated in place. Thus, WebDAVA was not, strictly speaking, a distributed file system. However, we examine it as another example of a system using authorization credentials to allow access across organizational boundaries.

Authentication in WebDAVA was performed using a challenge-response protocol. When the server received a file request it responded with a challenge containing a nonce and the server's public key. The client response included the user's public key, the file-access credential, and a newly created *nonce credential* signed with the user's private key. While the protocol details are somewhat vague, it appears only the client was being authenticated. It is possible that the server was authenticated by other means, for example, using TLS [Dierks and Allen 1999].

Authorization in WebDAVA was handled by KeyNote [Blaze et al. 1999b] trust-management credentials. The credentials authorized desired actions corresponding to the HTTP GET or PUT methods. Downloading a file from the server was done via the HTTP GET method. The PUT method allowed file creation or modifying a stored file

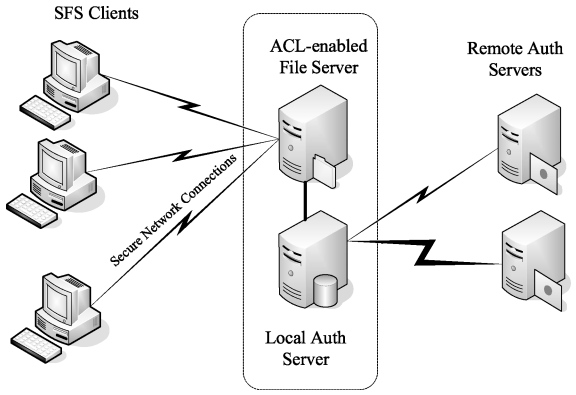

Fig. 14. Overview of the GSFS authentication architecture (from Kaminsky et al. [2003]).

by overwriting it. Deleting a file was done by saving an empty file; the server notices that the file is empty and removes it. Granularity of access control in WebDAVA was at the file level.

WebDAVA had full support for autonomous delegation between users in distinct administrative domains. Users could delegate a subset of their access to any other users by retrieving their public keys and issuing them a credential. Credentials were protected from tampering by a signature and thus could be sent over email or downloaded from the Web. For example, when Alice wants to allow Charlie to access a file stored on the WebDAVA server, she needs to retrieve Charlie's public key, construct the credential delegating access to Charlie's key, and then send this credential along with her own access credentials to Charlie. Charlie must import these credentials and use them to access the file. While Alice may use any mechanism to get Charlie's key, WebDAVA provides a key-server that stores the keys of the various users to simplify credential management. To transfer a credential to Charlie, Alice simply selects the credential and enters Charlie's email address. The WebDAVA client then creates an email message using these credentials and sends it to Charlie. Charlie can use these credentials to download the file from the server. No administrator involvement is required, and Charlie need not have any kind of relationship with the WebDAVA server in order to download the files.

Revocation in WebDAVA was handled by credential expiration and certificate revocation lists. Each file in the system had an associated file that stored hashes of revoked credentials and thus acted as a CRL. Credentials were passed on to the KeyNote compliance checker for evaluation only if their hash was not found in the revocation file. The original issuer of a credential could revoke it by uploading it to the CRL using the PUT method.

3.2.11. GSFS. GSFS [Kaminsky et al. 2003], a further development of SFS, is of particular interest to this survey as it was conceived with the explicit goal of allowing file sharing between users in different administrative domains. GSFS tried to achieve this goal with an access control mechanism based on ACLs.

Authentication in GSFS was based on public keys, similar to SFS. However, to facilitate global file sharing, the authentication server was modified to contact servers in other administrative domains and retrieve remote user and group definitions (see Figure 14). For the purposes of this discussion we define remote users to be users outside of the local administrative domain. Remote authentication servers were referenced

with self-certifying hostnames, similar to file servers. A GSFS authentication server had to contact the remote authentication servers of any remote users or groups listed as members of local groups. Because of network latency and failures, it is not feasible to do this at the time an authentication request is made. Thus, the GSFS designers traded off freshness for availability by having the authentication server periodically (e.g., every hour) contact the remote authentication servers of any remote users or groups listed in local group definitions. This introduced a delay between when a decision to grant access had been made and when the actual access could occur.

Authorization in GSFS was done using ACLs. The ACLs were similar to those used in AFS, but were extended to differentiate between files and directories. Access rights available in GSFS ACLs included the right to modify the ACL itself. GSFS ACLs could list four different kinds of principals: local user names, local group names, public key hashes, and anonymous entries. Public key hashes were the only way of listing a remote principal directly on a GSFS ACL. Remote groups could not be listed directly on the ACL, but could be included indirectly by making them a member of a local group.

As with SFS, object access granularity in GSFS was at the file level.

There are two scenarios for autonomous delegation in GSFS. In the first scenario, user Alice may choose to share a file with user Bob in a different administrative domain by listing a hash of Bob's public key on the ACL of the file (assuming that Alice has the right to modify the ACL of the file). However, if Bob wants to then further share access to the file with another user, Bob must also be given the right to modify the ACL of the file. As delegation chains grow longer, this approach will lead to longer and harder to manage ACLs on the fileserver. It is also impossible to allow fine-grained multilevel delegation, for example, if Alice gives Bob read access to the file and wishes him to be able to delegate that access, she must also give him the right to modify the ACL. However, in this case there is nothing to prevent Bob from modifying the ACL and granting himself write access. Thus, this approach is only suitable for limited one-hop delegation from a local user to a remote user.

In the second scenario, Alice can create a local group (e.g., alice.friends) and list remote users (e.g., Bob) or groups from another administrative domain (e.g., friends@otherdomain) as members of the local group. This assumes that there is a remote authentication server for the domain that Bob or Alice's other friends belong to. Remote groups can in turn contain other groups and the nesting can be arbitrarily deep. Thus, indirection through authentication servers can provide delegation. In contrast to public key hashes, multilevel delegation can be achieved, for example, if Alice allows access to a group owned by Bob, then Bob can add new members (which can be other groups) to the group. However, this approach still makes it difficult for a principal to delegate only a subset of his access rights. For example, if Alice has allowed members of the group managed by Bob read/write access, Bob cannot delegate read-only access to Charlie.

Listing a public key hash directly has several advantages over using group or user names:

—*Latency*—because the user record does not have to be pulled from a remote authentication server, the user can begin accessing files immediately.

—*Simplicity*—users in a different administrative domain need not be associated with an authentication server.

—*Privacy*—public-key hashes offer a degree of privacy by obfuscating the usernames on a group membership list. Because anyone can query an authentication server and usernames could correspond to e-mail addresses, group membership lists could be harvested for purposes of sending unsolicited bulk electronic mail ("SPAM").

Table I. File System Classification

	Status[1]	Authentication	Authorization	Granularity	Autonomous Delegation	Revocation
NFS	P	AUTH_SYS, Kerberos	ACL (UNIX)	File system	No	ACL
NFSv4	P	Kerberos, LIPKEY, SPKM	ACL (NT)	File	No	ACL
AFS & Coda	P	Kerberos	ACL (AFS)	Directory	No	ACL
CIFS	P	Plaintext password, Challenge-Response, Kerberos	ACL	Directory	No	ACL
xFS	E	AUTH_SYS, Kerberos	ACL (UNIX)	File system	No	ACL
Truffles	E	Public Key (X.509)	ACL (UNIX)	Volume	Limited	No
Bayou	E	Public Key	AC Certificate	Data Collection	Limited	Revocation certificate
WebFS	E	Public Key (X.509)	Hybrid	File	Limited	ACL, CRL, OLA,[2] Certificate Expiration
CapaFS	E	No	Capability	File	Limited	CRL, Timeout
SFS	E	Public Key	ACL (UNIX)	File	No	ACL, CRL
GSFS	E	Public Key	ACL (SFS)	File	Limited	ACL, CRL
DisCFS	E	Public Key	Trust Mgmt. Credential	File	Yes	Credential Expiration
WebDAVA	E	Challenge-Response	Trust Mgmt. Credential	File	Yes	CRL, Credential Expiration
Fileteller	E	Public Key	Trust Mgmt. Credential	File	Yes	Credential Expiration

[1]Production (P) or experimental (E) file system.
[2]locally trusted on-line agent.

Group and usernames on the other hand offer the following advantages over public key hashes:

—*Indirection* allows for multi-level delegation. The remote authentication servers also provide a single point of update if a user needs to change his key or revoke it.

—*Naming*—names are easier for users to keep track of than hashes and thus would improve accountability and scalability.

Beyond the mechanisms for revocation available for SFS, GSFS had to handle revocation involving remote users and groups. Thus, revocation in GSFS was closely related to freshness. If a remote user changed his key or was removed from a remote group record, it would take an update cycle for the change to be reflected on the local authentication server. On the other hand, access granted to public-key hashes in GSFS could be instantly revoked by editing the ACL or group record.

Table II. Autonomous Delegation Support in Distributed File Systems

	Autonomy	Organizational Independence	Low Latency	Transitivity	Fine Granularity	Accountability
Truffles	•		•	•		•
Bayou	•		•	•	•	•
WebFS	•		•	•	•	•
CapaFS	•	•	•	•		•[1]
Fileteller	•	•	•	•	•	•
DisCFS	•	•	•	•	•	•
WebDAVA	•	•	•	•	•	•
GSFS[2]	•	•	•		•	•
GSFS[3]	•[4]	•		•		•

[1]Only if user's public key is included in capability filename.
[2]Public key hashes of remote users listed on ACL.
[3]Remote groups listed on ACL.
[4]Remote users must be associated with remote authentication server.

4. DISCUSSION

Table I classifies the file systems studied in Section 3 within the framework defined in Section 2. Systems that were not designed for file sharing across organizational boundaries (NFS, AFS, xFS, CIFS, SFS) require substantial administrator involvement for merging realms or account creation. The inability to list non-local users using ACLs in NFS, AFS, xFS, CIFS and SFS makes it impossible for these systems to support autonomous delegation across organizational boundaries.

The remaining systems reviewed in Section 3 exhibit varying degrees of support for autonomous delegation. We present a more detailed comparison in Table II.

OBSERVATION 1. *Systems that support autonomous delegation across organizational boundaries use public-key cryptography for authentication.*

It is hardly surprising that public-key cryptography is used as a building block for the authentication mechanism employed by systems that need to scale beyond the local administrative domain. Public-key cryptography eliminates the need for synchronous communication with a trusted third party. The public keys of every host and user can be freely distributed. Knowledge of the respective public keys allows two principals to establish a secure communication channel without external administrative involvement. Of the systems supporting autonomous delegation, CapaFS is the only one that does not employ public key cryptography (in the original design).

OBSERVATION 2. *Mechanisms based on pure capabilities cannot provide accountability.*

Systems based on pure capabilities like CapaFS exhibit a high degree of user autonomy. However, our review of CapaFS revealed that if the capabilities are not tied to user identities in any way, it is impossible to meet the accountability requirement for delegation. In addition, exchanging capabilities becomes problematic, because their content should not be disclosed to third parties. The CapaFS authors recognize the problems of using capabilities with no ties to user identities, however the proposed solution does not meet the requirement for fine-grained delegation.

OBSERVATION 3. *Mechanisms based solely on ACLs do not scale well to a user base distributed across organizational boundaries.*

The difficulty of supporting autonomous delegation in GSFS best exemplifies this observation. GSFS tries to address the problem of global file sharing using ACLs. However GSFS offers only limited support for delegation. If public-key hashes are used to identify non-local users, the formulated requirement of multilevel delegation is not met. If groups are used instead, multilevel delegation is possible, however the requirement for fine-grained delegation is not met. This illustrates the difficulty of using an ACL-based authorization mechanism when the users are distributed in different administrative domains.

OBSERVATION 4. *Authorization certificates come closest to fulfilling all requirements for autonomous delegation across organizational boundaries.*

Bayou, WebFS, DisCFS, WebDAVA and Fileteller meet most of the requirements for autonomous delegation. These systems rely on some form of authorization certificates: access granting and delegation certificates, transfer certificates, or trust-management credentials. Transitivity of delegation is achieved by chaining the certificates. Successive links in a delegation chain can only refine, and never expand, the access rights of the original certificate. This ensures that the fine granularity requirement for delegation is met. By supporting both transitive and fine-grained delegation, the systems based on authorization certificates distinguish themselves from systems based on ACLs that tend to support either transitive or fine-grained delegation, but not both.

OBSERVATION 5. *There is a tradeoff between user autonomy and ease of revocation.*

Systems based on ACLs (e.g., GSFS) do not provide full support for autonomous delegation. However, access to an object can be revoked by simply editing that object's ACL.

Some systems based on authorization certificates or ACL/authorization certificate hybrid schemes (e.g., Bayou, WebFS) make provisions for delegation. These systems require users in different administrative domains to have a common root CA. While this limits the users' organizational independence, it also makes revocation easier, since only a limited number of CAs must be contacted to update CRLs.

In DisCFS and WebDAVA, a more user-centric approach is taken. Users act as CAs and sign trust-management credentials they issue themselves. Thus, delegation in these systems has the highest degree of user autonomy. However, because access control is completely decentralized, revocation must rely on certificate expiration or online revocation authorities.

5. CONCLUSIONS

This survey provided a new framework for analyzing the suitability of distributed file system access-control mechanisms to the challenge of supporting global file sharing across organizational boundaries. We identified authentication, authorization, granularity, autonomous delegation, and revocation as necessary features of a system aiming to address this challenge. Thus, these features formed the axes of the comparison framework we used to survey selected systems.

While the focus of the survey has purposely been on distributed file system design, the framework might prove useful in understanding the tradeoffs inherent in global access to any form of shared data. The central concerns of scalability and ease of use pervade much of system design and evaluation. While file systems provide naming and persistence, concerns of access control for a networked shared memory (with more dynamic state) would require access control as well. Overlay solutions (of which Web-based file systems can be seen as an example) exist at least in part to overcome administrative

inertia. For example, port 80 is left open through most Internet firewalls to accommodate user web browsing, and therefore file-sharing can overcome administrative resistance simply by accessing data using HTTP. If the issue is achieving global "user-controlled" access in a secure manner, it seems more effective to address this problem directly rather than employing a workaround. Our analysis, summarized in Tables I and II, suggests how the problem might be addressed effectively.

Systems based on authorization certificates generally provide better support for autonomous delegation of access rights between users in different administrative domains, compared to systems based on ACLs or pure capabilities. Therefore they are attractive from the perspective of scalability, but there are some operational and ease-of-use concerns, in addition to the problem indicated by Table I: all of these systems are experimental and do not enjoy widespread use.

The major operational concern is revocation. Authorization certificates resemble capabilities, in that revocation is a challenge. Many systems attempt to achieve revocation semantics with artificial means such as "keep-alives" or timeouts, which are inelegant and introduce a window of risk: a certificate might have unwarranted access until a certain expiration time is reached. Future research on revocation of authorization certificates should seek to minimize the existing tradeoff between user autonomy and ease of revocation.

The major ease-of-use concern is the management of the Public-Key Infrastructure (PKI) and certificates required for users to access data. Widespread PKI deployment would greatly ease the deployment of systems that use trust management techniques, and a focus on ease-of-use would ensure that users have tools (or automated management systems) that make the use of certificates for remote access to be transparent and seamless, relative to local access of data. However, we must acknowledge the significant logistical difficulties in building, deploying, and operating a real-world PKI.

Both the revocation and ease-of-use concerns would be most effectively addressed by moving at least one file system from experimental to production status.

REFERENCES

ADAMS, C. 1996. The simple public-key GSS-API mechanism (SPKM). RFC (Proposed Standard) 2025, Bell-Northern Research.

ANDERSON, T. E., DAHLIN, M. D., NEEFE, J. M., PATTERSON, D. A., ROSELLI, D. S., AND WANG, R. Y. 1995. Serverless network file systems. In *Proceedings of the 15th Symposium on Operating Systems Principles*.

BAIZE, E. AND PINKAS, D. 1998. The simple and protected GSS-API negotiation mechanism. RFC (Proposed Standard) 2478, Bull.

BALENSON, D. 1993. Privacy enhancement for Internet electronic mail: Part III: Algorithms, modes, and identifiers. RFC (Proposed Standard) 1423, IAB IRTF PSRG, IETF PEM WG.

BELANI, E., VAHDAT, A., ANDERSON, T., AND DAHLIN, M. 1998. The CRISIS wide area security architecture. In *Proceedings of the USENIX Security Symposium*. 15–30.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. 1999a. The role of trust management in distributed systems security. In *Secure Internet Programming*. Lecture Notes in Computer Science, vol. 1603. Springer-Verlag Inc., Berlin, Germany, 185–210.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. D. 1999b. The KeyNote trust management system version 2. RFC (Proposed Standard) 2074, AT&T Labs - Research.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized trust management. In *Proceedings of the 17th IEEE Symposium on Security and Privacy*. Oakland, CA, 164–173.

BLAZE, M., IOANNIDIS, J., AND KEROMYTIS, A. D. 2001. Offline micropayments without trusted hardware. In *Proceedings of the 5th International Conference on Financial Cryptography*.

CALLAGHAN, B. 1996a. WebNFS client specification. RFC (Proposed Standard) 2054, Sun Microsystems, Inc.

CALLAGHAN, B. 1996b. WebNFS server specification. RFC (Proposed Standard) 2055, Sun Microsystems, Inc. October.

CALLAGHAN, B. 2000. *NFS Illustrated*. Addison-Wesley.

CALLAGHAN, B., PAWLOWSKI, B., AND STAUBACH, P. 1995. NFS version 3 protocol specification. RFC (Proposed Standard) 1813, Sun Microsystems, Inc.

CCITT. 1989. *X.509: The Directory Authentication Framework*. International Telecommunications Union.

DENNIS, J. B. AND VAN HORN, E. C. 1966. Programming semantics for multiprogrammed computations. *Comm. ACM 9*, 3, 143–155.

DIERKS, T. AND ALLEN, C. 1999. The TLS protocol version 1.0. RFC (Proposed Standard) 2246, Internet Engineering Task Force.

DYER, S. P. 1988. The Hesiod name server. In *Proceedings of the USENIX Winter Technical Conference*. 183–190.

EISLER, M. 2000. LIPKEY—A low infrastructure public key mechanism using SPKM. RFC (Proposed Standard) 2847, Zambeel.

EISLER, M., CHIU, A., AND LING, L. 1997. RPCSEC_GSS protocol specification. RFC (Proposed Standard) 2203.

FARMER, D. AND VENEMA, W. 2004. *Forensic Discovery*. Addison Wesley Professional Publishing.

FU, K., KAASHOEK, M. F., AND MAZIÈRES, D. 2002. Fast and secure distributed read-only file system. *Comput. Syst. 20*, 1, 1–24.

GUY, R. G., HEIDEMANN, J. S., MAK, W., PAGE, JR., T. W., POPEK, G. J., AND ROTHMEIR, D. 1990. Implementation of the Ficus replicated file system. In *Proceedings of the Summer USENIX Conference*. 63–71.

HARKINS, D. AND CARREL, D. 1998. The Internet key exchange (IKE). RFC (Proposed Standard) 2409, Internet Engineering Task Force.

HERTEL, C. R. 2003. *Implementing CIFS: The Common Internet File System*. Prentice Hall.

HILDRUM, K., KUBIATOWICZ, J., RAO, S., AND ZHAO, B. 2002. Distributed object location in a dynamic network. In *Proceedings of the ACM Symposium on Algorithms and Architecture (SPAA)*. 41–52.

HOUSLEY, R., POLK, W., FORD, W., AND SOLO, D. 2002. Internet X.509 public key infrastructure: Certificate and certificate revocation list (CRL) profile. RFC (Proposed Standard) 3280.

HOWARD, J., KAZAR, M., MENEES, S., NICHOLS, D., SATYANARAYANAN, M., SIDEBOTHAM, R., AND WEST, M. 1988. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1, 51–81.

HOWARD, J. H. 1988. An overview of the Andrew file system. In *Proceedings of the USENIX Winter Technical Conference*. Dallas, TX, 213–216.

IBM CORP. 1984. *IBM PC Network Technical Reference Manual, No.6322916*, 1st Ed.

IOANNIDIS, J., IOANNIDIS, S., KEROMYTIS, A., AND PREVELAKIS, V. 2002. Fileteller: Paying and getting paid for file storage. In *Proceedings of the 6th International Conference on Financial Cryptography*.

SCHÖNWÄLDER, J. AND LANGENDÖRFER, H. 1993. Administration of large distributed UNIX LANs with BONES. In *Proceedings of the World Conference On Tools and Techniques for System Administration, Networking, and Security*.

KALISKI, B. 1993. Privacy enhancement for Internet electronic mail: Part IV: Key certification and related services. RFC (Proposed Standard) 1424, RSA Laboratories.

KAMINSKY, M., SAVVIDES, G., MAZIÈRES, D., AND KAASHOEK, M. F. 2003. Decentralized user authentication in a global file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*. Bolton Landing, NY, 60–73.

KENT, S. 1993. Privacy enhancement for Internet electronic mail: Part II: Certificate-based key management. RFC (Proposed Standard) 1422, IAB IRTF PSRG, IETF PEM WG.

KENT, S. AND ATKINSON, R. 1998. Security architecture for the Internet protocol. RFC (Proposed Standard) 2401, Internet Engineering Task Force.

KEROMYTIS, A. D. 2001. STRONGMAN: A scalable solution to trust management in networks. Ph.D. thesis, University of Pennsylvania.

KEROMYTIS, A. D. AND SMITH, J. M. 2007. Requirements for scalable access control and security management architectures. *ACM Trans. Intern. Techn. 7*, 2, Article 8.

KISTLER, J. J. AND SATYANARAYANAN, M. 1991. Disconnected operation in the Coda file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*. Vol. 25. ACM Press, 213–225.

KOHL, J. AND NEUMAN, C. 1993. The Kerberos network authentication service (V5). RFC (Proposed Standard) 1510.

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. 2000. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

LAMPSON, B. 1971. Protection. In *Proceedings of the 5th Princeton Conference on Information Science and Systems*. 437–443.

LEACH, P. AND PERRY, D. 1996. CIFS: A common Internet file system. *Microsoft Internet Developer*.

LEVINE, A., PREVELAKIS, V., IOANNIDIS, J., IOANNIDIS, S., AND KEROMYTIS, A. D. 2003. Webdava: An administrator-free approach to web file-sharing. In *Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETIC), Workshop on Distributed and Mobile Collaboration*. Linz, Austria, 59–64.

LEVY, H. M. 1984. *Capability-Based Computer Systems*. Butterworth-Heinemann, Newton, MA.

LINN, J. 1993a. Generic security service application program interface. RFC (Proposed Standard) 1508, Geer Zolot Associates.

LINN, J. 1993b. Privacy enhancement for Internet electronic mail: Part I: Message encryption and authentication procedures. RFC (Proposed Standard) 1421, IAB IRTF PSRG, IETF PEM WG.

LINN, J. 1996. The Kerberos version 5 GSS-API mechanism. RFC (Proposed Standard) 1964, OpenVision Technologies.

LINN, J. 1997. Generic security service application program interface, version 2. RFC (Proposed Standard) 2078, Internet Engineering Task Force.

LYON, B. 1984. Sun remote procedure call specification. Tech. rep., Sun Microsystems, Inc.

MAZIERES, D. 2000. Self-certifying file system. Ph.D. thesis, MIT, Cambridge, MA.

MAZIERES, D., KAMINSKY, M., KAASHOEK, M. F., AND WITCHEL, E. 1999. Separating key management from file system security. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*. 124–139.

MICROSOFT CORPORATION. 1996. Microsoft networks SMB file sharing protocol (document version 6.0p).

MICROSOFT CORPORATION. 2005. Microsoft access control model. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/access_control_model.asp.

MILLER, S. P., NEUMAN, B. C., SCHILLER, J. I., AND SALTZER, J. H. 1987. Kerberos authentication and authorization system. Tech. rep., MIT, Cambridge, MA.

MILTCHEV, S., PREVELAKIS, V., IOANNIDIS, S., IOANNIDIS, J., KEROMYTIS, A., AND SMITH, J. 2003. Secure and flexible global file sharing. In *Proceedings of the Annual USENIX Technical Conference, Freenix Track*. 165–178.

NEEDHAM, R. M. AND SCHROEDER, M. D. 1978. Using encryption for authentication in large networks of computers. *Comm. ACM 21*, 12, 993–999.

PAWLOWSKI, B., SHEPLER, S., BEAME, C., CALLAGHAN, B., EISLER, M., NOVECK, D., ROBINSON, D., AND THURLOW, R. 2000. The NFS version 4 protocol. In *Proceedings of 2nd International System Administration and Networking Conference (SANE)*.

PETERSEN, K., SPREITZER, M., TERRY, D., AND THEIMER, M. 1996. Bayou: Replicated database services for world-wide applications. In *Proceedings of the 7th ACM SIGOPS European Workshop*.

POLK, W., HOUSLEY, R., AND BASSHAM, L. 2002. Algorithms and identifiers for the Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC (Proposed Standard) 3279.

REGAN, J. AND JENSEN, C. 2001. Capability file names: Separating authorization from user management in an Internet file system. In *Proceedings of the USENIX Security Symposium*. 211–233.

REIHER, P., PAGE, T., CROCKER, S., COOK, J., AND POPEK, G. 1993. Truffles—a secure service for widespread file sharing. In *Proceedings of the Privacy and Security Research Group Workshop on Network and Distributed System Security*.

RHEA, S., EATON, P., GEELS, D., WEATHERSPOON, H., ZHAO, B., AND KUBIATOWICZ, J. 2003. Pond: The OceanStore prototype. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. USENIX.

ROSENSTEIN, M. A., JR., D. E. G., AND LEVINE, P. J. 1988. The Athena service management system. In *Proceedings of the Winter USENIX Conference*. 203–212.

SAMBA PROJECT. Samba. http://www.samba.org.

SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D., AND LYON, B. 1985. Design and implementation of the Sun network file system. In *Proceedings of the Summer USENIX Conference*.

SATYANARAYANAN, M. 1989. Integrating security in a large distributed system. *ACM Trans. Comput. Syst. 7*, 3, 247–280.

SATYANARAYANAN, M. 1990. Scalable, secure, and highly available distributed file access. *Computer 23*, 5, 9–18, 20–21.

SATYANARAYANAN, M. 1992. The influence of scale on distributed file system design. *IEEE Trans. Softw. Engin. 18*, 1, 1–8.

SATYANARAYANAN, M. 2002. The evolution of Coda. *ACM Trans. Comput. Syst. 20*, 2, 85–124.

SATYANARAYANAN, M., KISTLER, J. J., KUMAR, P., OKASAKI, M. E., SIEGEL, E. H., AND STEERE, D. C. 1990. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Comput. 39*, 4, 447–459.

SHEPLER, S., CALLAGHAN, B., ROBINSON, D., THURLOW, R., BEAME, C., EISLER, M., AND NOVECK, D. 2003. Network file system (NFS) version 4 protocol. RFC (Proposed Standard) 3050.

SNIA CIFS TECHNICAL WORK GROUP. 2002. Common Internet file system (CIFS) technical reference. SNIA technical proposal, Storage Networking Industry Association.

SRINIVASAN, R. 1995. RPC: Remote procedure call protocol specification version 2. RFC (Proposed Standard) 1831, Sun Microsystems.

SWIFT, M., TROSTLE, J., AND BREZAK, J. 2002. Microsoft Windows 2000 Kerberos change password and set password protocols. RFC (Proposed Standard) 3244, University of Washington, Cisco Systems, and Microsoft.

SWIFT, M. M., BRUNDRETT, P., DYKE, C. V., GARG, P., HOPKINS, A., CHAN, S., GOERTZEL, M., AND JENSENWORTH, G. 2002. Improving the granularity of access control for windows 2000. *ACM Trans. Inform. Syst. Secur. 5*, 4.

TANENBAUM, A., MULLENDER, S., AND VAN RENESSE, R. 1986. Using sparse capabilities in a distributed operating system. In *Proceedings of the 6th International Conference on Distributed Computing Systems*. 558–563.

TERRY, D., THEIMER, M., PETERSEN, K., DEMERS, A., SPREITZER, M., AND HAUSER, C. 1995. Managing update conflicts in Bayou, a weakly connected storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*.

THOMPSON, K. L. 1978. UNIX implementation. *Bell Syst. Tech. J. 57*, 6, Part 2, 1931–1946.

TROSTLE, J. T., KOSINOVSKY, I., AND SWIFT, M. M. 2001. Implementation of crossrealm referral handling in the MIT Kerberos client. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

VAHDAT, A. 1998. Operating system services for wide-area applications. Ph.D. thesis, University of California, Berkeley.

WESTERLUND, A. AND DANIELSSON, J. 2001. Heimdal and Windows 2000 Kerberos: How to get them to play together. In *Proceedings of the USENIX Annual Technical Conference, Freenix Track*. 267–272.

WRAY, J. 1993. Generic security service API : C-bindings. RFC (Proposed Standard) 1509, Digital Equipment Corporation.

ZHAO, B. Y., KUBIATOWICZ, J. D., AND JOSEPH, A. D. 2001. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. rep. UCB/CSD-01-1141, University of California, Berkeley.

Received April 2006; revised February 2007, September 2007; accepted November 2007