January 2006

Decentralized Access Control in Networked File Systems

Stefan Miltchev
University of Pennsylvania, miltchev@seas.upenn.edu

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

Vassilis Prevelakis
Drexel University

Angelos Keromytis
Columbia University

Sotiris Ioannidis
Stevens Institute of Technology

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-06-02.

Decentralized Access Control in Networked File Systems

Abstract
The Internet enables global sharing of data across organizational boundaries. Traditional access control mechanisms are intended for one or a small number of 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. This survey provides a taxonomy of decentralized access control mechanisms intended for large scale, in both administrative domains and users. We identify essential properties of such access control mechanisms. We analyze popular networked file systems in the context of our taxonomy.

Keywords
authentication, authorization, certificates, credentials, decentralized access control, networked file systems, trust management

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-06-02.

Decentralized Access Control in Networked File Systems

STEFAN MILTCHEV and JONATHAN M. SMITH
University of Pennsylvania
and
VASSILIS PREVELAKIS
Drexel University
and
ANGELOS KEROMYTIS
Columbia University
and
SOTIRIS IOANNIDIS
Stevens Institute of Technology

The Internet enables global sharing of data across organizational boundaries. Traditional access control mechanisms are intended for one or a small number of 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. This survey provides a taxonomy of decentralized access control mechanisms intended for large scale, in both administrative domains and users. We identify essential properties of such access control mechanisms. We analyze popular networked file systems in the context of our taxonomy.

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

1. INTRODUCTION

The Internet offers the possibility of global data sharing and collaboration. To that end, one commonly used class of mechanisms is data shared via file sharing, in the form of distributed/networked filesystems. However, most existing systems do not offer secure,

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

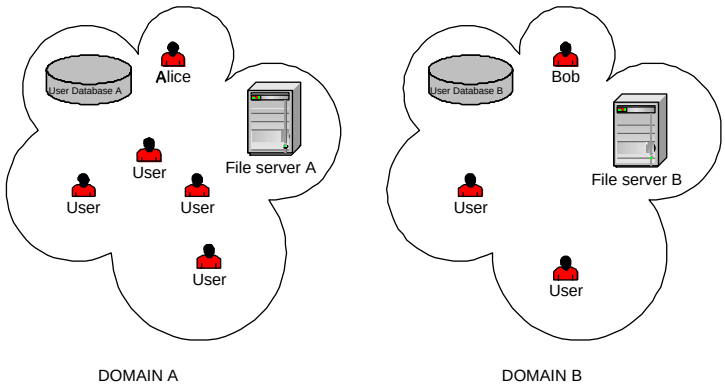

Fig. 1. File sharing across distinct administrative domains. Each administrative domain keeps track of its users in a user account database. Alice cannot grant Bob access to files on file server A because Bob is not listed in domain A's user database.

scalable and dynamic cooperation across organizational boundaries. When users in distinct administrative domains try to share files, they are faced with either inefficient and cumbersome exchange of information or compromises in security.

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 with which to share 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 as we increase the number of 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, *i.e.*, 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 even remotely sensitive.

(4) e-mail the files back and forth. 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 e-mails are sent in the clear, there are obvious security concerns as well.

While more approaches can be imagined, the four listed illustrate the problem of file sharing across organizational boundaries. This survey examines how access control mechanisms of different networked file systems handle file sharing across distinct administrative

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

Fig. 2. An Access Control Matrix

domains.

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

2. COMPARISON FRAMEWORK

We survey a number of networked file systems with the aim of determining their suitability for file sharing across organizational boundaries. To classify the surveyed systems we use the following properties to define a comparison framework:

(1) **Authentication:** determines and verifies the identity of a principal in the system, *i.e.,* 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:** determines the access rights of a principal, *i.e.,* it provides an answer to the question: "Is user X allowed to access resource R?" The common way of performing authorization is to lookup a principal's rights in an access control matrix [Lampson 1971], *e.g.,* 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, *i.e.,* 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, *i.e.,* 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, because the possession of the capability implicitly authenticates the user[1]. Capabilities can be transferred among users, which makes them suitable for authorization across organizational boundaries. However,

[1] To contrast ACLs and capabilities consider the real life analogy of controlling access to a building. A doorman might grant a visitor access after checking that his name appears on a visitors list. This identity approach is similar to using ACLs to control access to files. In contrast a tenant might gain access to the building by using the key that was issued to him upon signing a lease. The key is similar to a capability that is not explicitly tied to an identity. The key can be conveniently shared, however care must be taken that it does not fall into the wrong hands.

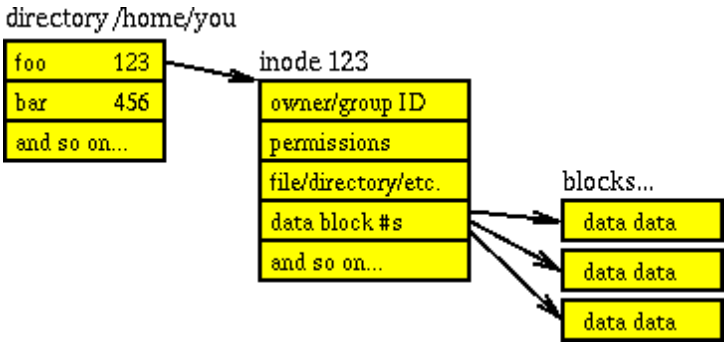

Fig. 3. Simplified structure of the UNIX file system (from [Farmer and Venema 2004]).

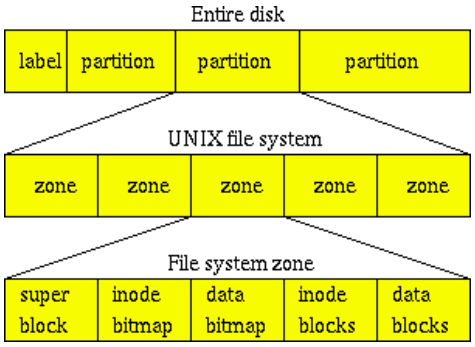

Fig. 4. On-disk layout of a typical UNIX file system (from [Farmer and Venema 2004]).

because possession of a capability conveys access rights, capabilities must be protected from theft, which requires that they be transferred over secure and authenticated channels [Tanenbaum et al. 1986].

(3) **Granularity:** is the extent to which a system contains discrete components of ever-smaller size. *E.g.* 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 network file system must strike a balance between too coarse-grained and too fine-grained authorization. Some systems work at a coarser granularity of higher-level container objects, *e.g.*, directories or volumes. While coarser granularity decreases the amount of access control meta-data 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 can become difficult to manage, *e.g.* having to specify block-level access controls if only file-level control is desired can quickly become infeasible. Ideally, the system should allow a frlexible level of access control granularity.

(4) **Autonomous delegation:** We evaluate the suitability of file systems for file sharing

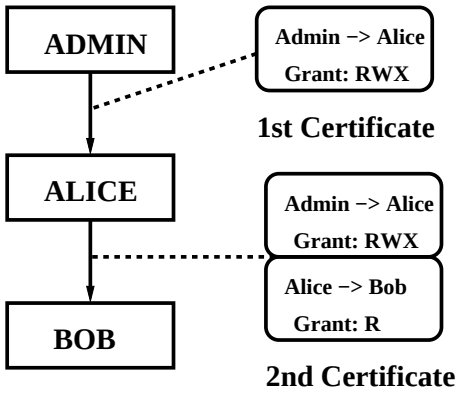

Fig. 5. Delegation of privileges, from an administrator to Alice, and from Alice to Bob. The administrator grants Alice full access by issuing her the first certificate. Alice can then delegate read access to Bob by issuing him the second certificate. To be granted access Bob must present a certificate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

across organizational boundaries with minimal administrative overhead. A user should be able to delegate access rights to another user. 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, *i.e.*, no administrator involvement should be required.

—**Accountability** It should always be possible to determine who delegated access to a particular user.

—**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, *e.g.*, if Alice delegates access to Bob, Bob should be able to further delegate to Carl (creating a chain from Alice to Carl). 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, *e.g.*, 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 network 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 editing the object's ACL. Capability based systems have to rely on timeouts encoded in the capabilities or centralized revocation mechanisms, *e.g.*, 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].

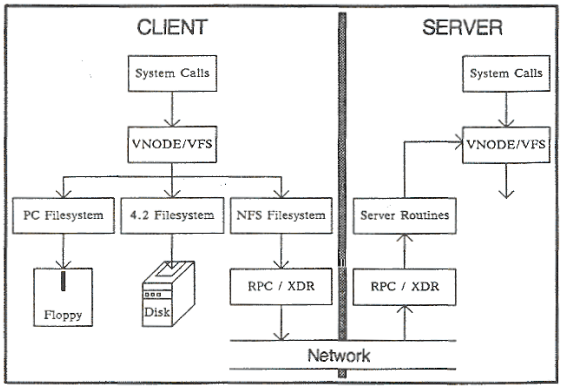

Fig. 6. NFS architecture (from [Sandberg et al. 1985])

We survey a number of networked file systems in this comparison framework in Section 3 and present the results in Table I and Table II.

3. NETWORKED FILE SYSTEMS

3.1 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 was largely 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.

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) where a user's credentials can easily be forged (see Figure 7). Even though support for Diffie-Hellman and Kerberos version 4 authentication flavors was added later on, UNIX style authentication (AUTH_SYS) was the only mandatory flavor and thus most commonly implemented. Host authentication is also weak, because it relies on easily spoofable IPs 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[2]. The rights that can be given to each principal are Read, Write and Execute. Before users can access a remote file, the administrators on their workstation have to 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 to the desired file systems by listing exported file systems and the hosts allowed to mount them in a file called */etc/exports*. 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

[2]Thus, we refer to UNIX mode bits as UNIX ACLs throughout the rest of the discussion.

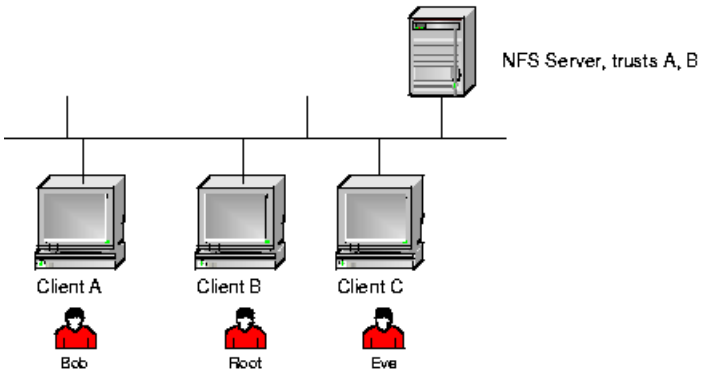

Fig. 7. NFS trust model when using the AUTH_SYS authentication flavor (adopted from [Callaghan 2000]). The NFS server trusts client hosts A and B. Access control is enforced by inspecting the source IP address of RPC requests. User Bob can legitimately access his files after authenticating to client A. However, a superuser 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 look like they come from A. Now client C is trusted even though it is not in the server's access list.

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

While at first sight it seems 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[3] (see Figures 7 and 8). Because no strong host authentication mechanism is used, security is based merely on matching the IP or DNS name of the client workstation with an entry in the exports list. 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.

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. Accord-

[3]This security problem was addressed with the introduction of the ACCESS procedure in NFSv3.

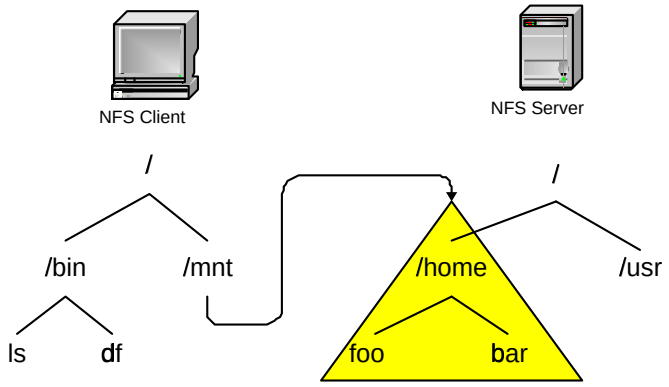

Fig. 8. NFS access control granularity of the mount protocol. The server exports a file system (*e.g.* /home) to the client. An administrator on the client mounts the exported file system (*e.g.* under /mnt). Because the server trusts the client to enforce file access rights, object-access granularity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

ing to them, the numerous ACLs in the system were difficult to administer. In addition, additional intermediate levels of access between administrators and users were desirable. The authors of the Bones system [J. Schönwälder and H. Langendörfer 1993] point out similar problems.

3.2 NFSv4

NFS version 4 [Shepler et al. 2003] proposes many improvements over earlier versions. Stronger security and better suitability to deployment on the Internet are among the design requirements. 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 ONCRPC [Srinivasan 1995]. NFSv4 mandates the use of strong RPC security flavors for authentication[4]. This is achieved by adding a new security flavor based on the Generic Security Services API (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. Currently the mechanisms implementing the GSS-API are Kerberos Version 5, LIPKEY and SPKM:

—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],

[4]Older methods (*e.g.*, AUTH_SYS) can optionally still be supported.

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].

—The Low Infrastructure Public Key (LIPKEY) [Eisler 2000] system 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*, *i.e.*, 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, *i.e.* 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.

—In contrast to Kerberos, the Simple Public-Key GSS-API Mechanism (SPKM) [Adams 1996] 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 Subsection 3.12).

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 based on the Windows NT model [Microsoft Corporation 2005; Swift et al. 2002]. 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 as in WebNFS [Callaghan 1996a; 1996b]. 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.

3.3 AFS

The Andrew file system (AFS) [Howard et al. 1988; Howard 1988; Satyanarayanan 1989; 1990; 1992] was developed at CMU as a secure distributed file system with centralized user authentication. Scalability was a primary consideration in the design of AFS.

Authentication in AFS was initially based on a variant of the Needham-Schroeder authentication protocol [Needham and Schroeder 1978]. The Kerberos authentication system [Miller et al. 1987] was later adopted for purposes of standardization. In earlier versions of AFS, users could only share files with other users in the same administrative domain. Later versions of AFS introduced the notion of *cells* (also known as *realms* in Kerberos) defined along administrative boundaries. Similarly to NFS with Kerberos support, this enables users in different administrative domains to share files. 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 Access Control Lists (ACLs). ACLs are 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, *i.e.*, modify the ACL

AFS ACLs can also specify *negative rights*, *i.e.*, explicitly list operations that a principal is *not* allowed to perform. In the case of conflicts, negative rights override positive rights. This mechanism facilitates rapid and selective revocation, *e.g.*, 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.

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. In addition, negative rights allow for rapid revocation if resolving and updating the user's group membership is expected to take significant time.

3.4 xFS

xFS [Anderson et al. 1995], a serverless network file system, was developed as part of the NOW project at UC Berkeley. Any node in the system can 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 does not carry over to its access control mechanisms.

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

3.5 CIFS

The Common Internet File System (CIFS) [Leach and Perry 1996; SNIA CIFS Technical Work Group 2002; Hertel 2003] is the network file system native to the Microsoft Windows family of operating systems[5]. 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 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. Clients 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, *e.g.* 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 client must first authenticate and get a valid 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 (LM, NTLM, NTLMv2), and, in more recent versions, Microsoft's implementation of Kerberos [Kohl and Neuman 1993; Linn 1996; Swift et al. 2002] or other mechanisms based on the GSS-API [Linn 1997] and SPNEGO [Baize and Pinkas 1998].

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

[5] There are open source implementations for other platforms, *e.g.* Samba [sam]

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, authorization in CIFS can vary from none (when anonymous access is allowed) to access control by ACLs.

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. 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.

3.6 Truffles

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

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

Authorization in Truffles relies on standard UNIX and Ficus access control mechanisms, where each file has a standard UNIX ACL associated with it. The authors propose that a Truffles file system layer stacked on top of the Ficus logical layer perform mapping between local UIDs and globally unique X.500 distinguished names.

Truffles does not address revocation. The authors plan to provide a method of destroying volumes as a way to end a sharing relationship. However, revocation at a finer granularity, *e.g.,* denying a particular user access, is left as future work.

3.7 Bayou

Bayou [Terry et al. 1995; Petersen et al. 1996] is a replicated, weakly consistent storage system designed for the mobile computing environment. Authentication in Bayou is based on public-key cryptography. Every user possesses a public/private key pair and is authenticated by the server using a challenge/response protocol.

Authorization in Bayou is based on digitally signed access control certificates. There are three types of certificates:

—**access granting certificates** grant a user access (one of read, write, or server) to a data

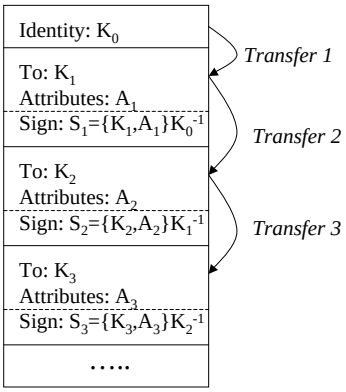

Fig. 9. Structure of a CRISIS transfer certificate (from [Vahdat 1998]). The transfer certificate is a chain of X.509 certificates. The first certificate is an identity certificate identifying the principal wishing to make the transfer by his public key, K_0. In each subsequent certificate the issuer transfers a subset of his available privileges to another principal. *E.g.* 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, *e.g.* in this example K_1 transfers privileges to K_2, who in turn transfers privileges to K_3.

collection. Access granting certificates are signed by a single trusted signing authority.

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

—**revocation certificates** allow the original signer of a certificate to revoke it.

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

All certificates in Bayou are signed by a single trusted signing authority. This limits autonomous delegation across organizational boundaries, because a user in a different administrative domain might be unknown to the signing authority. However, the authors suggest that moving to a web of signing authorities should not be difficult. The access control model in Bayou provides authorization at the granularity of a whole data collection.

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

3.8 WebFS

WebFS is part of the WebOS [Vahdat 1998] project at UC Berkeley. WebFS is 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 is called CRISIS [Belani et al. 1998]. Authentication in CRISIS is based on X.509 certificates [CCITT 1989; Polk et al. 2002; Housley et al. 2002].

Authorization in CRISIS uses a hybrid model to best exploit the tradeoffs between ACLs and capabilities. Principals that should have long-term access to an object are listed on the ACL for that object. In the case of WebFS, each file has an associated list of users authorized to read, write or execute. The principals listed on an ACL can then further delegate a subset of their rights to an object by creating *transfer certificates*, short-lived

and revocable capabilities. Transfer certificates are encoded in X.509 format, digitally signed and can be chained. Figure 9 shows the structure of a CRISIS transfer certificate.

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

CRISIS has good support for revocation. If a principal is listed on an object's ACL his access can be revoked simply by modifying the ACL. When access is granted with certificates, revocation relies on timeouts. Each certificate is first signed by the principal making a statement with a longer timeout. The certificate is then counter-signed by a principal of the signer's choosing. The counter-signature is issued with a shorter timeout. The counter-signer acts as a locally trusted on-line agent (OLA). The OLA checks if a certificate has been revoked before refreshing its counter-signature with a new short timeout. While the CRISIS approach allows for shorter timeouts, it also introduces the need for trusted on-line agents.

3.9 CapaFS

CapaFS [Regan and Jensen 2001] uses self-certifying file names as sparse capabilities to control access to files by users in different administrative domains. A capability file name consists 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 contains the hostname and port of the server. The server part contains the local path name and access rights on the server and is encrypted to protect it from tampering. However, the resulting capability file names are long and meaningless to users and necessitate the use of symbolic links to assign meaningful names to remote files.

There is no explicit user authentication in CapaFS: knowledge of the filename (*i.e.*, possession of the capability) is sufficient to obtain access to a file. Authorization is based on the access rights encoded in the server part of the capability file name. Object access granularity is at the file level.

Because there is no local user identification in CapaFS, autonomous delegation is easily achieved. To share a file, a user need only communicate the file name to another user. Thus, no system administrator involvement is required. However, there are a number of problems with the original CapaFS. Because knowledge of the file name provides access to the file, communicating file names to other users must be done over a secure and authenticated channel (however, no infrastructure for that is developed as part of the system). The original CapaFS is also vulnerable to a man-in-the-middle attack because there is no server authentication. The author describes how to implement server authentication by adding the server's public key to the capability filename. Because no client authentication is performed, there is no accountability in the original CapaFS, *i.e.*, there is no way of telling which particular user accessed a file. The author describes 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 allows for delegation to specific users by including their public keys as an extension of the capability file name. However, there is no way for a user to delegate only a subset of his access rights to another user, *e.g.*. a user possessing a read/write capability file name cannot 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

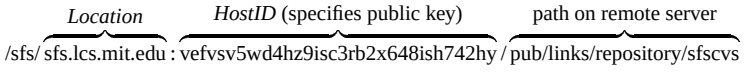

Fig. 10. SFS self-certifying pathname (from [Mazieres et al. 1999])

unlikely to scale well as the list grows with time. Another approach suggested by the author is to limit the lifetime of a capability file name by including a timeout in it.

3.10 SFS

SFS [Mazieres et al. 1999; Mazieres 2000; Fu et al. 2002] is a global decentralized file system. SFS introduces the notion of *self-certifying pathnames* – file names that effectively contain the appropriate remote server's public key (see Figure 10). Thus, SFS needs no separate key management machinery to communicate securely with file servers. By convention, SFS files can 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 the file on the server. As with CapaFS file names, the strings become difficult to remember due to the embedded cryptographic information, so symbolic links must be used as a mnemonic aid.

SFS separates user authentication from the file system. Users in SFS are authenticated using public key cryptography. An agent on the client side authenticates the user to a separate authentication server on the remote server. The authentication server maintains a database mapping public keys to UNIX credentials (a user ID and a list of group IDs). If a user does not have an account on a file server, the server defaults to anonymous access.

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

Autonomous delegation in SFS is 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.

Revocation of a user's access in SFS is simple and similar to what can be done in NFS. Because the authentication server hosts a centralized user database, the user's entry in the database can be easily removed/disabled. A user can also be removed from groups that appear on ACLs for files he is 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 can 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, *e.g.*, due to system policy.

3.11 GSFS

GSFS [Kaminsky et al. 2003] is a further development of SFS with the goal of allowing file sharing between users in different administrative domains. To achieve this goal GSFS introduces changes to the SFS authentication server and extends the standard UNIX file system with SFS ACLs.

Authentication in GSFS is based on public keys, similar to SFS. However, to facilitate global file sharing, the authentication server is modified to contact servers in other admin-

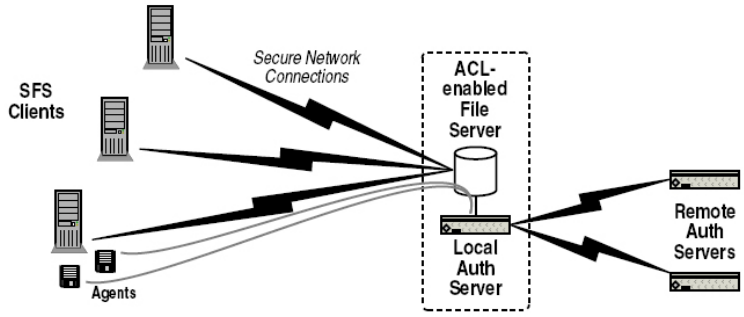

Fig. 11. Overview of the GSFS authentication architecture (from [Kaminsky et al. 2003])

istrative domains and retrieve remote user[6] and group definitions (see Figure 11). Remote authentication servers are referenced with self-certifying hostnames, similar to file servers. A GSFS authentication server must 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 trade 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 introduces a delay between when a decision to grant access has been made and when the actual access can occur.

Authorization in GSFS is done using ACLs. The ACLs are similar to those used in AFS, but are extended to differentiate between files and directories. Access rights available in GSFS ACLs include the right to modify the ACL itself. GSFS ACLs can list four different kinds of principals:

—**User Names** allow a way of listing users with UNIX accounts on the local machine. They are matched against the User Name field of UNIX credentials returned by the authentication server.

—**Group Names** refer to groups defined on the local authentication server. Remote groups cannot be listed directly on the ACL, but can be included indirectly by making them a member of a local group. Group Names are matched against the groups returned by the authentication server in the Group List credentials.

—**Public Key Hashes** are matched against the Public Key credentials returned by the authentication server. Public key hashes are the only way of listing a remote principal directly on a GSFS ACL.

—**Anonymous** entries simply match all users regardless of credentials.

As with SFS, object access granularity in GSFS is 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

[6]For the purposes of this discussion we define remote users to be users outside of the local administrative domain.

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 multi-level delegation, *e.g.*, if Alice gives Bob read access to the file and wishes him to be able to delegate that access, she has to 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[7]. 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, multi-level delegation can be achieved, *e.g.*, 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 Carl.

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”).

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 must deal with revocation involving remote users and groups. Thus, revocation in GSFS is closely related to freshness. If a remote user changes his key or is removed from a remote group record, it will 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 can be instantly revoked by editing the ACL or group record.

[7] This assumes that there is a remote authentication server for the domain that Bob or Alice’s other friends belong to.

```
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. 12. 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.

3.12 DisCFS

The Distributed Credential File System (DisCFS) [Miltchev et al. 2003] uses *trust management credentials* to identify: *(1)* files being stored; *(2)* users; and *(3)* conditions under which their file access is allowed. 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. DisCFS uses KeyNote [Blaze et al. 1999b] as its trust-management engine. An example credential is shown in Figure 12.

Users in DisCFS are identified by their corresponding public keys. In contrast to traditional capabilities, trust-management credentials contain the identities (*i.e.*, public keys) of the user authorizing an action and the user authorized to perform the action (respectively the *authorizer* and *licensee* in Figure 12). While there is no need for authentication in the traditional sense of lookup in a user database, the server needs to verify that a user is the legitimate owner of the public key it presents, *i.e.*, that the user has knowledge of the corresponding private key. In DisCFS this is accomplished by the IKE [Harkins and Carrel 1998] key management daemon as part of the establishment of a secure IPsec [Kent and Atkinson 1998] connection between the clients workstation and the file server. File sharing then takes place over this IPsec association between the client and server hosts.

Authorization in DisCFS is based on trust-management credentials. When a user wishes to access a remote file, the software on the client's workstation sends the relevant credentials with a request to access the file on behalf of the user. The file server passes the credentials along with a query to the KeyNote system. KeyNote checks the signatures on all credentials, evaluates whether the conditions specified in the credentials are met and returns an answer to the query. If the query is successful, the file server grants the user access to the file[8].

[8]The question has been raised whether the DisCFS access control mechanism based on trust-management credentials could be implemented within the NFSv4 framework. Such a mechanism must implement the GSS-API. SPKM could be used as the authentication mechanism because, both clients and servers are represented by public keys. However, the GSS-API makes no provisions for authorization. Appendix A in [Linn 1993a] addresses proposals to add support for Privilege Attribute Certificates (PACs) that carry authorization data. Trust-management credentials seem functionally equivalent to PACs. The appendix states that PACs are currently not visible across the GSS-API interface, and there are no plans to modify the interface, since there are concerns about it losing its

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

DisCFS has 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 12). 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 acts as a CA in DisCFS, the need for higher-level certification authorities is eliminated. Credentials are signed to prevent tampering and can be sent in the clear or posted on the web[9]. DisCFS provides good delegation latency: users can begin accessing files as soon as they are issued a credential.

DisCFS supports multi-level delegation, *i.e.*, if Alice delegates access to Bob, he can then further delegate to Carl by creating a new credential. It is 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, *i.e.*, 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.

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

Revocation in DisCFS is not as straightforward as in ACL-based systems, because it is not always evident who has access to a resource[10]. Thus, DisCFS relies on timeouts in credentials to limit their useful life. As a more user-centric system, DisCFS makes a tradeoff and avoids the administrative overhead of running on-line agents for revocation checks at the expense of having to use longer timeouts.

3.13 WebDAVA

WebDAVA [Levine et al. 2003] is a web file sharing service designed specifically for users in distinct administrative domains. The system provides *file transfer* rather than *file access* services, *i.e.* files must be transferred in their entirety between server and client, rather than being manipulated in place. Thus, WebDAVA cannot be strictly classified as a network file system, however we examine it as another example of a system using authorization

---

generality:

> *" Given that the GSS-API's placement prevents it from providing a comprehensive solution to the authorization issue, the value of a partial contribution specific to particular authorization models is debatable."*

This suggests that a clean implementation of an access control mechanism based on trust-management credentials within the GSS-API would not be possible.

[9] Of course, this is not a good idea in environments where privacy of file access rights is desirable.

[10] In a multi-level delegation chain, a user is only aware of the next "hop", *e.g.*, if Alice delegates access to Bob, and Bob delegates access to Carl, Alice has no knowledge of Carl, and thus no way to revoke his access.

credentials to allow access across organizational boundaries.

Authentication in WebDAVA is performed using a challenge-response protocol. When the server receives a file request it responds with a challenge containing a nonce and the server's public key. The client response includes 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 seems like only the client is being authenticated. It is possible that the server is authenticated by other means, *e.g.* using TLS [Dierks and Allen 1999].

Authorization in WebDAVA is handled by KeyNote [Blaze et al. 1999b] trust-management credentials. The credentials authorize desired actions corresponding to the HTTP GET or PUT methods. Downloading a file from the server is done via the HTTP GET method. The PUT method allows file creation or modifying a stored file by overwriting it. Deleting a file is done by saving an empty file; the server notices that the file is empty and removes it.

Granularity of access control in WebDAVA is at the file level.

WebDAVA has full support for autonomous delegation between users in distinct administrative domains. Users can delegate a subset of their access to any other users by retrieving their public keys and issuing them a credential. No administrator involvement is required. Credentials are protected from tampering by a signature and thus can be sent over e-mail or downloaded from the web.

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

3.14 Fileteller

FILETELLER [Ioannidis et al. 2002] is a credential-based network file storage system with provisions for paying for file storage and getting paid when others access files. Users get access to arbitrary amounts of storage anywhere in the network, and use a micropayments system to pay for both the initial creation of the file and any subsequent accesses. 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.

Authentication in FILETELLER is public key based. There are three participants in the system: *Network Users* (NUs), *Network Storage Providers* (NSPs), and *Check Guarantors* (CGs). All participants are identified by their public keys. An NU needs to authenticate with the NSP before any file operation can take place. The authentication protocol must provide strong authentication and, optionally, let the user piggy-back credential delivery to the NSP. Security protocols like IPsec [Kent and Atkinson 1998] or TLS [Dierks and Allen 1999] can be configured to meet these requirements.

Authorization in FILETELLER is based on KeyNote [Blaze et al. 1999b] trust-management credentials. An NU holds one or more credentials issued by a CG indicating the NU's credit line with the CG, as shown in Figure 13. There are four kinds of credentials used in different parts of the system:

(1) Check Guarantor credentials, which specify a user's line of credit.

(2) Microchecks, which authorize a payment from an NU to an NSP, or to another NU.

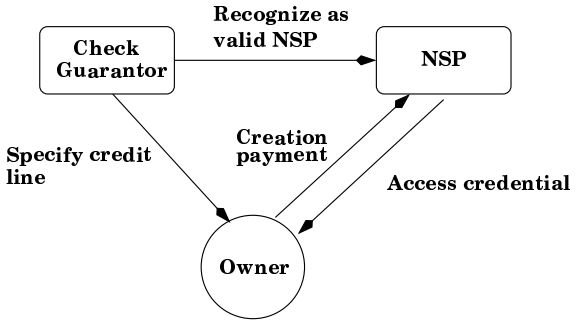

Fig. 13. NSPs issue a KeyNote credential to each Check Guarantor (CG) authorizing them to act as introducers of users, by in turn issuing them credentials. A file owner needs to convince a CG to provide them with a credit line, also expressed as a KeyNote credential. The file owner needs to provide these two credentials to the NSP, along with a microcheck conveying payment to the NSP. In response, the NSP returns to the file owner a KeyNote access credential, granting her full privileges in accessing the file.

(3) Server credentials, issued by the CGs, that identify complying NSPs the NUs can use.

(4) File access credentials, initially issued by NSPs when a file is created, authorizing subsequent access to that file by the NU owner. File owners can then issue further file access credentials, delegating access to other NUs.

After authenticating, the NU needs to tell the NSP the name of the file to be accessed, the operation type (create, replace, read, append, delete), the size of the file (for replace, append, or create), the file disposition (ordinary or "pay-back"), the CG credential(s), a transaction ID (a random number used to match requests with responses), and who the owner of the file will be (if someone other than the creator). When the NSP receives this description, it makes sure that the CG is one that it knows about, and sends back an *offer*, consisting of the transaction ID, a nonce (for billing purposes), and the cost (in real or "play" currency). The client then writes a microcheck[11], and sends it along with the credentials necessary to approve the operation: the CG credential(s), the file ownership chain of credentials (if this is an access to an existing file), followed by a copy of the offer, the operation, the file disposition, and the attributes. If the operation was a 'create,' 'replace,' or 'append,' the contents of the file are sent. If the operation was a 'read,' or a 'delete,' nothing else is sent.

When a file is created, the server responds with a file access credential, granting the creator of the file full access. If a file is replaced or appended to, the server responds with a signed 'receipt' for the transaction. Otherwise, the file is sent to the client.

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

[11] If the client request was for a file with a "pay-back" disposition, two microchecks must be issued; one to the server, and one to the owner. The server verifies that the owner will get paid before releasing the file.

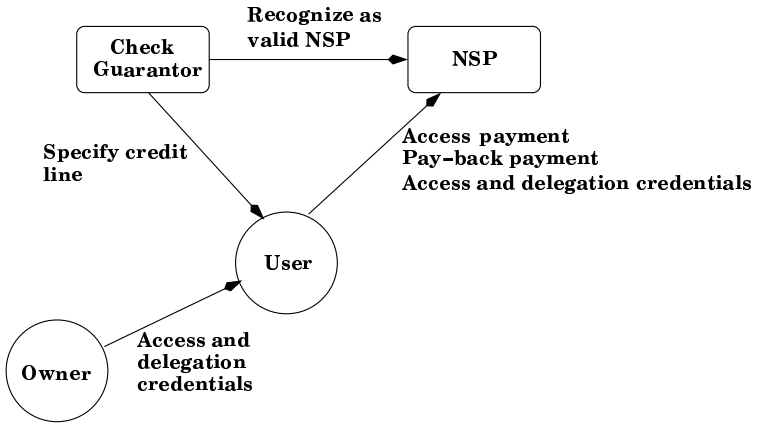

Fig. 14. A user wishing to access another user's file needs to have their own line of credit with a Check Guarantor (CG), as well as a credential from the file owner granting them access to that file. When accessing the file, the user needs to provide the credit-backing credential from the CG, a microcheck 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.

Autonomous delegation across organizational boundaries is supported in FILETELLER as shown in Figure 14. An NU with access to a file can delegate a subset of his access rights to another NU by issuing a file access credential. This delegation mechanism is transitive and does not require administrator involvement. Users need not reside in the same administrative domain, however a user wishing to access a file served by a given NSP must establish a line of credit with a CG that recognizes the NSP as valid. Because users are vouched for by a CG and uniquely identified by their public keys, accountability is preserved. File attributes are used in file access credentials to allow fine granularity delegation. These attributes are meta-data associated with the file by the owner, and can be used to implement easy file grouping, associate security labels with files, or for any other similar scheme.

Revocation in FILETELLER is time-based and relies on credential expiration. As with [Blaze et al. 2001], CG credentials issued to users are 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.

4. ANALYSIS

Table I classifies the file systems studied 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*

Table I. File system classification

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

[1] locally trusted on-line agent

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. Out 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, if the capabilities are not tied to user identities in any way, it is impossible to

Table II. Autonomous delegation support in networked file systems

	Autonomy	Organizational Independence	Low Latency	Transitivity	Fine Granularity	Accountability
Truffles	•	[1]	•	•		•
Bayou	•	[2]	•	•	•	•
WebFS	•	[3]	•	•	•	•
CapAFS[4]	•	•	•	•		
CapAFS[5]	•	•	•	•		•
GSFS[6]	•	•	•		•	•
GSFS[7]	•[8]	•	[9]	•		•
DisCFS	•	•	•	•	•	•
WebDAVA	•	•	•	•	•	•
Fileteller	•	•	•	•	•	•

[1] Users from different administrative domains must have a common root CA.
[2] All access granting certificates are signed by a single trusted signing authority.
[3] Remote users must have a certificate from a CA trusted by the local domain.
[4] Original design without user authentication.
[5] With user authentication.
[6] Public key hashes of remote users listed on ACL.
[7] Remote groups listed on ACL.
[8] Remote users must be associated with remote authentication server.
[9] User records must be periodically fetched from remote authentication servers.

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.*

GSFS, a further development of SFS, 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 multi-level delegation is not met. If groups are used instead, multi-level 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

In this article we provided a framework for comparing the suitability of access-control mechanisms of networked file systems for global file sharing across organizational boundaries. We surveyed a number of systems in the context of our taxonomy. Systems based on authorization certificates generally provide better support for autonomous delegation of access rights between users in different administrative domains than systems based on ACLs or pure capabilities. Future research on revocation of authorization certificates could seek to minimize the existing tradeoff between user autonomy and ease of revocation. We believe that our taxonomy provides a valuable contribution to the networked file systems design space.

REFERENCES

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

ADAMS, C. 1996. The Simple Public-Key GSS-API Mechanism (SPKM). RFC (Proposed Standard) 2025, Bell-Northern Research. October.

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 *Proc. 15-th Symposium on Operating Systems Principles.*

BAIZE, E. AND PINKAS, D. 1998. The Simple and Protected GSS-API Negotiation Mechanism. RFC (Proposed Standard) 2478, Bull. December.

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. February.

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., New York, NY, USA, 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. September.

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 Fifth International Conference on Financial Cryptography*.

CALLAGHAN, B. 1996a. WebNFS Client Specification. RFC (Proposed Standard) 2054, Sun Microsystems, Inc. October.

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. June.

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. *Communications of the ACM 9,* 3 (March), 143–155.

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

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. June.

EISLER, M., CHIU, A., AND LING, L. 1997. RPCSEC_GSS Protocol Specification. RFC (Proposed Standard) 2203. September.

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

FU, K., KAASHOEK, M. F., AND MAZIÈRES, D. 2002. Fast and secure distributed read-only file system. *Computer Systems 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 1990 USENIX Conference.* 63–71.

HARKINS, D. AND CARREL, D. 1998. The Internet Key Exchange (IKE). RFC (Proposed Standard) 2409, Internet Engineering Task Force. November.

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

HOUSLEY, R., FORD, W., , POLK, W., AND SOLO, D. 2002. Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile. RFC (Proposed Standard) 3280. April.

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 Transactions on Computer Systems 6,* 1 (February), 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,* First ed.

IOANNIDIS, J., IOANNIDIS, S., KEROMYTIS, A., AND PREVELAKIS, V. 2002. Fileteller: Paying and Getting Paid for File Storage. In *Proceedings of the Sixth International Conference on Financial Cryptography.*

J. SCHÖNWÄLDER AND H. LANGENDÖRFER. 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. February.

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, New York, 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. February.

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

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. Requirements for Scalable Access Control and Security Management Architectures. *To appear in the ACM Transactions on Internet Technology (ToIT).*

KOHL, J. AND NEUMAN, C. 1993. The Kerberos Network Authentication Service (V5). RFC (Proposed Standard) 1510. September.

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, USA.

LINN, J. 1993a. Generic Security Service Application Program Interface. RFC (Proposed Standard) 1508, Geer Zolot Associates. September.

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. February.

LINN, J. 1996. The Kerberos Version 5 GSS-API Mechanism. RFC (Proposed Standard) 1964, OpenVision Technologies. June.

LINN, J. 1997. Generic Security Service Application Program Interface, Version 2. RFC (Proposed Standard) 2078, Internet Engineering Task Force. January.

LYON, B. 1984. Sun Remote Procedure Call Specification. Tech. rep., Sun Microsystems, Inc.

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

MAZIERES, D., KAMINSKY, M., KAASHOEK, M. F., AND WITCHEL, E. 1999. Separating key management from file system security. In *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. December.

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. *Commun. 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 Second International System Administration and Networking (SANE) Conference*.

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. April.

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*.

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.

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. Eng. 18,* 1, 1–8.

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. April.

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

SRINIVASAN, R. 1995. RPC: Remote Procedure Call Protocol Specification Version 2. RFC (Proposed Standard) 1831, Sun Microsystems. August.

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. February.

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 Transactions on Information and System Security 5,* 4 (November).

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. *The Bell System Technical Journal 57,* 6, Part 2 (July–August), 1931–1946.

TROSTLE, J. T., KOSINOVSKY, I., AND SWIFT, M. M. 2001. Implementation of Crossrealm Referral Handling in the MIT Kerberos Client. In *NDSS.*

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

WESTERLUND, A. AND DANIELSSON, J. 2001. Heimdal and Windows 2000 Kerberos: How to Get Them to Play Together. In *Proceedings of the 2001 USENIX Annual Technical Conference, Freenix Track.* 267–272.

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

Received Month Year; revised Month Year; accepted Month Year