USENIX Association

Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference

San Antonio, Texas, USA
June 9-14, 2003

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Secure and Flexible Global File Sharing

Stefan Miltchev
miltchev@dsl.cis.upenn.edu
University of Pennsylvania

Vassilis Prevelakis
vp@drexel.edu
Drexel University

Sotiris Ioannidis
sotiris@dsl.cis.upenn.edu
University of Pennsylvania

John Ioannidis
ji@research.att.com
AT&T Labs – Research

Angelos D. Keromytis
angelos@cs.columbia.edu
Columbia University

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

Abstract

Trust management *credentials* directly authorize actions, rather than divide 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.

The Distributed Credential FileSystem (DisCFS) uses trust management credentials to identify: *(1)* files being stored; *(2)* users; and *(3)* conditions under which their file access is allowed. Users share files by delegating access rights, issuing credentials in the style of traditional *capabilities*. Credentials permit, for example, access by remote users not known in advance to the file server, which simply enforces sharing policies rather than entangling itself in their management. Throughput and latency benchmarks of our prototype DisCFS implementation indicate performance roughly comparable to NFS version 2, while preserving the advantages of credentials for distributed control.

Keywords: Filesystems, access control, trust management, credentials.

1 Introduction

The Internet offers the possibility of global information sharing and collaboration. Existing file-sharing systems and their access control models have been challenged, however, by the scale and administrative complexity of the Internet. Such systems are either primarily distribution-oriented, such as the Web, or small in scale, such as NFS.

Both types of systems share the property that a (relatively) small set of users have read/write access to files, as current access control systems rely on authentication requiring that the user is known to the system. This works in closed administrative domains (*e.g.,* NIS domains or Kerberos realms [21]), where an *administrator* creates a user account and assigns access rights to it.

Consider a local user, Alice, who wishes to share files with Bob, who does not have an account on Alice's file server. The local system administrator must open an account for Bob, but this may create administrative and legal problems, and may conflict with local policies (*e.g.,* only employees may have accounts).

We propose a mechanism that allows Alice, without the intervention of any centralized administrative authority, to authorize Bob to access her files. This is done by having Alice create a credential that contains Bob's key, the DisCFS file handle and the permissions. Alice signs the credential, and confers to Bob the authority to access the file. Alice may simply e-mail the authorization credential to Bob as cleartext because the credential itself does not contain any secrets (apart from the information that Alice wishes to share a file with Bob). By combining Alice's credential with one signed by himself, Bob may further delegate access to the file.

We show that this simple mechanism is secure and scalable. Further, by requiring the cooperation of only the users involved in the file exchange, this mechanism offers great flexibility and low administrative overheads. The system monitors all access to files, and can identify, using the offered public key, any entity issuing file requests. Mechanisms for restricting access or imposing access controls are also provided.

The access control mechanism that is presented in this paper is independent of the actual mechanism used for the exchange of data (*e.g.,* ftp, NFS, http, and so on). We implement two prototypes, one for *ftp*-like access (not covered in this discussion) and another using the NFS protocol.

In the following sections we demonstrate how our mechanism may be deployed in practice using the NFS prototype as an example. We integrate our access mechanism with a user-level NFSv2 server using IPsec [16]; our intention is to offer this access mechanism eventually as part of the standard NFS authentication framework. The performance measurements collected by running common file-related benchmarks indicate performance roughly comparable to existing systems.

Organization The next section expands on the challenges addressed by DisCFS. Section 3 discusses prior and related work. Sections 4 and 5 describe our design and prototype implementation for OpenBSD 3.1, and Section 6 presents measurements of the prototype. Section 7 presents user and administrator experiences with our first prototype. Sections 8 and 9 discuss future work and conclude the paper.

2 Motivation

Let us consider two typical examples of information sharing. In the first case, Alice, a salesperson, would like her ten best clients to have access to advance information about a product. She does not want other clients or the general public to have access to this information, however.

The second example involves servers used to share information such as digital photographs (*e.g.,* `www.ofoto.com`). Alex is given the authority to store his personal photographs on a server. Apart from Alex, access to this information may be restricted to small groups of users. These groups may be different, depending on the material (*e.g.,* pictures of family events to relatives, pictures of social events only to those who participated, *etc.*).

In both cases, local users (Alice and Alex), known to their systems, wish to provide access to other external users. For this type of activity to be feasible, the following conditions must be met:

- The system should be able to cope with large numbers of files and an even larger number of users accessing these files.
- Administrator involvement to allow external users access to files should be eliminated. Local users should be able to authorize access to files by external users.
- The file access conditions must be flexible and expandable: there should be no constraints by the access mechanism as to what conditions may be imposed for access.
- Delegation is extremely important for the operation of the system. There is already an implicit delegation of access authority from the administrators to the local users and from the local users to external users, but we want to generalize the ability to delegate to arbitrarily long delegation chains.
- Most sites have access policies and these must be enforced regardless of the actions of individual users. The administrator should be able to specify default access policies for the entire site.
- Apart from the actual files, the system should maintain as little additional state as possible.
- The access mechanism should work for both centralized servers and in a distributed environment where the files are stored on multiple servers.

Existing systems have several major shortcomings when used for sharing information. First, traditional user authentication implies that a user is known to the system before file requests can be processed. Second, file and directory permissions are concepts inherited from multiuser operating systems. Sharing is achieved by either account sharing (which defeats accountability) or through the use of group access permissions on files and directories. Such permissions lack flexibility and fine granularity, and perhaps most importantly, extensibility: there is no way of adding new permission mechanisms if the existing ones prove inadequate.

In the salesperson example, because the information is not intended to be widely available, Alice must place the literature in a restricted part of the corporate Web site and make arrangements so that only the designated clients have access to the material. The traditional way of doing things implies that accounts and passwords should be created and given to the customers. A more sophisticated way of achieving the same goal is to use X.509 credentials for user authentication [7]. Although this approach addresses the well-known security problems of password authentication, it does not address the problem of access control, necessitating the maintenance of additional state (*e.g.,* access lists) on the server.

Faced with complex and inflexible mechanisms, some sites abandon access restrictions, and instead rely on obscurity (*e.g.,* non-obvious URLs, files in unreadable directories, *etc.*). Others use cookies, despite the fact that they are known to have numerous weaknesses [11].

Before we continue with the description of the Distributed Credential File System (DisCFS), which was designed and implemented to meet the listed requirements, we shall discuss previous work done in the area of wide-area file sharing.

3 Related Work

Network file sharing has attracted the attention due it; the need for distributed information sharing has expanded with the reach and services of the Internet, and will continue to do so. There are some undesirable tradeoffs in existing systems that inhibit large-scale network file sharing.

3.1 File Systems

Network file systems such as NFS and AFS [27, 14] are the most popular and widespread mechanisms for sharing files in tightly administered domains. However, crossing administrative boundaries creates numerous administrative problems (*e.g.,* merging distinct Kerberos realms or NIS domains). The developers of the Athena, Hesiod and Bones systems recognize and address some of these problems [26, 8, 15].

Encrypting file systems such as CFS [4] place great emphasis on maintaining the privacy of the user information by encrypting the file names and their contents. The limitation of such systems is that sharing is particularly difficult to implement; the file owner must communicate the secret encryption key for the file to all the users who wish to access it. Even then, traditional access controls must still be used to enforce access restrictions (*e.g.,* read-only, append-only, immutable file, *etc.*). Furthermore, because CFS encrypts data stored in files, modifying files or altering their permissions is inefficient. Our system assumes that the server is trustworthy, so that the files can be stored in the clear. An administrator may still choose to deploy CFS-like encryption mechanisms on top of DisCFS.

WebFS is part of the larger WebOS [29] project at UC Berkeley. It implements a network file system on top of the HTTP protocol. WebFS relies on user-level HTTP servers, used to transfer data, along with a kernel module that implements the file system. The security architecture for WebOS, called CRISIS [3], uses X.509 certificates to identify users and to transfer privileges. Associated with each file are access control lists (ACLs) that enumerate which users have read, write, or execute permission on individual files. We have taken a more general and scalable approach in that there is no need for traditional ACLs because each credential is sufficient to identify both users and their corresponding privileges.

Bayou [28, 23] is a replicated, weakly consistent storage system, designed for the mobile computing environment. Like CRISIS, it uses certificates to identify users, and permits read or write access to data collections. It also supports delegation and revocation certificates. DisCFS supports finer-grained access than Bayou; DisCFS credentials can contain much richer security policies.

The design rationale of the *capafs* system [24] is similar to that of DisCFS. Security for file access is is done by encoding the access capabilities in the file name. This both results in incomprehensible file names (requiring symbolic links for user interactions) and knowledge of the file name providing access to the file (requiring protection of the file name and raising the question of how file names are communicated amongst remote users). In addition, limitations in the way directories are handled restrict file creation operations to local users.

The system that is most closely related to our work is the secure file system, or SFS [20]. SFS introduces the notion of *self-certifying pathnames* — file names that effectively contain the appropriate remote server's public key. In this way SFS needs no separate key management machinery to communicate securely with file servers. However, because SFS relies on a trusted third party to mutually authenticate server and client (or otherwise requires the use of a secure password protocol between them), collaboration is possible only if the client and the server have a common root for their Certification Authorities. DisCFS goes a step further. It uses credentials to identify both the files stored in the file system and the users that are permitted to access them, as well as the circumstances under which such access is allowed. Furthermore, users can delegate access rights simply by issuing new credentials, providing a natural and very scalable way of sharing information. This is not the case for SFS, where access control relies on user and group ID, which are translated from one machine to another. This forces users to have accounts on file servers to access protected files, and defeats the purpose of a truly distributed file system.

Putting DisCFS in the secure storage framework devised by Riedel, *et al.* [25], DisCFS has the following characteristics:

Players. As with iSCSI [10], there is no notion of individual users; readers, writers and owners are undifferentiated with respect to the credentials they possess.

Likewise, because access is not based on membership groups, there is no group server. There is no namespace server either, only a DisCFS storage server.

Trust assumptions. Data is not protected from the server, making it vulnerable to a collusive storage server, as with iSCSI.

Security primitives. Servers and hosts authenticate with mechanisms external to DisCFS. As with iSCSI, any authentication keys are distributed by an external mechanism.

The server does differentiate between reads and writes. However, whether the server grants a read or write request depends solely on information contained in the credential presented by the requesting party.

Credentials provide appropriate access control and can be sent in the clear. For further security, data and commands can be encrypted while on the wire using IPsec.

Revocation is achieved by revoking credentials. Keromytis discusses a variety of mechanisms for credential revocation in trust management systems in [17].

Granularity. DisCFS is agnostic to the duration of the keys and credentials, but our expectation is that they are reasonably persistent.

Convenience. DisCFS is convenient to use. Once credentials for a file are available, delegation is easier than in identity-based systems, as we describe in some detail later in the paper.

3.2 Operating Systems

The Taos operating system [30] uses a *narrow* API to control access to security-aware services, *e.g.,* the file system. It uses credentials for delegation of access rights between principals. DisCFS utilizes these well-known techniques to extend a real world protocol such as NFS. This makes our approach easily portable and more flexible than specialized operating system mechanisms.

The concept of credential-based access control also appears in the Exokernel [19]. In this system, users can create new capabilities at will, but the new capability must be dominated by an existing one. This is similar to our chains of certificates, but is limited by the fact that permissions are hardwired into the system, and the hierarchical capability tree may be only up to 8 levels deep. In our system, certificate chains can be of arbitrary length, and the access policy can consider factors such as time-of-day, so that, for example, leisure-related files may not be available during office hours.

3.3 Other Protocols

To share files across wide-area networks, a number of protocols have been deployed, the most commonly used ones being FTP and HTTP. Anonymous FTP, where there is no need for authentication, offers high flexibility

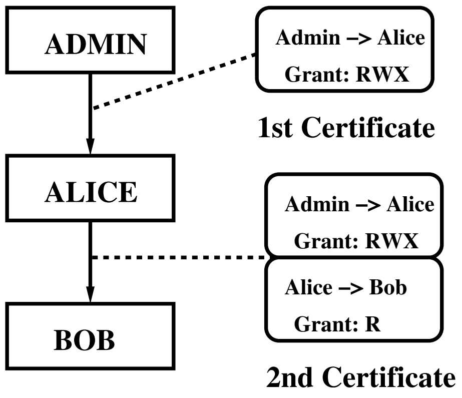

Figure 1: Delegation of privileges, from the administrator to Alice, and from Alice to Bob. The administrator grants Alice full access by issuing her the first certificate. Alice can then delegate read access to Bob by issuing him the second certificate. To be granted access Bob must present a certificate chain consisting of both certificates.

because any user can download or upload files to FTP servers. Similarly, in the Web architecture, access is either anonymous or subject to some sort of *ad hoc* authentication mechanism. This configuration is useful only in the case where file content is not sensitive. In the case where authentication is required, flexibility is greatly reduced. The only users allowed to access the server, in that case, are users who are already known to the system. As with existing network file systems, this restricts the possibility for users outside the same administrative domain to collaborate.

4 DisCFS Design

4.1 System Architecture

DisCFS dispenses with user names as an indirect means for performing access control; in other words, it does not require users to first authenticate to the system and then have their identities checked against access control lists in order to determine whether the server should honor a client's request. Instead, DisCFS uses a direct binding between a public key and a set of authorizations. This results in a decentralized authorization system that is flexible enough to cope with a large variety of authentication scenarios. Requests are signed with the requester's key and must be accompanied by other credentials that form a chain of trust linking the requester's key to a key that is trusted by the system. In our first example in Section 2, we looked at Alice's predicament in trying to allow her sales clients access to internal files. In the DisCFS system, the server trusts only the administrator's key. An administrator signs Alice's credential, binding her key to the files in question. The credential allows Alice read, write and execute access to the files.

If Alice then wishes to allow Bob to read these files, she simply creates a new credential which grants Bob's key read access to the files. Bob issues a request signed with his key. If the system is to honor Bob's request, Alice's credential must accompany it. This credential forms a link between the external user (Bob) and the internal user (Alice). Alice's own credential (issued by the administrator) must also be available, to link the internal user to the administrator. Thus, Bob's request must be accompanied by both credentials in order to be granted (see Figure 1). Credential caching can reduce the number of credentials that have to be exchanged.

In DisCFS the traditional problem of credential (or certificate) revocation is fairly straightforward to address: because the DisCFS server controls access to files by examining a user's credentials, revocation is accomplished by notifying the server about bad keys or credentials. If the credentials are relatively short-lived, the server need only remember such information for a short period of time.

To express access rights and the diverse conditions under which these are granted, we need some form of policy definition language. There are a number of possible choices such as, PolicyMaker [6], KeyNote [5], QCM [12] and SPKI [9]. In our system we use the KeyNote trust management system for this purpose. Our choice was based on the fact that KeyNote is also integrated into IPsec which protects communication between the client and the file server. By using IPsec and Keynote, we can also use the file access credentials to establish the IPsec link (see Section 4.3).

4.2 Access Control in DisCFS

Generally speaking, access control systems check whether a proposed action conforms to policy. An administrator specifies actions as a set of name-value pairs, called an action attribute set. Policies written in the KeyNote assertion language either accept or reject action attribute sets that the system presents to the policy engine (non-binary results are also possible). An administrator can break up policies and distribute them as credentials, which are signed assertions that he can send over a network. A local policy can then refer to the credentials when making its decisions. The credential mechanism allows for arbitrarily complex graphs of trust, in which credentials signed by several entities are considered when authorizing actions, as seen in Figure 2. An administrator can handle group-based access

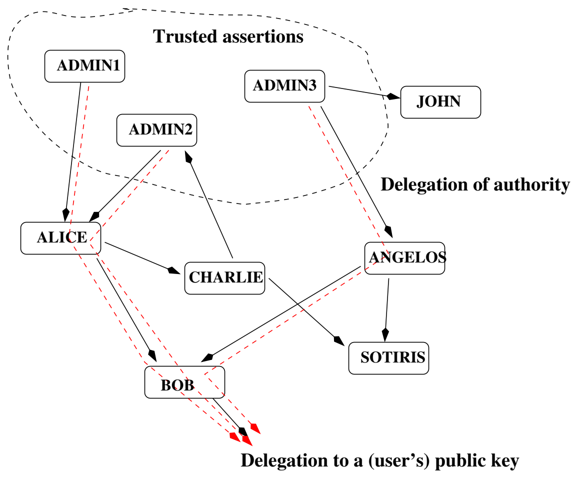

Figure 2: Delegation in KeyNote, starting from a set of trusted keys. The dotted lines indicate a delegation path from a trusted public key (the administrator's) to the user making a request. If all the credentials along that path authorize the request, it will be granted. Intermediate credentials can only refine the authority granted to them (*i.e., they can never allow a request that was denied by policy*).

simply by issuing the appropriate credentials to all the group members. Furthermore, an administrator can create sub-groups at any level in the hierarchy.

Figures 3 and 4 show two credentials that form part of a delegation chain in our system. The administrator issues a credential to user *miltchev* granting access to a particular directory. User *miltchev* then grants user *sotiris* access to the same directory during November 6, 2002. KeyNote allows this delegation because the authority granted to user *sotiris* is a subset of the authority of user *miltchev*. Notice that users are identified only by their public keys.

The advantage of using DisCFS is that an administrator no longer needs to have *a priori* knowledge of the user base. Thus, the system does not need to store information about every person or entity that may need to retrieve a file. DisCFS also provides its users with the ability to propagate access to the files by passing on (delegating) their rights to other users. In this way, users pass credentials rather than passwords, allowing the system to associate access requests with keys and to reconstruct the authorization path from the administrator to the user making the request (and thus grant access). The system may not know that Bob is trying to get at a file, but can log that key B (Bob's key) was used and that key A (Alice's key) authorized the operation. The administrator can then use this audit trail to validate that the access request follows the appropriate usage policy. Note

```
Authorizer: "<Administrator's Public Key>"
Licensees: "<Miltchev's Public Key>"
Conditions:     (app_domain == "DisCFS") && (HANDLE == "666240") -> "RWX";
Comment:        "testdir"
signature: "<Signature by Administrator>"
```

Figure 3: Credential granting user *miltchev* (as identified by his public key, in the *Licensees* field) access to directory *testdir*. The 1024-bit keys and signatures in hex encoding have been omitted in the interest of readability.

```
Authorizer: "<Miltchev's Public Key>"
Licensees: "<Sotiris' Public Key>"
Conditions:     (app_domain == "DisCFS") && (HANDLE == "666240") &&
                (localtime >= "20021106000001") &&
                (localtime <= "20021106235959") -> "RWX";
Comment:        "testdir"
signature: "<Signature by Miltchev>"
```

Figure 4: Credential by user *miltchev* granting (delegating) user *sotiris* access to directory *testdir* for one day. Again, the keys and signatures have been omitted in the interest of readability.

that a user can at most pass on the privileges she holds (there is no *rights amplification*); furthermore, the user can choose to delegate only a subset of her privileges, *e.g.,* access only to a specific file in the user's directory.

4.3 DisCFS over NFS

We implemented DisCFS over NFS. This allows easy integration into existing systems without extensive modification. Moreover, the entire scheme works with both monolithic and distributed servers. Each DisCFS repository is responsible only for the part of the distributed filesystem that is stored locally, thus there is no need to distribute and synchronize authentication and access control databases (such as NIS).

The NFS protocol is particularly suitable for our needs for the following reasons:

- NFS is widely used and supported by numerous platforms.
- The NFS protocol is portable, stable and reliable.
- The NFS server is available as a user-level program, so development is possible without modifying the operating system. This is particularly useful because it is not always possible to have access to the operating system source code.

Like NFS, the DisCFS system consists of a client and a server. The client runs on the user workstation and establishes a connection to the DisCFS server. We use IPsec [16] to protect traffic between client and server. The mutual authentication required for building an IPsec connection is based on the submitted file access credential (and additional delegation credentials). The client can authenticate the server, because the file access credential contains the server key, while the server only proceeds with the connection if the submitted credentials allow access to the requested file (thus establishing a chain of trust to the user's key).

When a file is stored in DisCFS, the server generates a credential containing information that allows the future retrieval of the file contents, as well as information about the file creator. Because DisCFS closely follows NFS semantics, it appears to the user as another mounted file system. Files for which credentials have been supplied appear under the mount point of the DisCFS file system. Without an appropriate credential, retrieval of a file is not possible.

Once a user submits the necessary file credentials, the file appears under the DisCFS mount point using the same name it had when its credential was created. The client may then use file I/O requests similar to NFS. The system also permits a user to override the default file name, allowing files to be placed in user-specified locations. This is because DisCFS access credentials allow direct access to files, making file naming optional. The name is stored as a comment in the credential and is used as the default file name. The operation of establishing a connection to the server is similar to the Unix *mount(8)* command whereby an entire filesystem is grafted to the file tree. See Section 5 for a detailed explamation of how users access files on a DisCFS server.

If additional files must be accessed from the same server, the existing IPsec connection is used. This optimization allows the cost of the IPsec connection establishment to be spread over requests for multiple files.

4.4 Security Analysis

Our system allows users of a server to access their files securely over an insecure medium (Internet). Unlike ACL-based systems that authenticate users and then check their access rights, our system is concerned only with capabilities associated with a key. The system must, therefore, decide whether a request signed with a given key should be granted or denied. The decision depends on whether the system can form a chain of trust between the issuing key and another key that the server can trust (*e.g.,* the key of a system administrator).

The chain of trust is formed by merging information that is pre-stored in the server (default policy) and policy statements that the user supplies. These policy statements are encapsulated in credentials. Each credential makes some assertions about one key and are signed with another key. All these assertions together with the authorizing keys are fed to the policy engine in the access control system.

Credentials are signed but not encrypted. They contain policy assertions and the public component of the key that they apply to. Both key and policy are signed with the authorizing key. The implication of this feature is that credentials may be retrieved on demand. For example, if Alice wants to send Bob a pointer to some information, instead of sending a URL to the relevant page, she may send Bob a URL pointing to the credentials needed to access the information. Because these credentials contain a reference to the file, Bob needs no further information to access the file. Obviously, using the http protocol to download credentials is one of many ways of acquiring them.

If such a credential is intercepted, the only information that may be obtained is that key A makes some assertions regarding key B. Credentials are signed and therefore cannot be forged. Because they relate to specific files (defined by the assertions that they contain), they cannot be used to obtain access to other information.

The default policies may define the rules that apply to a particular server. For example, they may allow access only between certain hours of the day, or they may exclude or limit access granted to a particular user.

Data is encrypted only while in transit. It is stored as cleartext on the server, which implies that users trust the server and its administrators. Existing data-protection schemes (such as a CFS-like filesystem) can be used on top of DisCFS to protect the secrecy of the user's data from the server.

Credential-based access control systems usually have problems handling revocation, as it is difficult, if not impossible, to know who may have access to a file. However, by controlling the file server, administrators have a number of ways for disallowing access to files:

- Because the initial access credential uses the DisCFS file handle to address the corresponding file, an administrator can invalidate the access credential by changing the handle (see Figure 3).
- Administrators can disallow particular keys from being used (thus revoking the keys, rather than the credentials that grant them privileges) by modifying the site's security policy. This approach, however, is not scalable and should be used in conjunction with relatively short lived credentials (*i.e.,* with expiration periods measured in weeks, rather than years).
- Administrators can tie the access credentials to short-lived "refresher" credentials. Thus the access credential is not useful without a valid (non-expired) refresher.

Note that delegation, although extremely useful, can be turned off. Administrators can also limit the maximum length of the delegation chain (by inserting policy code in the access credentials), thus restricting the spread of delegated credentials.

The main advantage of having policy embedded within the credentials is that administrators can have multiple schemes operating at the same time. Different schemes may be used on different categories of files depending on, *e.g.,* the security classification of each category. Thus, for restricted files (lowest classification) administrators may rely on the expiration of the credentials; on more sensitive files they can use the default site policy, and if a file has to be unconditionally removed, administrators can change its handle and issue new credentials to the users that should access it.

4.5 Threat Analysis

At the object level, the threat model of DisCFS does not seem any different from any simple file access control protocol. DisCFS does not encrypt files on disk, thus users have the option of trusting the system administrator or using encryption mechanisms on top of DisCFS.

At first glance the threat model of DisCFS at the network level does not differ from that of NFS used over a secure channel, *e.g.* a trusted LAN or VPN. However, what sets DisCFS apart is its ability to address the threat of implicit rights amplification inherent in identity-based access control. All co-authors writing an arbitrary paper using CVS need to have login access to the serving machine. In contrast, the credential-based access control of DisCFS allows us to trust the co-authors no more than we have to - for authorship and nothing else.

4.6 Scalability

An important advantage of the access mechanism we present is scalability. This is due to the fact that we do not maintain user access rights on the DisCFS server. In fact, the server does not even have the concept of a "user" in the traditional operating system sense; it merely processes requests from keys that can supply a valid trust chain to one of the keys contained in the default policy of the server.

This design decision has two implications: (a) users must supply (sometimes long) chains of credentials, and (b) the server's trust state remains constant irrespective of the number of potential users. Caching alleviates some of the processing overhead associated with the long trust chains.

Keeping servers uncontaminated by user information allows data set partitioning and replication across systems even in separate administrative domains. The disk capacity of the server must be proportional to the amount of information it contains, and its connection to the Internet should be appropriate for the actual traffic it experiences, not the user set or the number of policies that must be enforced.

Although the number of *potential* users is irrelevant, the number of *actual* users that connect to the server to access information is important because the access control operations they initiate load the server. It is clear that a KeyNote-based access control mechanism places greater load on the processing resources of the server, however this can be addressed in two ways: *(a)* by using hardware acceleration for the cryptographic operations, and *(b)* by replicating or partitioning datasets across servers, thus spreading the load over many access points.

5 Implementation Details

The DisCFS prototype was implemented on OpenBSD 3.1 by creating a user-level NFS server with the access control mechanism described in Section 4. The implementation is portable and does not depend on features unique to OpenBSD.

Administrators must set up initial NFS mountpoints on each client for each server that offers DisCFS services. To construct a secure path to the DisCFS server, the client constructs an IPsec tunnel between the client system and the DisCFS server. We extended the NFS protocol with an RPC procedure that enables the client to attach the remote directory to the mountpoint over the IPsec connection. This allows the DisCFS server to retrieve the public key used for authentication in the IKE [13] protocol (as part of the IPsec key establishment phase) and associate it with a Unix-style *userid*. IPsec protects future NFS requests, allowing the DisCFS server to associate them with the user's public key. Each user on a multi-user client has his own IPsec connection to the server.

After a user executes our RPC attach procedure, the desired directory appears under the client's default DisCFS mount point (*e.g., /discfs*). However, because the user has not yet provided any credentials, the file permissions of the attached directory are set to deny all access. File/directory ownership is set to the *userid* provided by the attach procedure. The *userid* is irrelevant to the DisCFS server, and thus no prior arrangement with the system administrator is needed. Similarly, no file ownership conflicts are possible; the *userid* is only manipulated in this way to make possible the use of unmodified NFS clients.

To get access to the attached directory or any other files/directories in it, the user must have a credential like the one shown in Figure 3. This credential is issued by the administrator (as identified by the public key appearing in the Authorizer field) to a specific user (as identified by the public key appearing in the Licensees field), and contains enough information for the DisCFS server to determine what permissions should be granted to the client system.

A file/directory is identified by a *handle*, which in our prototype implementation is simply the i-node number of the file/directory on the server. A DisCFS server uses this handle to locate the physical file in its local file storage. The handle specifics should change in the future because inode numbers are not suitable as globally unique identifiers across a network. A possible solution would be to build a handle from the inode number and a *generation number*, similar to the 4.4 BSD NFS implementation. In the following discussion we refer to file handles used by DisCFS as *DisCFS file handles* to distinguish them from NFS file handles.

The "Conditions" field can contain additional restrictions *e.g.,* allowing access during working hours, specifying that multiple authorizers are required, *etc.* By combining credentials with different policy assertions we construct a rich access framework that allows fine grained access control.

The credential assertions in our implementation grant standard Unix permissions. The return values for the assertions form a partial order of 8 combinations (“false”, “X”, “W”, “WX”, “R”, “RX”, “RW” and “RWX”) and translate directly into the standard octal representation. Thus, in the credential of Figure 3 the user is granted read, write, and execute access on the *testdir* directory. We wrote a utility which allows a user to submit credential assertions to the DisCFS daemon over RPC. The DisCFS server adds credential assertions to a persistent session. Following this operation, the permissions of the attached directory change accordingly. To improve performance, we use a simple round-robin cache of requested operations and policy results.

The semantics of some of the procedures defined by the NFS protocol change in our implementation. For example, because access control is managed through credential assertions, it makes no sense to use the *setattr* procedure for setting mode bits on a file. There is also a problem with the *create* and *mkdir* procedures. A user can create a file in the attached directory because he has read, write, and execute access. However, he cannot access the newly created file because he does not have a credential assertion for it. Thus, we added our own RPC procedures to the NFSv2 protocol that, upon successful creation of a file/directory, return a credential with full access to the creator of the file. Furthermore, this credential is also added to the user’s active session, so he can immediately use the newly-created filesystem object. The owner can then issue other credentials further delegating access to this file/directory to other users.

6 Experimental Evaluation

The architectural discussion is largely qualitative, and consequences for system performance are useful to understand. The best evaluation would be application-oriented benchmarks for applications in distributed environments, however this is always difficult for a new prototype system. Instead, we use micro-benchmarks and macro-benchmarks to obtain first-order quantification of performance, as well as identification of overhead introduced by the access control mechanism.

Our test hosts are 1 GHz Intel PIII machines with 256 MB of memory and 10 GB Western Digital Protege IDE hard drives. In the two-host, client-server tests that explore the network performance of our system, we connect our machines with 100 Mbps Ethernet. We did not use IPsec for the measurements, because results would vary considerably depending on the specific IPsec con-

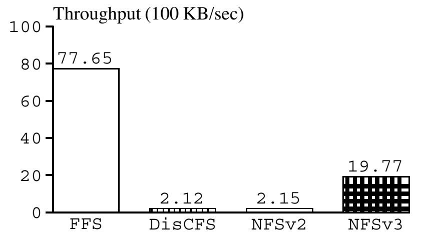

Figure 5: Bonnie Sequential Output (Block). DisCFS throughput is comparable to NFSv2 throughput.

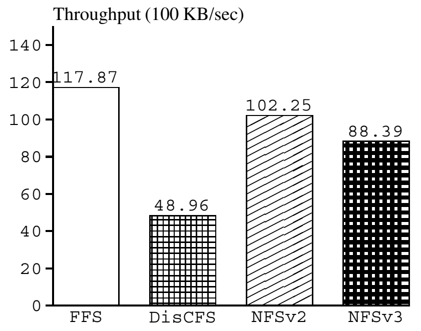

Figure 6: Bonnie Sequential Input (Block). DisCFS throughput is less than 50% of NFSv2 throughput, mostly due to its lack of prefetching.

figuration parameters. For an evaluation of the performance impact of using IPsec in various configurations we refer the reader to [22]. In the following tables, *FFS* means measurements taken on the local file system. *NFSv2* and *NFSv3* denote measurements over NFS protocol versions 2 and 3 respectively using the UDP protocol. We did not measure our performance when running over TCP.

6.1 Micro-benchmarks

We use the *Bonnie* benchmark [1] to evaluate performance when writing and reading a large file. To eliminate caching effects we use a 512MB file, twice the size of main memory. In our results we also include the performance of *FFS* and *NFSv3* for completeness.

Figure 5 presents results for block writes. The perfomance of *DisCFS* is equivalent to that of *NFSv2* because the credential related overheads of *DisCFS* are amortized over the time of the experiment. Remember that this experiment is on a single file.

Figure 6 presents results for block reads. We observe that for read operations *DisCFS* trails *NFSv2* by more than 100%. The reason for this behavior is that *NFSv2* prefetches blocks, an optimization which we have yet to incorporate in the current implementation of *DisCFS*.

Finally, in Figure 7 we experiment with the cost of rewriting blocks. More specifically, in this test Bonnie

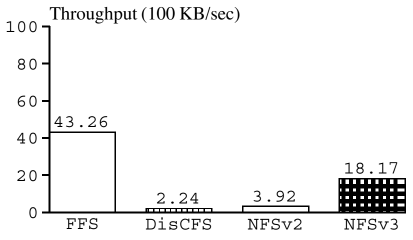

Figure 7: Bonnie Sequential Output (Rewrite). DisCFS throughput is closer to NFSv2 throughput than for reads, but still suffers from its lack of prefetching.

reads a block, dirties it, and then writes it back. As expected, the performance of *DisCFS* is closer to *NFSv2* than in the read benchmarks, however *DisCFS* still suffers from its lack of prefetching.

6.2 Macro-benchmarks

The Bonnie micro-benchmark gives us an idea about the raw performance of DisCFS. However, it does not reflect the access pattern typical of most modern applications. Internet software like electronic mail, netnews and web-based commerce depends on a large number of relatively short-lived files.

To simulate heavy small-file system loads we use the *PostMark* benchmark [2]. PostMark was designed to create a large pool of continually changing files and to measure the transaction rates for a workload approximating a large Internet e-mail server.

We use the default PostMark configuration parameters. The initial number of files created is 500. Files range between 500 bytes and 9.77 kilobytes in size. PostMark then performs 500 transactions on each file. Block sizes for reads and writes are 512 bytes and UNIX buffered I/O is used. We run each PostMark test 10 times and take the average.

We compare three versions of DisCFS to *FFS*, *NFSv2* and *NFSv3*. *DisCFS_NK* is a crippled version of our system offering no security; no KeyNote queries are made. Instead, full access is returned for every file. *DisCFS_COLD* is a fully functional system, but the server was restarted between each successive run of the benchmark. Results for *DisCFS_WARM* reflect the effects of using a cache of 1024 policy results and not restarting the server between successive runs of the benchmark.

Figure 8 shows results for the average creation rate (files/second) for files created before other transactions were performed and the average deletion rate (files/second) for files deleted after all other transactions were performed. When PostMark creates a file, it selects a random initial size, and writes text from a random pool up to the chosen length. File deletion selects a random file from the list of active files and deletes it. The performance of *DisCFS_NK* is approximately equivalent to *NFSv2* when the credential processing overhead is eliminated. *DisCFS_COLD* results show that the credential processing overhead is significant. Performance drops by more than 50%. This is not surprising because upon each file creation the DisCFS server must create a new credential, sign it and evaluate it in the KeyNote session. Upon each deletion a query must determine whether the operation should be permitted. As the DisCFS server begins to service most requests from the cache after the first run of PostMark, numbers for *DisCFS_WARM* return to just below the performance of *DisCFS_NK*.

Figure 9 also presents average file creation and deletion rates, but in this case the files are created and deleted during a sequence of other transactions. As expected,

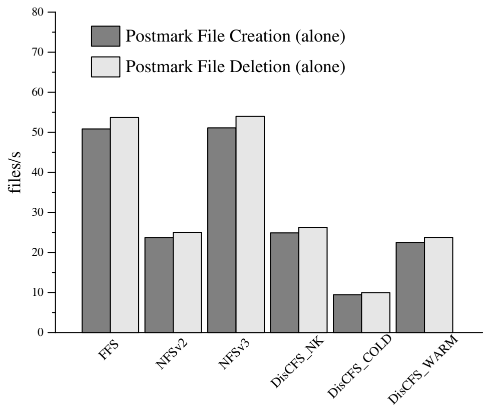

Figure 8: PostMark Average File Creation/Deletion Rate. Without the credential processing overhead, DisCFS performance is comparable to NFSv2. Full KeyNote functionality reduces performance by more than half. If requests are served from a warm cache, the performance is close to NFSv2 again.

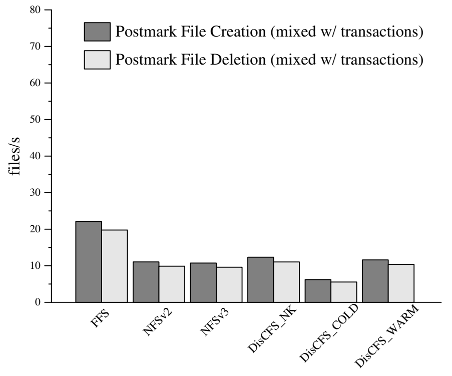

Figure 9: PostMark File Creation/Deletion Mixed w/ Transactions. Performance drops when file create/delete operations are mixed with other transactions. However, DisCFS performance remains the same in relation to NFSv2.

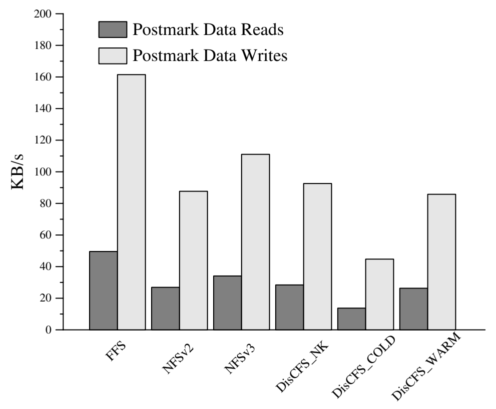

Figure 10: PostMark Data Read/Write Throughput. Throughput for writes is better than for reads due to the buffer cache. DisCFS performance in relation to NFSv2 performance remains the same as in Figures 8 and 9.

overall performance drops compared to the previous isolated case, but *DisCFS* performance remains the same in relation to *NFSv2* performance.

Finally, results presented in Figure 10 reflect system data throughput. For the read test, PostMark opens a randomly selected file and reads the entire file into memory using the configured block size (512 bytes). For the append test, PostMark opens a random file, seeks to the end of it, and writes a random amount of data. As expected, the throughput for writes is better than for reads because writes go through the buffer cache. Performance of *DisCFS* is again comparable to *NFSv2* when the credential overhead is eliminated artificially (*DisCFS_NK*), or by caching (*DisCFS_WARM*). With a cold cache (*DisCFS_COLD*) the performance drops by more than half due to the frequent KeyNote queries.

We use the `top` utility to monitor CPU utilization during the PostMark benchmarks. The *NFSv2*, *NFSv3* and *DisCFS_NK* servers utilize less than 1% of the CPU. Utilization for *DisCFS_COLD* reaches up to 60% during the file creation test due to the number of cryptographic operations the server must perform. Caching brings the number down to 4% for *DisCFS_WARM*.

To explore the overhead of credential handling imposed by real world applications, we time a recursive `grep` for `ifdefs` in every file of the OpenBSD kernel source tree. We conduct the test with a cache size of 128 policy results and a pool of 5000 sessions. The cache contains the permission bits returned by previous policy lookups. The DisCFS server adds credential assertions to a session. Having more sessions helps to distribute them and speeds up the query evaluation. We use three versions of our system: DisCFS with full credential functionality (*DisCFS1*), DisCFS with no signature verification of the

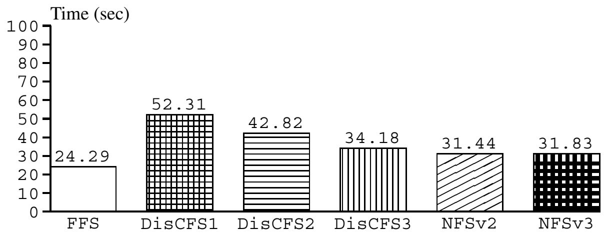

Figure 11: Recursive Grep. The results give us a good break up of the various overheads of DisCFS. Without the credential processing overhead (DisCFS3), performance is close to NFSv2. Credential processing overhead without signature verification is presented by the DisCFS2 numbers. Finally, DisCFS1 numbers show the full KeyNote overhead with signature verification.

credentials (*i.e.,* all credentials are trusted) (*DisCFS2*), and finally a version of DisCFS that does not use credentials at all and just returns full access for every file (*DisCFS3*). We summarize our results in Figure 11.

The total number of files accessed is 5236. By comparing the access times of DisCFS1 and DisCFS3 we see that the overhead per file is less than 3.5 *ms.* This overhead is caused by the credential processing and signature verification. Note that this is a worst-case scenario, since our test accessed every file exactly once. During a normal session, we expect that users will make multiple accesses to the files they have attached, thus amortizing the verification overheads over the lifetime of their session. Additionally, while the DisCFS overhead may be significant in the local area network access scenario, in accesses over a wide area network the verification overheads become less significant.

As a more representative test, we compare the time to compile the OpenBSD kernel over the local filesystem (FFS), NFSv2, NFSv3, and the three versions of DisCFS. This experiment involves access control decisions for approximately 4500 source and 2600 generated files (object and header files, as well as the produced kernel image). We show our results in Figure 12. As expected, the local filesystem is the fastest; however, the cost of the full DisCFS implementation (including signature verification and complete policy evaluation per file access) is negligible, compared to the plain NFSv2 case. (Notice that DisCFS seems to be slighly faster than NFSv2. The primary reason for this is the greatly simplified code running on the DisCFS server, which is effectively a very minimal NFS server.) The overhead due to credential signature verification, *i.e.,* the difference between DisCFS1 and DisCFS2, is just 4 seconds. The overhead of using credentials, *i.e.,* the difference between DisCFS1 and DisCFS3 is about 30 seconds. The access control costs that are more evident in Figure 11 are amortized over the actual operation of the system. This conclu-

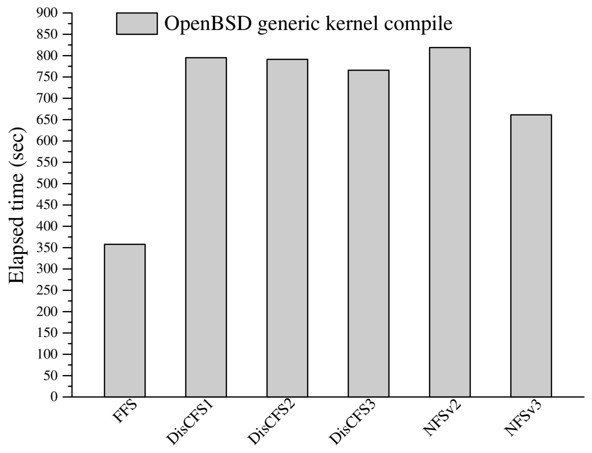

Figure 12: Compilation of the OpenBSD kernel. In this experiment the overheads due to credential evaluation and signature verification have been amortized over the actual operation of the system. Thus, the difference between DisCFS1 and DisCFS3 is only about 30 seconds.

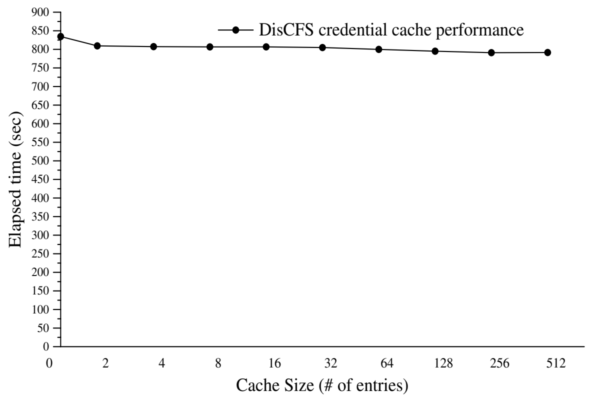

Figure 13: DisCFS credential cache performance. Increasing the cache size results in gradual improvement.

sion matches our previous experience in evaluating the relative costs of security mechanisms: although cryptographic (or other) costs may seem high when viewed in the context of a micro-benchmark, their impact during regular user activities is minimal [22].

In our last experiment, we evaluate how different cache sizes affect DisCFS performance in our compilation experiment. In Figure 13, we see that increasing the cache size results in a gradual improvement, leveling off at about 256 entries. Even limited caching of policy decisions improves performance by more than 5%. Inevitably this will vary depending on the file access patterns and an extensive evaluation of optimum cache size is beyond the scope of this paper. However, we believe that it is beneficial to include even a small cache for keeping recent policy results.

7 User and Administrator Experiences

So far, our first DisCFS prototype has been tested in a small laboratory environment. Although this is not a wide-scale deployment by any means, we learned some important lessons.

7.1 Administrator Experiences

Administrators were happy to be relieved of dealing with users after the initial setup. After an administrator granted a user access to a desired directory with a signed credential, there was little reason for any user to contact an administrator. Further files could be created, and access could be delegated to other users without external administrative intervention. However, initial setup of the system was at times cumbersome. As our first DisCFS prototype was based on NFS, administrators had to deal with setting up initial NFS mountpoints on each client for each server that offered DisCFS services. In hindsight, it would have been better to circumvent the *mountd* protocol. Setting up DisCFS initially also required a good understanding of IPsec configuration.

Administrators also pointed out some security concerns they had with the first prototype. Currently, a KeyNote query is not performed for every nfs call, only on *getattr* calls. While this improves performance, it also means that we trust the client software to enforce the returned UNIX permission bits, something that will change in future releases of DisCFS.

7.2 User Experiences

Users initially thought that creating and signing credentials using a text editor and the KeyNote command-line utilities was somewhat cumbersome (*e.g.,* great care had to be taken with whitespace, so that signatures would be correct). This was quickly addressed with a Perl script that streamlined the process.

Sending credentials for each file over to the server proved to be inconvenient when large numbers of files were involved. A solution offering more end-user transparency is desirable.

8 Future Work

User and administrator experiences with the first prototype of DisCFS suggest that improvements can be made both in ease of use and security. We have started work on a second prototype which should address most issues.

We will eliminate the requirement to administer NFS mount points by including the name of the server hosting a file in the credential. It will then not be necessary to use the *mountd* protocol in order to determine a file's location. KeyNote queries will be made on all NFS calls, so that we do not have to trust the client software.

As mentioned previously, the choice of inode numbers as file handles was not optimal. Consider the case where Bob creates and shares file *song.mp3* with Alice. Bob soon gets tired of the song, deletes it, and proceeds to create another file, *tax_return.dat*. If the inode of *song.mp3* happened to be used for *tax_return.dat*, Alice might be granted access beyond the initial intent. To improve security, the new file handles will not use inodes.

To simplify semantics, our second prototype will not be used for sharing directories. The next version of DisCFS will support sharing of files only. Users will be able to choose what directory structure they want the files to appear in. Manual sending of credentials will be eliminated by using a loopback NFS server or a stackable filesystem layer on the client. When a file containing a credential is encountered (*e.g., Makefile.cred*), the loopback server will translate the name (*Makefile*), read the credential, connect to the remote server specified in the credential, and finally serve the file if the remote server grants access. While this additional layer will hurt performance somewhat, it will greatly enhance the transparency of DisCFS to the end user.

For our first prototype we chose a user-level NFS server because it made development a lot faster. Our benchmarks suggest that the additional context switching overhead compared to a kernel-level server is not substantial. In the future we might try to move the DisCFS server into the kernel which would likely bring about a small improvement in performance at the expense of portability. However, we suspect that larger benefits can be obtained by incorporating some of the NFS optimizations not incorporated in DisCFS (*e.g.,* block prefetching).

DisCFS is based on NFSv2 because we had convenient examples of userland servers implementing it. It would be worthwhile to integrate our authentication and authorization model with NFSv4, *e.g.* as a plugin into to the GSS-API [18].

We used a simple round-robin cache of policy results in our first prototype. We would like to implement a practical cache replacement algorithm and experiment more to find an optimal cache size.

Unix permission bits can be inflexible and limiting. Future versions of DisCFS could take advantage of the flexibility of credentials to provide more fine-grained access control, *e.g.,* allow appending to a file but no truncation.

Looking further forward, new file-sharing policies are achievable with DisCFS beyond the Unix NFS support demonstrated. An example of such use would be enabling controlled access to file storage for the untrusted users that are characteristic of the Web. DisCFS should also offer advantages for emerging P2P systems, as its avoidance of centralized control is well-suited to cooperative resource-sharing at a large scale and with disparate administrative models. Finally, while DisCFS's decentralized control should result in "scalability", this assertion requires robust quantitative modeling and supporting measurement.

9 Conclusions

This paper introduced a completely credential-based mechanism for authentication and access control of files. This is a new approach to distributed file systems, as it separates the *policy* for controlling the file from the access control *mechanism* used by the underlying file storage. This gives DisCFS advantages in flexibility and scalability over traditional file systems, and even over some recent secure file system efforts.

The DisCFS prototype implementation combines a credential-based access control system with common Unix file operations. It is straightforward to implement and deploy DisCFS because it uses components that exist in common operating systems, such as NFS and IPsec, and supports the traditional Unix filesystem semantics.

The system's performance was evaluated with both micro- and macro-benchmarks. The performance of individual DisCFS operations is bounded by that of the same primitives, such as remote RPC times, which limit the performance of other distributed systems. Used in larger contexts such as software builds or file-tree searches (where many files are "touched" sequentially, a worst case for DisCFS) the performance impact of DisCFS's enhancements is relatively low. In normal usage, the DisCFS-imposed overhead is negligible.

DisCFS source code is available for download at *http://www.seas.upenn.edu/~miltchev/discfs/*.

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

References

[1] Bonnie Filesystem Performance Benchmark. http://www.textuality.com/bonnie/.

[2] PostMark Filesystem Performance Benchmark. http://www.netapp.com/tech_library/3022.html.

[3] Eshwar Belani, Amin Vahdat, Thomas Anderson, and Michael Dahlin. The CRISIS Wide Area Security Architecture. In *Proceedings of the USENIX Security Symposium*, pages 15–30, August 1998.

[4] M. Blaze. A Cryptographic File System for Unix. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, November 1993.

[5] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. RFC 2704, September 1999.

[6] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of the 17th Symposium on Security and Privacy*, pages 164–173, 1996.

[7] CCITT. *X.509: The Directory Authentication Framework*. International Telecommunications Union, 1989.

[8] S. P. Dyer. The Hesiod Name Server. In *Proceedings of the USENIX Winter Technical Conference*, pages 183–190, 1988.

[9] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC (Proposed Standard) 2693, Internet Engineering Task Force, September 1999.

[10] J. Satran et al. IPS Internet Draft - iSCSI. Internet Draft, Internet Engineering Task Force, Septermber 2001.

[11] K. Fu, E. Sit, and N. Faemster. Dos and Don’ts of Client Authentication on the Web. In *Proceedings of the USENIX Security Symposium*, pages 251–269, August 2001.

[12] C.A. Gunter and T. Jim. Policy-directed certificate retrieval. *SP&E*, 30(15):1609–1640, 2000.

[13] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC (Proposed Standard) 2409, Internet Engineering Task Force, November 1998.

[14] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[15] J. Schönwälder and H. Langendörfer. Administration of large distributed UNIX LANs with BONES. In *Proceedings of the World Conference On Tools and Techniques for System Administration, Networking, and Security*, April 1993.

[16] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC (Proposed Standard) 2401, Internet Engineering Task Force, November 1998.

[17] A. D. Keromytis. *STRONGMAN: A Scalable Solution to Trust Management in Networks*. PhD thesis, University of Pennsylvania, December 2001.

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

[19] D. Mazieres and M.F. Kaashoek. Secure Applications Need Flexible Operating Systems. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems*, May 1997.

[20] D. Mazieres, M. Kaminsky, M. Frans Kaashoek, and E. Witchel. Separating key management from file system security. In *Symposium on Operating Systems Principles (SOSP)*, pages 124–139, December 1999.

[21] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos Authentication and Authorization System. Technical report, MIT, December 1987.

[22] S. Miltchev, S. Ioannidis, and A.D. Keromytis. A Study of the Relative Costs of Network Security Protocols. In *Proceedings of the Annual USENIX Technical Conference, Freenix Track*, pages 41–48, June 2001.

[23] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Bayou: Replicated Database Services for World-Wide Applications. In *Proceedings of the 7th ACM SIGOPS European Workshop*, 1996.

[24] J. Regan and C. Jensen. Capability File Names: Separating Authorization from User Management in an Internet File System. In *Proceedings of the USENIX Security Symposium*, pages 211–233, August 2001.

[25] E. Riedel, M. Kallahalla, and R. Swaminathan. A framework for evaluating storage system security. In *Proceedings of the 1st Conference on File and Storage Technologies (FAST)*, January 2002.

[26] M. A. Rosenstein, D. E. Geer Jr., and P. J. Levine. The Athena Service Management System. In *Proceedings of the Winter USENIX Conference*, pages 203–212, 1988.

[27] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network File System. In *Proceedings of the Summer USENIX Conference*, June 1985.

[28] D.B. Terry, M.M. Theimer, K. Petersen, A.J. Demers, M.J. Spreitzer, and C.H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, December 1995.

[29] A. Vahdat. *Operating System Services for Wide-Area Applications*. PhD thesis, UC Berkeley, December 1998.

[30] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. *ACM Transactions on Computer Systems*, 12(1):3–32, 1994.