Secure and Flexible Global File Sharing *

Stefan Miltchev, Vassilis Prevelakis, Sotiris Ioannidis, and Jonatham M. Smith
{miltchev,vassilip,sotiris,jms}@dsl.cis.upenn.edu
CIS Department
University of Pennsylvania

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

Abstract

Sharing of files is a major application of computer networks, with examples ranging from LAN-based network file systems to wide-area applications such as use of version control systems in distributed software development. Identification, authentication and access control are much more challenging in this complex large-scale distributed environment.

In this paper, we introduce the Distributed Credential Filesystem (DisCFS). Under DisCFS, credentials are used 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. As with traditional *capabilities*, users can delegate access rights (and thus share information) simply by issuing new credentials. Credentials allow files to be accessed by remote users that are not known *a priori* to the server. Our design achieves an elegant separation of policy and mechanism which is mirrored in the implementation.

Our prototype implementation of DisCFS runs under FreeBSD, using a modified user-level NFS server. Our measurements suggest that flexible and secure file sharing can be made scalable at a surprisingly low performance cost.

*This work was supported by DARPA under Contract F39502-99-1-0512-MOD P0001.

Keywords: Filesystems, access control, Trust Management, KeyNote, credentials.

1 Introduction

The promise of the Internet is that of global information sharing and collaboration. However, existing systems have not kept up with the rapid expansion of the Internet. As a result systems are either primarily one-way (*e.g.* Web where the number of readers far outnumbers the number of writers), or small scale (*e.g.* NFS where only a relatively small group of users defined by the local authentication database may have read/write access to files).

This is due to the fact that current access control systems still rely on authentication which in turn requires that the user is known to the system. This concept works only in closed administrative domains (*e.g.* NIS domains or Kerberos realms [18]). Under these conditions, the administrator creates a user account and assigns access rights to it.

Let us consider the scenario where Alice, a local user wishes to share files with Bob who does not have an account. The obvious way is to ask the local system administrator to open an account for Bob, but may create administrative and legal problems and in any case may conflict with local policies (*e.g.* only employees may have accounts).

The traditional approach is to assume that Bob has an account on some other domain and then attempt to merge the authentication databases of the two domains. Proposals for global authentication hierarchies (*e.g.* X.500), or mechanisms for cross certifying Certification Authorities only make matters worse by introducing new problems that dwarf the original.

Our approach is far simpler: Alice authorizes Bob's public key to have access to the file. This is done by having Alice create a credential that contains Bob's key, the file handle and the permissions. The credential is signed by Alice, and confers to Bob the authority to access the file. By combining Alice's credential with one signed by himself, Bob may further delegate access to the file.

The authorization credential may be simply emailed to Bob as cleartext since the credential itself does not contain any secrets (apart from the information that Alice wishes to share a file with Bob).

We will 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. Access to the files may be monitored by the system and the entity issuing the requests may be identified through its public key. Mechanisms for restricting access or imposing access controls are also provided.

We have integrated our access mechanism with a user-level NFSv2 server running on many Unix systems including FreeBSD [2]. The performance measurements collected by running common file related benchmarks indicate that our approach is very efficient. We endeavor to eventually offer this access mechanism as part of the standard NFS authentication framework.

Organization The paper is organized into six further sections. The next section provides a more complete motivation for our system. Section 3 discusses related work. Sections 4 and 5 describe our design and implementation under the FreeBSD operating system. Section 6 evaluates the system using both micro- and macro-benchmarks. Finally Section 7 concludes the paper with a summary of our results and our future plans.

2 Motivation

Let us consider two typical examples of information sharing. In the first case, Bob, a salesman, would like his ten best clients to be able to have access to advance information about a product. He does not want other clients or the general public to have access to this information.

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

In both cases we have local users (Bob and Alice), known to the system, who wish to provide access to other external users. For this type of activity to be feasible, the following conditions must be met:

- Scaling. The system should be able to cope with large numbers of files and even larger number of users accessing those files.
- There should be no involvement of the administrators in the process of allowing external users access to files in the system. Local users should be able to authorize access to files by external users.
- The file access conditions must be flexible and expandable. In any case there should be no constrains by the access mechanism as to what conditions may be imposed for access.
- Delegation is extremely important for the operation of the system, since there is already an implicit delegation of access authority from the administrators to the local users and from the local users to external users.

- Default policy. Most sites have access policies and these must be enforced regardless of the actions of individual users. The administrator should be able to specify the 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 in multiple servers.

Existing systems have several major shortcomings when used to carry out information sharing tasks:

First, traditional user authentication implies that the user is known to the system, before file requests can be processed. However, the commonly used information access model on the Web is that browsers can download pages from Web servers without prior registration (*i.e.*, anonymously).

Second, file and directory permissions are inherited from multi-user computer operating systems. Sharing is achieved by either account sharing (which is extremely ill-advised, as it defeats accountability) or through the use of group access permissions on files and directories. Such permissions lack flexibility and granularity, and perhaps most importantly, extensibility: there is no way of adding new permissions if the existing ones prove inadequate.

In the salesman example, since the information is not intended to be widely available, Bob will have to 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 are created and handed over to the users. A more sophisticated way of achieving the same goal would be to use X.509 [7] credentials for user authentication. While this approach addresses some of the well-known security problems of password authentication, it leaves much to be desired in terms of flexibility and required administrative intervention.

Faced with complex and inflexible mechanisms some sites abandon access restrictions relying 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 [15].

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

3 Related Work

Network file sharing is an area that has attracted a lot of attention given the need for information exchange. The explosion in the growth of the Internet over the past several years, and the projections that the growth will continue at a similar pace, makes file sharing an even more important issue. There are however a number of problems in the proposed and already existing sharing mechanisms.

3.1 File Systems

Network file systems, like NFS and AFS [21, 13] are the most popular and widespread mechanisms for sharing files in tight administration domains. However, crossing administrative boundaries creates numerous administrative problems (*e.g.,* merging distinct Kerberos realms or NIS domains).

Encrypting file systems like CFS [3] 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 somehow communicate the secret encryption key for the file to all the users that 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.*). Our system assumes that the server is trustworthy, so that the files can be stored in clear text. CFS-like encryption mechanisms may still be used on top of DisCFS.

WebFS is part of the larger WebOS [22] 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. Access control lists (ACLs) are associated with each file that enumerate users who have read, write, or execute permission on individual files. Users are uniquely identified by their public keys. We have taken a more general and scalable approach in that there is no need for ACLs since each credential is sufficient to identify both the users and their privileges.

The capafs system [20] has a rationale that is very close to that of our system. However, the authors provide a very weak security system for file access by encoding the access capabilities in the file name. Apart from producing horrible file names (incomprehensible strings of alphanumeric characters necessitating the use of symbolic links to convert them to something the user can actually type in), knowledge of the file name provides access to the file. Thus, file names themselves must be protected raising the question of how the file names may be communicated between remote users. Moreover, 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 [16]. 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, since SFS relies on a trusted third party to mutually authenticate server and client, collaboration is possible only if the client the server have a common root for their Certification Authorities. Our 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.

3.2 Operating Systems

The Taos operating system [23] uses a *narrow* API to control access to security service, *e.g.* the file system. It makes use of credentials for delegation of access rights between principals. DisCFS utilizes these well known techniques to extend a real world protocol such as NFS. We are not operating system dependent, and any NFS client can utilize our system. This makes our approach more flexible than specialized operating system mechanisms.

The concept of credential-based access control appears also in the Exokernel [17]. 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 rather limited by the fact that permissions are hardwired into the system, the hierarchical capability tree may be up to 8 levels deep, and the access-list based control mechanism is inflexible. 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 [19, 10]. Anonymous FTP, where there is no need for authentication, offers total flexibility since 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 non-critical. In the case where authentication is required, the flexibility is reduced to an absolute minimum. The only users allowed to access the server, in that case, are users that are already known to the system. This, as is the case with existing network file systems, limits the collaboration possibilities only between users in the same administration domain.

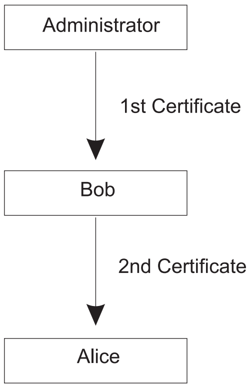

Figure 1: **Delegation of privileges, from the administrator to Bob, and then to Alice.**

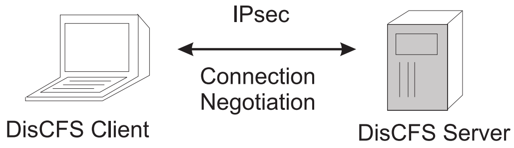

Figure 2: **Client establishes IPsec connection with DisCFS server.**

4 DisCFS Design

4.1 System Architecture

The basic principle behind DisCFS is *Trust Management* [5, 6, 4]. Trust Management dispenses with unique names as an indirect means for performing access control. Instead, it uses a direct binding between a public key and a set of authorizations. This results in an extremely decentralized authorization system that is flexible enough to cope with a large variety of authentication scenarios.

Rather than having users authenticated by the system and then checking access lists to see whether their requests should be honored or not, our system is based entirely on keys and authorizations. User requests are signed by the user's key and must be accompanied by other credentials that form a chain of trust linking the user's key to a key that is trusted by the system. In our first example in Section 1, we looked at Bob's predicament in trying to allow his clients access to internal files. Utilizing a trust management system, the server would trust only the administrator's key. Bob will be given a credential that binds Bob's key with the files in question and is signed by the administrator. The credential may allow Bob read and write access to the files.

If Bob then wishes Alice to be able to only read these files, he will simply need to create a new credential which will grant Alice's key read access to the files. Alice will issue a request signed by her key. If Alice's request is to be honored by the system, it has to be accompanied by Bob's credential. This credential forms a link between the external user (Alice) and the internal user (Bob). Bob's own credential (issued by the administrator) must also be available, to link the internal user to the administrator. Thus, Alice's request must be accompanied by both credentials in order to be granted (see Figure 1). Credential caching may be used to reduce the number of credentials that have to be exchanged.

It is interesting to observe that in DisCFS the traditional problem of credential (or certificate) revocation is fairly straightforward to address: since the credentials related to a specific file have to be examined by the DisCFS server where the file is stored, revocation (especially if it is infrequent) can be done 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.

In order to be able 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 [5], KeyNote [4], QCM [11] and SPKI [9, 8]. In our system we use the KeyNote trust management system [4] for this purpose.

4.2 KeyNote in DisCFS

The basic service provided by the KeyNote system is compliance checking; that is, checking whether a proposed action conforms to policy. Actions in KeyNote are specified as a set of name-value pairs, called an action attribute set. Policies are written in the KeyNote assertion language and either accept or reject action attribute sets presented to it (non-binary results are also possible). Policies can be broken up and distributed as credentials, which are signed assertions that can be sent over a network and to which a local policy can defer in 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.

The advantage of using Keynote is that we no longer need 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. We also provide our users with the ability to propagate access to the files by simply passing on (delegating) their rights to other users. In this way users pass credentials rather than passwords, thus allowing the system to associate access requests with keys and also to be able to reconstruct the authorization path from the administrator to the user making the request. The system may not know that Alice is trying to get at a file, but it can log that key A (Alice's key) was used and that key B (Bob's key) authorized the operation.

During the writing of this paper we encountered one obvious application of our system. The administrator of the host that we were using for editing the paper had failed to create a group for all of us. Since we could not find a group that we all belonged to, the only way for all of us to be able to access the CVS repository with the files was to make them world writable. If the central server supported DisCFS then the owner of the repository would simply need to issue read-write certificates to all the other authors.

4.3 DisCFS over NFS

As the actual network filesystem we use NFS. This allows for easy integration into existing systems without need for extensive upgrades. Moreover, the entire scheme works with both monolithic and distributed servers. Since the servers do not need to share information about users, there is no synchronization overhead. Each repository is responsible for only the part of the distributed filesystem that is stored locally and there is no need to distribute and synchronize authentication and access control databases (like NIS).

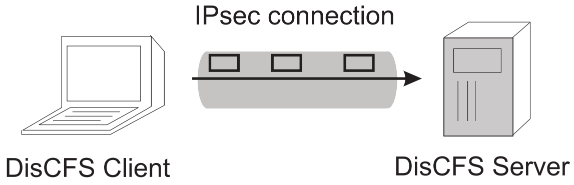

Figure 3: **Client sends file-related credentials to the DisCFS server; file becomes visible to client.**

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 modifications to the operating system kernel. This is particularly useful since it is not always possible to have access to the operating system source.

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 [14] for the connection between the client and the server (as shown in Figure 2) thus ensuring the following:

- User authentication is handled through the creation of the IPsec Security Associations between the client and the server.
- A secure link between the client and the server is established so that subsequent communications are secure.
- All requests coming over the IPsec link can be safely assumed to come from the authorized user.

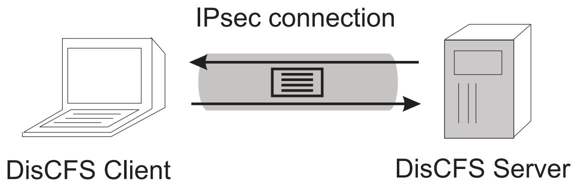

Figure 4: **Client sends read requests, server sends file blocks to client if policy allows the operation.**

When a file is stored in DisCFS, a credential is generated containing information that allows the future retrieval of the file contents as well as information about the file creator. Since the entire 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. It is important to note that without the credential, retrieval of the file is not possible.

Once the user submits the necessary file credentials (Figure 3, the file will appear 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 (Figure 4). The system also permits the user to override the default file name and allows files to be placed in user-specified locations.

5 Implementation Details

We built our implementation of DisCFS by modifying the existing user-level daemon of the cryptographic file system CFS [3]. In the prototype, we replaced the encryption functionality of CFS with the access control mechanism described in Section 4. For our platform, we used FreeBSD 4.3-RELEASE [2]. However, the implementation is fairly portable across different systems and we already have a working OpenBSD port.

The main task in implementing DisCFS was the integration of KeyNote credentials with NFS. To that end, we used a modified version of the CFS *cattach* utility that sets up an IPsec tunnel between the client system and the DisCFS server and attaches the remote directory over the IPsec connection. This allows the DisCFS server retrieve the public key used for authentication in the IKE [12] protocol (as part of the IPsec key establishment phase) and associate it with a unix-style *userid*. Future NFS requests are protected with IPsec, allowing the DisCFS server to associate them with the public key of the user.

As a result of the attach operation, the desired directory would appear under the default DisCFS mount point (*e.g., /discfs*). However, since the user has not provided a KeyNote credential assertion, the file permissions of the attached directory are set to 000 (meaning no access is granted). The file/directory ownership is set to the *userid* provided during the attach operation. This value has no local significance for the DisCFS server, and thus no prior arrangement with the system administrator is needed. Similarly, no file ownership conflicts are possible; the *userid* is irrelevant to the DisCFS server, and is only manipulated in this way to make possible the use of unmodified NFS clients.

To get any privileges to the attached directory or any other files/directories in it, the user would have to have a credential like the one shown in Figure 5. This credential was 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 inode number of the file/directory on the server. This handle is used by the DisCFS server to locate the actual file in its local file storage. The handle specifics need to be changed in the future since inodes are not suitable as globally unique identifier 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.

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

```
Authorizer: "dsa-hex:3081de0240503ca3b98b754259d8b3bdd6ed3960"
Licensees: "dsa-hex:3081de02405be60a70c5321e7fd20fd4d0d2a4f6"
Conditions:     (app_domain == "DisCFS") &&
                (HANDLE == "666240") -> "RWX";
Comment:        "testdir"
signature: "sig-dsa-sha1-hex:302e021500eeb15af1a109800171649"
```

Figure 5: **KeyNote credential granting user *miltchev* access to directory *testdir*. The keys and signatures have been truncated in the interest of readability.**

octal representation. Thus, in the credential of Figure 5 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. Succesfully submitted credential assertions are added to a persistent KeyNote session. Following this operation, the permissions of the attached directory are changed accordingly. When read or write operations occur however, the KeyNote is consulted again on whether the specific requests should be granted; thus, the DisCFS server does not have to trust the client to enforce the file permissions. To improve performance, we use a cache of requested operations and policy results.

It should also be noted that some of the procedures defined by the NFS protocol do not make semantic sense for our implementation. For example, since access control is managed through credential assertions the *setattr* procedure becomes superfluous. The careful reader will also notice that there is a problem with the *create* and *mkdir* procedures. A user could create a file in the attached directory since he has read, write, and execute access. However, he would not be able to access the newly created file since he would not have a credential assertion for it. Thus, we had to add our own procedures that upon successfull creation of a file/directory return a credential with full access to the creator of the file. The owner can then issue other credentials further delegating access to this file/directory.

6 Experimental Evaluation

While the architectural discussion is largely qualitative, some estimates of the system performance are useful. With a design such as this, the most useful data would be system benchmarks for applications in distributed environments. We performed several experiments, both micro-benchmarks and macro-benchmarks, to get a quantitative evaluation. The experiments are focused on any possible performance overheads introduced by our access control mechanism.

Our test machines are x86 architecture machines running FreeBSD 4.3-RELEASE and interconnected by 100 Mbps Ethernet. More specifically, in the two-host tests (source to sink) that explore the network performance of our system, Hera and Zeus are 1 GHz Intel PIII machines with 256 MB of memory and 10 GB Western Digital Protege IDE hard drives. The single host tests, that explore the storage performance of our system were performed on Hera. Our prototype system is running on Hera, with Zeus playing the role of the client.

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. Finally *DisCFS* is our prototype.

6.1 Micro-benchmarks

The *Bonnie* benchmark [1] was used in order to evaluate the performance when writing and reading a very large file (512MB). Figures 6, 7, and 8 present results for single-character writes, block writes and re-writes respectively. The results for single character reads and block reads are presented in Figures 9 and 10. The performance advantage of the local file system (*FFS*) comes as no surprise. This benchmark also demonstrates that the write performance of the userland *DisCFS* implementation is comparable to that of *NFSv2*. The read performance of *DisCFS* lags behind *NFS*, which can probably be attributed to the more advanced caching performed by the latter.

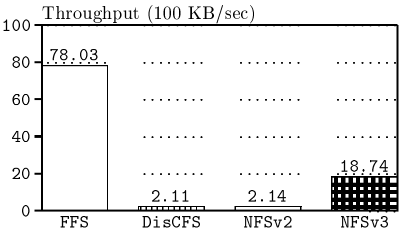

Figure 6: **Bonnie Sequential Output (Char)**

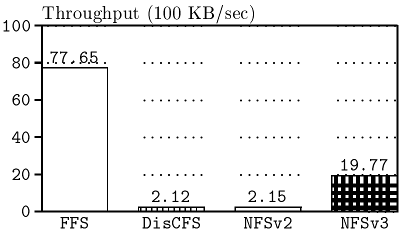

Figure 7: **Bonnie Sequential Output (Block)**

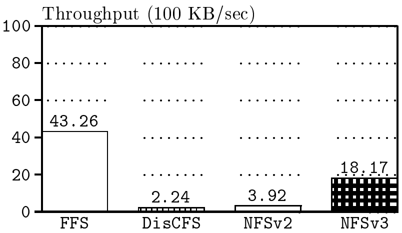

Figure 8: **Bonnie Sequential Output (Rewrite)**

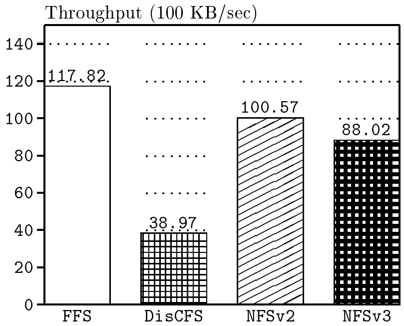

Figure 9: **Bonnie Sequential Input (Char)**

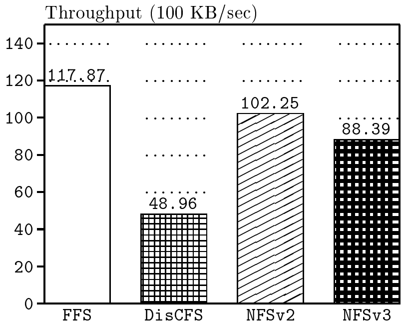

Figure 10: **Bonnie Sequential Input (Block)**

6.2 Macro-benchmark

While the micro-benchmarks give us an idea about the raw read and write performance of *DisCFS* they do not provide much insight into the overhead imposed by credential verification and evaluation. The Bonnie benchmark only operates on one file and thus only one credential needs to be verified and evaluated. To get a better idea about the overhead of credential handling we timed a recursive grep for ifdefs in every file of the FreeBSD kernel source tree. The test was conducted with a cache size of 128 policy results and a pool of 5000 keynote sessions. To gain better insight into the overhead of credential handling we used three versions of our system: DisCFS with full credential functionality (*DisCFS1*), DisCFS with no credential signature verification (i.e. all credentials are trusted) (*DisCFS2*), and finally a hacked DisCFS which did not use credentials at all and just returned full access for every file (*DisCFS3*). *DisCFS3* can be thought of as *CFS* without the encryption functionality. The results are presented in Figure 11.

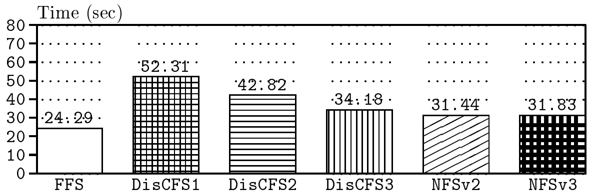

Figure 11: **Recursive Grep**

7 Conclusions

There are three major contributions of this paper.

First, we have introduced the idea of a completely credential-based mechanism for authentication and access control of files. We argue that this design is a fundamental improvement, as it *completely* separates the *policy* for controlling the file (*i.e.,* its associated users and access rights) from the access control *mechanism* used by the underlying file storage. As we have argued in the paper, this gives DisCFS advantages in flexibility, security and scalability relative to previous designs.

Second, we have described our DisCFS prototype, which is based on FreeBSD 4.3-RELEASE and CFS [3]. The implementation uses the KeyNote trust management system as the basis for robust scalable credential management. It supports common unix file operations. The prototype shows that it is remarkably easy to both implement and deploy DisCFS, as it uses components such as NFS and IPsec, which already exist in most common operating systems. Furthermore, the traditional semantics of the unix filesystem can easily be supported by DisCFS.

Third, we evaluated the system's performance with a set of micro-benchmarks which measured primitive operations in the context of our access control mechanism. This demonstrated that DisCFS was constrained by the same factors, such as remote RPC times, which plague other distributed systems. In a second evaluation, we compared the performance of DisCFS to CFS, a more "macro" benchmark, and showed that the performance impact of DisCFS's enhancements is low.

Among the directions we will pursue for future work are investigation of new file sharing policies for unusual scenarios, such as the untrusted users characteristic of the WWW, and attempting to rigorously quantify the scalability advantages offered by DisCFS.

References

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

[2] The FreeBSD Operating System. http://www.freebsd.org/.

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

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

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, Los Alamitos, 1996.

[6] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance Checking in the PolicyMaker Trust-Management System. In *Proc. of the Financial Cryptography '98, Lecture Notes in Computer Science, vol. 1465*, pages 254–274. Springer, Berlin, 1998.

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

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

[9] Carl Ellison. SPKI Requirements. Request for Comments (Proposed Standard) 2692, Internet Engineering Task Force, September 1999.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1997.

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

[12] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force, November 1998.

[13] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

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

[15] Emil Sit Kevin Fu and Nick Faemster. Dos and Don'ts of Client Authentication on the Web. In *Proceedings of the 2001 USENIX Security Symposium*, pages 251–269, Boston, MA, August 2001.

[16] David Mazieres, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Symposium on Operating Systems Principles*, pages 124–139, 1999.

[17] David Mazieres and M. Frans Kasshoek. Secure Applications Need Flexible Operating Systems. In *The 6th Workshop on Hot Topics in Operating Systems*, May 1997.

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

[19] J.B. Postel and J. Reynolds. File Transfer Protocol, 1985.

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

[21] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the sun network file system. In *Proceedings of the 1985 Summer Usenix Conference*, Portland, OR, June 1985.

[22] Amin Vahdat. *Operating System Services for Wide-Area Applications*. PhD thesis, University of California, Berkeley, December 1998.

[23] Edward Wobber, Martín Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, 1994.