# Strong Security for Distributed File Systems

Ethan Miller Darrell Long William Freeman Benjamin Reed
University of California, Santa Cruz TRW IBM Research

**Abstract**

*We have developed a scheme to secure network-attached storage systems against many types of attacks. Our system uses strong cryptography to hide data from unauthorized users; someone gaining complete access to a disk cannot obtain any useful data from the system, and backups can be done without allowing the superuser access to unencrypted data. While denial-of-service attacks cannot be prevented, our system detects forged data. The system was developed using a raw disk, and can be integrated into common file systems.*

*We discuss the design and security tradeoffs such a distributed file system makes. Our design guards against both remote intruders and those who gain physical access to the disk, using just enough security to thwart both types of attacks. This security can be achieved with little penalty to performance. We discuss the security operations that are necessary for each type of operation, and show that there is no longer any reason not to include strong encryption and authentication in network file systems.*

## 1. Introduction

Computer storage is an increasingly important part of the Internet, and ensuring the security and integrity of stored data is a crucial problem. Attacks by intruders and insiders have led to billions of dollars in lost revenue and expended effort to fix the problems. Most organizations rely heavily on their distributed computing environment, which usually consists of workstations and a shared file system. This file system is often stored on a centralized file server that is managed by a system administrator who has access to the whole file system, leaving the data vulnerable to anyone who can prove (legitimately or otherwise) that he is the administrator.

Recently, however, network-attached disks have begun to replace traditional centralized storage systems [7]. In such systems, disks are attached directly to a network, and rely upon their own security rather than using the server's protection. This arrangement makes security more difficult because the disk is directly exposed to potential attacks instead of being hidden behind a single server that can be "hardened."

We have developed a security system for network-attached storage that relies upon strong cryptography to protect data stored in a distributed file system. Our system stores and transfers all data encrypted, only decrypting it at a client workstation. The drives themselves lack sufficient information to decrypt the data they hold or to undetectably forge new data, so physically stealing the media will not enable an attacker to gain access to the data or to plant false data. Similarly, an administrator backing up the file system only has access to encrypted copies of the files; the authorized users of a particular file are the only ones with access to its unencrypted contents.

Despite this level of security, our system imposes little overhead on the file system. The disks need only perform secure hashes, with encryption and decryption done at the client. This arrangement allows the system to be built using inexpensive embedded processors for the disks, leaving the computationally intensive task of encryption and decryption to the relatively faster clients.

We begin by summarizing previous work in securing file systems, discussing the strengths and weaknesses of each system. We then describe Secure Network-Attached Disks (SNAD), our system for protecting data on network-attached disks. We then present three alternate security schemes, each appropriate for different levels of client and server CPU performance. Finally, we conclude with a description of our plans for integrating strong security into modern distributed file systems.

## 2. Related Work

Many systems have been designed to address the security problems of modern distributed file systems. However, these systems have suffered either from weak security, poor performance, or both. It is only recently that CPU performance has advanced to the point where strong cryptography can be done quickly with inexpensive processors. This allows its use on low-cost proces-

This paper appeared in the 20th International Performance, Computing, and Communications Conference (IPCCC 2001), Phoenix, AZ, April 2001.

sors that can be associated with each disk in a distributed file system [7].

Most file systems include some measure of security. However, systems such as NFS [13] and xFS [1] pass most of their data in the clear, generally relying on relatively insecure networks and trusted hosts for data protection. Other systems, such as AFS [9] and NASD (Network Attached Secure Disk) [7] use third-party authentication such as Kerberos [11] to provide security. These systems provide stronger security by requiring users to obtain "tickets" from a third party. These systems are considerably stronger than those that rely upon simple authentication, but they still store files in the clear.

SCARED [12] and the network-attached disks described by Gobioff [8] also use cryptographic techniques to authenticate users of remote network storage. The SCARED design supports the use of end-to-end encryption of data in the network, but does not encrypt data on the disk itself. Similarly, Gobioff leaves encryption of stored data to the application level of the software. We use similar mechanisms in our design, but with the added safety of encryption in the file system itself, guaranteeing that data on disk cannot be read by an intruder who gains access to the disk.

The Secure File System (SFS) [6] provides strong authentication and a secure channel for communications, including an extensive authentication mechanism for individual users and strong security for data in transit between clients and servers. However, it still relies upon trusted file servers that do not alter data stored on them and may allow an intruder to recover data by physically compromising the server.

Though many file systems use authentication, few file systems actually protect data disk. Many users have implemented their own "secure file system" by simply encrypting their files using standard encryption software. While this can provide security and integrity guarantees, it is an *ad hoc* mechanism, and does not deal with issues such as sharing files between users.

The Cryptographic File System (CFS) developed at AT&T Bell Laboratories [2,3] encrypts all data and potentially sensitive metadata stored on disk. A user who supplied the correct password could read files in the directory. However, CFS also required that the server be trusted to "actually store (and eventually return) the bits that were originally sent to it." In the Internet era, there is no guarantee that a server will do this, so there must be a mechanism to ensure that the server has not maliciously altered the data. In addition, CFS does not discuss mechanisms for distributing keys among users for sharing files.

# 3. System Design

The goal of our system is to address the security shortcomings of previous file systems while preserving the flexibility and performance of standard distributed file systems.

We propose three security alternatives for network-attached storage; these schemes vary the CPU time required at both the client and the server, with more security requiring more CPU cycles. The first scheme places the highest demand on both the client and the server, but provides the strongest security. The second scheme removes the need for the server to check signatures (a computationally expensive task), but requires the clients to both generate and check signatures. The third scheme eliminates signature check and verification completely, at the expense of a slight loss in security.

Given current (and near future) CPU speeds, we believe that the third scheme will be the best for current file systems, but the first two schemes will become more attractive as CPU speeds increase.

## 3.1. Design Goals

Our security schemes provide several important features for a secure file system. The first feature is end-to-end encryption of all file system data and metadata, including storage on disk. This is necessary to restrict access to cleartext to only authorized users while still allowing administrators and backup systems to do their jobs. The disk must not contain sufficient information to decrypt the data stored on it. Rather, files should only exist in cleartext on the client.

Ensuring data integrity is a second goal. A user reading data from the server must be sure that the files received are those he originally stored; data modified at the disk or introduced into the system by an malicious intruder must be detectable. Storing a non-linear checksum of the cleartext in a block along with the ciphertext allows any authorized user to detect a change made to the encrypted block by an intruder.

Flexibility is a third feature that is desirable in a secure file system. While it would certainly be possible to simply encrypt each file with a user's password, this approach is impractical because it makes file sharing difficult. Instead, a file system should have sharing at least as powerful as that in standard Unix, and preferably as flexible as the access control lists provided by AFS [9].

High performance and scalability is the fourth feature desirable for a secure distributed file system. Though it may be possible to build a secure file system,

no one will use it if the file system is too slow. If encryption and decryption are performed at the client, encryption throughput will limit a single client's bandwidth. Minimizing the effort required by the network-attached disk's CPU allows a distributed file system to be used by hundreds of clients, each of which can decrypt the data intended for itself.

## 3.2. Basic Mechanisms

The basic mechanism behind SNAD is to encrypt all data at the client and give the server sufficient information to authenticate the writer and the reader sufficient information to verify the end-to-end integrity of the data.

SNAD relies upon several standard cryptographic tools. The client uses a standard algorithm such as RC5 [14] or Blowfish [14] to encrypt the data, ensuring that the data is unreadable by anyone until it is decrypted by the client that reads it. Public-key cryptography is used to allow disks to store information that can be used to decrypt their files; because public-key encryption is asymmetric, however, only a user with the appropriate private key can use this information.

SNAD also makes extensive use of cryptographic hashes and keyed hashes. Cryptographic hashes such as MD5 and SHA-1 [14] use a one-way function to compute a large number (128 or 160 bits) from a block of data. Any modification in the input data will cause the resulting hash value to change. It is currently believed NP-hard to find two input texts that result in the same hash value.

Keyed hashes such as HMAC (hashed message authentication code) [10] use a cryptographic hash in conjunction with a shared secret to check integrity and authenticate a writer. If the sender and receiver share a key, the key can be included in the cryptographic hash, preventing anyone who intercepts the data from undetectably modifying it unless they know the shared key.

## 3.3. SNAD Data Structures

All of the SNAD security schemes use four basic structures: *data objects*, *file objects*, *key objects*, and *certificate objects*. While these objects are all shown as contiguous blocks of data, there is no requirement that they be stored contiguously on disk. In particular, data objects may be broken apart, storing the data itself in a "normal" file system and the remainder of the data object in a special structure (analogous to i-nodes and index blocks in Unix) if desired.

**3.3.1. Secure Data Objects.** A *secure data object* (SDO) is the minimum unit of data that can be read or written in the secure file system, and corresponds to a file block in a standard file system. Files are composed of one or more secure data objects; a sample secure data object is shown in Figure 1.

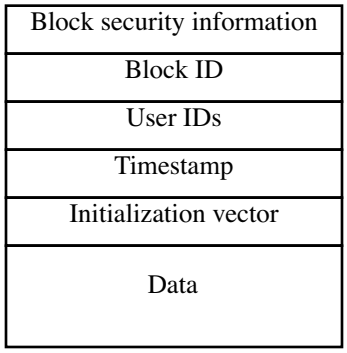


**Figure 1. Secure data object.**

The block security information is different for each of the three schemes discussed in Section 3.4, but is on the order of 32 bytes long. The block ID uniquely identifies each block in the file system, and consists of a file ID and block offset within the file. The first user ID in the list is the creator of the SDO and is used by the SNAD server to determine which key to use to check the security of the block.

The initialization vector (IV) is used to prevent identical data blocks encrypted with the same key from encrypting to the same ciphertext. Using a unique value such as the block ID concatenated with the file ID will guarantee that blocks with identical content encrypt to different ciphertexts. The timestamp is used simply to prevent replay attacks; it need not be an actual timer, but instead could simply be a counter incremented at each client.

The data stored in the data object is encrypted using a symmetric encryption algorithm such as RC5. The key used to encrypt the data is obtained from the key object associated with the file, If each data object is large, files will waste relatively large amounts of space because of internal fragmentation. However, minimizing cryptographic overhead, both storage and operational, requires that data objects not be too small. Like file blocks, secure data objects could be variably sized within a single file system.

**3.3.2. File Objects.** *File objects* are composed of one or more data objects along with per-file metadata. In addition to the usual file metadata such as block pointers, file size, and timestamps, a file object contains a pointer to a key object. This pointer is used to find the

keys that may be used to access the file. Except for the pointer to the key object and perhaps pointers to the extra information for secure data objects, the structures for file objects are identical to those for standard files.

**3.3.3. Key Objects.** Each key object, as shown in Figure 2, contains several types of information. The key file ID is just the unique identifier for the block on the system. The user ID in the header of the key object is that of the last user to modify the key object. When a user writes the object, he hashes the entire object and signs the hash with his private key, storing the result in the signature field. Anyone using the key object verifies the integrity of the object by performing the same hash and verifying the provided signature. This mechanism prevents anyone with low-level access to the disk from undetectably modifying a key object — a client using the key object can check to ensure that the signature on a key object belongs to someone authorized to change the key object. Because someone who modifies a key object must sign it, there is a way of tracing illegitimate modifications to a particular user.

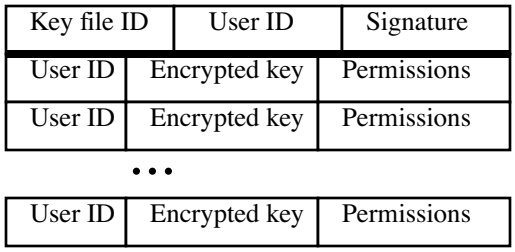


**Figure 2. Key object.**

Each tuple in the body of the key object includes a user ID, encrypted key, and permissions for that user. The user ID need not correspond to a single user; it could, instead, be an equivalent to a Unix group and correspond to several users with shared access to a single private key. The second field in the tuple contains the key for the symmetric RC5 algorithm. Rather than storing this key in the clear, the key object stores the key encrypted with the user's public key. The disk cannot decrypt any key unless it obtains a user's private key, but the only way to get a user's private key is to steal it from a client or the user himself because keys are kept on the client and never sent to the disk. The permissions field is used by the disk to determine whether the user is allowed to write the key object.

A key object may be used for more than one file. If this is done, all files that use the key object are encrypted with the same symmetric encryption key and are readable by the same set of users. In this way, a key object corresponds to a Unix group.

**3.3.4. Certificate Objects.** Each server contains a single certificate object, shown in Figure 3, which contains administrative and cryptographic information about each SNAD user. The disk uses the information in the certificate object to authenticate users and do basic storage management.

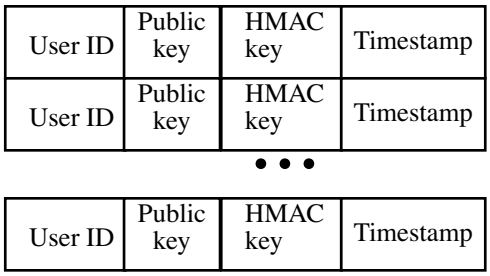


**Figure 3. Certificate object.**

The certificate object contains a list of tuples, each of which includes a user ID, public key, HMAC key (for Schemes 2 and 3), and timestamp. The user ID identifies the user or group to which the remainder of the tuple pertains. The public key is stored on the disk for two reasons: as a convenience so that the disk and those using it need not consult a centralized key server, and for writer authentication in Scheme 1 as described in Section 3.4.1.

The HMAC key is used in the second and third schemes to verify the identity of the user writing data, and is stored encrypted, with the decryption key for the HMAC keys held in non-volatile memory on the disk. Storing the HMAC keys encrypted allows them to be backed up without compromising them. When the certificate object is loaded into memory on disk startup, the HMAC keys are decrypted and cached in volatile memory.

The timestamp field is updated each time a user writes a file object, and is used to prevent replay attacks. A centralized clock is not necessary unless requests for a particular user ID may come from several clients at about the same time. The sole purpose of the timestamp is to prevent replay attacks; clocks may be synchronized using any number of common approaches, or replay attacks may be thwarted as described in Schneier [14].

**3.3.5. Overall Data Structure Organization.** The relationship between the objects described above is shown in Figure 4. The diagram shows multiple file objects using a single key object; this corresponds to a situation where two files have the same access controls. It is likely that there will be relatively few key objects on a disk, just as there are relatively few unique groups in a standard Unix file system.

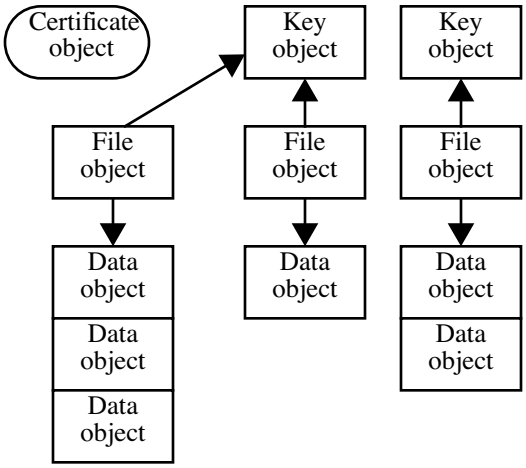


**Figure 4. Relationships between objects in a Secure Network-Attached Disk.**

All of the objects shown in Figure 4 require relatively little overhead. Each data object requires 36–100 bytes of overhead, depending on which security scheme is being used. Even for 100 bytes of overhead, using 8 KB blocks requires just 1.2% overhead for cryptographic metadata. File objects require little overhead — just a pointer to a key object. Key objects are also small: a key object requires 72 bytes for the header and 72 bytes for each user. If each of 10,000 users is part of 200 different groups, there will need to be 144 MB of key objects, or 0.7% of a 20 GB disk. The certificate object requires less than 100 bytes per user, adding just 1 MB to the total. Thus, all of the security information for SNAD requires less than 2% overhead for a 20 GB disk.

## 3.4. SNAD Security Schemes

Our goal for SNAD was to provide authenticated, encrypted storage. Encryption and decryption in SNAD is done at the client. Though the time to do this cannot easily be reduced, symmetric algorithms are relatively fast. There are several different methods for authentication, however, varying in security and speed.

The most secure authentication mechanism is for users to sign the checksum of every block they write using public-key encryption, and for the disk to authenticate every block before writing it. If the checksum is cryptographically strong (i.e., finding two blocks with the same hash requires random guessing), this mechanism is very secure, and allows the system to track the last writer for each block. Unfortunately, signature generation and checking are slow operations, so this method, which we call Scheme 1, is relatively slow and requires fast CPUs on the disk as well as the client.

Scheme 2 reduces the load on the disk's CPU by replacing the signature check at the server with a message authentication code (MAC) check. The client still generates a signature and checks it upon reading a block, but the disk need not perform such an expensive check, improving overall performance.

Scheme 3 further improves performance by dispensing with signatures altogether. Instead, it uses combinations of cryptographic hashes to ensure data integrity throughout the system. This scheme is considerably faster because it requires no signature generation or checking; however, it is not possible to verify who last wrote a file.

Reading and writing data in each of the three schemes follow similar paths. First, the user must give his private key to the client, which is assumed to be trusted by the user. This can be done via password, authentication server, or smartcard. For each file, the user opens the file and reads the key object for the file; file system caching may be transparently used for this operation as for any other. The appropriate field of the key object is then decrypted the to obtain the symmetric encryption key for the file. This key is then used to encrypt the data before sending it to the server and after receiving it from the server.

**3.4.1. SNAD Scheme 1.** The first SNAD scheme provides security on each block of data similar to that provided by some cryptographic electronic mail security schemes. Writes in this scheme encrypt each data block, compute a hash over the entire data object (including the metadata), and sign the hash using the user's private key. This hash can then be verified by anyone with the user's public key. In particular, the disk can recompute the hash and compare it against the hash signed by the user who sent the block. If they match, the disk successfully verifies the provided signature, and the user has the permission to write the file, the SNAD server writes the block to disk.

Reads in this scheme require no operations by the SNAD server CPU, but do require that the client CPU check the hash and signature just as the SNAD server did on a write.

Table 1 summarizes the operations that must be done for each read and write request. This scheme requires relatively expensive signature and verification operations for each disk request; in particular, the CPU on the server must perform a slow signature verification for each block write. Because this CPU is likely to be slow, the verification will reduce write performance.

**Table 1. Operations necessary for Scheme 1.**

| Operation | Read | | Write | |
|---|---|---|---|---|
| | Host | NAS | Host | NAS |
| En/Decrypt | √ | | √ | |
| Hash | √ | | √ | √ |
| Signature | | | √ | |
| Verification | √ | | | √ |

**3.4.2. SNAD Scheme 2.** Scheme 2 replaces the SNAD server's verification with an HMAC. In this scheme, the client performs a cryptographic hash on the block and signs it. However, this signature is only verified by the client when it reads a block. The client also calculates an HMAC on the secure data object using the shared secret HMAC key and sends it to the SNAD server. The SNAD server computes an HMAC using the shared secret key from the certificate object and checks it against the HMAC received from the client. Recalculating the entire hash including the HMAC key would be time-consuming; instead, the client simply performs an HMAC over the hash.

The replacement of a signature verification by an HMAC reduces the load on the SNAD disk CPU, but does not reduce the load on the client CPU, which still must perform signatures on writes and verifications on reads. Table 2 shows the operations that the client and server perform for SDO reads and writes.

**Table 2. Operations necessary for Scheme 2.**

| Operation | Read | | Write | |
|---|---|---|---|---|
| | Client | NAS | Client | NAS |
| En/Decrypt | √ | | √ | |
| Hash | √ | | √ | √ |
| Signature | | | √ | |
| Verification | √ | | | |

**3.4.3. SNAD Scheme 3.** The third scheme eliminates the signed hash of Scheme 2, and uses only the keyed-hash (HMAC) to authenticate a writer of a data block and verify the block's integrity. HMACs differ from signed hashes in that a user able to verify a keyed-hash is also able to create it. Scheme 3 still uses public-key authentication for key objects because writing key objects, while slower with public-key controls, is very infrequent.

Write operations in this scheme require the client to encrypt the SDO and calculate an HMAC over the ciphertext. This information is then sent to the disk, which authenticates the sender by recomputing the HMAC using the shared secret key from the certificate object. If the write is authentic and the user has the permissions to modify or create the SDO, the SNAD disk commits the write to disk, updating structures as necessary. Note that the disk does not store the HMAC because it must recalculate a new HMAC if the reader is a different user from the user who wrote the SDO.

Unlike the previous two schemes, this scheme requires the SNAD disk to perform a cryptographic operation on a read: the disk must calculate a new HMAC using the key from the user requesting the data. The data object, along with the new HMAC, is then sent to the client requesting the data. If the disk were forced to write blocks without the proper encryption key, a client could detect this during a read by checking the non-linear checksum against the decrypted data.

The operations performed by the client and SNAD disk are summarized in Table 3. Note that this scheme requires no signature generation or verification operations; however, the SNAD disk must now compute an HMAC on both reads and writes.

**Table 3. Operations necessary for Scheme 3.**

| Operation | Read | | Write | |
|---|---|---|---|---|
| | Client | NAS | Client | NAS |
| En/Decrypt | √ | | √ | |
| Hash | √ | √ | √ | √ |
| Signature | | | | |
| Verification | | | | |

## 4. Preliminary Results

We implemented all three security schemes in a block-oriented client and server, and tested their speed against that of a similar system without encryption. The results are shown in Figure 5. As expected, Scheme 3 performs the best on both reads and writes, 15%–20% slower than unprotected operations. Schemes 1 and 2 are comparable in performance, but are considerably slower than Scheme 3 because of the need to generate and check signatures.

## 5. Future Work

There is still much work to do on cryptographically secure file systems, particularly with actual implementations. We are investigating the actual performance of a file system using the security described in this paper. Issues such as key revocation and general security infrastructure also need to be explored further.

Another area that we are currently investigating is the scalability of the different security schemes. Schemes 1 and 2 are slow because the clients must generate a signature. With one client and one server, this

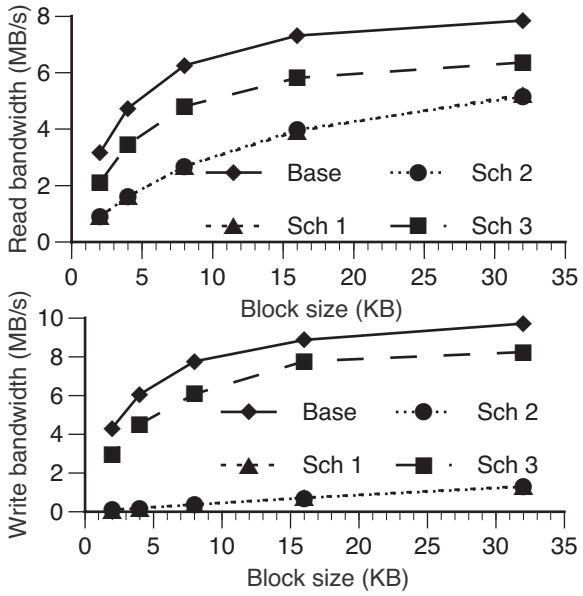


**Figure 5. SNAD performance on sequential accesses.**

reduces performance. However, with many relatively low-bandwidth clients, the overhead of generating signatures is distributed to many machines. In such a system, even a relatively slow CPU on a SNAD server can handle several clients simultaneously.

## 6. Conclusions

We presented the details of the Secure Network Attached Disk system, showing that cryptographic security is possible for distributed file systems and network-attached storage. This type of system is feasible with today's computing power, and will become even more attractive as processors become faster.

This security mechanism for distributed file systems solves many of the performance and security problems in existing systems today. This system provides user data confidentiality and integrity from the moment it leaves the client computer. The distributed disks should perform substantially better than centralized file servers, and provide better reliability. Having the security functionality decentralized will improve performance and scalability. Distributed security also removes the single point of failure that plagues many proposed centralized security schemes to date.

Integrating SNAD and schemes like it into modern distributed file systems is essential. Such integration can cost relatively little in performance but provides tremendous advantages in security. Given the hostile environment on the Internet, distributed file systems can no longer afford to be without strong security.

## References


[1] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang, "Serverless Network File Systems," *ACM Transactions on Computer Systems*, Feb. 1996, pages 41-79.

[2] M. Blaze, "A Cryptographic File System for Unix," *Proceedings of the First ACM Conference on Computer and Communication Security*, Nov. 1993, pages 9-15.

[3] M. Blaze, "Key Management in an Encrypting File System," *Proceedings of the Summer 1994 USENIX Conference*, 1994.

[4] W. Freeman, *Decentralized Security for Network Attached Storage*, Ph.D. thesis, University of Maryland Baltimore County, April 2000.

[5] W. Freeman and E. Miller, "An Experimental Analysis of Cryptographic Overhead in Performance-Critical Systems," *Proceedings of the 7th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 99)*, College Park, MD, October 1999, pages 348-357.

[6] K. Fu, M. F. Kaashoek, and D. Mazieres, "Fast and secure distributed read-only file system," *4th Symposium on Operating Systems Design and Implementation* (San Diego, CA), October 2000, pages 181-196.

[7] G. Gibson, et al., "A cost-effective, high-bandwidth storage architecture," *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, CA), October 1998, pages 92-103.

[8] H. Gobioff, "Security for a High Performance Commodity Storage Subsystem," Ph.D. thesis, Computer Science Department, Carnegie Mellon University, July 1999. Available as Technical Report CMU-CS-99-160.

[9] J. Howard, et al., "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems* **6**(1), February 1988, pages 51-81.

[10] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," *IETF Network Working Group RFC2104*, February 1997.

[11] B. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications Magazine* **32**(9), September 1994, pages 33-38.

[12] B. Reed, E. Chron, R. Burns, and D. Long, "Authenticating Network Attached Storage," *IEEE Micro*, **20**(1)January 2000, pages 49-57.

[13] J. Reid, "Plugging the Holes on Host-Based Authentication," *Computers and Security*, 1996, pages 661-671.

[14] B. Schneier, *Applied Cryptography*, Wiley (New York), 1994.

[15] M. J. Wiener, "Performance Comparison of Public-Key Cryptosystems," *RSA CryptoBytes*, **4**(1), Summer 1998.