Strong Security for Network-Attached Storage

Article · February 2002

Source: CiteSeer

CITATIONS
119

READS
1,438

4 authors, including:

Ethan L. Miller
University of California, Santa Cruz
241 PUBLICATIONS **7,428** CITATIONS

SEE PROFILE

Darrell D. E. Long
University of California, Santa Cruz
307 PUBLICATIONS **8,756** CITATIONS

SEE PROFILE

Strong Security for Network-Attached Storage

Ethan Miller elm@cse.ucsc.edu *University of California, Santa Cruz*
Darrell Long darrell@cse.ucsc.edu *University of California, Santa Cruz*
William Freeman william.freeman@trw.com *TRW*
Ben Reed breed@almaden.ibm.com *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 super-user access to unencrypted data. While denial-of-service attacks cannot be prevented (attackers with sledgehammers can deny service to *any* system), our system detects forged data. The system was developed using a raw disk, and can be integrated into common file systems.

All of this security can be achieved with little penalty to performance. Our experiments show that, using a relatively inexpensive commodity CPU attached to a disk, our system can store and retrieve data with only a 15-20% performance loss over raw transfer rates for sequential disk requests, and virtually no penalty for random disk requests. With such a minor performance penalty, 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 hackers 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 typically stored on a centralized file server that is managed by a system administrator with super-user privileges, leaving the data vulnerable to anyone who can prove (legitimately or otherwise) that he is the system administrator.

Recently, however, network-attached disks have begun to replace traditional centralized storage systems [1,9]. 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 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 does not impose much overhead on the file system. Our experiments using raw disks (the worst case — any real file system will impose additional overhead further hiding any cryptographic overhead) show that the encryption and verification provided by our system imposes less than a 20% penalty for large sequential transfers and almost no penalty for small random accesses to blocks on disk.

We begin by describing 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 present three alternate security schemes, each appropriate for different levels of client and server CPU performance. Next, we describe the experiments we ran to test our system's performance and show that security for network-attached disks is possible without much performance penalty. We conclude with a description of our plans for integrating strong security into modern file systems.

This paper has been submitted to the 2001 USENIX Technical Conference (June 2000 in Boston, MA).

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 processors that can be associated with each disk in a distributed file system [9].

2.1 Controlling Access to File Systems

Most file systems include some measure of security. However, systems such as NFS [19], xFS [1], and Petal [13] pass nearly all of their data in the clear, relying on relatively insecure networks and trusted hosts for data protection. Such a tactic works well if a network is totally disconnected from the rest of the world, but is a poor solution for modern systems that are exposed to the Internet. Some protection can be provided via firewalls or by secure network protocols [2,11,16], but these mechanisms do not protect data stored on disk.

Other systems, such as AFS [10,21] and NASD (Network Attached Secure Disk) [9] use Kerberos [17] to provide security. These systems provide stronger security by requiring users to obtain "tickets" from a third party. The tickets are then presented to the file server (AFS) or NASD disk as proof of identity and access rights. These systems are considerably stronger than those that rely upon simple authentication, but they still suffer from several problems. First, files are left in the clear on the disks themselves, and are normally transferred in the clear. Second, Kerberos-based systems rely upon a centralized security authority that is separate from the disks themselves. This is advantageous for sharing within a well-connected organization, but can become more difficult for widely distributed systems.

SCARED [18] is another file system that uses encryption to authenticate remote network storage. The SCARED design supports the use of end-to-end encryption of data, and, similar to SNAD, uses timestamps and counters to protect against replay attacks. However, SCARED does not implement end-to-end data encryption, leaving that for the underlying file system. SCARED, like the authentication we propose in Scheme 3, uses secure hashes for authentication.

The Secure File System (SFS) [8,15] provides strong authentication and a secure channel for communications. It includes an extensive authentication mechanism for individual users, and provides strong security for data in transit between clients and servers. It also allows servers to authenticate their users and clients to authenticate servers. However, it still relies upon trusted file servers that do not alter data stored on them. If a "trusted" server is physically compromised, the data on it may be readable to the intruder. In an environment where data storage is outsourced to companies, this security risk is unacceptable.

2.2 Protecting Data on Disk

While most file system security has focused on access control and protecting data in transit, there have been a few file systems that have protected data on disk as well. There has been some work on protecting data on disk by making it impossible to delete [22]; however, our focus is on protecting data on disk from discovery by an intruder.

Many users have implemented their own "secure file system" by simply encrypting their files using standard encryption software. This provides security and, if the user also signs the file, a mechanism for ensuring that the server did not corrupt the data. However, this is an *ad hoc* mechanism, and does not deal with many issues such as sharing files between users.

The Cryptographic File System (CFS) developed at AT&T Bell Laboratories [3,4] encrypted all data and potentially sensitive metadata stored on disk. When a user desired access to an encrypted directory, he issued a command to attach the encrypted directory to a subdirectory of /crypt. If the correct password was entered, the data was subsequently available in decrypted form. Because the structures to support this were stored in a "normal" directory structure, they could be used with NFS and other file systems. 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 mecha-

nisms for distributing keys among users for sharing files.

The design of a trusted database system such as Trusted DataBase (TDB) [14] could be adapted to file systems; however, TDB is not easily scalable, making it less useful for large-scale file systems.

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; the first two are considerably more CPU-intensive, but are also more secure. The third alternative is feasible given current low-cost CPUs, and provides nearly as much security as the first two alternatives.

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, including storage on disk. This is necessary to restrict access to data to only authorized users, specifically excluding system administrators and backup systems. An adversary with full access to all of the bits on the disk or the network should be unable to decipher any user files — the disk must not contain sufficient information to decrypt the data stored on it. Rather, data should only exist in unencrypted form on the client.

A second desirable feature is data integrity. A user reading data from the server must be sure that the files received are those he originally stored. It is no longer a good idea to trust that a disk is secure against intruders; data modified at the disk or introduced into the system by an malicious intruder must be detectable. Storing a non-linear checksum of the unencrypted data in a block along with the encrypted block, as described in Section 3.4.3, allows any authorized user to detect a change made to the encrypted block by an intruder who did not have the symmetric key to encrypt the file.

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 [10].

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. By minimizing the effort required by the network-attached disk's CPU, however, it is possible to build a distributed file system that can be used by hundreds of clients, each of which can decrypt the data intended for itself.

3.2 Basic Mechanisms

The basic mechanism behind our security system 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 the RC5 algorithm [20] to encrypt the data before it leaves the client, though other algorithms such as Blowfish [20] would also be acceptable. This ensures 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. This process is described in Section 3.4. The security provided by SNAD is very strong; the symmetric algorithms use 128 bit keys — the key length Schneier recommends for highly secure information with a lifetime longer than 40 years [20]. If 128 bit keys are too short, longer keys may be used at the expense of extra processing at the client.

SNAD also makes extensive use of cryptographic hashes and keyed hashes. Cryptographic hashes such as MD4, MD5, and SHA-1 [20] 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. While it is possible to find two sets of input data that will result in the same MD4 hash (weak collision) [5], there is still no known way to produce a second input that hashes to the same value as a given first input. MD5 and SHA are vari-

ations on MD4 for which it is currently believed NP-hard to find two input texts that result in the same hash value.

Keyed hashes such as HMAC [12] 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 inodes 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.

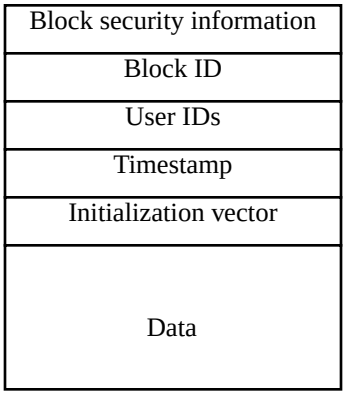

Figure 1. Secure data object.

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 is a unique identifier for the block in the file system, and could be a combination of a a unique file identifier and block number in the file. The first user ID in the list is the creator of the secure data object and is used by the SNAD server to determine which public key or writer authentication 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, as described in Section 3.3.3.

If the data contained in each object is too large, each file will waste relatively large amounts of space. However, minimizing cryptographic overhead, both storage and operational, requires that objects not be too small. Like file blocks, secure data objects could be variably sized within a single file system; however, we assumed fixed sized secure data objects. We explore the performance tradeoffs with respect to object size in Section 4.

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, 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 the disk (or anyone with access to it, such as a system administrator) 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.

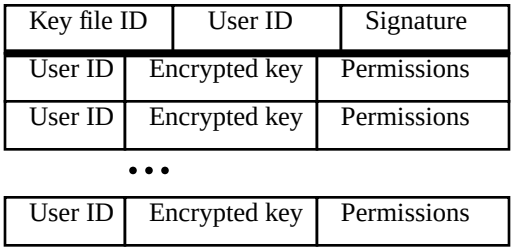

Figure 2. Key object.

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 network-attached disk 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.

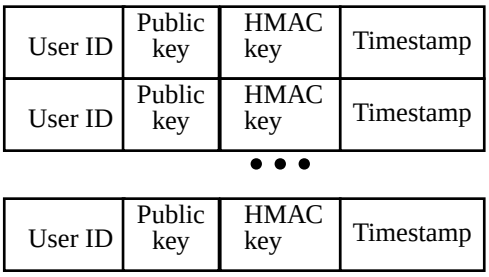

Figure 3. Certificate object.

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. This can occur if a user ID actually corresponds to a group, or if a user is logged on to several systems at once. 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 [20].

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.

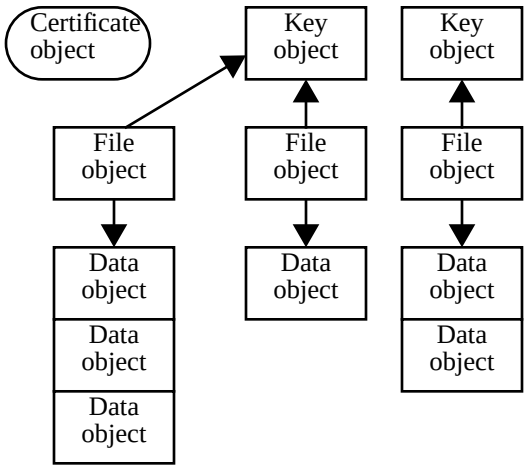

Figure 4. Relationships between objects in a Secure Network-Attached Disk.

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 occupies less than 2% overhead for a 20 GB disk.

3.4 SNAD Security Schemes

The three schemes we considered differ only in the way in which authentication is done, resulting in different performance levels because of the number, type, and location of the cryptographic operations. We focus on the operations performed in each of the schemes; details on the security of the schemes can be found in [6].

All of the SNAD protection schemes provide strong security by encrypting each block of data using RC5 at the client. Because the RC5 keys are stored on the drive encrypted with the public key of any user permitted to access the file, even gaining access to both the encrypted data and the encrypted keys would be of no use without the necessary private key. As a result, the disks provide an encrypted block of data and encrypted keys to anyone who requests them. Assuming that the encryption is sufficiently strong, the encrypted information will not benefit an attacker, so there is little use in having the disk attempt to verify the identity of a requester. If the user can decrypt the symmetric key, he can read the data in the block.

Writing blocks in all three schemes is controlled in much the same way as a standard file system: only users with permission to write a block are allowed by the disk to do so. Traditional file systems, however, are vulnerable to attackers placing bogus data on the disk by gaining access to low-level write routines. SNAD guards against this with encryption and checksumming; secure data objects written without knowledge of the symmetric key for the object will give a checksum error when decrypted by a client. The only way for an unauthorized write to occur is for an authorized reader to gain physical access to the disk, use his symmetric key to write a secure data object, and (for Schemes 1 and 2) sign the cryptographic hash. This weakness is present in any security scheme that uses symmetric key encryption to protect files: anyone that can decrypt the file can encrypt it as well.

Reading and writing data in each of the three schemes have much in common. 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 (e.g., as used in Kerberos [17]), or smartcard. For each file, the user opens the file and reads the key object for the file;

for this operation as any others, file system caching may be transparently used. 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. Note that this scheme requires relatively expensive signature and verification operations for each disk request; in particular, the CPU on the network-attached disk must perform an expensive signature verification for each block write. Because this CPU is likely to be slow, the verification will reduce write performance.

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

Table 1. Cryptographic operations necessary for Scheme 1.

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.

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

Table 2. Cryptographic operations necessary for Scheme 2.

3.4.3 SNAD Scheme 3

The previous two schemes use a public-key signature to identify the originator of a data block and ensure that the block hash has not been modified. The third scheme uses a keyed-hash (HMAC) approach 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.

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

Table 3. Cryptographic operations necessary for Scheme 3.

4 Performance

All of the security schemes we presented would go a long way towards securing data in distributed file systems. However, few would use such strong security if doing so meant crippling the file system's performance. Our measurements, however, show that strong security can be achieved without sacrificing too much performance. Using slightly longer keys has relatively little effect on encryption speed, but doubles the time required for brute-force cryptanalysis for each bit added to the key length.

4.1 Cryptographic Overhead

We first tested the raw speed of the cryptographic algorithms used by SNAD; this provided insight into how fast each of our schemes was likely to be. We previously found that using encryption in time-critical systems is feasible [7]; performance tests on additional (newer) hardware are summarized in Figure 5.

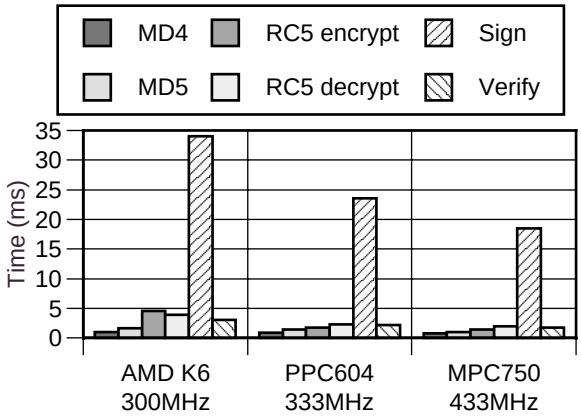

Figure 5. Performance of cryptographic algorithms on modern CPUs. Block size is 32 KB except for sign & verify, which are done on 128 bit inputs.

As Figure 5 shows, the most expensive operation by far is signature generation. We used a modulus of 512 bits in the RSA algorithm. with 32,767 as the public exponent, which allowed verification times to be much faster than signature generation times. Similar tests on a 200 MHz Pentium Pro with 1024 bit keys [23] required 43 ms for a public key signature; the faster processors available today should be able to complete this operation in times similar to those we measured for 512 bit keys.

The length of time required to compute a signature suggests that Schemes 1 and 2 are likely to be considerably slower than Scheme 3 on a workload that includes many writes. On a read-mostly file system, however, the long time required to calculate a signature is less important and the benefits of the stronger protection available from Schemes 1 and 2 may be more important.

While this data was measured on relatively modern CPUs, progress marches on. As a result, a 500 MHz AMD K6 is currently available for $50 retail; a 300 MHz K6 is even less expensive, and both are inexpensive enough to serve as an embedded processor.

By combining Tables 1, 2, and 3 and Figure 5, we can derive the theoretical overhead for each security scheme. Figure 6 shows the overhead for each

scheme if the MPC750 is used in both client and server; different processors will have different overheads, but the ratios between the schemes will be similar.

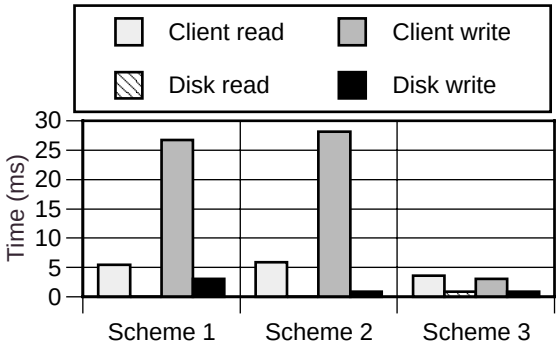

Figure 6. Cryptographic overhead for SNAD using a 360 MHz MPC750 for both client and disk, assuming 32 KB data blocks.

From Figure 6, we can derive the theoretical "speed limit" for performance using a MPC750 for both client and disk. Schemes 1 and 2 are limited to nearly 6.4 MB/s for reads, but only 1.4 MB/s for writes. Scheme 3, on the other hand, can read at up to 10 MB/s and write even faster — 12.7 MB/s. These rates are based on cryptographic overhead only; they do not include network and disk delays. However, they are useful in showing how fast a cryptographic file system *could* go given sufficiently fast disks and networks. Note, too, that Schemes 1 and 2 are limited primarily by the amount of time needed by the client to compute the signature; thus, they may work well in environments with many clients and relatively few disks.

4.2 SNAD Performance Measurements

Though measuring the performance of cryptographic operations is useful, it does not show the full impact of end-to-end security on a distributed file system. We constructed prototype SNAD disks and clients, and ran experiments to see how much performance degradation was incurred when cryptographic overhead was added to a block-level SNAD server. The observations in this section present the worst-case scenario for cryptographic overheads because real file systems will likely have other overheads not present in a raw block server.

Our experimental setup consisted of multiple VME boards running a real-time kernel (Wind River's VxWorks). Each board was based on the MPC750 running at either 333 or 360 MHz. The VME chassis was used only for power; the boards were connected to each other by 100 Mbit/s Ethernet switched through a Cisco 2900 XL switch. In addition, each server was connected to a Seagate Cheetah 10K RPM Ultra SCSI disk drive. We used 360 MHz boards for both client and server for the one-to-one tests; our multiple client and server tests used different configurations that are detailed later.

4.2.1 Baseline: No Security

Our first set of tests stressed the system without any cryptography, showing how fast the system could read and write data unencrypted and unencumbered by any security mechanisms. Figure 7 shows the performance of a one client, one disk SNAD system without any cryptographic overhead. There is a knee in the performance curve around 8 KB, and a block size of 32 KB delivers nearly the maximum performance permitted by a 100 Mbit/s Ethernet for sequential access. As expected, random accesses are slower than sequential accesses, though the large write buffer on the disk allows write performance for random writes to approach that of sequential writes for large blocks.

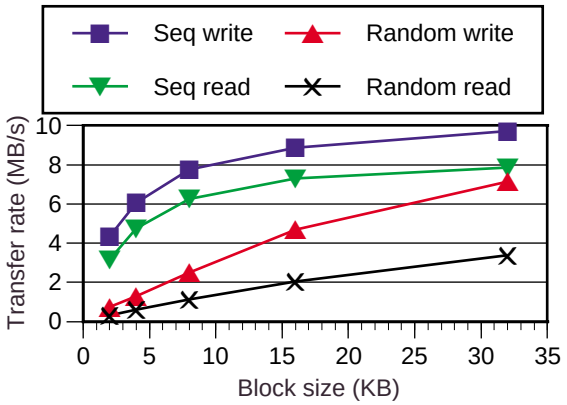

Figure 7. SNAD performance without cryptographic controls.

We used the performance measurements shown in Figure 7 as a baseline for our other performance measurements, showing the effect of strong crypto-

graphic security on file system performance for each security scheme in Section 3.4.

4.2.2 Performance of Scheme 1

As described in Section 3.4.1, Scheme 1 provides the best security, albeit at the cost of lower performance. Our experiments showed that, as expected, Scheme 1 suffers greatly on both sequential and random writes. However, Scheme 1 can keep up with random reads of blocks up to 32 KB, though it cannot keep up with sequential reads. These results are shown in Figure 8.

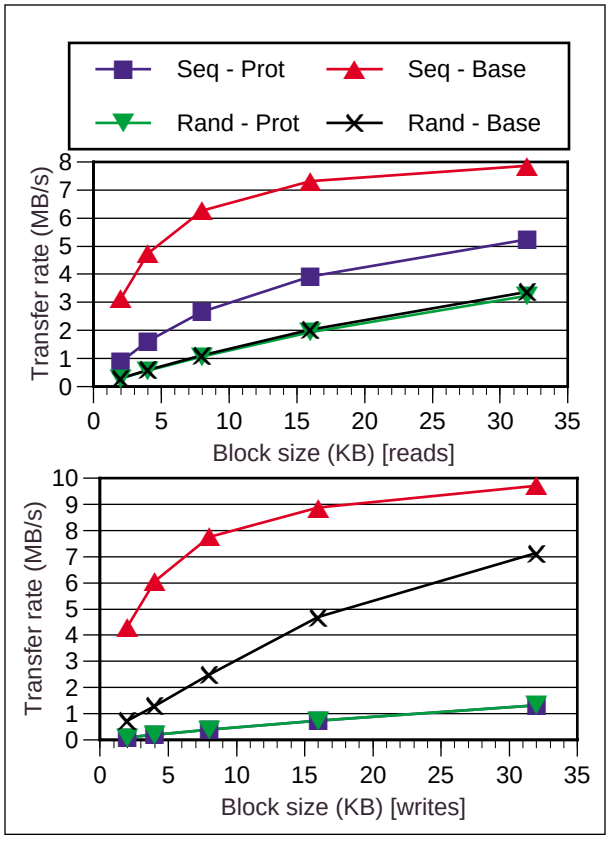

Figure 8. Performance of Scheme 1.

The performance shown in Figure 8 indicates that, with current processors, Scheme 1 is unsuitable for distributed file systems that require good performance with one exception: file systems that are dominated by small random reads. This is exactly the access pattern exhibited by clients accessing key objects, so Scheme 1 would be a good choice for protecting key objects. For most access patterns, though, we must use other security schemes until processor speeds increase sufficiently to permit use of Scheme 1.

4.2.3 Performance of Scheme 2

Scheme 2 improves upon the first scheme by changing the write operation to be less CPU-intensive at the SNAD server with little loss in security. The read operations in both Schemes 1 and 2 are identical, and the graph in Figure 9 indeed shows that the two schemes perform identically, with sequential reads suffering a significant performance loss and random reads running at the same speed encrypted and in the clear. However, the hoped-for performance gains on writes did not materialize with a single client. Instead, the write performance of Scheme 2 is similar to that of Scheme 1; neither is currently suitable for systems with large sequential writes.

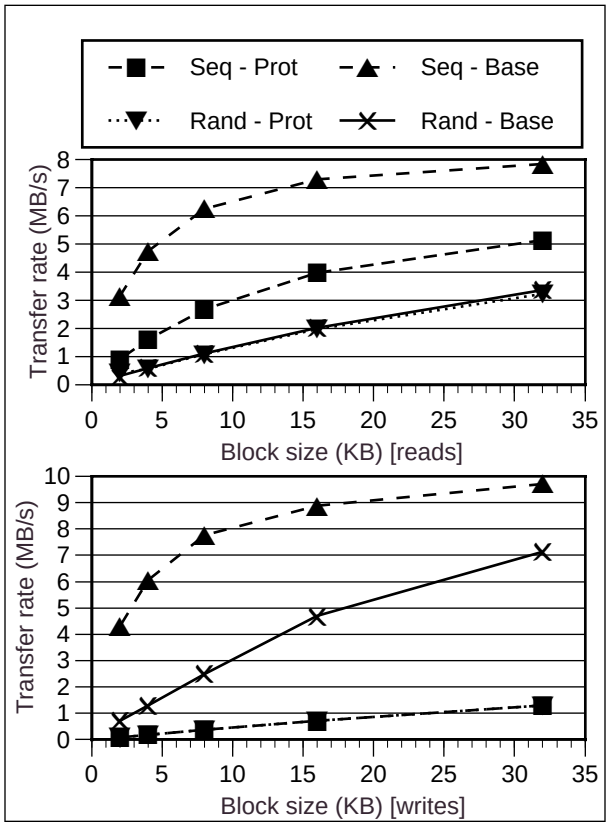

Figure 9. Performance of Scheme 2.

4.2.4 Performance of Scheme 3

Scheme 3 replaces the signed hash for block integrity and writer authentication with a keyed hash (HMAC). While this results in slightly less security, performance for this scheme is greatly improved over the first two schemes, as shown in Figure 10.

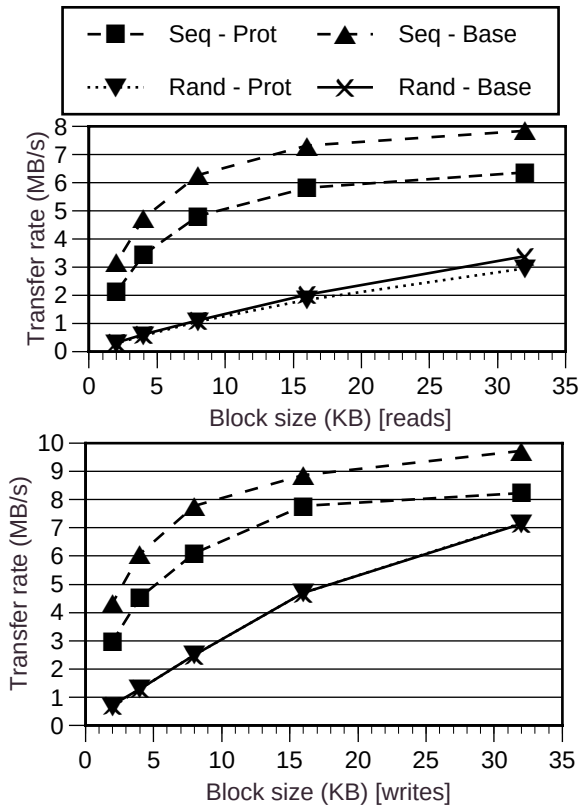

Figure 10. Performance of Scheme 3.

Figure 10 shows that, for Scheme 3, random I/O operations (read and write) suffer little or no performance penalty for cryptographic controls with block sizes between 2 KB and 32 KB. Long sequential transfers, on the other hand, do suffer a performance penalty. However, large sequential writes with encryption run at 88% of the bandwidth of unencrypted writes, and large sequential reads run at 81% of the bandwidth of unprotected reads. We believe that this relatively small performance penalty is an acceptable price to pay for a large increase in file system security.

5 SNAD Design Issues

There are many design issues that must be considered when building a secure file system, particularly in the area of key management. Mazieres, et al. [15] discuss many of these issues in more detail; however, we feel that there a few problems of particular importance that should be mentioned here. These issues include creating key objects, adding and removing users from a key object, and providing an key escrow system.

5.1 Creating a Key Object

The creation of file objects and data objects is relatively straightforward, assuming that an appropriate key object and certificate object already exist. However, there must be a way to create new key objects.

The primary requirement for a new key object is a new RC5 key that will be used to encrypt files that use the key object. The key object creator must ensure that the RC5 key is truly random (not merely pseudo-random), and then encrypt it with his own public key as well as that of anyone else he wishes to have access to the file. Once this is done, the key object may be stored on a SNAD disk, and is ready for use. This procedure is relatively simple, and only relies on the ability to generate truly random numbers for the RC5 key.

5.2 Modifying Access Permissions

One of the largest difficulties with many systems for maintaining security is dealing with the modification of access groups. Adding users to an access group is relatively straightforward — a user with the rights to add a new user can simply use his private key to obtain the RC5 key, and encrypt that key with the new user's public key. The new user can now access the files associated with this key object.

Revoking permissions is a more difficult problem for which there are several possible solutions. The first solution is to simply delete the user's line from the key object; if this is done, the user will be unable to obtain a new copy of the RC5 key, though he may still have the RC5 key cached somewhere. A second solution is to immediately reencrypt the file using a different key object containing only those users who should still have access to the file. This solution is slower, but will ensure that the revoked user cannot access the file. A third solution is to apply the second solution lazily. This allows the revoked user to continue to access old data, but denies him access to any new data, which is encrypted with a different key.

The choice of revocation method is still an open issue with no well-accepted solution. We are currently investigating tradeoffs between these three mechanisms for changing access permissions.

5.3 Key Escrow

One potential problem with an encrypted file system is that a user may abscond with his key (or simply lose it), making it impossible to access files that only he was allowed to see. In many organizations, this is an important argument against encryption.

However, this problem can be solved with key escrow: including an escrow "user" in every key object. This private key for this escrow "user" may be kept in a safe (or even spread across multiple safes); the system only requires that the corresponding public key be available for the creation of entries in new key objects. This solution in no way weakens the strong security present in the file system; an intruder would still need the private key (which is not kept online) to break into any file.

Note that escrow is *not* required in SNAD, though it may be included if desired.

6 Integrating SNAD into a File System

The security schemes we presented are not themselves a file system; rather, they are mechanisms to be used by a distributed file system or network-attached storage system to provide strong system security. Thus, it is crucial that existing file systems be able to use these techniques to greatly reduce the risk of compromised data.

All of the security schemes described in Section 3.4 use the same basic structures from Section 3.3 and similar algorithms and security procedures. Thus, all should be equally easy (or difficult) to integrate into an existing file system.

We integrated the cryptographic controls into a Linux ext2fs system using FiST [24] and hand-crafted code. While this implementation does not transfer data over a network, it does show the magnitude of the cryptographic overhead for our design when implemented in a real file system. For this implementation, we stored key objects as regular files and interposed routines to do encryption and decryption on file reads and writes. We did not store verification information for each file because it is unlikely that a user will not trust the local file system. Storing HMACs or signatures for each block is required for security over a network, however, and will be implemented in the future.

Our implementation cached shared keys for decrypting files; this is a reasonable situation because most users will only belong to a few groups and thus need only a few key objects at any time. We used 128 bit Blowfish encryption and 1024 bit public keys running on a 266 MHz Pentium; a 1 GHz processor should be able to run 3-4 times faster. We ran experiments with both standard ext2fs and ext2fs protected with encryption. Each experiment created 1000 files of a particular size, with the size varying by experiment, as shown in Figure 11. Our preliminary results show that our security mechanism results in an overhead of about 15% over standard ext2fs for most file sizes.

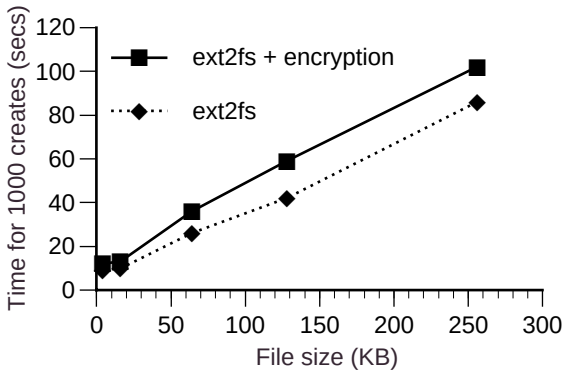

Figure 11. Time required for 1000 creates of varying sized files with and without encryption.

This experiment, combined with our more extensive experiments on block-level operations, demonstrate that strong encryption and authentication can be included in file systems with an acceptably small performance penalty.

7 Future Work

There is still much work to do on cryptographically secure file systems, particularly with real implementations. Issues such as key revocation and security infrastructure in general need to be explored further.

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

The performance of SNAD is quite good; it can provide strong security and authentication for a penalty of between 1% and 20%, depending on workload. This overhead can be reduced further by placing special-purpose encryption hardware on CPUs, making it possible to do cryptographic operations considerably faster than the general purpose processors used in this study. If this is done, SNAD with Scheme 1 security would be feasible.

8 Conclusions

We presented the details of the Secure Network Attached Disk system along with performance measurements 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. A cryptographically controlled write operation was able to run at over 88% of the optimal performance, while the read was limited to 81% of optimal performance.

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. As we have shown, such integration costs 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.

Code availability

The code for our encrypted file system running under Linux will be available online well before the 2001 USENIX Technical Conference.

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, "Transparent Mistrust: OS Support for Cryptography-in-the-Large," *Proceedings of the Fourth Workshop on Workstation Operating Systems*, 1993, pages 98-102.

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

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

[5] H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption Workshop, Lecture Notes in Computer Science*, vol. 1039, Springer Verlag, 1996, pages 53-69.

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

[7] 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.

[8] 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.

[9] 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.

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

[11] J. Ioannidis and M. Blaze, "The Architecture and Implementation of Network-Layer Security Under Unix," *Proceedings of the Fourth Usenix Unix Security Symposium*, October 1993, pages 29-39.

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

[13] E. Lee and C. Thekkath, "Petal: Distributed Virtual Disks," *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996, pages 84-92.

[14] U. Maheshwari, R. Vingralek, and W. Shapiro, "How to Build a Trusted Database System on Untrusted Storage," *4th Symposium on Operating Systems Design and Implementation* (San Diego, CA), October 2000, pages 135-150.

[15] D. Mazieres, M. Kaminsky, M. Kaashoek, and E. Witchel, "Separating key management from file system security," *Proceedings of the 17th ACM Symposium on Operating System Principles*, December 1999, pages 124-139.

[16] K. Nakayoshi, N. Yamai, T. Matsuura, K. Abe, and K. Murakami, "A Secure Private Network File System with Minimal System Administration," *IEEE Communications, Computers, and Signal Processing*, 1997, pages 251-255.

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

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

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

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

[21] M. Spasojevic and M. Satyanarayanan, "An Empirical Study of a Wide-Area Distributed File System," *ACM Transactions on Computer Systems* **14**(2), May 1996, pages 171-199.

[22] J. Strunk, G. Goodson, M. Scheinholtz, C. Soules, and G. Ganger, "Self-Securing Storage: Protecting Data in Compromised Systems," *4th Symposium on Operating Systems Design and Implementation* (San Diego, CA), October 2000, pages 165-180.

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

[24] E. Zadok and J. Nieh, "FiST: A Language for Stackable File Systems," *Proceedings of the 2000 USENIX Technical Conference* (San Diego, CA), June 2000, pages 55-70.