Protecting Multimedia Data in Storage: A Survey of Techniques Emphasizing Encryption

Paul Stanton[†‡], William Yurcik[†], Larry Brumbaugh[†]

[†]National Center for Supercomputing Applications (NCSA), Champaign, IL, USA;
[‡]Department of Computer Science, University of Illinois at Urbana-Champaign

ABSTRACT

Protecting multimedia data from malicious computer users continues to grow in importance. Whether preventing unauthorized access to digital photographs, ensuring compliance with copyright regulations, or guaranteeing the integrity of a video teleconference, all multimedia applications require increased security in the presence of talented intruders. Specifically, as more and more files are preserved on disk the requirement to provide secure storage has become more important. This paper presents a survey of techniques for securely storing multimedia data, including theoretical approaches, prototype systems, and existing systems ready for employment. Due to the wide variety of potential solutions available, a prospective customer can easily become overwhelmed while researching an appropriate system for multimedia requirements. Since added security measures inevitably result in slower system performance, certain storage solutions provide a better fit for particular applications along a security/performance continuum. This paper provides an overview of the prominent characteristics of several systems to provide a foundation for selecting the most appropriate solution. Initially, the paper establishes a set of criteria for evaluating a storage solution based on confidentiality, integrity, availability, and performance. Then, using these criteria, the paper explains the relevant characteristics of select storage systems providing a comparison of the major differences. Finally, the paper examines specific applications of storage devices in the multimedia environment.

1. INTRODUCTION

With the proliferation of stored data in all environments, organizations face an increasing requirement to both temporarily and permanently retain information. The storage medium for housing this information becomes a prime target for attack by a malicious intruder. If an outsider can successfully penetrate the data storage, the intruder can potentially gain information that violates privacy, discloses valuable secrets, or prevents the access of legitimate users. The deleterious effects of such an attack are not quantifiable. If the organization takes no storage security measures, the data store becomes a lucrative single point-of-attack for an intruder. The avoidance of this obviously unfavorable condition has generated a detailed field of computer research. The research community has actively pursued options for securing stored information, and has consequently developed many potential schemes for ensuring information confidentiality, integrity, and availability without substantially degrading performance.

A major problem associated with storing large amounts of data is how to properly weigh the costs and benefits associated with security measures. The most secure systems are so because of the increased measures to protect the data, but each additional measure comes with a cost in terms of both convenience and processing time. In order to effectively select the best security scheme, users must have an understanding of the primary security features available in the storage security community and then be able to quantifiably compare the systems. Developing an understanding of these aspects will help to motivate the direction for future research and assist the selection of the appropriate storage solution for a set of specific requirements. A single organization may even benefit by employing different storage techniques for differing forms of multimedia data. Certain types of information imply stricter control of privacy, whereas others are deadline driven and cannot afford the additional performance cost of added security. Streaming video, for instance, cannot suffer a significant performance penalty without severely affecting the viewing condition.

The remainder of the paper is organized as follows. Section 2 provides a standardized set of criteria to evaluate secure storage systems. Section 3 provides a survey of eight storage systems. Section 4 provides a classification and comparison of the surveyed systems. Section 5 contains applications to multimedia and Section 6 concludes.

2. CRITERIA FOR EVALUATION

This section establishes a common set of criteria for evaluating a storage security system. There are many different ways to approach storage systems but for the purposes of establishing a common reference, confidentiality, integrity, availability, and performance have been selected. While this paper does not approach any criteria in exhaustive detail, it is necessary to describe the evaluation criteria prior to assessing the individual systems. Confidentiality, integrity, and availability are commonly referred to in the computer security arena, and performance was added to ensure systems achieve an appropriate balance between security and processing ability. Prior to discussing each aspect in more detail, it is important to understand that none of these attributes is mutually exclusive, and, in fact, to have a secure system all attributes must be satisfied.

2.1 Confidentiality

From a security perspective, ensuring confidentiality implies that no one has access to data unless specifically authorized. Different systems control this authorization process in various ways. The first step in authorizing access to information is to properly identify users via authentication. The storage system must define the means for a user to be properly identified prior to gaining access, and then having appropriately identified a user, the system must allow access to only specified data associated with that user. Proper authorization to access the storage system does not imply access to the entire system; in fact, the contrasting principle of least privilege is generally applied. Data owners must, however, have a method for allowing others to access information when appropriate via a delegation of authorization scheme.

To prevent unauthorized access to information, confidentiality also implies that the system must encrypt data and, therefore, requires either users or servers to apply cryptographic keys. Determining how the keys are managed has had a significant impact on the overall design – whether distributed by a centralized group server or by individual file owners, the effects of key management on performance and user convenience must be analyzed. Cryptographic operations are often the most computationally expensive aspect of accessing securely stored data so deciding where and how the cryptographic keys are applied is necessary.

An additional critical discussion concerning key management involves how keys are revoked. Once the system determines to revoke a particular user's access, the user's keys must no longer work within the system, or at a minimum not allow access to future versions of the files. The cost associated with revoking a user manifests itself in the re-encryption effort required to secure confidentiality. It is not possible to physically revoke a user's keys to prevent that user's ability to perform operations since copies could have been produced, so the system must render all keys of a revoked user obsolete and re-encrypt all of the data with a new key. A resulting argument then turns, once again, to a tradeoff between security and performance. There are two primary methods for securing the data after key revocation: lazy revocation and aggressive revocation. When using lazy revocation the system does not re-encrypt the data that the revoked user previously had authorization to access until the next valid user attempts to access the file. This essentially defrays the cost over time, but it leaves data vulnerable to the revoked user for an unspecified period of time. By contrast, aggressive revocation immediately re-encrypts all files that the revoked user could potentially access. Once re-encrypted, new keys must be distributed to all personnel who are affected by the changed encryption (adding additional weight to the key distribution scheme); clearly this option requires time. Lazy re-encryption sacrifices a measure of security to save time while aggressive revocation sacrifices time to improve security.

2.2 Integrity

Integrity is a broadly based topic that includes maintaining data consistency in the face of both accidental and malicious attacks on data. For the purposes of this paper, the scope of the integrity analysis is limited to the methods used to prevent malicious alteration or destruction of information. The resulting expectation is that when a user accesses stored information, no data has been subjected to unauthorized modification. Many systems enforce integrity by ensuring that data comes from the expected source. For stored data, the discussion of integrity implies that files have not been changed on the disk.

Integrity enforcement procedures fall into two categories: data modification prevention and data modification detection. Similar to confidentiality, modification prevention requires users to receive authorization prior to changing files and requires that files are only changed in an approved manner. Integrity differs from confidentiality in that confidentiality is only worried about whether or not data has been compromised, whereas integrity includes ensuring the correctness of the data. Detection schemes generally assume that attacks are inevitable and that there must be suitable ways to assess any damage done, recover from the damage, and apply lessons learned to future prevention mechanisms.

2.3 Availability

The paper considers availability in terms of time, space, and representation. Information needs to be available to an authorized user within an acceptable time period, without monopolizing the available storage space, and in an understandable representation. A system can not allow an adversary to prevent authorized access to information via a denial-of-service attack. It is important to note that the goals of availability conflict to a degree with those of confidentiality, because ensuring confidentiality often requires increased time to access data or offering access only within a limited environment. The two characteristics, however, must be considered within the security domain.

2.4 Performance

The level of security and the system performance often conflict. In order to provide the requisite layers of security to avoid harmful attacks, the system performance suffers. The two goals of an efficient system and a secure environment intrinsically conflict. Each additional security measure requires computationally expensive processing that detracts from the system's ability to perform other operations; all security measures are overhead for the system. Each of the evaluated storage techniques attempts to minimize the performance cost associated with the particular measures of the system. The most dominant performance cost is associated with encryption due to its computationally expensive nature. The two fundamentally different approaches to storage security, encrypt-on-wire and encrypt-on-disk, place the burden of encryption on different aspects of the system. Riedel et al [1] provide a detailed explanation of the tradeoffs between the two.

3. SURVEY

The following survey briefly describes five storage security approaches using confidentiality, integrity, availability, and performance as a framework. Sections 3.1 and 3.2 refer to encrypt-on-wire systems that store data in clear text and encrypt upon transmitting the data across the network. Sections 3.3 through 3.5 refer to encrypt-on-disk systems that store data encrypted and require no additional encryption prior to transmission.

3.1 NASD – Network Attached Secure Disks

In traditional distributed file systems, a client must make a request for file access via the file server. The server then must verify the client's authorization and distribute the file if the appropriate criteria are met. Since the server must interact with every file access request for every client, the server can quickly become a bottleneck. NASD's primary goal is to relieve the server bottleneck by interacting with a user one time providing a "capability key." With the capability key the user can access the appropriate disk(s) directly without any further server interaction. The disks themselves must be "intelligent" such that they possess enough internal ability to process the capability key and handle file access requests directly [2, 3].

<u>Confidentiality.</u> There are two servers in the NASD design, one to provide authentication and the actual file server. NASD does not specify the authentication scheme and recommends using any existing third party method similar to Kerberos. Upon receipt of authentication, a user sends a request to the file server. The server verifies the authenticity of the request and then provides the user with a capability key that corresponds to the user's rights for file access. After obtaining the capability key, a user can communicate directly with the data disk for all future access requests during a given session.

The capability object is the critical aspect pertaining to both the confidentiality and integrity of the system. A file manager agreeing to a client's access request privately sends a capability token and a capability key to the client;

together these form a capability object. The token contains the access rights being granted for the request and the key is a message authentication code (MAC) consisting of the capabilities and a secret key shared between the file server and the actual disk drive. Clients can then make a direct request to a NASD drive by providing the capability object. Using its internal processor, the drive then uses the secret key that it shares with the file server to interpret the capability token to verify the user's access rights and service the request. Since the MAC can only be interpreted using the drive/server shared secret key, any modifications to the arguments or false arguments will result in a denied request.

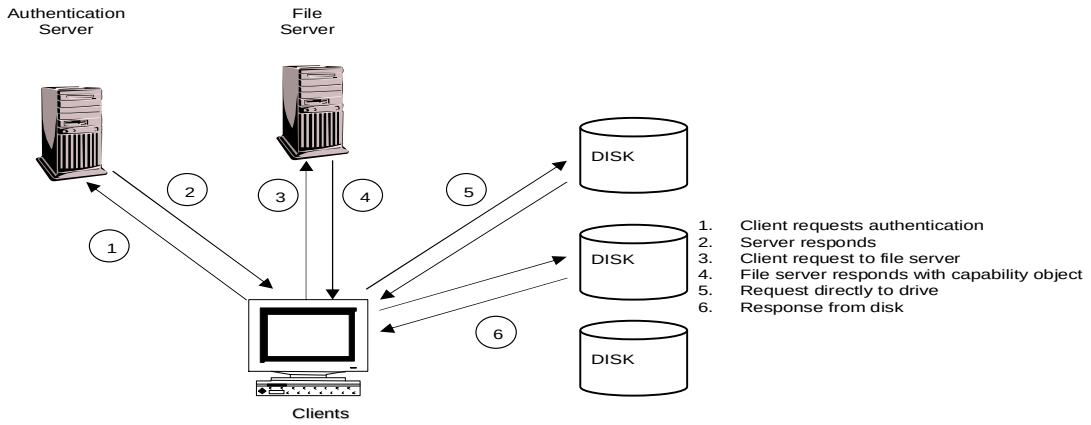

Figure 1 NASD

Integrity. The novel concept associated with NASD is placing part of the data integrity requirement on the disks themselves. The "intelligent" disks interpret the capabilities objects, encrypt data, and transmit results to clients. To ensure integrity on the client end, the disk uses the same hash MAC combination that allowed it to authorize a client access to encrypt and send the data to the client. The client can then verify the integrity of the transmission during the decryption process.

Availability. The fact that NASD allows direct access to the disk promotes scalability; the system throughput scales linearly with the number of clients and disks. However, since the file server must be trusted to initially provide capability keys, the server presents a single point of attack. If the server becomes compromised there is no way to prevent a denial of service attack.

Performance. A motivating factor for using NASD is the ability to scale bandwidth linearly with the number of disks in the system; however, these benefits are partially offset by the cost of cryptography. A large performance problem associated with NASD is the dual cost of cryptographic operations incurred because it is an encrypt-on-wire scheme. Every data transmission must be encrypted prior to being sent and then decrypted at its destination. In an attempt to reduce the performance penalty, NASD uses a "hash and MAC" cryptographic approach instead of a standard MAC. In a traditional MAC algorithm, a client's secret key is used throughout the computation. In contrast, hash and MAC uses the raw data from the file to pre-compute a series of message digests that are generic for the given file. Hash and MAC then applies a client's secret key to the message digests only as a client requests a file. The result is that the secret key is only required for a small subset of the overall computation, thus significantly decreasing latency associated with on-the-fly cryptography. Experiments demonstrated that the latency for using cryptographic operations was bounded by a 20% increase in the time to service a request when compared to a request with no cryptography [2].

3.2 Self Securing Storage (S4)

S4 is a self-securing storage medium that introduces a new aspect to storage security: the disks do not trust even the host machine operating system. S4 treats all requests as suspect. The driving security motivation for the system has been to negate the effects of a clever intruder who is able to successfully penetrate the operating system and disguise any adversarial efforts. The disks themselves in S4 require a small set of fundamental operations for managing a file system and therefore have an embedded instruction set for internally versioning and auditing all data and metadata. S4 uses a

daemon on the client machine to service file access requests as remote procedure calls and then translate them into S4-specific requests to make the system transparent to users. [4, 5, 6, 7].

Confidentiality. S4 does not provide any method for authentication, but rather assumes that the self-securing disks will be used in conjunction with a file server that uses one of many standard authentication protocols. All access requests properly sent to S4 will be serviced without any additional verification.

Integrity. An underlying theme for the design of S4 is that confidentiality will eventually become breached in any system. S4 uses a comprehensive versioning protocol that creates a new version of every modified block and creates a journal entry of metadata attributes for every file access performed within a detection window. The period of the detection window is largely defined by the amount of space available for keeping the overhead of multiple versions and has proven to be approximately two weeks in research [7]. Considerable research has resulted in a combination of a journal-based structure and multi-version B-trees to efficiently keep all of the requisite information in a space efficient manner. If at any time during the detection window an intrusion occurs, the system can guarantee the integrity of all data up to the point of the intrusion. Additionally, after verifying the access logs for individual files, files can be safely recovered if no anomalies exist. Files that have questionable accesses in the post-intrusion period may result in lost data, but the user can still be guaranteed not to receive the tainted file. This is a critical aspect of S4, other systems may allow a user to unknowingly access a file modified during an intrusion, but S4 will prevent such an integrity violation. The system administrator establishes the detection window, and once set, no user can prevent any access to data from being versioned. This precludes an intruder from altering a file in an undetectable fashion.

Availability. Through comprehensive versioning, self-securing storage inherently ensures data is available to users. Immediately after the system administrator detects an intrusion, all of the files can be reliably restored to the last access prior to the intrusion. Legitimate changes made after the intrusion but prior to its detection may result in lost data, but the lost data can be minimized. S4 includes a storage based intrusion detection scheme that can analyze file modifications for suspicious behavior [22]. If, for instance, an intruder attempts to change the contents of a password file which the administrator is "watching," the intrusion can be detected immediately. By minimizing the deleterious effects of an intrusion, the system naturally increases data availability.

Performance. Considering the costs of maintaining versions for every file access, the designers searched for both space and time efficient means for achieving comprehensive versioning. They use journal-based metadata and multi-version b-trees to meet their objectives. The system has proven to operate at a comparable speed with traditional NFS for current-file lookup. For back-in-time access, the lookup time is dependent on the number of versions which must be traversed. However, this can be bounded by a system administrator's determination to checkpoint the file system.

3.3 SFS-RO – Secure File System – Read Only

SFS-RO relies on self-certifying path names to provide high availability to read-only data in a distributed environment. SFS-RO uses some of the concepts from its SFS predecessor, but strives for better performance by providing read-only data that does not require any server-based cryptographic operations. The concept is to preserve data integrity while producing multiple copies of read-only material; traditionally such copying resulted in a degradation of security [8, 9].

Confidentiality. SFS-RO relies on a mutual authentication protocol between the users and the server, performed via self-certifying pathnames that have the public key for a file embedded in them. The creator of the file has the ability to assign the key, offering a wide range of cryptographic options. To properly encrypt files, an administrator bundles the contents of the file system into a database that is signed with a digital signature containing the private portion of an asymmetric key. Once signed, the database can be replicated and distributed to many untrusted machines without the threat of compromise. In order to access the files, a user must provide the location of the storage server (either a DNS hostname or IP address) and a *HostID*. The HostID is a cryptographic hash of the server location and the public portion of the asymmetric key with which the file creator encrypted the database. The database creator must provide the public key to all potential users separate from the SFS-RO system.

Once granted permission to the files via the mutual authentication, the users can then access files by providing the appropriate *handle*, comprised of a cryptographic hash of the file's blocks. Groups of handles are recursively hashed

and stored in hash trees such that the handle to the root inode provides the ability to verify the content of individual file blocks, reducing the number of handles required throughout the system. Knowing the handle of the root inode provides a client with the ability to verify the contents of a given file by recursively checking the hashes.

Integrity. SFS-RO relies on three critical elements: the SFS database generator, the SFS read-only server daemon, and the client. Traditional directories are converted to a database and digitally signed in a secure client environment. This database is then distributed to any number of servers that all run the SFS-RO server daemon. The server daemon simply receives requests from clients to look up and return data. The SFS client runs on a client machine and is a conduit between a standard file system protocol (like NFS) and the server. Upon receipt of a file transmission, it converts the SFS-RO database "chunks" into traditional inodes and blocks that a typical file system would expect to see. Additionally the client must posses a private key to verify the digital signature on data passed from the untrusted server. This verification process ensures data integrity.

SFS-RO also uses a timestamp protocol to help detect integrity violations. When a user creates a database, the time is recorded. Additionally, the creator must establish a no-later-than time to resign the database so that the time has an upper and lower bound. Users of the files maintain a record of the current timestamp which they compare against all data that they receive to prevent a rollback attack.

Availability. One of the primary goals of SFS-RO is to extend access to read only data in a global environment. To accomplish this, a file system creator can copy the securely generated database onto any server that is running the SFS-RO daemon. The result is a system that scales to the number of servers multiplied by the number of connections per server. Since the system is designed for multiple copies of read-only material to be distributed among multiple machines, it is logical to deduce that destruction of one server will not affect the availability of the file.

Performance. All cryptographic operations are performed on client machines; the creator establishes the database in a secure non-networked environment and users receive encrypted data which they must decrypt on their client machines. The cryptographic operations proved to be the most costly aspect in comparison to traditional NFS. For small files, SFS-RO was twice as slow as NFS with the primary additional cost due to timestamp verification (to ensure integrity). SFS-RO incurs additional latency when compared to NFS because NFS is run in the kernel while SFS-RO must rely on system calls. In larger files where the proportion of system calls to data passed is smaller, the slow down was approximately 30% which, while much improved over small files, is still a considerable performance penalty. SFS-RO began to perform favorably with very large files (40MB), proving 4% faster than NFS.

3.4 PLUTUS

PLUTUS is a lockbox scheme that provides highly scalable key management while providing file owners with direct control over authorizing access to their files. All data is encrypted on the disk with the cryptographic and key management operations performed by the clients to alleviate server cryptographic overhead. Users can customize security policies and authentication mechanisms for their own files using the client-based key distribution scheme. This places the responsibility for key management on the user, forcing the file owner to ensure proper secure distribution of keys to those they wish to authorize access. A prospective user must contact the owner in order to get the appropriate key [10]. Riedel et al [1] argue that this task can be performed with an acceptable cost to user convenience.

Confidentiality. To provide the liberty of owner customization while ensuring confidentiality, PLUTUS relies on an intricate lockbox scheme with multiple levels of keys. At the data level, PLUTUS uses a block structure to encrypt each individual block with a unique symmetric key. These block keys are then encrypted within a lockbox accessed via a file-lockbox key common to all files within a filegroup. The filegroup owner creates the file-lockbox key when the file is created and then distributes it to all users. PLUTUS uses an asymmetric file-verify key or a file-sign key protocol to differentiate between readers and writers respectively (see Figure 3). These keys are used to sign or verify a cryptographic hash of the file block contents to provide integrity. Upon requesting a file, the server passes the encrypted lockbox and encrypted block contents to the user. The user then "unlocks" the lockbox with the file lockbox key and decrypts each block with its respective file block key.

The proliferation of keys and the use of filegroups in PLUTUS complicate the key revocation scheme. When multiple files across the file system (related only by access rights, not like a traditional directory) are encrypted with the same key, key revocation could cause mass re-encryption and key management problems. However, the designers have implemented a clever key rotation scheme that minimizes the effects. PLUTUS uses lazy revocation such that a revoked user can still read files that were accessible at the time of revocation. A problem arises due to the use of filegroups because upon re-encrypting a file, different files within the same group will require different keys. Since a primary motivation for using filegroups in the first place was to minimize the number of keys, Kallahalla et al [10] designed a rotation scheme that ensures that the new encryption key is related to the keys for all files in the filegroup. The system performs the re-encryption with the latest filegroup keys, but all valid users can generate previous versions from the latest key. The result is that all valid users can "regenerate" the proper key for a given file if they have the latest filegroup key. The new filegroup key is only disseminated to currently valid users such that revoked users.

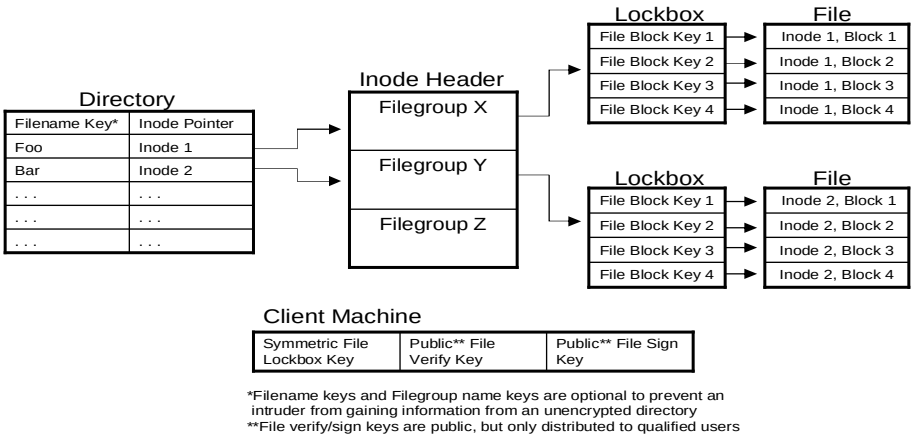

Figure 2 PLUTUS Key System

Integrity. PLUTUS does not trust the file server and cannot, therefore, rely on it to distinguish between writers and readers. Instead it uses two types of keys, file-sign keys and file-verify keys, to make the respective determination. Upon attempting to read or write, the user verifies the digital signature and hashed contents of the file with these keys. If the user obtains unexpected results, the user can determine that the file has been illegally modified.

Availability. The entire design of PLUTUS is intended to provide scalability. Placing the key management responsibility on the clients instead of on a trusted server prevents a server bottleneck due to computationally expensive cryptographic operations. PLUTUS relies heavily on filegroups to limit the number of cryptographic keys. Filegroups consist of all files with identical sharing attributes and can, therefore, be protected using the same key. This allows users with filegroup privileges to access a file within the group even if the owner is not on-line, avoiding the requirement for a user to contact the owner directly to get the key with every file access. The filegroup concept does not rely on a hierarchical structure so that the grouping is strictly a product of the associated files' permission attributes.

PLUTUS does provide much greater scalability and a clever key rotation process to minimize key management responsibilities associated with key revocation, but the system still requires the file's owner to provide a copy of the file's symmetric key to each user. Filegroups help to minimize the file owner-user communication, but they do not eliminate the original responsibility for the owner to distribute the keys.

Performance. The designers of PLUTUS used OpenAFS to construct their system. The performance of the file system itself is comparable to the unmodified OpenAFS system, because the server does not have to perform any extraneous operations during file access. However the cost to the client system where cryptographic operations are performed demonstrated to be 1.4 times slower than SFS. The authors present a credible argument to justify this decrease in performance, because they used worst-case scenarios to derive their figures. This comparison only takes into account a single file read and write combination, ignoring the design enhancement offered by PLUTUS. PLUTUS is designed for

scalability and to relieve server bottleneck which may arise with multiple file access requests in standard SFS. Kallahalla et al [10] reason that the average latency for protracted use would favor the use of PLUTUS.

3.5 SiRiUS – Securing Remote Untrusted Storage

SiRiUS is designed to provide its own cryptographic read-write file access on top of any existing untrusted networked file system (NFS, CIFS, Yahoo, etc.). Via a software daemon, the system intercepts all file access system calls and converts them accordingly. The concept is to be able to establish a secure file sharing environment without significantly modifying the performance of an existing network storage medium. SiRiUS can provide security to an existing system without requiring any hardware modifications; the developers view the system as a "stop-gap" measure to provide additional security to existing systems. Often times, organizations cannot afford to upgrade their current systems and must continue to operate with limited security until which time the option to upgrade security measures becomes available; SiRiUS can provide an interim solution [11].

Confidentiality. All files are encrypted in a secure environment prior to being stored on the server, such that neither the server nor the server administrator ever has access to unencrypted data. Additionally, the computationally costly encryption operations are performed on the relatively lightly loaded client machine, and the fact that the data is already encrypted obviates any requirement to establish a secure channel to send the file to the server. Each file owner maintains a master encryption key (MEK) and a master signing key (MSK). Each file has a unique symmetric file encryption key (FEK) provided to all users and a file signing key (FSK) provided only to authorized writers to the file. The system provides a "freshness guarantee" by maintaining a metadata freshness file for each directory. All files are separated into two parts: an md-file metadata file and a d-file data file. The metadata file contains a block for the file owner's MEK, a block for every valid user's FEK (and FSK if authorized to write to the file), and a block with a hash of the metadata file's contents signed with the owner's MSK. If the owner or a user has a key maintained in the file's metadata, that person can decrypt the file. User key revocation is quick and efficient; the file owner removes the revoked user's key block from the metadata file, creates a new FEK, re-encrypts the file with the new key, and then updates the remaining users' key blocks with the new FEK. The result is immediate revocation.

Integrity. In addition to added measures, SiRiUS keeps certain file system specific metadata unencrypted so that the file system can perform standard integrity checking operations. SiRiUS keeps all access control information encrypted with the file data. This facilitates using the legacy file system's standard backup procedures – if the system must recover from a crash, all of the needed access information is already available with the file. SiRiUS uses the "freshness guarantee" to ensure that users have the most current version of a file preventing a rollback attack. At a user-designated interval, the user timestamps the metadata freshness file.

Availability. The design decision to make no modifications to the underlying file server prevents SiRiUS from defending against denial of service attacks; an attacker could conceivably compromise the server and delete all files. SiRiUS has no ability to intervene in such a circumstance and, therefore, requires users to backup their own files on multiple servers to limit the effects of such an attack. In order to add users, the perspective reader/writer of a file must send a public key to the file owner who will then use the public portion of the MEK to encrypt that key and add it to the files metadata. Once the new user's key is added to the metadata, the user has access to the file. The key passing mechanism is not addressed in the system. As previously addressed, SiRiUS supports active key revocation such that once a user's access rights have been revoked, the user no longer has any form of access to the file via the freshness guarantee. SiRiUS attempts to improve on other secure networked file system designs by allowing fine grained file access while providing the ability for a file owner to grant read-only or read-write access to shared files. Other systems either allow access to entire directories or cannot distinguish between readers and writers.

Performance. The first time a file is accessed, the file system must return the associated metadata file as well as the original (to support appropriate authorization checking). The metadata file is then cached to prevent the overhead of looking up and sending the metadata file on subsequent retrievals for the same file. Additionally, many SiRiUS file system calls require checking the freshness file resulting in increased network traffic and additional file I/O for each. 63% of the cost associated with using SiRiUS for a 1MB data read is due to the decryption cost. Similarly, 40% of the cost for writing a 1MB file is due to signature generation.

4. CLASSIFICATION AND COMPARISON

Table 1 provides a summary of the major characteristics of each surveyed system. Table 2 provides a quantitative scale (0-2) for assessing security attributes. 0 denotes that the security measure does not affect the attribute, 1 denotes that the measure enhances security, 2 denotes that the measure significantly enhances security. For performance, the scale is modified such that a 0 denotes a significant, a 1 a moderate, and a 2 a limited performance penalty (all systems suffer some performance degradation as a result of added security measures). While these figures are based on empirical data derived from the authors' research of each system, it is important to note that they are a subjective representation.

Storage System	Encrypt Location	Trust Server	Key Revocation Policy	Confidentiality Measures	Integrity Policy	Availability Policy	Estimated Performance Overhead
NASD	Wire	Yes	Aggressive by using timestamps, key issued one time	Capability keys, separate authentication server	Hash MAC checksums to send data, not secure on disk	Scalable to many users, subject to DOS	20% increase over system with no security
S4	Wire	No	N/A	N/A	Detection scheme: Comprehensive versioning	Intrusion detection and diagnosis to provide "recent" version	Comparable to NFS
SFS-RO	Disk	No	Revocation list	Self-certifying pathnames	Self-certifying pathnames, timestamps	Multiple distributed copies of RO files	2 times slower than NFS for small files, comparable for files > 40 MB
PLUTUS	Disk	No	Lazy revocation, revoked user retains same file permissions as at time of revocation	Lockbox with user control over key dissemination. Users must secure distribution themselves	Stored encrypted, but requires augmentation to ensure integrity	Uses filegroups, but requires file owner to distribute keys	1.4 times decrease from SFS for single file access
SiRiUS	Disk	No	Aggressive Revocation	Combination of Master Encryption Key and File Encryption Keys	Freshness guarantee timestamp	Scalable to Internet, but requires file owner to distribute FEK	20 times slower than NFS for small files, 2-6 times slower for 1MB files

Table 1 System Description

Security Measure	Confidentiality	Integrity	Availability	Performance	Total
Encrypt-on-disk	2	2	0	0	4
Encrypt-on-wire	1	1	0	0	2
Threshold scheme	1	1	2	2	6
Timestamps	0	2	0	2	4
Digital signatures	0	2	0	0	2
Checksums	0	1	0	1	2
Lazy revocation	1	0	2	2	5
Aggressive revocation	2	1	1	0	4
Key distribution server	1	0	1	1	3
Manual key distribution	2	0	0	0	2
Lockbox key mechanism	2	0	1	1, 2*	4,5*
Self-Certifying pathnames	1	0	2	1	4
Filegroups	0	0	1	1	2
Comprehensive versioning	0	2	2	2	6

Table 2 Quantitative Comparison of Security Measures

*PLUTUS' key rotation scheme enhances performance.

Security Measure	NASD	S4	SFS-RO	PLUTUS/ SNAD	SiRiUs
Encrypt-on-disk			X	X	X
Encrypt-on-wire	X	X			
Threshold scheme					
Timestamps	X		X		X
Digital signatures			X		
Checksums	X			X*	
Lazy revocation			X	X	
Active revocation	X				X
Key distribution server	X				
Manual key distribution					
Lockbox key mechanism				X	
Self-certifying pathnames			X		
Filegroups				X	
Comprehensive versioning		X			

Table 3 Security Measure Applications to Surveyed Systems

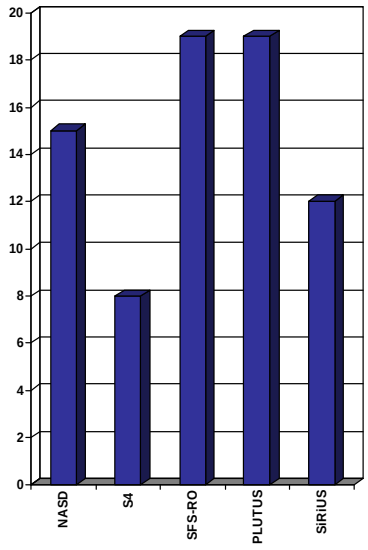

Figure 3 Quantitative Comparison

Table 3 maps the security measures to the systems that employ them. The quantitative values in Figure 3 are derived by summing the total scores of each security measure that applies to each system. While the numbers provide some measure for comparison, one must consider the purpose of each system in conjunction with the evaluation. For instance, S4 is a survivable storage system which should be used as part of a larger, secure storage system. Most authors mention that using their design in conjunction with a survivable storage system would provide the best security. Additionally, SFS-RO receives a high quantitative score, but it is limited to read-only applications. SiRiUs receives a moderate score, but potential users must account for its performance penalties.

5. APPLICATIONS TO MULTIMEDIA

Multimedia applications generally present a restrictive set of criteria for storage because performance is critical; real-time deadlines must be met for data retrieval or the application may degrade to an unacceptable level. The design of certain systems may eliminate them from consideration immediately. Specifically, the following secure systems were omitted from the survey because they do not lend themselves to multimedia applications. Pasis [4, 25] distributes file data among multiple servers such that compromising a portion of the data will not reveal anything significant about the original file. The system is heavily reliant on network traffic and would likely not be conducive for use with any real-time application. Secure Network Attached Disks (SNAD) [12, 13] uses a lock-box key management mechanism similar to PLUTUS and was omitted in favor of PLUTUS due to PLUTUS' key rotation scheme. The Cryptographic File System (CFS) [14] is one of the original encrypt-on-disk schemes that others have used as a building block, but it is almost obsolete.

Of the systems included in the survey, some lend themselves better to specific types of multimedia applications. Some applications require the ability to make changes to data, while others simply need a secure location to store a read-only file. Table 4 lists several multimedia applications and matches a suitable secure storage solution to the specific requirements of each. For an application that does not require any modifications, SFS-RO provides a high degree of security along with scalability and accessibility, if, however, data must be changed SFS-RO is not an option. For applications that may need to be modified, PLUTUS provides a high degree of security, but can potentially suffer from server bottleneck and is therefore not appropriate for deadline dependent applications. NASD, by contrast, provides a secure solution that eliminates the server bottleneck. SiRiUs can be layered on top of any existing networked storage solution and can be applied to any application. It is a suitable system to provide immediate security to applications prior to implementing a long-term solution. S4 can be used in conjunction with any of the other systems, but requires additional hardware.

Application	Modifiable	Deadline Dependent	Recommended System
Streaming video/audio	No	Yes	SFS-RO
Downloaded video/audio	No	No	SFS-RO
Images	No	No	SFS-RO
Graphics	Yes	No	PLUTUS
Interactive features	Yes	Yes	NASD
Slide shows/presentations	Yes	No	PLUTUS

Table 4 Application to System Mapping

6. CONCLUSIONS

All of the systems share a common goal: to protect stored data from the effects of a malicious adversary. From this common ground, however, the design approaches to reach this goal vary tremendously. Some systems aim to prevent an adversary from ever having access to data, while others assume that intrusions are inevitable and try to limit the amount of damage an intruder can cause. Some systems separate data onto multiple storage servers to eliminate a single point of attack, and others rely on centralized trusted servers to effectively manage cryptographic keys. Some systems store encrypted data and others require encryption prior to transmitting messages on the wire. All of these examples present large fundamental differences that provide options to potential users of a storage security medium. It is very difficult to make direct comparisons between the systems because of the varied approaches, but potential users can select the most applicable solution to their specific problems.

The most secure solution will likely be a combination of the systems described. In fact, the majority of the designers of the systems recommend that their solution be part of a larger security plan. For example, if a user can accept additional cryptographic latency there is no reason to avoid encrypting data before applying a threshold scheme. The result would provide the security of encryption without relying on a trusted server and would increase the degree of availability. The problem with such layering, however, is the performance penalty. It is therefore a design requirement to analyze the tradeoffs between security and performance.

REFERENCES

1. E. Riedel, M. Kallahalla, and R. Swaminathan, “A Framework for Evaluating Storage System Security,” *Proceedings of the 1st Conference on File and Storage Technologies*, January 2002.
2. H. Gobioff, D. Nagle, and G. Gibson, “Embedded Security for Network-Attached Storage”, *CMU SCS technical report CMU-CS-99-154*, June 1999.
3. H. Gobioff, G. Gibson, and D. Tygar, "Security for Network Attached Storage Devices," *CMU SCS technical report CMU-CS-97-185* 1997.
4. G. Ganger, P.K. Khosla, M. Bakkaloglu, M.W. Bigrigg, G. Goodson, S. Oguz, V. Pandurangan, C. Soules, J. Strunk, and J.J. Wylie, “Survivable Storage Systems,” *DARPA Information Survivability Conference and Exposition*, pp. 184-195 vol 2. IEEE, June 2001.
5. A. Pennington, J. Strunk, J. Griffin, C. Soules, G. Goodson, and G. Ganger, “Storage-based Intrusion Detection: Watching Storage Activity For Suspicious Behavior,” *12th USENIX Security Symposium*, August, 2003.
6. C. Soules, G. Goodson, J. Strunk, and G. Ganger, “Metadata Efficiency in Versioning File Systems,” *2nd USENIX Conference on File and Storage Technologies*, April, 2003.
7. J. Strunk, G. Goodson, M. Sheinholtz, C. Soules, and G. Ganger, “Self-Securing Storage: Protecting Data in Compromised Systems,” *4th Symposium on Operating System Design and Implementation*, October 2000.
8. K. Fu, M.F. Kaashoek, and D. Mazieres, “Fast and Secure Distributed Read Only File System,” *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation*, pages 181-196, October 2000.
9. D. Mazieres, M. Kaminsky, M. Kaashoek, and E. Witchell, “Separating Key Management From File System Security,” *17th ACM Symposium on Operating Systems Principles*, pp. 124-139, December 1999.
10. M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, “PLUTUS: Scalable secure file sharing on untrusted storage,” *Conference on File andStorage Technology*, pp. 29-42, April 2003.

11. E. Goh, H. Shacham, N. Modadugu, and D. Boneh, “SiRiUS: Securing Remote Untrusted Storage,” *Proceedings of the Internet Society (ISOC) Network and Distributed Systems Security (NDSS) Symposium*, 2003.M. Blaze, “A Cryptographic File System for Unix,” *First ACM Conference on Communications and Computing Security*, November, 1993.
12. E. Miller, D. Long, W. Freeman, and B. Reed, “Strong security for distributed file systems,” *Proceedings of the 20th IEEE International Performance, Computing and Communications Conference (IPCCC '01)*, pp. 34–40, April 2001.
13. E. Miller, D. Long, W. Freeman, and B. Reed, “Strong security for network-attached storage,” *Proceedings of the 2002 Conference on File and Storage Technologies*, pp. 1–13, January 2002.
14. Matt Blaze, “A Cryptographic File System for Unix,” *First ACM Conference on Communications and Computing Security*, November, 1993.
15. M. Bakkaloglu, J.J. Wylie, C. Wang, and G.R. Ganger, “On Correlated Failures in Survivable Storage Systems,” *CMU SCS Technical Report CMU-CS- 02-129*, May 2002.
16. W. Freeman and E. Miller, “Design for a decentralized security system for network-attached storage,” *Proceedings of the 17th IEEE Symposium on Mass Storage Systems and Technologies*, pages 361–373, March 2000.
17. G. Gibson, D. Nagle, K. Amiri, F. Chang, H. Gobioff, E. Riedel, D. Rochberg, and J. Zelenka, "Filesystems for Network-Attached Secure Disks," *CMU Computer Science Technical Report, CMU-CS-97-118*, July 1997.
18. G. Goodson, J.J. Wylie, G. Ganger and M. Reiter, “Decentralized Storage Consistency via Versioning Servers,” *Carnegie Mellon University Technical Report CMU-CS-02-180*, September 2002.
19. G. Goodson, J.J. Wylie, G. Ganger, and M. Reiter, “A Protocol Family for Versatile Survivable Storage Infrastructures,” *Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-03-104*, December 2003.
20. D. Mazieres and D. Shasha, “Building Secure File Systems Out of Byzantine Storage,” *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, pp. 108-117, 2002.
21. B. Reed, E. Chron, R. Burns, and D. Long, “Authenticating network-attached storage,” *IEEE Micro*, 20(1):49–57, January 2000.
22. J. Strunk, G. Goodson, A. Pennington, C. Soules, and G. Ganger, “Intrusion Detection, Diagnosis, and Recovery with Self-Securing Storage,” *CMU SCS Technical Report CMU-CS-02-140*, May 2002.
23. C. Wright, J. Dave, and E. Zadok, “Cryptographic File Systems Performance: What You Don’t Know Can Hurt You,” *IEEE Security in Storage Workshop (SISW 2003)*, October 2003.
24. C. Wright, M. Martino, and E Zadok, “NCryptfs: A Secure and Convenient Cryptographic File System,” *USENIX 2003 Annual Technical Conference*, June 2003.
25. J.J. Wylie, M. Bigrigg, J. Strunk, G. Ganger, H. Kiliccote, and P. Khosla, “Survivable information storage systems,” *IEEE Computer, 33(8):61-68*, August 2000.