# SGFS: Secure, Efficient and Policy-based Global File Sharing

Vishal Kher
*University of Minnesota*
*vkher@cs.umn.edu*

Eric Seppanen
*University of Minnesota*
*seppanen@cs.umn.edu*

Cory Leach
*University of Minnesota*
*leach@cs.umn.edu*

Yongdae Kim
*University of Minnesota*
*kyd@cs.umn.edu*

## Abstract

*This paper presents SGFS - a secure global file sharing system. SGFS is designed based on important design requirements that include: efficiency for high performance data access, flexibility of cross-domain file sharing without administrative interference, support for flexibly policies and off-the-shelf policy managers, ability to be deployed in diverse environments, ease of management and low administrative overheads. Unlike existing systems that satisfy a proper subset of these requirements, SGFS is designed to satisfy all of these requirements. In this paper, we present the architecture and design of SGFS. We illustrate how these requirements have influenced our design and present the implementation of the SGFS user-space prototype.*

## 1. Introduction

There is a rising trend of collaboration and global sharing of information across multiple domains. For example, consider a group of faculty members in a certain University that want to collaborate with a group of faculty members from another University and share their project's source code repository and experimental results. As another example, consider a group of scientists who want to share files generated by their simulation applications with scientists from a different organization to allow them to analyze their data and share the knowledge gathered through those results. In both of these examples, local users need to freely share data and collaborate with remote users. In addition, in the second example, users need high performance data access.

Existing cross-domain file sharing systems [4, 20, 25] are not tailored for high performance data access. These systems assume that users have to share a small number of files (e.g., class project files or photos) and also assume that these files are stored on traditional centralized servers. Recently, network-attached intelligent storage devices [17, 21, 23, 27, 28] have gained importance in industry and academia. These devices enable low-latency data transfers directly between the client and the storage device to provide high performance data access. They utilize the available embedded processing power, which is typically less than general purpose servers, to perform activities such as block management [17, 27], remote execution [2, 29], search and indexing [18], and light-weight security operations [16,30]. One can envision a network of heterogeneous storage servers[1] composed of traditional storage servers as well as special purpose storage devices providing *fast global data access* to its clients. As a result, cross-domain file sharing systems should be designed to work efficiently, not only in the presence of traditional storage servers, but also in the presence of such intelligent storage devices.

As can be seen from these aforementioned examples, collaboration can be performed between two independent parties that may not have any pre-established administrative relationships. Therefore, existing solutions such as Kerberos [22,26] do not work in these settings as in order to use Kerberos the administrators of the two domains should collaborate in order to setup their systems for cross-domain authentication, which is known to be tiresome and may not be always feasible. In practice, very few independent organizations actually setup joined Kerberos realms. If the collaborating users' realms are not joined, then the only way to collaborate is to set-up accounts for remote collaborators. Therefore, user-to-user delegation, that is delegation of access from one user to another is important to increase flexibility of file sharing and reduce administrative burden.

Typically, delegation of access rights from one user to another is performed using X.509 certificate chains [1]. In order to access files, users present a signed request and en-

[1] A storage server can be any entity that serves data, for example, traditional file servers, network-attached disks, etc.

tire chain of certificates to the storage servers [4, 20, 25]. In this case, the storage servers verify the chain of certificates before granting access to users. Verifying certificate chains involve traversing trust hierarchies to find common ancestors. During this process, the storage server has to verify multiple public-key signatures, which is a computationally expensive process and may require accessing remote databases. This verification process will increase access latencies since all these operations have to be performed *during data-path*. The whole purpose of storage devices is to allow fast and direct access to data. Therefore, verification of certificate chains on these devices should be avoided, which will allow them to utilize their CPU without any disruption to perform assigned tasks, such as search and indexing, high-performance data delivery, versioning, etc. Further, verification of certificate chains can introduce access latencies even when used on traditional storage servers. Therefore, a light-weight authorization mechanism is always desirable.

The main contribution of this paper is to present a complete system that is designed to provide efficient global file sharing without any administrative interference. It is designed to work with existing policy managers [7, 8] so that system administrators can set appropriate local policies. The system ensures that users behave according to these policies. SGFS offers great flexibility, low administrative overhead, and can be used in diverse environments. The authentication protocols are designed to be efficient and resilient to central point of failures. These protocols are carefully tailored for network-attached storage devices (they also work efficiently on traditional servers).

SGFS uses *symmetric-key certificates* (SKC) that resemble X.509 attribute certificates. SKCs achieve the nice properties of X.509 certificates, but are light-weight as compared to X.509 certificates. As a result, SGFS can support different access control models and can utilize existing policy languages that are designed with X.509 certificates in mind. In SGFS, user-to-user delegation is performed using SKC, which includes all the necessary information required by the file server to verify user credentials. To ensure traceability of delegation, the authentication protocols are designed to leave audit trails that can be used by the system administrators to selectively revoke users. Further, the use of SKC obviates the need to map remote group names to local identifiers and greatly reduces the administrative burden.

## 2. System Design

### 2.1. The SGFS Symmetric Key Certificates

In SGFS, the authentication server grants every user *symmetric key certificates* (SKC) that are used by the user to authenticate herself with the storage server and share keys with them. The SGFS SKCs are designed to mimic the X.509 attribute certificates [1], but in a symmetric key setting.

A SKC is generated by an entity $T$ for a user $U$ to be verified by a verifier $V$ as follows:

$$SKC^{U}_{T,V} = \{P^{U}_{T}, prf_{K_{TV}}(P^{U}_{T})\} \quad (1)$$

Where, $prf$ is a pseudo-random function (HMAC [5] in practice). $K_{TV}$ is the key shared between the verifier $V$ and the certificate generation entity $T$. $P^{U}_{T}$ is the public information of user $U$ defined by $T$, and $K^{U}_{T} = prf_{K_{TV}}(P^{U}_{T})$ is the secret key for $U$. If $P^{U}_{T}$ is made available to $V$, then $V$ can generate $K^{U}_{T}$, and, thus, share a key with $U$. Using $K^{U}_{T}$, $U$ can authenticate with $V$ and assert the privileges listed in $P^{U}_{T}$. A similar but restrictive type of symmetric key certificate was used in [28].

**Properties** A symmetric key certificate binds $P^{U}_{T}$ to the holder of the corresponding $K^{U}_{T}$ and the certificate has a designated verifier $V$, but the verifier does not need to contact $T$ each time to generate $K^{U}_{T}$. Similar to X.509 attribute certificates, SKCs bind information such as *identity*, *privileges*, *roles* to the holder of the key $K^{U}_{T}$. They can be long-lived and one can apply policies similar to that applied to X.509 attribute certificates. Further, the verifier does not have to contact any other online entity to verify the certificate. However, since they are based on symmetric key techniques, SKCs differ from public key certificates - they can be verified by a single designated party, they can be used by the user to establish a shared key only with a single verifier (and vice-versa), they do not provide non-repudiation, and SKCs cannot be entirely stored in a public database ($K^{U}_{T}$ is secret).

A different (and commonly known) notion of symmetric key certificates was used in [3, 10]. In these approaches, a SKC for a user $B$ is of the form $E_{K_T}(K_{BT}, B)$. Where, $K_T$ is the key known only to a central trusted entity $T$ and $K_{BT}$ is the key shared by a user $B$ with $T$. Any user $A$ (that has $E_{K_T}(K_{AT}, A)$) who wants to send an authentic message (or a key) to $B$ should encrypt the message with $K_{AT}$ and send the encrypted message, and $B$'s SKC to $T$. $T$ then translates the message from $A$ to $B$ by decrypting $B$'s SKC and re-encrypting $A$'s message to $B$. The main drawback of this approach is that $T$ has to be online to translate messages between any two entities.

**The Contents of a SKC** A symmetric key certificate is comprised of a public part $P^{U}_{T}$ (see eq. 1) and a secret part $K^{U}_{T}$. The public part contains the following information:

- a unique identifier of this SKC
- unique identification of the holder
- unique identification of the issuing server

- issuing servers DNS/IP address, if applicable
- list of privileges granted to the holder
- validity period of this SKC
- delegator
- constraints

The holder and the issuer can be identified by local user name, email address, or reference to public key. If the issuer is a server, then the identifier can be servers global identifier, such as DNS name or reference to public key. Privileges can be a list of roles, list of groups, list of file groups, etc. Constraints can restrict certain type of access. For example, they can specify if access granted to the holder is read-only or the time duration during which a user can access files. Constraints can also specify whether the holder can delegate a subset of her privileges to other users.

### 2.2. Design Rationale

In this section we explain the factors that have influenced the design of SGFS.

**Low cryptographic overhead on storage servers** In order to achieve our first goal of designing authentication protocols that impose minimal cryptographic overhead on the storage servers, we decided to *avoid performing public key operations* on the storage servers. SGFS is designed to be used for high performance data access in the presence of network attached storage devices. Therefore, we do not perform any public key operations on the storage servers and the authentication protocols are based on symmetric key operations. User-to-user delegation is also performed without using certificate chains. This allows the storage servers to perform their assigned task (indexing, self-securing, high performance data provisioning etc.) without any disruption.

**Resilience to central point of failures** To make our system resilient to central point of failures, we attempted to reduce interactions between the user and any online entity (except storage servers), especially between the users and the authentication server. If the user has to frequently contact the authentication server to get access credentials, then if the authentication server is down or overloaded with authentication requests, the user will not be able to access files even if the storage server is available. Therefore, we desire a solution in which *files could be unavailable to the user only when the storage server is unavailable.* To achieve this goal, users are granted long-term access keys. We believe that in most of the cases, changes to user credentials are infrequent. For example, in Role-based Access Control (RBAC) [14], users' roles are usually associated with their job in the organization, which do not change frequently [11,13]. Similarly in UNIX environments, a particular user's group membership does not change daily. One common tradeoff of long-lived access key is revocation. SGFS design includes revocation servers which periodically publish appropriate new revocation lists to the storage servers. The revocation server can also perform emergency updates if immediate revocation is required.

**Flexible file sharing with minimal system administrative interference** In SGFS if a local user Alice wants to share files with an external user Bob, Alice can delegate a subset of her access rights to Bob (if the policies allow her to do so). If Alice's organization policies allow Bob to delegate to other users (e.g., his group members), then Bob can further delegate the access rights acquired from Alice. User-to-user delegation does not require any system administrative interference, increases flexibility of sharing, and also reduces the management burden.

**Traceable delegation and audit trails** In practice, even if Alice is allowed to delegate to Bob, the delegation should be traceable. Audit logs should be maintained to clearly indicate the delegator-delegatee relationships. This information can be used for auditing as well as revocation. The SGFS delegation protocol is designed in such a way that Bob has to perform one time set-up with the Alice's authentication server (AS) to receive a SKC from the AS. During this process the AS can verify polices and create audit logs.

**Flexible policy support** Organizations use different policies in different settings. Many of the existing policy languages, such as PolicyManager [8], KeyNote [7], and SPKI [12], offer the ability to formally express policies making it possible to automate enforcement. These systems were designed with X.509 certificates in mind. To exploit the flexibility of X.509 certificates in a non-public key setting and at the same time use the existing policy languages, SGFS uses symmetric key certificates that contain similar information as that contained in the X.509 certificates.

## 3. System Architecture

Figure 1 depicts the SGFS architecture. It consists of five entities: authentication server (AS), policy manager (PM), storage servers, clients[2] (or end users), and revocation servers.

The AS is trusted by all other entities. It is responsible to authenticate users and give them appropriate credentials. The AS shares a unique symmetric key with every storage server and is responsible for securely managing these keys.

[2] "User" and "Client" refer to the end user of the system and are used interchangeably.

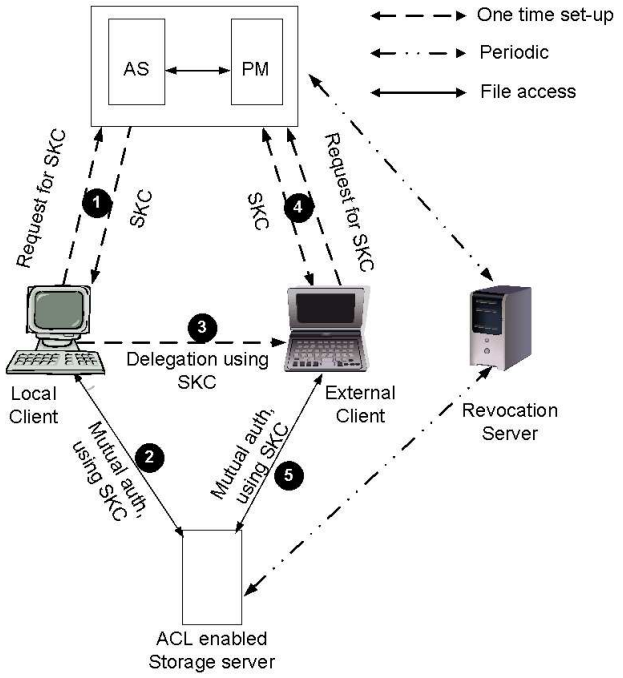


**Figure 1. The SGFS Architecture.**

It also maintains a database of local users and their associated privileges and group memberships. We assume that the AS can communicate securely with all entities. The PM is trusted to set appropriate policies.

The revocation server is responsible to store and publish revocation lists. It periodically publishes the appropriate new revocation lists to the storage servers. It can also send emergency revocation messages to storage servers, if immediate revocation is required. It is assumed that the revocation server can securely communicate with the storage servers and the AS.

Clients are not trusted. They can launch various active and passive attacks. Communication links between the clients and the storage servers are assumed to be insecure. Since in a global file sharing system a client can access files from any computer (e.g., home computer, remote domains etc.), we do not assume any time synchronization between the clients and the storage servers.

Storage servers are trusted to perform their part of authentication and authorization securely. The data stored on these servers is not encrypted. In the future, this can be performed using existing file encryptors [6, 19, 24].

### 3.1. Usage Overview

Let us denote the key shared between the AS and a SGFS storage server $S$ as $K$. Let Alice be a local user belonging to group `genomics` and Bob be an external user (belonging to a different organization). Let user Alice be denoted as $A$ and let user Bob be denoted as $B$.

**Local User Auth.** Only the AS is trusted by the storage servers. Therefore, if Alice wants to access files stored on $S$, Alice should acquire a SKC for $S$ from AS. This is denoted by step 1 in figure 1. After receiving Alice's request, AS authenticates Alice and acquires necessary information, such as Alice's group membership list, policies, and constraints. It can also consult with the PM to perform some initial policy verification. Using this information, AS issues a symmetric key certificate for Alice as follows.

$$SKC^A_{AS,S} = \{P^A_{AS}, K^A_{AS} = MAC_K(P^A_{AS})\}$$

Alice securely stores $SKC^A_{AS,S}$ and uses $K^A_{AS}$ to initiate a mutual authentication protocol (during step 2 of figure 1) with $S$. The storage server knows $K$, and, hence can generate $K^A_{AS}$ using $P^A_{AS}$ sent by Alice during the authentication process. Using $K^A_{AS}$ Alice and $S$ can authenticate each other and securely communicate with each other. After successful completion of the mutual authentication phase (step 2 of figure 1), $S$ asserts the policies and constraints listed by AS in $P^A_{AS}$. Finally, the storage server extracts the group membership information or role-privilege information (depending upon the access control model) and performs access control using the ACLs stored locally along with the files.

**Delegation** Now let us see how a local user Alice and her `genomics` group can share files with an external user Bob. Alice first generates a symmetric key certificate for Bob $SKC^B_{A,AS} = \{P^B_A, K^B_A = MAC_{K^A_{AS}}(P^B_A)\}$ and sends $SKC^B_{A,AS}$ along with her $P^A_{AS}$ securely to Bob (step 3 of figure 1). Alice includes necessary information in $P^B_A$ and includes her `genomics` group in the group membership list in $P^B_A$. The $SKC^B_{A,AS}$ is a notification that tells AS that Alice wants to add Bob to the `genomics` group.

After receiving $SKC^B_{A,AS}$, Bob can go to the AS and authenticate himself using $K^B_A$ (step 4 of figure 1). Using $P^A_{AS}$ the AS can re-generate $K^A_{AS}$ and verify $SKC^B_{A,AS}$. AS then performs policy checks, for example it verifies if Alice is allowed to delegate. If all checks succeed, AS then generates a new $SKC^B_{AS,S}$ for Bob and sends it securely to Bob. Bob has thus become a local user with rights to access files belonging to group `genomics`. As in the case of Alice, Bob uses $SKC^B_{AS,S}$ to authenticate with $S$ and access files stored on $S$. The file server does not need know that Bob is an external user (although this information is included in $P^B_{AS}$ for auditing). It only verifies that Bob has a valid SKC from the AS and grants access based on the information embedded in SKC. Further, the file server does not need to map any remote group-ids as all the necessary local group information is already in SKC. If Bob needs to acquire a key for a different storage server, Bob can use $SKC^B_{AS,S}$ again to get a new SKC from the AS.

**Transparency** It is important to note that step 1,2,4, and

5 are done transparently and the user is not aware of these operations. All SKCs are automatically stored securely at the client. Once the SKC for the storage server is available, the SGFS client initiates the authentication protocol with the server whenever necessary.

### 3.2. Summary of Protocols

In this section we present a brief summary of our authentication protocols. The details of the authentication protocols are omitted due to space restrictions[3]. The client-server mutual authentication protocol (during step 2 of figure 1) prevents replay attacks and does not require any time synchronization between the clients and the storage servers. During mutual authentication, the server has to maintain two random numbers. After mutual authentication, if secure data transfer is desired, then the storage server has to maintain one session key for the duration of the data transfer. Only symmetric key operations are performed on the storage servers, which are computationally inexpensive. Administrators of collaborating domains do not have to perform any manual co-operation. A user can securely delegate access rights to another user without any administrative intervention. If necessary, Administrators can set appropriate policies to ensure that users do not misuse their delegation powers.

SKC certificates gives us the flexibility to use SGFS in various access control models and with existing policy managers. SKCs are long-lived and can be transferred in an offline manner to local users, e.g., via email. Users do not have to frequently contact AS; therefore, users can keep accessing the data without any disruption even when the AS is down or overloaded. Further, users can securely transfer SKCs from one machine to another and access files seamlessly from any machine. SGFS authentication protocols do not require to maintain any long-term state on the storage servers. Using SKCs storage servers can make on-spot authorization decisions without having to contact any remote server. Therefore, SGFS allows secure, flexible, failure resistant and efficient global file sharing without any administrative interference.

## 4. Current Status and Future Directions

We have implemented all of the authentication protocols briefly described in section 3.1. In addition, we have implemented easy to use tools that assist users to acquire SKCs and delegate SKCs to other users. Figure 2 represents the SGFS system. The SGFS client runs in user space and is layered on the top of FSFS [9], a user space file system

[3]A full version of this paper will be available at `http://www.dtc.umn.edu/publications/publications.php`

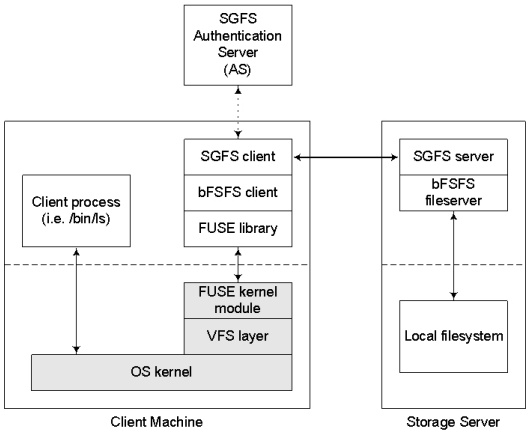


**Figure 2. The SGFS system components**

designed to be mounted through an interface provided by FUSE [15]. The SGFS server is layered on top of the FSFS server, which is a multi-threaded user space daemon that accepts clients' requests on a TCP socket.

We chose FUSE because it allowed us to implement our concepts in user space without having to manipulate kernel code. We chose FSFS because it is already a distributed file systems that allows users to access files stored on a remote server. The original FSFS code is embedded with its own security layer. We stripped-down the FSFS (bFSFS) code and used the bare version that allowed multiple users to access files stored on the remote server. The SGFS client and server code is not specifically tied to FSFS and can be layered on top of any file system. Our ideal goal is to integrate SGFS into more general purpose file systems, such as NFS. Hence, we decided to build the SGFS system in a modular fashion without tying it to any particular file system.

The SGFS system is in a preliminary implementation stage. Even though the authentication protocols are in place, there are several challenges that need to be addressed. Currently FSFS client can mount only one file server, which has to be specified while running the FSFS client. It allows multiple users on the same machine to access the mounted file server. However, since one FSFS client can access only one server, the SGFS users can access only one server through one SGFS client. To eliminate this problem, the next version of SGFS system will include two new features: *global naming* and *auto mounting*.

The current implementation does not include revocation servers and policy managers. One of the main challenges related to policy verification is choosing the appropriate policy manager. To the best of our knowledge all of the existing policy managers are tailored for public key certificates. Therefore, to exploit the existing policy managers and give us the flexibility of public key certificates we defined SKC that mimic public key certificates. We are investigating appropriate policy languages that are easy to

customize and flexible enough to be used in various access control models.

## 5. Conclusion

In this paper, we have presented the architecture and design of SGFS, a secure global file sharing system tailored for efficient data access. We have discussed the important requirements for a global file sharing system that have influenced our design. SGFS provides secure, efficient, and flexible global file sharing with minimal administrative interference. Users can delegate access permissions to remote users based on the local policies. Further, SGFS supports off-the-shelf policy engines that can be used by the system administrators to control user delegations. Due to its minimal cryptographic overhead on the storage servers, SGFS is suitable for emerging intelligent storage devices. We have developed a easy to use user-space prototype that features our authentication protocols and simple tools that assist users to create keys and delegate access rights to remote users. All symmetric key certificates are stored securely and can be moved from one machine to another to access in a seamless manner. SGFS offers great flexibility, low administrative overhead, and can be used in diverse environments.

## 6. Acknowledgements

This work was supported in part by the National Science Foundation (NSF) under Grant CNS-0448423 and by the Intelligent Storage Consortium at the Digital Technology Center (DTC), University of Minnesota. We would like to thank Mark Shaneck and Anjali Joshi for their helpful discussions and comments on earlier drafts.

## References

[1] Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF RFC 2459, January 1999.

[2] A. Acharya, M. Uysal, and J. Saltz. Active disks: programming model, algorithms and evaluation. In *ASPLOS*, 1998.

[3] G. Ateniese and S. Mangard. A new approach to DNS security (DNSSEC). In *ACM CCS*, 2001.

[4] E. Belani, A. Vahdat, T. Anderson, and M. Dahlin. The crisis wide area security architecture. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[5] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash function for message authentication. CRYPTO, 1996.

[6] M. Blaze. A cryptographic file system for UNIX. In *Proceedings of the ACM CCS*, 1993.

[7] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The keynote trust management system version 2.

[8] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Security and Privacy*, 1996.

[9] N. Cocchiaro. FSFS - the Fast Secure File System. `http://fsfs.sourceforge.net/`.

[10] D. Davis and R. Swick. Network security via private-key certificates. *ACM Operating System Review*, 1990.

[11] D. G. D.F. Ferraiolo and N. Lynch. An examination of federal and commercial access control policy needs. In NIST-NCSC National Computer Security Conference, 1993.

[12] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. IETF RFC 2693.

[13] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, Inc., 2003.

[14] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role based access control. *ACM TISSEC*, August 2001.

[15] Fuse: Filesystem in userspace. `http://fuse.sourceforge.net/`.

[16] G. R. Ganger and D. Nagle. Better security via smarter devices. In *HotOS*, pages 100–105, 2001.

[17] G. Gibson, D. Nagle, K. Amiri, F. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenk. File server scaling with network-attached secure disk. In *SIGMETRICS*, June 1997.

[18] L. Huston, R. Sukthankar, R. Wickremesinghe, M. Satyanarayanan, G. Ganger, E. Riedel, and A. Ailamaki. Diamond: A storage architecture for early discard in interactive search. In *USENIX FAST*, 2004.

[19] C. F. J. Hughes. Architecture of the secure file system. In *IEEE Symposium on Mass Storage Systems*, April 2001.

[20] M. Kaminsky, G. Savvides, D. Mazieŕes, and M. F. Kaashoek. Decentralized user authentication in a global file system. In *ACM SOSP*, October 2003.

[21] K. Keeton, D. A. Patterson, and J. M. Hellerstein. A case for intelligent disks (IDISKs). *ACM SIGMOD Rec.*, 27(3):42–52, 1998.

[22] J. Linn. The kerberos version 5 GSS-API mechanism. RFC 1964, June 1996.

[23] R. V. Meter, S. Hotz, and G. Finn. Derived virtual devices: A secure distributed file system mechanism. In *IEEE Mass Storage Systems and Technologies*, September 1996.

[24] E. Miller, D. Long, W. Freeman, and B. Reed. Strong security for distributed file systems. In *FAST*, January 2002.

[25] S. Miltchev, V. Prevelakis, S. Ioannidis, J. Ioannidis, A. Keromytis, and J. Smith. Secure and flexible global file sharing. In *Freenix*, 2003.

[26] B. C. Neumann and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[27] Information technology - SCSI Object-Based Storage Device Commands -2 (OSD-2). T10 Working Draft, October 2004. `http://www.t10.org/ftp/t10/drafts/osd2/osd2r00.pdf`.

[28] B. C. Reed, M. A. Smith, and D. Diklic. Security considerations when designing a distributed file system using object storage devices. In *SISW*, December 2002.

[29] E. Riedel, G. A. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *Proceedings of the 24th Conference on Very Large Data Bases*, 1998.

[30] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-Securing storage: Protecting data in compromised systems. In *OSDI*, October 2000