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

Vishal Kher, Eric Seppanen, Cory Leach, and Yongdae Kim
Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
{vkher,seppanen,leach,kyd}@cs.umn.edu

***Abstract*— This paper presents SGFS - a secure global file sharing system. SGFS is designed based on important design requirements for building a secure global file sharing system. These requirements 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 and the preliminary evaluation of the SGFS user-space prototype.**

I. 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 [14], [36], [47], [58] 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 [11], [29], [30], [37], [45], [54], [57] 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 [29], [54], remote execution [7], [59], search and indexing [33], and light-weight security operations [28], [66]. One can envision a network of heterogeneous storage servers 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.

In this paper we present SGFS - *A secure global file system* - that enables efficient cross-domain data sharing even in the presence of such heterogeneous environments. The goal of SGFS is to provide high performance data access and at the same time enable cross-domain collaboration. For better understanding of the design challenges of a global file sharing system, we list the requirements that in our opinion should be satisfied by a global file sharing system with heterogeneous storage servers[1].

- A global file sharing system should be deployable in diverse environments, for example, on the top of recent intelligent storage devices, such as those similar to [7], [28]–[30], [33], [59], [66]. It could be used for distributed high performance data access as well. Therefore, the cryptographic overhead on the storage servers should be minimal.
- In a global file sharing system, sharing of files should not be restricted only to users belonging to previously joined domains. It should be flexibly enough to allow users to collaborate without requiring the administrators of various domains to work together in order to setup a collaborative environment.
- If a local user Alice wants to collaborate with a remote user Bob, then Alice should be able to do so without asking her administrator to create an account for Bob. To reduce administrative interference, increase flexibility of sharing, and reduce management overhead, users should be able delegate access rights to other users. At the same time the system should be able to keep track (and maintain audit trails) of such delegations and the administrator should be able to revoke these delegations.
- Different organizations choose different policies. For example, policies used in Universities can be more relaxed than those used in enterprises. Therefore, a global file sharing system should support flexible policies and off-

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

the-shelf policy engines so that the administrators can set the policies as per their requirements.

- The system should not have a central point of failure. Clients should be able to directly access files from the storage servers without having to frequently contact any online entities (such as an authentication server). Such entities can become single point of failure and an attractive attack target. If this entity fails, the entire system can come to a halt.
- In order to share files with remote groups, users should not have to list remote group names on the local access control lists (ACLs). Listing remote groups on local ACLs increases the overhead of naming and locating remote groups, resolving remote group memberships, and mapping them to groups within the local domain. Besides, if the local and remote domains are not joined, then local users may not be able to list remote groups names on local ACLs.
- The authorization protocols should be flexible enough so that they can be used with various access control models, such as UNIX groups, role-based access control (RBAC) [25], or even file-groups [35][2].

Previous solutions such as Kerberos [39], [50] and those using public-key certificates [14], [36], [47], [58] only satisfy a *proper subset* of these requirements. 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. The machines used to mount file systems by remote users can under different administrative domains. Further, the remote group may be a group of independent researchers that may not be under a single administrative domain. Therefore, existing solutions such as Kerberos 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 [5]. In order to access files, users present a signed request and entire chain of certificates to the storage servers. 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 satisfy of all the requirements listed above. SGFS is designed to provide efficient global file sharing without any administrative interference. It is designed to work with existing policy managers [18], [19], [24] so that system administrators can set appropriate local policies. The system ensures that users behave according to these policies. SGFS offer 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.

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.

A. Organization

The rest of the paper is organized as follows. Section II presents a detail overview of previous work. Section III describes the architecture and design of SGFS. The SGFS authentication protocols are presented in sectionIV. Section V introduces our user-space prototype. Future work is outlined in section VII and section VIII concludes this paper.

II. Related Work

Distributed File Systems AFS [62], [63] and NFSv4 [51], [55], [64], the most commonly used networked file systems use Kerberos [39], [50] for authentication. As explained in the introduction section, in order to enable cross-domain collaboration between two domains, their system administrators should collaborate and setup shared secretes - a complicated operation which is rarely performed between two organizations. Even if the realms are joined, a user should be registered to in remote realms in order to access files from that realm. There is no way to avoid this problem as Kerberos does not

[2]Group of files with identical sharing attributes, e.g., files with same access permissions.

allow user-to-user delegation [3]. Finally, trust in Kerberos is binary, that is a if realms are joined a user can grant access to any remote user or if realms are not joined user cannot share files with remote users. SGFS support flexible policy that can be used to perform flexible and fine-grained trust management. Microsoft Windows 2003 servers [3] is based on Kerberos as well and has similar shortcomings.

In addition to Kerberos, NFSv4 supports SPKM3 [8] a public key based authentication mechanism. Every user (local and remote) has to be registered with the local domain, which requires system administrator intervention. Further, currently NFSv4 does not support user-to-user delegation of certificates.

SFS [36], [42] and DisCFS [47] were designed to avoid some of the drawbacks of AFS and NFS when used with Kerberos. SFS [42] introduced a notion of `self-certifying pathnames` - pathnames that effectively contain public key of the host file server which is used by the client to securely communicate with the server. The basic SFS protocol was extended further to provide cross domain authentication [36]. In this approach, a local user adds to the ACL the remote authentication server name and the remote group name with which she wants to collaborate. The local authentication server uses this information to periodically download the group memberships list from the remote authentication server. Thus, remote groups become local groups. Since local authentication server periodically downloads remote group information, a newly added member of the remote group cannot access files until the next download cycle. SFS does not provide client-to-server mutual authentication. In order to provide mutual authentication every client machine will be required to have a certificate from some CA trusted by the server. In addition, this will increase the computational overhead on the file server. SFS does not have any policy support and a local user can grant access to any user. SGFS uses a natural way of delegation; a user creates symmetric-key certificates for a remote user. AS does not need to download any remote group membership lists, which keeps the local group membership information clean and easy to manage. The SGFS authentication protocols are efficient, ensure client-server mutual authentication, and have flexible policy support.

CapaFS [58] and DisCFS [47] attempts to provide flexible and global file sharing by using capability certificate. In order to access a file, a user has to acquire a capability certificate from the administrator, which identifies the files and the operations that can be performed on that file. A user can delegate his access rights to a remote user by issuing a capability certificate for that user. The file server trusts the administrators certificate and verifies the chain of certificates before granting access to file listed in the certificate. In DisCFS files are identified by their i-node numbers; however, inode numbers are not suitable for global identification and can be reused (a fact that was admitted by the authors). Verifying a chain of certificates for every file can overload the file server and increase access latencies since the verification has to performed during data path. Further, since certificates are generated for every file revocation of certificates can result in bigger revocation lists. Finally, this model is not suitable for sharing files within a group of users, rather it is suitable for simple file sharing, such as for sharing a photo with a friend.

[3]Kerberos allows proxy-delegation, which is sometimes also referred as user delegation. However, proxy delegation is equivalent to giving away user password, and, thus, creating a proxy. This delegation model is different from user-to-user delegation where the delegator and the delegatee are two *independent* entities.

Network Attached Storage Devices Network-attached storage devices [11], [29], [30], [45], [54], [56], [57] enable data transfers directly between client and storage to provide high performance data access. The NASD [29], [30] project from CMU was one the first to widely demonstrate the advantages of network-attached storage devices. The NASD drives offload the block management and security operations from the file system. The drives were intelligent enough to a authorize access to the locally stored objects. In order to access an object, every client has to acquire a capability key for that object from a trusted third party (called file manager). A *capability key* identifies an object and operations that can be performed on that object. Using the capability key a user can authenticate and communicate securely with the drives. The NASD project was further extended in [11] and laid foundations to the emerging Object-based Storage (OSD) paradigm [54]. Files systems for OSD were presented in [12], [68].

The client obtains a capability for each object; therefore, the file manager has to be online and presents an attractive attack target and a central point of failure. If the file manager is down the entire system comes to a halt. Further, the capability is bound to the device and the object; therefore, if the object is replicated (or migrated) to a different device, the clients have to acquire one or more fresh capability keys [52]. SCARED [11] replaced used *identity keys* instead of capability keys. These keys can contain a user's identity and group information. SGFS uses SKCs that are designed to mimic X.509 attribute certificates in a symmetric key setting, which gives us flexibility to represent different kinds of privileges, constraints, and policy requirements. For example, our SKC can be used to represent roles and can be used in systems that use RBAC [25]. Further, symmetric key certificates offer us the flexibility of using off-the-shelf policy languages such as [18], [19], [24] that are designed for X.509 certificates in mind (to the best of our knowledge a powerful policy manager for symmetric key techniques is not yet designed).

Similar to our basic authentication protocol described in section IV-C, all of protocols described above are based on symmetric key techniques and do not perform any public key operations on the storage device. These protocols are focused on providing efficient mutual authentication protocol between the client and the server. However, they do not provide global file sharing and secure user-to-user delegation functionality.

Encrypting File Systems There have been numerous proposals for securing data at rest [17], [20], [26], [31], [32], [34], [40], [46], [49], [60]. These systems do not trust the storage

servers. Data is encrypted by the writers before it is sent to the storage and decrypted by readers after is received from storage. In addition to encryption, some of these systems also ensure integrity of data and meta-data [26], [31], [40], [46], [60]. Plutus [35], SiRiUS [60], SNAD [31], SFS [32], [34] and Cepheus [46] also allow sharing of encryption keys in a group of users. Naor et al. proposed an encrypting file system that avoids public-key operations for performance reasons [49].

Other Systems In general, public key certificates are used for authentication in several different areas [13], [14], [67], [69], [70]. The Taos operating system [70] provides a operating system component that manages principles and credentials. Authentication is based on credentials certificates that are assigned for each individual. WebFS, the file system component of WebOS [67] implements a network file system on the top of HTTP protocol. It runs a user-level web server that translates file system requests to HTTP requests. It uses the CRISIS [14] security architecture. Every CRISIS user has a X.509 certificate and uses this certificate to authenticate with remote servers. The rest of the systems cited above work on the same model.

Shibboleth [65] is a joint project of Internet2 and IBM. It is investigating architectures to support inter-institutional sharing and access to web services. The Infocard [1] is an effort from Microsoft to unify databases of identity providers (such as government and e-commerce sites) to form a federated identity meta-system. The goal is to minimize the users' overhead of identity management. It requires sites to perform changes in order to conform to the Infocard requirements and have to adopt to SAML [4]. Similar to Kerberos different domains have to co-operate in order to set-up a federated identity management system. A comprehensive survey of storage security literature can be found in [38], [60].

III. System Design

A. 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 [5], 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:

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

Where, prf is a pseudo-random function (HMAC [15] 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 [57].

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 [44] was used in [10], [22]. In these approaches, a SKC 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} 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.

B. 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 as well as traditional file servers. 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) [25], users' roles are usually associated with their job in the organization, which do not change frequently [6], [23]. Similarly in UNIX environments, a particular user's group membership does not change daily. The lifetime of access keys should be decided based on the policies. In SGFS, access keys are stored *securely* on the client side.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. The file server only trusts the AS and verifies whether the requester has acquired the SKC from AS. This allows AS to maintain audit logs and verify policies, keeps the file server simple, and eliminates any need of verifying certificate chains at the file server.

Flexible policy support Organizations use different policies in different settings. Many of the existing policy languages, such as PolicyManager [19], KeyNote [18], and SPKI [24], 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.

C. System Architecture

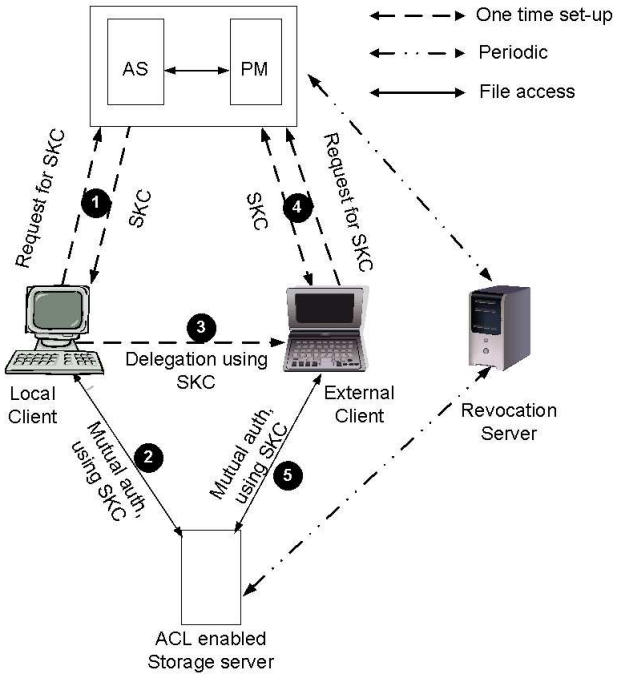


Fig. 1. The SGFS Architecture.

Figure 1 depicts the SGFS architecture. It consists of five entities: authentication server (AS), policy manager (PM), storage servers, clients[4] (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. 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

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

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 [17], [31], [32], [34], [40], [46], [60].

D. 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 (details in section IV) 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 to 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.

IV. Authentication Protocol Details

Notations used for the protocol description are as follows:

S	Storage Server
AS	Authentication Server
$SKC^{C}_{T,V}$	Symmetric key certificate generated by T for verifier V given to user C
K	Secret key shared between AS and S
MAC_{k_i}	Keyed-MAC function using key k_i
r_C	unique random number generated by C
H	Collision resistant hash function

A. Acquiring SKC for Local Users

A local user Alice acquires her SKC from the AS on the first access to a storage server S. Initial SKCs can also be transferred to Alice when her account is created, e.g., using email. Let A denote Alice and B denote Bob. The protocol messages are shown below.

$$A \rightarrow AS : SKC\ request \quad (1)$$

$$AS \rightarrow A : SKC^{A}_{AS,S} = \{P^{A}_{AS}, MAC_K(P^{A}_{AS})\} \quad (2)$$

Upon receipt of message (1), AS authenticates Alice and verifies that Alice is a legitimate user of the system. The authentication protocol can be any of the standard authentication protocols. AS then queries the user database and acquires Alice's group membership information. AS also acquires constraints, and other policy information from PM and asks PM to perform initial policy verification, if any. AS then fills the necessary fields (described in section III-A) of P^{A}_{AS}, generates the SKC as shown in (2) and securely transfers it to Alice. Since SKCs are long-lived no further communication with AS

is required. Alice can directly access files from S without contacting AS.

B. Acquiring SKC for Remote Users

Now let us see how an external user Bob acquires a SKC to access files stored on S.

1) Naive Approach: The protocol messages are shown below:

$$A \rightarrow B : SKC^B_{A,S}, P^A_{AS} \quad (3)$$

$$B \rightarrow S : M, P^B_A, P^A_{AS}, MAC_{K^B_A}(M) \quad (4)$$

Where $M = \{r_B, SKC\ request\}$, and $SKC^B_{A,S} = \{P^B_A, K^B_A = MAC_{K^A_{AS}}(P^B_A)\}$. In order to share files with Bob, Alice sends $SKC^B_{A,S}$ to Bob in message (3). A adds `genomics` to the group list in P^B_A. SKC is signed by Alice using K^A_{AS} and anyone who can generate this key can verify the authenticity of $SKC^B_{A,S}$.

Upon receipt of (4), S can generate $K^A_{AS} = MAC_K(P^A_{AS})$. S can then generate $K^B_A = MAC_{K^A_{AS}}(P^B_A)$. Thus, S and B now share K^B_A and they can authenticate each other using this key. After successful authentication, S extracts the group membership information from P^B_A and performs access control accordingly.

Discussion This approach is flexible since Alice can delegate access rights to any user. Further, B can directly access file stored on S, S can efficiently authenticate B, and S can make access control decisions by itself.

However, the first drawback of this scheme is that since A and B both know K^B_A, A can access files as B. Therefore, S cannot distinguish between A and B.

The second drawback of this scheme is that S cannot perform any policy verification since S cannot uniquely identity B or B's parent organization. For example, if policies allow A to delegate access rights to users from a certain company S cannot determine whether B belongs to that company. Even if S was able to determine B's parent organization, in order to verify policies S should have all the policy information available to it. This will either require replication of PM on all the storage servers or constant online communication with the PM. Both the approaches are inefficient.

The third drawback is that AS is completely unaware of any delegations by A. Therefore, all storage servers will have to maintain audit trails and AS will have to constantly gather these audit trails to ensure that policies are met and gather information for revocation purposes.

Summary Therefore, we need - 1) a way to uniquely identify B, 2) a way to uniquely identify B's parent organization 3) a way to efficiently verify policies, and 4) a way to maintain audit trails at AS.

2) Final Approach: Alice sends the following information to Bob:

$$A \rightarrow B : SKC^B_{A,AS}, P^A_{AS} \quad (5)$$

Where, $SKC^B_{A,AS} = \{P^B_A, K^B_A = MAC_{K^A_{AS}}(P^B_A)\}$. Alice can define her own policies for Bob by setting appropriate fields in the P^B_A. For example, Alice can set the expiry period of $SKC^B_{A,AS}$; she can also specify whether Bob can further delegate $SKC^B_{A,AS}$ to his group members.

Using $SKC^B_{A,AS}$ Bob acquires a SKC from AS as follows:

$$B \rightarrow AS : M, P^B_A, P^A_{AS}, MAC^*_{K^B_A}(M) \quad (6)$$

$$AS \rightarrow B : SKC^B_{AS,S} = \{P^B_{AS}, K^B_{AS}\} \quad (7)$$

Where, $M = \{r_B, SKC\ req\}$ and $K^B_{AS} = MAC_K(P^B_{AS})$. Upon receipt of message (6), the authentication server generates $K^A_{AS} = MAC_K(P^A_{AS})$ and $K^B_A = MAC_{K^A_{AS}}(P^B_A)$. It then generates $MAC_{K^B_A}(M)$ and compares it with MAC^* sent in (6). If they are same, AS concludes that A has delegated some of her rights to B. Then, AS checks whether A is allowed to delegate the rights included in $SKC^B_{A,AS}$.

If yes, AS checks with the PM whether the delegation is conforming to the policies. P^B_A which is included in Bob's symmetric key certificate generated by Alice ($SKC^B_{A,AS}$), contains the serial number of Bob's public key certificate that can be issued by his organization. This is included because Alice's local policies may dictate that Alice can delegate only to users from certain organizations. Using Bob's certificate Alice's PM can easily perform this check by verifying whether Bob's certificate is from a trusted CA. This feature can be more useful in enterprise where users are partially trusted and the PM wants to prevent a malicious insider from delegating access rights to a user from a competing organization. Universities, for example, may trust their local faculty members to delegate permission at will. In this case, the system administrator can to configure the PM accordingly and the PM will not verify Bob's certificate. Thus, by setting suitable policies, one can achieve a balance between user delegation and centralized control.

If all checks succeed, AS creates a new SKC for Bob and sends it to Bob in (7). Bob now has rights to access files belonging to groups listed in P^B_{AS}. $SKC^B_{AS,S}$ allows Bob to access files belonging to the `genomics` group stored on S. If Bob needs additional SKCs to access files stored (belonging to the `genomics`) on other storage servers, then Bob can repeat step (6) above indicating the name of the storage server in the request.

C. Client - Server Mutual Authentication

The SGFS storage servers only trust the AS. Any client no matter remote or local should get a SKC from the AS to authenticate with the storage server. We have already discussed above how local and external users can acquire a SKC from the AS. Below we discuss how a user A with a $SKC^A_{AS,S}$ can authenticate with S. The protocol messages are as follows.

$$A \rightarrow S : r_A, P^A_{AS} \quad (8)$$

$$S \rightarrow A : r_S, MAC^*_{K^A_{AS}}(r_A, r_S, A) \quad (9)$$

$$A \rightarrow S : R, MAC_{K^A_{AS}}(r_S, r_A, R, S) \quad (10)$$

Upon receipt of (8), S checks if A's SKC is expired by checking the expiry field in P^A_{AS}. If not expired, S generates

K^A_{AS} = $MAC_K(P^A_{AS})$ and sends a response to A as shown in (9). Upon receipt of (9) A generates $MAC_{K^A_{AS}}(r_A, r_S, A)$ and verifies that this MAC is same as the MAC^* sent by S in (9). If yes, the client concludes that S is authentic (since only AS and S know K^A_{AS}). A uses the actual request R (e.g, read,write) and r_S acquired in (9) to build message (10). S verifies the authenticity of message (10) by generating $MAC_{K^A_{AS}}(r_S, r_A, R, S)$ and by checking it against the MAC sent by A. At the end of step (10) S concludes that A has authentic SKC created by AS (and thus A is an authentic client). S then checks if the request meets all the policy requirements listed in P^A_{AS}. Finally, S extracts the group membership list from P^A_{AS} and checks if the user can perform the requested operation on the requested object by checking that object's ACL. If all checks succeed, then S serves the request and sends back the appropriate response.

A and S can optionally setup a session keys one to encrypt network traffic and another to ensure integrity (using HMAC) as follows:

$$k1 = MAC_{K^A_{AS}}(\text{“}ENC'', A, r_B)$$
$$k2 = MAC_{K^A_{AS}}(\text{“}HMAC'', B, r_A)$$

The client and the server can use these keys to ensure confidentiality and integrity of messages and the data transferred in subsequent communication. Finally, before verifying message (10) the server should also check if $SKC^A_{AS,S}$ is revoked by searching its local revocation list.

D. Discussion

Summary The client-server mutual authentication protocol (section IV-C) 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.

Extensions For mutual authentication, we purposely used random numbers instead of timestamps, since in a global file sharing system it is not realistic to assume that all client machines will be time synchronized with all of the storage servers. Due to this requirement, the mutual authentication protocol as described in section IV-C requires two extra rounds of communication as compared to a mutual authentication protocol between time synchronized entities [44]. In environments where time synchronization is feasible, these communication rounds can be avoided by using timestamps instead of random numbers and making simple changes to the protocol [30]. Using time stamps will also avoid the requirement of caching of two random numbers on the storage servers during the authentication process.

If storage servers have the ability to cache some information per session, then session keys can be cached for a longer duration to avoid performing mutual authentication for each client request. In addition standard techniques, such as maintain a single counter or using server encrypted cookies can be used to reduce the cache state on the server.

It can happen that Bob can receive multiple SKCs for one server from Alice. In this case we can extend the client-server mutual authentication by including multiple $MACs$ in (8). Optionally, the SGFS client can automatically “redeem” multiple SKCs for one server by sending them to AS and receive a singe combined SKC in exchange. In this case, Bob will have to maintain only one symmetric certificate per storage server.

Revocation Since the SGFS SKCs are long-lived we need to maintain revocation lists at the revocation server. Each SKC has an associated expiry information. Storage servers check if the SKC is expired before granting access to the users.

In addition to expiry, AS can selectively revoke a user's SKC by publishing the SKC identifier to the revocation servers. The revocation servers can periodically send the appropriate revocation lists to the appropriate servers. The SKC can contain the identity of the verifier (storage servers). The revocation server only needs to inform that storage server.

The storage servers keep a copy of revocation lists locally so that they do not have to contact the revocation server on every request. The revocation lists on the storage servers can be organized according to groups to enhance searching operations. For example, Aguilera et al. maintained a bitmap of identities of revoked certificates in the RAM of a controller of a block based disk [9]. Each user certificate was associated with a unique ID and a group ID. The disk maintained a list of group IDs and a bitmap indicating revocation state of the certificate IDs. Certificate IDs were recycled by removing group IDs from the list. The revocation table contained a tuple of 6-bit index, a 64-bit group ID value, and 8K-bit bitmap for certificate IDs. On average, each certificate took only 1.01 bits in a table of size 64KB. We can use a similar technique to store revocation lists on the storage servers.

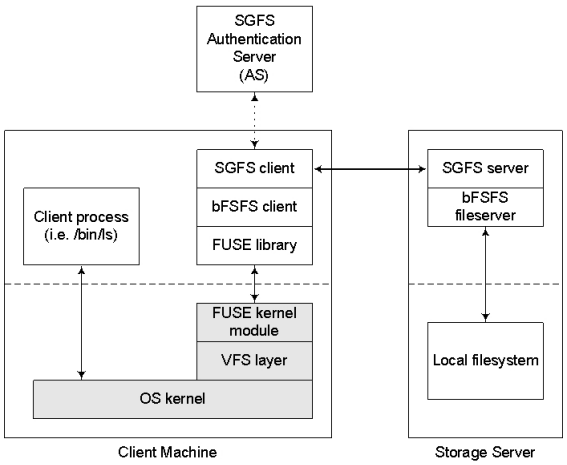


Fig. 2. The SGFS system components

V. Prototype Implementation

Figure 2 represents the SGFS system. The SGFS client runs in user space and is layered on the top of FSFS [21], a user space file system designed to be mounted through an interface provided by FUSE [27]. 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 complex 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.

Authentication Server AS is portable to any Unix-like operating system. All cryptographic operations are performed using the OpenSSL 0.9.8 [53] library. The AS uses MySQL database v4.1 [48] to store user information of local users. There is one entry in the database for every user. Each record contains the username, user ID, user group membership list, user's public key, and constraints. Currently only UNIX style user groups are supported. This will be extended to support other access control mechanisms, such as roles and file groups. The AS shares a unique symmetric key with each storage server. Currently, this is done manually and in the future secure key distribution algorithms can be used.

SGFS Client Every SGFS user is associated with a public-private key pair. These keys are used to securely store SKCs (encrypted using the user's public key), to securely communicate with the AS, and to securely communicate with other users. SGFS includes tools that assist users to create these key-pairs and get them signed from local certification authority. By default the public-key certificates, encrypted private keys, and encrypted SKCs are stored in user's home directory at `/home/$username/.sgfs/keys`

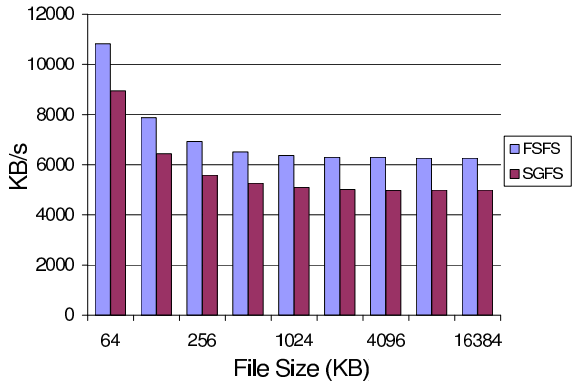
All of the authentication protocols described in section IV are implemented and fully functional. Below we demonstrate how the client works by an example. Lets assume that the SGFS server `sgfs.umn.edu` is mounted at `/sgfs`. Lets say that the local user Alice attempts to access `/sgfs/foo`. The SGFS client uses Alice's private key to automatically retrieves the SKC for the destination server, decrypts it, and initiates the mutual authentication protocol (section IV-C). The server behaves according to the protocol and grants access based on information included in the public part of Alice's SKC (P^A_{AS}). If the server is able to cache session keys for a longer time, the SGFS client maintains a SGFS context for Alice after the mutual authentication phase. This context contains the shared session keys that are used to secure subsequent communications with the storage server. All file tansfers between the client and the server are encryted using Blowfish in OFB mode and then HMACed to guarantee integrity. If Alice does not have a SKC, then the SGFS client uses Alice's private key to authenticate with the AS and acquires a SKC. It stores the encrypted SKC in `/home/alice/.sgfs/keys/skc_store` and then initiates the mutual authentication protocol with the AS. The AS is contacted only when the SKC is expired.

Since all the keys are stored securely, Alice can easily transfer her files from his office computer to her home computer and remote files just as she would access from her office computer. Thus, a user can move from one machine to another and access file seamlessly.

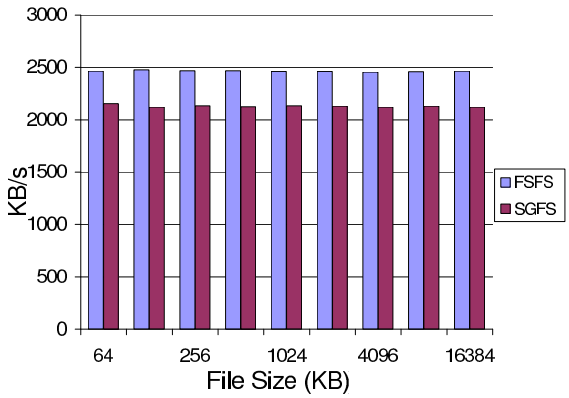
We have also develped a command-line tool to assist Alice to delegate her access rights to a remote user Bob and generate a SKC for Bob (see section IV-B.2 for details). The SKC is encrypted using Bob's public key. Alice can send this SKC to Bob. After receiving the delegated SKC from Alice, the only operation that Bob has to perform is to add this SKC to his `/home/bob/.sgfs/keys/skc_delegation`. When Bob tries to access `/sgfs/foo`, Bob's SGFS client automatically, acquires the delegated SKC from `/home/bob/.sgfs/keys/skc_delegation`, authenticates with Alice's AS and (as described in IV-B), securely stores the SKC returned by AS in `/home/bob/.sgfs/keys/skc_store`. Once the SKC is available, Bob's SGFS client can use it to mutually authenticate with `sgfs.umn.edu` and access files securely.

VI. Preliminary Evaluation

In order to measure the performance of SGFS, two Pentium 4, 2.5GHz computers with 1GB memory running Suse Linux 9.3 were connected via 100Mbps Ethernet. One computer was configured as a client and one as a server. First, bFSFS was run on the client and server and performance analysis was conducted. Then, bFSFS with SGFS was run on the client and server and the performance analysis was repeated.



a. SGFS read performance.



b. SGFS write performance

Fig. 3. Iozone Throughput

Iozone 3.257 [2] was used to measure the read/write performance of the filesystem. Figure 3 illustrates the performance of read and write using 16KB blocks, reading and writing file sizes from 64KB to 1024KB. During this experiment shared session keys were established between the client and the server and all data was encrypted using Blowfish in OFB mode [44] and HMACed using SHA-256.

The performance difference between bFSFS and SGFS is due to several factors: first, encryption and decryption is not done in a pipelined fashion. That is, a large block of data is encrypted completely before attempting to send over the network. As a result, the the receiving end of the SGFS system remains idle for that period of time. Pipelining the encryption and networking stages should reduce this penalty considerably. In addition, FSFS read throughput is much higher than write throughput, which results in more overhead for reads, as encryption becomes a larger fraction of total request service time.

Further, we are using encrypt-then-mac authenticated encryption by following the recommendation by Bellare and Namprepre [16]. However, this is inefficient and difficult to pipeline since we first need to encrypt and then compute HMAC for the whole encrypted block. In the future implementation, we will use new modes of operation such as GCM [43] and OCB [61], which provides authenticated encryption without such serialization.

The gap between the read and the write performance is due to underlying filesystem and OS characteristics, as well as peculiarities of the FSFS protocol that require more network traffic for writes. FSFS splits every request into four requests: a stat request for the destination directory, a stat request for the file, a open request for the file, and finally, followed by multiple write requests. These problems are straight-forward to amend with little modifications to the current code.

VII. Future Implementation Work

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.

Global Naming and Automounting 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*.

We plan to modify FSFS client so that it will be mounted on `/sgfs`. All SGFS pathnames will be of the following format:`/sgfs/file.server.name/actual.pathname`. For example, in order to mount filesystem `/genomics/experiments` stored on `sgfs.umn.edu`, Alice will run `emacs~/sgfs/sgfs.umn.edu/genomics/experiments`. FUSE will transfer control to the FSFS client. The FSFS client will hand-over the file server name `sgfs.umn.edu` and the request to read file `foo` to the SGFS client. The SGFS client will acquire Alice's SKC for `sgfs.umn.edu`, mutually authenticate with `sgfs.umn.edu`, and automatically mount `/genomics/experiments`. This will greatly improve the flexiblity of the SGFS client. A user will be able to access any file server in the world as long as she has the appropriate SKC to authenticate with that server. We draw this inspiration from the SFS file system [42] that itself has the file server information in the pathname and uses a automounter to mount remote file systems on demand. Several challenges in building such a system were identified in [41]

Revocation and Policy Verification The current implementation does not include revocation servers and policy managers. We have already described the high-level design issues related to revocation servers in section IV-D. One of the main challenge realted 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. We also have to evaluate the policy engines associated with these languages to choose the one that provides better perfomance.

VIII. 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 outlined 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. Experimental results confirm that with little modifications, overhead of SGFS protocols will be negligible.

IX. 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.

References

[1] Infocard. `http://blogs.msdn.com/andyhar/`.

[2] IOzone Filesystem Benchmark. http://www.iozone.org/.

[3] Kerberos authentication in windows server 2003. `http://www.microsoft.com/windowsserver2003/technologies/security/kerber%os/default.mspx`.

[4] Security assertion markup language. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`.

[5] Internet x.509 public key infrastructure certificate and crl profile. IETF RFC 2459, January 1999.

[6] *Role-Based Access Control*. Artech House, Inc., 2003.

[7] A. Acharya, M. Uysal, and J. Saltz. Active disks: programming model, algorithms and evaluation. In *ASPLOS-VIII: Proceedings of the eighth international conference on Architectural support for programming languages and operating systems*, pages 81–91, 1998.

[8] W. Adamson and O. Kornievskaia. Low infrastructure mutual authentication using spkm-3. Internet-Draft, October 2005.

[9] M. K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D. G. Andersen, M. Burrows, T. Mann, and C. Thekkath. Block-Level Security for Network-Attached Disks. In *Proc. 2nd USENIX Conference on File and Storage Technologies*, March 2003.

[10] G. Ateniese and S. Mangard. A new approach to dns security (dnssec). In *Proceedings of the 8th ACM conference on Computer and Communications Security*, New York, NY, USA, 2001.

[11] A. Azagury, R. Canetti, M. Factor, S. Halevi, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, and J. Satran. A two layered approach for securing an object store network. In *SISW*, December 2002.

[12] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory, , and L. Yerushalmi. Towards an object store. In *In the 20th IEEE Symposium on Mass Storage Systems*, 2003.

[13] J. Bacon, K. Moody, and W. Yao. A model of oasis role-based access control and its support for active security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, 2002.

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

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

[16] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, 2000.

[17] M. Blaze. A cryptographic file system for UNIX. In *Proceedings of the 1st ACM Conference on Communications and Computing Security*, Fairfax, VA, 1993. ACM Press.

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

[19] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*, Washington, DC, USA, 1996.

[20] E. Z. Charles P. Wright, Michael C. Martino. Ncryptfs: A secure and convenient cryptographic file system. In *USENIX Annual Technical Conference*, June 2003.

[21] N. Cocchiaro. Fsfs - the fast secure file system. `http://fsfs.sourceforge.net/`.

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

[23] 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.

[24] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Spki certificate theory. IETF RFC (Proposed Standard) 2693, September 1999.

[25] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for rolebased access control. *ACM Transactions on Information and System*, August 2001.

[26] K. Fu. Group sharing and random access in cryptographic storage file system. Master's thesis, MIT, 1999 June.

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

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

[29] 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.

[30] H. Gobioff, G. Gibson, and D. Tygar. Security for network attached storage devices. Technical report, Carnegie Mellon University, Pittsburgh, PA, October 1997.

[31] E. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing Remote Untrusted Storage. In *Proceedings of the Tenth Network and Distributed Systems Security (NDSS) Symposium*, pages 131–145, February 2003.

[32] J. Hughes and D. Corcoran. A universal access, smart-card-based, secure filesystem. In *Atlanta Linux Showcase*, October 1999.

[33] 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 *Proceedings of USENIX File and Storage Technologies (FAST)*, April 2004.

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

[35] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus — scalable secure file sharing on untrusted storage. In *USENIX File and Storage Technologies (FAST)*, San Francisco, CA, March 2003.

[36] M. Kaminsky, G. Savvides, D. Mazieŕes, and M. F. Kaashoek. Decentralized user authentication in a global file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

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

[38] V. Kher and Y. Kim. Securing distributed storage: challenges, techniques, and systems. In *Proceedings of the 2005 ACM workshop on Storage security and survivability (StorageSS)*, pages 9–25, 2005.

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

[40] E. Mauriello. TCFS: Transparent cryptographic filesystem. *Linux Journal*, 40, August 1997.

[41] D. Mazieres. A toolkit for user-level file systems. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, 2001.

[42] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Kiawah Island, South Carolina, December 1999.

[43] D. A. McGrew and J. Viega. The security and performance of the galois/counter mode of operation (full version). Cryptology ePrint Archive, Report 2004/193, 2004.

[44] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.

[45] R. V. Meter, S. Hotz, and G. Finn. Derived virtual devices: A secure distributed file system mechanism. In *Proceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, College Park, MD, September 1996.

[46] E. Miller, D. Long, W. Freeman, and B. Reed. Strong security for distributed file systems. In *Proceedings of the Conference on File and Storage Technologies (FAST 2002)*, pages 1–13, January 2002.

[47] S. Miltchev, V. Prevelakis, S. Ioannidis, J. Ioannidis, A. Keromytis, and J. Smith. Secure and flexible global file sharing. In *Proceedings of the USENIX Technical Annual Conference, Freenix Track*, 2003.

[48] Mysql database. `http://dev.mysql.com/`.

[49] D. Naor, A. Shenhav, and A. Wool. Toward securing untrusted storage without public-key operations. In *StorageSS '05: Proceedings of the 2005 ACM workshop on Storage security and survivability*, 2005.

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

[51] NFS Version 4. `http://nfsv4.org/`.

[52] C. Olson and E. L. Miller. Secure capabilities for a petabyte-scale object-based distributed file system. In *StorageSS '05: Proceedings of the 2005 ACM workshop on Storage security and survivability*, 2005.

[53] OpenSSL Project team. Openssl, May 2001. http://www.openssl.org/.

[54] 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`.

[55] B. Pawlowski, S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow. The NFS version 4 protocol. SANE 2000, May 2000.

[56] B. Reed, E. Chron, R. Burns, and D. D. E. Long. Authenticating network attached storage. *IEEE Micro*, 20(1):49–57, January 2000.

[57] 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.

[58] J. T. Regan and C. D. Jensen. Capability file names: Separating authorisation from user management in an internet file system. In *USENIX Security Symposium*, 2001.

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

[60] E. Riedel, M. Kallahalla, and R. Swaminathan. A framework for evaluating storage system security. In *Proceedings of the Conference on File and Storage Technology*, January 2002.

[61] P. Rogaway, M. Bellare, and J. Black. Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3), 2003.

[62] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[63] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5), May 1990.

[64] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. NFS version 4 protocol. RFC 3530, April 2003.

[65] The shibboleth project. `http://shibboleth.internet2.edu/`.

[66] 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.

[67] A. Vahdat, P. Eastham, C. Yoshokawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin. Webos: Operating system services for wide area applications. In *The Seventh IEEE Symposium on High Performance Distributed Computing*, 1997.

[68] F. Wang, S. Brandt, E. Miller, and D. Long. OBFS: A file system for object-based storage devices. In *21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, 2004.

[69] B. S. White, M. Walker, M. Humphrey, and A. S. Grimshaw. Legionfs: a secure and scalable file system supporting cross-domain high-performance applications. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, 2001.

[70] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the taos operating system. *ACM Transactions on Computer Systems (TOCS)*, 1994.