AUTHENTICATING NETWORK-ATTACHED STORAGE

WE PRESENT AN ARCHITECTURE FOR NETWORK-AUTHENTICATED DISKS THAT IMPLEMENTS DISTRIBUTED FILE SYSTEMS WITHOUT FILE SERVERS OR ENCRYPTION. OUR SYSTEM PROVIDES NETWORK CLIENTS WITH DIRECT NETWORK ACCESS TO REMOTE STORAGE.

Benjamin C. Reed
Edward G. Chron
Randal C. Burns
IBM Almaden Research Center

Darrell D.E. Long
University of California, Santa Cruz

The need to access anything from anywhere has increased the role of distributed file servers in computing. Distributed file systems provide local file system semantics for access to remote storage. This allows network clients to incorporate the remote storage into their local file system. File semantics are well understood by users and applications, making distributed file servers a convenient tool in developing distributed applications.

As the role played by distributed file systems expands, problems with their design become increasingly evident. Faster clients, high-bandwidth connections, and larger drive capacities increase the demand on file servers. Although it would seem that the I/O capacity of the system storage devices would limit network file server performance, in actuality, file servers frequently are CPU bound. Riedel and Gibson discovered that, even with low overall CPU utilization, burst loads were sufficiently intense to overuse the server.[1]

In addition to the performance problems of distributed network file systems, security also presents a problem. Applications that rely on distributed file systems should not be compromised by security weaknesses of the file systems on which they are built. Local file systems have a single kernel that restricts access to file data, but because a distributed file system involves multiple servers and clients, it cannot rely on a single kernel to restrict access. The security risk is even greater since the network that connects servers and clients may also pose a threat.

The authenticated network-attached disks we present address these problems by providing an architecture based on one-way hash functions that make available mutual authentication of the network disks and the clients. This architecture obviates the need for more performance-intensive authentication methods such as public-key encryption and Kerberos,[2] but does not preclude their use. The authentication protocol used by the network storage is very simple and flexible, and allows keys to be created and managed using existing authentication systems.

Distributed file systems

In classical distributed file systems, all accesses to the data store are through the file server. The file server verifies the data's accessibility before carrying out the client request. The data store is usually locally attached to the file server. Since the data storage is only attached to the file server, it can simply carry out the requests of the file server without having to authenticate or check access permissions.

File systems such as Swift[3] and Zebra[4] as well

0272-1732/00/$10.00 © 2000 IEEE

SCARED'S end result is a network-storage device we use to build a serverless file system that can run in an untrusted environment.

as the Network-Attached Storage (NASD)[5] prototypes, separate the file server and the data stores. The clients get file system metadata from the file server, but get file data directly from the data stores. One complication of separating data and metadata is determining client access privileges. Both the file servers and the data stores must verify that the client has access permission to the data.

In file systems that separate the authentication and file serving components, such as Andrew File System (AFS),[6] an authentication server is generally present. The client authenticates the user to the authentication server by a password, ticket, or other method. The authentication server gives the client new tickets that are presented to the file server. These tickets may grant access to specific files on the file server or may simply authenticate the identity of the user.

Serverless file systems

The serverless file system,[7] developed in conjunction with the Network of Workstations (NOW) project at the University of California, Berkeley, consists of a network of trusted workstations that provide the functionality normally provided by a network server. The result is a file system that has no central repository of files. Instead, the file system data and metadata is spread among the network of trusted workstations.

By spreading the file system data and metadata across multiple machines, the aggregated resources are much greater than that found on a normal file server. These resources include cache size, network bandwidth, and processing power. Striping and logging is also used to boost performance.

Network-attached storage devices

The NASD project at Carnegie Mellon University keeps the file server, but allows clients direct access to file data stored on network-attached disks to boost performance. Before network clients access files, their requests must be authenticated and permissions checked. The file servers generally perform this checking. However, if clients are allowed to directly access the disks, the disks must verify the authority of the client's access to the data.

In NASD, when a client wishes to access data on the disk, the client obtains a capability from the file server. The client then presents the capability with a request to the network disk, which verifies that the capability allows the requested action before carrying it out. NASD requires a file server to serve the metadata and generate capabilities for file system clients.

While the serverless file system yields dramatic performance improvements when compared to a more centralized file system, these enhancements can reduce security. The serverless file system is designed to run in a trusted environment, where the client and manager kernel protect the file system from malicious access. This kind of environment is found in NOW and in networks where all machines are administered and trusted equally.

SCARED

Our project extends the network-attached storage model of NASD to enable directory data as well as file data storage on the network. We also allow the clients to share keys to access the network storage. The end result is a network-storage device we use to build a serverless file system that can run in an untrusted environment.

The SeCure Authentication for Remotely Encrypted Devices (SCARED) protocols were developed at IBM Research for use in network-attached storage. One of the main design requirements was minimizing the management overhead of the storage devices. File servers require a substantial investment in management resources. Pulling the storage out of the servers and attaching them to the network increases the number of managed network devices. If the administrative requirement increases proportionally to the number of devices, the system would quickly become unmanageable. The management of network-attached storage is further complicated due to the lack of a management console. For these reasons we push the administrative overhead out to the clients, where the storage device administration can occur along with the normal configuration of the client to use the network storage.

Storage devices are deployed in environments with a wide variety of existing authentication systems such as Kerberos and public key-based systems, so we did not make assumptions about the deployment environments of the devices. The authentication operations performed at the storage device are simple and allow the device to remain oblivious to the existing security environment. Since the keys used to interact with the storage devices are generated and exchanged by users and administrators without communicating with the storage, the key exchanges can take place within the existing systems.

SCARED addresses authentication

The confidentiality requirements of storage devices are best solved by encrypting and decrypting at the clients. Encrypting data is expensive in terms of processing overhead and introduces latency. Encryption and decryption at the client allows the data to be encrypted over the network and on the storage media itself without any overhead at the server. Of course, SCARED does not preclude link-level encryption.

In the SCARED environment there are three roles: the client, the administrator, and the storage device. The storage device shares a key with the administrator. The administrator uses this key to generate other keys for clients. Clients use the derived keys to access the storage devices.

An important feature of the SCARED protocol is that the administrator does not need to be online with the disk when granting access to clients. Not only does this relax the network topology requirements, but it also allows the administrator to give access keys to clients using offline methods such as e-mail.

A client uses the keys received from the administrator to generate message authentication codes (MACs)[8] that are included in all message exchanges between the client and storage devices. A MAC function takes a string and a secret key, and outputs a fixed-length string. The MAC has some cryptographic properties that allow either party to verify if the message sender was in possession of a specific key and whether the message was changed in transit. Once the storage device checks the MAC, it grants access to the client based on the key used to generate the MAC.

Key distribution without key exchange

Using MACs to authenticate messages between the clients and storage devices requires that they share a key. SCARED uses a key distribution scheme that does not require any key exchange or encryption.

Encryption and decryption at the client allows the data to be encrypted over the network and on the storage media itself without any overhead at the server.

We wanted to keep the device from having to perform key management or from involvement with distributing keys to clients, so the storage device itself contains only one key: the disk key. The storage administrator and the storage device share this key. All other keys are based on this key, which is used to bootstrap the security of the disk. We assume that the administrator receives the disk key with the storage device. This may be in the form of a smart card, disk, or paper that comes with the device. Another method, which is used by NASD, is to allow the administrator to generate and send the disk key to the disk at initial network connection.

SCARED uses the disk key to generate the other keys needed by clients to generate and verify the MACs used when communicating with the storage device. The key derivation is based on a keyed, one-way, hash function, $H(D,K)$ that takes a public value D and a secret key K as input, and outputs a new secret.

For keys to be meaningful to the storage device, they need to contain associated data that conveys identity and capability. The hash function binds the data associated with a key to the key itself. In general, a key K' is the result of $H(D',K)$, where D' is some public data associated with K', and K is the key that is used to derive K'. For example, the administrator can use the disk key to derive a key and send it to the client over a secure channel. When the client sends a message to the disk, he or she will include the public data associated with the key in the message, then MAC the message using the key obtained from the administrator. The device can regenerate the client's key using the public data and the disk key to verify the MAC.

The public key data lets the storage device not only derive the key that the client is using, but also check the client's access status to the

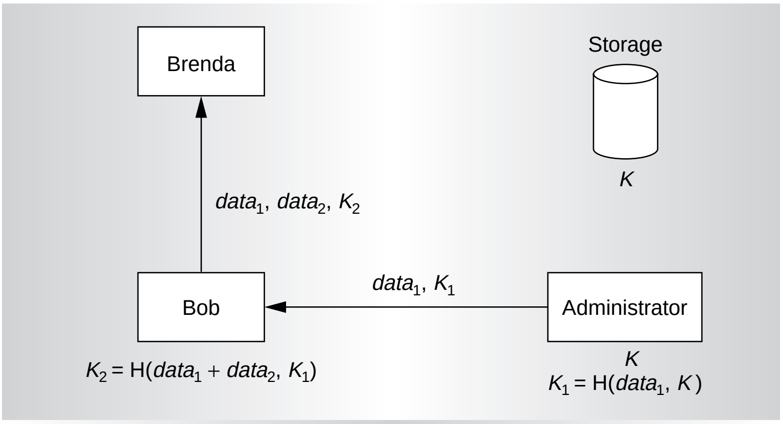

Figure 1. The administrator shares a key, *K*, with the storage device, which generates keys for the clients. In this example the messages must be exchanged over secure channels.

device. Because the key is derived using a one-way hash and the key data, when the client uses a key, the client must also send the key data associated with the key. The binding between the key and key data allows the administrator to put information in the key data that the storage device uses to grant access to the client. By including an expiration date as part of the key data, the administrator can limit the key's lifetime.

Key types

The authentication needs of a client and storage device differ, so the keys they use also differ. The client needs to verify that responses received from a storage device actually came from a given device. The device needs to verify that the client has the authority to make a request. When a client uses a key to send a request to the storage device, we refer to the key as an access key. A key used to verify the origin of a response is called a response key.

Another way of classifying keys is by the type of public data associated with them. If the data is related to the type of operations possible using the key and with the targets of the operations, the key is called a capability key. If the data is related to the identity of the possessor or group membership, the key is called an identity key.

Both capability and identity keys can be used as access keys. If the objects have access lists associated with them, the device will use identity keys to check access. If access lists are not used, the device must check access using capability keys. Access lists imply fewer keys managed at the clients, but more metadata managed at the devices. Capability keys require very little metadata managed at the devices, but more keys managed by the clients.

Since clients are only interested in authenticating the device that generated a response, response keys are always identity keys. A client receives a response key, generated specifically for that client by the administrator, to authenticate responses from a specific device.

Generating capability keys

A capability key allows performance of a specific operation on a storage device. The data used to generate the key govern the type of operation permitted and the details of that operation.

The capability key is generated by hashing the disk key with the key data. The key and its corresponding data are given to the client. Note that the capability key must be kept secret, requiring a secure channel to send the key to the client.

A capability key can generate another capability key that is a restricted subset of the capabilities of the first key. Anyone in possession of a capability key can do this, not just the administrator, which makes it convenient for highly distributed file systems. When distributing the new capability key, the new key's corresponding data includes the data used to compute the new key and the key data from the original capability key.

For example, in Figure 1 if the administrator wishes to grant Bob the ability to read and write object 232 on the storage device, the administrator would generate K_1 with the READ and WRITE attributes in $data_1$ along with object 232. Bob could then grant Brenda the ability to read object 232 by only including the READ attribute and object 232 in $data_2$. Brenda could generate another capability key to read object 232, but could not generate a capability key to write to object 232, since the WRITE attribute is not among the capabilities of the key that Brenda possesses.

Generating identity keys

Identity keys allow a receiver to check a sender's identity by including an identification string as part of the key data. Like capability keys, identity keys are generated by hashing the identification string as part of the key data and the disk key. The client receives, via a secure channel, the resulting identity key and the corresponding key data.

As with capability keys, identity key owners can generate other identity keys. When a new identity key is generated, the possessor of the original key vouches for the new key holder's identity. This lets a nonadministrative user create a new identity key to access objects that the original user can already access.

For example, in Figure 1 if the administrator wishes to identify Bob to the storage device, the administrator would include a string identifying Bob in $data_1$. Bob could then create a new key identifying Brenda to the disk by including a string identifying Brenda in $data_2$. Note that the storage device would only recognize K_2 as valid if Bob were authorized to identify other users or if Brenda were only accessing objects that Bob can access.

SCARED security protocol

SCARED addresses three aspects of security: identity and capability, integrity, and freshness. When a message is received, the recipient must validate who sent the message or at least that the sender was authorized to send the message. Next, the receiver needs to validate the message's integrity by verifying that the message was not changed in transit. Since the recipient can validate the sender, it would seem to imply that the recipient could also validate that the message is the one sent by the sender. In practice, this integrity guarantee is not always available. SCARED enables the network storage to validate both the identity of the sender and the message's integrity. Finally, the receiver must validate that the message was sent recently (that the message is fresh) or at least validate that the message is not a replay of an older message.

The first phase of the SCARED protocol is to establish a freshness guarantee. After a freshness guarantee is established, the clients and storage use the message protocol to send requests and receive replies. When presenting the protocols, it is assumed that clients are in possession of the keys needed for accessing the storage, and that the storage is only in possession of the disk key. The access key used by the client is denoted by K_a, and the key data corresponding to K_a is denoted by D_a. The response key is denoted by K_r and its key data D_r.

Freshness guarantees

To guarantee the freshness of messages, SCARED uses timers, nonces (numbers generated in such a way that the same number is not generated twice), and counters. When using timers, all parties involved in a transaction have timers that are reasonably synchronized. Nonces and counters do not require clocks, but do require that the nonce and counters never take on the same value. Counters must be monotonically increasing.

SCARED addresses three aspects of security: identity and capability, integrity, and freshness.

The clients always use nonces to check the freshness of a response, since a nonce is the freshness guarantee with the fewest requirements. When illustrating the protocol exchange, we denote the client nonce using F_c.

Storage devices require clients to include a timer or counter in the request to check the request's freshness. Since the client must calculate the freshness guarantee that the device is using, nonces cannot be used. If the communication with the device is session oriented, the device can key a counter synchronized with the device based on the number of messages sent, otherwise, a timer must be used.

We denote the storage counter or timer using F_s. The FGRequest and FGResponse are op codes used to request and receive the initial freshness guarantee. Before making requests to the storage, the client must request the storage counter or nonce using the following protocol:

$$\begin{aligned} &\mathrm{C} \rightarrow \mathrm{S{:}M}=\{\mathrm{FGRequest}, F_c, D_r\}, \mathrm{MAC}_{Kr}(\mathrm{M}) \\ &\mathrm{S} \rightarrow \mathrm{C{:}M}=\{\mathrm{FGResponse}, F_c, F_s\}, \mathrm{MAC}_{Kr}(\mathrm{M}) \end{aligned}$$

When the storage receives the request in the first message, the storage generates K_r using D_r, as shown previously. If $\mathrm{MAC}_{Kr}(\mathrm{M})$ as calculated by the storage device matches $\mathrm{MAC}_{Kr}(\mathrm{M})$ included in the request, the device knows that M was generated by a client in possession of K_r, so it will generate a response using K_r. The storage device copies F_c unchanged into the response.

When the client receives the second message, it can check the MAC since it is in pos-

The two approaches used by SCARED to check access are identity and capability based.

session of K_r and thus knows it came from the storage. The presence of F_c in the response lets the client know that the message is in response to the first message. After the message exchange, the client has F_s, the server freshness guarantee, to establish the freshness of future communications with the server.

Verifying freshness using counters

If communication with the storage is session oriented, counters are convenient to use for checking the freshness of requests, since they do not require clocks. At the beginning of the session the client will obtain F_s, the initial session counter. Each time the client transmits a packet, it includes the counter in the request and increments the counter for the next request.

The device can verify the freshness of the request by ensuring that the request includes a counter that is one greater than the previous request from the client. This means that the storage device must maintain a counter for each active session. The initial counter sent to the client must be generated in such a way that a counter used in a session with the device was never used before.

Verifying freshness using timers

If the communication with the storage device is not session oriented, timers are used to let the device check freshness without keeping freshness information about all clients. To use timers, all clients intending to communicate with a storage device must synchronize their timers with those of the storage device. This is done by setting F_s to the current device timer in the first phase of communication with the disk.

The client synchronizes its timer with the storage device by saving the difference between its timer and the storage device's timer. Since the client maintains a delta between its timer and the device's, the storages devices with which it communicates need not, and in most cases will not, have synchronized timers. The client includes the device's current timer in all requests to the storage device. This enables the devices to check that the message was sent recently.

Because of network latencies, clock drift, and the latency of responses to requests, checking time stamps alone does not provide a strict guarantee of freshness. In particular, an attacker could replay transactions in a small time window. To thwart this recent-past replay attack, the storage device keeps a list of message authentication codes used in the recent past and checks them with each message. If the code exists in the list, the message is considered a replay.

To compensate for clock drift, the storage device includes its current timer in all responses. The clients can then resynchronize their timers each time the storage device sends a response.

The request protocol

Clients communicate with the storage devices using a request and response protocol. The client request has the form:

$$C \rightarrow S: M = \{\text{Operation, data}, D_a, D_r, F_c, F_s\}, \text{MAC}_{Ka+Kr}(M)$$

The operation requested and the data that goes with the operation are followed by the key data for the access and response keys used in this communication with the network storage. The device regenerates K_a and K_r using D_a and D_r, so that it can verify the MAC. F_s is included to ensure the freshness of the message using either the counter- or timer-based techniques.

If the MAC is valid, the device knows the message arrived intact and that it was sent by a client in possession of K_a, but it still must verify the client's ability to request the operation. The two approaches used by SCARED to check access are identity and capability based. In identity-based systems, the disk checks access authority based on the requester's identity. In capability-based systems, the disk is only interested in the requester's ability to perform a transaction.

Since the administrator or a client in possession of a capability grants capabilities by generating access key K_a with the capabilities contained in key data D_a, the client in possession of K_a also has the capabilities listed in D_a. Since K_a may be derived from other access keys, the disk must ensure that the derived key's capabilities are a subset of the original key. To check if the client can carry out the requested operation, the device checks that it is listed as one of the capabilities.

If identities are used, D_a will contain the identity of the requester. In order for the disk to check the requester's ability to perform an operation, the disk maintains access lists on each object. When a request arrives, the disk checks the identity in D_a against the access list of the requested object.

Response protocol

The authentication needs of the clients are simpler than the needs of the disk. The client only has to verify that the disk sent the response in reply to the client's request. The device response has the form:

$$S \rightarrow C: M = \{\text{Response, data}, F_c, F_s\}, \text{MAC}_{K_r}(M)$$

K_r is used in the MAC since it is the secret shared by the client and disk. Different clients may share the capability and identity keys, but response key K_r is held by only one client. After validating the MAC, the client knows that the response arrived intact from the disk. The presence of F_c allows the client to check that the response is for the request containing F_c. F_s is included to compensate for clock drifts if the disk uses timers.

The key data are not included in the response since the requester must already possess K_r.

Revocation

The best way to address the problem of key revocation is to make the keys secure. Smart cards and tamper-resistant chips provide key security. However, the smart cards can be lost, which would again necessitate the revocation of the keys in the cards.

SCARED implements three ways of revoking keys: keys have a limited lifetime; valid keys are controlled at the target; and all keys for the storage device can be revoked by changing the disk key.

The storage device only revokes access keys, since it isn't necessary to revoke response keys. Because the client uses response keys to authenticate responses from the disk, the client simply stops using a revoked key. The response key does not have access rights associated with it, so an attacker could not gain access to a storage device using a revoked response key. No client would recognize responses using the revoked key, so an attack against a client with a revoked key would also be useless.

The best way to address the problem of key revocation is to make the keys secure.

Key expiration. When giving a key to a client, the administrator can include an expiration time in the key data. Since a key is only usable at one target, the expiration time is relative to the timer on that target. Using relative time removes the need for synchronized clocks. The device checks whether an access key is expired by comparing the expiration time in the key data to its current timer.

Capability key revocation. To aid in capability key revocation, we associate *salt* to the key. Salt is a number, much like a nonce, that will never change to a previously held value. It is not considered secret and is stored with every object or metadata entry. When a capability key is generated for an object or entry, the salt is included in the key data. When the key is used, the salt in the key data must match the salt in the object or entry for which an operation occurred.

When the salt is changed at an object or entry, all keys that included the original salt are invalidated since the salt in the keys won't match the new salt.

Identity key revocation. Identity key revocation is possible in two ways. One method uses revocation lists for unexpired and invalid identities. The other method is a simpler revocation scheme that requires the storage device to know a priori the identity of clients with whom it will communicate.

When key expiration information is present in the key data, only keys that haven't expired need revocation. If it is assumed that most keys that aren't expired are valid, then an efficient way of revoking keys is to give a list of key revocations to the storage device. Based on the previous assumption, the revocation list should be short so the identities present in requests to the disk could be checked against the list before accepting them as valid. Once a revoked key is expired, it would be removed from the revocation list to keep the list size small.

The second way of identity-based authentication is to include a counter in the identity key calculation. The counter is stored in a table on the storage device and indexed by the client identifier. When a client makes a request, the device verifies that the counter in the table is less than or equal to the counter included in the key data of the request. If the counter in the table is less than the key data counter, the device sets it to equal. To revoke a key, the storage device generates a new key with a new counter. When the new key is used, the table is updated and the old keys become invalid.

What to do about encryption

A main feature of a secure distributed file system is the confidentiality of file data. Currently, of the commercial distributed file systems, only DFS[9] has the option of encrypting data exchange between client and server.

A stronger level of data privacy is obtained if the client encrypts the data before sending it to the server for storage. The Cryptographic File System[10] (CFS) performs this kind of client-side encryption. CFS encrypts data before it is stored in a shadow file system and decrypts the data as it is read. Using CFS with SCARED would keep the data confidential and avoid the negative performance impact of encrypting at the storage devices.

CFS has a key distribution problem, since users must remember and distribute the encryption keys. To overcome this problem, we propose storing the encryption keys in the metadata encrypted with group and user encryption keys. This lets users obtain keys at the moment they are needed.

One problem with storing the encryption keys in the metadata is that if group or user encryption keys are changed, all the metadata must be updated by reencrypting the keys using the new keys.

If the storage devices are trusted to keep data confidential, encrypting and decrypting at the storage devices avoids problems with encryption key distribution. To encrypt the data between the client and storage devices, they must share an encryption key. They already share a response key, so rehashing the response key with a public constant generates an encryption key. Requiring the storage device to do link-level encryption increases the processing requirements of the device.

Whether or not the network storage is involved in ensuring the confidentiality of the data, the SCARED protocol satisfies the authentication requirements of network storage.

The importance of distributed computing as the pivotal approach to managing computing resources and data is well recognized. Scaling distributed computing solutions is a challenge. Network-attached storage provides a solution for creating scalable network access to data, but requires reliable and efficient authentication techniques to ensure that while data is widely accessible, its content is secure from unauthorized access. The SCARED architecture provides a mechanism for efficient and reliable authentication to network accessible storage. We are building a distributed file system on top of SCARED which we call Brave. It is serverless in the sense that there is no central file server, but it stores all data and metadata on network-attached storage unlike the serverless file system described earlier. Currently, we have a prototype virtual file system (VFS) for Linux and a SCARED device written in Java. MICRO

Acknowledgments

We thank Ted Anderson of Transarc for his invaluable comments while writing this article.

References

1. E. Riedel and G. Gibson, "Understanding Customer Dissatisfaction with Underutilized Distributed File Servers," *Proc. Fifth NASA Goddard Space Flight Center Conf. on Mass Storage Systems and Technologies*, 1996, http://www.pdl.cs.cmu.edu/PDL-FTP/NASD/Goddard96.abstract.html.
2. C. Neumann and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks, *IEEE Communications Magazine*, Sept. 1994, pp. 33-38.
3. L.-F. Cabrera and D.D.E. Long, "Swift: Using Distributed Disk Striping to Provide High I/O Data Rates," *Computing Systems*, Vol. 4, No. 4, 1991, pp. 405-436.
4. J.H. Hartman and J.K. Ousterhout, "The Zebra Striped Network File System," *Proc. 14th Symp. on Operating Systems Principles*, ACM Press, New York, 1993. pp. 29-43.

5. G.A. Gibson et al., "File Server Scaling with Network-Attached Secure Disks," *Proc. ACM Int'l. Conf. on Measurement and Modeling of Computer Systems* (Sigmetrics), ACM Press, 1997, pp. 272-284.
6. J. Howard et al., "Scale and Performance in a Distributed File System," *ACM Trans. on Computer Systems*, Vol. 6, No. 1, Feb. 1988, pp. 51-81.
7. M. Dahlin, *Serverless Network File Systems*, PhD dissertation, University of Calif. at Berkeley, Computer Science Dept., 1995.
8. H. Krawczk, M. Bellare, and R. Canetti, "Keyed-Hashing for Message Authentication," *Request for Comment* (*RFC) 2104*, Internet Engineering Task Force (IETF), Feb. 1997, http://www.ietf.org/.
9. C. Everhart, "Security Enhancements for DCE DFS," *OSF RFC 90.0*, Open Software Foundation (OSF), Feb. 1996.
10. M. Blaze, "A Cryptographic File System for Unix," *Proc. First ACM Conf. Communication and Computing Security*, ACM Press, 1993, pp. 9-16.

Benjamin C. Reed is a software engineer in the Computer Science Department at the IBM Almaden Research Center working in the area of networking and systems management. He received his BA from Miami University, his MS from DePaul, and is currently a doctoral candidate at the University of California Santa Cruz.

Edward G. Chron is a senior developer in the Computer Science Department at the IBM Almaden Research Center. He received his BSEE from the University of Illinois at Urbana-Champaign.

Randal C. Burns is research associate in the Department of Computer Science at the IBM Almaden Research Center. He is also a doctoral candidate at the University of California, Santa Cruz. He received his BS from Stanford University and his MS from the University of California, Santa Cruz. His research interests include storage systems, distributed computing, fault tolerant computing, and concurrency control.

Darrell D.E. Long is a professor of computer science, and associate dean in the Jack Baskin School of Engineering at the University of California, Santa Cruz. He is also a visiting scientist at the IBM Almaden Research Center. He received his BS degree from San Diego State University, and his MS and PhD from the University of California, San Diego. His research interests include storage systems, distributed computing, security, and multimedia.

Direct questions about this article to Benjamin C. Reed, IBM Almaden Research Center, 650 Harry Rd., San Jose, CA 95120; breed@almaden.ibm.com.