Architecture of the Secure File System

James P. Hughes
Storage Technology Corporation
jim@network.com

Christopher J. Feist
Storage Technology Corporation
chris.feist@network.com

October 23, 2000

Abstract

The Secure File System (SFS) provides transparent, end-to-end encryption support to users accessing files across any network. In this paper the architecture of the Secure File System is described.

1 Introduction

In this paper we describe the architecture of the Secure File System (SFS) which provides end-to-end encryption and key management support to users accessing files on local or networked file systems. The current architecture consists of an access control list, smart card authentication, a Group Server, and SFS Client. In the rest of this paper we describe previous work in encrypting file systems and outline the current Secure File System architecture.

2 Previous Work

This section reviews other research and product efforts in encryption for shared, persistent storage.

2.1 Volume Encryptors

Several disk encryption systems are available which use a device driver layer (called a filter driver in Windows) to encrypt and decrypt data as it is sent to and received from a disk. These systems include PGP Disk from Network Associates, the Secure File System by Peter Gutmann at the University of Auckland [1], and TorDisk by Alexander Tormasov [2]. Though convenient for protecting whole disk volumes, volume encryptors do not allow access controls on fine-grain objects like directories and files.

2.2 File Encryptors

To achieve end-to-end encryption the encryption can be done in the presentation layer or application layer. Brute force encryption at the application layer requires that all applications that need to work with encrypted files be rewritten to include support for encryption. This is clearly unacceptable for storage systems.

Files may be encrypted on a per-file basis using tools like PGP, developed by Phil Zimmerman [3]. Though useful for the short-term encryption requirements of a single user, it is not generally useful when managing shared information stored for the long term because it is based on identity of the consumer. If the consumer is not directly known or changes (as is the case in many organizations) then any PGP-encrypted file must be re-encrypted.

2.3 File System Encryptors

The Cryptographic File System (CFS) [4] was developed by Matt Blaze at AT&T. CFS allows users to encrypt files on a per-directory basis using a single key. An NFS layer implemented the encryption, decryption, and key management locally on a trusted client: files were encrypted while in transit between the trusted client the untrusted network and server. In the original CFS implementation, sharing files require sharing the keys. The key distribution problem makes this difficult. Blaze proposed a key management scheme that helps address these issues in [5].

The Transparent Cryptographic File System (TCFS) [6] was developed at the University of Salerno, Italy. It improves on the CFS design by removing the NFS client encryption layer but still has a limited key management scheme.

The Satan File System was developed at Carnegie-Mellon University. The implementation employs C library modifications that read a file into main memory, decrypt the data and then deliver it to the application. The basic idea is to link the applications against a set of libraries that provide encrypted versions of standard library calls. This solves the problem of rewriting the applications, but the applications still need to be recompiled or at least re-linked. Also every program will have to have an unencrypted *and* encrypted version for working with encrypted files or unencrypted files.

IBM's Distributed File System (originally known as AFS and then commercialized by Transarc, a company acquired by IBM) assumes that security is a network problem. Many systems expect users and administrators to assume that their implementation is trusted and that network security measures can be effectively implemented independently of other security measures. This also assumes that the security measures for the backups, HSM, file caches and administrators themselves are flawless.

The Networked Attached Secure Disks (NASD) project [7] at CMU created security for accessing files on storage devices (NASD disk drives) attached directly to a network. Network keys are generated and distributed to the users. The entire system is based on the file system controller being trusted. NASD also has a single symmetric master key between each file system and disk drive.

Microsoft's Encrypted File System (EFS) is available in Microsoft NT 5.0. It can encrypt and decrypt on a per-file and per-directory basis. This system specifies that there are backup ”persona's” that have access to all data in the clear and thus administrative data protection.

Other related work includes [8], [9], [10], [11].

3 The Secure File System

The Secure File System is being developed as an OS- and application-independent security middleware. SFS takes an end-to-end approach to data protection by cryptographically protecting data at the source and unprotecting the data at the destination.

Cryptographically sealing a file or directory at the source allows the information infrastructure security to be relaxed. Routine system administration, backup/restore, and archiving need not be trusted nor even physically secure because the data is protected at all times.

SFS also allows decentralized access rights, where small groups of people can define who is allowed to look at certain information (without the help of an untrusted system administrator) and what set of rules must be followed to do so. This is critical because secure information sharing is required for effective cross-organiaztional decision making.

We define the following terms:

- **Information Producer** — has (by definition)

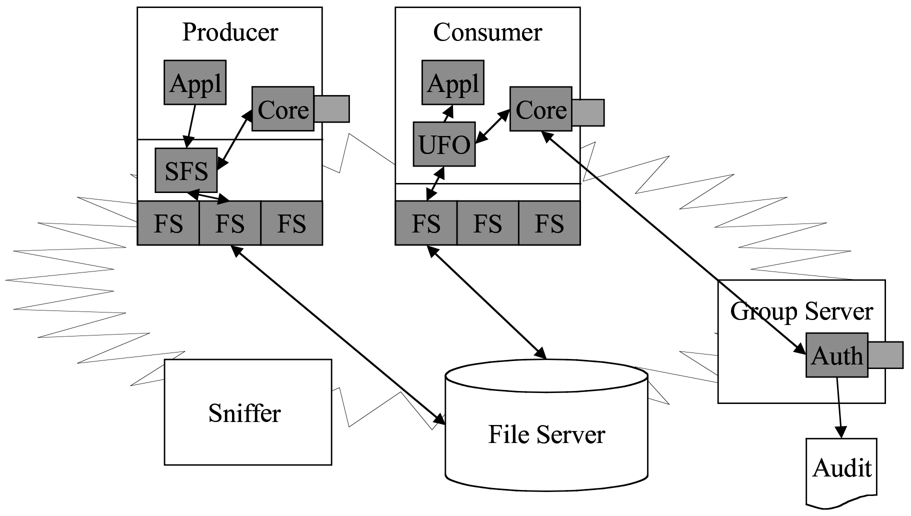


Figure 1: Secure File System Overview

the data in the clear and has the authority to define who can ”see” their data.

- **Information Consumer** — needs (by definition) the data in the clear and has the (undeniable) ability to pass the information on.
- **Group Agent** — determines group membership (who needs to know), and provides a non-repudiable audit trail.

Information needs to be communicated between the producer and the consumer (see Figure 1): except for the group agent, no other entity is allowed to ”see” the information in the clear.

Information ownership is based on organizations and their mandate: producers and consumers can dynamically define and create their own groups for information sharing.

Trusted systems are difficult. Trusted networks of file systems are virtually impossible. This system is focusing instead on creating information security middleware to protect information from producer to the consumer so that the number of entities that must be trusted is minimized.

- True ”End-to-End” — other systems encrypt a link to a web server, but SFS encrypts from the information producer to information consumer.
- Distributed Membership — defined by the user deparments, not by system administrators.
- Stored Encrypted — backups, archives, networks, etc. are all protected.
- Audit Trails — non-repudiable audit trails of file access requests are maintained by the Group Agent.
- Flexible Tag Information — by its purpose(s), not just the producer’s identity: Board of Directors, Executive Policy Makers, Project X, etc.
- Cryptographically Enforced — strong encryption used for the information and smart cards and PKI used to secure and manage keys.

The producer defines ”Access Control Metadata”. The access control list (ACL) is a specification of purpose of the information which defines the need to know. This specification of purpose is defined through an access formula which defines the actions of the consumer to agents of the producer (or others) to unlock the information produced by the consumer.

By solving the formula, the key to the file is revealed (potentially only) to the consumer. The formula can be any combination of users, groups, and projects. Any combination of logical AND and OR operations as well as N-person control can be used. N-person control for backups and sensitive information requires that multiple identities approve access or the user has organizational membership. An example of the meta data format is shown in Figure 2.

4 Real-world examples

4.1 Private projects within an organization

Every organization has projects that are more sensitive than others. These projects include people that are not typical IS support personnel. These projects include corporate reduction projects, employee medical information, mergers and acquisitions, Board of directors, executive compensation, etc. This sensitive information is typically handled by locking up paper copies. Long term centralized storage of this kind of information is both dangerous to a corporation’s viability and can potentially result in SEC, civil and criminal penalties.

This kind of sensitive information should not be stored on centralized file servers because of the large number of people outside the project membership required to manage the networks, servers, backups, etc. It is typical that 5 percent of every organization is Information Systems support personnel. While IS support is a necessity, the fact that a sensitive 10 person project within a 10,000 person company has 500 people to support them is a significant vulnerability.

While each support person does not have universal access or knowledge, each of the professional systems administrators (SA) have access to a portion of the process. The email SA can surf the email, the desktop SA has remote access to the desktop machine, the file server SA can surf the files, the network SA can sniff the networks, the backup SA can surf the backups and even the person that works in the warehouse that contains the backup tapes can either access the tapes or give others access to this sensitive information.

This system is designed to protect the data from one project member’s desk to the other. Management of the project membership is not an IS job, a simple to operate, tamper resistant ”group server” allows group membership to be directly managed by a project member in a decentralized manner.

Once data is protected in this manner, malicious, disgruntled or just curious employees are no longer even tempted to access this information because the information is protected.

A side issue to note is that if a hacker gains access to an internal network or even gains access to a SA account, the protected information will not be vulnerable.

4.2 Outsourced Information storage

While IS employees are a potential risk, the tendency of companies to outsource their IS professionals as well as the IS equipment to outside companies either at their own location or the other companies location. This increases the risks to sensitive information. The same outsourcing company can have a competitor as a customer and the potential for information cross pollination is there.

By protecting the information while it is still under control of the project members, it does not matter who manages the storage.

4.3 Outsourced Intranet Servers

Most companies have many more internal Intranet web servers than Internet servers. While today it is possible to outsource the management of the Internet web server, it is not possible to securely outsource the intranet web sites without protecting the data in a way that the administrators of the hosting web site are not a vulnerability.

This technology can allow all of a companies sensitive intranet information be stored and managed by outside internet service providers such as America Online (AOL).

4.4 Sharing sensitive information

Many organizations have partnerships with other organizations and have a necessity to share sensitive information with that organization. Today, this is managed by allowing one organization to have access to the other, maybe with leased lines behind the firewalls. This results in a significant amount of administrative overhead for networks and foreign access to file servers and significantly higher vulnerability. SFS allows sensitive information to be protected and then published outside the firewalls so that the other organization can access the information that they need without having unfettered access to the entire organization.

This sharing can be anonymous to the producing information or can be with audit trails back to the producing organization.

Another aspect of sharing sensitive information is the ability of project members to be able to mandate multi person control of information access and data recovery. Other systems lack this fundamental feature.

5 Components

5.1 Access Control List

The access control list allows the producer to explicity define who has access to their data. It is defined using XML (eXtensible Markup Language) and allows for several different methods of specifying access. The following is a list of keyword and their definitions which are present in the Secure File System Access Control Lists.

- OwningGroup — The group which owns the file.
- Writer — The producer of the file.
- ACL — Specifies the begining of the Access Control List.
- any m = n — Specifies that 'n' XML sub-blocks must be satisfied in the current XML block for access to be permitted.
- individual — Specifies an individual which has access to the file.
- group — Specifies a group server which will be used to verify membership in the project.
- project — Specifies a project which has access to the file.
- key data — The ciphertext of the file's key encrypted to an individual or group server depending on the enclosing XML block.

An example ACL is shown in Figure 2. This example starts out by defining the owning group and owner of the file. This is then followed by several blocks of XML which define the access permissions of the file.

The statement ’any m=”1”’ means that only one of the enclosed blocks must be true to access the file. The first block uses the individual tag and allows the user (me@asdf.com) access to the file.

The second block allows access to members of the ”Celeron” project through ”Group Server A”.

The third block shows how access can be serially specified. To obtain access to the file the user must be a member of the ”Design” project on ”Group Server A” AND a member of the ”Pentium” project on ”Group Server B” (Note: the file key is encrypted to Group Server B”).

The fourth block specifies that the user must be a member of the ”Xeon” project on ”Group Server C” and a member of the ”Merced” project on ”Group Server B”. The difference between this XML block and the third XML block is that the key is cryptographically split between the two group servers. So successful access from ”Group Server C” results in obtaining half of the file key and successful access from ”Group Server B” results in obtaining the other half of the file key.

The fifth and final block states that three of the five enclosed sub-blocks must be satisfied to permit access. So three of the individuals listed in must request access to the file together to obtain access the file.

It is important to note that an Access Control List is not valid unless it has been digitally signed by the producer of the file. This prevents forging of ACL’s which would undermine the security of the Secure File System.

5.2 Group Server

The Group Server is the only trusted entity of the SFS architecture. All file keys are encrypted to the Group Server’s public key and stored in a header attached to every encrypted file. This header contains the access control list documented in Section 5.1.

```
<?XML VERSION="1.0"?>

<FileData>

  <!-- What authority am I doing this for -->
  <OwningGroup id="myproject"/>

  <!-- I write this. -->
  <Writer id="me@asdf.com"/>

  <ACL>

    <!-- Any one of the following options accesses the data -->
    <any m="1">

      <!-- Or I can read my own data, or else -->
      <individual id="me@asdf.com" >
        <key data="12341234 12341234 12431234"/>
      </individual>

      <!-- Or access through the following group -->
      <group id="Group Server A" project="Celeron">
        <key data="12341234 12341234 12431234"/>
      </group>

      <!-- Access through the following groups in series -->
      <group id="Group Server A" project="Design">
        <group id="Group Server B" project="Pentium" >
          <key data="12341234 12341234 12431234"/>
        </group>
      </group>

      <!-- Or get half the keys from the following locations-->
      <any m="2">
        <group id="Group Server C" project="Xeon">
          <key data="12341234 12341234 12341234 12431234"/>
        </group>
        <group id="Group Server B" project="Merced">
          <key data="12341234 12341234 12341234 12431234"/>
        </group>
      </any>

      <!-- Escrow -->
      <any m="3">
        <individual id="VP Engineering">
          <key data="12341234"/>
        </individual>

        <individual id="VP Manufacturing">
          <key data="12341234"/>
        </individual>
        <individual id="VP Operations">
          <key data="12341234"/>
        </individual>

        <individual id="CTO">
          <key data="12341234"/>
        </individual>

        <individual id="CEO">
          <key data="12341234"/>
        </individual>
      </any>
    </any>
  </ACL>
</FileData>
```

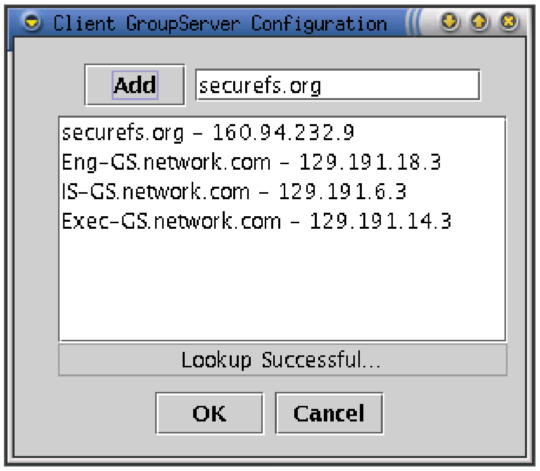
Figure 2: XML Access Control List

When a user requests access to a file the header is forwarded to the Group Server which then determines if the user has access to the file. If the user does have access to the file the file key is then returned to the user so they are able to decrypt the data.

The Group Server also serves as the single administrative point of the SFS architecture. All addition and removal of users from groups and projects is done on the Group Server. An extensive audit trail is also kept on the Group Server which can detail what file was opened, when it was opened, from where, and who opened it (see Figure 5).

5.3 Smart Cards

Smart Cards are used for authentication of individual users to the Secure File System. Each smart card contains its own microprocessor and a small amount of non-volatile RAM. This allows the user's private key to be stored only on the smart card and nowhere else. Whenever a document or key must be signed or decrypted all crytpographic functions requiring the private key are performed by the microprocessor on the smart card. Using this method the private key never leaves the smart card. In fact, even the owner of the smart card will never know his or her private key, only the PIN which unlocks it.



securefs.org

securefs.org - 160.94.232.9
Eng-GS.network.com - 129.191.18.3
IS-GS.network.com - 129.191.6.3
Exec-GS.network.com - 129.191.14.3

Lookup Successful...

Figure 3: Group Server Modification

5.4 SFS Client

This component of the SFS Architecture resides on each individual user's machine and communicates directly with the Group Server, Smart Cards and file system interface. It contains the GUI which the user uses to control SFS and view the current status of SFS (see Figure 6). The SFS Client also allows the user to configure which Group Servers to use when reading and creating XML files (see Figure 3).

```
<?XML VERSION="1.0"?>
<FileData>
  <OwningGroup id="Single Person Project"/>
  <Writer id="chris.feist@network.com"/>

  <ACL>
    <any m="1">
      <individual id="chris.feist@network.com">
        <key data="12341234 12341234 12431234"/>
      </individual>
    </any>
  </ACL>
</FileData>
```

Figure 4: One Person Access ACL

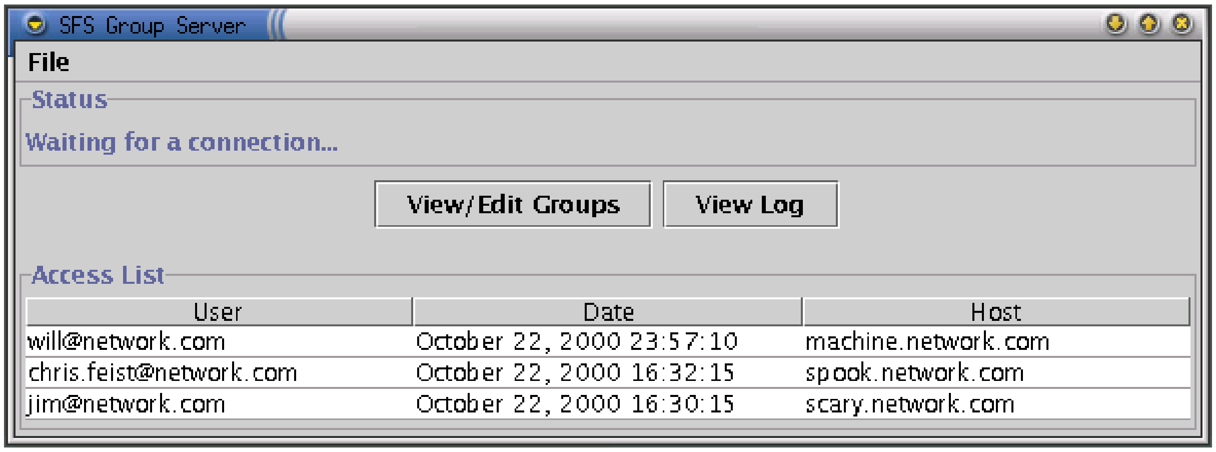
5.5 File System Interface

The file system interface is the OS dependent portion of the Secure File System. It is designed to catch all system calls relating to filesystem access and reroute them to the SFS Client which performs the protection and unprotection of files.

6 Examples

6.1 File Creation — One Person Access

A private file will be created by the ”producer” and will only be readable by the ”producer”. This example is illustrated in Figure 7.



Waiting for a connection...

User	Date	Host
will@network.com	October 22, 2000 23:57:10	machine.network.com
chris.feist@network.com	October 22, 2000 16:32:15	spook.network.com
jim@network.com	October 22, 2000 16:30:15	scary.network.com

Figure 5: Group Server Audit Trail

6.1.1 File System Interface

The user clicks on the save button of their word processor which makes a close() file system call to save the file. The call is then intercepted by the SFS File System Interface and rerouted to the SFS Client.

6.1.2 SFS Client

SFS Client recieves the close() call from the File System Interface and creates an Access Control List based on what the producer previously specified in the SFS Client GUI (files are readable only by producer). A random symmetric key for file encryption is then generated by SFS Client and the ACL is examined to determine whose public key the symmetric key will be encrypted to. SFS Client then determines that the file key must only be protected with the producer's public key and encrypts the symmetric key and inserts it into the ACL (Figure 4). A secure hash of the ACL is then generated and sent to the Smart Card for signing.

6.1.3 Smart Card

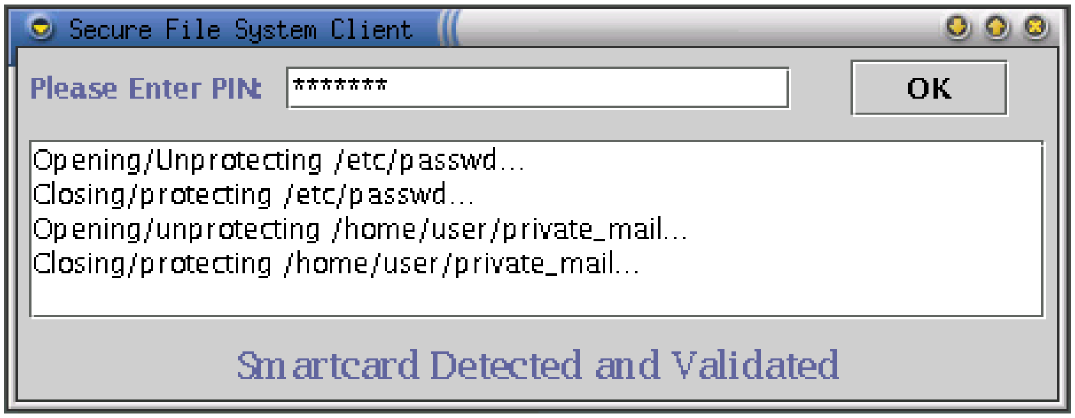
The Smart Card recieves the hash and requires that the user enter their PIN if it hasn't been entered in the current session or if the PIN has timed out. If the PIN was entered correctly, the Smart Card then signs the secure hash with the user's private key and returns the result to the SFS Client.

6.1.4 SFS Client

SFS Client then recieves the signed hash and attaches it to the ACL so its authenticity can be verified when opening the file. The cleartext file is then encrypted using the symmetric file key previously generated. The ACL is inserted at the beginning of the file and passed back to the File System Interface.

6.1.5 File System Interface

The File System Interface recieves the file created by SFS Client and passes it back to the filesystem.



```
Opening/Unprotecting /etc/passwd...
Closing/protecting /etc/passwd...
Opening/unprotecting /home/user/private_mail...
Closing/protecting /home/user/private_mail...
```

Smartcard Detected and Validated

Figure 6: SFS Client Main Screen

6.1.6 User Application

The file is now safely stored on a storage device and can only be read by the producer.

6.2 File Access — One Person Access

The file created in the previous example will be opened by the creator of the file. This example is illustrated in Figure 8.

6.2.1 File System Interface

The user clicks on the open button of their word processor which makes an open() system call to open the file. The call is then intercepted by the SFS File System Interface and rerouted to the SFS Client.

6.2.2 SFS Client

SFS Client recieves the open() call from the File System Interface and extracts the ACL header (Figure 4) from the file for processing. SFS Client then checks to see if the user has access to the file by examining the ACL. Since the user is permitted access through the ”individual” keyword no communication with the Group Server is necessary. The encrypted file key is then forwarded to the Smart Card.

6.2.3 Smart Card

The Smart Card recieves the encrypted file key and if the PIN is entered or has been entered in the current session the file key is decrypted with the private key on the Smart Card and returned to the SFS Client.

6.2.4 SFS Client

SFS Client recieves the file key from the smart card and then decrypts the file and returns it to the File System Interface.

6.2.5 File System Interface

The File System Interface recieves the decrypted file from the SFS Client and now can return the file to the user’s word processor.

6.2.6 User Word Processor

The user’s word processor successfully opens the file.

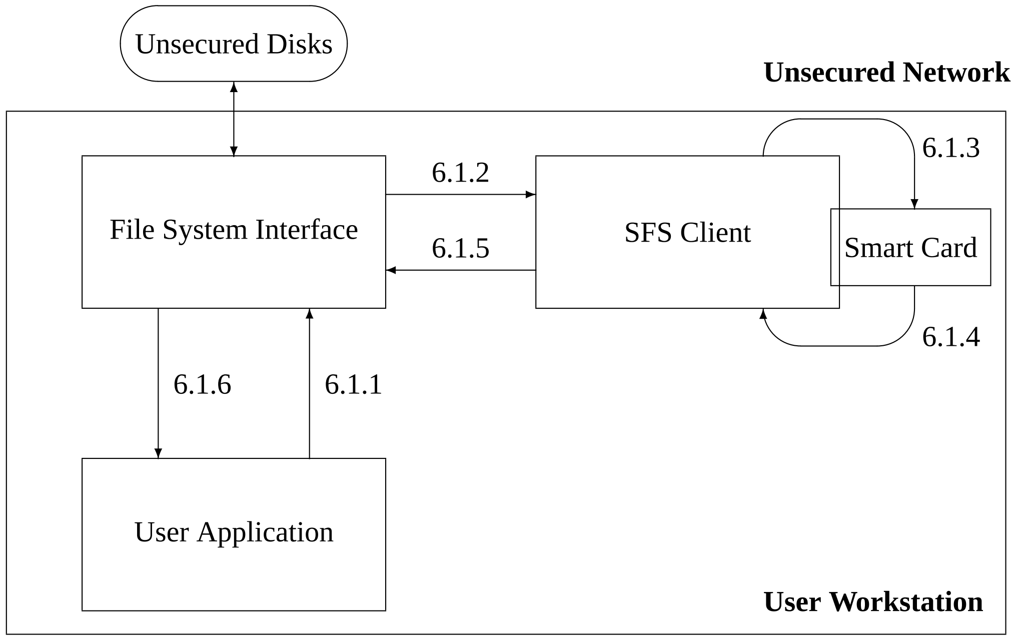


Figure 7: Example: File Creation — One Person Access

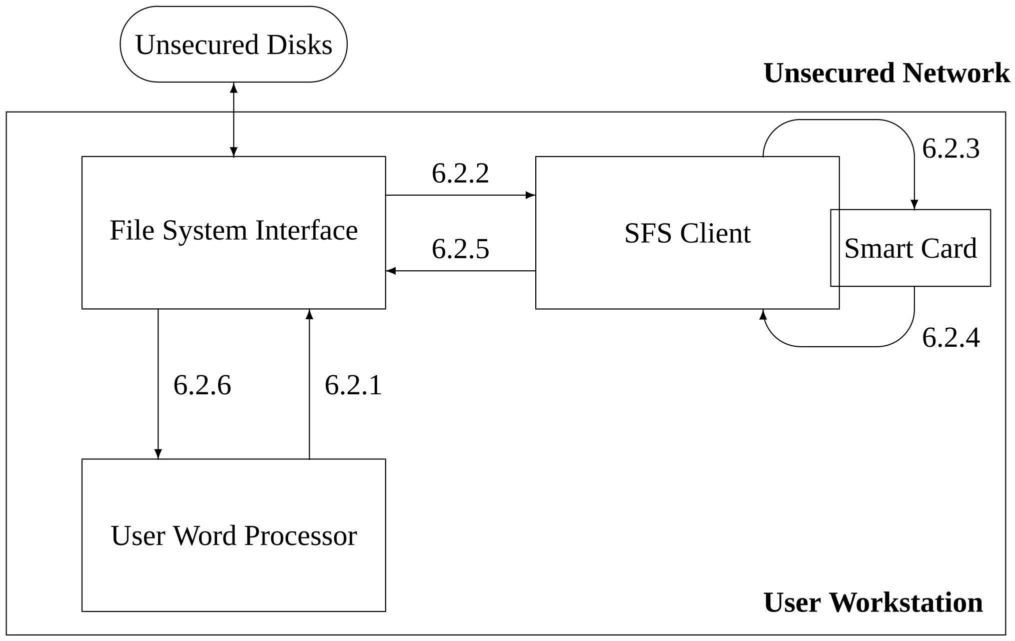


Figure 8: Example: File Access — One Person Access

6.3 File Creation — Multi-Project Access

A file will be created by the ”producer” and will be readable only by the producer and users who are either members of the ”Hardware Development” and ”ACME 886 Processor” projects or members of the ”Software Development” and ”ACME 886 Processor” projects. Access will be denied to all other users.

This example is virtually identical to Example 6.1, Figure 7.

```
<?XML VERSION="1.0"?>
<FileData>
  <OwningGroup id="Single Person Project"/>
  <Writer id="chris.feist@network.com"/>

  <ACL>
    <any m="1">
      <individual id="chris.feist@network.com">
        <key data="12341234 12341234 12431234"/>
      </individual>

      <group id="Eng. GS" project="Hardware Development">
        <group id="Eng. GS" project="ACME 886 Processor">
          <key data="12341234 12341234 12431234"/>
        </group>
      </group>

      <any m="2">
        <group id="Eng. GS" project="Software Development">
          <key data="12341234 12341234 12431234"/>
        </group>

        <group id="Eng. GS" project="ACME 886 Processor">
          <key data="12341234 12341234 12431234"/>
        </group>
      </any>

    </any>
  </ACL>
</FileData>
```

Figure 9: Multi-Group Access ACL

6.3.1 File System Interface

The user creates the file in their application and saves it to a SFS protected directory. The File System Interface intercepts the close() system call and reroutes it to SFS Client.

6.3.2 SFS Client

SFS Client recieves the close() call from the File System Interface and creates and Access Control List based on the previously stated user specifications. The generated ACL is displayed in Figure 9.

The remaining steps are identical to the first example (Steps 6.1.2 - 6.1.5), except the file key is encrypted to the producer and the ”Eng. GS” Group Server.

6.4 File Access — Multi-Project Access

The file created by the last example will be opened by a user who is a member of the ”Software Development” project and ”ACME 886 Processor” project. This example is illustrated in Figure 10.

6.4.1 File System Interface

The user opens the SFS protected file inside of their application which causes an open() system call to be generated. This call is then intercepted by the SFS File System Interface and rerouted to the SFS Client.

6.4.2 SFS Client

SFS Client recieves the open() call from the File System Interface and extracts the ACL header (Figure 9) from the file for processing. SFS Client examines the header and finds that the user is not ”chris.feist@network.com”, causing the first XML block to fail. The user is also not a member of the ”Hardware Development” project so the second block fails. Finally, in the third XML block SFS Client finds that the user is both a member of the ”Software Development” project and the ”ACME 886 Processor” project thus fulfilling the ”any m=2” requirement in the third block and the original ”any m=1”.

Since the SFS Client has determined access should be permitted, it generates a secure hash of the ACL and sends it to the Smart Card.

6.4.3 Smart Card

The Smart Card recieves the secure hash and checks to see if the correct PIN has been entered. If it has, it signs the secure hash with the user's private key and returns the signature to SFS Client.

6.4.4 SFS Client

SFS Client now sends the XML file along with the newly created signature to the Group Server to verify that access is permitted.

6.4.5 Group Server

The Group Server examines the ACL that it has just recieved and verifies that the original signature (by the producer) is valid. It then checks to see who is requesting access, and verifies this be examining the second digital signature on the ACL. Once this is done the ACL is examined to see if the user does in fact have access to the file as the SFS Client does in section 6.4.2. Since the user does have valid access to the file the Group Server decrypts the file keys using its private key and re-encrypts them the the user's smart card. The newly encrypted file key is then sent back to SFS Client.

6.4.6 SFS Client

SFS Client recieves the encrypted file key and passes it to the Smart Card for decryption.

6.4.7 Smart Card

The Smart Card checks to make sure it has a valid PIN and decrypts the file key using the user's private key. The file key is then returned to the SFS Client.

6.4.8 SFS Client

SFS Client now decrypts the file using the file key returned from the Smart Card and then returns access to the File System Interface.

6.4.9 File System Interface

The File System Interface recieves the unprotected file passed from SFS Client and passes it to the user's application.

6.4.10 User Application

The user's application recieves the unprotected file and continues functioning as if nothing had happened.

7 Conclusion

To securely protect the information on a shared network, information must be protected from producer to consumer. By only securing the infrastructure (Networks, Backups, etc.) the information is not protected. Any person with administrative access to the network, disks or backups can obtain the information. The goal of the Secure File System is to combat this problem which it does using end-to-end encryption. Thereby protecting the data from the producer to the consumer and providing the user with true *Information Security*.

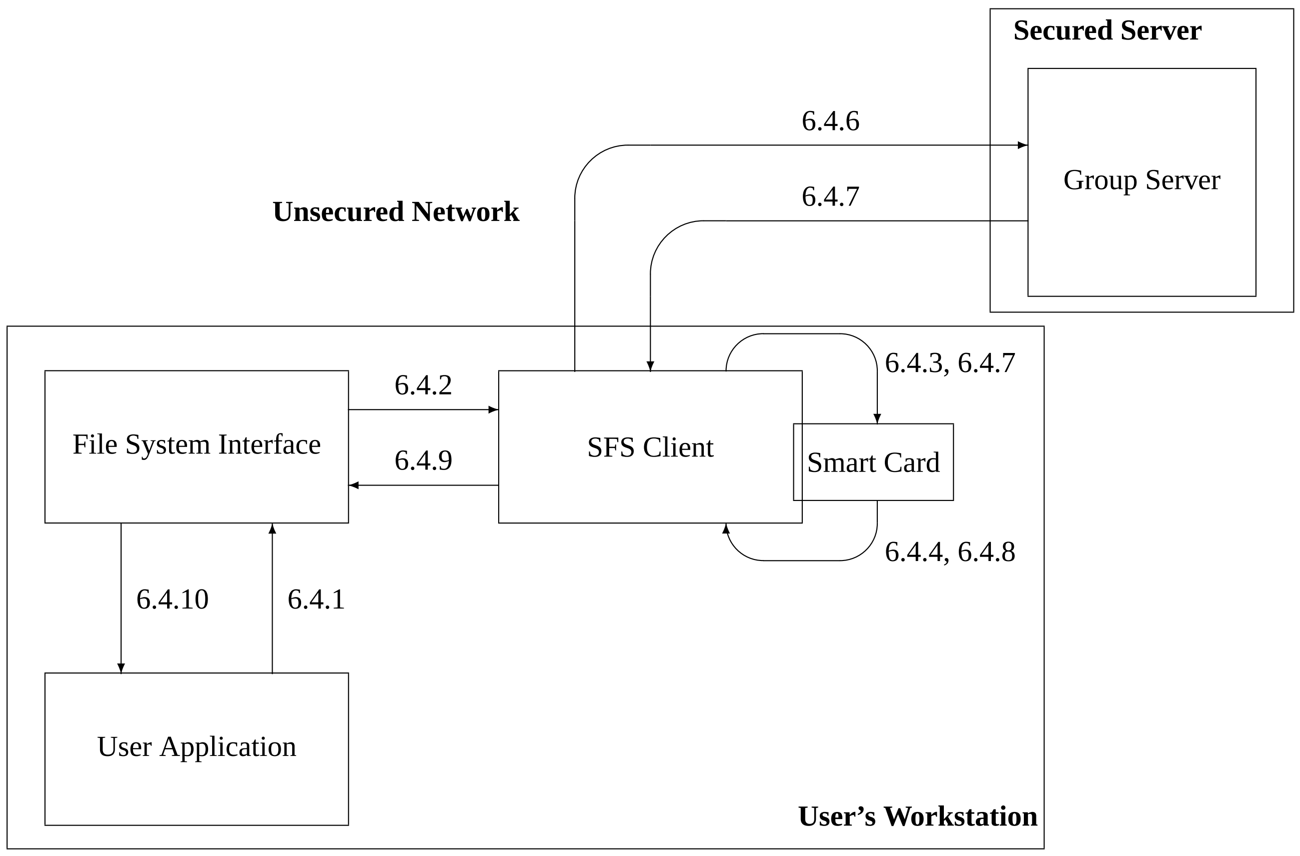


Figure 10: Example: File Access — Multi-Project Access

References

[1] Peter Gutmann, University of Auckland, New Zealand. The secure filesystem (sfs) for dos/windows. http://www.cs.auckland.ac.nz/ pgut001/sfs/index.html, September 1996.

[2] Alex Tormasov. The tordisk project. http://www.cs.auckland.ac.nz/ pgut001/sfs/index.html, November 1997.

[3] Phil Zimmerman. Pgp home page. http://web.mit.edu/pgp/.

[4] Matt Blaze. A cryptographic file system for unix. In *First ACM Conference on Communications and Computing Security*, pages 33–43, Fairfax,VA, November 1993.

[5] Matt Blaze. Key management in an encrypting file system. Boston,MA, June 1994.

[6] University of Salerno. Tcfs home page. http://www.globenet.it/ ermmau/tcfs/index.html, April 1997.

[7] Parallel Data Lab. Nasd home page. http://www.pdl.cs.cmu.edu/NASD/.

[8] Albert Alexandrov, Maximilian Ibel, Klaus Schauser, and Chris Scheiman. Extending the operating system at the user level: the ufo global file system. In *Proceedings of the USENIX Annual Technical Conference*, pages 77–90, 1997.

[9] Peter Braam and Philip Nelson. Removing bottlenecks in distributed filesystems. In *Proceedings of the 5th Annual Linux Expo*, pages 131–139, Raleigh, North Carolina, May 1999.

[10] Scott Guthery and Timothy Jurgensen. *Smart Card Developer's Kit*. Macmillan Technical Publishing, 1998.

[11] David Corcoran. http://www.linuxnet.com.