GSFS – A NEW GROUP-AWARE CRYPTOGRAPHIC FILE SYSTEM

Claudia Eckert, Florian Erhard and Johannes Geiger

Technische Universität München, Department of Computer Science

eckertc@in.tum.de

Abstract Cryptographic file systems ensure information confidentiality even in the presence of attacks which successfully circumvented the access controls of the underlying operating system. The paper presents our new system called GSFS which overcomes several severe deficiencies found in popular cryptographic file systems. In contrast to these systems GSFS offers a transparent group-based access control which supports a comfortable sharing of file between user groups. A prototype is available running integrated into the Linux operating system.

Keywords: File System, Cryptography, Access Control, Management, Operating System

1. INTRODUCTION

With the ever increasing number of mobile systems and networks a huge amount of sensitive information is managed within file systems. Controlling file accesses is one of the main tasks of a conventional multi-user operating system. At the same time, as the value of the stored data steadily increases, the theft of single data files or even whole storage media or mobile devices becomes very attractive. Therefore, the access control services of a traditional file system are insufficient to meet the security requirements coming along with open and cooperative environments. Additional measures are required to guarantee information confidentiality even in the presence of hostile attackers who successfully circumvented the access control mechanisms of the operating system. Of course, these additional measures must ensure that authorized users or user groups are still able to access their data without any restrictions. Obviously, cryptography [3] offers useful techniques to defeat the security threats we have already mentioned. This leads to cryptographic file systems. Such systems should fulfill essential security requirements in order to be widely accepted and used for daily work by users who are far from being security specialist.

Our paper is organized as follows. Section 2 outlines essential security requirements we think a cryptographic file system should meet. We then present

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35515-3_53](https://doi.org/10.1007/978-0-387-35515-3_53)

results of an analysis we have performed to study existing cryptographic file systems with respect to these requirements. Section 3 presents the main features of our system GSFS which was developed to meet the security requirements explained in section 2. The overall architecture and some implementation details of our prototype integrated into the Linux operating system are presented in section 4. Finally, we summarize the main contributions of our paper.

2. CRYPTOGRAPHIC FILE SYSTEMS

The design of a secure and nevertheless easy to use cryptographic file system is by no means an easy task. In order to give some guidelines for developers as well as users we identify main security properties a cryptographic file system should have.

2.1. SECURITY REQUIREMENTS

Preserving *data confidentiality* is the most important issue. As meta-data like directory structure, file names and file-sizes contain valuable information for attackers, encrypting the meta-data is necessary as well. Data confidentiality must be guaranteed even in the presence of an unauthorized data access resulting, for instance, from spoofing attacks. That is, all files and their meta-data should be stored in encrypted format as soon as and as long as possible. The file system should avoid storing files unencrypted while they are open for use.

Besides confidentiality, a cryptographic file system of high quality should provide for data integrity as well as for authentication of the data origin and for non-repudiation.

Cryptographic file systems should support a fine-grained *group-based* access control policy. That is, users should be enabled to securely share files by granting access privileges to dedicated user groups. Therefore, each user must be able to correctly encrypt and decrypt all shared files he is authorized to use. This service must be offered in a transparent way to reduce the users efforts.

The well known principle of ease of use [5] requires to reduce user interception to the minimum. This avoids security breaches stemming from faulty user actions. The need for an *automatic security management* with an transparent selection of appropriate encryption algorithms requires a high quality key management facility. Such a management should comprise key recovery capabilities as well as appropriate *backup techniques*. Whenever a backed up file must be restored, the management must be able to decrypt data which potentially is encrypted with an out-of-date encryption key.

To minimize the performance penalties introduced by security measures the user should be enabled to specify his *application-specific* security requirements. It is then up to the management to adjust its efforts spent for security measures

to meet these specifications and to reduce performance degradation as far as possible.

2.2. EXISTING SYSTEMS

We have classified existing systems offering file encryption facilities into three classes. We distinguish between (1) programs which just perform a simple file encryption, (2) systems for single user applications and (3) systems for multiple users offering group facilities.

2.2.1 Programs for File Encryption.

These programs are merely able to encrypt and decrypt single files lacking usual file system capabilities. The encryption and decryption operations must be explicitly invoked by the user, files are kept in cleartext on hard disks as long as no such encryption command has been initiated and data files are stored unencrypted while they are opened for usage. Systems of this class neither encrypt meta-data nor offer any support for a group-based sharing of encrypted data. Users are forced to explicitly select the files they wish to encrypt. But as no automatic management is offered it is up to the user to select passwords and keys to protect his data appropriately. This raises a lot of security problems, because it is the task of the user to maintain and memorize those passwords and keys, respectively. The weakness of password-based security burdening users with the management of sensitive information is well known.

Software products like PGP (c.f. <http://www.pgpi.com>) and RSA SecurPC (c.f. <http://www.rsa.com>) belong to this class of encryption technology.

2.2.2 Cryptographic File Systems for Single Users.

These systems are characterized by an encryption service which is fully integrated into the file system management. But they are restricted to a single-user usage without a support for group-based access control. The keys needed to encrypt or decrypt files are usually derived from the user password. We distinguish between three different technologies, namely systems performing encryption directly at the hardware interface, systems using container files, and finally, systems which use a traditional file system as a repository.

If files are encrypted at the hardware interface then data blocks are automatically encrypted and decrypted before they are written to or read from the hard disk. As a consequence, no cleartext is ever stored on hard disk and the encryption is applied to the meta-data as well. But notice that all files are encrypted in a uniform way. Different security levels or group-based privileges are not supported. Examples for this class of systems are, for instance, Secure File System (c.f. <http://www.cs.auckland.ac.nz/~pgut001/sfs/index.html>)

and Practical Privacy Disc Driver (c.f. <http://pweb.de.uu.net/flexsys.mtk/ppdd-0.7.zip>).

If a container file is deployed to store an encrypted file system together with its encrypted meta-data then this container is considered as a unit which can be copied and/or moved. Several container files may exist simultaneously. Therefore, users are able to encrypt different file systems with different keys. But notice that the container is always encrypted as a whole. This all-or-nothing encryption policy is very coarse-grained and prevents from establishing customized security levels for single files. A container concept is deployed, for example, in SafeHouse (c.f. <http://www.pcdynamics.com/SafeHouse/>) and ScramDisc (c.f. <http://www.hertreg.ac.uk/ss/>).

If an encrypted directory tree is mapped to a sub-directory of an existing file system called a repository then meta-data are not concealed, because the repository concept retains the directory structure and file lengths. In addition, these systems do not support group-based accesses. A repository approach has been taken in the Cryptographic File System (c.f. <http://www.replay.com/redhat/cfs.html>) as well as in the Transparent CFS (c.f. <http://tcfs.dia.unisa.it/>).

2.2.3 Cryptographic File Systems for Multiple Users.

These approaches integrate the encryption and decryption capabilities into the file system as well and offer means for group accesses to shared files. Unfortunately, we have found just one relevant system belonging to this class. It is Microsoft's Encrypting File System [2, 1] (EFS) announced for Windows 2000. The concepts of EFS, though not yet usable, seem to be quite attractive but this attractiveness is reduced by the facts that dynamically changing user groups are not well supported, a proprietary public key infrastructure (PKI) is required and that meta-data is not encrypted.

Our analysis revealed severe deficiencies of popular cryptographic file systems. Substituting a conventional file system by one of those cryptographic file systems will result in reduced file system functionality together with an increased management overhead burdened on the user. Especially, an efficient and automatic multi-user capability which supports group-based accesses is not yet available.

3. BASIC CONCEPTS OF GSFS

We have developed a new cryptographic file system called GSFS to overcome some of the above mentioned shortcomings. GSFS (Group-Aware Secure File System) aims to be a multi-user system and is conceptually related to EFS. Its architecture is similar to UNIX-style file systems and directories are treated as

files. Since directories maintain meta-data file encryption under GSFS includes the concealing of meta-data information.

3.1. GROUP-AWARENESS

The integration of a transparent and efficient multi-user capability was one of the main goals in the design of GSFS. We aimed at supporting group-based access controls with fine-grained user groups and dynamically changing group membership.

3.1.1 Access Rights and Group Concept.

Under GSFS every file has a dedicated owner who may grant and revoke access privileges for his file according to a discretionary access control policy. As usually, we have *read*, *write* and *execute* access rights for files whereas directories may be *browsed*, *crossed* and *modified*. The access permissions for a file are stored in its associated access control list (ACL). An ACL-entry identifies either a single user or a user group and enumerates the access privileges which have been granted to users and groups (see figure 1). Under GSFS users are able to create user groups dynamically according to their individual needs whereas in traditional operating systems group management is the exclusive task of the system administrator.

Each group has a group chair. Only the chair is allowed to dynamically change group membership. Whenever a user creates a new group he automatically becomes the chair of the group. We maintain a group database (GDB) which contains information about all currently existing groups as well as their current members and the group chair (see figure 1).

3.1.2 File Encryption.

Like EFS, GSFS deploys symmetric as well as asymmetric encryption. Presently, GSFS only uses Twofish [4] with up to 256-bit keys and RSA but other encryption algorithms can easily be integrated.

GSFS generates a pair of keys comprising a public key UEK_U (= user encryption key) and a private key UDK_U (= user decryption key) for every user U . All public keys are managed in a local database PKDB (= public key database). This database is protected against unauthorized modifications. For every authorized user GSFS employs a personalized, PIN-protected smartcard to store the private user key. As the smartcard is considered to be a tamper-resistant device the private keys which are needed for GSFS-file system access are stored securely. Furthermore, smartcards and private keys can be used for authentication purposes as well.

GSFS uses symmetric cryptography to encrypt files. Each time an encryption key FK is required GSFS automatically generates one. Key generation and key management is performed fully transparent by GSFS. Hence, a user is

ACL for file f

U	Alice	rw	$E(\text{UEK}_{\text{Alice}}, \text{FK})$
U	Bob	r	$E(\text{UEK}_{\text{Bob}}, \text{FK})$
G	Project	rw	$E(\text{GEK}_{\text{Project}}, \text{FK})$

GDB Entries

<i>Project</i>	Chair: <i>Alice</i>
Cindy	$E(\text{UEK}_{\text{Cindy}}, \text{GDK}_{\text{Project}})$
David	$E(\text{UEK}_{\text{David}}, \text{GDK}_{\text{Project}})$
Alice	$E(\text{UEK}_{\text{Alice}}, \text{GDK}_{\text{Project}})$

PKDB Entries

U	Alice	$\text{UEK}_{\text{Alice}}$
U	Bob	UEK_{Bob}
U	Cindy	$\text{UEK}_{\text{Cindy}}$
U	David	$\text{UEK}_{\text{David}}$
G	Project	$\text{GEK}_{\text{Project}}$

Figure 1 GSFS data structures

not bothered with memorizing and securely managing a lot of different keys. GSFS encrypts the file key FK with the public key UEK of every user who possesses access rights for the file encrypted with FK. The encrypted file keys are automatically stored in the file's ACL (see the ACL for a file f in figure 1). Notice, that a key FK is never stored unencrypted on a storage medium.

3.1.3 Group-Keys.

Our flexible group capability enhancing the proposed EFS facilities is based upon key pairs $(\text{GEK}_G, \text{GDK}_G)$ generated for each group G . The public keys GEK are managed in the database PKDB, too. A file key FK can now be encrypted with all those public keys GEK which belong to groups appearing in the file's ACL. Hence, public keys of groups and such of users are handled in exactly the same way enabling a uniform management. Unfortunately, this does not work for the private keys. This is, because private user keys are stored on smartcards which are not available for groups. As GSFS requires the private group key whenever an authorized group member wants to access an encrypted file, these keys must be stored securely in the GSFS-system. To accomplish this, for each group member A a copy of the private group key encrypted with the public key UEK_A of A is stored in the group database GDB (see GDB entries in figure 1).

3.1.4 Data Structures.

Figure 1 depicts a simplified excerpt of the data structures transparently maintained by GSFS. $E(K, M)$ denotes the encryption of a string M with key K . The ACL of the example file f contains entries for users Alice and Bob as well as for a specific group named Project. An ACL entry for file f includes the encrypted file encryption key FK used to encrypt file f . Public keys are main-

tained in the public key database PKDB which is used by GSFS to perform the necessary encryptions. The group database GDB provides information about existing groups. In our example, the GDB contains just one group, namely the group *Project*. Alice is the chair person of group *Project*. A GDB entry enumerates all group members, i.e. Cindy, David and Alice, and contains the private group key $\text{GDK}_{\text{Project}}$ encrypted under the public keys of the group members, e.g. $E(\text{UEK}_{\text{Cindy}}, \text{GDK}_{\text{Project}})$.

3.1.5 File Access.

Consider a user Alice who wants to access a file f being encrypted with file key FK. First, the private key $\text{UDK}_{\text{Alice}}$ of Alice must be read from her smartcard. If Alice has been directly granted access privileges for file f then there is an entry in f 's ACL comprising the file key FK encrypted with $\text{UEK}_{\text{Alice}}$ (see figure 1). GSFS uses the private key $\text{UDK}_{\text{Alice}}$ to decrypt FK and is able to decrypt f as well using FK. If Alice has been granted access rights via one of her groups, say the group *Project*, then GSFS uses the private key $\text{UDK}_{\text{Alice}}$ to decrypt the private group key $\text{GDK}_{\text{Project}}$ stored in the GDB. Since there exists an entry in f 's ACL for group *Project* which contains the file key FK encrypted with $\text{GEK}_{\text{Project}}$ GSFS is now enabled to decrypt key FK. Hence, file decryption in cases of a file access does not require any other user intervention than providing an appropriate smartcard and activating it with the correct PIN.

3.1.6 Authentic Public Keys.

The secure and authentic management of public keys in the database PKDB is a security critical task. We rejected to utilize a trusted and protected external server machine, because it requires an online connection between client and server whenever a client has to look up public keys. This is inappropriate especially for mobile devices which often work off-line but urgently require file system encryption. Therefore, GSFS implements the database PKDB locally which ensures a permanent availability of public keys. Key authentication is certified by a trusted third party, namely the file system administrator who signs the key with his private signature key. Hence, his public key is required for signature verification. We decided to store this public key on the smartcards of the users to defeat cheating. But we are aware that changing the verification key requires to change all smartcards of authorized users as well. Future work aims at providing improvements to our present solution.

3.2. OTHER FEATURES

To support different security profiles a user is enabled to select a security level (e.g. encrypt or cleartext) for each file or directory he owns. These specified security attributes trigger the management of GSFS to apply appropriate encryption measures adjusted to the application-specific needs.

To economize resource utilization, GSFS just backups those files which have been modified. Since the public user keys are employed to encrypt file keys stored in backed up ACLs, a key exchange affects backed up data as well. GSFS handles the necessary management activities without any user intervention. To this end, we encrypt and store the out-of-date private user keys UDK in an archive file whenever a key change occurs. In this way, file keys encrypted with old public keys are still available whenever an access to backed up and encrypted data is required.

Key recovery can be initiated via a recovery option. To enable key recovery GSFS encrypts individual file keys with a special recovery key and stores the encrypted key in the file's ACL. To restrict potential key abuse we use a shared secret schema to split the private key which belongs to the recovery key. The parts of the private recovery key are securely managed by several trusted users. In this way we ensure that compromising one part of the recovery key does not compromise the whole key.

4. PROTOTYPE

Our prototype has been implemented integrated into the Linux operating system. Figure 2 depicts the overall architecture of our prototype. The file system providing the security services runs in kernel mode whereas the modules implemented in user mode mainly offer a user interface.

4.1. USER-MODE MODULES

Among other services the *Login Service* initiates the transfer of private user keys stored on the smartcard. The *Object Service* offers means to list all file attributes including GSFS-extensions and to edit ACLs. The *Key Service* provides interfaces to generate new public key pairs for users and offers services for changing keys and deleting accounts. Via the *Group Service* interface kernel-mode commands can be invoked to create/delete user groups and to manage group memberships.

Notice, that the user-mode modules do not perform any security critical services but only make kernel-mode services available at the user interface. As a consequence, the security of GSFS solely depends on the strength of the encryption algorithms employed and on the trustworthiness of the kernel-mode security modules.

4.2. KERNEL-MODE MODULES

GSFS is integrated into the Linux virtual file system via the *VFS-Interface* which forwards file system calls to the *File System Manager* for further processing. The *File System Manager* is the central part of the GSFS file system. It transforms management orders into GSFS commands and performs file en-

Figure 2 The architecture of GSFS

cryption and decryption. The available cryptographic algorithms are summed up in the *Cryptographic* module including a random number generator as well as services for generating and validating digital signatures.

The *Key Management* provides the *File System Manager* as well as the *Access Control* module with the required cryptographic keys. Furthermore, it maintains the public key database PKDB as well as the archive of out-of-date user keys.

File ACLs are managed by the *Access Control* module which, in addition, supervises all accesses to file objects. If an access is authorized the module provides the *File System Manager* with the required file encryption key by interacting with the *Key Management* module to obtain the necessary private user key UDK. If the control module needs the private group key GDK to decrypt the file key it interacts with the *Group Management* module which manages the group database GDB containing the encrypted group keys.

Files are stored on hard disk using the *Repository Subsystem*. The repository is mostly independent from the underlying storage device. It only assumes a block-oriented structure and random access facilities. Presently, the GSFS

prototype uses a file system to store encrypted data rather than using a device driver directly.

The prototype is available as a special Linux file system. All the services described above are running but there are still some restrictions. In future work we will replace the intermediate repository file system with direct device driver access and we will fully integrate the authentication service in the underlying operating system.

5. CONCLUSION

Hand in hand with the rapid developments in the field of open and mobile computing the amount of sensitive data stored in these systems steadily increases. This data is exposed to lots of hostile attacks which can no longer be repelled by traditional access control mechanisms of an operating system. Cryptographic file systems aim at closing this security gap. Such a file system should not restrict the functionality of a traditional file system. Especially, the group-based access capabilities must be retained. Furthermore, a cryptographic file system should not burden the user with additional management overhead.

We have analyzed existing file systems with respect to these requirements. The analysis revealed severe deficiencies particularly concerning group-awareness. With the development of GSFS we tried to overcome some of these shortcomings. We focused on the description of GSFS's flexible management of access privileges with a group facility which can be used to grant fine-grained access privileges to shared encrypted files. File encryption and decryption as well as management of the required keys is done in a fully transparent manner by GSFS. A prototype implementation is available on Linux platforms.

References

- [1] Microsoft. *Encrypting File System for Windows 2000*. <http://www.microsoft.com/Windows/server/zipdocs/encrypt.exe>, 1999.
- [2] Microsoft Co., Developer Network CD Juli 1998, mk:@ivt:conf/pdc97/encrypting_file_system.htm. *Encrypting File System for Windows NT Version 5.0*.
- [3] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.
- [4] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. *The Twofish Encryption Algorithm*. John Wiley & Sons., 1999.
- [5] J.H. Saltzer and M.D. Schroeder. Protection of Information in Computer Systems. In *Proceedings of the IEEE*, 63(9), pages 1278 – 1308, March 1975.