INTEGRATED FILE-LEVEL CRYPTOGRAPHICAL ACCESS CONTROL

Ryan Seifert
ryanseifert@earthlink.net

T. Andrew Yang
Yang@UHCL.edu

Division of Computing and Mathematics
University of Houston - Clear Lake, Houston, Texas

Abstract

In this paper, we propose the design of the Integrated File Level Cryptographical Access Control (IFLCAC) system, which makes file security much easier for the end user to utilize. The proposed system combines the benefits of the traditional file-level cryptography and the full-disk cryptography system, making it both secure and easy to use. We first investigate the existing file cryptography systems, comparing their pros and cons, and then describe the components of the Integrated File Level Cryptographical Access Control system, and interactions among the components. We also discuss in brief how the system may be used in classroom teaching.

1. Introduction

File level cryptographical access control is important to guarantee that sensitive data are stored and accessed securely. File cryptography systems today follow three basic designs, file level cryptography, file system cryptography, and virtual partition cryptography. Each design brings its own strengths and weaknesses when implemented.

File level cryptography is the traditional method. This design offers unparalleled control over which files are encrypted; each file is manually encrypted and decrypted by the user. Unfortunately, this method is also the most difficult for the end user to utilize.

There are many file level cryptography applications in existence today. One such application is AxCrypt for windows [1]. AxCrypt allows the user to encrypt files using the AES 128 bit standard encryption algorithm. While AxCrypt provides security on any file the user selects, the application still forces the user to navigate to the file and manually encrypt or decrypt the file.

File system cryptography secures the entire file system. It utilizes a special file system that encrypts all data going to the file system, and decrypts all data coming from the file system [4]. All files written to the disk are stored securely. This design is the easiest for the user to utilize, but consumes the most overhead. Besides, this method is file system dependent, meaning that when the user changes its file system (say, from FAT32 to NTFS), the whole file system will need to be recreated.

PGP Whole Disk Encryption [6] is one such file system cryptography implementation. The application has the option to encrypt the entire contents of a drive connected to the computer, including the boot sector and swap files.

Virtual partition cryptography attempts to find a balance between file level and file system cryptography. A separate partition of the disk is created to store sensitive files that need to be stored securely, while non-sensitive files are stored in the regular partition. While allowing more granularity than file system cryptography, this design forces all secure files into a central location. Virtual partition cryptography allocates the necessary space for the partition upon configuring the encryption system. The user must know the maximum amount of necessary space to store all

users' sensitive files. Many times this information is unknown at the time of configuring, forcing the user to recreate the virtual partition periodically. Creating the virtual partition is a costly operation; the system must decrypt each layer of the existing partition and encrypt it into the new partition. Additionally, because the space is allocated upon creation and recreating is a costly operation (and tends to be avoided when possible), typically a large chunk of disk space is allocated upon creation of the partition, resulting in possible waste of hard drive space in the system.

TrueCrypt [7] is a popular open source virtual partition encryption application. It has all features of virtual partition cryptography and utilizes symmetric key algorithms such as AES, Serpent and Twofish.

The above discussed designs can be utilized in conjunction to provide a more secure system. File system and file level cryptography are utilized to provide the most secure systems today. The goal of our work is to develop a file level secure access control system that is secure, easy to use, and flexible to configure.

In the rest of this paper, we first define the problem of designing a secure file level cryptography system. Our proposed design is discussed in section 3. In section 4 we briefly discuss how the file level cryptography system may be used to enhance teaching of various courses. Section 5 concludes the paper.

2. Problem Definition

File system and file level cryptography, when used in conjunction, create an extremely secure system. Such design allows for all files to be secured, with an additional layer of security over highly sensitive files. The overhead of file system cryptography is offset by the ever increasing speeds of personal computing, and the lack of granularity is remedied by the utilization of file level cryptography. Unfortunately this common solution leaves the user in much the same situation as before. That is, the user must still manually encrypt and decrypt the highly sensitive files on the system.

This overhead necessarily incurred by the user introduces a dangerous element to the encryption scheme, the user. It is all too easy for the user to forget to encrypt the file after accessing it. Additionally this user overhead will limit the total number of files deemed necessary for the additional encryption; this limit changes on a per user basis. The limit is simply the amount of overhead the user is willing to incur. In order to ensure all necessary files are secured properly, this user element in the system needs to be minimized as much as possible.

A user friendly, efficient, and secure file level cryptography method is needed to fill the void left in file cryptography. The new method must meet certain criteria to completely fill the void:

(a) The method must be *secure*; the method should not introduce a new weak link into a secure system.

(b) The method must be *intuitive* for the end user to operate, and simple enough for the non-computer literate users to operate and maintain.

(c) The method must be *file system independent*; this will ensure portability among all computers in the system. (Note: Being *file-system independent* allows something to be implemented without regard to the file system, e.g., the program does not care if it is running on FAT16, FAT32, NTFS, etc. This is very different than *platform independence*, which means the program can be run on different operating systems, such

as Windows, Solaris, etc.). Platform independence is virtually impossible to achieve for file level cryptography systems.)

(d) The method must be *easy to update*; this will ensure the method can evolve with the changing state of security.

3. Proposed Solution

Our proposed solution in response to the above criteria is discussed in this section. We first present the structure and the various components of the new system, and then in section 3.2 sample interactions among the components are discussed to illustrate how a typical operation is carried out in the new system.

3.1. Integrated File Level Secure Access Control

As shown in Figure 1, the design of the new file level cryptography method consists of four main parts, the file system gateway, encryption database, user program, and administrator program. The file system gateway sits just above the file system, intercepting calls made to the file system. The encryption database stores necessary information regarding the encrypted files. The user program provides a simple interface for the end user. The administrator program provides access to the methods configuration options. All encryption and hash algorithms are stored in dynamic link libraries exporting a pre-defined interface. Storing the encryption and hash algorithms in this way allows for easy insertion and updating of the algorithms.

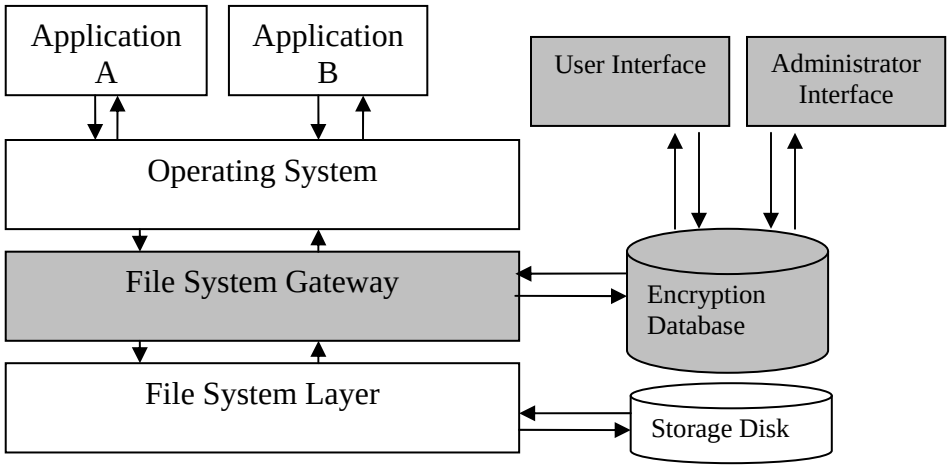


Figure 1 – The Integrated File Level Secure Access Control

The *file system gateway* is the most critical element of the new cryptography method. This element is where all data encryption and decryption take place. The gateway controls all communication between the upper application layers and the underlying file system. The database holds all the configuration options for the file system gateway element. The gateway handles all file request calls from applications, querying the database to check if the file is stored securely. The unique name from the file is used as an index into the encryption database. If cryptographical operations are required, the proper dynamic link library is loaded. The cryptographical operations are performed and the resulting data is sent back to the requesting application.

An important aspect of the gateway needs to be pointed out. The gateway is implemented as a layer above the file system; as such it must remain completely independent of the file system. This includes depending on any given file system's unique file pointer information. Additionally

in order to implement some special functionality relating to parsing the unique names for a given file system, an additional dynamic link library can be used. The only location for file system dependence is within this library.

The gateway calculates the encryption key based on the user's password upon logging into the system. Additional passwords to open files encrypted with a different key must be retrieved directly from the user. This adds the requirement that the gateway resides on the client machine if accessing a distributed file system. Performing the cryptography on the user machine allows for a minimum trust relationship between the local computer and the remote file server [2], as well as removing the necessity for asymmetric key cryptography to be preformed before communication between the remote file server and the local computer [5].

The *encryption database* is used to store all information relating to the encrypted files. Furthermore it is also used to store configuration options for the other elements. The database needs to be designed to be accessed efficiently, as every call to open a file queries the database. The encryption database consists of two major elements, the file system database and the configuration options. The file system database stores information about which files are encrypted and which algorithm was used to encrypt the file. The other major element of the database is the configuration options; this element holds information on the available dynamic link libraries, current file system in use, and other miscellaneous general options such as password rules and administrative settings.

The encryption database holds all necessary information to allow correct access to the file system. Unfortunately this introduces a single point of failure into the integrated file level encryption method. At worst if the database fails, the administrator program can be accessed to decrypt files manually. If communication with the encryption database is unavailable, the gateway will enter an invisible state, simply returning the data directly as it resides on the disk. This state will allow normal operations on non-encrypted files, while encrypted files will be accessed via the administrator program.

The *user program* allows the end user to interact with the integrated file level encryption system. The program displays paths to encrypted files, and the files modifiers. The program will also allow the user to add a file to the encryption list. The user program also allows basic maintenance for the user, such as changing their password. For most users this program will be the only interface into the system.

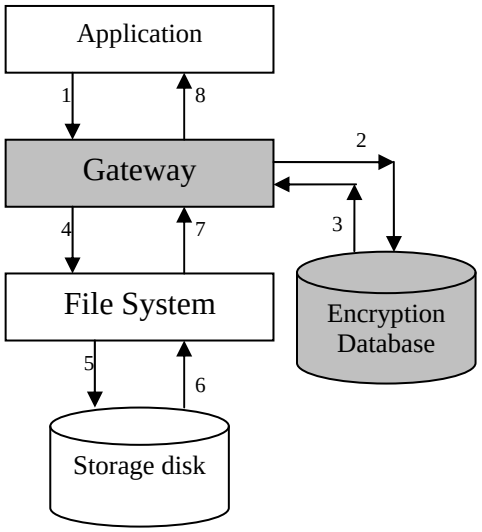
The *administrator program* allows the configuration of the system and advanced access to the database. The program also allows files to be manually encrypted and decrypted; this tool is not available in the user program. The program consists of two interfaces, the system configuration interface and the encrypted file viewer. This program allows the administrator to perform general configuration and maintenance to the system, such as adding new encryption algorithm dynamic link library and configuring group access files. The encrypted file view creates a list of all encrypted files. Through this interface the administrator can select and manually encrypt and decrypt available files. Other modifiers can be updated on the files such as the encrypted algorithm used and which users have access to the files.

3.2. Element Interactions

The four elements incorporate together to form the new cryptography method. Both the file system gateway and the encryption database must be running continuously; both must be executing for a user to retrieve an encrypted file in the system. The other two elements, the user

program and the administrative program, need not be executed unless the current system configuration needs to be changed.

The system can also be utilized to ensure file integrity. A message digest of any file can be computed and stored in the encryption database. When the file is accessed, a new message digest is computed and compared to the stored value. If the message digests are not equal, the file has been modified outside the integrated file level cryptography system and the user is informed.



Steps:

1) Application makes a file request
2) Gateway queries database.
3) Gateway performs necessary crypto graphical functions.
3) Database returns file information.
4) File request continues to File System.
5) File system handles request.
6) File system returns data to Gateway.
7) Gateway makes necessary crypto graphical functions.
8) Decrypted information is returned to the calling application.

Figure 2 – Normal Operation of Retrieving an Encrypted File

The most common interaction will be a simple single user decryption for a file being opened. As illustrated in Figure 2, the file's unique name is searched though the database of modified files (step 2), and if located the information is returned to the gateway (step 3). The gateway loads the encryption algorithm and, if necessary, the message digest algorithm from the appropriate dynamic link libraries. The file stream is opened via the normal file layer calls (step 4). All the necessary data is logged into a memory list to support multiple files being opened simultaneously. The open file pointer is then returned to the calling application (step 7). For a file read request, the open file is only decrypted enough to fill the request from the calling application. File writes are slightly more complicated depending on the encryption algorithm used. Many symmetric key algorithms encrypt data in blocks of a set size. Before the data can be formally written to the disk, either a full block or a file close is required. Finally closing the file completes any outstanding data encryption. The single user file access needs to utilize only the symmetric key algorithms.

The integrated file level cryptography method allows for easy modification to the encryption algorithms used in the system. New encryption algorithms can be inserted though the dynamic link library system as stronger and more efficient algorithms are published. When updating to new encryption algorithms, all the currently encrypted files must be converted. This is an unavoidable and costly operation. The encryption database can be used to perform many of these operations in a just-in-time style. Upon accessing the out-of-date file, the original encryption scheme is used to decrypt the file while the new encryption scheme is used to re-encrypt the file. If the old encryption algorithm was compromised, the administrator has the option to convert all files immediately.

For files with multiple user access and user/administrator access a key escrow system must be implemented. When a file is flagged for multiple user access a random symmetric key is

generated for the file. A copy of the symmetric key is encrypted with each user's public asymmetric key [3]. The symmetric key is now secured by the asymmetric encryption algorithm currently in use. The encrypted key is then stored in the database for later retrieval and decryption by the user(s). This method of key escrow allows for easy updating of group access files by the administrator and keeps the encryption of the file using the stronger and more efficient symmetric key encryption.

4. Applications to Classroom Teaching

The IFLCAC system may be adopted in teaching various courses, such as Operating Systems, Data Bases, Advanced Operating Systems, Computer Security, and Capstone Projects. Many possibilities exist. In this section we explore some of the approaches. In the Advanced Operating Systems or the Capstone Projects courses, for example, the students may be asked to create the detailed design of the IFLCAC system, followed by the implementation and evaluation of the system. One way to use the system in a Computer Security class is to provide the whole developed system to the students, while asking them to implement and evaluate different cryptographical algorithms (DES, AES, Blowfish, etc.) in the system. For the beginning Operating Systems course, the students may be required to implement the File System Gateway component, while the other three components (Encryption Database, User Interface, and Administrator Interface) are provided by the instructor. On the other hand, in a Data Bases course, the instructor may provide the students the File System Gateway and the User Interface components, while requiring the students to design and implement the Encryption Database and the Administrator Interface components. Depending on the instructor's discretion, other combinations of available components versus to-be-developed components are possible.

5. Conclusion

A new file level cryptography method is needed, one that gives the security and flexibility of file level encryption with the ease of use of file system encryption. A new solution is proposed in this paper. The new method finds a medium ground between file level encryption and file system encryption. It successfully merges the strengths of the two systems, providing an easy to use, secure, and efficient file cryptography method. With this new component, a fully integrated system can be created that is simple to operate and has varying strengths of security on a per-file and overall disk status.

References

[1] AxCrypt. 2007. Axantum Software AB. October 31, 2007 http://www.axantum.com/AxCrypt/

[2] Kher and Kim, "Securing Distributed Storage: Challenges, Techniques, and Systems". *Proceedings of Workshop on Storage Security and Survivability.* ACM Press, 2005.

[3] Lee, Boyd, Dawson, Kim, Yang, and Yoo "Secure Key Issuing in ID-based Cryptography". Australasian Information Security Workshop. Australian Computer Society, 2004.

[4] Ludwig and Kalfa, "File System Encryption with Integrated User Management". *ACM SIGOPS Operating Systems Review*. ACM Press, 2001.

[5] Naor, Shenhav, and Wool "Toward Securing Untrusted Storage without Public-Key Operations". Workshop on Storage Security and Survivability. ACM 2005.

[6] PGP Whole Disk Encryption. 2007. PGP Corporation. October 31, 2007 http://www.pgp.com/products/wholediskencryption/

[7] TrueCrypt. October 28, 2007. TrueCrypt Foundation. October 31, 2007 http://www.truecrypt.org/