PCFS: Portable EnCrypted File System

6.858 Final Project Report

Xinyue Ye heyitsye@mit.edu
Maosen Hu mjhu@mit.edu
Jiashan Liang jliang3@mit.edu

December 13, 2013

Abstract

PCFS is a portable encrypted file system built on top of the Unix file system and is compatible with the Linux operating system. While there are many encrypted file systems built for sharing files across a network of users, there are not many systems that allow users to easily encrypt and share files from any portable device such as a USB. Our project seeks to build a quick portable plug-and-use file system with a Python shell user interface to allow such encryption and sharing.

1 Introduction

PCFS supports the same basic set of file I/O functionalities as any filesystem. Users can create, read, write, and delete files and directories. They can also set permissions and share files and directories with other users. To provide confidentiality, PCFS encrypts all filenames, directory names, and file content so they cannot be seen by unwanted eyes. Additionally, any unauthorized changes to files or directories will be detected by PCFS and reported to authorized users of the modified files and directories.

An advantage of PCFS is that it is not tied to any single file server, so clients can use the encrypted file system with any untrusted file server, including the local file server on their machines. It can also be used in conjunction with other file systems and easily integrates into the Linux OS.

In the following sections, we discuss in depth the design of PCFS (§ 2), the implementation details (§ 3), and conclude with possible future improvements (§ 4).

2 Design

We present the design of PCFS and discuss the data structures used and overall control flow through the system.

2.1 File Data Structure

A regular file in a Unix file system is represented in PCFS slightly differently. There is some extra metadata that PCFS needs to keep track of, similar to the metadata the UnixFS needs to keep track of. Each file contains a checksum, a list of users with read and/or write permissions for that file, and the actual file contents that the user inputs (See Figure 1).

As shown in Figure 1, the header is used to signify that the file is an encrypted file that belongs to PCFS. For files that do not contain the header, PCFS simply reads the content in the file without any pre-processing. The checksum is used to detect any unauthorized modifications to the file. Whenever an authorized user opens a file for reading or writing, the checksum stored in the file is validated against a newly computed checksum of the file and the user will be notified if the validation fails. The permissions list is simply a representation of the users that have read and/or write permissions for that file and it is stored in the format:

```
username1:r, username2:rw
```

Figure 1: . File format data structure showing the header, checksum, permissions, and content for each file.

This additional metadata for each file allows PCFS to ensure the confidentiality and security of the system.

2.2 PCFS Encryption/ Decryption

When a user first logs onto the file system, a unique key is generated from their password using PBKDF2 which we will call K. When the user creates a file/directory, a new random key will be generated and stored in a master key file as shown in Figure 2b along with its checksum. This master key file will be encrypted with K and is hidden from all users.

For encryption, the entire file data is encrypted using AES-CBC and each file has its own file key, then the encryptions are encoded in hex. An untrusted file server will only see the encoded version of the filename, directory name, and filecontent, as shown in Figure 2a; unauthorized users will not even be able to see the files on PCFS. This provides confidentiality and security against malicious users or servers attempting to read the files.

Now when the user wants to open a file, assuming they have authorized access to the file, PCFS first gets the encrypted filename of the file and then uses the AES key for that file from the master key file to decrypt the entire file, including the permissions and checksum. If the user does not have authorized access to a file that they are trying to open, ie. the filename is not in the master key file, then PCFS will attempt to use a default key, which is simply 16 null bytes that will fail to decrypt anything.

3 Implementation

We discuss the implementation details of our file system by by breaking it into the milestones of the project.

3.1 Basic I/O Functionalities

Read/write functionalities of a file/directory depends on the users unique key since everything is stored as its AES encryption. If they are authenticated then everything will able to be decrypted, otherwise the files/directories will be decrypted using the wrong key and hence remain confidential.

Most of the basic I/O operations that can be performed on the Unix file system can be performed on PCFS, including ('ls', 'rm', 'mv', 'cat', 'vi', and 'mkdir'). These commands are executed by making their respective OS command calls after thorough processing by the PCFS.

For 'cat' and 'vi', these two commands are trying to gain read access to the file, so the checksum and permissions must be first verified, and then the decryption process takes place as described in § 2.2. After the entire file is decrypted, the user should only be able to see the file contents, so only the file contents are printed to the user or written to the file and all other metadata stay hidden.

In order to delete a file, the user must have authorized access to the file and have the file key before they can call 'rm' on the filename.

The other commands pertain more to checksum verification and directories so they will be discussed further in § 3.2 and § 3.4.

3.2 Checksum

Checksums are generated for each file and directory to ensure that unauthorized activity can at least be detected. For files, the checksum is a MD5 hash of the contents in the file. For directories, the checksum is a MD5 hash of the contents in the directory.

The command 'ls' on a directory will activate the checksum validation to check that no malicious activity occurred. When an unauthorized user or untrustworthy file server tries to

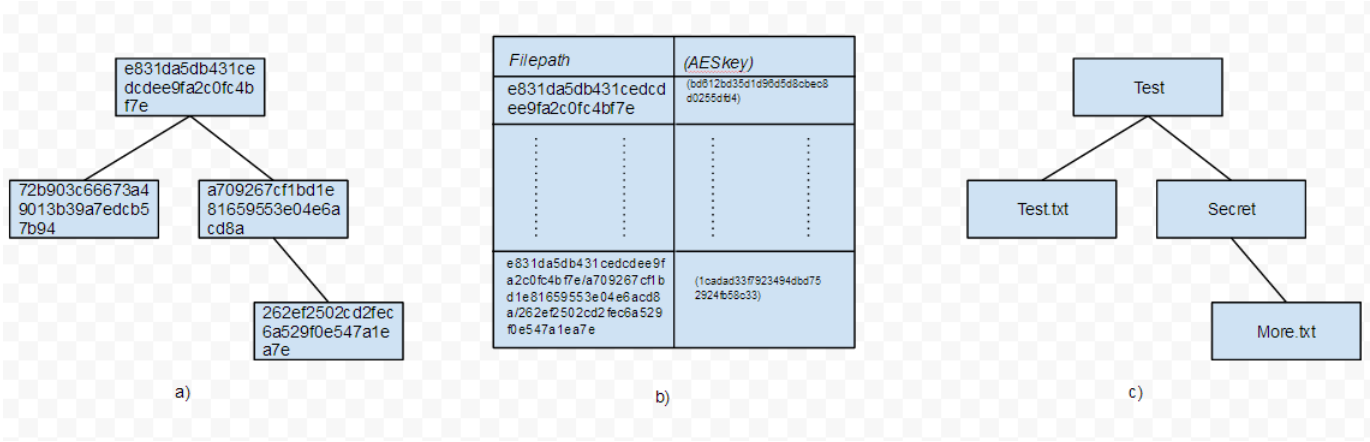

Figure 2: . PCFS Structure. **Figure 2a** shows the encoding that an untrusted file server who does not have the file or directory keys would see. **Figure 2b** shows the control flow of the file system. When a file needs to opened, PCFS will get the file key from the master file, which is encrypted with the user's key K. The master file is written to disk and is structured as a hashtable in memory that maps an encrypted filename to the file key. **Figure 2c** shows the successfully decrypted names of the directories and files

modify the structure of the directory, ie. by removing files, making new files, making new inner directories, or moving and renaming files, then the new recomputed checksum for that directory will not match the checksum stored in the directory originally and the user will be notified of this unauthorized activity.

For files, when a file is opened for reading or writing, such as in commands like 'vi' and 'cat', the checksum will be validated. The file will open successfully if the validation passes and fail otherwise and the user will be notified.

3.3 File Permissions and Sharing

There are three possible permissions mode, read (r), write (w), or both (rw). For users that have both read and write permissions, normal expected behaviour follows calling any command. If a user only has read permissions for a file but not write, then calling a command like 'vi' will open the file in read-only mode. If a user only has write permissions but not read permissions, then a command like 'vi' will display a message notifying the user that they have been denied read permissions so anything they write in the text editor will simply overwrite the existing file.

File sharing is possible through the 'share' command. The command takes in a filename, a username, and the allowed permissions.

```
share [filename] [username] [rw]
```

The 'share' command adds the new user and his/her permissions to the file's permissions and returns a file sharing code. For the new user, he/she must log in and call the 'unlock' command and paste the file sharing code as the argument to the command. Once the file is unlocked, then the file name is displayed in its plain text form and the new user can perform any action within his/her permissions boundary. Because each file has a unique key, sharing one file will not compromise any other file. With the added permissions component, even if an adversary obtains the key, without given the correct permissions, they will not be able to read or write to the file.

3.4 File Directories

File directories are treated similarly to files. They are encrypted and decrypted just as files are, however there are some differences.

The checksum for directories are different from those in files as described in § 3.2. Furthermore, although directories support permissions as well, users that have permissions for a directory are not guaranteed permissions for the files or other directories within the directory. Therefore, both the permissions for the directory and

the files within the directory are needed to read or write the files.

4 Conclusion

PCFS offers confidentiality and security benefits on top of the basic features that regular file systems provide. It not only satisfies all the minimum requirements outlined in the Lab 7 description that relate to our design, but is also portable and easily integrated into the Linux operating system.

However, there is still much to improve on for future work, not including code refactoring, testing, and evaluation. For instance, currently to share multiple files, the user needs to use the share command once for every file. For more robust sharing, it would be better to implement a command that can share multiple files to the same user. In addition, we would like to implement a command that will share one file to multiple users at the same time. Additionally, there are constraints as to what users can choose as valid names and new features and functions can always be added to make PCFS more user-friendly and convenient.

References

[1] J. Li, M. Krohn, D. Mazieres, and D. Shasha, Secure untrusted data repository (SUNDR)," In *OSDI 2004*.