

Cryptographic Filesystems, Part One: Design and Implementation

Ido Dubrawsky 2003-03-07

As security becomes a greater focus in networks, every aspect of online information needs a level of protection from the network-level use of firewalls and IDS to the host-level use of IDS. However, an additional level of security has recently come to the forefront of security - cryptographic filesystems. While the technology for cryptographic filesystems has been available for quite a while, the deployment of cryptographic filesystems in production environments has not taken hold. This article will discuss some of the background and technology of cryptographic filesystems and will then cover some example implementations of these filesystems including Microsoft's Encrypting File System for Windows 2000, the Linux CryptoAPI, and the Secure File System.

Cryptographic Filesystem Types

There are several approaches to cryptographic filesystems. These vary from volume encryptors to filesystem encryptors to file encryptors. Each of these approaches has its merits as well as its drawbacks and is discussed in more detail below.

Volume Encryptors

Volume encryptors use the device driver layer to encrypt and decrypt information to and from a physical disk. Such systems include systems as PGPDisk, the Secure File System (SFS), the Linux CryptoAPI and ScramDisk. Volume encryptors encrypt whole drives and are convenient to use since they are transparent to the end user. However, volume encryptors do not provide fine-grained access control to individual directories or files.

File Encryptors

File encryptors operate at the application or presentation layer to provide true end-to-end file encryption. In order to provide some sort of transparency to the end-user file, encryptors typically require some measure of application rewrite in order to support encryption. File encryptor systems include such tools as PGP. For small numbers of files these types of encryptors are adequate but they do not scale well to storage systems.

File System Encryptor

File system encryptors allow the encryption of files on a per-file or a per-directory basis using a single key. Systems such as these include the Cryptographic File System (CFS) developed by Matt Blaze, the Transparent Cryptographic File System (TCFS) supported under Linux and BSD, CryptFS, as well as Microsoft's EFS under Windows 2000.

Cryptographic Filesystem Design

CryptFS

One implementation of a cryptographic filesystem is through the use of a kernel-resident filesystem. This implementation model is used in [CryptFS](#). Using this implementation model, the file system can be mounted on any directory as well as on top of another file system such as UFS or NFS[1]. This model also removes the need for additional daemon processes that can possibly be exploited to gain access to the system or possibly to the files. The interface used by CryptFS is through a stackable V-node (Virtual Node). Unix-based operating systems use vnodes to represent an open file, directory, device, or other objects. The vnodes do not expose what type of physical file system they implement. CryptFS uses a concept called V-node stacking, which allows for filesystem function modularization where one V-node interface calls another. V-node stacking is shown in Figure 1 below.

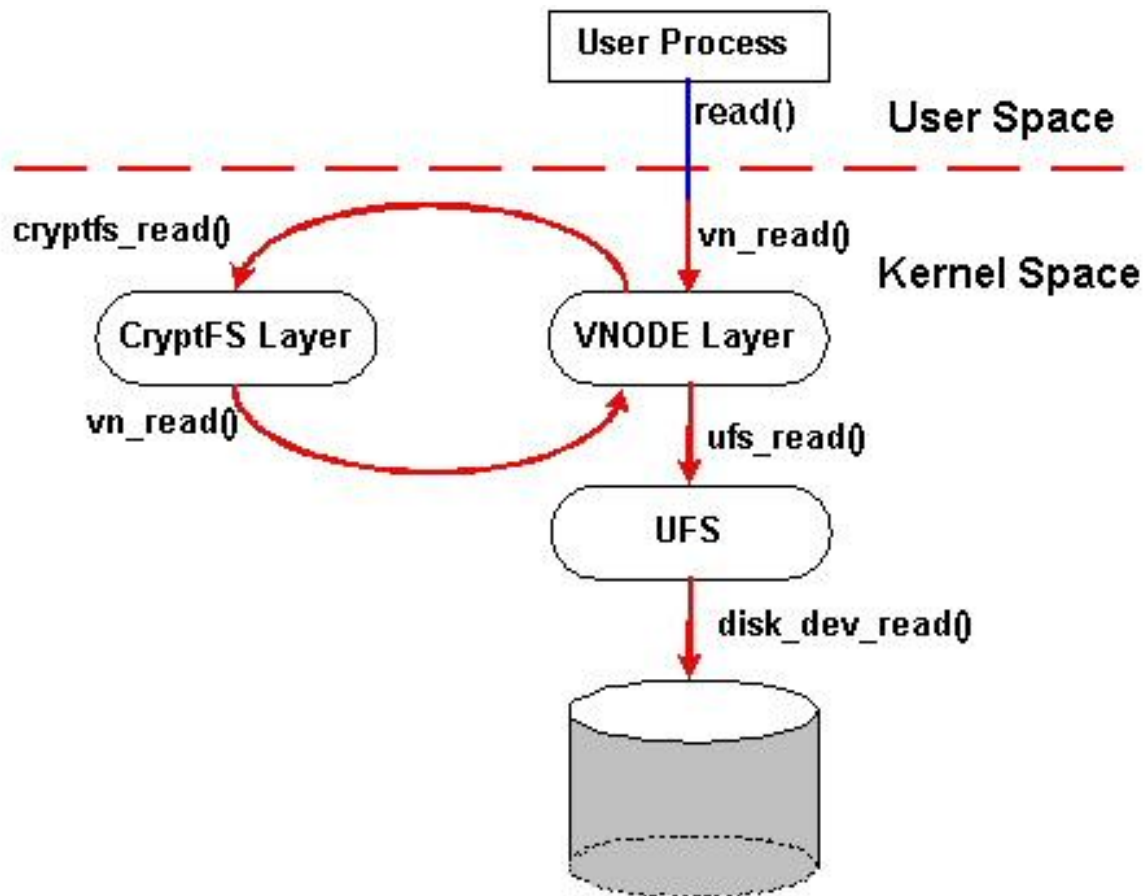


Figure 1: Stackable V-node Interface Design

Each V-node operation in CryptFS calls its next-lower layer function for filesystem specific operations. Originally developed for Solaris, CryptFS has also been ported to Linux and FreeBSD. CryptFS inserts itself on top of any directory, encrypts file data before it is passed to the file system, and decrypts it in the reverse direction. The designers of CryptFS also provide for a key management scheme where only the root user can actually mount an instance of CryptFS and user keys would be associated not only with a UID but also a session ID. An attacker would not only have to break into an account but also have his processes use the same session ID as the users process in order to acquire or change a user's key. CryptFS uses blowfish in CBC mode as its encryption algorithm.

The CryptFS is available to the general public as part of the File Systems Software Package (fstgen). It can be found [here](#).

Linux CryptoAPI

While the Linux CryptoAPI is not, strictly speaking, a filesystem encryption system, it does provide for that capability. The original kernelint patch was designed to provide filesystem

encryption capabilities in the 2.2 series kernels. The development of the kernelint patch into a general purpose API for kernel-space encryption represents the next logical evolutionary step. The loopback device allows for a level of indirection when mounting a filesystem whereby system calls can be intercepted to provide encryption and decryption of filesystem data. Instead of mounting the filesystem directly on a given directory the filesystem can be mounted on the loopback device. The loopback device is then mounted on the directory mount point. This configuration has the effect of sending all kernel commands to the filesystem through the loopback device. The process is shown in Figures 2 and 3.

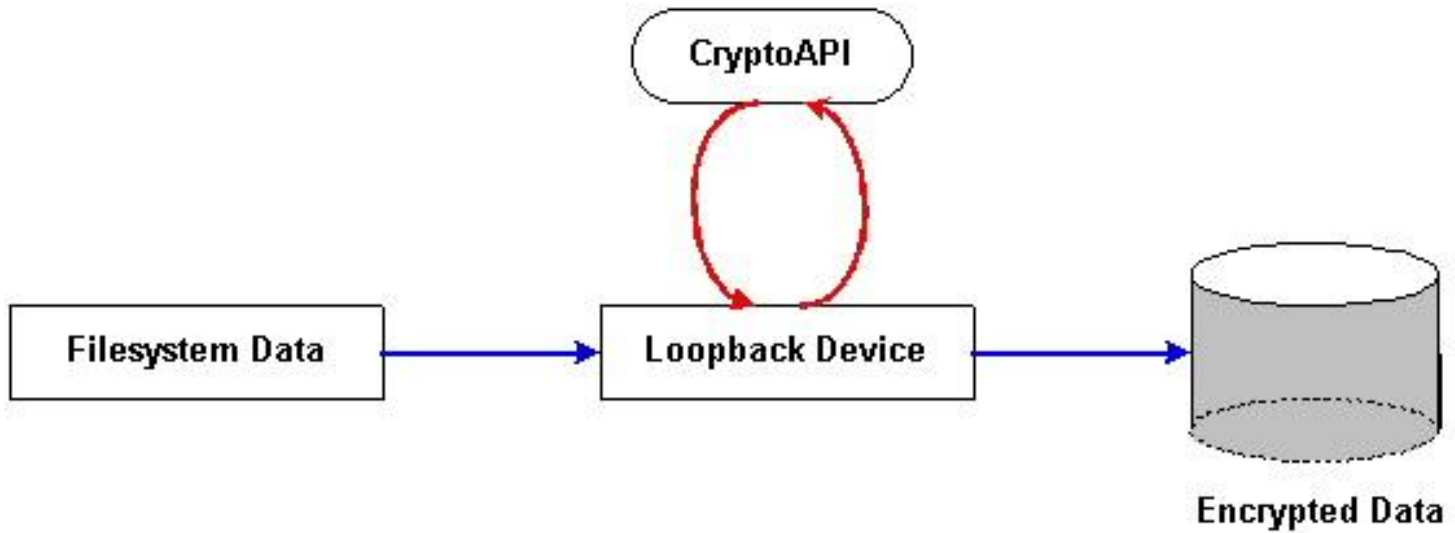


Figure 2: CryptoAPI Encryption

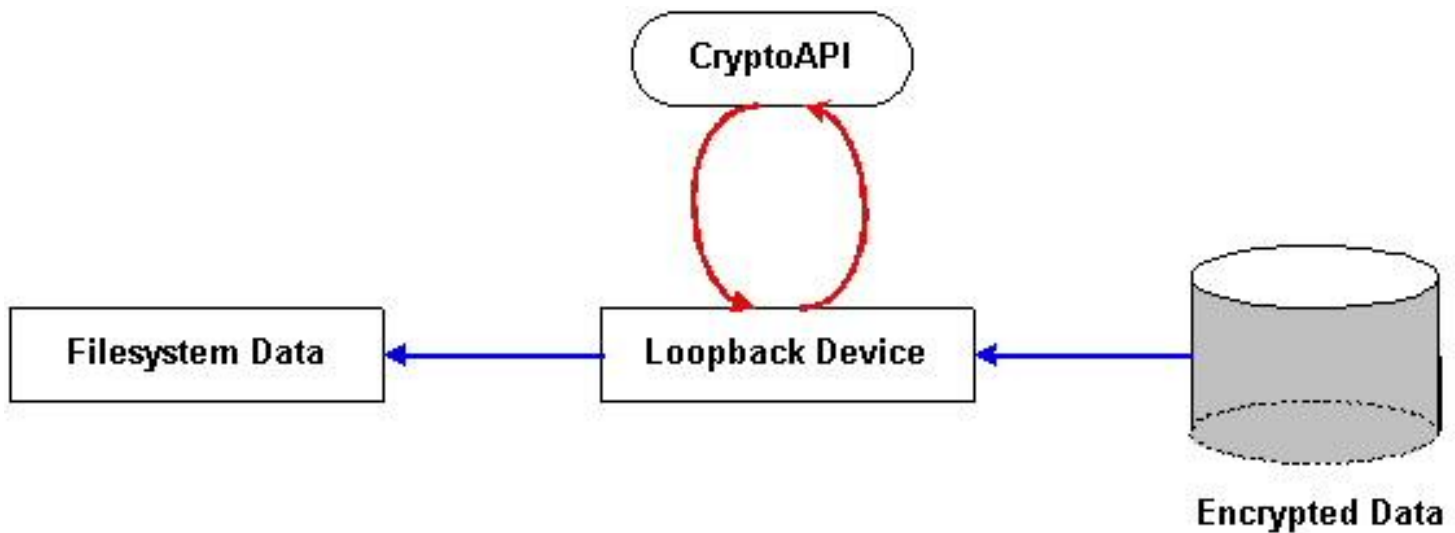


Figure 3: CryptoAPI Decryption

The CryptoAPI software can be downloaded from the [GNU/Linux CryptoAPI site](#).

Microsoft's Encrypted File System (EFS)

Microsoft's EFS is implemented using a public key-based scheme. File data is encrypted using a fast symmetric algorithm with a file encryption key that is randomly generated. This key is encrypted itself using one or more public keys obtained from a user's X.509 version 3 certificate. The private portion of the private-public key is used to decrypt the file encryption key, which is then used to decrypt the file. Microsoft's EFS does not support using a symmetric algorithm using a password-based key because of the concern that these password based schemes are weaker due to their susceptibility to dictionary attacks. EFS also provides for the encryption of the File Encryption Key with one or more recovery key public keys, this allows for the possibility of the recovery of the file data should an individual leave an organization or lose their key. The encryption and decryption processes are shown in Figures 4 and 5 below.

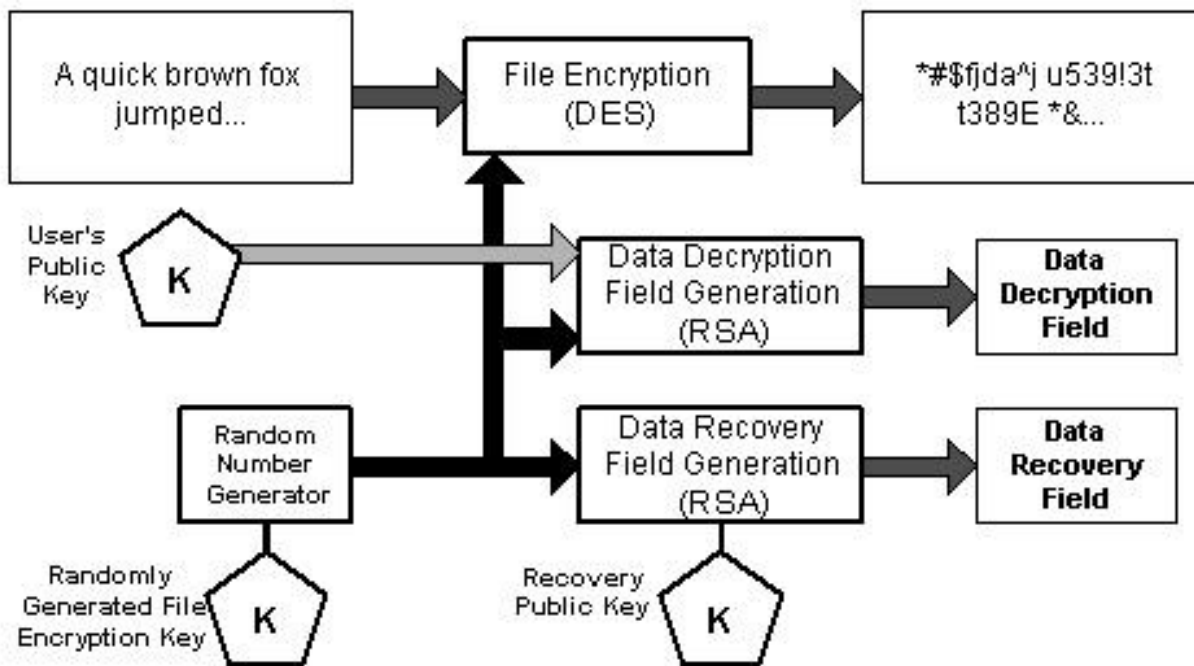


Figure 4: Microsoft EFS Encryption (from [3])

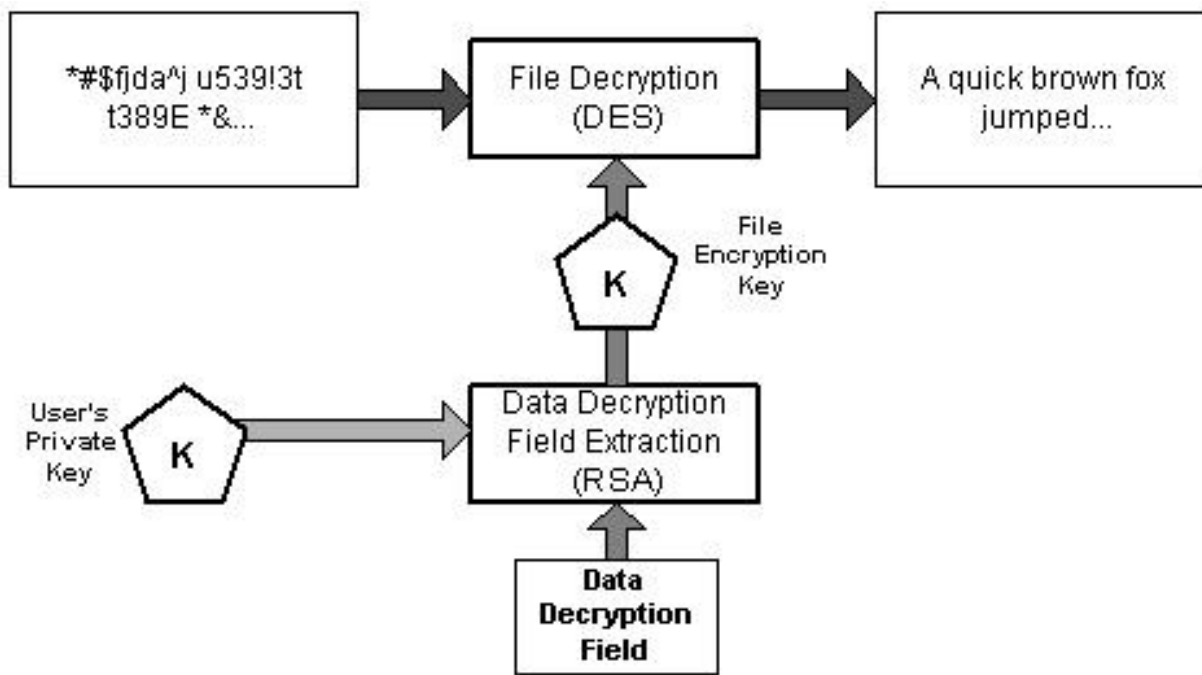


Figure 5: Microsoft EFS Decryption (from [3])

EFS is designed to be transparent to the end-user under normal circumstances. When a user attempts to access one of their encrypted files, the EFS will locate the private key used to encrypt the File Encryption Key that encrypted the file, decrypt the File Encryption Key, and decrypt the file. If an attempt is made to access a file encrypted by another user, EFS will fail to find the private key that can be used to decrypt the File Encryption Key and the user will be presented with an "Access Denied" condition.

Microsoft's implementation of EFS is shown in Figure 6. This architecture is broken up into four primary categories: the EFS driver, FileSystem Run-Time Library, the EFS service, and various Win32 APIs. The driver is layered on top of the NTFS filesystem. It communicates with the EFS service to request file encryption keys and other services. The driver passes this information to the run-time library in order to perform various file system operations. The EFS driver and run-time library do not communicate directly but rather use the NTFS file control call-out mechanism to communicate. The EFS service is part of the security subsystem and uses the port between the Local Security Authority (LSA) and the kernel-mode security reference monitor in order to communicate with the EFS driver. The EFS service interfaces with the CryptoAPI (not the same CryptoAPI for Linux) in user-mode to provide file encryption keys and other services. It also provides support for Win32 APIs that provide the programming interfaces for encrypting, decrypting or recovering files, as well as the importing and exporting of encrypted files.

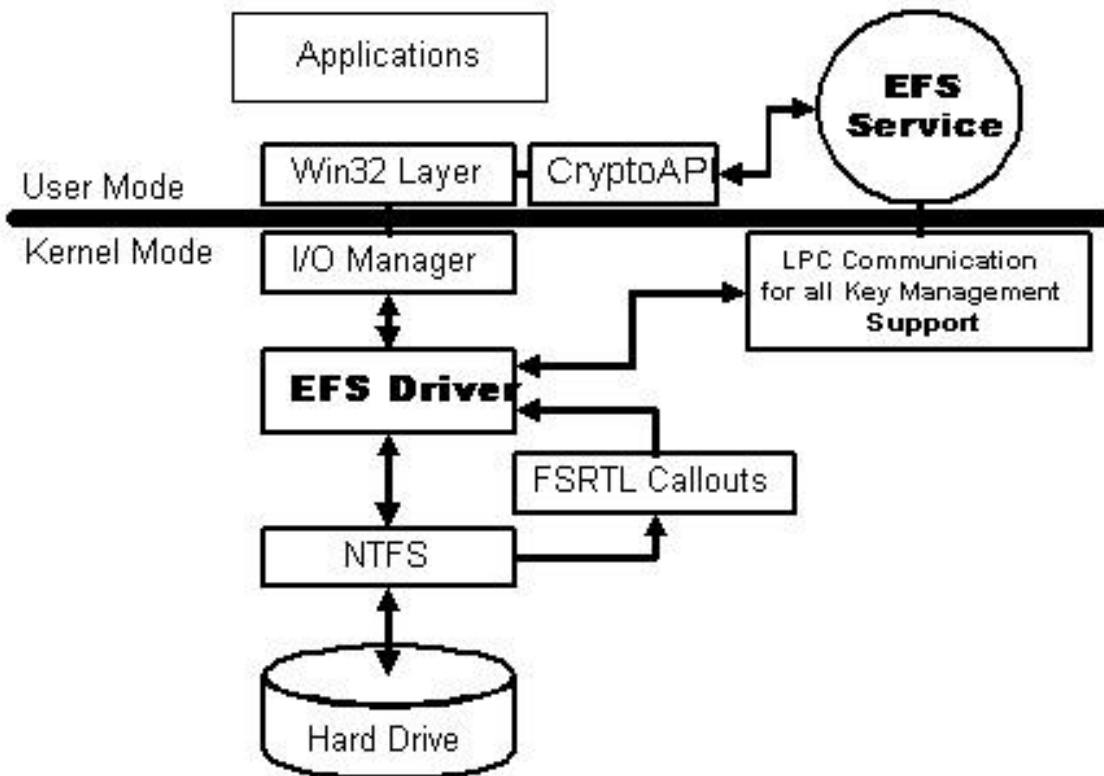


Figure 6: EFS Architecture (from [3])

EFS currently only supports the DESX encryption algorithm, which is based on a 128-bit encryption key. Microsoft says that future releases of EFS will support alternate encryption algorithms.

Summary

Cryptographic filesystems have made significant progress in the past few years. The capability of providing transparent encryption services to users in order to protect the data stored on a filesystem has become more enticing as networks are increasingly coming under attack from both external and internal sources. One of the key tenets of a solid security policy is the capability to ensure the integrity of data stored on network systems. Cryptographic filesystems make this increasingly possible by easing the effort of encrypting files.

References

- [1] Zadok, Erez, Ion Badulescu and Alex Shender, "Cryptfs: A Stackable Vnode Level Encryption File System", CUCS-021-98,
<http://www.cs.columbia.edu/~ezk/research/cryptfs/cryptfs.pdf>, 1998

[2] Bryson, David, "Using CryptoAPI",
<http://www.kerneli.org/howto/node3.php>, 2002.

[3] Microsoft, "Encrypting File System for Windows 2000",
<http://www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp>, 1998.

[Privacy Statement](#)

Copyright 2006, SecurityFocus