

Cryptographic File Systems, Part Two: Implementation

Ido Dubrawsky 2003-04-14

This is the second article in a two-part series looking at cryptographic filesystems. The [first article](#) in this series covered the background on cryptographic filesystems from the underlying concepts to some of the mechanics of those systems. This article will cover implementation. The focus will be on implementing the Microsoft's EFS under Windows 2000 and the Linux CryptoAPI.

One point to clarify from the first article involves the note that Microsoft's EFS does not support using a password-based symmetric algorithm. This is due to the concern that such schemes are weaker because of their susceptibility to dictionary attacks. While technically accurate, the fact remains that the public portion of the user's X.509v3 certificate (which is used to encrypt the File Encryption Key, or FEK, used by EFS) is used to encrypt the FEK. To decrypt the FEK requires the use of password or passphrase and unless password-based logon is disabled completely this password or passphrase is typically the user's domain password.

Microsoft Windows 2000 EFS

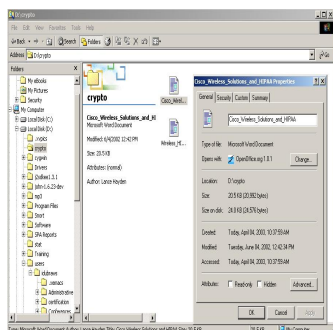
Microsoft's Windows 2000 product introduced the encrypting file system (EFS) to the Windows product line. While third party add-on encrypting software has been available for some time, EFS was the integration of such a concept into Windows 2000. Under Windows 2000, EFS supports the DESX algorithm only. With Windows XP that encryption algorithm now includes 3DES as well and will eventually include the Advanced Encryption Standard (AES) algorithm.

Set-Up

Using EFS is very easy. Windows 2000 supports both a command line method as well as a graphical method of encrypting files with EFS. Encrypting individual files and folders is very easy under Windows 2000. The tricky part is when those files and folders are moved either to different computers or stored on another computer because the appropriate certificate and private keys must also be stored with the files on that computer in order to be able to decrypt the files.

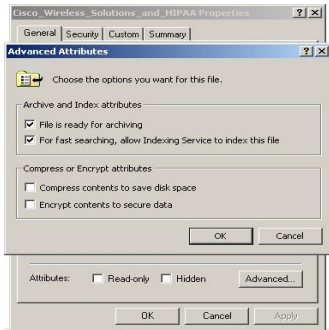
Encrypting/Decrypting a File: Graphical Method

Encrypting a file is as simple as clicking on the file with the right mouse button and opening up the file properties as shown below.



Click image to enlarge

Next, select the "Advanced" button at the bottom right of the properties panel to bring up the "Advanced Attributes" window. At the bottom of this window is a checkbox entitled "Encrypt Contents to secure data". Select the checkbox and click "OK".

**Click image to enlarge**

When the "Advanced Attributes" window has closed, select "Apply" at the lower right of the properties window. This will bring up a new dialog window asking if you want to encrypt only the selected file or the selected file and its parent folder. Select one of these two options and click the "OK" button at the lower right.

**Click image to enlarge**

Once the file has been encrypted it can still be accessed just like any other file. The file does not have to be decrypted to view it because the entire encryption/decryption process is automatic and transparent. To decrypt a file completely (that is where it is not stored on the disk encrypted) simply open the file properties panel by right clicking on the file or folder. Select the Advanced Attributes button and de-select the "Encrypt Contents to secure data" checkbox. Apply the properties to the file and it reverts back to a normal, unencrypted file which can be viewed by other users (depending on the NTFS attributes of the file).

Encrypting/Decrypting a File: Command Line Method

Encrypting and decrypting a file using the Windows command line is easier than with the graphical method. The **cipher.exe** program is used to encrypt/decrypt a file or a folder and its contents. An example is shown below. To encrypt a file the command is:

```
c:cipher.exe /e "D:\crypto\file.txt"
```

To encrypt a folder and all of its subfolders and files:

```
c:cipher.exe /e /s "D:\crypto"
```

To decrypt, simply change the `/e` flag to a `/d` as shown below:

```
c:cipher.exe /d /s "D:\crypto"
```

Remember, to just decrypt a file don't use the `/s` flag.

There are significantly more complex challenges with regards to EFS when files are moved from one computer to another. In order to provide encryption/decryption capabilities with the same FEK on different computers, users will have to export their keys using the Microsoft Management Console on one system and import them on other systems. Microsoft provides a step-by-step guide to using EFS including how to export and import FEKs. It is available [here](#).

Linux CryptoAPI Implementation

Implementing a cryptographic filesystem under Linux is not much harder than the implementation under Windows 2000. Three (and possibly four, depending on your Linux software) items are required to implement CryptoAPI under Linux -- a current version of the [Linux Kernel](#) the CryptoAPI [source code](#) and the [cryptoloop](#) software. Both of these should be downloaded, their signatures checked, and then unpacked. The CryptoAPI version should match relatively closely, if not exactly, with the kernel version.

```
[root@charybdis cryptoapi-0.1.0]# bzip2 -dc /opt/software/kernel/linux-2.4.20.tar.bz2 | tar -xvf -
[root@charybdis cryptoapi-0.1.0]# bzip2 -dc cryptoapi-0.1.0.tar.bz2 | tar -xvf -
[root@charybdis cryptoapi-0.1.0]# bzip2 -dc cryptoloop-0.0.1-pre1.tar.bz2 | tar -xvf -
```

Two patches to the loopback filesystem device driver are provided in the CryptoAPI sources. The first is the `loopiv` (`iv` stands for initialization vector) patch, which provides minimal support for the CryptoAPI, and the second is the `loop-jari` patch, which also provides support for CryptoAPI as well as some bug fixes.

Once the kernel and the CryptoAPI source code have been unpacked, you will need to configure the kernel *before* applying the CryptoAPI patches. The process can be done for an existing kernel by using the current configuration in place. The following instructions are for building a new kernel with the CryptoAPI. Once the kernel has been configured you can apply the CryptoAPI patches using the command:

```
[root@charybdis cryptoapi-0.1.0]# make patch-kernel KDIR=< kernel dir > LOOP= < iv|jari >
```



```
[root@charybdis cyrptoapi-0.1.0]# losetup -e des /dev/loop0 /oracle/testfs
```

Password:

Init (up to 16 hex digits):

Once the loopback device is set up the filesystem needs to be made on it:

```
[root@charybdis cyrptoapi-0.1.0]# mkfs -t ext3 /dev/loop0
```

Now the filesystem is ready to be mounted:

```
[root@charybdis cyrptoapi-0.1.0]# mount -t ext3 /dev/loop0 /mnt/testfs
```

```

# root@charybdis:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       18G  11G  6G   61% /
/dev/sda2       48G  23G  25G  48% /home
/dev/sda7       151G  33G  118G  21% /opt
/dev/sda8       127G  147G  128G  116% /usr
/dev/sda5       48G  23G  25G  48% /var
/dev/sda3       151G  85G  66G  56% /opt/software
/dev/sda4       180G  178G  100G  99% /oracle
/dev/loop0      50  0  50 100% /mnt/testfs
  
```

[Click image to enlarge](#)

All files placed in this filesystem will be encrypted. Be aware that it is not recommended that the filesystem be too large because of the potential performance hit that will occur. Remember that anything written to and read from must be encrypted and decrypted respectively. CryptoAPI provides the capability to create encrypted filesystems that are transparent to system users.

Summary

The main benefit of using encrypted filesystems comes from the ability to secure sensitive data beyond what is normally capable in a system. This additional security is dependent not only on the strength of the encryption algorithm but also on its proper use by users and administrators. An encrypted filesystem is of no use if the user or the administrator do not properly handle the keys used to encrypt the data on the filesystem.

Relevant Links

[Cryptographic Filesystems, Part One: Design and Implementation](#)

Ido Dubrawsky

[Privacy Statement](#)

Copyright 2006, SecurityFocus