Encrypting File System for Windows 2000

Abstract

This document provides an executive summary and a technical overview of the encrypting file system (EFS) that will be included with the Microsoft® Windows® operating system.

EFS provides the core file encryption technology to store Windows NT file system (NTFS) files encrypted on disk. EFS particularly addresses security concerns raised by tools available on other operating systems that allow users to access files from an NTFS volume without an access check. With EFS, data in NTFS files is encrypted on disk. The encryption technology used is public key-based and runs as an integrated system service making it easy to manage, difficult to attack, and transparent to the user. If a user attempting to access an encrypted NTFS file has the private key to that file, the user will be able to open the file and work with it transparently as a normal document. A user without the private key to the file is simply denied access.

EFS Table of Contents

EFS Introduction and Rationale

A standard safety measure on a personal computer system is to attempt to boot from a floppy disk before trying to boot from the hard disk. This guards users against hard drive failures and corrupted boot partitions. Unfortunately, it also adds the convenience of booting different operating systems. This can mean someone with physical access to a system can bypass the built-in security features of the Microsoft® Windows NT® file system access control by using a tool to read Windows NT file system (NTFS) on-disk structures. Many hardware configurations provide features like a boot password to restrict this kind of access. Such features are not in widespread use, and in a typical environment, where multiple users are sharing a workstation they don't work very well. Even if these features were universal, the protection provided by a password is not very strong. Additionally, the hard disk can be removed from a secure computer and then plugged into a computer where the perpetrator has sufficient access.

Typical scenarios where unauthorized data access becomes an issue include:

A stolen laptop. It only takes a moment for someone to pick up an unattended laptop. What if the thief is not interested in reselling your computer, but is interested in the sensitive information stored on its hard drive?

Unrestricted access. Office desktop systems are left unattended and anyone can come in and quickly steal information from an unattended computer.

The root of these security concerns is sensitive information, which typically exists as unprotected files on your hard drive. You can restrict access to sensitive information stored on an NTFS partition if Windows NT is the only operating system that can be run and if the hard drive cannot be physically removed. If someone really wants to get at the information, it is not difficult if they can gain physical access to the computer or hard drive. Availability of tools that allow access to NTFS files from MS-DOS® and UNIX operating systems makes bypassing NTFS security even easier.

Data encryption is the only solution to this problem. There are a number of products on the market that provide application-level file encryption using password-derived keys. However, there are some limitations with most of these approaches:

Manual encryption and decryption on each use. Encryption services are not transparent to the user in most products. The user has to decrypt the file before every use and re-encrypt it when finished. If the user forgets to encrypt a file, the file is unprotected. And, because the user must go to the trouble of specifying that a file be encrypted (and decrypted) on each use, it discourages the use of encryption.

Leaks from temporary and paging files. Many applications create temporary files while a user edits a document (Microsoft Word for one). These temporary files are left unencrypted on the disk, even though the original document is encrypted, making data theft easy. And, application level encryption runs in Windows NT user mode. This means that the user's encryption key may be stored in a paging file. It is fairly easy to gain access to all documents encrypted using a single key by simply mining a paging file.

Weak security. Keys are derived from passwords or pass-phrases. Dictionary attacks can easily breach this kind of security if easy to remember passwords are used. Forcing more complicated passwords makes for more complicated usability.

No data recovery. Many products do not provide data recovery services. This is another discouragement to users, especially ones who do not want to remember another password. In the cases where password-based data recovery is provided, it creates another weak point of access. All a data thief needs is the password to the recovery mechanism to gain access to all encrypted files.

EFS addresses all the problems mentioned above and more. The following four sections go into detail on the encryption technology, where encryption takes place in the system, user interaction, and data recovery.

EFS ENCRYPTION TECHNOLOGY

EFS is based on public-key encryption, taking advantage of the CryptoAPI architecture in Windows. Each file is encrypted using a *randomly generated key,* called the file encryption key, which is independent of a user's public/private key pair; thereby stifling many forms of cryptanalysis-based attack on the encrypted files.

File encryption can use any symmetric encryption algorithm. The first release of EFS will expose DESX as the encryption algorithm. Future releases will allow alternate encryption schemes.

EFS also supports encryption and decryption on files stored on remote file servers. Note: in this case EFS only addresses encrypting data on disk. It does not encrypt data that is transferred over the network. Windows 2000 provides network protocols such as SSL and IPSEC to encrypt data over the network.

Where EFS Lives

EFS is tightly integrated with NTFS. When temporary files are created, the attributes from the original file may be copied to temporary files as long as all files are on NTFS volume. If the original file is encrypted, EFS encrypts its temporary copies when attributes are transferred during file creation. EFS resides in the Windows 2000 kernel and uses the non-paged pool to store file encryption keys, ensuring that they never make it to the paging file.

User Interaction

The default configuration of EFS allows users to start encrypting files with no administrative effort. EFS automatically generates a public-key pair and gets the public key certified by a configured Certificate Authority (CA); or self-signs it—if there is no CA available to issue certificates.

File encryption and decryption is supported on a per file or entire directory basis. Directory encryption is transparently enforced. All files (and subdirectories) created in a directory marked for encryption are automatically encrypted. Each file has a unique encryption key, making it safe for rename operations. If you rename a file from an encrypted directory to an unencrypted directory on the same volume, the file remains encrypted. Encryption and decryption services are available from Windows Explorer. Additionally, command line tools and administrative interfaces are provided for advanced users and recovery agents so they can take full advantage of this capability.

A file need not be decrypted before use¾encryption and decryption is done transparently when bytes travel to and from the disk. EFS will automatically detect an encrypted file and locate a user's certificate and associated private key in user's certificate and key stores. Since the mechanism of key storage is based on CryptoAPI, users will have the flexibility of storing keys on secure devices, such as smart cards.

The initial release of EFS will not expose file sharing from the user interfaces; however, the APIs will expose the capability for future applications to leverage the capability. EFS is designed to allow file sharing between any number of people by the simple use of their public keys. Users can then independently decrypt files using their own private keys. Users can be easily added (if they have a configured public key certificate and associated private key) or removed from a group of permitted sharers.

The reason for not exposing file sharing to end users in the initial release of Windows 2000 is existing applications currently may perform operations, such as copying files, which may inadvertently cause sharing information to be lost thereby creating a usability problem. The reason for not exposing file encryption is similar—most applications can leave files unencrypted after editing is done on them. These features will be become available for end-users when applications developers are more aware of file encryption.

EFS DATA RECOVERY

EFS also provides built-in data recovery support. The Windows 2000 security infrastructure enforces the configuration of data recovery keys. You can use file encryption only if the system is configured with one or more recovery keys. EFS allows recovery agents to configure public key certificates that are used to enable file recovery. Only the file's randomly generated encryption key is available using the recovery key, not a user's private key. This ensures that no other private information is revealed to the recovery agent accidentally—only the data that falls in the scope of influence of a recovery agent is recoverable by the agent.

Data recovery is intended for most business environments where the organization expects to be able to recover data encrypted by an employee after an employee leaves or when encryption keys are lost. The recovery policy can be defined at the domain controller of a Windows 2000 domain. Like most other policies in Windows 2000, the policy defining Encrypted Data Recovery Agents is configured as part of Group Policy Objects (GPOs). These GPOs can then be assigned at different scopes—Domain or Organizational Units. The policy defined at the closest scope to a given computer takes effect on that computer. *There is no accumulation of Encrypted Data Recovery Agents Policy—* therefore if there are multiple policies configured at different scopes, then the policy applied last gets enforced. To understand how group policies work, see the technical White Papers and other information on Group Policy and Windows Administration available from www.microsoft.com. For information on security policies, refer to the technical White Paper S*ecurity Configuration Tool Set*, also available from www.microsoft.com.

By default, recovery policy is under the control of domain administrators. To reduce any need for administration, EFS automatically configures a default recovery policy making the domain administrator account the recovery agent for the domain. The certificate used may be a self-signed one if there is no Certificate Authority available. Domain administrators can delegate this to designated data security administrator accounts using Windows 2000 Directory Service delegation features. This provides better control and flexibility on who is authorized to recover encrypted data. EFS also supports multiple recovery agents, by allowing for multiple recovery key configurations to provide organizations with redundancy and flexibility in implementing their recovery procedures. You can also leverage the scope-based enforcement of Group Policy to have different recovery agents for different parts of your organization. For example, the recovery agent(s) for company executives may be different from rest of the employees.

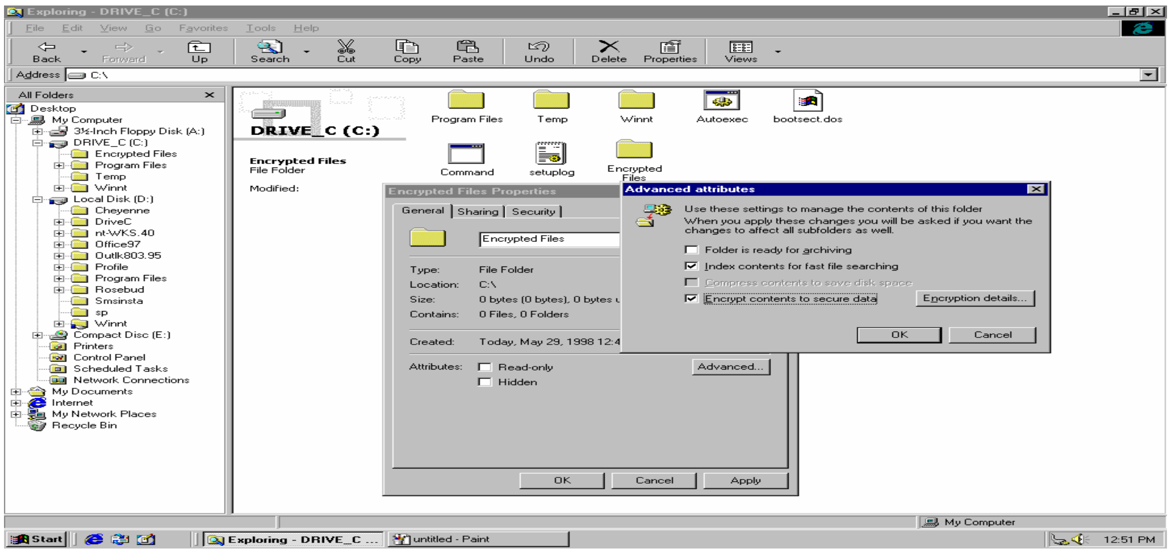
EFS can also be used in small office or home office environments. EFS will automatically generate a recovery key, issue a self-signed certificate to the local administrator account on first logon and save it in the administrator's certificate store just as is the case for default policy in the domain. This makes the local administrator the default recovery agent on stand-alone workstation/servers allowing the local administrator to recover any encrypted file on the system. Note that this is only the default policy. Users may change this to suit their requirements.

USING THE ENCRYPTING FILE SYSTEM

The following sections provide user scenarios that demonstrate how EFS works.

User Operations

The following figure shows the Windows Explorer folder property page that displays encryption services and the dialog box generated by selecting Advanced on General tab.


Encrypted Files Properties

General | Sharing | Security

Encrypted Files

Type: File Folder
Location: C:\
Size: 0 bytes (0 bytes), 0 bytes u
Contains: 0 Files, 0 Folders

Created: Today, May 29, 1998 12:4

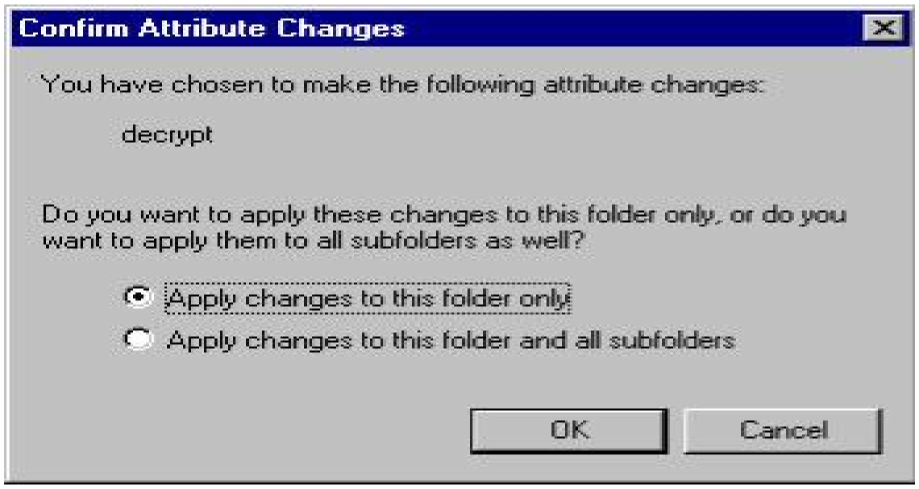
Attributes: ☐ Read-only ☐ Hidden

Advanced attributes

Use these settings to manage the contents of this folder
When you apply these changes you will be asked if you want the changes to affect all subfolders as well.

☐ Folder is ready for archiving
☑ Index contents for fast file searching
☐ Compress contents to save disk space
☑ Encrypt contents to secure data

This figure shows the dialog box that appears when the encryption checkbox is modified on the folder and Apply or OK is selected on the property page.


Confirm Attribute Changes

You have chosen to make the following attribute changes:

decrypt

Do you want to apply these changes to this folder only, or do you want to apply them to all subfolders as well?

● Apply changes to this folder only
○ Apply changes to this folder and all subfolders

Selecting the Advanced button on the folder property page exposes the following EFS features to the user:

Encryption—Selecting the Encrypt contents to secure data check box allows the user to encrypt the currently selected folder. Additionally, the user can encrypt all files (and subdirectories) in the directory.

Decryption—This option is converse to encryption Clearing the Encrypt contents to secure data box allows the user to decrypt the currently selected folder. It also lets users decrypt all files and sub-folders in the directory, in addition to resetting the directory as unencrypted.

NOTE: *Encryption, decryption operations are NOT exposed on individual files from the graphical user interface. This is to encourage users to turn encryption on at the folder level rather than on individual files. The reason being that different file-based applications manipulate files in a variety of ways and can inadvertently leave files in plaintext on modifications. To protect the users' sensitive data, encryption is exposed at folder level for this release—this ensures all files created in the folder (including temporary files) are encrypted. Once more applications become encryption aware, file level feature will be exposed.*

In addition to the graphical interface, Windows 2000 includes a command line tool for richer functionality needed for administrative operations. The command line tool is:

Cipher command line utility—This provides the ability to encrypt and decrypt files and folders from a command prompt.

Examples:

- To encrypt the C:\My Documents directory, the user types:

  ```
  C:\>cipher /e My Documents
  ```

- To encrypt all files with "cnfdl" in the name, the user types:

  ```
  C:\>cipher /e /s *cnfdl*
  ```

The complete cipher command supports the following options:

```
D:\>cipher /?

Displays or alters the encryption of files on NTFS partitions.

  CIPHER [/E | /D] [/S:dir] [/P:keyfile] [/K:keyfile] [/L:keyfile] [/I] [/F] [/Q] [filename [...]]

   /E     Encrypts the specified files. Directories will be marked so that files added afterward will be encrypted.
   /D     Decrypts the specified files. Directories will be marked so that files added afterward will not be encrypted.
   /S     Performs the specified operation on files in the given directory and all subdirectories.
   /I     Continues performing the specified operation even after errors have occurred. By default, CIPHER stops when an
              error is encountered.
   /F     Forces the encryption operation on all specified files, even those which are already encrypted. Already-encrypted
              files are skipped by default.
   /Q     Reports only the most essential information.

 Used without parameters, CIPHER displays the encryption state of the current directory and any files it contains. You may
use multiple filenames and wildcards. You must put spaces between multiple parameters.
```

File/Folder Encryption

All that the user needs to do is select one or more folders and select the encryption check box on the folder properties advanced attributes dialog box. Windows Explorer will call EFS to encrypt the selected folders and provide the user a pop-up option to encrypt all existing files and any subfolders in the selected folders. Marking a folder encrypted will ensure that all future files in that folder are encrypted by default and all future subfolders under it are marked encrypted. The folder's list of files is not encrypted and you can enumerate files as usual, provided you have sufficient access to the folder. Folder encryption provides users the ability to manage their sensitive files by simply copying them to encrypted folders.

Once a file is encrypted, it is stored encrypted on the disk. All reads and writes to the file are decrypted and encrypted transparently. To find out if the file is encrypted, users can check the properties on the file to see if the check box is selected. The list view on Windows Explorer can also be extended to see the attributes—“E” in the attributes column indicates the file is encrypted. Since the encryption is transparent, the user can use the file as before. For example, the user can still open the Word document and edit it as before or open a text file using Notepad and do the same. Any other user trying to open this encrypted file will get an access denied error as the user will not posses a key to decrypt the file.

It is important to note that users (administrators, in this case) should not encrypt files in the system directory. This is because these files are needed for the system to boot. During the boot process, a user's private key is not available to decrypt the files. Such an operation can render the system useless. EFS will safeguard this by failing encryption attempts on files with the system attribute. Future releases of Windows will provide secure boot capabilities that will support encryption of system files.

EFS also provides users the ability to transfer encrypted files across systems. This is achieved through standard backup and restore mechanisms. All that the user needs to do is back up the encrypted file to removable media using a backup tool. The backed up file is still encrypted. The user can then copy this file backup to different file systems including FAT, backup tapes, or send it as an e-mail attachment like a normal file. To be able to use the file on a system where it is copied to, the user restores the file on an NTFS volume (Note: Windows 2000 only) and the restored file is created as an encrypted file.

Simply copying the file to a non-NTFS version 5.0 volume will make a copy in plaintext. This is because the normal copy command uses file reads which are transparently decrypted by EFS. This can be used to create plaintext copies of an encrypted file for distribution. If the copy is saved to a NTFS version 5.0 volume, the copy will end up being encrypted. If it is saved on the same system as the original, the encryption will be identical to the original. However, if the copy is on a remote system, the file will be encrypted but will not be identical to the original—this is because the file encryption key can not be exported across the wire securely and hence the remote copy will be encrypted with a new FEK.

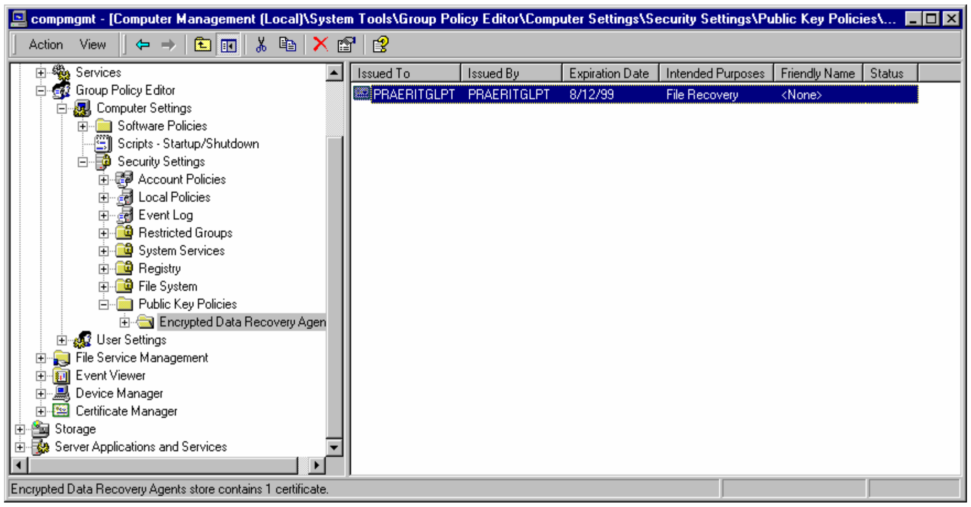
File or Directory Decryption

Users will not need to decrypt files or directories for normal operations because EFS provides transparent encryption and decryption during data writes and reads. Such operations may however be required under special circumstances where a user needs to share an encrypted file with other users (sharing of encrypted files is not available in this release).

Users can decrypt files and mark directories unencrypted using the Windows Explorer—encryption check box on the folder property page. The operation is similar to encryption. Performing this operation on one or more folders will cause EFS to decrypt the all files and sub-folders under the selected folder and mark each of them as unencrypted.

Recovery Operations

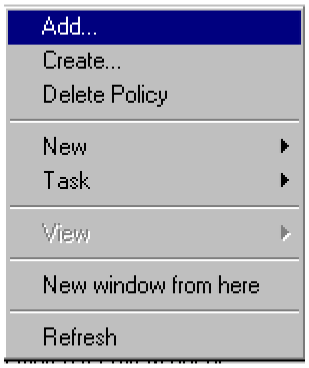
EFS recovery policy is implemented as part of the overall security policy for the system. It may be configured in Group Policy Objects at the domain level or organizational units in the Active Directory, such that it applies to all Windows 2000-based computers within the defined scope, or it may be configured locally on the computer. The user interface is integrated with Security Settings extension snap-in to Group Policy Editor. The node is called “Encrypted Data Recovery Agents” and appears under Public Key Policies area.

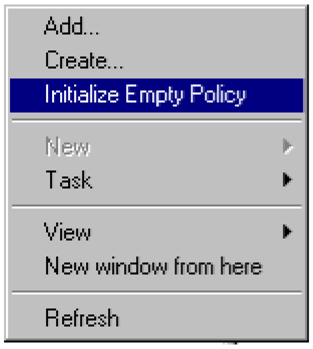

compmgmt - [Computer Management (Local)\System Tools\Group Policy Editor\Computer Settings\Security Settings\Public Key Policies\...

Issued To	Issued By	Expiration Date	Intended Purposes	Friendly Name	Status
PRAERITGLPT	PRAERITGLPT	8/12/99	File Recovery	<None>	

Encrypted Data Recovery Agents store contains 1 certificate.

This interface allows administrators to define “no recovery policy”, an “empty” recovery policy or a recovery policy with a one or more X509 version 3 certificates belonging to individuals identified as recovery agents for that scope of administration (domain, organizational unit or the computer). The following menu options allow you to add existing certificates or create new ones.





NOTE: *Setting up an “empty policy” will turn EFS off, thereby not allowing users to encrypt files on computers that fall in that category. Setting up “no policy” (deleting policy) will allow the default local policy on computers to be used, in effect allowing local administrators to control the recovery of data on their individual computers.*

Integrating the recovery policy with a system security policy provides a coherent security enforcement model. The Windows 2000 security subsystem takes care of enforcing, replicating, caching and updating the recovery policy. Therefore, users are able to use file encryption on a temporarily offline system, such as a laptop, much like they are able to logon to their domain account using cached credentials.

Encryption Recovery

EFS requires that a data recovery policy be set up at a domain or OU level (or even locally) before EFS can be used. The recovery policy is set up by domain administrators (or by delegated personnel known as *recovery agents*) that control the recovery keys for all machines in the scope of influence for the policy.

If a user loses a private key, a file protected by that key can be recovered by backing up the file and sending it in e-mail to one of the recovery agents. The recovery agent will restore the encrypted file on a secure machine with the private recovery keys and then simply decrypt the file using cipher command line or Windows Explorer. The recovery agent then returns the plaintext file back to the user. Alternatively, the recovery agent can go to the machine with the encrypted file and load their recovery certificate and private key and perform the recovery on the machine. The latter may not be safe and is not recommended as a general method because of the sensitivity of the recovery key which should not be left on an unsecured machine.

In a small business environment or home environment where there are no domains, recovery can be done on the stand-alone computer itself using the local administrator account, which is configured as the default recovery agent.

Examples

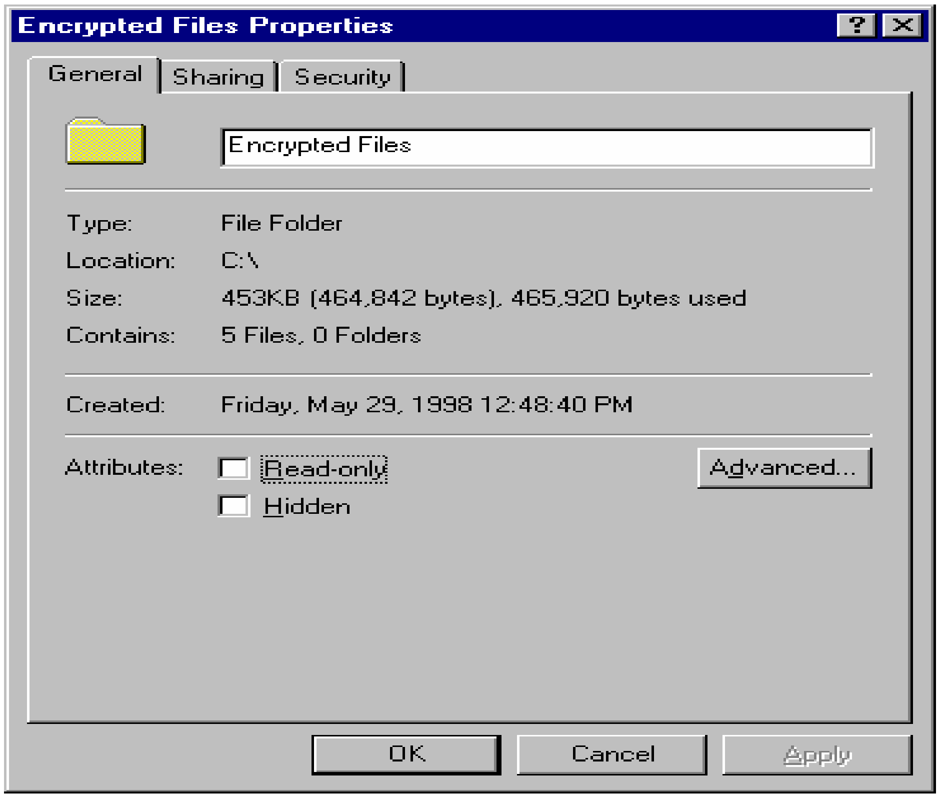
To understand the usage of this technology better, let's walk through some examples.

Example 1: Encrypt a folder on local machine

The steps are:

· Right-click on the selected folder to bring up Properties.

· Click Advanced on the General tab.


Encrypted Files Properties

General | Sharing | Security

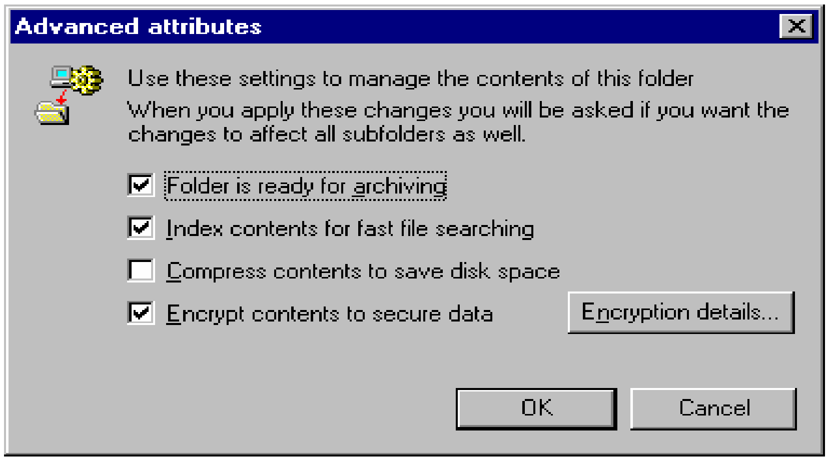
Encrypted Files

Type: File Folder
Location: C:\
Size: 453KB (464,842 bytes), 465,920 bytes used
Contains: 5 Files, 0 Folders

Created: Friday, May 29, 1998 12:48:40 PM

Attributes: Read-only, Hidden

· Select Encrypt contents to secure data.



Advanced attributes

Use these settings to manage the contents of this folder

When you apply these changes you will be asked if you want the changes to affect all subfolders as well.

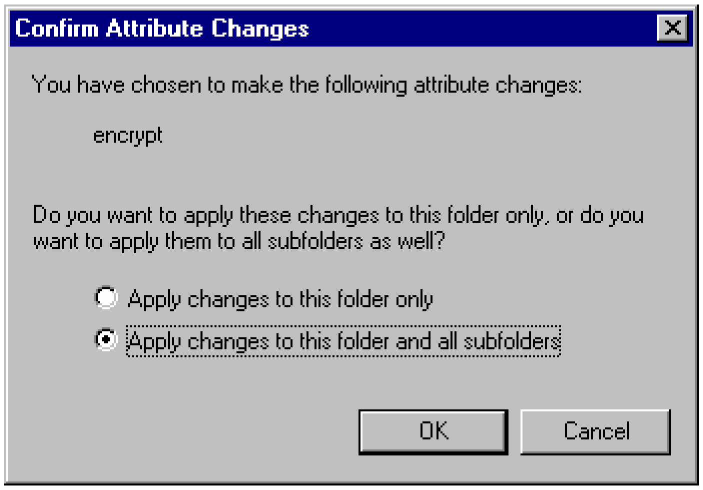
[✓] Folder is ready for archiving

[✓] Index contents for fast file searching

[ ] Compress contents to save disk space

[✓] Encrypt contents to secure data

· Click OK to close the dialog box.
· Click OK to apply and close the property page.
· A dialog box will prompt you to encrypt the folder only or all existing content.



Confirm Attribute Changes

You have chosen to make the following attribute changes:

encrypt

Do you want to apply these changes to this folder only, or do you want to apply them to all subfolders as well?

○ Apply changes to this folder only

● Apply changes to this folder and all subfolders

· Choose to encrypt the folder and all existing content (files and subfolders).
· Click OK. You can confirm that encryption happened by verifying the state of the attribute checkbox.

Example 2: Encrypt a folder on a remote machine

The steps are:

· Use the Tools menu in Windows Explorer to map a network share on the remote machine as a drive.
· Once mapped, you can navigate to the folder as in the local case above.
· Follow the steps in previous example to perform the operation.
· Note that if the remote volume is not NTFS version 5, this operation will not be allowed.

NOTE: *If the remote machine is a “trusted server” (trusted for delegation), EFS will be able to use the key from user’s roaming profile so that same key is used across systems. If the remote machine is not “trusted”, then a local profile is created on the machine and key is local to that machine and can be used on that machine only. Thus moving these files between machines would require you to move your keys also.*

Example 3: Decrypt a folder

The steps are:

- Right-click on the selected folder to bring up Properties.
- Click Advanced on the General Tab.
- Click to clear the Encrypt contents to secure data check box.
- Click OK to close the dialog box.
- Click OK to apply and close the property page.
- A dialog will prompt you to decrypt the folder only or all existing content.
- Choose to decrypt the folder and all existing content (files and subfolders).
- Click OK. You can confirm that decryption happened by verifying the state of the attribute checkbox.

Example 4: Copy entire encrypted folder

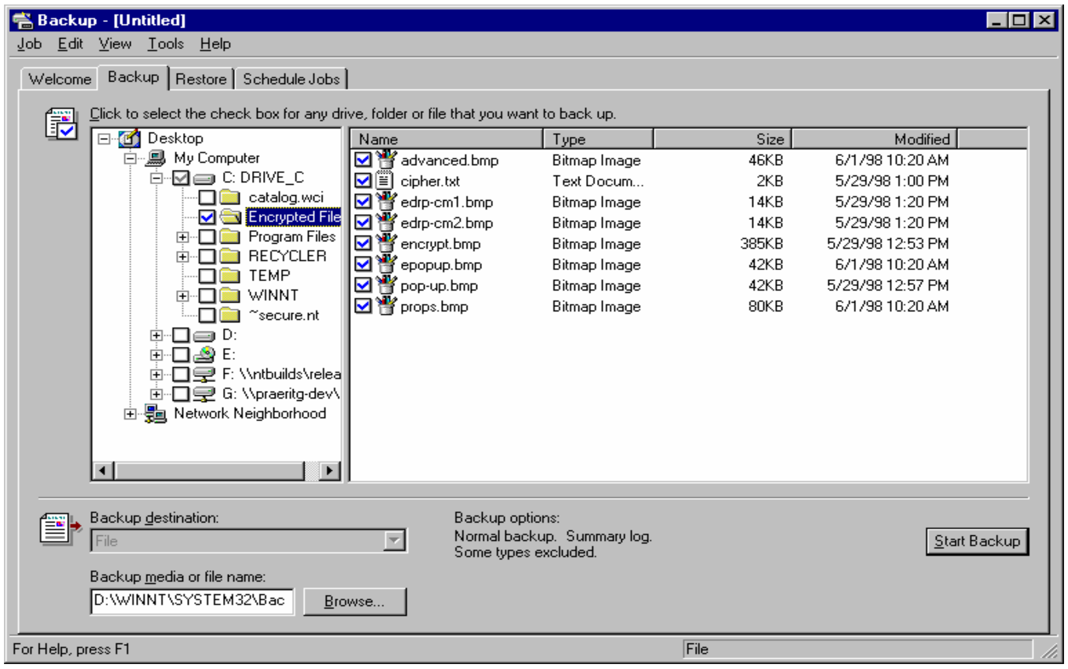
The steps are:

- Select the folder in Windows Explorer.
- Right-click and select Copy
- Open the folder where you want to place the copy.
- Right-click and select Paste.
- The destination folder and contents will remain encrypted.

Example 5: Backup an encrypted folder

The steps are:

- Start Backup (Start à Programs à Accessories à Backup).
- Use the browser to locate and check the folder you want to back up.



Click to select the check box for any drive, folder or file that you want to back up.

Desktop
My Computer
C: DRIVE_C
catalog.wci
Encrypted File
Program Files
RECYCLER
TEMP
WINNT
~secure.nt
D:
E:
F: \\ntbuilds\relea
G: \\praeritg-dev\
Network Neighborhood

Name	Type	Size	Modified
advanced.bmp	Bitmap Image	46KB	6/1/98 10:20 AM
cipher.txt	Text Docum...	2KB	5/29/98 1:00 PM
edrp-cm1.bmp	Bitmap Image	14KB	5/29/98 1:20 PM
edrp-cm2.bmp	Bitmap Image	14KB	5/29/98 1:20 PM
encrypt.bmp	Bitmap Image	385KB	5/29/98 12:53 PM
epopup.bmp	Bitmap Image	42KB	6/1/98 10:20 AM
pop-up.bmp	Bitmap Image	42KB	5/29/98 12:57 PM
props.bmp	Bitmap Image	80KB	6/1/98 10:20 AM

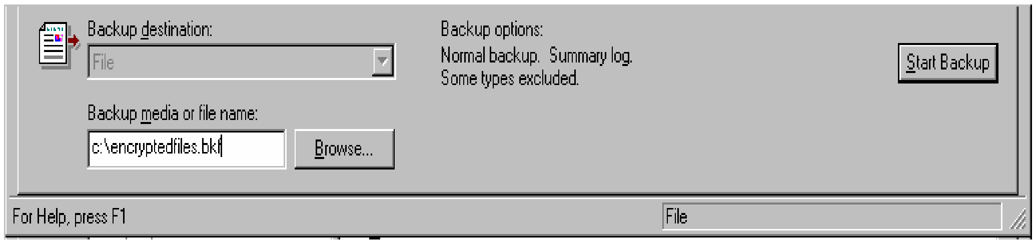
Backup destination: File

Backup options:
Normal backup. Summary log.
Some types excluded.

Backup media or file name: D:\WINNT\SYSTEM32\Bac

For Help, press F1 | File

- Select the backup file (.e.g. ENCRYPTED.BKF) where you want to back up the entire folder.

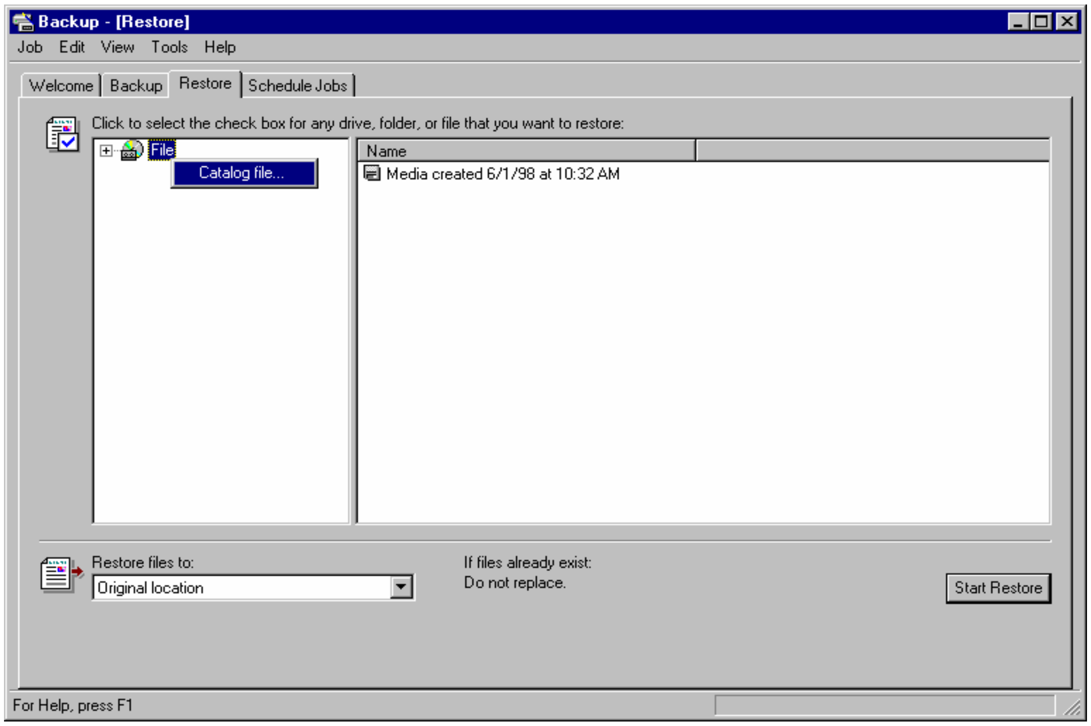


- Click Start Backup. Click Backup on the pop-up dialog box so the process continues.
- This will back up the entire encrypted folder to the backup file (e.g. ENCRYPTEDFILES.BKF).
- This file can be copied to removable media like floppy disks and will be secure because it will remain encrypted.

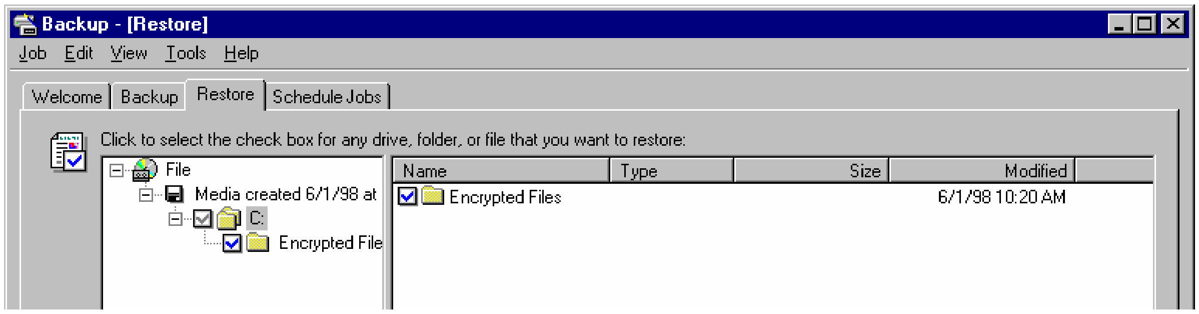
Example 6: Restore an encrypted folder

The steps are:

- Start Backup (Start à Run à Backup).
- Right-click File and select Catalog file.



- Enter the path to the backup file (for example ENCRYPTEDFILES.BKF)
- Select the encrypted folder that needs to be restored. All its contents are restored automatically.

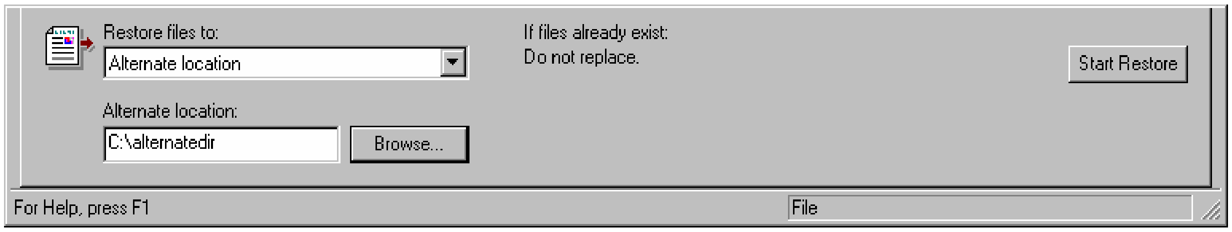


Click to select the check box for any drive, folder, or file that you want to restore:

File
Media created 6/1/98 at
C:
Encrypted File

Name	Type	Size	Modified
✓ Encrypted Files			6/1/98 10:20 AM

- Choose to restore files to Alternate location.


Restore files to: Alternate location

Alternate location: C:\alternatedir

If files already exist: Do not replace.

- Create or designate the folder under which you want the encrypted folder to be restored.
- Click Start Restore.
- On the dialog box, click OK to confirm.
- Click OK to confirm the backup file.
- The restore progress dialog box will show you the progress of the encrypted folder and its files being restored. You can check the folder Properties to confirm that indeed it was restored encrypted.

Example 7: Recover an encrypted folder

The steps are:

- You will back up the folder using Example 5 above.
- You will send the backup file as a mail attachment to your Recovery Agent.
- Recovery Agent will restore the folder using Example 6 above on the machine where the recovery keys are kept.
- Recovery Agent will then use Windows Explorer to simply decrypt the folder by clearing the Encrypt contents to secure data check box. A dialog box will confirm if the operation has to be performed on just the folders, but any subfolders as well or not.
- Once the entire folder has been decrypted, Recovery Agent can use the back up folder steps in Example 5 to create a backup file and return it back to the user.

EFS Recommendations

Encryption is a sensitive operation. It is important that encrypted data does become unencrypted inadvertently. To this end, we recommend the following:

- Encrypt the My Documents folder (%UserProfile%\My Documents)—this ensures that most Office documents will be encrypted by default.
- Encrypt the Temp folder (%TEMP%)—this ensures that any temporary files created by various applications are encrypted, thus avoiding any possible leaks.

- Always encrypt folders rather than individual files. Windows Explorer only allows folder-based encryption, however CIPHER.EXE will let you encrypt individual files. Applications work on files in various ways, for example creating temporary files in the same folder as the original during editing—encrypting at folder level ensures that files don't get decrypted transparently.
- The private keys associated with recovery certificates are extremely sensitive. Never leave them unsecured. Either generate them on a computer that is physically secured or export the key and certificate into a .pfx file, protected under a strong password and secure that file on a floppy disk.

EFS Cryptographic Architecture

This section provides a brief technical and architectural overview of EFS.

Cryptography

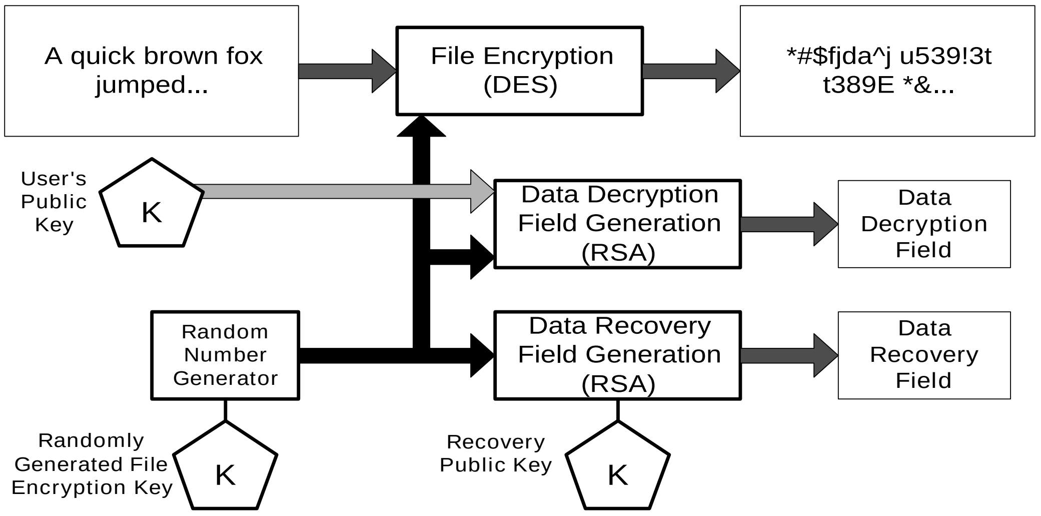
EFS implements data encryption and decryption using a public key-based scheme. File data is encrypted using a fast symmetric algorithm with a file encryption key (FEK). The FEK is a randomly generated key of a certain length required by the algorithm or by law if the algorithm supports variable length keys. Export issues relating to EFS are discussed below in this document.

The FEK is encrypted using one or more key encryption public keys to generate a list of encrypted FEKs. The public portion of a user's key pair is used to encrypt FEKs. This public portion of the user's key pair is obtained from the User's X509 version 3 certificate, with enhanced key usage as "File Encryption". The list of encrypted FEKs is stored along with this encrypted file in a special EFS attribute called the *Data Decryption Field* (DDF). The file encryption information is tightly bound to the file. The private portion of the user's key pair is used during decryption. The FEK is decrypted using the private portion of the key pair. The private portion of a user's key pair is stored safely elsewhere in smart cards or other secure storage such as the integrated software-based protected store used by CryptoAPI.

NOTE: *A user's key encryption can also be done using a symmetric algorithm such as a password-derived key. EFS does not support this because password-based schemes are inherently weak due to their susceptibility to dictionary attacks.*

The FEK is also encrypted using one or more recovery key encryption public keys (obtained from the recovery agent X509 version 3 certificates stored in the Encrypted Data Recovery Agent (EDRA) Policy for the computer—the enhanced key usage for these certificates must be "File Recovery"). Again, the public portion of each key pair is used to encrypt FEKs. This list of encrypted FEKs is also stored along with the file in a special EFS attribute called the *Data Recovery Field* (DRF). Only public portions of the recovery key pairs are needed for encryption of the FEK in the DRF. These public recovery keys are required to be present at all times on an EFS system for normal file system operations. They are present in EDRA policy as X509 version 3 "File Recovery" certificates. Recovery itself is expected to be a rare operation required only when users leave organizations or lose keys. Because of this, recovery agents can store the private portions of the keys safely elsewhere (on smart cards and other secure storage devices).

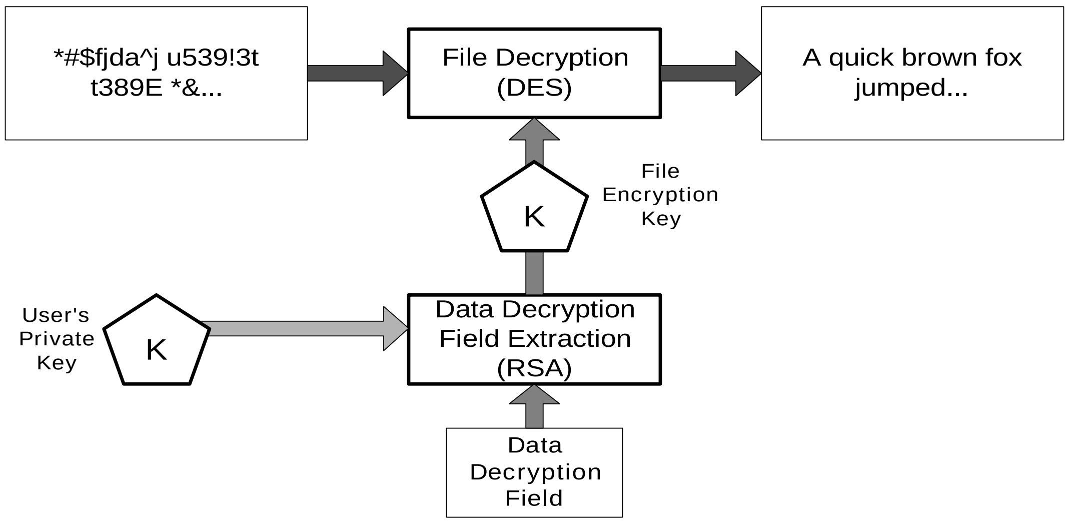
The following diagrams illustrate the encryption, decryption, and recovery processes. The encryption process:


A quick brown fox jumped...
File Encryption (DES)
*#$fjda^j u539!3t t389E *&...
User's Public Key
K
Data Decryption Field Generation (RSA)
Data Decryption Field
Random Number Generator
Data Recovery Field Generation (RSA)
Data Recovery Field
Randomly Generated File Encryption Key
K
Recovery Public Key
K

The user’s plaintext file is encrypted using a randomly generated FEK. This file encryption key is stored along with the file, encrypted under a user’s public key in the DDF and encrypted under the recovery agent’s public key in the DRF.

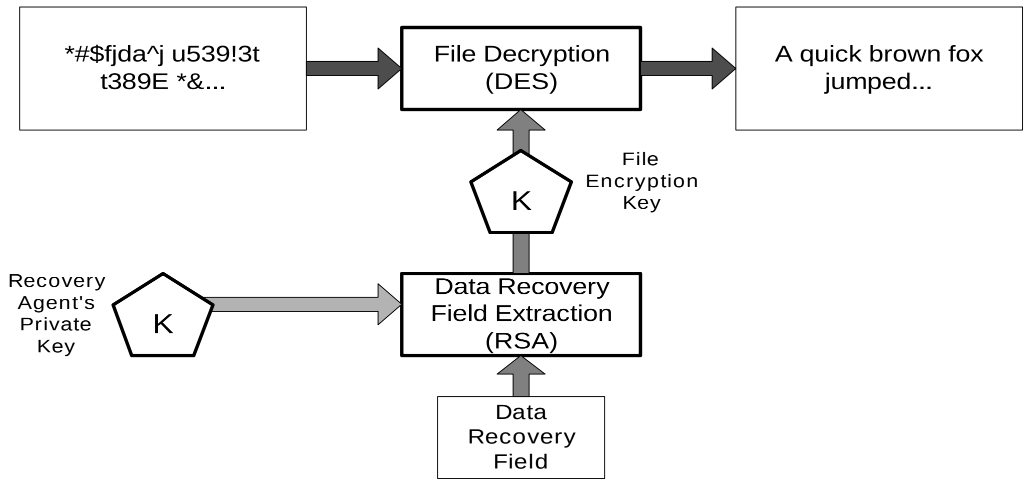
NOTE: *The figure shows only one user and one recovery agent¾this can actually be a list of users and a list of recovery agents with independent keys.*

The decryption process:


*#$fjda^j u539!3t t389E *&...
File Decryption (DES)
A quick brown fox jumped...
K
File Encryption Key
User's Private Key
K
Data Decryption Field Extraction (RSA)
Data Decryption Field

A user’s private key is used to decrypt the FEK using the corresponding encrypted FEK item in the DDF. The FEK is used to decrypt file data reads on a block by block basis. Random access to a large file will decrypt only the specific blocks read from disk for that file. The entire file does not have to be decrypted.

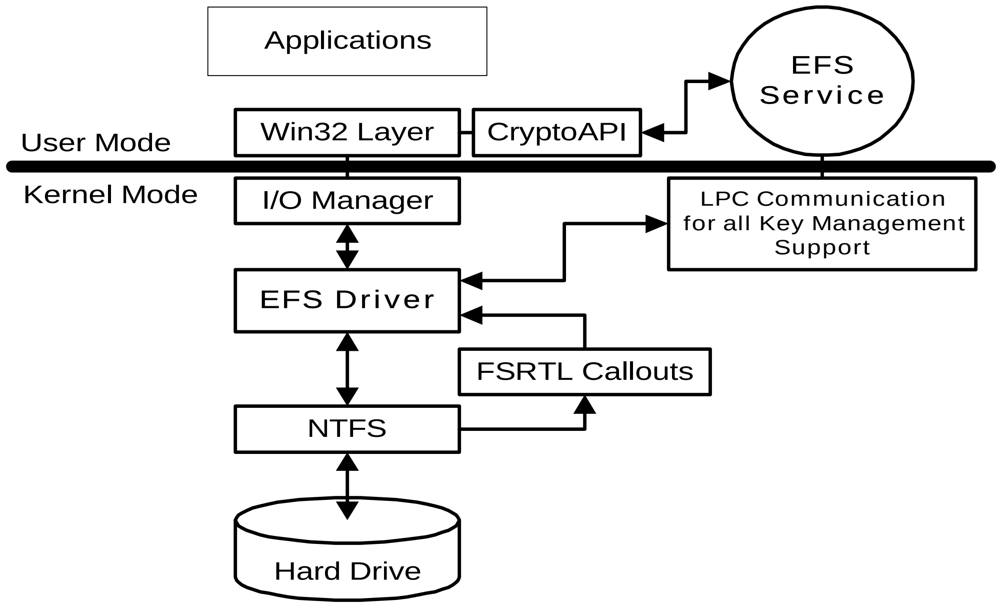
The recovery process:


*#$fjda^j u539!3t t389E *&...
File Decryption (DES)
A quick brown fox jumped...
K
File Encryption Key
Recovery Agent's Private Key
K
Data Recovery Field Extraction (RSA)
Data Recovery Field

The process is similar to decryption except that it uses the recovery agent’s private key to decrypt the FEK in the DRF. This simple scheme provides a strong encryption technology and the ability to let multiple users share an encrypted file, as well as allowing multiple recovery agents the ability to recover the file if so required. The scheme is fully algorithm agile and any cryptography algorithms can be used for various encryption phases. This will be very important as new and better algorithms are invented.

Implementation

EFS architecture is shown in the figure below.



EFS consists of the following components in the Windows 2000 operating system:

EFS driver. The EFS driver is logically layered on top of NTFS. It communicates with EFS service (running as part of security subsystem) to request file encryption keys, DDFs, DRFs, and other key management services. It passes this information to the EFS file system run-time library (FSRTL) to perform various file system operations (open, read, write, and append) transparently.

EFS FSRTL. File System Run Time Library (FSRTL) is a module within the EFS driver that implements NTFS call-outs to handle various file system operations such as reads, writes, and opens on encrypted files and directories as well as operations to encrypt, decrypt, and recover file data when it is written to or read from disk. Even though, the EFS driver and FSRTL are implemented as a single component, they never communicate directly. They use the NTFS file control callout mechanism to pass messages to each other. This ensures that NTFS participates in all file operations. The operations implemented using the file control mechanisms include writing the EFS attribute data (DDF and DRF) as file attributes and communicating the FEK computed in the EFS service to FSRTL such that it can be set up in the open file context. This file context is then used for transparent encryption and decryption on writes and reads of file data from disk.

EFS service. The EFS service is part of the security subsystem. It uses the existing LPC communication port between the Local Security Authority (LSA) and the kernel-mode security reference monitor to communicate with the EFS driver. In user mode it interfaces with CryptoAPI to provide file encryption keys and generate DDFs and DRFs. The EFS service also provides support for Win32® APIs, which are programming interfaces for encryption, decryption, recovery, backup and restore. These Win32 APIs support remote encryption, decryption, backup and restore operations

Win32 APIs. These provide programming interfaces for encrypting plaintext files, decrypting or recovering plaintext files, and importing and exporting encrypted files (without decrypting them first). These APIs are supported in a standard system DLL, advapi32.dll.

Policy Enforcement

Even though the basic technology behind EFS is straightforward as explained by previous sections, its policy infrastructure is not. The technology provides several policy driven services to ensure security, assurance, reliability and ease of use.

Polices include:

Recovery Policy Enforcement

Encrypted Data Recovery Policy consists of zero or more X509 version 3 certificates. The Key Usage of these certificates must be “File Recovery”. The private keys corresponding to these certificates are held by individuals who are issued the certificates. These individuals are referred to as “Recovery Agents”.

A recovery policy can be enforced at any of the following scopes of influence:

- Domain
- Organizational Unit
- Individual Computer

At each scope, the policy applies to all computers in that scope. For example, a recovery policy configured at a particular OU applies to all computers under that OU.

NOTE*: Policy applies to computers and not users. The encrypted data is stored on computers irrespective of who encrypted it—therefore recovery agents are based on organization of the computer and not users.*

By default, a recovery policy will be configured at the domain, so it applies to all computers in an Active Directory-based Windows 2000 domain. The domain “administrator” account is the default recovery agent. Similarly, a default policy is configured at the computer that is not joined to the domain. The local “administrator” account is the default recovery agent.

A policy with zero recovery certificates turns off EFS on computers under the corresponding policy. Note that zero recovery certificates policy is distinct from no policy, where no policy implies “don’t care” and is therefore interpreted as that each computer can have a locally defined policy.

A recovery policy with any *invalid* certificate is considered invalid as a whole and EFS is turned off for any new encryption. Note that existing encrypted files can still be decrypted. An invalid certificate is based on policy described next.

EFS also does policy enforcement each time encrypted file is opened. The existing recovery information is checked to ensure that it is based on current policy. If it is not, new recovery information is generated for the file. This keeps the recovery information on all active files up-to-date. To perform recovery, one can query the information about recovery agents and provide the file to any one of them to perform recovery.

Certificate Validity Enforcement

All certificates, recovery or user, are checked for validity when used. A valid certificate is one that is:

- Not expired.
- Not revoked.
- Has correct key usage
- Where certificate chain evaluation results in a trusted root certificate and each intermediary certificate authority certificate is trusted to issue certificates with appropriate key usage.

The only exception to this validity rule is a self-signed certificate. Self-signed certificates are accepted as valid as long as they have not expired.

Recovery policy is valid only if all recovery certificates are valid. Encrypting files is only allowed if user has a valid EFS certificate. EFS provides a transparent, zero-administration usage to end users. This is accomplished by automatically renewing certificates for users. In a corporate environment, EFS will use a configured Certificate Authority to obtain a certificate. If there is no CA configured, a self signed certificate will be generated and used.

Similar to how recovery information is re-generated if it is invalid or not current, user information is also re-generated if user’s certificate is invalid or is changed for any reason. This ensures that frequently used files are always current with respect to encryption information.

EFS Security Considerations

Encrypting File System is a strong security technology for physical protection of stored data. To that end, it is necessary to look at its various features and do a security analysis. In this section we look at various threats and how EFS handles them:

- Attempt to open other users’ encrypted files—EFS is designed to be transparent under the normal mode of operation. When a user attempts to open a file encrypted by another user, EFS attempts to locate the private key which will decrypt the FEK during the open. Since the calling user will not possess the key, FEK will not get decrypted and hence the attempt will failed with “Access Denied”.

- Attempt to bypass recovery policy—EFS does not allow any new encryption of files/folders if there is no recovery policy. If the machine is joined to a domain, the EFS policy is propagated from the domain as part of Group Policy and enforced by EFS on the machine. A local administrator's attempt to define a local EFS policy also does not work because policy from the domain takes precedence. The only option for a local administrator on the machine would be to remove the machine from the domain—doing so will no longer allow users to logon to the machine using domain credentials
- Attempt to destroy recovery policy—A local administrator may attempt to locate the EFS policy storage and attempt to delete or replace it. Deletion will not help because that will disable EFS. Replacing EFS with another recovery policy will not work because it will soon be overwritten by policy from domain.
- Physically access to the media—An individual with physical access to the machine could potentially attempt sophisticated attacks by going to the disk directly. Attempts to read the data this way will fail because it is encrypted and a successful process would require implementing EFS itself. Another possible attack with physical access can be to invalidate or delete the recovery portion on the encrypted file. This will not still not work because EFS will automatically recreate the recovery information when the file is successfully opened next time.
- Recovery from fatal failures during encryption/decryption operations—EFS also incorporates a *crash recovery* scheme whereby no data is lost in the event of a fatal error such as system crash, disk full, or hardware failure. This is accomplished by creating a plaintext backup of the original file being encrypted or decrypted. Once the original is successfully encrypted or decrypted, the backup is deleted. OTE: Creating a plaintext copy hasthe side-effect that the plaintext version of the file may exist on the disk, until those disk blocks are used by NTFS for some other file. For this reason, it is recommended that it is always better to start by creating an empty encrypted folder and creating files directly in that folder. Doing so, ensures that plaintext bits of that file never get saved anywhere on the disk. It also has a better performance as EFS does not need to create a backup and then delete the backup, etc.
- Handling recovery policy changes—As discussed above, a user with physical access to the machine may attempt to scramble the recovery information on the file. It is also possible that recovery policy is changed by administrators at the domain because of various reasons such as the expiration of certificates, change of recovery agent, and so forth. When a particular encrypted file is opened, EFS will check whether the recovery information on the file is current. If not, it is recomputed. This is because recovery information for the file can not be updated without a decrypted FEK which becomes available only when the file is opened. Encrypted files that are not touched for long periods of time may have stale recovery policy, it is therefore very important that recovery certificates and private keys be maintained for several years even after the recovery policy has changed.
- Handling user certificate or key changes—Just like the recovery policy changes, user certificate or key changes are handled when a particular file is opened. EFS determines if the key used to open the file is current. If not, the data decryption field is updated on the file using the user's current key. Note, that recovery agents should also continue to hold on to there old keys unless they are sure that all encrypted files have started using the new key. However, users can be more relaxed than because users can depend on recovery agents to decrypt their data in case they lose or destroy keys.
- Protecting the system from becoming unbootable—Another important piece to understand is that EFS is intended to encrypt or decrypt user data. System data such as the registry, system DLLs and other files needed during system boot up must never be encrypted because EFS doesn't become active until the operating system is running. Therefore, if any of the files used by the operating system are encrypted, the system will be rendered useless. EFS provides some level of protection by disallowing encryption of files or folders with system attribute designations.

EFS Application Programming Interfaces

EFS provides the following API set to expose its features. These APIs are used by various tools like Explorer, Cipher, NTBackup, EDRP Policy snap-in that expose EFS capabilities to end users and administrators.

EncryptFile

```
BOOL
EncryptFile(
    LPCTSTR lpFileName
    );
```

EncryptFile encrypts a plaintext file represented by lpFileName. The file may be local or remote.

DecryptFile

```
BOOL
DecryptFile(
    LPCTSTR lpFileName,
    DWORD    dwReserved
    );
```

DecryptFile decrypts an encrypted file represented by lpFileName. The file may be local or remote.

FileEncryptionStatus

```
BOOL
FileEncryptionStatus(
    LPCTSTR lpFileName,
    LPDWORD  lpStatus
    );
```

FileEncryptionStatus returns TRUE if the file is encryptable. A file is not encryptable if it is not on the NTFS version 5 file system, if it is marked System, and so forth.

QueryUsersOnEncryptedFile

```
DWORD
QueryUsersOnEncryptedFile(
     IN LPCTSTR lpFileName,
     OUT PENCRYPTION_CERTIFICATE_HASH_LIST *
      pUsers
    );
```

QueryUsersOnEncryptedFile returns the information on the list of users who can decrypt the file represented by lpFileName. The information returned contains a security identifier (SID) of users (if available), user’s name from the certificate that was used and a thumbprint of the certificate.

QueryRecoveryAgentsOnEncryptedFile

```
DWORD
QueryRecoveryAgentsOnEncryptedFile(
     IN LPCTSTR lpFileName,
     OUT PENCRYPTION_CERTIFICATE_HASH_LIST *
      pRecoveryAgents
    );
```

QueryRecoveryAgentsOnEncryptedFile returns the information on the list of recovery agents who can recover the encrypted file represented by lpFileName. The information returned contains a SID of recovery agents (if available), their names from the certificates and the thumbprint of the certificates.

RemoveUsersFromEncryptedFile

```
     DWORD
     RemoveUsersFromEncryptedFile(
          IN LPCTSTR lpFileName,
          IN PENCRYPTION_CERTIFICATE_HASH_LIST
           pHashes
         );
```

RemoveUsersFromEncryptedFile allows the caller to remove one or more users from the list of users who can decrypt the file. The caller must be able to decrypt the file in order to successfully perform this operation.

AddUsersToEncryptedFile

```
     DWORD
     WINAPI
     AddUsersToEncryptedFile(
          IN LPCTSTR lpFileName,
          IN PENCRYPTION_CERTIFICATE_LIST pUsers
         );
```

AddUsersToEncryptedFile allows the caller to add one or more users to the list of users who can decrypt the file.

SetUserFileEncryptionKey

```
     DWORD
     SetUserFileEncryptionKey(
         IN PENCRYPTION_CERTIFICATE
          pEncryptionCertificate
             );
```

SetUserFileEncryptionKey allows the user to change the certificate or private key that is used by EFS to encrypt new files or update existing files. Normally, EFS automatically handles cases where user doesn't have a key setup or if the certificate is expired. This is done by transparently generating a key pair for the user and getting it certified. In certain cases, such as compromise of a key or if it is lost, user may want to change their key.

FreeEncryptionCertificateHashList

```
     VOID
     FreeEncryptionCertificateHashList(
         IN PENCRYPTION_CERTIFICATE_HASH_LIST pHashes
         );
```

FreeEncryptionCerttificateHashList allows the caller to free memory allocated during the Query APIs.

In addition to the basic APIs described above, EFS also provides four APIs for backup/restore purposes. These APIs are for the Windows 2000 Release ONLY. Applications that use them will need to handle the rewrite for subsequent releases where these APIs will be encapsulated into the planned comprehensive backup and restore APIs.

OpenRaw

```
        DWORD
        OpenRawW(
            LPCTSTR        lpFileName,
            ULONG           ulFlags,
            PVOID *         pvContext
            );
```

EFS provides fortransparent normal file operations like open, read, write. Therefore, in order to support the capability where the file may be opened to read encrypted bits for back up purposes, this new open API is provided. Because backup operators are not expected to possess private keys to decrypt every file, it is important that they be able to back up files in encrypted form itself. OpenRaw allows backup operators to open the file in this special mode and setup a context for subsequent APIs.

ReadRaw

```
typedef
DWORD
(*PFE_EXPORT_FUNC)(
    PBYTE pbData,
    PVOID pvCallbackContext,
    ULONG ulLength
    );
DWORD
ReadRaw(
    PFE_EXPORT_FUNC pfExportCallback,
    PVOID           pvCallbackContext,
    PVOID           pvContext
    );
```

ReadRaw allows the caller to read all the encrypted streams on the opened file (using OpenRaw) in an opaque format. The caller provides a *export function* which is used by the API to return the opaque serialized data stream to the caller. This call returns only when all data has been provided to the caller using the supplied export function.

WriteRaw

```
typedef
DWORD
(*PFE_IMPORT_FUNC)(
    PBYTE pbData,
    PVOID pvCallbackContext,
    ULONG ulLength
    );
DWORD
WriteRaw(
    PFE_IMPORT_FUNC pfImportCallback,
    PVOID           pvCallbackContext,
    PVOID           pvContext
    );
```

WriteRaw allows the caller to write back all the opaque serialized data stream created using an earlier call to ReadRaw to recreate the original file. The caller provides an *import function* which is used by the API to obtain the opaque serialized data stream to the caller and restore the original encrypted file. This call returns only when entire file is restored or there is a failure. There may be multiple calls to the import function from within this function.

CloseRaw

```
VOID
CloseRaw(
    PVOID           pvContext
    );
```

CloseRaw is the cleanup API that allows EFS to cleanup the context after the file has been backed up or restored.

SUPPORT ISSUES WITH EFS

EFS provides data recovery to authorized recovery agents. The data recovery architecture is part of Microsoft's effort to meet current encryption export policy regulations and provide stronger than 40-bit encryption to our international customers. Towards this effort, EFS uses the standard DESX encryption algorithm, which is based on a 128-bit encryption key. EFS is designed to support different encryption algorithms with varying key strengths for future enhancement.

Windows 2000 products for the North American market use the full 128-bit DESX encryption. Files that are encrypted using the 40-bit version of EFS may be restored and used with EFS versions that support the 128-bit DESX. However, files encrypted using the 128-bit version of EFS will not be restorable into EFS versions restricted to 40-bit DESX to ensure U.S. export regulations are met. In the future, when the regulations allow export of stronger cryptography, customers worldwide will be able to migrate transparently and use new and stronger encryption algorithms with EFS.

- EFS in Windows 2000 provides users the ability to encrypt NTFS directories using a strong public key-based cryptographic scheme whereby all files in the directories are encrypted. Individual file encryption though supported, is not recommended because of unexpected behavior of applications.
- EFS also supports encryption of remote files accessible via file shares. If users have roaming profiles, the same key and certificate may be used on certain trusted remote systems. On others, local profiles are created and local keys are used.
- EFS provides enterprises the ability to set up data recovery policies such that data encrypted using EFS can be recovered when required.
- The recovery policy is integrated with overall Windows 2000 Security policy. Control of this policy may be delegated to individuals with recovery authority. Different recovery policies may be configured for different parts of the organization.
- Data recovery in EFS is a contained operation. It only discloses the recovered data, not individual user's key that was used to encrypt the file.
- File encryption using EFS does not require users to decrypt and re-encrypt the file on every use. Decryption and encryption happens transparently on file reads and writes to disk.
- EFS supports backup and restore of encrypted files without decryption. NtBackup supports backup of encrypted files.
- EFS is integrated with the operating system such that it stops the leaking of key information to page files and ensures that all copies of an encrypted file, even if moved, are encrypted.
- The North American version of EFS will use DESX as the file encryption algorithm with full 128-bit key entropy. The international version of EFS will also use DESX as the encryption algorithm, however the file encryption key will be reduced to have only 40-bit key entropy.
- Several protections are in place to ensure that data recovery is possible and there is no data loss in case of total system failures.