Global Information Assurance Certification Paper

GIAC Securing Windows 2000 ® Curriculum
Practical Assignment for SANS Baltimore 2001

Robert CURRIE

Windows 2000 ® Encrypting File System

Table of Contents

Introduction

Today's average office worker carries a laptop computer to and from their place of work for increased productivity. The productive worker often has many confidential files on their laptop hard disk critical to the well being of the entity that employs them. If these files were to fall into the wrong hands, results could be embarrassing, disastrous or both depending on whether the files belong to a fast food chain manager or an intelligence agency analyst. Laptops are easy prey for well-organized groups that target airports and busy office buildings as hunting grounds. Files are not only vulnerable on mobile targets but could be obtained from backup media or unprotected workstations/servers as well. The common element is physical access to the target system. There are tools readily available that bypass the protection offered from the Windows NT ® operating system allowing unchallenged access to file system.

A traditional remedy to protecting confidential data is encryption. There are many products available that provide application-level file encryption using password-derived keys. This approach to encryption implementation is not without its limitations as detailed in "Encrypting File System for Windows 2000 ® (Microsoft ® white paper)".

> **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.

Windows 2000 ®'s improved NTFS Version 5 comes with a new feature that can mitigate the above risks. The Encrypting File System (EFS) provides transparent file and folder level encryption of confidential files. So transparent that a user might not even realise they are using EFS. If someone does steal your confidential EFS protected data, they must be willing to overcome the encryption algorithm. This is often not feasible for the average thief. Again this depends who you work for ☺

This document will describe how EFS works, how to implement EFS properly as well as some guidelines for the safe use of this new security feature with common usage examples (encryption, decryption and recovery procedures). The author's research discovered some contradictions regarding EFS implementation. Every effort has been made to deliver the most accurate and current data regarding EFS.

How EFS works

There are a few essential concepts that key to understanding EFS. Firstly, we need a clear definition of encryption.

> *Encryption* is the transformation of data into a form that is as close to impossible as possible to read without the appropriate knowledge (a key). Its purpose is to ensure privacy by keeping information hidden from anyone for whom it is not intended, even those who have access to the encrypted data. *Decryption* is the reverse of encryption; it is the transformation of encrypted data back into an intelligible form. [RSA Security]

The next element we need to understand is how Public Key Infrastructure (PKI) works. PKI is the use of two mathematically related keys. Either of these keys can be used to encrypt clear text that only the other key can decrypt. Knowledge of one of the keys is

not enough to mathematically derive the other key or it is at least considered computationally infeasible. PKI implementation dictates that one of the keys is made public and the other is kept private and secure. These keys can be used for different cryptographic operations such as authentication, encryption, or digitally signing.

A simple PKI starts with a Certificate Authority (CA). The CA issues public/private key pairs for a requesting entity that trusts the CA. The CA also binds the requestor's credentials to their public key in the form of a digital certificate. These credentials are a set of descriptors for the requestor such as name, organisation, country, e-mail or others. The CA assures that the public key in the certificate truly belongs to the owner (original requestor). The certificate is composed of the user's public key, credentials and a hash of the public key and the credentials. The hash is encrypted with the CA's private key. Windows 2000 ® uses industry standard X.509v3 certificates. For more information on how to implement PKI in Windows 2000 ®, Microsoft ®'s "Step-by-Step Guide to Setting up a Certification Authority" is an excellent reference.

We now realize that before any cryptographic operations can take place we need a key. A cipher is then used to transform the data from plain text to cipher text with assistance of the key. The cipher is the mathematical algorithm responsible for the transformation. In general there are two kinds of ciphers; symmetric and asymmetric. The symmetric or secret cipher as it is often referred uses the same key transform the plain text to cipher text and back to plain text.

> Symmetric cipher(clear text, key) = cipher text
> Symmetric cipher(cipher text, key) = clear text

An asymmetric cipher uses a public key to transform the data from plain text to cipher text and a private key to reverse the operation.

> Asymmetric cipher(clear text, public-key) = cipher text (private key is decryptor)
> Asymmetric cipher(clear text, private-key) = cipher text (public key is decryptor)

The second equation might throw some reader's off at first glance. The public key and private key pair is related mathematically in that one is a factor of the other. By convention we would never do this but mathematically it is correct. Try this at home ☺ Encrypt some plain text with a private key. The public key will decrypt it. Again, this is not done in practice. As previously mentioned, the public key resides in the user's certificate. Where is this private key kept? Currently, Windows 2000 ® Crypto API supports the private key storage in either the operating system protected store (software based) or offline in a smart card. The following figure # 2 shows the private keys for user S-1-5-21-1801674531-1383384898-854245398-1000 stored in their profile (C:\Documents and Settings\rcurrie.ARVIAT\Application Data\Microsoft\Crypto\RSA\S-1-5-21-1801674531-1383384898-854245398-1000). The private keys are encrypted in a very

complex fashion that goes beyond the scope of this paper.

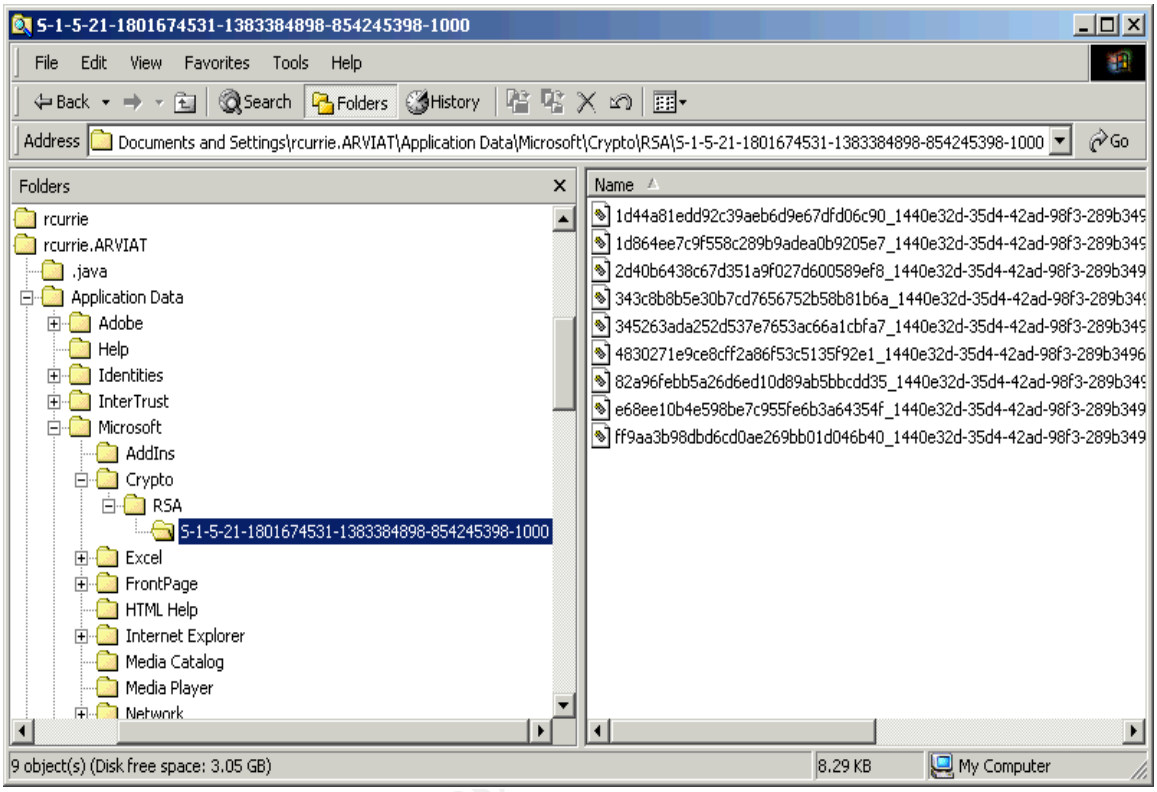

Figure 1 - User's local private key store

The numeric names of the system files correspond to the certificate where the public key is stored (C:\Documents and Settings\rcurrie.ARVIAT\Application Data\Microsoft\SystemCertificates\My\Certificates). They are actually the SHA thumbprint of the certificate. Opening the corresponding private key with a text editor reveals this thumbprint value.

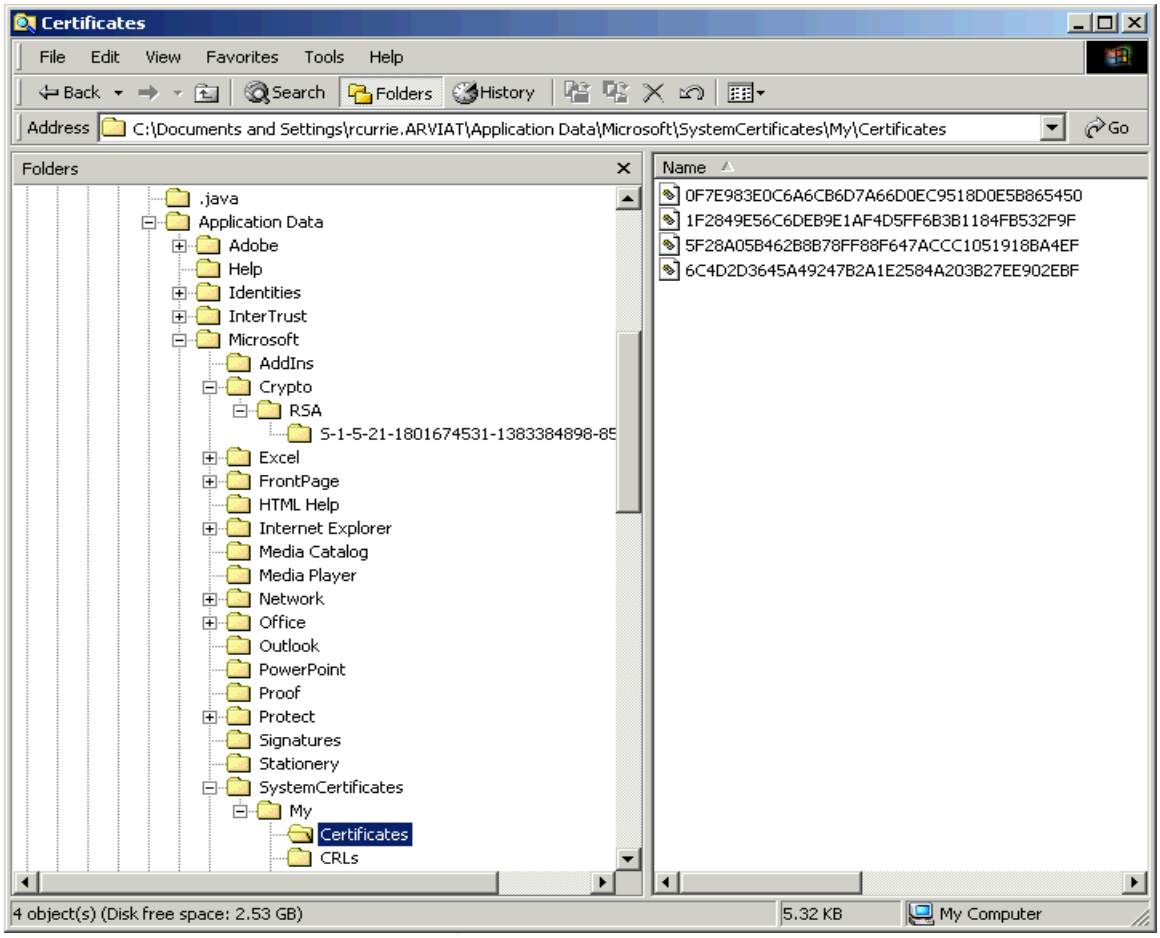

Figure 2 - User's local certificate store

Secret ciphers are considerably faster (factor varies from approx. 100 to 1000) than their public key counterparts but they require a protected key distribution system for secure implementation. As we shall see, EFS leverages a combination of both ciphering algorithms categories to maximize performance and security.

As previously mentioned EFS is only available on volumes formatted with NTFSv5. EFS as we shall see can be applied to either a specific file or a folder. When applied to the folder all files within the folder are encrypted. What exactly happens during this encryption process?

When a system is configured to use EFS, each file is encrypted using a randomly generated key, called the file encryption key or FEK. The FEK is independent of a user's public/private key pair. The current release of EFS only uses the symmetric cipher DESX. Microsoft ®'s documentation indicates development of other encryption ciphers for future releases. This randomness is necessary to thwart many forms of cryptanalysis-based attack on the encrypted files. If the FEKs weren't random, every

encrypted file would provide another cipher text sample. The samples could be compared with the knowledge that the FEKs are the same, weakening seriously the cryptographic implementation. The FEK is then encrypted by a public key taken from the encrypting user's certificate (located in the encrypting user's profile).

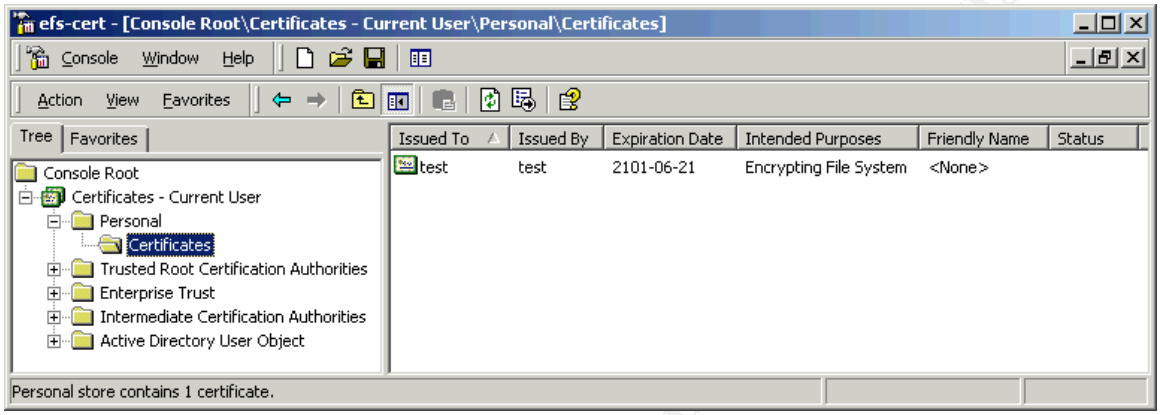

Figure 3 – User's certificate for EFS operations

The encrypted FEK is embedded in the encrypted file's header and is unique to it. Its exact location is the Data Decryption Field (DDF). To decrypt the FEK, EFS uses the users private key. The FEK is also encrypted by at least one other public key, that of the default recovery agent. This encrypted FEK is also embedded in the file. Its exact location is the Data Recovery Field (DRF). There can be multiple recovery agents for any given file. There is a DRF for each recovery agent's encrypted FEK. These recovery agents' role will be examined shortly.

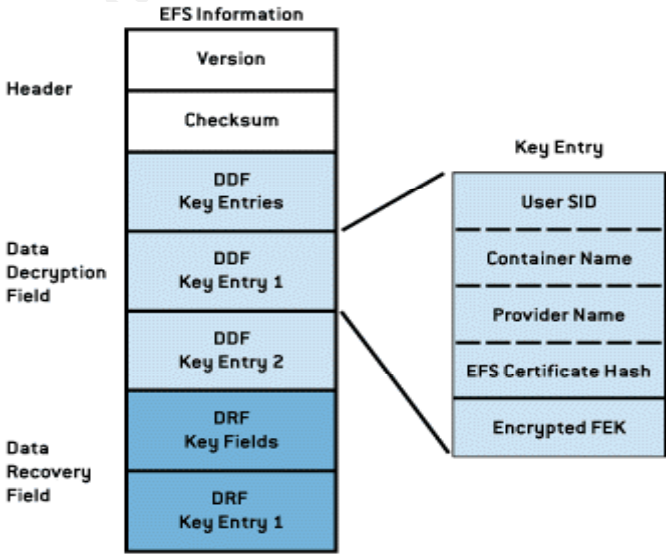

Figure 4 – EFS file header structure

The checksum ensures there is no logical "damage" to the header fields either intentional or otherwise. EFS operations verify validity of header components against this value. The header can even be rebuilt in some instances where there is a discrepancy e.g. Invalid DRF entries.

Figure # 5 shows what an encrypted file header looks like on disk. We can see the EFS Certificate Hash, Provider Name and Container Name.

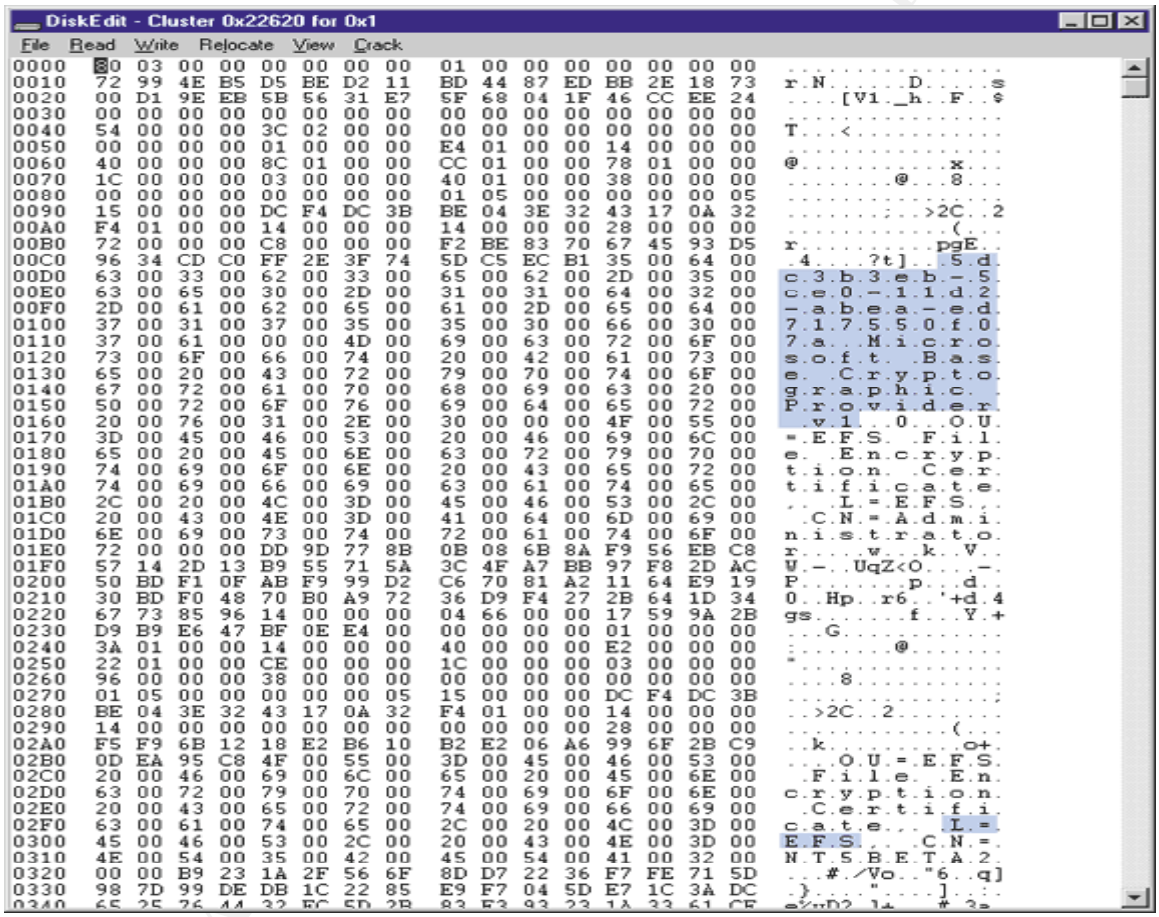

Figure 5 – Raw on disk view of EFS structures

This is a good opportunity to indicate the protection offered by this key and cipher combination. DESX was developed by RSA Data Security in an effort to strengthen DES against cryptanalytic attacks. It is considered to be much stronger than DES or Triple DES. The other important factor to consider in the "protection" equation is key length. The DESX cipher uses either 56 or 168 bit keys by design. Microsoft ® documentation indicates that domestic key length is 40 bits and can be increased to 128 bits with the installation of the High Encryption Pack. These key lengths are padded from 40 to 56 or 128 to 168 depending of course whether the High Encryption Pack is

installed. It is important to understand that the "padding" renders the effective key length to 40 or 128 bits. In a recent experiment sponsored by RSA Data Security, a 40-bit encoded message was cracked in approximately eight hours. A 128-bit message is 309,485,009,821,345,068,724,781,056 times harder to unscramble than a 40-bit message. If the technology applied to crack the 40-bit message in eight hours were applied to break a 128-bit message, it would take more than two trillion years.

Now that we have a basic understanding of the EFS encryption process we can examine the finer details. Marc RUSSINOVICH's article "Inside Encrypting File System, Part 2" details the inner workings of this process. Author's entries in reference are italicized.

1) The user profile loads to the Registry, if necessary.

2) EFS creates a log file named efsX.log in the System Volume Information subdirectory. X is a unique number in the filename (e.g., efs0.log). EFS writes to the log file when performing subsequent steps in the encryption process so that EFS can recover the file in case of system failure during the encryption process.

3) Microsoft ® Base Cryptographic Provider generates a random 128-bit *(this assumes High Encryption Pack is installed)* FEK for the file.

4) EFS reads the HKEY_CURRENT_USER\Software\Microsoft ®\Windows NT\CurrentVersion\EFS\CurrentKeys\CertificateHash Registry value to identify the user's public key/private key pair.

5) EFS creates a DDF key ring with an entry for the user and associates the key ring with the file. The entry contains a copy of the FEK that the user's EFS public key encrypted.

6) EFS creates a DRF key ring for the file with an entry for each Recovery Agent on the system. Each entry contains a copy of the FEK that the Recovery Agent's EFS public key encrypted.

7) EFS creates a backup file, efsX.tmp, in the directory in which the file undergoing encryption resides. X is a unique number in the filename (e.g., efs0.tmp).

8) EFS places the DDF and DRF key rings in a header and adds the header to the file as the file's EFS attribute.

9) EFS marks the backup file as encrypted and copies the original file to the backup file.

10) EFS destroys the original file's contents and copies the backup to the original file. The copy operation results in the data's encryption, because the backup file is marked as encrypted.

11) EFS deletes the backup file.

12) EFS deletes the log file.

13) The user profile unloads from the Registry if it loaded in step 1.

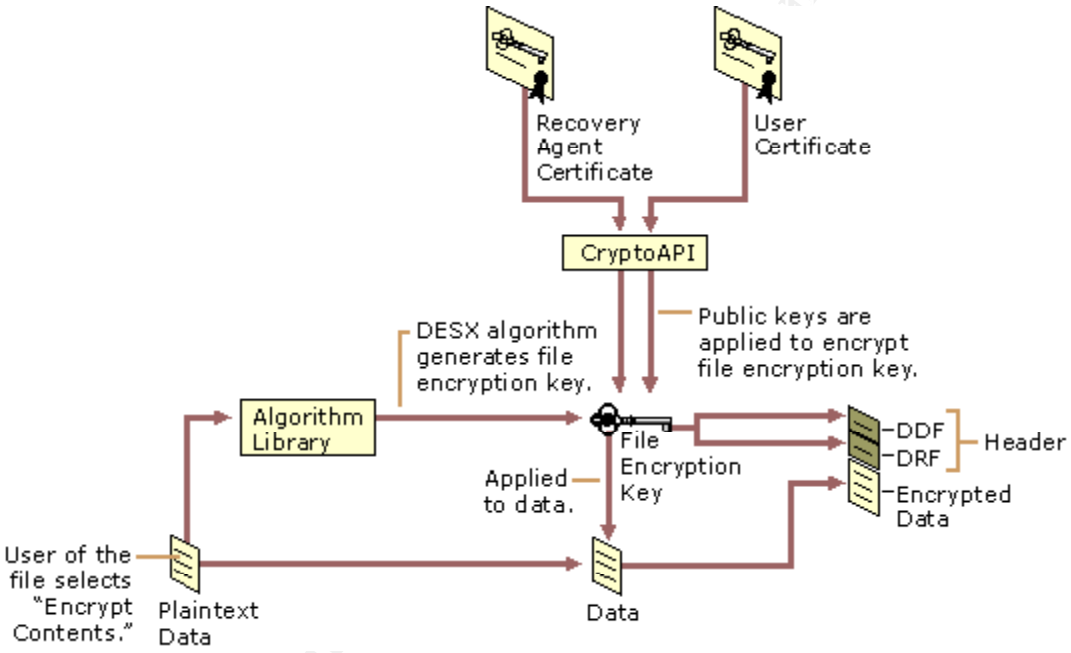

Figure 6 – Recap of encryption process

At this point, an inquisitive mind might be thinking of possible ways to garner a FEK from artefacts in memory so as to compromise EFS. The EFS was designed to live in the Windows 2000 ® kernel as shown in figure # 7. Kernel mode operations use a non-paged memory pool to store FEKs, ensuring that they never make it to the paging file (typically C:\%systemroot%\pagefile.sys), which remains available on disk even after shutdown.

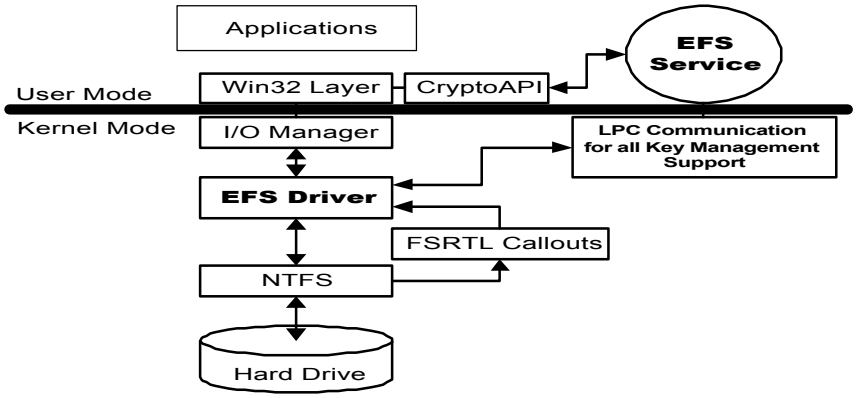

Figure 7 – O/S integration of EFS

Typical usage scenarios

We are now ready to discuss the inner working of EFS with the help of some concrete examples. Recognizing the liability unprotected confidential data poses, we decide EFS can help our IT environment – what happens next? For the remainder of this document the author will perform EFS operations on Windows 2000 ® Professional system ARVIAT. As previously mentioned, EFS will only work only NTFSv5 volumes.

Encrypting a Folder or File

On the test system I have created a folder C:\SECRET. I modified a sample file SECRET COMPANY INFO.PDF, which I copied to the folder C:\SECRET. This folder is no different than any other folder on the author's system at present. If we want to encrypt this file we have two choices. Encrypt the folder and therefore all its contents or just the file itself. Folder level encryption is the preferred method. Imagine for a moment many files in the same directory. The graphical user interface does not show whether individual files in a folder are encrypted or not (default Explorer view settings don't show attributes). It is easier for the end user to dedicate a folder for EFS protection and apply the encryption to the whole folder to ensure all designated data is protected. Also, many applications are not EFS aware and may manipulate files in such a fashion that plain text remnants are left behind in a folder where only file level EFS is implemented. Again, it is preferable to implement EFS at the folder level to mitigate this risk.

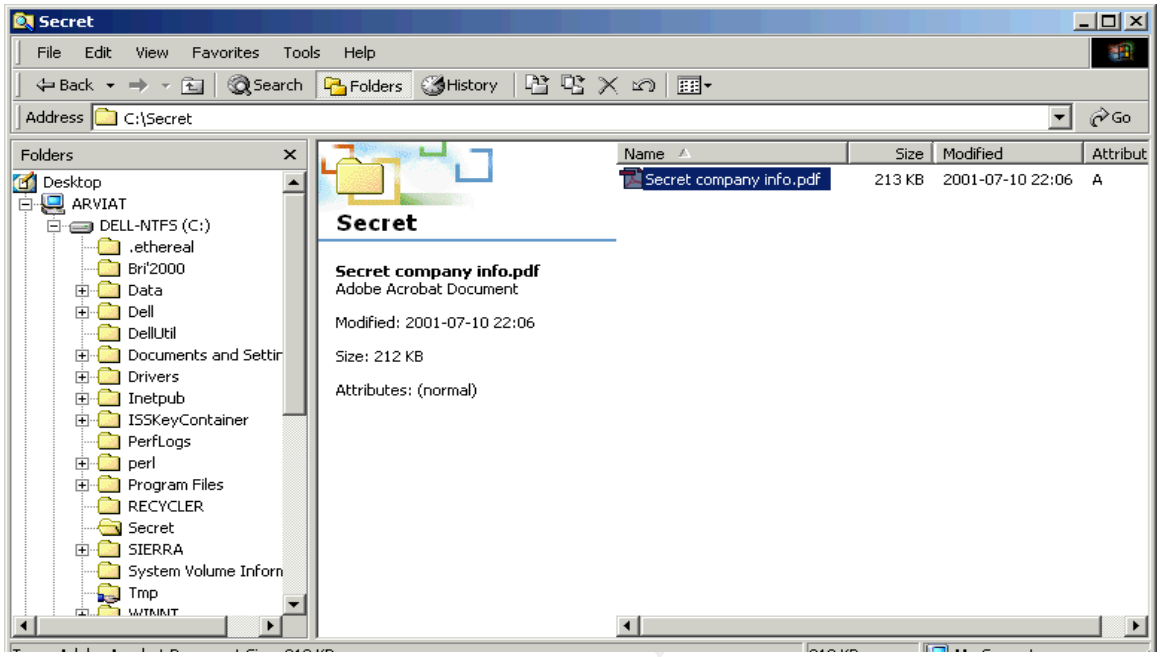

Figure 8 – Sample file for encryption operations

To implement EFS at the folder level for our sample directory, we open Explorer and select our folder C:\SECRET. Right-click and choose Properties, then Advanced. Check off the "Encrypt contents to secure data" box and then OK.

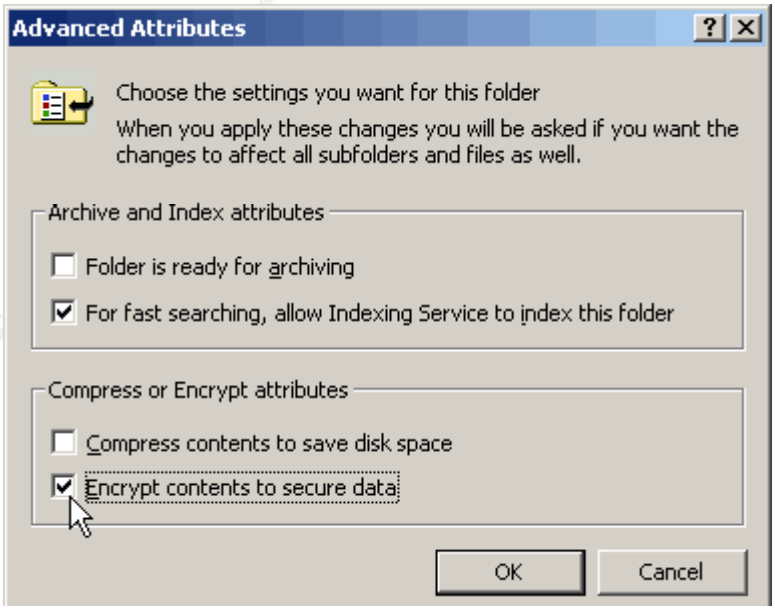

Figure 9 – Advanced attributes screen

After clicking OK, we are offered the choice to encrypt only this folder *or* this folder and its sub folders and files. For this example we will choose to encrypt the folder and its contents.

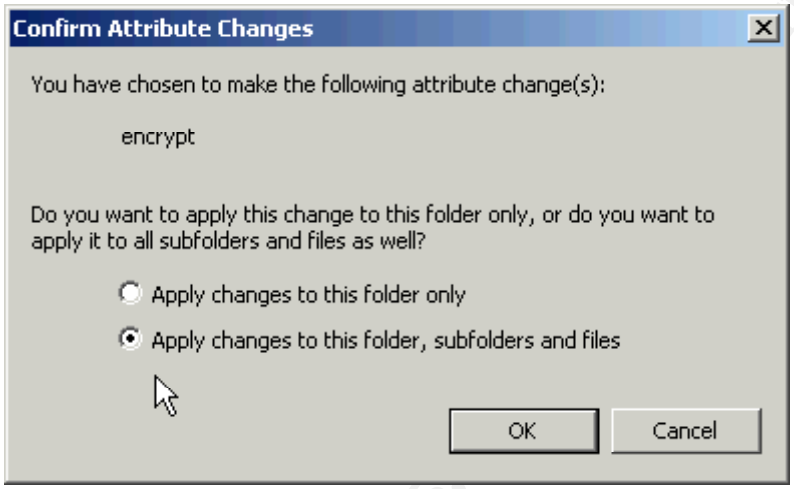

Figure 10 – Encryption scope option screen

After these changes are applied, graphically there is very little difference in the display of C:\SECRET or its contents. The size and the name of our original file are exactly the same as well. How can we tell if a folder is EFS protected or not? Right clicking the folder in question and selecting Advanced will present us with the "Encrypt contents to secure data" box, which should now be checked off already. As well, if the Explorer view options are set to show file attributes, an "E" will show up.

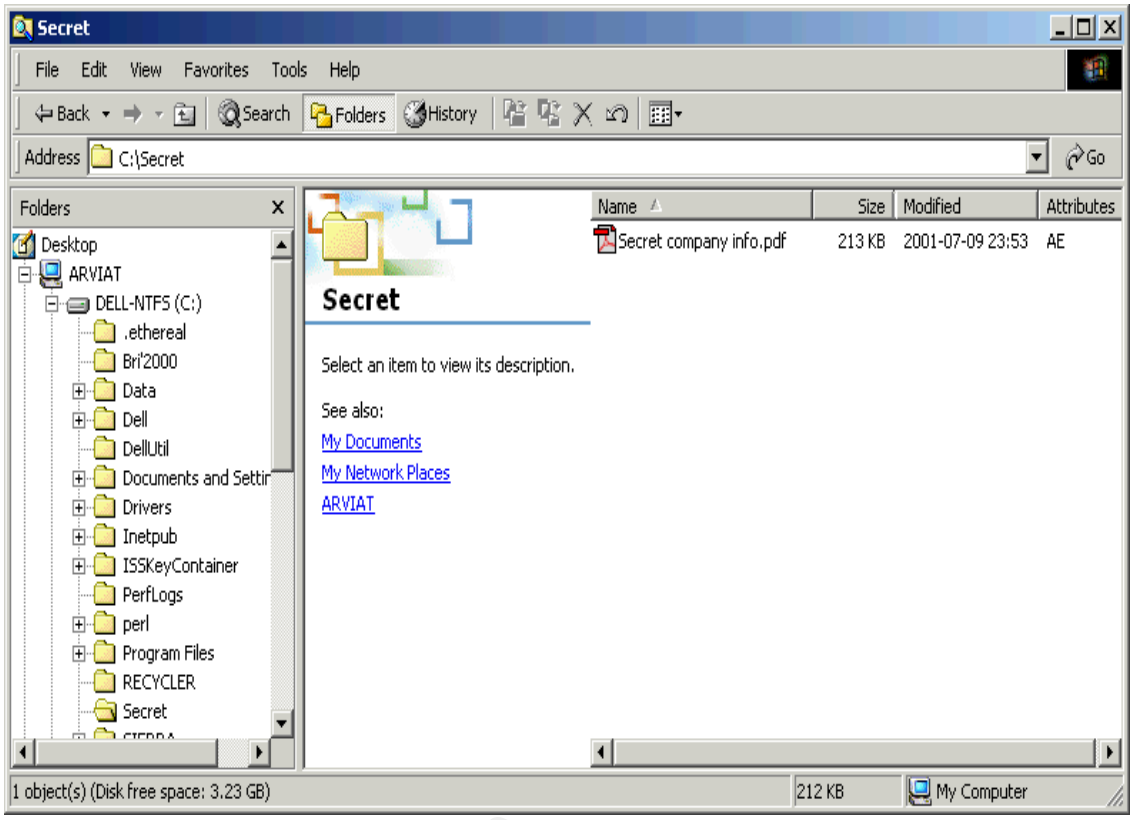

Figure 11 – Modified attributes of encrypted sample file

This is a good opportunity to explain the command line utility CIPHER, which can be found on any Windows 2000 ® platform. This early introduction to such a utility is necessary to assist explaining EFS behaviour. CIPHER offers more functionality than the graphical user interface. It can display or alter the encryption of directories and files on NTFS partitions. The CIPHER utility supports several commands, which are implemented through switches. The following details are available by executing the following command → CIPHER /?

/E Encrypts and marks the directories you specify so that EFS will encrypt new files you add.

/D Decrypts and marks the directories you specify so that EFS won't encrypt new files you add.

/S Performs the operation you specify in a given directory and all its subdirectories.

/A Performs the operation you specify on files and directories.

/I Continues performing the operation you specify even after errors occur. (By default, Cipher stops when it encounters an error.)

/F Forces the encryption operation on the directories you specify, including already-encrypted directories. (By default, Cipher skips already-encrypted directories.)

/Q Reports only the most essential information.

/H Displays file with the hidden (i.e., system) attributes. (By Default, Cipher omits these files from display.)

/K Creates a new file encryption key for you when you run Cipher. When you choose this option, Cipher ignores all other options.

Used without parameters, CIPHER displays the encryption state of the current directory and any files it contains. You may use multiple filenames and wildcards. The following diagram shows the root of our C:\ drive. The listing indicates all directories on C:\ and their current EFS status. This is a screen shot of drive C: on sample system ARVIAT after we implemented EFS on C:\SECRET. Notice the "U" for unencrypted folders and the "E" for our confidential folder in figure # 12.

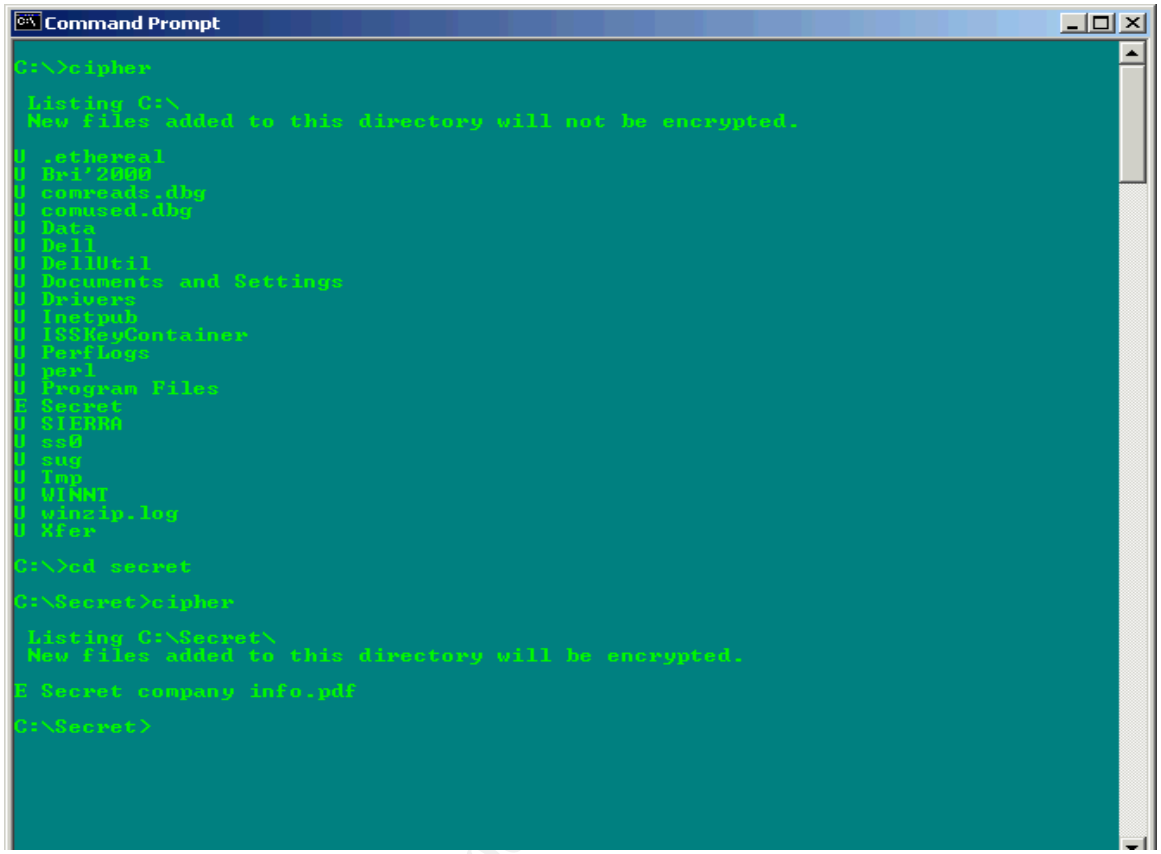

Figure 12 – CIPHER command line utility

CIPHER could have been used to encrypt our C:\SECRET folder and its contents by using the following command → C:\>CIPHER /E /A /S:"C:\SECRET". Care must be taken when using the command line utility CIPHER. If a user were to use CIPHER /E <directory> on a directory already containing files, CIPHER would report that the <directory> is encrypted and *new* files added to that it will be encrypted. The files that are already there don't change.

Now that we know how to encrypt folders and files, there are a few considerations worth mentioning. Special care should be taken not to encrypt system files that are required to boot the system. While a Windows 2000 ® system is booting, the encrypting user's private key is unavailable rendering the system files useless. EFS has built in safeguards that cause encryption attempts of system files to fail. DO NOT change the SYSTEM attribute in an attempt to bypass this safety feature. EFS will encrypt the files if it doesn't

detect the SYSTEM attribute. Microsoft ® has development plans for securing the boot process. Stay tuned ☺

Decrypting a Folder or File by owner

Under normal operations the user is not required to perform decryption operations. Using our example document C:\SECRET\ SECRET COMPANY INFO.PDF, the user who encrypted this document, may open it directly with Adobe Acrobat ®. He is not prompted for a password or any other authentication information. The document may be edited and resaved (re-encrypted). All reads and writes are intercepted by EFS where the decryption and re-encryption would take place.

There are situations where the user who encrypted the file might want to share the file with another user. In this case, manual decryption operations are required. If future releases of EFS support multiple DDFs in the encrypted file's header sharing will be possible (see figure # 4). As with encryption, we have two ways of manually decrypting; from graphical user interface or using the command line utility CIPHER.

For this example, we need to send a copy to co-worker for her input. Using Explorer, we can browse to our sample folder C:\SECRET where our confidential data is kept. Right click on the document "SECRET COMPANY INFO.PDF" and select Properties and then Advanced.

Figure 13 – Advanced attributes screen for decryption

From the Advanced Attributes dialogue box Uncheck "Encrypt contents to secure data". If our file was encrypted this box should already be checked off. Clicking OK and OK

again decrypts the file. The file can now be given to the co-worker as it is in plain text. This last operation left the folder C:\SECRET's EFS status the same. All subsequent files created in that directory will be encrypted.

If the goal was to decrypt the folder and the file(s) contained therein, the operation is almost the same. Instead of right clicking the file, we select the folder. Select Properties, Advanced Properties and uncheck the "Encrypt contents to secure data" box. Clicking OK and OK again presents us with the following dialogue box.

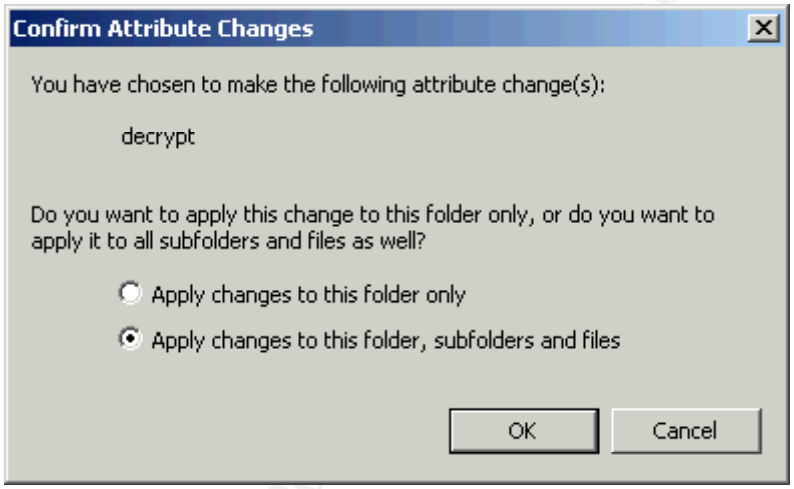

Figure 14 – Decryption scope option screen

To decrypt the folder and its contents, we will select the second choice, which is fairly explanatory and click OK. Our file is once again ready to be shared with our co-worker.

As previously mentioned we can use the command line interface to decrypt our files. In the following example, the author needs to decrypt our folder C:\SECRET. From the command line interface, CIPHER can be used to accomplish the same manual decryption by running the following command → CIPHER /D /A /S:"C:\SECRET"

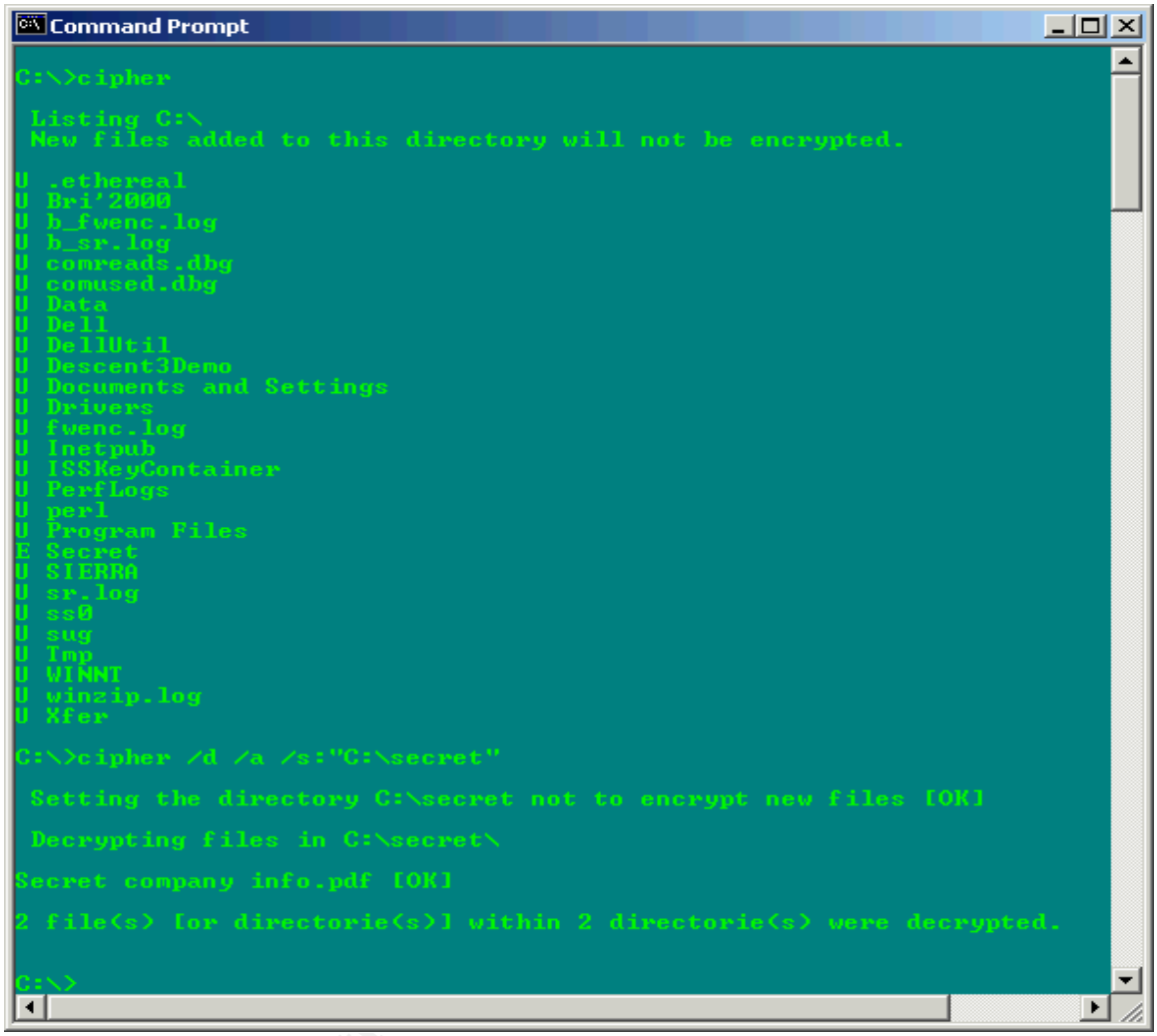

Figure 15 – Decryption using CIPHER utility

The decryption of FEKs is a costly. Marc RUSSINOVICH's article "Inside Encrypting File System, Part 2" states that in the process of decrypting a FEK, Crypto API uses results in more than 2000 Registry API calls and 400 file-system accesses on a typical system. The EFS driver, with the aid of NTFS, uses a cache to try to avoid this expense.

Decryption attempts by unintended users

If someone who does not have a private decryption key for the encrypted file attempts to access the file, they will be greeted with the following message. This particular message was generated on the sample system ARVIAT. The author logged onto the local machine with another user context. A double click of the file launched Adobe Acrobat ® but the open file process failed.

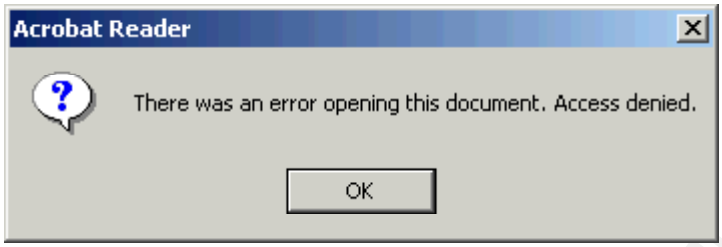

Figure 16 – Unauthorised decrypting user warning

A similar attempt was made from the command line. The author opened the command line interface, CMD.EXE, with Run As. User "test" attempted to decrypt a test text file with CIPHER. User "rcurrie" encrypted the file in question. The resulting error message indicates Access is Denied.

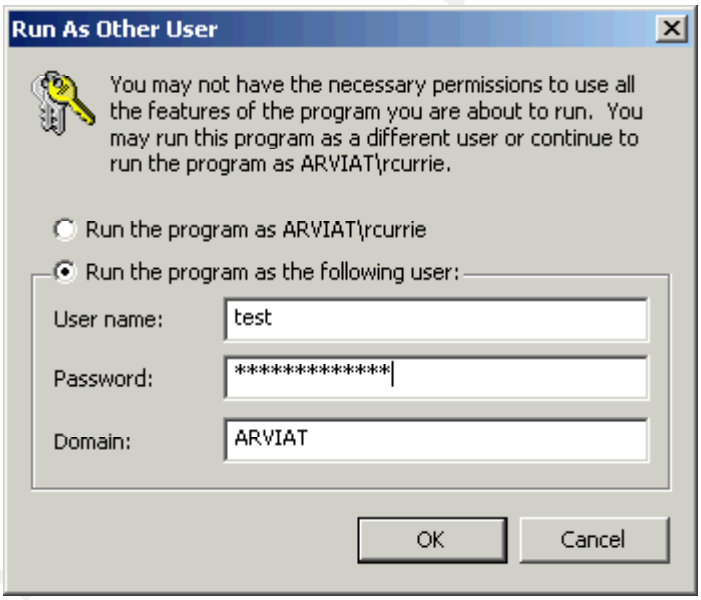

Figure 17 – RunAs screen

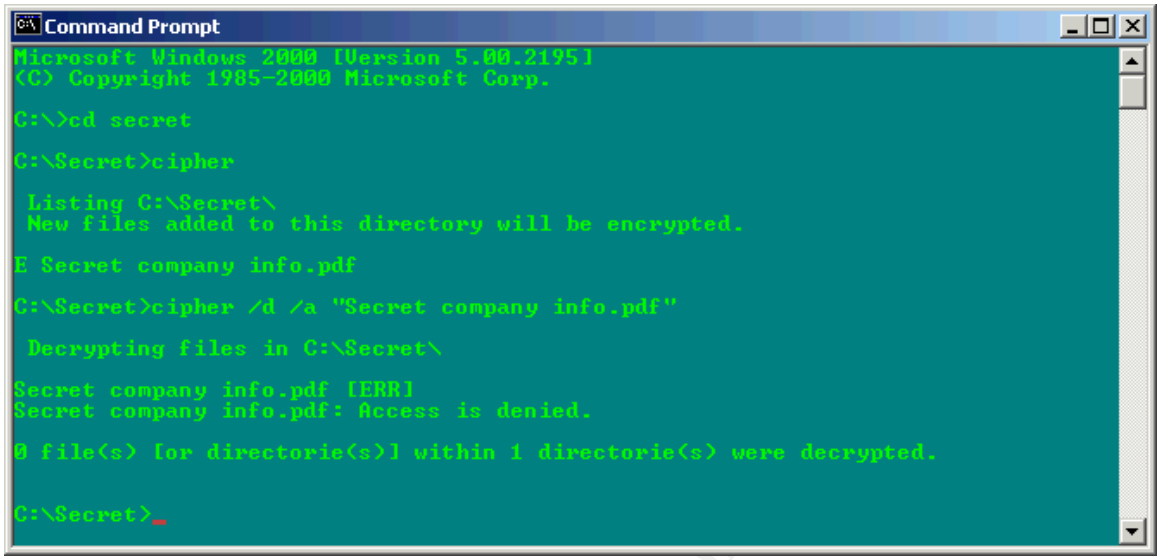

Figure 18 – CIPHER utility displaying unauthorised access message

Copying an Encrypted Folder or File

The copying process for an encrypted file or folder doesn't change if EFS is being used. This applies for both graphical user and command line interfaces. There are a few possible scenarios that can affect whether the data file remains encrypted or not. One of the design goals for EFS was to create transparency during usage. This same transparency can work against the user if he or she is not cognizant of the following situations.

1) Copying a file or folder on the same Windows 2000 ® system from one NTFSv5 partition to another NTFSv5 partition. From the graphical user interface use Explorer to copy the encrypted file or folder as you would any other file. From the command line, use the COPY command as you would any other file. In both cases the copied file remains encrypted.

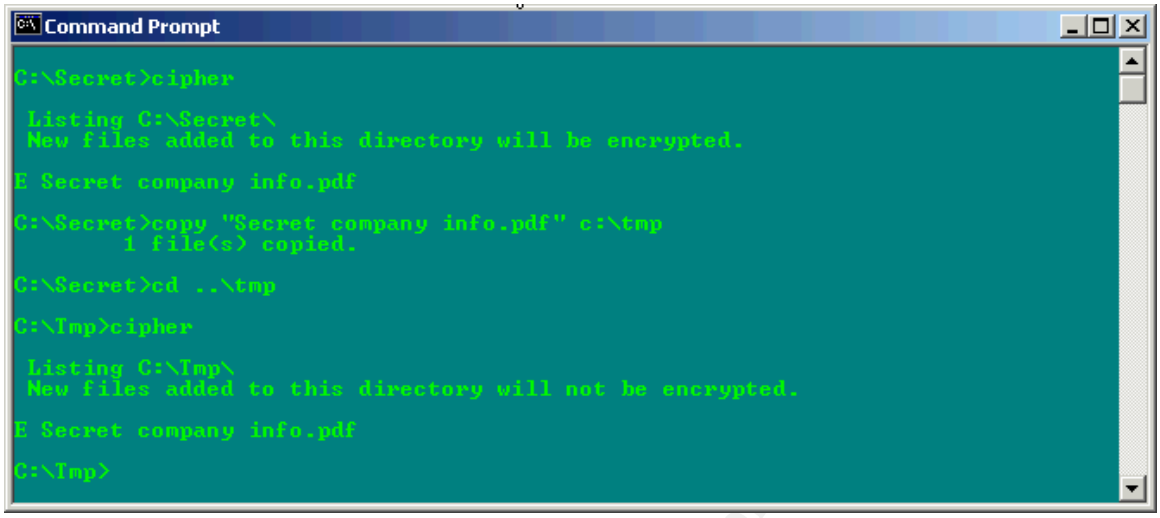

Figure 19 – CIPHER view of pre and post COPY operations

2) Copying a file or folder on the same Windows 2000 ® system from one NTFSv5 partition to a FAT partition. From the graphical user interface use Explorer to copy the encrypted file or folder as you would any other file. From the command line, use the COPY command as you would any other file. The FAT file system doesn't support EFS. During the copy operation the file is unencrypted and written to the FAT partition in clear text. The user must be aware of which partitions are NTFSv5 and which are not. This implies are certain level of computing skill or a uniform NTFSv5 environment.

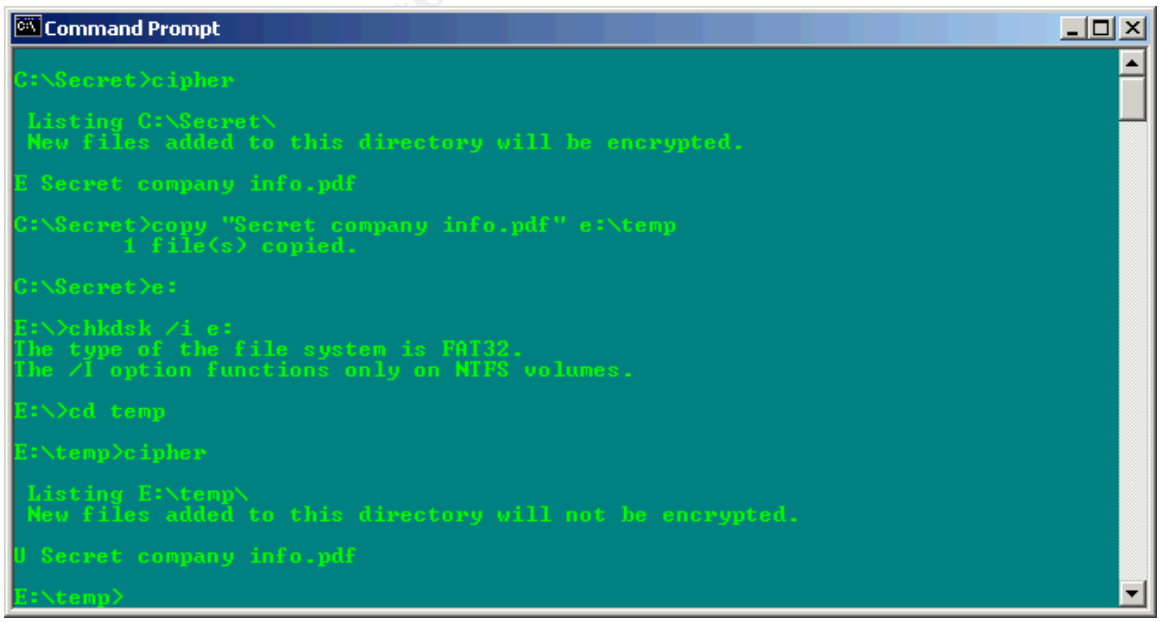

Figure 20 – CIPHER view of post COPY operation to non-NTFSv5 volume

3) Copying a file or folder on a Windows 2000 ® system to another Windows 2000 ® system where both use NTFSv5 partitions. From the graphical user interface use Explorer to copy the encrypted file or folder as you would any other file. From the command line, use the COPY command as you would any other file. If the remote Windows 2000 ® system allows you to encrypt files, the copy is encrypted; otherwise it is in clear text. The remote Windows 2000 ® system must be trusted for delegation in a given domain. The encryption certificates must be accessible by the remote system for the transparent encryption and decryption operations.

This scenario is especially interesting. The author will copy an EFS encrypted copy of the following file from the test system to a remote Windows 2000 ® Server file share G: The shared folder resides on a NTFSv5 partition. The file is a small README.TXT for a MD5 checksum generator program.

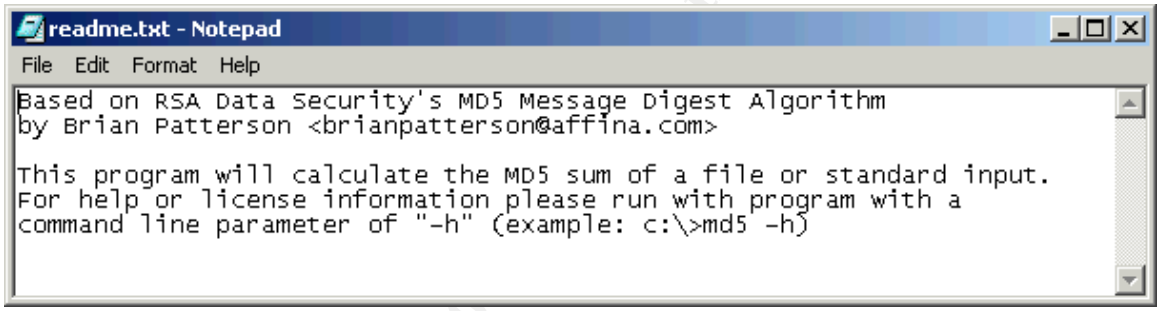

Figure 21 – Sample text document

The file is currently in our C:\SECRET folder and is encrypted. From the command line, README.TXT is copied to the remote share G:. The command reports a successful copy. Further verification with CIPHER indicates that the file is still encrypted on G:.