OSR File Encryption Solution Framework

Sample Solution Guide

V1.5

1 Introduction

An important part of OSR's File Encryption Solution Framework (FESF) is the provision of a fully working Sample Solution. This manual will provide a brief guide to that solution.

This manual assumes you have read and are familiar with the topics described in the FESF Solution Developer's Guide. If you have not read that document, please do so before reading any further.

Why did we create and provide a sample Solution? There are two primary reasons:

1. To demonstrate the use of FESF in a (more or less) real-world environment. The sample demonstrates one way to design and implement a complete (if simple) Solution using FESF. The sample also shows, end-to-end, how to interact with the various parts that comprise FESF.

2. To provide you with some "starter code" that you can customize. While it's important to understand that the Sample Solution is just that – a sample and not a finished product suitable for commercial distribution – we hope that parts of it can give you a head start on implementing your own Solution.

Of course, there's one other selfish reason that we created a fully functional Sample Solution: for our own internal testing. The development and use of this sample has provided us the opportunity to find and fix numerous problems in FESF, **before** those problems could affect you. Of course, it is possible that during your own development you will exercise FESF in ways that are sufficiently different than we have done to date. And this may uncover unexpected issues. If that happens, please report them to us so that we can work together to resolve those issues and ensure that FESF provides the best possible base for your product.

All of the code that makes-up the FESF Sample Solution is provided in source code format, and may be used by you as an FESF licensee as part of your own Solution implementation. While we are always happy to answer questions or accept bug reports on the provided solution code modules, we want to emphasize that the solution components are not an officially supported part of FESF. Bugs in the sample components will be fixed on a time-available basis. And all of the Sample Solution components are subject to change in future FESF releases.

2 Installing and Using the Sample Solution

To allow you to become familiar with FESF, and the Sample Solution we've provided, your FESF distribution kit contains an MSI file that will install FESF and the Sample Solution on a test system.

NOTE: The FESF Sample Solution is sample code. Do not install this code on a production system. Restrict the installation of the FESF Sample to test systems.

Important, please read before installing: To emphasize the point that the Sample Solution is not suitable for production use, the 64-bit versions of drivers that are provided are signed with test signing certificates, not production certificates. Before the driver will load, the system must have test signing enabled and the provided test signing certificate must be installed in the root Certificate Store of that system. Refer to the release notes for further installation directions and information.

Despite all these warnings, we do urge you to install the FESF Sample. This will help you get familiar with the kit and the sample.

3 Goals for the Sample Solution

The overall design goal for the Sample Solution was to create a simple but functional example that illustrates one way to create an FESF-based Solution. More specifically, we wrote the sample code with the following design goals in mind:

- **Limited Complexity**: While we wanted to create a representative Sample Solution, we worked hard to avoid as much of the complexity inherent in a real "enterprise level" product as possible.
- **Easy to Understand**: In designing and developing the Sample code modules, we tried to make it easy to understand the pieces that exist in the Sample Solution and their limitations, yet illustrate the typical modules that will need to be built for a Client's Solution. We hope you'll see the results of this goal in the code but also especially in the comments, particularly in the Sample Solution Client Policy DLL and the sample Policy and Key Management Service.
- **Useable as a Base for Implementation**: As previously described, we wanted the Sample Solution to contain code that Solution developers could use as a starting point for further customization.
- **Obvious Limitations**: We wanted the limits and limitations of the Sample Solution (both functionally during use, and internally in the code) to be very clear. We also tried to indicate which limitations are imposed by the Sample (because it's just a simplified example implementation), as opposed to which limitations are inherent in FESF.

To accomplish all these goals, we decided on the following architectural limitations:

- **Demonstrate FESF, not security**: We designed our Sample Solution to demonstrate the use of FESF's encryption infrastructure. It is not our objective to provide file or data security. That, after all, is your job in creating a real FESF-based Solution. On the other hand, we have tried to demonstrate best practices in the internal mechanisms that are implemented as part of the Sample Solution, including best practices in the realm of application security.
- **Single system solution**: Even though we expect that FESF Solutions will mostly function in an enterprise setting, the Sample Solution is limited to a single system. This means that all components of the Sample reside on a single system, and policy is to be stored on that same system (in the Registry). The Sample Solution does however support different policies for different users of that single system.
- **Trivial Key and Encryption Algorithm Management**: Key management can be very complex. The Sample deliberately takes a trivial approach to key and algorithm management. All files are encrypted using the AES 128-bit algorithm (in cipher block chaining – CBC – mode). Each time a new file is created, a new key is generated for use in encrypting that file. The key is a UUID. The header data that's returned from the Sample Solution's Policy DLL (and that's stored with the encrypted file) includes the key UUID converted to a text string and stored in cleartext. The Sample Solution doesn't use any PKI techniques, nor does it include any additional authentication. As a result of these limitations:
    - The Sample Solution provides no useful security.
    - Any user of the Sample Solution will be able to decrypt files that were encrypted by any other user running the same version of the Sample.
    - Any system running the FESF Sample Solution will be able to decrypt files written by any other system running the same version of the FESF Sample Solution.

- **Be Permissive in What We Allow Users to See or Do (within reason)**: We wanted the Sample Solution to be easy to use and clear. If this means the Sample Solution is more permissive in certain circumstances than a real enterprise security product might be, that's OK. For example, an actual enterprise security product would probably not allow an arbitrary user to view system-wide policy. It almost certainly wouldn't allow a user that's not running "as administrator" to encrypt or decrypt arbitrary files that they own (or allow users that are simply members of the "administrators" group to encrypt any file they can access). The Sample Solution allows all these things.

4 Implementation

The FESF Sample Solution is provided as a single Visual Studio Solution, named "UM_Sample.sln" As provided, the sample builds from source without either errors or warnings (at /W4) and passes Visual Studio Code Analysis at the Microsoft Native Recommended Rules level. Code Analysis is enabled for every build (so expect build times to be longer than you might otherwise expect).

Note that before building the Sample Solution, you must first build the FESF user mode components. These components reside in the Visual Studio Solution name "UM_FESF.sln"

5 Sample Solution Policy

In order to meet the goals we described previously, the Sample Solution bases its Policy decisions on only three factors:

- **Accessed File**: The path specification for the target file. The file path for local files is expressed by the user in standard drive-letter terms, and may include standard "DOS" wildcards (star and question mark) in any part of the path. The file path for network files is expressed by the user via UNC path and may also include standard “DOS” wildcards. Note that UNC paths, drive letters and directories may be wild-carded (not just file names).

- **Accessing Application**: The fully qualified path to an application accessing the target file. This is the same as above, but for the executing application accessing the target file. For example, "C:\Program Files\Microsoft Office\Office15\WINWORD.EXE" for Microsoft Word. DOS type wildcards are allowed, as above, and a value of "*" means "any".

- **User**: The Security ID (SID) of a user or a group under which the application is executing. Any user may be specified with the value of "*".

Policy rules can be defined using any combination of these three factors and the Policy to be implemented when the factors of the rule are met. For example, we could define a rule such as the following:

Accessed File: D:\MyEncryptedStuff*.*
Application: *
User: *
Access: Enc/Dec

This rule says that if any user, running any application, creates a new file or opens an existing file in the \MyEncryptedStuff\ directory on drive D: then that application and user should receive transparent encryption/decryption. That's not a very secure rule, of course, but it's a valid rule nonetheless.

While the Sample Solution bases its Policy decisions on only the three factors described, it is possible (and almost certainly required) that a real Solution will base Policy decisions on other factors provided by FESF (whether a new file is being created or an existing file is being accessed, the disposition of the file being accessed, or the access granted to the file being accessed, just for a few examples), or other factors a Solution can determine on its own. To keep things simple, our Sample Solution only uses these three factors.

The rules are organized into an ordered list. When the Sample Solution is called to determine if a given combination of file, application, and user should have raw or encrypted access to a file, the policy list is checked in order. The first policy rule that matches (either explicitly or through the use of one or more wildcards) defines the policy for this operation. If no rules are met, the default for our Sample Solution is to create new files in raw format and to allow Enc/Dec (transparent encryption and decryption) access to existing encrypted files.

Rules are defined and stored by the GUI sample Policy Manager (PolicyManager.exe). The Policy Manager's dialog box is shown in Figure 1.

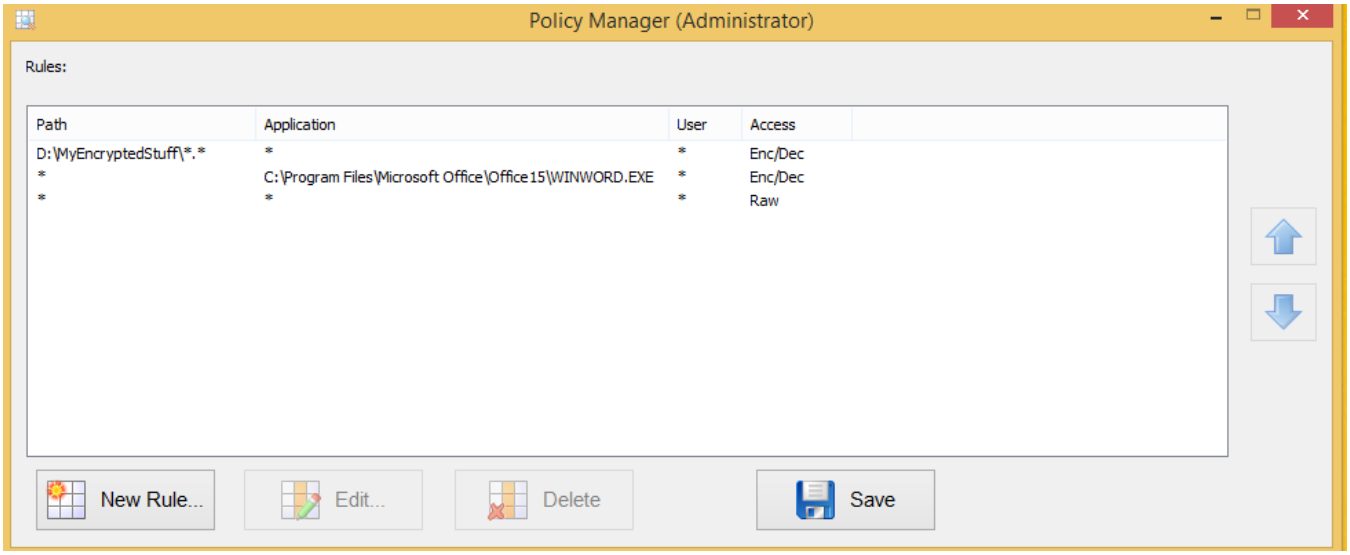

Figure 1 -- Sample Policy Manager Dialog Showing Three Rules

5.1 Defining Network Policy

As described in the **Solution Developer's Guide**, in FESF all network paths are handled in UNC format. Thus, when using Policy Manager to define policy for a network share, the path specification must be specified in UNC format. For example, the following policy would encrypt all files created in share MyShare on server MyServer:

Accessed File: \\MyServer\MyShare*.*
Application: *
User: *
Access: Enc/Dec

This rule says that if any user, running any application, creates a new file or opens an existing file via \\MyServer\MyShare, then that application and user should receive transparent encryption/decryption.

It is important to note that the Sample Solution Policy only uses string pattern matching to determine if a particular file path applies to the given policy. For example, if the server MyServer has an IP address of 10.1.1.199, the user may create a file using the path \\10.1.1.199\MyShare\Secrets.txt. While this path is functionally equivalent to \\MyServer\MyShare\Secrets.txt, the Sample Solution Policy treats these as distinct paths. If desired, an additional entry may be created to apply policy to the IP address based path (or the server may be specified using the wildcard syntax).

When accessing files via network mapped drive letters, the resulting paths are processed by the Sample Solution Policy in their UNC equivalent. For example, if drive letter X: is mapped to \\MyServer\MyShare, creation of the file X:\Secrets.txt will be processed by the Sample Solution Policy as a creation of \\MyServer\MyShare\Secrets.txt. Thus, network mapped drive letters must never be used to specify policy via the Sample Policy Manager. Always specify policy using the UNC path used to create the drive letter mapping.

Important Note: The limitations described here are a function of the simplified Sample Solution we are providing to help you get started with your own, more robust, solution. These are not limitations of the base FESF product. As you develop your own solution, you can determine how policy rules are defined and implemented.

5.2 Policy Examples

Keeping in mind that we have chosen to implement a very simple Sample Solution Policy, it might be helpful to walk through a couple of examples to explain how the sample uses the defined rules.

5.2.1 Manual Encryption of Microsoft Office Items

FESF is installed but no policy rules have been defined in Policy Manager. Let's say that we have just created a Microsoft Word document that contains employee salary information, and we want to encrypt that document. Since FESF is installed on the system, we select the document from Explorer, right-click and choose FESF Sample Encrypt. The document is now encrypted. This can be verified by the gold lock icon that now overlays the document icon (if the gold lock icon doesn't immediately appear, press F5 to cause Explorer to refresh its view).

A week later this document is updated but when saved, the document it is no longer encrypted! Why? It's because Microsoft Office applications save changes to the file under revision into a temporary file. When the user saves the document it is **copied** back to a file with the original file name, thus creating a new file. Since we don't have any Policy defined to do otherwise, the file is created in clear-text and is no longer encrypted.

If this isn't the desired behavior, there are two way to ensure that revised encrypted documents stay encrypted:

1) Create a directory for your encrypted files (e.g., D:\MyEncryptedStuff) and define a policy that ensures all files created in that directory have Enc/Dec access. This is illustrated by the first rule shown in Figure 1.
2) Define a policy that encrypts all files created by Word. This is illustrated by the second rule in Figure 1.

Note that if we had been using Notepad instead of Word, the encrypted file would remain encrypted since notepad.exe doesn't use temporary files followed by a copy. If we edit a manually encrypted text file with Notepad.exe, it will remain encrypted.

5.2.2 Copying Encrypted Files

We have a directory for encrypted files called D:\MyEncryptedStuff and have a rule defined ensuring that all files created in this directory get encrypted (see first rule in Figure 1). To take encrypted documents home, we copy the

directory onto a USB pen drive expecting that they will remain encrypted but they get decrypted on copy! Why? The rule defined which encrypts files written to the D:\MyEncryptedStuff directory also says to decrypt files read from that directory. Further, there is no rule that says documents written to my USB drive should be encrypted. Thus, the decrypted files.

One possible solution would be to define another rule which gives a copy application (e.g., RoboCopy.exe) raw access to files. It's important to remember that **rules are processed in order** so if the rule giving the copy application raw access comes after the rule which give all files in D:\MyEncryptedStuff Enc/Dec access then the copied files will still be decrypted. However, if the rule giving the copy application raw access is first in the list, then the files will not be decrypted before being copied and encrypted files will be copied to the USB drive.

5.3 Remember… The Sample is Just an Example

It's worth mentioning again these rules, syntax, and capabilities described here are defined by the OSR-provided Sample Solution. While the full FESF kit contains the source code for every part of this Sample for Solution developers to use if as they wish, the Sample Solution only demonstrates a subset of FESF's capabilities.

6 Sample Solution Components

Figure 2 shows the components and interfaces that comprise both FESF and the Sample Solution. Supported FESF components are shown in orange. Components that are part of the FESF Sample Solution are shown in green.

As in the Solution Developer's Guide, black solid and dashed lines indicate FESF architecturally defined interfaces that are documented and supported by OSR. The solid black line between the FESF Policy Service and the Sample Policy DLL represents the synchronous call and return interface between those two components. The black dotted lines represent COM interfaces for FESF-provided support and utility functions that are used by the Sample Solution.

The red lines are undocumented, unsupported, interfaces that are private and reserved to OSR and subject to change in future FESF releases.

Solid green lines represent conventional accesses to the Windows registry, performed by the Sample Solution. Dashed green lines represent COM interfaces defined and implemented by the Sample Solution.

While we've tried to be complete in illustrating all the interfaces used by the Sample Solution, it's possible we've missed some. But the primary interfaces are shown.

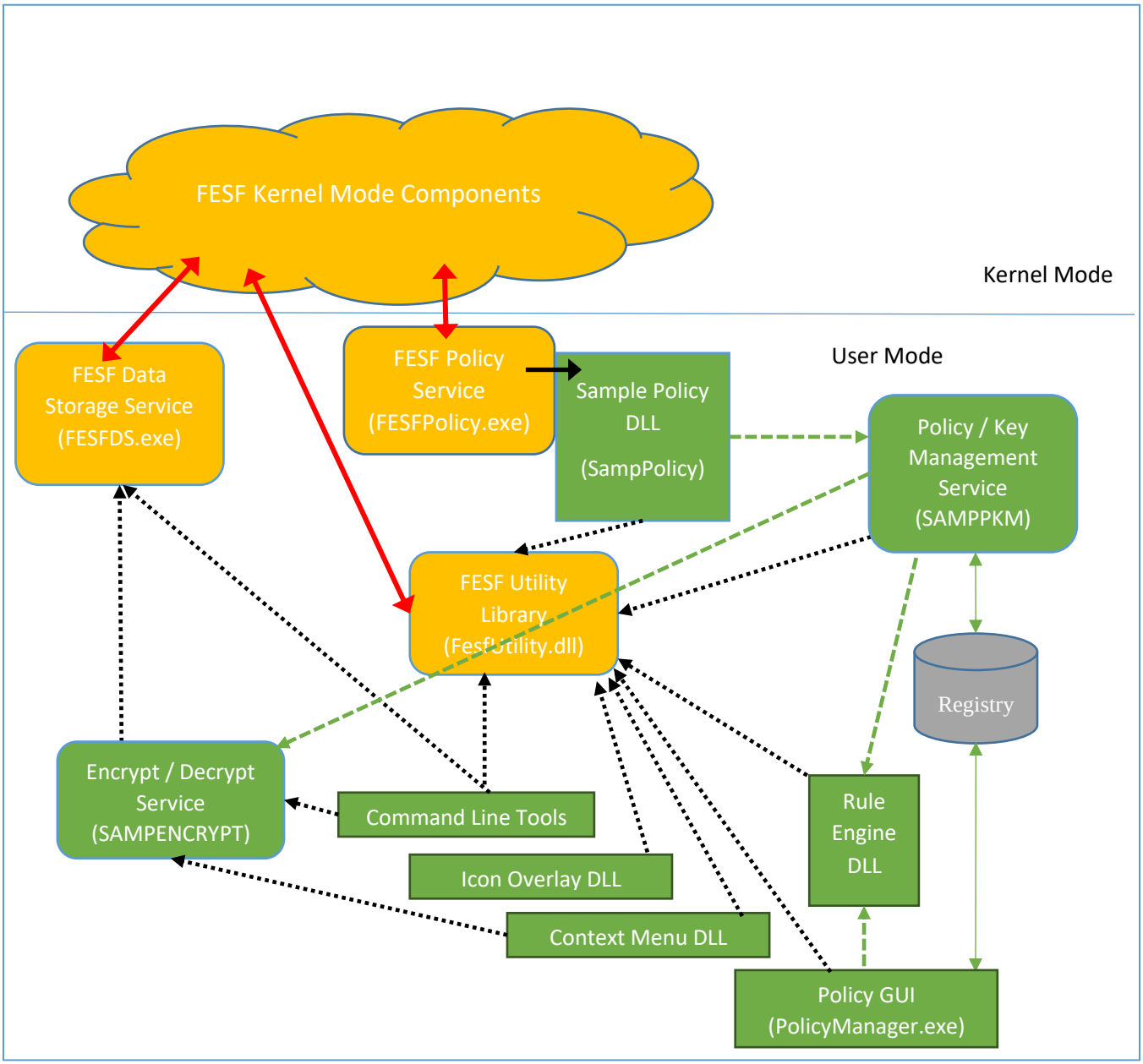

Figure 2 -- Sample Solution Components and their Primary Interfaces

Remember, when a Client builds their own Solution the components for that solution will replace the green boxes in Figure 2. The Client Solution will, by necessity, include **some** of the same modules shown in green boxes Figure 2 (for example, the Solution will need to provide a Policy DLL that takes the place of the Sample Policy DLL). But the actual implementation of the Client Solution may (and likely will) look very different from the Sample Solution. For example, a Client Solution's Policy DLL may make calls over the network to a remove Key Management Server.

The remainder of this section describes each of the Sample Solution components.

6.1 Sample Policy DLL (SampPolicy.dll)

The Sample Policy DLL is the interface between requests from FESF and the portions of the Sample Solution that define policy. The sample Policy DLL is designed to be stateless, and to serve as a "forwarder" of FESF's Policy requests to the Sample Policy and Key Management Service.

In this role, SampPolicy does very little other than receive requests, convert incoming arguments from FESF (for example, Thread ID) into more usable forms (such as the fully qualified path of the application attempting to access a file), and pass through arguments to the Sample Policy and Key Management Service.

Within the UM_Sample Visual Studio Solution, the Sample Policy DLL is part of the SampPolicy project.

6.2 Policy / Key Management Service (SampPkm.exe)

The sample Policy and Key Management Service (SampPKM) receives Policy requests from the sample Policy DLL, and uses the Sample Solution Rule Engine to make Policy decisions.

SampPKM publishes a COM interface that is consumed by the Sample Solution Policy DLL to pass it Policy requests. SampPKM calls the COM interface provided by the Rule Engine DLL to evaluate rules. The rules database is read from the Registry (where it was stored by the Policy Manager GUI utility).

In addition to rules evaluation, SampPKM provides special assistance to the Sample Solution's on demand encryption and decryption process. This is described more fully as part of the description of SampEnc's processing, later in this document.

Within the UM_Sample Visual Studio Solution, the Policy / Key Management Service is in the \Services\ folder with the project name SampPKM.

6.3 Rules Engine (RuleEngine.dll)

The Sample Solution Rules Engine is a DLL that provides service to both the Policy GUI (PolicyManager.exe) and the Policy/ Key Management Service (SampPKM.exe). The Rules Engine stores the Policy rules that were described previously, after they've been defined by the user using the PolicyManager GUI. The rules engine is also responsible for evaluating the rule set, in order, and determining a Policy result, when requested by the SampPKM.

Within the UM_Sample Visual Studio Solution, the Rules Engine comprises the RuleEngine project.

6.4 Icon Overlay Handler and Context Menu Handler DLLs

These Sample Solution DLLs are add-ins to the Windows shell (Explorer.exe).

The sample Icon Overlay Handler determines if files being displayed by Explorer are in FESF encrypted format (by calling an FESF utility function in the FesfUtility DLL). If a file is in FESF encrypted format, the Icon Overlay Handler displays a golden lock icon overlay on the file.

Within the UM_Sample Visual Studio Solution, the Icon Overlay Handler is in the \Shell\ folder with the project name IconOverlayHandler.

The sample Context Menu Handler provides a right-click context menu option that allows users to encrypt or decrypt files on demand. The Context Menu Handler does its job by calling support routines provided by the FesfUtility DLL. When a file is to be encrypted or decrypted, the Context Menu Handler passes the fully qualified path of the file to the sample Encrypt / Decrypt Service (SampEncrypt.exe).

Within the UM_Sample Visual Studio Solution, the Context Menu Handler is in the \Shell\ folder with the project name ContextMenuHandler.

6.5 Encrypt/Decrypt Service (SampEncrypt.exe)

The sample Encrypt / Decrypt Service (SampEncrypt) publishes an interface that's used by the sample shell Context Menu Handler, as well as by the sample Encrypt and Decrypt command line utilities. SampEncrypt's primary job is to cause existing files to be encrypted or decrypted on command. Because FESF only automatically encrypts newly created file, and only automatically decrypts or encrypts existing encrypted files, we think developers will be particularly interested in how SampEncrypt does its job. Note that SampEncrypt will only work in an Online FESF environment, but FESF does support (and the full FESF distribution kit includes) offline utilities that enable encryption, decryption, and Solution Metadata manipulation on Windows and Linux systems without FESF installed.

SampEncrypt performs on demand encryption and decryption of existing files using what we've termed a "brute force" approach. Let's say a user requests that the file "c:\Users\Dan\Documents\MyResume.Docx" be encrypted. SampEncrypt will attempt to encrypt the specified file using the following steps:

1. SampEncrypt checks to ensure that the requester is either the owner of the file or is running with administrator privileges. If the requester is neither, an error is returned.
2. SampEncrypt generates an arbitrary temporary file name. SampEncrypt bases this temporary name on a GUID to avoid the chance of collisions.
3. SampEncrypt adds the temporary file name to an internal list (using the appropriate locking, of course). Along with this entry, SampEncrypt stores whether the request received was to encrypt or decrypt the original file.
4. SampEncrypt copies the original file to the temporary file.

As part of copying the original file to the temporary file, when the new temporary file is created FESF will call the Sample Solution's Policy DLL at its *PolGetPolicyNewFile* callback. The sample Policy DLL will marshal the appropriate arguments and forward them to the sample Policy / Key Management Service (SampPKM) for a Policy decision. When it is called, SampPKM notices that the new file is being created by the SampEncrypt Service (e.g. by recognizing the SID under which SampEncrypt is running). In this case, SampPKM calls SampEncrypt to determine if the file should be created in encrypted format (which will be the case if the original file is being encrypted) or raw format (which will be the case when the original file is being decrypted).

5. SampEncrypt is called by SampPKM to determine if the new file should be created encrypted or raw.
6. SampEncrypt looks up the target file name in its list, and returns the stored answer to SampPKM.

SampPKM then returns this Policy decision (based on the return value SampPKM received from SampEncrypt) to SampPolicy. SampPolicy returns this decision to the kernel mode FESF components.

7. The copy from the original file to the temp file completes. If it was successful, SampEncrypt copies any file attributes from the original file to the temp.
8. SampEncrypt renames the temp file to the name of the original file, replacing the original file.

The requested encryption or decryption operation is now complete.

These steps are only a brief summary, but we hope they're detailed enough to give you the overall idea of how things work. It's almost harder to explain in writing than it is to read the code. The "trick", if you can call it that, is that SampEncrypt itself never does any encryption or decryption at all. Rather, it relies on FESF to provide encryption and decryption services by clever use of its facilities.

Within the UM_Sample Visual Studio Solution, the SampEncrypt service is in the \Services\ folder with the project name SampEncrypt.

6.6 Command-Line Tools (SampEnc, SampDec, SampDir, and SampUpdateHeader)

The Sample Solution also provides a small set of sample command line tools. These tools provide support for on-demand encryption, decryption, solution header management, and a simple way of enumerating which files are encrypted by FESF.

Within the UM_Sample Visual Studio Solution, the sample command line tools are in the \Tools\ folder with the project names SampDir, SampEncDec, and SampUpdateHeader.

6.6.1 SampDir

The SampDir utility shows the encryption status of the files in the specified path.

Usage: SampDir [/? | /r] [/v] [file or path (wild cards supported)]

/? – help

/r – recurse into the directory

/v – prints the status for decrypted files (as well as encrypted files)

If /v is omitted, then only the status of encrypted files is shown. Note that if no path is provided the files in the current directory will be processed.

6.6.2 SampEncDec

SampEnc.exe and SampDec.exe will encrypt/decrypt the files in the specified path.

Usage: SampEnc/SampDec [/? | /r] <file or path (wild cards supported)>

/? – help

/r – recurse into the directory

The SampEnc.exe and SampDec.exe utilities are really the same and will encrypt files in path if the executable is named SampEnc.exe and will decrypt the files in the path if the executable is named SampDec.exe. Note that an error will be returned if no file/path is passed in to these utilities.

6.6.3 SampUpdateHeader

The SampUpdateHeader utility verifies and optionally updates the solution header of an encrypted file (or files).

Usage: SampUpdateHeader [/? | /r] [/x /s /m /i] <file or path (wild cards supported)>

/? – help

/r – recurse into the directory

/x – examine the header in the file(s) to determine if it is valid

/s – update the sample header's sequence number after verifying the header

/m – verify the file header, update the sequence number, write out a new randomly sized header, and finally verify the new header was written

/i – perform the same steps as the /m option, but always make the header larger

SampUpdateHeader demonstrates the *ReadHeader*, *UpdateHeader*, and *UpdateHeaderWithExtension* functions provided by the FESF Data Storage Service. See the Solution Developer's Guide for more information on these functions.

6.7 Stand-Alone Utilities

The full FESF kit includes a set of Stand-Alone tools for use on Windows and Linux systems where FESF has not been installed. These tools allow files to be encrypted and decrypted, and for Solution Metadata to be added, deleted, or modified. These tools are not included in the FESF Demonstration Package.