FreeOTFE Explorer

v3.00

By Sarah Dean
sdean12@sdean12.org

Last updated: 22nd July 2009

http://www.FreeOTFE.org/

A free on-the-fly transparent disk encryption program for MS Windows,
featuring an MS Windows Explorer-style interface, and driverless operation.

Contents

1 Introduction

This software allows FreeOTFE (and Linux) encrypted volumes to be mounted on MS Windows PCs, and their contents accessed (both reading and writing) via a Windows Explorer-style user interface.

Unlike *all other* disk encryption systems, FreeOTFE Explorer does *not* need any device drivers to be installed on the computer it is run on, making it *perfect* for use with USB drives (and other removable media), where volumes need to be accessed on PCs where FreeOTFE hasn't been installed, and administrator rights are not available to the user in order to start FreeOTFE in "portable mode". For example, in Internet Cafés (AKA Cybercafés), where PCs are available for use, but only as a "standard" user.

FreeOTFE Explorer is *fully* compatible with both FreeOTFE and FreeOTFE4PDA volumes, and runs on for MS Windows 2000/XP/Vista/Windows 7 PCs (both 32 and 64 bit)

1.1 Features

- **Driverless operation - making it *ideal* for carrying your data securely on USB drives or other removable media**. (*No administrator rights are required* to carry out initial installation or start portably - unlike *all* other forms of disk encryption.)
- **Fully compatibile with FreeOTFE on-the-fly disk encryption**
- **Source code freely available**
- No installation needed
- Easy to use; full wizard included for creating new volumes
- Data encrypted on your PC can be read/written on your PDA, and vice versa
- Powerful: Supports numerous hash/encryption algorithms, and provides a greater level of flexibility than a number of other (including many commercial!) OTFE systems
- Available in English, German, Italian, French and Spanish, with support for other language translations
- Hash algorithms include: MD5, SHA-512, RIPEMD-320, Tiger and *many* more
- Cyphers include AES (256 bit), Twofish (256 bit), Blowfish (448 bit), Serpent (256 bit) and *many* more
- Cypher modes supported include XTS, LRW and CBC (including XTS-AES-128 and XTS-AES-256)
- Security tokens/smartcards supported for extra (optional) security
- Linux compatibility (Cryptoloop "losetup", dm-crypt and LUKS supported)
- "Hidden" volumes may be concealed within other encrypted volumes, providing "plausible deniability"
- Encrypted volumes have no "signature" to allow them to be identified as such
- Encrypted volumes can be either file or partition based.
- Modular design allowing 3rd party drivers to be created, incorporating new hash/cypher algorithms
- Supports password salting (up to 512 bits), reducing the risks presented by dictionary attacks.
- Allows users to backup and restore the critical areas of volume files.
- Keyfile support included; store volumes and their associated metadata separately.
- Volume file timestamps and attributes are reset after dismounting, increasing "plausible deniability"

- Supports volumes files up to 2^63 bytes (8388608 TB)
- Comprehensive documentation
- ***Plus more...!***

Screenshots of FreeOTFE Explorer are available

Cyphers included:

Cypher	Key length (in bits)	Block Length (in bits)	Modes	Source Library	Comments
AES	128	128	CBC/XTS	Dr. Brian R. Gladman	XTS version aka XTS-AES-128
AES	192	128	CBC/XTS	Dr. Brian R. Gladman	
AES	256	128	CBC/XTS	Dr. Brian R. Gladman	XTS version aka XTS-AES-256
AES	128	128	CBC/LRW/XTS	LibTomCrypt	XTS version aka XTS-AES-128
AES	192	128	CBC/LRW/XTS	LibTomCrypt	
AES	256	128	CBC/LRW/XTS	LibTomCrypt	XTS version aka XTS-AES-256
Blowfish	128	64	CBC	LibTomCrypt	
Blowfish	160	64	CBC	LibTomCrypt	
Blowfish	192	64	CBC	LibTomCrypt	
Blowfish	256	64	CBC	LibTomCrypt	
Blowfish	448	64	CBC	LibTomCrypt	
CAST5	128	64	CBC	LibTomCrypt	aka CAST-128
CAST6	128	128	CBC	Dr. Brian R. Gladman	aka CAST-256
CAST6	160	128	CBC	Dr. Brian R. Gladman	aka CAST-256
CAST6	192	128	CBC	Dr. Brian R. Gladman	aka CAST-256
CAST6	224	128	CBC	Dr. Brian R. Gladman	aka CAST-256
CAST6	256	128	CBC	Dr. Brian R. Gladman	aka CAST-256
DES	64	64	CBC	LibTomCrypt	
3DES	192	64	CBC	LibTomCrypt	Standard encrypt, decrypt, encrypt

MARS	128	128	CBC/XTS	Dr. Brian R. Gladman	
MARS	192	128	CBC/XTS	Dr. Brian R. Gladman	
MARS	256	128	CBC/XTS	Dr. Brian R. Gladman	
Null	0	(variable)	n/a	n/a	
RC-6	128	128	CBC/XTS	Dr. Brian R. Gladman	
RC-6	192	128	CBC/XTS	Dr. Brian R. Gladman	
RC-6	256	128	CBC/XTS	Dr. Brian R. Gladman	
RC-6	128	128	CBC/LRW/XTS	LibTomCrypt	
RC-6	192	128	CBC/LRW/XTS	LibTomCrypt	
RC-6	256	128	CBC/LRW/XTS	LibTomCrypt	
RC-6	1024	128	CBC/LRW/XTS	LibTomCrypt	
Serpent	128	128	CBC/XTS	Dr. Brian R. Gladman	
Serpent	192	128	CBC/XTS	Dr. Brian R. Gladman	
Serpent	256	128	CBC/XTS	Dr. Brian R. Gladman	
Twofish	128	128	CBC/XTS	Dr. Brian R. Gladman	
Twofish	192	128	CBC/XTS	Dr. Brian R. Gladman	
Twofish	256	128	CBC/XTS	Dr. Brian R. Gladman	
Twofish	128	128	CBC	Hi/fn and Counterpane Systems	x86 systems *only*
Twofish	192	128	CBC	Hi/fn and Counterpane Systems	x86 systems *only*
Twofish	256	128	CBC	Hi/fn and Counterpane Systems	x86 systems *only*
Twofish	128	128	CBC/LRW/XTS	LibTomCrypt	

Twofish	192	128	CBC/LRW/XTS	LibTomCrypt	
Twofish	256	128	CBC/LRW/XTS	LibTomCrypt	
XOR	(variable)	(variable)	n/a	n/a	

Hash algorithms included:

Hash	Hash Length (in bits)	Block Length (in bits)	Source Library
MD2	128	128	LibTomCrypt
MD4	128	512	LibTomCrypt
MD5	128	512	LibTomCrypt
Null	(variable)	(variable)	n/a
RIPEMD-128	128	512	LibTomCrypt
RIPEMD-160	160	512	LibTomCrypt
RIPEMD-160 (Linux; Twice, with A)	320	512	LibTomCrypt
RIPEMD-256	256	512	LibTomCrypt
RIPEMD-320	320	512	LibTomCrypt
SHA-1	160	512	LibTomCrypt
SHA-224	224	512	LibTomCrypt
SHA-256	256	512	LibTomCrypt
SHA-384	384	1024	LibTomCrypt
SHA-512	512	1024	LibTomCrypt
Tiger	192	512	LibTomCrypt
Whirlpool	512	512	LibTomCrypt

2 Download

The latest release of FreeOTFE Explorer, together with its source code, may be downloaded from the official site at: http://www.FreeOTFE.org/download.html

PGP, MD5 and SHA-1 signatures for this software may also be found on the above WWW site.

If you would like to build your own copy of FreeOTFE Explorer, you will also need the SDeanComponents package (required for building the GUI), which may be downloaded from: http://www.SDean12.org/SDeanComponents.htm

3 Installation and Upgrading

3.1 Automatic installation

1. Download the "installer" version of FreeOTFE Explorer, and run the executable.
2. After prompting you for some simple details, FreeOTFE Explorer will be automatically installed on your PC.

3.2 Manual installation

1. Uncompress the release ZIP to where you would like the software installed.
2. Create shortcuts to "FreeOTFEExplorer.exe" if required.

These steps will install and setup a FreeOTFE Explorer in a minimalist installation. With these steps complete, you may now use FreeOTFE Explorer to create and mount FreeOTFE and Linux encrypted volumes.

3.2.1 Upgrading from a Previous Version

Because of potential changes within the driver API, you must ensure that you completely uninstall your existing FreeOTFE Explorer installation before installing and using the latest version. Please see the section on uninstalling for details on how to do this.

4 Getting Started Guide

Pretty much everything in FreeOTFE Explorer works as it seems, and should be fairly self explanatory. If there's anything you're not too sure of, most times an educated guess will give you the right answer.

4.1 Installation

Before FreeOTFE Explorer can be used, it must first either be installed.

Please see the section on Installation and Upgrading for instructions on how to do this.

4.2 Using FreeOTFE Explorer

Once FreeOTFE Explorer is installed, securing your data is simple:

1. Create a volume to store your encrypted data on (or several, if you choose!)
2. Mount the volume created.
3. Files and directories may then be stored in, or extracted from your mounted volume.

   Anything and *everything* stored on the mounted volume (documents, pictures, videos, software - whatever you like) will be automatically encrypted and stored within the volume you created, at the point that it's stored.

   The encryption process is totally transparent to the user, and is carried out on-the-fly as data is written to the volume. Similarly, decryption is carried out transparently when data is read from it.

   You can drag files and folders onto the mounted volume FreeOTFE Explorer to store them.
4. To secure your data, simply dismount the drive, or close FreeOTFE Explorer. At that point, the encryption/decryption key is overwritten in memory - making your data totally inaccessable until the password (and other parameters, if needed) are supplied, and the volume is mounted again.

The following sections give more detailed instructions on how do carry out each of these steps.

4.3 Creating a new volume

In order to use FreeOTFE Explorer, you must first create a "disk image" (called a "volume") to represent your virtual drive.

This is a fairly straightforward process, and consists of using FreeOTFE Explorer to create a large file (or setup a partition) on your computer's hard drive.

This volume will hold an encrypted "disk image" of your virtual drive, and is where FreeOTFE Explorer will store all data written to your virtual drive.

This file (or partition) can subsequently be "mounted" within FreeOTFE Explorer - anything stored to which will be automatically encrypted before being written to volume file.

To create a new volume, select "File | New..." menuitem to display the "new volume wizard", which will guide you through the process in a series of simple steps.

New volume wizard

When prompted to select between creating a file or partition based volume, new users should select "File". This is the safer of the two options - partition based volumes are intended for more advanced users.

Some users who are unfamiliar with disk encryption systems may not understand all of the options they are presented with. If you feel that you are in this position, you should probably simply accept the default values you are presented with, which will give you a volume that will be secure enough for your needs.

FreeOTFE Explorer is a highly flexible system that caters for both novice and advanced users alike; many of the options that the volume creation wizard provides you with are intended for more advanced users who understand the implications of the options provided (e.g. storing a volume's CDB separately to the volume file it relates to), and how they operate.

You may want to create and use multiple volumes; one to store work related files, one for personal files, etc

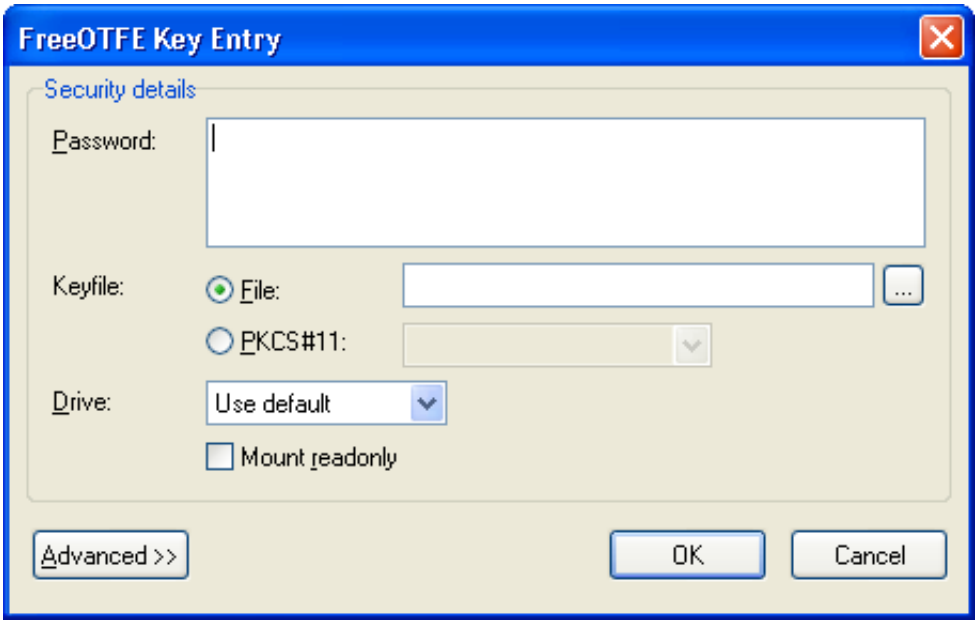
SECURITY Tip!	Do not simply copy an existing volume file to create a new one - even if you change the password on the "new" volume. If you do this, both volumes will have the same encrypted master key, which reduces the amount of security offered.

4.4 Mounting volumes

Once you have created a volume, it must be "mounted" in order for it to accessed via FreeOTFE Explorer.

Select "File | Mount file..." menuitem to mount a file based volume, or "File | Mount partition..." to mount a partition based volume.

You will then be prompted to select which volume you wish to mount; do so, and click "OK" to display the password entry dialog.



Password entry dialog

Enter your volume's password, and click "OK". If the correct password has been entered, the volume will be mounted and shown in the main FreeOTFE Explorer window.

Tip!	To reduce the time taken FreeOTFE Explorer spends mounting volumes, see the FAQ How can I speed FreeOTFE up when mounting my volumes?

To increase security, it is recommended that after a volume is formatted, it is overwritten with random data. However, this process can take some time and may be skipped if required. See section on plausible deniability for further details

Once mounted, files and folders can be stored and extracted from an encrypted volume, which will be transparently encrypted and decrypted as and when needed.

You can run more than one instance of FreeOTFE Explorer running at the same time to mount multiple volumes at the same time

---

4.5 Storing files in a volume

Once mounted, files and folders may be stored within an encrypted volume, by either:

- Selecting "Edit | Store" followed by either "File" or "Folder" from the main menu
- Clicking the "Store" icon on the toolbar, and selecting either "File" or "Folder" from the menu displayed
- Rightclicking on the main window to display the context menu, and selecting "Store" followed by either "File" or "Folder"
- Dragging files or folders from MS Windows Explorer and dropping them onto the main FreeOTFE Explorer window

After storing files/folders in a encrypted volume, use a file overwriter to overwrite the original copy, making them unrecoverable and leaving the only remaining copy securely encrypted within the volume.

4.6 Extracting files from a volume

Once mounted, files and folders stored within an encrypted volume may be extracted to the local hard disk by selecting the items to be extracted, and either:

- Selecting "Edit | Extract" from the main menu
- Clicking the "Extract" icon on the toolbar
- Rightclicking on the main window to display the context menu, and selecting "Extract"

Extracting files/folders from an encrypted volume will not remove them from the volume, but simply makes a *copy* of them to the computers local disk.

4.7 Dismounting volumes

Once you have finished using your secured volume, it should be "dismounted". This will remove access to it, and wipe any sensitive information FreeOTFE Explorer has stored in the computers memory.

Click the "File | Dismount" menuitem; or the toolbar icon.

4.8 Changing a volume's password

To change a volume's password (or a keyfile's password), select the "Tools | Change volume/keyfile password/details..." to display the "change password wizard", which will guide you through the process in a series of simple steps.

Note that volumes must be *dismounted* first before they can be modified in this way.

You may also change certain volume/keyfile details via this wizard; for example, the default drive letter which the volume will normally be mounted as. Advanced users may also change more technical details, such as the length of salt used in encrypting the volume's CDB/keyfile

In common with most disk encryption systems, FreeOTFE Explorer uses an "encrypted master key" system to secure volumes.

Every FreeOTFE volume has its own "master encryption key" which is generated when the volume is created. This master key is used to carry out the actual encryption/decryption process used to secure data stored within the volume.

A volume's master encryption key is, in turn, encrypted with the (PBKDF2 processed) user's password. As a consequence, FreeOTFE Explorer doesn't need to decrypt and re-encrypt the entire volume to change the user's password - only the encrypted master encryption key. This makes changing a volume/keyfile's password an extremely quick, and risk free, operation when compared to a complete volume re-encryption.

5 Advanced Topics

5.1 Keyfiles

A "keyfile" is a small file (about 512 bytes) which can optionally be created for a volume, and contains a copy of the information required to mount a FreeOTFE volume. Keyfiles are encrypted based a user-supplied keyfile password, which must be supplied in order to use the keyfile.

More than one keyfile can be created for the same volume.

Keyfiles are useful as they allow critical information which is required in order to mount a particular volume to be stored separately to the volume which they relate to; on a floppy disk, or USB drive, for example - which would be too small to store the entire volume on. In this way, your volume may be stored on your computer, but the information required to access it can be stored in a physically more secure location (e.g. in a locked safe)

In a business environment, keyfiles may be used as a form of password recovery, or to reset forgotten passwords. When confidential information is held within a FreeOTFE volume, a keyfile can be created for that volume and stored in a safe location. Should the employee which normally uses the volume be unavailable, or cannot remember the volume's password, the volume may still be mounted using a keyfile that has was previously created for it (together with that keyfile's password) - even if the *volume's password* has been subsequently changed.

Keyfiles may also be used to provide multiple users with access to mount and use the same volume; each using a password of their own choosing.

Note: Keyfiles are *specific* to the volume they are created for! Although a keyfile for one volume may be able to successfully *mount* another volume, the virtual drive shown will appear to be unformatted - the files within the volume will remain securely encrypted and unreadable.

5.1.1 Creating a new keyfile

To create a new volume, select "Tools | Create keyfile..." to display the "keyfile wizard", which will guide you through the process in a series of simple steps.

5.1.2 Mounting a volume using a keyfile

The process of mounting a volume using a keyfile is identical to the normal mount procedure, with the exceptions that:

1. The password used should be the *keyfile's* password, and *not the volume's password*.
2. The full path and filename of the keyfile should be entered as the "keyfile file"

5.2 Creating Hidden Volumes

FreeOTFE Explorer offers users the ability to create "hidden volumes" stored inside other "host" volumes.

To create a hidden volume:

1. If the volume you wish to create a hidden volume in is mounted, dismount it.
2. Start the volume creation wizard as normal (select "File | New..." from the main menu).
3. When prompted to select between creating a file or partition based volume, select "File" or "Partition", depending on whether the *host volume* you wish to use is file or partition based.
4. When prompted for the filename/partition to create your hidden volume on, select the *host* file/partition you wish to create the hidden volume inside.
5. The next step in the wizard will prompt you to enter an offset. The offset is the number of bytes from the start of the host volume where you wish the hidden volume to begin, and must be a multiple of 512. Make sure that the offset you specify is large enough such that it does not overwrite any of the system areas of that host volume (e.g. the FAT), or files already written to it.
6. Continue with the volume creation wizard as normal.

To mount your hidden volume, proceed as if mounting the *host* volume, but when prompted to enter your password, click the "Advanced" button and enter the offset. (See the section on advanced password entry options).

Make sure you remember the value you enter for the offset value! For security reasons, FreeOTFE Explorer doesn't store this information anywhere, and so you will have to enter the same offset into the password entry dialog every time you wish to mount your hidden volume.

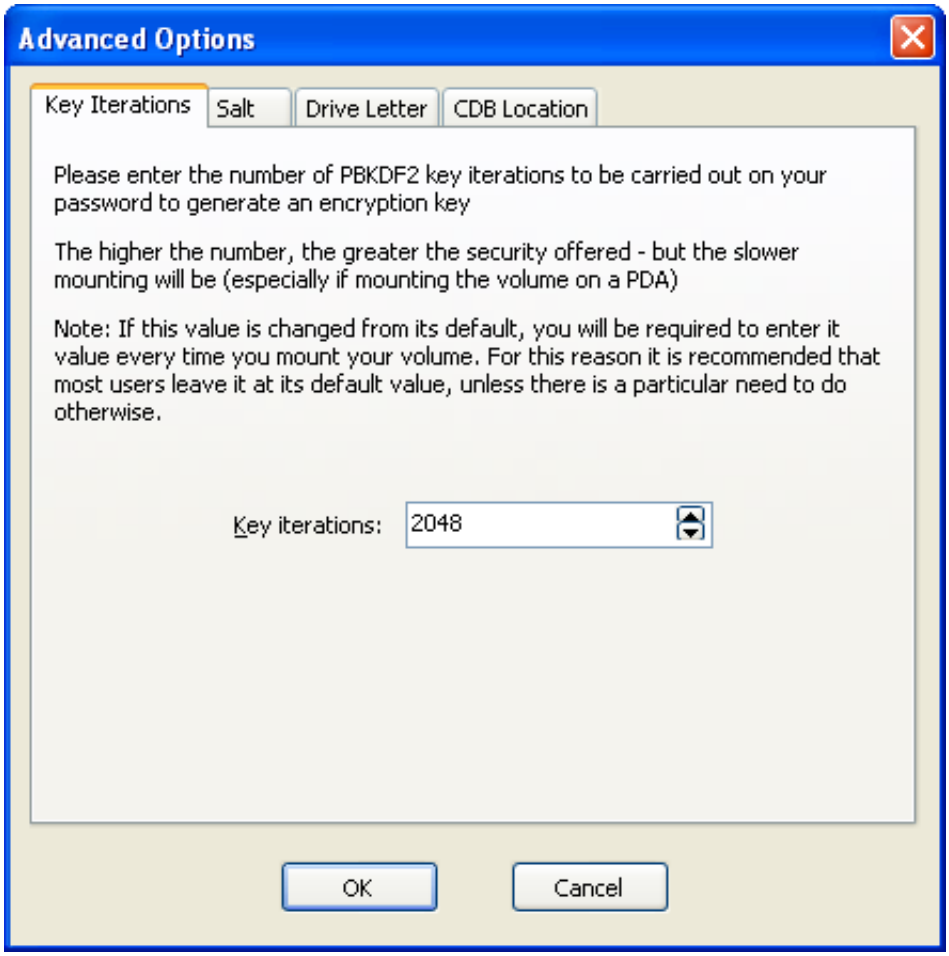
More than one hidden volume can be stored within the same host volume, by using different offsets

If you create a hidden volume within an existing volume, *be warned*: subsequently mounting and adding data to the *host* volume can potentially result in parts of the hidden volume being overwritten, and its data destroyed. This is by design, and increases the security of the hidden volume.

Please see the Plausible Deniability section for further information on the practical uses and considerations of hidden volumes.

5.3 Volume Creation: Advanced Options

At the end of the volume creation process, FreeOTFE Explorer will display a summary of the volume it is about to create. At this stage, more advanced options be configured for the new volume, by selecting the "Advanced..." button.



Advanced volume creation options

5.3.1 Key Iterations

Before the user's password is used to encrypt/decrypt the CDB, it is processed using PBKDF2 to increase security.

This tab allows the number of PBKDF2 iterations to be set by the user; higher values increase security, but will also increase the amount of time taken to mount the volume. This becomes more significant when mounting volumes on a PDA, which typically have slower CPUs.

The default number of key iterations is 2048.

5.3.2 Salt

Before the user's password is used to encrypt/decrypt the CDB, it is processed using PBKDF2 to increase security.

Part of this processing involves the use of a random "salt" value, which reduces the risk of dictionary based attacks. This tab allows the length of the salt value (in bits) to be set by the user.

It should be noted that every time a volume which has a non-default (256 bit) salt length is mounted, the user must specify the correct salt length (unless using a keyfile; in which case the keyfiles salt length must be specified) by using the "Advanced" options available on the FreeOTFE Explorer password entry dialog.

The default salt length is 256 bits. Any salt length entered must be a multiple of 8 bits.

5.3.3 Drive Letter

When mounting a volume using *FreeOTFE*, FreeOTFE will use the next available drive letter when mounting a volume.

This behaviour can be changed to use a specific drive letter on a volume-by-volume basis by setting it on this option.

The default setting here is "Use default"; use the next available drive letter

Note: If the chosen drive letter is in use at the time of mounting, the next free drive letter will be used

This setting has no effect on FreeOTFE Explorer, and it is only used when mounting volumes using FreeOTFE.

5.3.4 CDB Location

Normally, a volume's CDB will be stored as the first 512 bytes of the volume.

However, this does increase the size of the volume by the size of the CDB, which can FreeOTFE volumes more distinctive, and making it slightly more obvious that a volume file *is* volume file.

This is most clearly shown when creating a file based volume: a 2GB volume, for example, will be 2,147,484,160 bytes in length - made up of a 2,147,483,648 byte (2GB) encrypted disk image, plus a 512 byte embedded CDB.

To reduce this, it is possible to create a volume *without* an embedded CDB; the CDB begin stored in a separate file as a standard FreeOTFE Explorer keyfile.

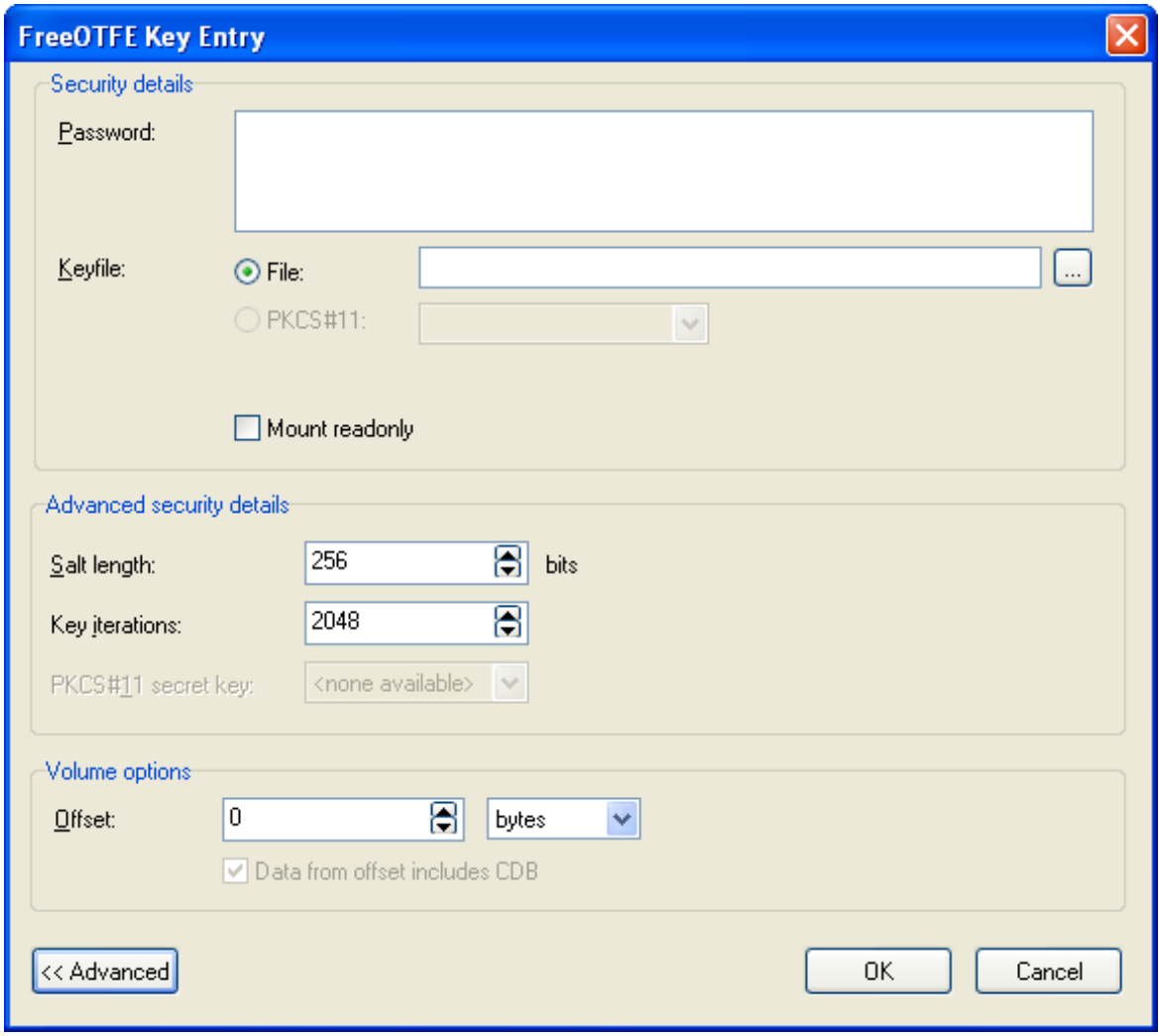
In this case, a 2GB volume would comprise of a 2,147,483,648 byte (2GB) encrypted disk image, plus a separate 512 byte keyfile which may be stored in a separate location to the volume.

Note that if you store the volume's CDB in a keyfile, you will *always* need to supply a keyfile when mounting the volume, and ensure that the "Data from offset includes CDB" advanced option shown on the FreeOTFE Explorer password entry dialog shown when mounting must be *unchecked after the keyfile is specified*.

By default, FreeOTFE Explorer includes the CDB will be included as part of the volume.

5.4 Password Entry: Advanced Options

Note: This section only covers the password entry dialog shown when mounting FreeOTFE volumes. For mounting Linux volumes, please see the section on Linux volumes.



Advanced mount options

5.4.1 Advanced Security Details

5.4.1.1 Salt length

This should be set to the number of salt bits used in the PBKDF2 processing of the user's password, before using it to decrypt the volume's CDB/keyfile being used.

By default, this is set to 256 bits - the same default length used when creating a new volume.

5.4.1.2 Key iterations

This should be set to the number of key iterations used in the PBKDF2 processing of the user's password, before using it to decrypt the volume's CDB/keyfile being used.

By default, this is set to 2048 iterations - the same default number used when creating a new volume.

5.4.1.3 PKCS#11 secret key

This option is only available if PKCS#11 support is enabled (see the section on Security Token/Smartcard Support for more information on how to use this setting.

5.4.2 Mount Options

5.4.2.1 Volume Options

These options are intended for use with hidden volumes, and volumes which were created without a CDB embedded at the start of the volume

Offset

When attempting to mount a *hidden volume*, this should be set to the offset (in bytes) where the hidden volume starts, as specified when creating it.

By default, this is set to an offset of 0 bytes.

Data from offset includes CDB

This checkbox is only enabled if a keyfile has been specified.

If you are attempting to mount either a hidden, or normal, volume which was created *without* a CDB embedded at the start of the volume, this checkbox should be changed so that it is *unchecked*.

For mounting all other volumes, this checkbox should be checked.

By default, this checkbox is checked.

6 Security Token/Smartcard Support

FreeOTFE Explorer supports all security tokens/smartcards (referred to as "tokens" in this documentation) which conform to the PKCS#11 (aka Cryptoki) standard, providing two factor authentication of FreeOTFE volumes.

There are two ways in which tokens can be used:

1. As a secure keyfile store
2. To add an additional level of encryption to keyfiles/volumes

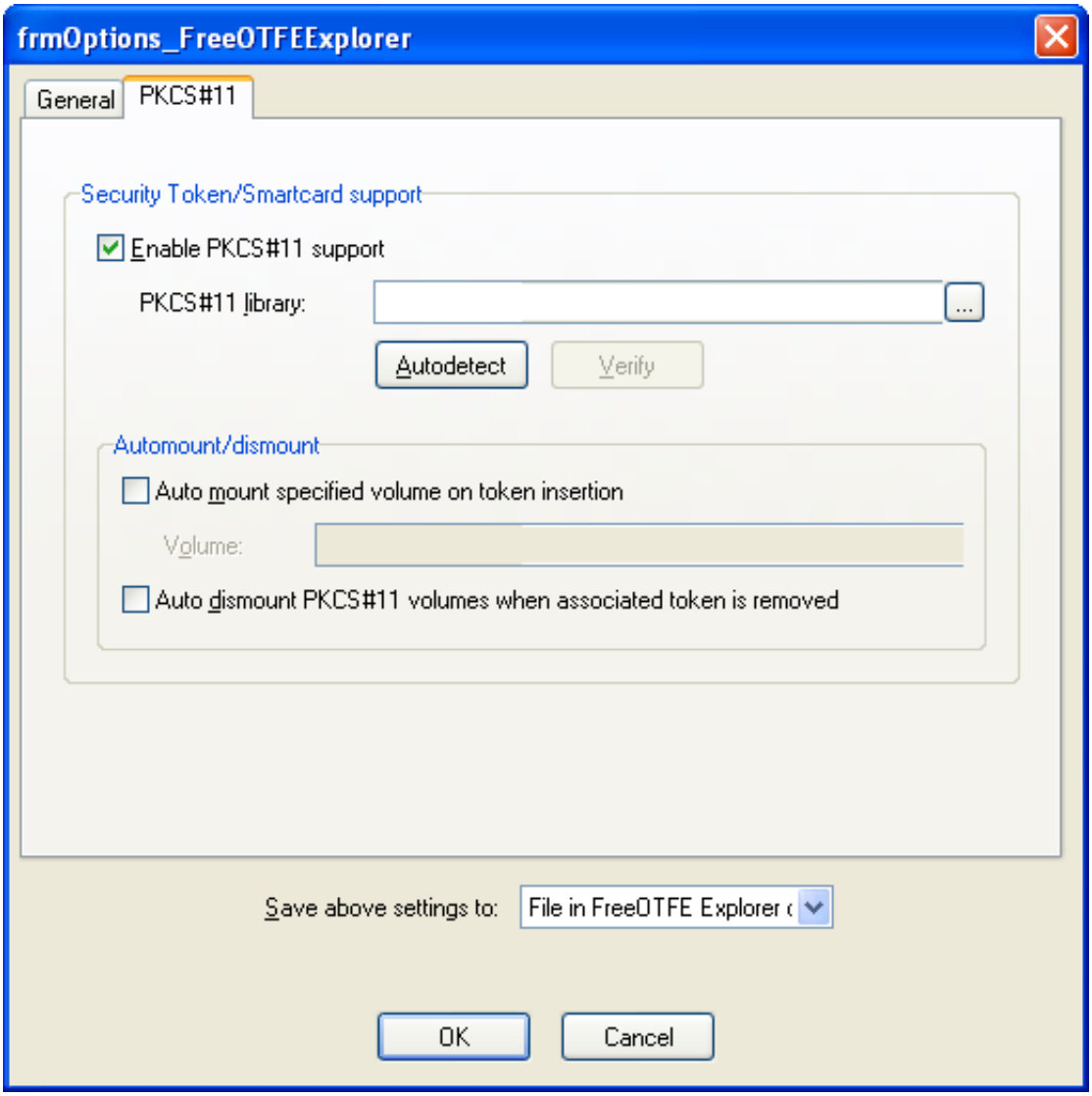
In both cases case, the token's password (typically called a "PIN" - although not limited to numbers) is required in order for the token to be used.

These two methods can be used independently, or combined together.

6.1 Initial configuration

In order to use tokens, FreeOTFE Explorer must first be configured to use the appropriate PKCS#11 library:

1. Go to "View | Options..."
2. Select the "PKCS#11" tab
3. Check the "Enable PKCS#11 support" checkbox
4. The filename of the PKCS#11 library you wish to use (see table below). Note: In most cases you shouldn't need to enter the full path to the DLL, just its filename.
5. Click "Verify" to run a quick sanity check to ensure that the library looks viable
6. Set "Save above settings to" be (for example) "FreeOTFE Explorer executable dir"
7. Click "OK"



Options dialog; PKCS#11 tab

The menuitem under the "Tools" menu should then be enabled, as should the options (when appropriate) on the password entry dialog when mounting FreeOTFE volumes

A list of driver library names supplied by common manufacturers may be found at Appendix E: PKCS#11 Driver Libraries

---

6.2 Secure Keyfile Store

Keyfiles may be stored on tokens in a similar fashion to which they can be stored on (for example) a USB flash drive. However, unlike storing a keyfile on a USB flash drive, those stored on a token require the token’s PIN to be entered before they can be accessed.

6.2.1 Usage

To add a keyfile to your token:

1. Create a keyfile for your volume as per normal
2. Plug in/insert your token
3. Go to "Tools | PKCS#11 Token management..."
4. Enter your token's PIN and click "OK"
5. The token management dialog should be displayed; select the "Keyfiles" tab
6. Click "Import..."
7. Select the keyfile previously created and click "OK".

To use a keyfile stored on a token:

1. Follow the normal procedure for mounting your FreeOTFE volume
2. When shown the password prompt, select "PKCS#11" as the keyfile option; you will then be prompted to authenticate yourself to the token
3. Enter your token's PIN and click "OK"
4. Select the keyfile stored on your token, and proceed as normal by entering your keyfile's password, etc and clicking "OK" to mount

Note: More than one keyfile can be stored on a single token; however they must all have different names.

6.3 Token Encryption

PKCS#11 tokens can also be used to add a further level of encryption to volumes, by using the token to encrypt the volume's CDB and/or keyfile(s).

The keys ("secret keys") used for this encryption are automatically generated by a token and can never be duplicated, extracted or in any way copied from the token, *even if the token's PIN is known*. All encryption/decryption operations used to secure a keyfile/volume CDB are carried out *by the token itself*.

This mechanism therefore provides a means of "tying" a volume/keyfile to a physical token; preventing it from being mounted unless the token is present and its PIN is known.

It should be noted however, that since it is inherent that *no backups of the secret keys stored on a token can be made*, the loss of the token will result in the loss of all data stored on the volume it protects, unless a separate means of accessing the volume (e.g. a keyfile which isn't secured by the same PKCS#11 token) is available.

6.3.1 Usage

To encrypt a volume's CDB/keyfile:

1. Plug in/insert your token
2. Go to "Tools | PKCS#11 Token management..."
3. Enter your token's PIN and click "OK"

4. The token management dialog should be displayed; select the "Secret keys" tab
5. Click "New..."
6. Enter a meaningful name of the token, and select the cypher to be used for the encryption
   It should be noted that the range of cyphers available for use is determined by the capabilities of your token, and not FreeOTFE Explorer
7. Click "OK" and the new key will be created
8. Select the secret key to be used to encrypt your volume CDB/keyfile.
9. Click "Secure..."
10. In the dialog shown, specify the volume/keyfile to be encrypted. If you are trying to secure a hidden volume, enter the host volume's filename/partition.
11. Leave the "offset" field set to zero, unless you are trying to secure a hidden volume - in which case this should be set to the offset where the hidden volume may be found.
12. Click "OK".

Note:

- More than one secret key can be stored on a single token; however they must all have different names.
- The same secret key can be used to encrypt more than one volume CDB/keyfile.

To use a volume/keyfile which has been double-encrypted by a token:

1. Follow the normal procedure for mounting your FreeOTFE volume
2. When shown the password prompt, click the "Advanced >>" button to display the "Advanced security details" options
3. Enter your token's PIN and click "OK"
4. Select the secret key used to secure your volume/keyfile, and proceed as normal by entering your volume/keyfile's password, etc and clicking "OK" to mount

---

6.4 PIN Entry

FreeOTFE Explorer will only prompt you to enter your token's PIN as and when it's needed.
FreeOTFE Explorer does *not* cache your PIN in any way

PKCS#11 PIN entry

The PIN entry prompt will display a list of all tokens found on your system, showing the slot number the token is inserted in, and the token’s label. If you have not yet inserted your token, do so and click "Refresh" to refresh the list.

If only one token is found, it will be selected automatically, and the token selection control will be disabled. Otherwise, select the token you wish to use, enter your PIN, and click "OK" to continue.

6.4.1 Secure authentication path

If your token hardware features a secure authentication path (e.g. a smartcard reader with PIN entry keypad), you can take advantage of it by selecting the "Use secure authentication path" checkbox when FreeOTFE Explorer prompts for the token’s PIN.

7 Linux Volumes

IMPORTANT: This is obvious, *but*... If you are using FTP to transfer your Linux volumes between your Linux and MS Windows systems, *make sure you transfer the volume file in binary mode!*

7.1 Creating Linux Volumes

IMPORTANT: If you select the wrong options when creating a Linux volume using FreeOTFE Explorer, you will not be able to read it under Linux! (Although this is patently obvious, there are some people who...!)

NOTE: At time of writing (17th July 2005), although FreeOTFE Explorer can read and write LUKS volumes, it cannot create them itself.
To create a new encrypted Linux-compatible volume:

1. Launch FreeOTFE Explorer
2. If you are creating a file-based volume (as opposed to an encrypted partition):
    1. Select "File | Linux volume | New..."
    2. Enter the filename and size of the volume required.
    3. Click the "OK" button This will create a file of the appropriate size, and is the equivalent of the Linux command:

        ```
        dd if=/dev/zero of=./vol_none bs=1M count=x
        ```

        Where "*x*" is the size of the volume in MB.
        This step is unnecessary in case of using an existing partition on your HDD, and is only required in order to create the file which is to store the encrypted data.
3. Select "File | Linux volume | Mount..."
4. Enter the appropriate parameters into the mount dialog.
   See the "Linux Examples" section in the documentation for example configurations; the choices you make here must reflect those options that are supported by the version of Linux you wish to subsequently use the volume with.
5. Click the "OK" button This will mount the volume using the encryption system(s) specified, and is analogous to executing the Linux commands:

    ```
    losetup /dev/loop0 <volume file> <various options>mkdir ./mountpointmount /dev/loop0 ./mountpoint
    ```

    At this point a new drive should appear, although at this stage it is unformatted and you will be given an error if you attempt to access it (e.g. using Windows Explorer)
6. Select the new drive
7. Select "Tools | Format..."
   You will be shown the standard MS Windows format dialog. It is suggested that you select either the FAT or FAT32 filesystem.

8. Click "OK" button to format the volume.
   This is analogous to the Linux command:

    ```
    mkdosfs /dev/loop0
    ```

Your volume file is now fully ready for use, although for security reasons it is highly recommended that you now initialize the volume by writing data to all its sectors before making any further use of it.

This process is recommended because if omitted it may be possible for an attacker to determine the amount of data you have stored on your volume file (as stated above, the actual process of creating the volume file consists of creating a suitably large file, filled with zeros)
Note: It is important that this step be carried out if you intend using the volume file just created as a "host" file for storing a second, hidden, encrypted volume.

9. Switch back to FreeOTFE Explorer, and select "Tools | Overwrite free space..."
10. Click "Yes" when prompted if you wish to proceed.
    FreeOTFE Explorer will then write pseudorandom data to the drive, which will then be encrypted before being written to the volume file.

7.2 Hiding a Linux Volume Within Another Volume

To create a Linux volume within another volume file:

1. Create a FreeOTFE/Linux volume as normal, ensuring that you initialize the volume by mounting it, formatting it, and then overwriting all its free space.
2. Unmount the "host" volume
3. Remount the "host" volume, but specify a reasonable offset on the "File options" tab of the Linux mount dialog.

7.3 Mounting Volumes Created under Linux

Select "File | Linux volume | Mount file.../Mount partition".
Enter the volume's password, and set all appropriate options
Click "OK".

Note that if you do not:

1. Set the same options as used when the volume is mounted and used while under Linux
2. Format the volume using a filesystem MS Windows understands (i.e. NTFS/FAT/FAT32)

then although your Linux volume may well be mounted, its contents will probably be unreadable.

7.4 Cryptoloop Specific Information

7.4.1 Hash Selection

Cryptoloop ("losetup") Linux volumes use the hash of the user's key as the key used for reading/writing to the encrypted volume.

7.4.2 Cryptoloop and RIPEMD-160

If you pass "-H rmd160" to losetup in order to use RIPEMD-160 to process your password, losetup's behaviour changes slightly. The user's password is not simply hashed with RIPEMD-160 - instead, the following procedure is used:

1. The user's password is hashed once using RIPEMD-160
2. A copy of the first 129 characters of the user's password is made
3. The capital letter "A" is prepended to the start of the copy
4. The resulting string is then hashed with RIPEMD-160
5. This hash is then appended to the first hash to produce 320 bits of data
6. The appropriate number of bits is taken from the result, and used as the encryption/decryption key.

For this reason, FreeOTFE Explorer includes a RIPEMD-160 driver specifically modified ("RIPEMD-160 (Linux; Twice, with A)") to carry out this form of hashing.

(This does not appear to be documented; the above logic was derived from examining "util-linux-2.12p.diff" - one of the files included with loop-AES)

7.4.3 Cypher Selection

If the cypher selected ("-e" parameter passed as losetup) can support different keysizes (e.g. AES supports 128/192/256 bit keysizes), and the user doesn't specify the keysize to be used (i.e. you specify "-e AES" as opposed to "-e AES128"), then the cypher will default to using 128 bit keys.

(From: http://loop-aes.sourceforge.net/loop-AES.README)

7.5 dm-crypt Specific Information

7.5.1 /dev/loop1 Usage in the Examples Included in this Documentation

The examples shown in this documentation include the slightly odd step of:

```
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
...
```

as opposed to just straight:

```
mkdosfs /dev/mapper/myMapper
...
```

This is carried out as (in my tests) the latter appears to result in failure:

```
# mkdosfs /dev/mapper/myMapper
mkdosfs 2.8 (28 Feb 2001)
mkdosfs: unable to get drive geometry for '/dev/mapper/myMapper'
```

7.5.2 Hash Selection

If an attempt is made to mount a volume using a cypher with a larger keysize than the hash algorithm used to processes the user's password, dm-crypt appears to use the following algorithm to generate the actual encryption/decryption key used by the cypher:

1. The user's password is hashed.
2. If the hash output contains fewer bits than the cypher's keysize, the capital letter "A" is prepended to the user's password, and a new hash is generated.
3. This second hash is appended to the previous one
4. If the total hash output contains fewer bits than the cypher's keysize, another capital letter "A" is prepended to the user's password, and a new hash is generated.
5. This third hash is appended to the previous hashes
6. If the total hash output contains fewer bits than the cypher's keysize, another capital letter "A" is prepended to the user's password, and a new hash is generated.
7. ...and so on, until the required keylength is reached.

i.e. This is the same as Cryptoloop uses for its RIPEMD-160 hashing, but is extended to produce a key of arbitrary length, by adding multiple "A" characters to the password and hashing, until a key of the required length is obtained.

FreeOTFE Explorer supports this form of key processing, which can be invoked by selecting the option "Hash with "A"s, if hash output is too short" on the Linux mount dialog.

Note that, under linux, the actual encryption/decryption key can be shown in its hex representation by running "dmsetup table".

For example, if the volume's password is "password1234567890ABC", then:

If AES (256 bit key) is used for encryption/decryption, and the user's password is processed with RIPEMD-160, the actual encryption/decryption key will be:

```
FAFE56C3BAB4CD216BA02474AC157EA555FA5711D539285C28A6D8122D9464EE
```

This is made up as follows:

`FAFE56C3BAB4CD216BA02474AC157EA555FA5711`	The first 160 bits are the RIPEMD-160 hash of "password1234567890ABC"
`D539285C28A6D8122D9464EE0AA3C4811DE0D846`	The remaining bits are the first 96 bits taken from the RIPEMD-160 hash of "Apassword1234567890ABC"

If Blowfish (448 bit key) is used for encryption/decryption, and the user's password is processed with MD5, the actual encryption/decryption key will be:

```
4EAB90A0D00CE0086EB59DA838CC888DD1270498F52EFFA562872664BB514F8E2FA054980C9D92542F5801FDF82ADFEA121E587A4EEBDF3B
```

This is made up as follows:

4EAB90A0D00CE0086EB59DA838CC888D	The first 128 bits are the MD5 hash of "password1234567890ABC"
D1270498F52EFFA562872664BB514F8E	The next 128 bits are the MD5 hash of "Apassword1234567890ABC"
2FA054980C9D92542F5801FDF82ADFEA	The next 128 bits are the MD5 hash of "AApassword1234567890ABC"
121E587A4EEBDF3BD6CD437A1B2C32A	The remaining bits are the first 64 bits taken from the MD5 hash of "AAApassword1234567890ABC"

7.5.3 ESSIV

dm-crypt's ESSIV functionality is available with v2.6.10 and later Linux kernels.

The manner in which Linux uses ESSIV differs from FreeOTFE volumes in how the ESSIV encryption key is generated. Both hash the master encryption/decryption key to generate the key used for ESSIV, however dm-crypt uses the full hash output as the ESSIV key. This means that if you have a dm-crypt volume which is encrypted using 256 bit AES, and specify MD5 as the ESSIV hash, the ESSIV process will actually use AES-128 for creating the "salt" for ESSIV IVs (MD5 generates 128 bit hashes).

It is for this reason, you cannot create a dm-crypt volume under Linux using 256 bit Twofish, and specify SHA-512 as the ESSIV hash; Twofish doesn't support 512 bit keys, and so dm-crypt fails.

7.6 LUKS Specific Information

As LUKS is based on dm-crypt, please see also the section above relating to dm-setup.

FreeOTFE Explorer supports LUKS to v1.1 of the LUKS specification. This is the latest version at time of writing (2nd December 2007)

As well as using the "File | Linux | Mount file/partition..." menu items, LUKS volumes may also be mounted using the main "File | Mount file/partition..." menu items and toolbar buttons. (FreeOTFE Explorer detects LUKS volumes by their signature and offers to mount them appropriately)

7.6.1 ESSIV

FreeOTFE Explorer supports LUKS with ESSIV, subject to the condition that the ESSIV hash used generates hashes with the same, or less, bits than the cypher's block size.

Also at time of writing (25th February 2007), the current LUKS implementation of "cryptsetup" only supports the SHA1 hash algorithm, although other hashes may be used for ESSIV.

Because of the way in which dm-crypt operates (see also the "dm-crypt" section on ESSIV, above), LUKS ESSIV doesn't do what you'd probably expect it to do. Specifically, if you have (for example) a Blowfish-448 encrypted volume, and specify cbc-essiv:sha256 for use as IVs - LUKS (dm-crypt) will actually use Blowfish-256 as the ESSIV cypher, and not Blowfish-448. In other words, the ESSIV cypher used will be from the same "family" of cypher (AES, Blowfish, Serpent, etc) - but will use the keylength which matches the ESSIV hash output length.

As a result of this, another option appears on the LUKS password entry dialog; "Base IV cypher on hash length". If this is checked, then when mounting an ESSIV volume, the keylength of the cypher used for ESSIV generation will be that of the ESSIV hash. If this is unchecked, the ESSIV cypher used will have the same keylength as the main bulk encryption cypher used for securing the encrypted disk image.

Most users will want this option checked.

7.6.2 FreeOTFE Explorer Supported LUKS Cyphers

The following table lists compatibility with LUKS cyphers:

Cypher	Compatibility
aes	Supported by FreeOTFE Explorer.
twofish	Supported by FreeOTFE Explorer.
serpent	Supported by FreeOTFE Explorer.
cast5	Supported by FreeOTFE Explorer.
cast6	Supported by FreeOTFE Explorer.

7.6.3 FreeOTFE Explorer Supported LUKS Cypher modes

The following table lists compatibility with LUKS cypher modes:

Mode	Compatibility
ecb	Not supported by FreeOTFE Explorer. Note: This is a pretty insecure mode - the use of ECB is highly discouraged, and FreeOTFE Explorer is unlikely to ever support this mode.
cbc-plain	Supported by FreeOTFE Explorer.
cbc-essiv:<hash>	Supported by FreeOTFE Explorer
xts-plain	Supported by FreeOTFE Explorer

7.6.4 FreeOTFE Explorer Supported LUKS hashes

The following table lists compatibility with LUKS hashes:

Hash	Compatibility
sha1	Supported by FreeOTFE Explorer.
sha256	Supported by FreeOTFE Explorer.
sha512	Supported by FreeOTFE Explorer.
ripemd160	Supported by FreeOTFE Explorer.

7.7 Additional Notes on Linux Volumes

Linux volumes should be formatted as FAT/FAT32/NTFS in order for them to be recognised by MS Windows. Although it should be possible for MS Windows can to understand other filesystems (e.g. ext2/ext3/riserFS), this does require 3rd party filesystem drivers to be installed.

If you do wish to read an ext2/ext3 formatted volume from MS Windows, the filesystem drivers listed below are suggested. There may well be others, though at time of writing (23rd December 2005) these are the only ones that I have checked:

Package	URL	Description
Ext2 Installable File System For Windows	http://www.fs-driver.org/	Supports both read and write operations with ext2/ext3. Freeware, but closed source.
EXT2 IFS for Windows	http://uranus.it.swin.edu.au/~jn/linux/ext2ifs.htm	Supports ext2, readonly. Open source.

Further information on Linux volumes may be obtained from:

Cryptoloop	Cryptoloop HOWTO
loop-AES	loop-AES README
dm-crypt	dm-crypt WWW site dm-crypt Wiki
LUKS	LUKS - Linux Unified Key Setup

Note that for many of the controls on FreeOTFE Explorer’s Linux mount volume dialog, the equivalent Cryptoloop ("losetup") parameter for that control is displayed in brackets.

7.8 Examples: Cryptoloop

This section gives a series of examples of how to create Linux Cryptoloop (losetup) volumes, and then mount them using FreeOTFE Explorer.

These examples have been tested using SuSE 9.2; though they should work for all compatible Linux distributions.

7.8.1 Initial Setup

To begin using Cryptoloop under Linux, ensure that the various kernel modules are installed:

```
modprobe cryptoloop

modprobe deflate
modprobe zlib_deflate
modprobe twofish
modprobe serpent
modprobe aes_i586
modprobe blowfish
modprobe des
modprobe sha256
modprobe sha512
modprobe crypto_null
modprobe md4
modprobe md5
modprobe arc4
modprobe khazad
modprobe anubis
```

Typing "lsmod" will show you which modules are currently installed.
The examples shown below may then be followed to create and use various volume files.

7.8.2 Defaults

If not overridden by the user, Cryptoloop defaults to no encryption. If the user specifies that they do want encryption (i.e. passes "losetup" a "-e" parameter), Cryptoloop defaults to the following:

<table>
<tr><td>Cypher:</td><td>As specified by the user (no encryption takes place if no cypher is specified)</td></tr>
<tr><td>Cypher keysize:</td><td>128 bit</td></tr>
<tr><td>User key processed with:</td><td>The hash used to process the user’s key is dependant on the cypher’s keysize:
<table><tr><th>Cypher keysize</th><th>Hash</th></tr><tr><td>128 - 191 bits</td><td>SHA-256</td></tr><tr><td>192 - 255 bits</td><td>SHA-384</td></tr><tr><td>256+ bits</td><td>SHA-512</td></tr></table>
"Hash with "A"s, if hash output is too short" option - this option should not be selected; if the hash used outputs too few bits, its output is right-padded with 0x00 characters to the required length.</td></tr>
<tr><td>IV generation:</td><td>32 bit sector ID</td></tr>
</table>

7.8.3 Example #1: Volume Without Encryption

This is the simplest form of Linux volume file, and the recommended starting point for checking that FreeOTFE Explorer is operating correctly.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_none bs=1k count=1024
losetup /dev/loop0 ./vol_none
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Leave key blank
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "Null" hash
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "Null" cypher
   - Select the "Null IV" IV generation method
   - The "Hash with "A"s, if hash output is too short" makes no difference
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.8.4 Example #2: Volume Encrypted Using XOR

This is the second simplest form of Linux volume file, and is the simplest case to confirm that passwords are being accepted and used correctly.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_xor bs=1k count=1024
losetup -e XOR /dev/loop0 ./vol_xor
# Enter password: password1234567890ABC
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "Null" hash
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "XOR" cypher
   - Select the "Null IV" IV generation method
   - The "Hash with "A"s, if hash output is too short" makes no difference.
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.8.5 Example #3: Volume Encrypted Using 128 bit AES

This example demonstrates use of a Linux AES128 volume.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_aes128 bs=1k count=1024
losetup -e AES128 /dev/loop0 ./vol_aes128
# Enter password: password1234567890ABC
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "SHA-256 (256/512)" hash
   - Make sure that the "Hash with "A"s, if hash output is too short" is not checked.

- Leave iteration count at 0

4. "Encryption" tab:
   - Select the "AES (CBC; 128/128)" cypher
   - Select the "32 bits sector IV" IV generation method
   - Set "Sector zero location" to "Start of host file"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.8.6 Example #4: Volume Encrypted Using 256 bit AES

This example demonstrates use of a Linux AES256 volume.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_aes256 bs=1k count=1024
losetup -e AES256 /dev/loop0 ./vol_aes256
# Enter password: password1234567890ABC
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "SHA-512 (512/1024)" hash
   - Make sure that the "Hash with "A"s, if hash output is too short" is not checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 256/128)" cypher
   - Select the "32 bits sector IV" IV generation method
   - Set "Sector zero location" to "Start of host file"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter

- Leave readonly unchecked

7. Click the "OK" button

7.8.7 Example #5: Volume Encrypted Using 256 bit AES and rmd160 Hash

This example demonstrates use of a Linux AES256 volume using the rmd160 hash to process the user's password instead of the default SHA hash.

WARNING: Note that this example uses the "rmd160" and not "ripemd160" hash.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_aes256_rmd160 bs=1k count=1024
losetup -e AES256 -H rmd160 /dev/loop0 ./vol_aes256_rmd160
# Enter password: password1234567890ABC
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
cp TEST_FILE_1.dat ./test_mountpoint
cp TEST_FILE_2.dat ./test_mountpoint
cp TEST_FILE_3.dat ./test_mountpoint
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "RIPEMD-160 (Linux; Twice, with A)" hash
   - Make sure that the "Hash with "A"s, if hash output is too short" is not checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 256/128)" cypher
   - Select the "32 bits sector IV" IV generation method
   - Set "Sector zero location" to "Start of host file"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.8.8 Example #6: Volume Encrypted Using 256 bit AES and Seed Value

This example demonstrates use of a Linux AES256 volume with seeding. The seed used here is the string "seedvalue"

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_aes256_seeded bs=1k count=1024
losetup -e AES256 -S seedvalue /dev/loop0 ./vol_aes256_seeded
# Enter password: password1234567890ABC
losetup -a
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
cp TEST_FILE_1.dat ./test_mountpoint
cp TEST_FILE_2.dat ./test_mountpoint
cp TEST_FILE_3.dat ./test_mountpoint
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Change the seed to "seedvalue"
   - Select the "SHA-512 (512/1024)" hash
   - Make sure that the "Hash with "A"s, if hash output is too short" is not checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 256/128)" cypher
   - Select the "32 bits sector IV" IV generation method
   - Set "Sector zero location" to "Start of host file"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.8.9 Example #7: Volume Encrypted Using 256 bit AES and Offset

This example demonstrates use of a Linux AES256 volume, with the encrypted volume beginning at an offset of 2560 bytes into the volume file.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_aes256_2560 bs=1k count=1024
losetup -e AES256 -o 2560 /dev/loop0 ./vol_aes256_2560
# Enter password: password1234567890ABC
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "SHA-512 (512/1024)" hash
   - Make sure that the "Hash with "A"s, if hash output is too short" is not checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 256/128)" cypher
   - Select the "32 bits sector IV" IV generation method
   - Set "Sector zero location" to "Start of host file"
5. "File options" tab:
   - Change offset to 2560 bytes
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.8.10 Example #8: Volume Encrypted Using 256 bit Twofish

This example demonstrates use of a Linux Twofish 256 bit volume.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./vol_twofish256 bs=1k count=1024
losetup -e twofish256 /dev/loop0 ./vol_twofish256
# Enter password: password1234567890ABC
losetup -a
mkdosfs /dev/loop0
mkdir ./test_mountpoint
mount /dev/loop0 ./test_mountpoint
echo "This is a text test file" > ./test_mountpoint/SHORT_TEXT.txt
umount /dev/loop0
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "SHA-512 (512/1024)" hash
   - Make sure that the "Hash with "A"s, if hash output is too short" is not checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "Twofish (CBC; 256/128)" cypher
   - Select the "32 bits sector IV" IV generation method
   - Set "Sector zero location" to "Start of host file"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9 Examples: dm-crypt

This section gives a series of examples of how to create Linux dm-crypt volumes, and then mount them using FreeOTFE Explorer.

These examples have been tested using Fedora Core 3, with a v2.6.11.7 kernel installed; though they should work for all compatible Linux distributions.

7.9.1 Initial Setup

To begin using dm-crypt under Linux, ensure that the various kernel modules are installed:

```
modprobe cryptoloop

modprobe deflate
modprobe zlib_deflate
modprobe twofish
modprobe serpent
modprobe aes_i586
modprobe blowfish
modprobe des
modprobe sha256
modprobe sha512
modprobe crypto_null
modprobe md5
modprobe md4
modprobe cast5
modprobe cast6
modprobe arc4
modprobe khazad
modprobe anubis

modprobe dm_mod (this should give you dm_snapshot, dm_zero and dm_mirror?)
modprobe dm_crypt
```

At this point, typing "dmsetup targets" should give you something along the lines of:

```
crypt            v1.0.0
striped          v1.0.1
linear           v1.0.1
error            v1.0.1
```

Typing "lsmod" will show you which modules are currently installed.

7.9.2 Defaults

If not overridden by the user, dm-crypt defaults to encrypting with:

Cypher:	AES
Cypher keysize:	256 bit
User key processed with:	RIPEMD-160 (not "RIPEMD-160 (Linux; Twice, with A)"). "Hash with "A"s, if hash output is too short" option - selected
IV generation:	32 bit sector ID

7.9.3 Example #1: Volume Encrypted Using dm-crypt’s Default Encryption

This example demonstrates use of a dm-crypt volume using the dm-crypt’s default encryption system: AES128 with the user’s password hashed with RIPEMD160, using the 32 bit sector IDs as encryption IVs

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_default.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_default.vol
echo password1234567890ABC | cryptsetup create myMapper /dev/loop0
dmsetup ls
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "RIPEMD-160 (160/512)" hash
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 256/128)" cypher
   - Select "32 bit sector ID" as the IV generation method
   - Set "Sector zero location" to "Start of encrypted data"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9.4 Example #2: Volume Encrypted Using 128 bit AES

This example demonstrates use of a dm-crypt AES128 volume.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_aes128.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_aes128.vol
echo password1234567890ABC | cryptsetup  -c aes -s 128 create myMapper /dev/loop0
dmsetup ls
```

```
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "RIPEMD-160 (160/512)" hash.
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 128/128)" cypher
   - Select "32 bit sector ID" as the IV generation method
   - Set "Sector zero location" to "Start of encrypted data"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9.5 Example #3: Volume Encrypted Using 256 bit AES, using SHA256 ESSIV

This example demonstrates use of a dm-crypt AES256 volume using SHA-256 ESSIV sector IVs.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_aes_essiv_sha256.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_aes_essiv_sha256.vol
echo password1234567890ABC | cryptsetup  -c aes-cbc-essiv:sha256 create myMapper /dev/loop0
dmsetup ls
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
```

```
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "RIPEMD-160 (160/512)" hash
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 256/128)" cypher
   - Select "ESSIV" as the IV generation method
   - Set "Sector zero location" to "Start of encrypted data"
   - Select "SHA-256 (256/512)" as the IV hash
   - Select "AES (CBC; 256/128)" as the IV cypher
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9.6 Example #4: Volume Encrypted Using 448 bit Blowfish

This example demonstrates use of a dm-crypt Blowfish 448 volume.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_blowfish_448.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_blowfish_448.vol
echo password1234567890ABC | cryptsetup -c blowfish -s 448 create myMapper /dev/loop0
dmsetup ls
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
```

```
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "RIPEMD-160 (160/512)" hash
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "Blowfish (CBC; 448/64)" cypher
   - Select "32 bit sector ID" as the IV generation method
   - Set "Sector zero location" to "Start of encrypted data"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9.7 Example #5: Volume Encrypted Using 256 bit Twofish and Offset

This example demonstrates use of a dm-crypt Twofish 256 volume, with the encrypted volume beginning at an offset of 3 sectors (3 x 512 = 1536 bytes) into the volume file.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_twofish_o3.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_twofish_o3.vol
echo password1234567890ABC | cryptsetup -c twofish -o 3 create myMapper /dev/loop0
dmsetup ls
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
```

```
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "RIPEMD-160 (160/512)" hash
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "Twofish (CBC; 256/128)" cypher
   - Select "32 bit sector ID" as the IV generation method
   - Set "Sector zero location" to "Start of encrypted data"
5. "File options" tab:
   - Set offset to 1536 bytes (i.e. 3 sectors, each of 512 bytes)
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9.8 Example #6: Volume Encrypted Using 256 bit AES with MD5 Password Hashing

This example demonstrates use of a dm-crypt Twofish 256 volume, with the user’s password processed with MD5.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_aes_md5.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_aes_md5.vol
echo password1234567890ABC | cryptsetup -c aes -h md5 create myMapper /dev/loop0
dmsetup ls
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
```

```
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "MD5 (128/512)" hash
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (CBC; 256/128)" cypher
   - Select "32 bit sector ID" as the IV generation method
   - Set "Sector zero location" to "Start of encrypted data"
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9.9 Example #7: Volume Encrypted Using 448 bit Blowfish, MD5 Password Hashing and SHA-256 ESSIV

This example demonstrates use of a dm-crypt Blowfish 448 volume, with the user’s password processed with MD5 and ESSIV using SHA-256.

Note that although the main cypher is Blowfish 448, Blowfish 256 is used as the IV cypher as the IV hash outputs 256 bytes

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_blowfish_448_essivsha256_md5.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_blowfish_448_essivsha256_md5.vol
echo password1234567890ABC | cryptsetup -c blowfish-cbc-essiv:sha256 -s 448 -h md5 create myMapper /dev/loop0
dmsetup ls
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
```

```
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "MD5 (128/512)" hash
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "Blowfish (CBC; 448/64)" cypher
   - Select "ESSIV" as the IV generation method
   - Set "Sector zero location" to "Start of encrypted data"
   - Select "SHA-256 (256/512)" as the IV hash
   - Select "Blowfish (CBC; 256/64)" as the IV cypher
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.9.10 Example #8: Volume Encrypted Using AES-256 in XTS Mode (aka XTS-AES-256)

This example demonstrates use of a dm-crypt AES-256 volume in XTS mode (aka XTS-AES-256) and using SHA-512 for hashing

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_aes_xts.vol bs=1K count=100
losetup /dev/loop0 ./volumes/vol_aes_xts.vol
echo password1234567890ABC | cryptsetup -h sha512 -c aes-xts-plain --key-size 512 create myMapper /dev/loop0
dmsetup ls
dmsetup table
dmsetup status
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup remove myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. "Key" tab:
   - Enter "password1234567890ABC" as the key
   - Leave GPG executable blank
   - Leave GPG keyfile blank
   - Leave seed blank
   - Select the "SHA-512 (512/1024)" hash
   - Ensure "Hash with "A"s, if hash output is too short" is checked.
   - Leave iteration count at 0
4. "Encryption" tab:
   - Select the "AES (256 bit XTS)" cypher
   - Select "Null IV" as the IV generation method
5. "File options" tab:
   - Leave offset at 0
   - Leave sizelimit at 0
6. "Mount options" tab:
   - Select any unused drive letter
   - Leave readonly unchecked
7. Click the "OK" button

7.10 Examples: LUKS

This section gives a series of examples of how to create Linux LUKS volumes, and then mount them using FreeOTFE Explorer.

These examples have been tested using Fedora Core 3, with a v2.6.20.1 kernel installed and using cryptsetup-luks v1.0; though they should work for all compatible Linux distributions.

Note: The executable name in the following examples is "cryptsetup-luks"; most systems use "cryptsetup".

7.10.1 Initial Setup

To begin using LUKS under Linux, ensure that the various kernel modules are installed:

```
modprobe cryptoloop

modprobe aes
modprobe anubis
modprobe arc4
modprobe blkcipher
modprobe blowfish
modprobe cast5
modprobe cast6
modprobe cbc
modprobe crc32c
modprobe crypto_algapi
modprobe crypto_hash
```

```
modprobe cryptomgr
modprobe crypto_null
modprobe deflate
modprobe des
modprobe ecb
modprobe gf128mul
modprobe hmac
modprobe khazad
modprobe lrw
modprobe md4
modprobe md5
modprobe michael_mic
modprobe serpent
modprobe sha1
modprobe sha256
modprobe sha512
modprobe tea
modprobe tgr192
modprobe twofish_common
modprobe twofish
modprobe wp512
modprobe xcbc

# dm_mod should give you dm_snapshot, dm_zero and dm_mirror?
modprobe dm_mod
modprobe dm_crypt
```

At this point, typing "dmsetup targets" should give you something along the lines of:

```
crypt            v1.0.0
striped          v1.0.1
linear           v1.0.1
error            v1.0.1
```

Typing "lsmod" will show you which modules are currently installed.

7.10.2 Defaults

If not overridden by the user, LUKS defaults to encrypting with:

Cypher:	AES
Cypher keysize:	128 bit
Cypher mode:	cbc-plain
Hash:	SHA-1

7.10.3 Example #1: Volume Encrypted Using LUKS's Default Encryption

This example demonstrates use of a LUKS volume using the LUKS's default encryption system: AES128 with the user's password hashed with SHA1, using 32 bit sector IDs as encryption IVs

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_default.vol bs=1M count=1
losetup /dev/loop0 ./volumes/vol_default.vol
echo password1234567890ABC | cryptsetup-luks luksFormat /dev/loop0
cryptsetup-luks luksDump /dev/loop0
echo password1234567890ABC | cryptsetup-luks luksOpen /dev/loop0 myMapper
dmsetup ls
dmsetup table
dmsetup status
cryptsetup-luks status myMapper
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup-luks luksClose myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the volume file
3. In the dialog shown, enter "password1234567890ABC" as the key, and set any of the options wanted.
4. Click the "OK" button

7.10.4 Example #2: Volume Encrypted Using 256 bit AES

This example demonstrates use of a LUKS AES256 volume.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_aes_256.vol bs=1M count=1
losetup /dev/loop0 ./volumes/vol_aes_256.vol
echo password1234567890ABC | cryptsetup-luks -c aes -s 256 luksFormat /dev/loop0
cryptsetup-luks luksDump /dev/loop0
echo password1234567890ABC | cryptsetup-luks luksOpen /dev/loop0 myMapper
dmsetup ls
dmsetup table
dmsetup status
cryptsetup-luks status myMapper
losetup /dev/loop1 /dev/mapper/myMapper
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup-luks luksClose myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the losetup volume file
3. In the dialog shown, enter "password1234567890ABC" as the key, and set any of the options wanted.
4. Click the "OK" button

7.10.5 Example #3: Volume Encrypted Using 128 bit Twofish

This example demonstrates use of a LUKS Twofish 128 volume.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_twofish.vol bs=1M count=1
losetup /dev/loop0 ./volumes/vol_twofish.vol
echo password1234567890ABC | cryptsetup-luks -c twofish luksFormat /dev/loop0
cryptsetup-luks luksDump /dev/loop0
echo password1234567890ABC | cryptsetup-luks luksOpen /dev/loop0 myMapper
dmsetup ls
dmsetup table
dmsetup status
cryptsetup-luks status myMapper
losetup /dev/loop1 /dev/mapper/myMapper
#cat ./test_files/2MB_Z.dat > /dev/loop1
#cat ./test_files/2MB_0x00.dat > /dev/loop1
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup-luks luksClose myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the losetup volume file
3. In the dialog shown, enter "password1234567890ABC" as the key, and set any of the options wanted.
4. Click the "OK" button

7.10.6 Example #4: Volume Encrypted Using 256 bit AES-XTS

This example demonstrates use of a LUKS AES 256 volume in XTS mode.

Creating the volume file under Linux:

```
dd if=/dev/zero of=./volumes/vol_aes_xts.vol bs=5M count=1
losetup /dev/loop0 ./volumes/vol_aes_xts.vol
echo password1234567890ABC | cryptsetup-luks -c aes-xts-plain -s 512 luksFormat /dev/loop0
cryptsetup-luks luksDump /dev/loop0
echo password1234567890ABC | cryptsetup-luks luksOpen /dev/loop0 myMapper
dmsetup ls
dmsetup table
dmsetup status
cryptsetup-luks status myMapper
losetup /dev/loop1 /dev/mapper/myMapper
#cat ./test_files/2MB_Z.dat > /dev/loop1
#cat ./test_files/2MB_0x00.dat > /dev/loop1
mkdosfs /dev/loop1
mkdir ./test_mountpoint
mount /dev/loop1 ./test_mountpoint
cp ./test_files/SHORT_TEXT.txt        ./test_mountpoint
cp ./test_files/BINARY_ZEROS.dat      ./test_mountpoint
cp ./test_files/BINARY_ABC_RPTD.dat   ./test_mountpoint
cp ./test_files/BINARY_00_FF_RPTD.dat ./test_mountpoint
umount ./test_mountpoint
losetup -d /dev/loop1
cryptsetup-luks luksClose myMapper
losetup -d /dev/loop0
rm -rf ./test_mountpoint
```

Mounting the volume under FreeOTFE Explorer:

1. Select "Linux | Mount..."
2. Select the losetup volume file
3. In the dialog shown, enter "password1234567890ABC" as the key, and set any of the options wanted.
4. Click the "OK" button

8 Plausible Deniability

8.1 Overview

The subject of "plausible deniability" and OTFE systems is a lot more involved than "do my volume files have any kind of identifying signature?"

"Plausible deniability" in OTFE systems is largely based on the *theory* that you can claim that your volume files are not encrypted data; you don't know what they are - you can't be expected to know every operation that your OS carries out! Perhaps it's some corrupt data that the system recovered at some stage?

This *claim* is only possible with OTFE systems which do not embed any kind of "signature" into their encrypted data (typically an unencrypted critical data area). FreeOTFE volume files have no such signature.

However, this simplistic approach to plausible deniability has many drawbacks, and is *highly unlikely* to offer any significant form of protection; for example:

1. Encrypted data has relatively high entropy - something "corrupt files, recovered automatically by the OS" are not likely to have. (Recovered data is likely to have some form of recognisable structure, or signature, somewhere within it)
2. Having several GB of high-entropy data stored on your HDD, together with an OTFE package, is likely to be viewed as "suspicious" at the very least...

8.2 Legal Issues

(Note: IANAL)

Legally, the presence of large volume files (even without any form of signature) may very well be viewed as grounds for reasonable suspicion to be raised; in which case further action may be taken by an attacker (e.g. arrest, interrogation, beating, torture, and other rubber hose cryptanalysis techniques). Obviously, if reasonable suspicion can be raised, this simplistic approach provides very little in the way of *plausible* deniability!

Legally (in the US at least, and in theory) what it really boils down to is: does the fact that a prosecution cannot prove that the data held is encrypted data, together with a user's denial, produce reasonable doubt as to whether the data is not an OTFE volume or not? Raising reasonable doubt as to what your volumes files really are is the aim of "plausible deniability"; leaving it up to the prosecution to prove, beyond reasonable doubt, they store encrypted data.

In the UK (AIUI), things are slightly different with the "Regulation of Investigatory Powers Act (RIPA)" (see http://www.legislation.hmso.gov.uk/acts/acts2000/20000023.htm), although this is not yet in full force at time of writing (10th October 2004). Depending largely on how the courts may interpret it, a user (as defendant) may eventually find yourself in the position of effectively having to prove that the data found is not an OTFE volume (or any other form of encrypted data).

8.3 Hidden Volumes

More advanced OTFE systems go one step further: support for hidden volumes (as FreeOTFE Explorer does).

Here, you have a "normal" OTFE volume filled with data that you have no objection to disclosing to an attacker. Mounting the volume with a different password causes the OTFE system to read a different part of the "host" volume file; giving access to a separate "hidden" volume.

Here the concept of "plausible deniability" is much stronger; theoretically an attacker is not able to determine (let alone prove) whether or not such a hidden volume is present.

However, the implementation of such "hidden volumes" is not as trivial as it may seem at first.

In order for this approach to be successful, the host volume file must be initialized by writing random data to it. This is required since the host volume file may well have been created by simply writing 0x00's to your HDD in order to generate a large enough file. Any hidden volume stored within such a host volume file may well cause an attacker suspicion as to whether a hidden volume exists. (The hidden volume will appear as a large amount of high-entropy data, stuck in the middle of the volume file; interrupting the neat pattern of 0x00's!)

The "random data" used for this process cannot simply be pseudorandom data; given the size of a typical volume file (even ones as small as a MB), pseudorandom data can potentially be identified as such, and become predictable. In this case, your hidden volume will not appear as high-entropy data stuck in the middle of a series of 0x00 bytes, but as high-entropy data interrupting any pattern formed by the pseudorandom data!

Because truly random data can be difficult to rapidly generate in large quantities using a computer. Pseudorandom data can still be used though: by encrypting it before it is written to the host volume file. In principle, although not as good as a cryptographically secure RNG, this should give the data written to the volume file a suitable degree of entropy.

The easiest way of accomplishing this is, which will work with any OTFE system, is to mount the host volume as per normal and overwrite all of its free space with a single pass of pseudorandom data. The data written to the mounted volume will be encrypted as it is written to the host volume file.

8.4 More Advanced Hidden Volumes

The technique described above for mounting and overwriting a volume before creating a hidden volume on it still isn't enough though.

If you were to be forced to hand over the key to the outer, "host", volume; an attacker could then apply the same analysis - but this time to the mounted (plaintext) version of your host volume. Again, any hidden volume may well stick out in any pattern within the pseudorandom data.

The solution suggested is to encrypt the pseudorandom data before it is used to overwrite the mounted volume's free space; any attacker attempting to identify a hidden volume, even with the key to the outer "host" volume, would not be able to differentiate between your encrypted pseudorandom data, and an encrypted hidden volume.

This all assumes the cypher used is strong enough, of course...

For obvious reasons, all such overwriting must be carried out before the hidden volume is created (doing so afterwards would probably corrupt your hidden volume!)

8.5 In Practice

Needless to say, FreeOTFE Explorer offers full functionalty with overwriting and the encryption of random data used.

To ensure the maximum security for your volumes, the following procedure is suggested after creating each new volume:

1. Mount the new volume
2. Select the new volume just mounted, and then select the "Tools | Overwrite entire drive..." menuitem. (Note: The "Overwrite free space..." option should *not* be selected for this purpose, as this will miss overwriting parts of the volume which the filesystem reserves)
3. Doublecheck that you have selected the right volume, and confirm your actions at the prompt displayed
4. Select "Encrypted data", and a suitable cypher from the dropdown list. Note that the cypher selected does not have to be the same as the one used to secure your volume.
5. Click "OK"
6. Generate some random data to be used as the key for the cypher by "waggling" the mouse pointer over the space shown on the dialog displayed
7. Click "OK"
8. Click "Yes" to confirm you wish to proceed

At this point, the volume will be overwritten. Depending on your hardware, *this process may take some time!* After the overwrite completes, format the drive. The new volume will then be ready for use. To create a hidden volume within the volume just created, dismount the drive and carry out the normal procedure for creating a hidden volume (see the Advanced Topics section for instructions on how to do this).

9 Miscellaneous Notes

- Please, do *read the documentation* (the FAQ section, in particular) *before* emailing questions! The FAQ section in particular may well have the answer you're looking for.
- Both the PC and PDA versions of FreeOTFE, and FreeOTFE Explorer, are fully compatible with one another.
- From the main window, doubleclicking on an item displayed will explore that item. Rightclicking brings up a context menu.
- After creating a new FreeOTFE volume it is recommended that you make a backup of the volume's CDB.
  - In the case of volume files which have their CDB stored as part of the volume file, this can be achieved by selecting "Tools | Critical data block | Backup..."
  - In the case of volume files where the CDB is stored in a separate keyfile, simply make a backup copy of this keyfile.
- A number of FreeOTFE volume properties can be changed via the "Tools | Change volume/keyfile password/details..." menuitem. Note that volumes must be *dismounted* first before they can be modified in this way.
- An option is included to dump out a human readable version of the volume's critical data block/keyfile's contents (select "Tools | Critical data block | Dump to human readable file..."). This option is primarily intended to assist developers, and to future-proof volumes file by giving you access to the actual master encryption/decryption key used by the volume it dumps out. It should be noted that the inclusion of this option does not present a security risk as it requires that the user to enter the volume/keyfile's password immediately before it can operate (obviously, the volume/keyfile's password is needed in order to decrypt the critical data block). If an attacker has your volume/keyfile's password, clearly this option will give no further information away.
- A "Revert timestamps" option is available from the "Options" dialog. If selected, on mounting a volume file its timestamps will be noted. When the volume is subsequently dismounted, these timestamps will be restored. By default, the PC version of FreeOTFE, and FreeOTFE Explorer, have this option switched on, and the PDA version (FreeOTFE4PDA) has it switched off. If you are going to use both the PC and PDA software with the same volume file, syncing between the two platforms, it is recommended that this option is turned off on both your PC and PDA installations, in order for ActiveSync to recognise when your volumes have been modified.
- A password is not needed when backing up a volume's CDB as the backup copy is not stored in plaintext; it is a literal backup copy of a volume's (encrypted) CDB.
- A password is needed when creating a keyfile as this requires that the volume's CDB is decrypted, before being re-encrypted with the keyfile's password and written out to the keyfile.
- After new volumes are created, they will be automatically mounted and formatted. After this, it is highly recommended that you overwrite all the free space on the drive ("Tools | Overwrite free space"...)
- Linux encryption settings files (".les") files are straightforward text files which contain the settings entered.
- Peter Gutmann's "cryptlib" may (optionally) be used as an RNG, provided that it has been installed correctly. This may be downloaded from http://www.cs.auckland.ac.nz/~pgut001/cryptlib/. After installation, the "cryptlib" option will no longer be greyed out on RNG selection dialogs.
- User settings configured via the "View | Options" menu are stored within a configuration file (".ini" file) which is located in the same directory the FreeOTFE Explorer executable is launched from. User options are not stored within the registry, unless configured to store them in it. By storing user settings in a separate file, as opposed to the registry, FreeOTFE Explorer achieves

two things:

1. If FreeOTFE Explorer is stored on removable media (e.g. a USB flash drive, CDROM), your settings can be stored together with FreeOTFE Explorer; there is no need to configure FreeOTFE Explorer every time you use it on a different computer - this would not be possible to do if the registry was used.
2. When user settings are stored in a flat file, as opposed to the registry, security is increased. It is trivial to overwrite a simple file if needed, but removing registry entries completely is another matter.

- Creating an encrypted partition/disk will overwrite whatever data was stored on the partition/disk you select. Be careful!

10 FAQ

The latest version of this FAQ, along with the latest FreeOTFE Explorer user manual, can be found online at the FreeOTFE WWW site

---

10.1 FAQ Contents

10.1.1 General

- Whare are the differences between FreeOTFE and FreeOTFE Explorer?
- How can I help the FreeOTFE project?
- Which of the hash/cypher algorithms should I use?
- Which of the random number generators (RNGs) should I use?
- Is FreeOTFE based on CrossCrypt?
- Is FreeOTFE based on Linux’s "losetup"?
- Right now, FreeOTFE supports losetup volumes; do you have any plans to include support for DriveCrypt, BestCrypt, etc volumes?
- When I mount a FAT/FAT32 formatted Linux volume under FreeOTFE everything works perfectly. When I do the same with myext2/ext3/RiserFS/etc volume, I can’t see my files!
- Why do the Linux examples for LUKS/dm-crypt volumes show "losetup" being used twice?
- FreeOTFE comes with a set of command line decryption utilities! Can’t anyone can just decrypt my data!
- When I mount a volume and then view its properties under FreeOTFE, it states that the hash algorithm used is "n/a" - but I used a hash algorithm!
- FreeOTFE is currently available for free - are you intending to "sell out" later, and start charging for it once enough users have been "hooked" on it?
- FreeOTFE may always be free, but will an "enhanced" version (which is charged for) with extra features be released (perhaps under a different name)?
- What about klonsoft’s "LockDisk" and WinCrypto LLC’s "CryptoDisk"? Aren’t they paid-for packages which are based on FreeOTFE?
- How can I be sure that there are no backdoors in FreeOTFE?
- Do FreeOTFE volumes have any kind of identifying "signature"?
- By examining a FreeOTFE/encrypted Linux volume file, can anyone tell what it is?
- What is "plausible deniability?"
- What do the numbers and letters after a hash name mean?
- What do the numbers and letters after a cypher name mean?
- When creating a new volume file, why do I get a message asking me to ensure I have XX.XX GB free on the relevant drive?
- I tried to create a large volume (> 4GB), and FreeOTFE stopped halfway through with an error - why?
- What is the largest volume that I can create?
- Can I store an encrypted volume on a compressed NTFS drive?
- What hash algorithms does FreeOTFE use?
- What encryption algorithms does FreeOTFE use?

- Which cypher modes does FreeOTFE support?
- Help! I forgot my password! I know it was something like...
- Which is the best encryption algorithm to use?
- How safe is FreeOTFE?
- What happens if my volume file is corrupted or damaged in some way? Will I lose all my data?
- If someone steals my keyfile, will they be able to decrypt by data and read it?
- How do I know FreeOTFE is encrypting my data, and with the encryption algorithm I choose?
- When selecting a cypher to use, why do the some cyphers appear multiple times?
- Why are there duplicated cypher drivers?
- Which of the duplicated drivers should I use?
- Can FreeOTFE generate keyfiles which only allow read only access?
- Can I use the same encrypted volumes on *both* my PC and PDA?
- When creating a new volume, how do I enable the sector IV options?
- Is FreeOTFE vulnerable to "watermarking" attacks?
- Is FreeOTFE vulnerable to "Cold Boot Attacks on Encryption Keys" (aka "DRAM attacks")?
- Does FreeOTFE have any form of password recovery?
- Isn't FreeOTFE's "keyfile" functionality a security risk?
- What happened to the NULL hash and NULL/XOR cypher drivers?
- How do I resize an encrypted volume?
- How do I delete an encrypted volume?
- How do I backup an encrypted volume?
- Can I use *any* filename/file extension for my FreeOTFE volume?
- Does FreeOTFE support LVM2?
- Is it worth running file overwriter ("shredder") programs to securely delete existing data stored on my encrypted drive?
- What is the difference between the main FreeOTFE/FreeOTFE Explorer release and the PortableApps.com version?
- What is the difference between the main FreeOTFE/FreeOTFE Explorer release and the U3 version?
- When dismounting a file based volume, what does FreeOTFE do with the file timestamps?

10.1.2 FreeOTFE Specific (PC)

- When creating a FreeOTFE volume, the wizard shows me which stage of volume creation I am currently on - but it goes haywire, and the number of stages to complete keeps changing!
- Is it possible to dismount my FreeOTFE volumes when I hit a certain "hotkey"?
- Why can't I dismount my volume(s)?
- Why are the drivers written in C, but the GUI in Delphi?!
- Why aren't I prompted to enter a password when creating a Linux volume?
- Can I burn my volumes on a CD (or CDRW, or DVD), and mount them from there?
- Can I use FreeOTFE over a network?
- Why do I get "Unable to connect to the FreeOTFE driver" errors?
- Why do I get prompted to select a driver whenever I attempt to mount some of my FreeOTFE volume?
- Do I need Administrator privileges to use FreeOTFE on my computer?
- Why do I need Administrator rights to install FreeOTFE?
- Why do I need Administrator rights to start "portable mode"?

- Can FreeOTFE run under MS Windows 95/98/Me?
- Can FreeOTFE run under Linux?
- How can I get FreeOTFE to mount my volumes at startup/when I login?
- On the options dialog, what does the "Save above settings to" option do?
- Can I save my settings in the same directory as my FreeOTFE executable?
- Where, and in what order does FreeOTFE search for my settings?
- After associating FreeOTFE with ".vol" files from the options dialog, I doubleclicked my ".vol" volume file, and nothing happened!
- Why do volumes created with the FreeOTFE v2.00 and later have the extension ".vol"?
- What is the difference between the "Overwrite free space..." and "Overwrite entire drive..." options under the "Tools" menu?
- Does FreeOTFE support encrypting data with multiple cyphers (aka "cascaded" cyphers, or "superencryption")
- FreeOTFE supports different languages, but why isn't mine listed?
- How do I translate FreeOTFE into a different language?
- Can I defragment encrypted volumes?
- Can I use FreeOTFE with my USB flash drive?
- Why doesn't FreeOTFE run automatically when I insert my USB drive?
- Can I use FreeOTFE with "MojoPac"?
- Can FreeOTFE be used with RAID arrays?
- Does FreeOTFE try to connect to the internet?
- How do I check FreeOTFE's exit code when passing parameters via the command line?
- Why won't FreeOTFE accept my password when supplied via the command line parameter?
- Partition based volumes
    - Do I have to partition my drive to use FreeOTFE?
    - I want to create a FreeOTFE partition on my unallocated space, but can't see it in the partition display - where is it?
    - When I'm prompted to select a partition, some of the partitions on my USB drive are shown in red (or not at all) - why?
    - Why can't I use encrypted partitions on a USB drive, unless it's the first partition?
    - After creating an encrypted partition, MS Windows reports that partition I used as being type "RAW" and prompts me to format it - why?
    - How do I "hide" an encrypted partition such that MS Windows doesn't allocate it a drive letter?
    - Why does the partition/disk selection display sometimes display less information?
    - I accidentally selected the wrong disk/partition when creating a new volume and now can't see my files! How can I get my data back?
    - Does FreeOTFE offer whole disk encryption?
- Security Token/Smartcard Support (PKCS#11)
    - Do I *have* to use a security token/smartcard with FreeOTFE?
    - What is the difference between PKCS#11, Cryptoki, and "tokens"?
    - Does FreeOTFE encrypt my *entire* encrypted volume using my PKCS#11 token?
    - I've inserted my PKCS#11 (Cryptoki) token, but why is the "PKCS#11 token management..." menuitem disabled?
    - How do I change the password on a volume/keyfile which is secured with a PKCS#11 secret key?
    - Can I use more than one security token with FreeOTFE?
    - Why don't all of my volumes automatically dismount when I remove my security token?

- Windows Vista specific
  - Why do I get "unidentified program wants access to your computer" prompts when using FreeOTFE?
  - Why does FreeOTFE prompt me to enter my Administrator's password?
  - How do I stop the Windows Vista "consent/credential" (UAC) dialog from being displayed?
  - What are the little "shield" icons shown next to some menuitems?
  - I have problems starting any of the drivers under the 64 bit version of Windows Vista/Windows 7 - what's wrong?

10.1.3 FreeOTFE4PDA Specific (PDA)

- I created my volume file using the PDA version of FreeOTFE and can mount it on my PC - but why does it keep asking if I want to format it?
- I created a volume on my PC, and can mount it successfully on my PDA - but can't see any of my files!
- How can I speed FreeOTFE up when mounting my volumes?
- Does the PDA version support Linux volumes?
- Why does FreeOTFE4PDA's version numbering skip from v0.55 to v2.00; what happened to v1.00?
- Why does FreeOTFE4PDA's version numbering skip from v3.76 to v5.00; what happened to v4.00?
- When I use the "open" dialog to select my volume file/keyfile, it doesn't list the file I'm trying to specify - even when I select "All files" - where is it?
- How can I reduce the amount of storage space FreeOTFE4PDA takes up when installed?
- Why do I get the message "Unable to locate local copy of user guide; would you like to see the latest version on the Internet?" when I try to view the user guide by selecting "Help | User guide"?
- When I try to mount a volume, I sometimes the error: "Mount failed; the virtual storage device could not be activated at this time"
- Which PDAs will FreeOTFE4PDA work with?
- What does the "Support WM 5.0 soft keys" option do?
- I don't like the new two-item menus at the bottom of my display - how do I change them back to the older toolbar style menu?
- I don't like the multi-item menubar/toolbar at the bottom of FreeOTFE4PDA's display - how do I get it to use the newer two-item style (softkey) menus instead?
- Can I use my PC volumes with the PDA version?
- I upgraded to the lastest version of FreeOTFE4PDA, can I still mount my old volumes?
- I would like to use hash/cypher *XYZ*, why doesn't it appear as an option?
- How to I enable hash/cypher *XYZ*?

10.1.4 FreeOTFE Explorer Specific

- Does FreeOTFE Explorer support drag and drop with MS Windows Explorer?
- What filesystems does FreeOTFE Explorer support?
- Does FreeOTFE Explorer try to connect to the internet?

10.2 General

Q: Whare are the differences between FreeOTFE and FreeOTFE Explorer?

A: Please see the FreeOTFE v. FreeOTFE Explorer Comparison

Q: How can I help the FreeOTFE project?

A: If you are a native speaker of a language other than English, please take a look at translating FreeOTFE page. FreeOTFE v4.3 introduced support for translating the user interface into different languages, though at present the actual number of translations into other languages is fairly limited.

Alternativly, *FEEDBACK!* If you have any comments or suggestions for how FreeOTFE can be improved - get in touch!

Q: Which of the hash/cypher algorithms should I use?

A: This decision is left up to the user.

Most users can simply accept the default algorithms offered, which provides a fairly high degree of security.

Q: Which of the random number generators (RNGs) should I use?

A: This decision is left up to the user.

Using more than one RNG increases the security offered by FreeOTFE as the combined random data generated will be at least as random as the most random RNG selected. Should one of the RNGs subsequently be found to be weak (i.e. producing data that is not as random as it should be), the random data used will still be as strong as the strongest RNG used.

See the Technical Details: Random Number Generators (RNGs) section for further information.

Q: Is FreeOTFE based on CrossCrypt?

A: The answer to that is an emphatic NO! FreeOTFE and CrossCrypt are two completely separate projects, written by completely different people.

It’s easy to see why users may get the idea that FreeOTFE is based on CrossCrypt; CrossCrypt was released first, and the CrossCrypt’s GUI (CrossCryptGUI) looks practically identical to FreeOTFE’s interface.
The reality is that *CrossCrypt* itself is a command line based OTFE system; it has no GUI. *CrossCryptGUI* was a project I created to provide a GUI to CrossCrypt to improve its ease of use.

In actual fact, far from FreeOTFE looking a lot like CrossCryptGUI, it’s actually the other way around - CrossCryptGUI looks a lot like FreeOTFE! The Delphi GUI to FreeOTFE was already developed before CrossCrypt was released. For the sake of expediency, I dropped the CrossCrypt Delphi

component I wrote into FreeOTFE's GUI, hijacking it to produce CrossCryptGUI; a cannibalized version of the FreeOTFE interface.

The cyphers supplied with the first public release of FreeOTFE (v00.00.01) were the same as those used by CrossCrypt. Originally I had planned to release the first beta of FreeOTFE for compatibility testing with only the NULL, XOR, DES and AES cyphers; these apparently being the most common cyphers used with Linux volumes. After CrossCrypt was released (which uses AES and Twofish) DES was the only cypher in the above list I had not implemented. I decided to switch from DES to Twofish in order that people without Linux could easily use CrossCrypt to verify that FreeOTFE was operating correctly with AES and Twofish volumes (and vice versa; benefiting both systems).

Since its initial release, FreeOTFE has seen significant developments, including support for many more hashes, cyphers, and other options.

Q: Is FreeOTFE based on Linux's "losetup"?

A: No, FreeOTFE is a completely separate project in its own right. It was only after I realised how "simple" Linux encrypted losetup volumes are (they are nothing more than an encrypted partition image), that I added support for them into FreeOTFE.

Having said that the format of losetup volumes are "simple" - have you any idea how many different options, combinations, etc it has?! Each option on its own may be relatively simple, but there are a fair number of them...! (See the relative complexity of the FreeOTFE's Linux mount dialog - you have to tell it everything!)

Q: Right now, FreeOTFE supports losetup volumes; do you have any plans to include support for DriveCrypt, BestCrypt, etc volumes?

A: This is unlikely to happen as there is no standard for OTFE volume files (each system uses its own layout). Since adding support for other OTFE systems is non-trivial, and few OTFE systems have released proper technical documentation into the public domain, it may be awhile before such support is added

---

Q: When I mount a FAT/FAT32 formatted Linux volume under FreeOTFE everything works perfectly. When I do the same with my ext2/ext3/RiserFS/etc volume, I can't see my files!

A: FreeOTFE does one thing: when a volume file is mounted, FreeOTFE presents a new storage device to the operating system.
Like all OTFE systems, it has no comprehension at all of what FAT/FAT32/NTFS, let alone ext2/ext3/etc - this understanding lies well outside the scope of an OTFE system, and is the responsibility of the filesystem drivers installed.

Although MS Windows does come with filesystem drivers for FAT/FAT32/NTFS, it does not (natively) support other filesystems such as ext2.

As a result, in order to read/write to your encrypted Linux volumes under MS Windows, you will need to either:

1. Format the volume under Linux using one of the filesystems MS Windows understands (e.g. FAT), or
2. Install 3rd party software on your MS Windows system, which provides the filesystem (e.g. ext2) that you wish to use

---

Q: Why do the Linux examples for LUKS/dm-crypt volumes show "losetup" being used twice?

A: This actually has *nothing* to do with FreeOTFE(!), but appears to be an oddity with "mkdosfs"/dm-crypt.

Although this section of the documentation shows:

```
losetup /dev/loop1 /dev/mapper/myMappermkdosfs /dev/loop1
```

you should be able to simply use:

```
mkdosfs /dev/mapper/myMapper
```

However, when this section of the documentation was written and tested (under Fedora Core 3, with a v2.6.11.7 kernel installed and using cryptsetup-luks v1.0), this shorter (and more sensible) version resulted in mkdosfs generating the following error:

```
# mkdosfs /dev/mapper/myMappermkdosfs 2.8 (28 Feb 2001)mkdosfs: unable to get drive geometry for '/dev/mapper/myMapper'
```

YMMV, though you may well find that formatting the volume with a different filesystem will remove the "double loop" issue. (Please note though, that if you are intending to encrypted volumes which don't use FAT/NTFS under MS Windows, you will need a suitable filesystem driver)

Q: FreeOTFE comes with a set of command line decryption utilities! Can't anyone just decrypt my data?
A: The decryption software included with FreeOTFE is completely useless without the password used to encrypt your data. And anyone with that information can decrypt your data anyway!

The command line decryption utilities are not some form of "password cracking" tool - far from it; they actually act to increase your security by allowing you to verify that encryption is actually taking place.

Q: When I mount a volume and then view its properties under FreeOTFE, it states that the hash algorithm used is "n/a" - but I used a hash algorithm!
A: The hash algorithm shown is the one used to generate sector IVs. If the sector IV generation method used does not require the use of a hash algorithm (see the "Sector IVs" item on this dialog), "n/a" will be displayed for the hash algorithm.

This is separate from any hash algorithm used to process your password, which in the case of FreeOTFE volumes can be seen in the output file of a CDB dump (select "Tools | Critical data block | Dump to human readable file..."), or in the case of Linux volumes, is specified at time of mounting.

Q: FreeOTFE is currently available for free - are you intending to "sell out" later, and start charging for it once enough users have been "hooked" on it?

A: NO! ABSOLUTELY NOT! FreeOTFE is free, and will always be free. As much as anything else, it would look a little silly if people had to pay for "FreeOTFE"! ;)

Seriously though, I have no intention in turning FreeOTFE into a commercial product.

The nearest that I may do is request donations. This would, of course, be fully voluntary.

Q: FreeOTFE may always be free, but will an "enhanced" version (which is charged for) with extra features be released (perhaps under a different name)?

A: Personally, this sounds a lot like the "selling out" idea above - if such a "paid for" version was to be released, FreeOTFE development may become at risk of stalling, ceasing completely, or omitting particularly useful features. This would have practically the same effect as making FreeOTFE a paid-for commercial system.

Q: What about klonsoft's "LockDisk" and WinCrypto LLC's "CryptoDisk"? Aren't they paid-for packages which are based on FreeOTFE?

A: Both "LockDisk" and "CryptoDisk" are unlicensed (and unlicensable) commercial rip-offs of FreeOTFE. They are based on FreeOTFE's source code (and only a beta version at that in the case of "LockDisk") and, because they are closed-source, are in direct violation of FreeOTFE's licence.

I have nothing to do with either "LockDisk" or "CryptoDisk", nor any involvement in their creation.

Personally, I would strongly recommend against using these products:

- They have less functionality than FreeOTFE
- They're closed source; there's no way of knowing how secure it is, or what it does
- It is not possible to (legally) obtain a licence for these products
- In the case of "LockDisk" the so-called "free" version is *severely* crippled (only permitting 35MB volumes)
- In the case of "LockDisk", it's based on a pretty old and now obsolete (v0.59 BETA) version of FreeOTFE
- And for all this, you have to pay for them?!!

I could list another few dozen reasons for not using these products, but I think you get the picture - FreeOTFE is simply better!

Q: How can I be sure that there are no backdoors in FreeOTFE?

A: Review the source code to your satisfaction, and build your own (see section Building FreeOTFE)

This is strongly recommended, and the best way of ensuring that the software is not compromised.

However, this is not always practical (many people are not familiar with how to read source code, or lack the required tools to build their own). In which case, if you trust the author, and the system on which the release was built on, then you may prefer to simply check the SHA-1 and PGP signatures associated with the binary release.

Q: Do FreeOTFE volumes have any kind of identifying "signature"?

A: No!

Please see the FAQ: By examining a FreeOTFE/encrypted Linux volume file, can anyone tell what it is? for further information.

Q: By examining a FreeOTFE/encrypted Linux volume file, can anyone tell what it is?

A: Neither FreeOTFE nor encrypted Linux volumes have any kind of "signature" that would allow an attacker to identify them for what they are.

In particular, the "critical data block" in every FreeOTFE volumes is encrypted, and as such it is not possible to identify it for what it is

---

Q: What is "plausible deniability?"

A: See the documentation section on "Plausible Deniability" for details.

Q: What to the numbers and letters after a hash name mean?
A: When required to choose which hash you wish to use, FreeOTFE will present you with a list of all hashes that are provided by the FreeOTFE drivers installed. These lists will display hash names in the format:

<hash name> (<hash length>/<block size>)

Note: The hash length and block sizes shown are in bits, not bytes.

For example:

SHA-512 (512/1024)

This indicates that the hash used is SHA-512, which generates 512 bit hash values, and processes data in 1024 bit blocks.

If the hash length shown is zero, then the hash generates no output.
If the hash length shown is "-1", then the length of the hash values returned can vary.

If the block size is "-1", then the hash processes data using a variable block size.

Typically, when presented with a selection of different hashes to choose from, you will see a "?" or "..." button next to the list; clicking this button will display full details on the driver.

Q: What to the numbers and letters after a cypher name mean?
A: When required to choose which cypher you wish to use, FreeOTFE will present you with a list of all cyphers that are provided by the FreeOTFE drivers installed. These lists will display cypher names in the format:

<cypher name> ([<mode>;] <key size>/<block size>)

Note: The key and block sizes shown are in bits, not bytes.

For example:

AES (XTS; 256/128)

This indicates that the cypher is AES, operating in XTS mode with a key size of 256 bits and a block size of 128 bits.

If the key size shown is zero, then the cypher does need take a key (password) to carry out encryption (e.g. the "Null" test cypher).
If the key size shown is "-1", then the cypher can accept keys of arbitrary size.

If the block size is "-1", then the cypher encrypts/decrypts arbitrary block size.

Typically, when presented with a selection of different cyphers to choose from, you will see a "?" or "..." button next to the list; clicking this button will display full details on the driver.

Q: When creating a new volume file, why do I get a message asking me to ensure I have XX.XX GB free on the relevant drive?

A: If you get an error stating that:

Unable to create volume file; please ensure you have XX.XX GB free on the relevant drive

during volume creation, this is probably because the drive you are trying to create the volume on is formatted as FAT/FAT32 - both of which have a file size limit of 4GB.

Please see the FAQ "I tried to create a large volume (> 4GB), and FreeOTFE stopped halfway through with an error - why?"

Q: I tried to create a large volume (> 4GB), and FreeOTFE stopped halfway through with an error - why?
A: The most probable cause for this is that you were creating a volume file on a FAT/FAT32 filesystem, however FAT/FAT32 filesystems cannot support files larger than (4 GB - 1 byte).

See the FAQ: What is the largest volume that I can create? for further information and how to resolve this.

Q: What is the largest volume that I can create?
A: On a PC, FreeOTFE has a theoretical maximum volume size of 2^64 bytes (16777216 TB; 17179869184 GB). For fairly obvious reasons, I have not had the opportunity to test a volume this size!

In practice however, although partition based volumes may be able to realise volumes as large as this, file-based volumes may find that limitations with the filesystem that the volume file is to be stored upon may prevent this limit from being reached.

For example, a FAT32 drive cannot store a volume file which is 4GB or larger. In practical terms, this means that the largest volume you can create on a FAT32 filesystem is 3999 MB. An NTFS formatted drive *can* store volume files *much* larger; in excess of FAT32's 4GB limit, and up to FreeOTFE's maximum size stated above.

On a PDA, the largest volume supported is 2^32 (4GB). This is due to limitations with Windows Mobile.

Q: Help! I forgot my password! I know it was something like...
A: Oops. That was silly of you, wasn't it?

If you've secured your volume with something like AES, then you can pretty much kiss goodbye to your data.

If you know what most of your password is though, then you could certainly write an application which would carry out a brute force attack on your volume, assuming those known characters. How long this would take to run would depend on the cypher used, the strength of your password, and how much you remember of it.

Note: This is not a security risk; that last comment equally applies to pretty much any OTFE system which has been implemented correctly.

Q: Can I store an encrypted volume on a compressed NTFS drive?

A: Yes, though there is nothing to be gained from compressing encrypted data, as it is unlikely to compress by any significant amount (if at all)

Q: What hash algorithms does FreeOTFE use?

A: A full list of the hash algorithms used by FreeOTFE can be found on the introduction page

---

Q: What encryption algorithms does FreeOTFE use?

A: A full list of the cyphers and cypher modes used by FreeOTFE can be found on the introduction page

---

Q: Which cypher modes does FreeOTFE support?

A: With the exception of the NULL and XOR cyphers, FreeOTFE offers CBC, LRW and XTS modes, and has the flexibility for other modes to be easily added by simply changing drivers.

A full list of the cyphers and cypher modes used by FreeOTFE can be found on the introduction page

---

Q: Which is the best encryption algorithm to use?

A: *That* is a difficult question to answer!

The best advice that can be given here is to research the cyphers available, and make your own decision based on your particular security requirements.

FreeOTFE defaults to using the AES-256 cypher in XTS mode together with SHA-512 for hashing. This should prove more than enough for the overwhelming majority of users.

Q: How safe is FreeOTFE?

A: FreeOTFE is about as pretty much just as safe as writing directly data to your hard drive, without FreeOTFE encrypting it (see also the FAQ: "What happens if my volume file is corrupted or damaged in some way? Will I lose all my data?")

If you forget your password however, then by definition you will not be able to recover your data (see also the FAQ: "Help! I forgot my password! I know it was something like...")

Q: What happens if my volume file is corrupted or damaged in some way? Will I lose all my data?

A: As with pretty much all OTFE systems, if you were to corrupt a FreeOTFE volume is some way, the damage your data would receive would be about the same as if you had stored it directly on your hard drive, without FreeOTFE encrypting it.

For example: If you mount a FreeOTFE volume file and then write a byte of data, at random, to somewhere on that mounted drive, the effect would be exactly the same as if you had randomly written the same byte to a real hard drive.

On the other hand, if you were to write a byte to data to a random location within an umounted FreeOTFE volume, then the amount of damage caused would dependant on where that byte was written:

1. If the volume file was created with a critical data block (CDB) at the start of it, and the byte was written to the first 512 bytes of the volume file (where the CDB is located), then the volume would be unmountable, unless you had made a backup of this area of your volume, or created a keyfile - in which case, you could restore from your backup/mount from your keyfile, and continue as if nothing had happened.
2. If the volume file was created without a critical data block, or the byte was written to any other part of your volume file, then the sector that corresponded to the location that the byte was written to would be corrupted from approximately the point the byte was written, to the end of that sector; a maximum of 512 bytes.

To protect against (1), FreeOTFE included functionality to backup a volume’s CDB (see "Tools | Critical data block... | Backup..."), and to create keyfiles (see "Tools | Create keyfile...")
Should case (2) occur, the damage to your volume would be minimal (up to a maximum of 512 bytes), and restricted to the sector that was corrupted.

Q: If someone steals my keyfile, will they be able to decrypt my data and read it?

A: No, not unless they have the keyfile’s password as well.

Keyfiles are encrypted. Without the password used to encrypt it, a keyfile is pretty much just a useless block of random data.

Q: How do I know FreeOTFE is encrypting my data, and with the encryption algorithm I choose?

A: To verify that encryption/decryption is taking place for Linux volumes, create an encrypted volume using Linux; then mount it using FreeOTFE.

The encrypted Linux volume will be fully readable (and writable) using FreeOTFE - confirming that the same encryption is taking place under FreeOTFE as Linux.

For FreeOTFE volumes, the critical data block can be dumped out (see "Tools | Dump to human readable file..." menu), and the master encryption key used to mount the same volume under Linux (offsetting for the CDB) - again proving that encryption is taking place.

WARNING: *Contrary to popular belief*, a user interface which accepts and processes encryption test vectors does *not prove anything!* It is a trivial task to take a secure cypher, and use it to process test vectors provided by the user, while actually using a very weak and insecure cypher to carry out encryption/decryption on the data being stored!

Q: When selecting a cypher to use, why do the same cyphers appear multiple times?

A: This is because you have more than one version of a particular cypher driver installed. See also: Why are there duplicated cypher drivers?

Q: Why are there duplicated cypher drivers?

A: The "duplicated" drivers implement the same algorithms, but are built from different crypt libraries. For example, there are three Twofish drivers; one based on the Hi/fn and Counterpane Systems Twofish implementation, another which uses the libtomcrypt implementation, and a third which relies on the Gladman implementation.

They redundant drivers are primarily intended to allow verification of the implementations and increase confidence that they’re actually doing what it’s supposed to do.

These duplicated drivers do exactly the same thing. It is recommended that if you wish to use a cypher which has multiple supplied drivers, that you uninstall one of them. (See also: Which of the duplicated drivers should I use?)

Q: Which of the duplicated drivers should I use?

A: It doesn’t particularly matter too much; they both do exactly the same thing, but are based on different implementations.

Simply choose one and uninstall the other.

Q: Can FreeOTFE generate keyfiles which only allow read only access?

A: Not at present, though if I receive enough requests for it, I may add this functionality.

Until then, it should be borne in mind that anyone with a "read only" keyfile has, pretty much by definition, a copy of your master key and so has the potential to modify their "read only" keyfile, turning it into a "read-write" keyfile.

i.e. It is debatable how much use this functionality has; certainly it should not be relied upon to prevent users from gaining write access to your volume files.

Q: Can I use the same encrypted volumes on *both* my PC and PDA?

A: Yes - you can! Both the PC and PDA versions of FreeOTFE are *fully* compatible with each other.

However, please create your volume using the PC version of FreeOTFE. Volumes created using the PDA version will include additional partition information which will not be understood by your PC.

Make sure that before attempting to mount your volume using the PDA version, you have enabled the relevant hash/cypher drivers used in securing the volume. By default, FreeOTFE4PDA only has the SHA-*xxx* and AES algorithms enabled. See Advanced Topics, "Enabling/Disabling Hash/Cypher Algorithms" section, for how to enable/disable other hash/cypher algorithms.

Q: When creating a new volume, how do I enable the sector IV options?

A: Sector IVs are only used with cyphers using CBC mode; to enable the sector IV options, select an encryption algorithm which operates in CBC mode.

If you select a cypher which uses either LRW or XTS, the IV options are automatically disabled as these algorithms don’t use them.

Q: Is FreeOTFE vulnerable to "watermarking" attacks?

A: FreeOTFE volumes are *not* vulnerable to watermarking attacks, as long as they are created with a cypher using:

- XTS mode
- LRW mode
- CBC mode with ESSIV

(see the "Create new volume" wizard, encryption settings step).

By default, FreeOTFE creates volumes using XTS mode. Users would have to *deliberatly* create their volumes using CBC mode with predictable IVs in order to be vulnerable to this type of attack.

Q: Is FreeOTFE vulnerable to "Cold Boot Attacks on Encryption Keys" (aka "DRAM attacks")?

A: No, it isn't - assuming common sense is used.

Description

A "cold boot attack" involves rebooting a computer which has been handling sensitive information, and dumping contents of its memory out to a disk in order to try to examine information stored in memory immediately prior to rebooting. This form of attack is detailed at http://citp.princeton.edu/memory/

This attack is *nothing new*, and has been well known for a *long* time; despite the disproportionate amount of attention it's now getting.

Solution

If you mount an OTFE volume, and simply walk away from your computer, the encryption keys used to secure your volume will be held in your computer's physical memory (obviously). If someone reboots your computer at that point, there is a risk they could successfully recover your encryption key.

However, it is not generally recommended that you simply *walk away from your computer while you have volumes mounted* - if anyone can come along and attempt to launch the above attack, THEY CAN SIMPLY READ THE CONTENTS OF YOUR ENCRYPTED VOLUME DIRECTLY ANYWAY!

If you dismount your volumes after using them, the FreeOTFE driver overwrites all sensitive data (key information, etc) that it holds before releasing it - which should prevent the above attack.

If you suddenly press your computer's power off button or reset it (i.e. using the physical "power off" button on the front of its case) *while a volume is mounted*, then an attacker could theoretically dump out your encryption keys using this attack. Please note that:

1. *All* encryption systems are susceptible to this attack, since they have to store encryption keys in memory in order to use them
2. *Regardless of whether you use any form of disk encryption or not*, it is not recommended that you do power off/reset your computer without first shutting down cleanly via the "Start -> Turn off computer"!

To prevent this attack in the situation described above, ensure that the computer remains powered off for several minutes after it is turned off in order for the contents of RAM to effectively "bleed away"

Summary

In summary, to completely remove the threat of this attack against your encryption keys:

1. Dismount volumes after you have used them
2. If you must power off your computer *while one or more volumes are mounted*: prevent anyone from powering it back on and dumping it's memory out for at least the first few minutes after it was powered off (or the first 15-20 minutes if they open up the case and spray coolant on the memory chips)

In short - just use common sense.

Notes

It should be noted that this attack is *not limited in any way to disk encryption systems*. The focus on these systems by the authors of the above paper is a red herring. Essentially the attack consists of attempting to take a snapshot of the PC's memory at the time it was reset, which can then be picked over at leisure. *Any* encryption system can be attacked in this way.

Furthermore, because this attack may allow whatever was in the computer's memory at the point it was rebooted to be recovered, it should also be noted that any information that applications had in memory at the time the computer is reset (e.g. a document open in MS Word, or image being displayed on the screen) may potentially be recovered. Disk encryption systems encrypt data stored on disks - not in RAM.

Q: Does FreeOTFE have any form of password recovery?

A: Yes; FreeOTFE keyfiles can be used to provide a form of password recovery; see the Getting Started Guide

---

Q: Isn't FreeOTFE's "keyfile" functionality a security risk?

A: No. In order to create a keyfile, both the volume and the volume's password (or an existing keyfile, and that keyfiles password) are required.

If an attacker already has this information, your security has already been compromised anyway.

Q: What happened to the NULL hash and NULL/XOR cypher drivers?

A: To improve performance, these drivers have been moved into a "weak drivers" directory in the PDA release. Really, you shouldn't be using these drivers at all; they are of little use from a security perspective, and are only really included to allow testing. They're still included with the release though, if you *really* need them...

Q: How do I resize an encrypted volume?

A: To change the size of an encrypted volume:

1. Mount your existing volume
2. Create a new FreeOTFE volume of the size required
3. Mount the new volume (overwriting and formatting it if needed)
4. Copy all data from the old volume to the new one
5. Dismount both volumes
6. Delete the old volume

Obviously, this procedure requires enough storage space to hold both the old and newly created volumes.

It should be noted that, although a number of *other* disk encryption systems *claim* to offer volume resizing functionality, they typically either carrying out the procedure above "behind the scenes" (often failing completely if insufficient storage is available to hold the new volume), or by storing the volume in a "sparse" files - which can lead to security leaks.

Q: How do I delete an encrypted volume?

A: If your volume is stored within a file, simply dismount the volume if already mounted, and delete the file.

IMPORTANT: Before deleting a volume file, make sure that you mount it first and copy any information stored in it to somewhere safe! Once deleted, you will lose access to your encrypted volume, and anything it contains!

Q: How do I backup an encrypted volume?

A: How you backup an encrypted volume depends on whether it is a file or partition based volume. In both cases however, volumes should be *dismounted* before being backed up.

For file based volumes

A file based volume is a file just like any other (albeit a fairly big one); simply let your backup software backup the volume as it chooses, and your data should be safe.

This will work regardless of what backup software you use, though you may wish to turn off FreeOTFE's timestamp reverting functionality in order for your backup software to identify when volumes have been changed. (See "View | Options..." dialog, "General" tab, "Revert volume timestamps on dismount")

For disk/partition based volumes

Whether you can backup disk/partition based volumes depends on the backup software being used. If your backup software takes a *literal* backup image of a disk/partition, then it should successfully backup FreeOTFE volumes (even if the backup copy is compressed). However, not all backup systems do this, and instead try to be "smart" about what they store to backup - and fail to backup everything they need to.

(This issue is true for all disk encryption systems, not just FreeOTFE)

For example, with Paragon Drive Backup, if you create an encrypted volume using an *entire disk* (i.e. without creating a partition on the disk, and encrypting that partition), Paragon Drive doesn't appear to think there's anything worth backing up (i.e. it doesn't see any partitions *to* backup) and therefore backs up practically nothing. As a result, it will *not* back up your volume correctly.

However! If you create an encrypted volume on a *partition* (even one filling the entire drive), and back that partition up, Paragon Drive Backup does what it *should* do - generates a compressed backup copy of the *entire* partition, which can then be restored back later.

No matter *what* you're backing up, when you setup a backup system for the first time, it is *strongly recommended* that you go through the restore process at least *once* before "setting it and forgetting it". The *absolute worst* time for learning how your software's restore function works is when you actually need it (e.g. after a disk failure, and you want to get your data back)

This advice applies to **ALL** backups, and not just backups of FreeOTFE volumes.

By doing a "dry run", you can have confidence in both your backups, and in your ability to use them should you need to.

Q: Can I use *any* filename/file extension for my FreeOTFE volume?

A: Yes!

Filenames and file extensions have no special meaning to FreeOTFE, which means any filename can be used.

Q: Does FreeOTFE support LVM2?

A: Yes - it certainly can!

FreeOTFE fully supports Linux LVM2 volumes, provided that you have a suitable Windows driver which allows access to LVM2 volumes, this will allow FreeOTFE to carry out disk encryption either above or below the LVM management system (i.e. on physical or logical volumes)

(It should be noted however that LVM2 is *not* a disk encryption issue!)

Q: Is it worth running file overwriter ("shredder") programs to securely delete existing data stored on my encrypted drive?

A: For most users, no - it would only have the effect of replacing encrypted files with encrypted garbage; neither is particularly useful to an attacker.

However, if you have concerns of an attacker being able to gain your password (and other details required to decrypt your encrypted volume), it may still be wise to overwrite data before its deletion. This way, should an attacker be able to decrypt your volume(s), they will not be able to use data recovery tools to retrieve sensitive data.

Q: What is the difference between the main FreeOTFE/FreeOTFE Explorer release and the PortableApps.com version?

A: The PortableApps.com version is identical to the main FreeOTFE/FreeOTFE Explorer release, but includes an additional:

- "Launcher" executable, which simply starts FreeOTFE.exe/FreeOTFEExplorer.exe
- Directory structure required to integrate it into the PortableApps.com menu software.
- Configuration files required to integrate it into the PortableApps.com menu software.

Further, the installer has been created using the PortableApps.com installer-creator software instead of the standard FreeOTFE NSIS installer, and the translation source files (".po" files, which aren't needed to use the software) have been removed.

Q: What is the difference between the main FreeOTFE/FreeOTFE Explorer release and the U3 version?

A: The U3 version is identical to the main FreeOTFE/FreeOTFE Explorer release with the exception that a slightly different directory structure is used to support the U3 platform, and the translation source files (".po" files, which aren't needed to use the software) have been removed.

The ".u3p" file is simply a ZIP archive which has been renamed; it may be renamed to have a ".zip" file extension and uncompressed to verify its contents.

Q: When dismounting a file based volume, what does FreeOTFE do with the file timestamps?

A: By default, when mounting file based volumes, FreeOTFE stores the volume file's timestamps, and resets them back again after dismounting. This is carried out for security reasons (see section on plausible deniability).

This functionality can be turned off if needed (e.g. to assist backup processes; see FAQ "How do I backup an encrypted volume?") by turning off the "Revert volume timestamps on dismount" option on the Options dialog ("View | Options").

10.3 FreeOTFE Specific (PC)

Q: When creating a FreeOTFE volume, the wizard shows me which stage of volume creation I am currently on - but it goes haywire, and the number of stages to complete keeps changing!
A: The number of different stages to creating a new FreeOTFE volume varies, depending on what options you choose - for example, if you elect to the mouse movement to generate random data, then you will have to complete an extra step to actually generate this random data; if you switch to using the Microsoft CryptoAPI for generating random data, you can skip that step, as it is done for you automatically.

Q: Is it possible to dismount my FreeOTFE volumes when I hit a certain "hotkey"?

A: Yes; see under "View | Options..." - the "Hotkeys" tab

Q: Why can’t I dismount my volume(s)?

A: The most common reason for this is because FreeOTFE cannot gain an exclusive lock on the associated drive. This is normally caused by one or more files being open on the encrypted volume.

"Normal" (non administrator) users may also have problems dismounting drives (see the TODO list this documentation)

If a volume cannot be dismounted "normally", you will be prompted if you want to forcefully dismount it; it is only recommended that volumes are dismounted in this way if all open files and documents are closed.

Q: Why are the drivers written in C, but the PC versions GUI in Delphi?!

A: Good question. The drivers are written in C as the DDK pretty much requires it. The PC GUI is in Delphi as this was the easiest for me to implement.

The PDA version of the GUI was written in C; this may be ported to the PC platform at a later date

---

Q: Why aren’t I prompted to enter a password when creating a Linux volume?

A: This is covered in the documentation; see section relating to creating Linux volumes.

In a nutshell, creating a Linux volume only requires a file to be created of the appropriate size. It is when the volume is subsequently mounted that a password is required; the same process as when creating an encrypted Linux volume under Linux.

Q: Can I burn my volumes on a CD (or CDRW, or DVD), and mount them from there?

A: Yes; at the end of the day, volume files are just plain straight (albeit very large) files. Just ensure that when you mount them, you mount them as read only volumes, (for obvious reasons - even with CDRWs).

It is recommended that volumes which are to be written to CD are formatted using either the FAT or FAT32 filesystem. NTFS volumes will work (under Windows XP), though AFAIR Windows 2000 is unable to mount NTFS volumes read only (meaning the volume must be copied back to your HDD, the file set to read/write, and then mounted).

Q: Can I use FreeOTFE over a network?

A: Yes. By installing FreeOTFE on the computers you wish to access your data from, you can mount a volume file located on a networked server.

When mounting over a network, simply specify the UNC path (e.g. \\servername\sharename\path\volumefilename) to the volume file begin mounted.

When a volume is mounted over a network in this way, all data read/written to that volume will be sent over the network in encrypted form.

If you wish to mount a networked volume file by more than one computer at the same time, you may do so provided that they all mount the volume read only. If any computer has a volume file mounted as read/write, you should dismount all other computers (even if they were accessing the volume as read only), and ensure no other computer mounts the volume until the computer mounted as read/write has dismounted.

Q: Why do I get "Unable to connect to the FreeOTFE driver" errors?

A: This message indicates that you have either not installed the main FreeOTFE driver ("FreeOTFE.sys"), or you have not started it yet.

It is normal to see this message in the following circumstances:

1. The first time you run FreeOTFE, when no drivers have been installed
2. When exiting the driver installation dialog, if the main FreeOTFE driver hasn't been both installed and started.
3. When starting FreeOTFE after installing the main FreeOTFE driver, if the driver has not been started (e.g. you rebooted, and the driver was set for manual start, as opposed to at system startup)
4. When stopping all portable mode drivers, where the main FreeOTFE driver was started in portable mode.
5. When exiting FreeOTFE and stopping all portable mode drivers, where the main FreeOTFE driver was started in portable mode.

To eliminate this error message, ensure that that the main FreeOTFE driver is installed and started.

To prevent this error message from being displayed when FreeOTFE is run after rebooting, set the main FreeOTFE driver to start at system startup.

The status of all installed drivers can be checked by selecting "File | Drivers..."

Q: Why do I get prompted to select a driver whenever I attempt to mount some of my FreeOTFE volume?

A: If your volume looks as though it can be decrypted by using more than one cypher/hash driver combination, you will be prompted to select which combination you wish to use.

This happens, for example, if you used Twofish or AES to encrypt your data as FreeOTFE comes supplied with a choice of drivers for these cyphers (see also: Which of the duplicated drivers should I use?)

To prevent the prompt appearing, please uninstall one of the offending drivers.

Q: Do I need Administrator privileges to use FreeOTFE on my computer?

A: No - Although Administrator privileges are needed to install the FreeOTFE drivers, or start/stop portable mode.

To allow "standard" (non Administrator) users to use FreeOTFE, please install the FreeOTFE drivers by following the instructions in the Installation and Upgrading section. After which, any user will be free to use FreeOTFE (e.g. to create, mount, dismount and use encrypted volumes)

To access an encrypted volume on a PC which *doesn't* have FreeOTFE installed, and on which you *don't* have Administrator privileges, please use FreeOTFE Explorer.

Q: Why do I need Administrator rights to install FreeOTFE?

A: This is probably the most common FAQ with respect to OTFE systems.

In order for most (if not all) OTFE systems to operate, they require the use of "kernel mode drivers" to carry out drive emulation.

A "kernel mode driver" is special piece of software which operates at a very low-level within your computer's operating system. As such, it can do pretty much anything to your system - including carrying out privileged actions that normal users are not allowed to do (e.g. formatting your HDD). Because of this, MS Windows only allows users with Administrator rights to install such drivers.

NOTE: Administrator rights are not required in order to use FreeOTFE once installed.

To access an encrypted volume on a PC which *doesn't* have FreeOTFE installed, and on which you *don't* have Administrator privileges, please use FreeOTFE Explorer.

Q: Why do I need Administrator rights to start "portable mode"?

A: Administrator rights are required to start "portable mode" starting portable mode implicitly registers the FreeOTFE drivers on the computer it's running on. When portable mode is stopped, they are unregistered.

Administrator rights are required for this operation, for the same reasons as given for the answer to "Why do I need Administrator rights to install FreeOTFE?"

To access an encrypted volume on a PC which *doesn't* have FreeOTFE installed, and on which you *don't* have Administrator privileges, please use FreeOTFE Explorer.

Q: Can FreeOTFE run under MS Windows 95/98/Me?

A: No - and there are currently no plans to port FreeOTFE to Windows 9x based systems due to the different driver model used.

Q: Can FreeOTFE run under Linux?

A: No - although FreeOTFE can read, write and create volumes which can be used under Linux.

Q: How can I get FreeOTFE to mount my volumes at startup/when I login?

A: By creating a shortcut with suitable command line parameters in your "Startup" directory (click the MS Windows "Start" button, then go to "Programs | Startup"), FreeOTFE can mount volume files after your system starts up/you login.

See the Command Line Interface section for full details of FreeOTFE's command line options.

Q: On the options dialog, what does the "Save above settings to" option do?

A: This allows you to change where your FreeOTFE settings are stored; in your user profile (only accessible to you), or with the FreeOTFE executable (which is useful if you want to take FreeOTFE with you; on a USB drive, for example).

You may also choose to not save your settings; in which case, the next time you start FreeOTFE, you will begin again with the default options.

Q: Can I save my settings in the same directory as my FreeOTFE executable?

A: Yes, you can - and this makes FreeOTFE more portable, and easier to use, if you want to take it with you on (for example) a USB drive.

There is only one exception though; if you are using Windows Vista, and have User Account Control (UAC) switched on, you will not be allowed to store your settings with the FreeOTFE executable *if it is stored under your "Program Files" directory*. This is due to one of the limitations imposed by Windows Vista's security system; though you are still free to store FreeOTFE's settings in your user profile.

Q: Where, and in what order does FreeOTFE search for my settings?

A: If you have chosen to save your settings, FreeOTFE will store them un a "FreeOTFE.ini" file stored on your computer at your chosen location

When it starts up, FreeOTFE will attempt to locate this file and read in your settings, by first checking for it in the same directory the executable (FreeOTFE.exe) was located in. If a settings file cannot be found in this location, it will try and look for the same file in your user's profile. If a settings file still cannot be found, FreeOTFE will fallback to using configured default values for all settings.

Q: After associating FreeOTFE with ".vol" files from the options dialog, I doubleclicked my ".vol" volume file, and nothing happened!

A: The FreeOTFE drivers must be running in order for you to mount a volume by doubleclicking on it. Please either install the FreeOTFE drivers (see the installation section), or start FreeOTFE's portable mode (see portable mode section).

Q: Why do volumes created with the FreeOTFE v2.00 and later have the extension ".vol"?

A: This is purely to maintain consistency with the PDA version (see other FAQ for an explanation as to why the PDA version uses filename extensions). FreeOTFE gives you complete freedom over what you name your volume files.

Q: What is the difference between the "Overwrite free space..." and "Overwrite entire drive..." options under the "Tools" menu?

A: These options are largely self-explanatory.

The "Overwrite free space.." option will simply overwrite all *unused* storage on the selected volume.

The "Overwrite entire drive.." option is more destructive - it will overwrite *all* storage on the selected volume - including overwriting (destroying) any data that may have been present on it.

Because the latter option is more destructive, it may only be used when a single mounted volume has been selected within the FreeOTFE user interface.

Q: Does FreeOTFE support encrypting data with multiple cyphers (aka "cascaded" cyphers, or "superencryption")

A: Yes! FreeOTFE allows volumes to be nested one inside another, with *complete* flexibility as to which encryption options are used with each volume.

This means that you can (for example) have:

- An AES XTS (with SHA-512) encrypted volume, stored within
- A Blowfish LRW (using Tiger) encrypted volume, stored within
- A Serpent CBC-ESSIV (using RIPEMD-320)encrypted volume

In this example, any data stored within the "innermost" AES encrypted volume will be actually be triple-encrypted with AES, Blowfish and Serpent before written to disk.

Obviously, there is a performance impact in encrypting data more than once - as there would be in any system which encrypts data multiple times.

It’s debatable how much this will increases security by, though in principle the "innermost" volume, in which sensitve files are stored, will be secured at least as strongly as the strongest cypher used. Should any of the cyphers be found to be weak at a later date, this will still hold true.

Note: Volumes nested in this manner must be dismounted in the *reverse* order to which they were mounted.

Q: FreeOTFE supports different languages, but why isn’t mine listed?

A: Please see FreeOTFE’s translations page for up-to-date information on language translations.

Q: How do I translate FreeOTFE into a different language?

A: Please see FreeOTFE’s translations page for up-to-date information on language translations.

Q: Can I defragment encrypted volumes?

A: Yes! There are two things that you may wish to defragment:

1. (File based volumes only) The drive on which the volume file is stored (i.e. defragmenting a volume file)

Once dismounted, a volume file can be treated *just like any other file*. Volume files can be defragmented by then running *any* defragmentation tool on the drive it's stored on.

2. The filesystem stored within the encrypted volume (i.e. defragmenting the encrypted files stored within the volume)

   By mounting a volume, you can defragment the encrypted data stored within it. Again, you can use any tool for this, with the exception of:
   - Raxco's PerfectDisk 2008
   - Diskeeper Corporation's "Diskeeper"
   - The defragmentation tool which comes bundled with Windows (which is a simply a stripped down version of Diskeeper)

   The above systems have limitations which prevent them from "seeing" mounted volumes, all other tools will work as normal. Examples of defragmentation tools which work with FreeOTFE volumes include:
   - O&O Defrag
   - Defraggler
   - Auslogics Disk Defrag
   - IObit Smart Defrag
   - JkDefrag
   - Ultra Defragmenter
   - ...and the vast majority of other defragmentation tools

Q: Can I use FreeOTFE with my USB flash drive?

A: Yes! FreeOTFE has been designed to be portable; see the section on Portable Mode for details on which files to copy onto your USB drive. Alternativly, insert your USB drive and select the "Tools | Copy FreeOTFE to USB drive..." menuitem to automatically copy FreeOTFE to your USB drive.

You can then use FreeOTFE on any PC - even if it doesn't have FreeOTFE installed.

Q: Why doesn't FreeOTFE run automatically when I insert my USB drive?

A: If you used the "Tools | Copy FreeOTFE to USB drive..." function, and selected the "Setup autorun.inf to launch FreeOTFE when drive inserted" option, FreeOTFE will normally run automatically whenever the drive is inserted (or prompt the user if they want to run it).

However, this does depend on your PC's configuration.

If FreeOTFE doesn't launch automatically (and you don't get prompted to launch FreeOTFE after inserting the drive), you probably have autorun turned off for removable disks.

It is generally recommended that "autorun" functionality be *disabled*, as this can have security implications; should an untrusted USB drive be plugged in, the program specified in an autorun.inf file on the device may be launched - without offering the user the chance to prevent it

To reset (enable) autorun functionality:

1. Click the windows "Start" button
2. Select "Run"
3. Type in "gpedit.msc" and click "OK"
4. Reset the setting for the local computer policy
    1. Select "Local Computer Policy \ Computer Configuration \ Administrative Template \ System"
    2. Double click the "Turn off Autoplay" entry
    3. Change the "Not configured"/"Enabled"/"Disabled" selection to any of the three options
    4. Click "Apply"
    5. Change the "Not configured"/"Enabled"/"Disabled" selection to either "Not configured" or "Disabled"
    6. Click "Apply"
5. Reset the setting for users
    1. Select "User Configuration \ Computer Configuration \ Administrative Template \ System"
    2. Double click the "Turn off Autoplay" entry
    3. Change the "Not configured"/"Enabled"/"Disabled" selection to any of the three options
    4. Click "Apply"
    5. Change the "Not configured"/"Enabled"/"Disabled" selection to either "Not configured" or "Disabled"
    6. Click "Apply"

See also: Enable Autorun on DVD, CD and other removable media

---

Q: Can I use FreeOTFE with "MojoPac"?

A: Yes!

There are two basic ways of encrypting you data using FreeOTFE while using MojoPac:

1. By creating an encrypted volume and installing MojoPac onto it.
2. By installing MojoPac as normal (e.g. onto a USB drive), and running FreeOTFE from within MojoPac

Method one: Installing onto a FreeOTFE volume

The first method is probably the more secure, as your *entire* MojoPac setup is encrypted. Simply create a new FreeOTFE volume on your USB drive, mount it, and then install MojoPac onto the mounted volume.

In this way *everything* relating to your MojoPac system will be secured. Because of FreeOTFE's portable mode, MojoPac can be used as a fully mobile, secured, system by placing a copy of FreeOTFE onto your USB drive along with the volume file.

Method two: Running within the MojoPac environment

FreeOTFE can also be launched and used from within the MojoPac environment to create and use encrypted volumes in much the same way as on a normal PC.

In order to use FreeOTFE in this way, you must first either

- Start FreeOTFE's portable mode on the *host PC*, or
- Install and start the FreeOTFE drivers on the *host PC*

(See the Portable mode and Installation sections for further information)

When running MojoPac, your MojoPac device (i.e. your USB drive, iPod, etc) will appear as *both* the removable drive it is normally mounted as on the host PC (e.g. D:, E:), and as your MojoPac's C: drive.

To mount a FreeOTFE volume which is stored on your *MojoPac device*, you should select the volume file on the removable drive (e.g. D:, E:) and *not* the mirror copy which appears on you MojoPac's C: drive. Mounting volumes stored elsewhere should be unaffected.

Note that when a volume is mounted from within the MojoPac environment, it may also be accessed by the *host PC* by using the drive letter it is mounted as under the MojoPac session. Applications on the host PC will see the mounted volume as normal, with the exception of Windows Explorer which will not show a new drive icon for it - though even then, it can still be accessed by Windows Explorer on the host PC, by simply typing the drive letter the encrypted volume is mounted as, followed by a colon, into Windows Explorer's "Address" bar and pressing <ENTER>.

In the same manner, volumes mounted on the host PC will be accessible from within the MojoPac environment.

Q: Can FreeOTFE be used with RAID arrays?

A: Yes! FreeOTFE has been tested with, and works with, RAID arrays

---

Q: Does FreeOTFE try to connect to the internet??

A: No - not by default.

FreeOTFE and FreeOTFE Explorer will *only ever* try to connect to the internet if it has been configured to check for updates - and even then, they will only try to connect to the FreeOTFE WWW site to retrieve version information.

By default, both FreeOTFE and FreeOTFE Explorer are configured such that thtey will *not* check for updates - this functionality must be explicitly enabled by the user.

Q: How do I check FreeOTFE's exit code when passing parameters via the command line?

A: The easiest way is to check FreeOTFE's exit code is to run it via a batch file. For example, if you create a "FreeOTFE_cmdline.bat" file containing the following:

```
FreeOTFE.exe %1 %2 %3 %4 %5 %6 %7 %8 %9
@echo Exit code: %ERRORLEVEL%
```

and use "FreeOTFE_cmdline.bat" in places of "FreeOTFE.exe"

Q: Why won't FreeOTFE accept my password when supplied via the command line parameter?

A: If you're using the "/silent" switch, try removing it and just clicking "OK" on the password dialog to confirm that your password and other details have been entered correctly.

If FreeOTFE fails to mount, check your command line parameters carefully. If your password or volume filename have spaces in them, you'll need to surround them with doublequotes ("). Similarly "%" signs may be interpreted in batch files as batch file variables.

Q: Do I have to partition my drive to use FreeOTFE?

A: No. FreeOTFE volumes may be stored in files stored on your normal file system.

Q: I want to create a FreeOTFE partition on my unallocated space, but can't see it in the partition display - where is it?

A: For obvious reasons, the FreeOTFE only shows partitions which are reported to it by the OS.

Disk space which does not form any part of a partition (i.e. is not referenced in *any* partition table on the disk (primary or extended); reported as "Unallocated" by the Windows Disk Management tool) cannot be "seen" by FreeOTFE.

To make use of such space, use the Windows Disk Management tool to create a new partition for it, and *then* use FreeOTFE to turn it into an encrypted partition.

Please note that FreeOTFE is *not* responsible for partitioning your hard drive - you should be using a partitioning tool for that!

Q: When I'm prompted to select a partition, some of the partitions on my USB drive are shown in red (or not at all) - why?

A: See: Why can't I use encrypted partitions on a USB drive, unless it's the first partition?

Q: Why can't I use encrypted partitions on a USB drive, unless it's the first partition?

A: MS Windows has a limitation which prevents it from correctly using partitions on USB drives that are beyond the first one. As a result, the *current* version of FreeOTFE cannot use these partitions, and this is indicated by displaying such partitions in red (or not at all) in the partition selection display.

If you wish to use an encrypted partition on a USB drive under both Windows and Linux, please ensure that the encrypted partition is the *first* partition on the USB drive.

It should be noted that this limitation only applies to USB drives, and not physical disks installed inside the PC

A solution which will allow FreeOTFE to use second (and other) partitions on USB drives is currently under development.

Other possible solutions/information may be found at:

- Why is it not possible to partition a USB Flash Stick?
- Multi partition a USB flash drive in Windows
- Removable Media Bit

Q: After creating an encrypted partition/disk, MS Windows reports that partition I used as being type "RAW" and prompts me to format it - why?

A: After creating an encrypted partition/disk, if you have a drive letter associated with the physical partition used, MS Windows will report that drive as being "RAW" since it cannot understand what is stored on it (for obvious reasons, it can't understand what the encrypted data means).

WARNING: Do not let MS Windows format this partition! Although formatting the "virtual drive" FreeOTFE creates after mounting your encrypted partition is certainly a requirement before it can be used, formatting the partition it resides on could destroy your encrypted data!

The safest course of action is to prevent MS Windows from allocating a drive letter to the encrypted partition. By doing so:

- MS Windows will not prompt you every time this drive is accessed, since you will not be able to accidentally access it
- You'll be less likely to hit "OK" and format the partition, overwriting your encrypted data!

To do this, see the FAQ "How do I "hide" an encrypted partition such that MS Windows doesn't allocate it a drive letter?"

Q: How do I "hide" an encrypted partition such that MS Windows doesn't allocate it a drive letter?

A: Carry out the following steps:

1. Go to "Start -> Settings -> Control Panel -> Administrative tools -> Computer Management"
2. Select "Disk Management"
3. Rightclick on the partition you have setup an encrypted and select "Change Drive Letter and Paths"
4. Remove any drive letters associated with the partition

Windows should then remove any drive letters associated with the encrypted partition.

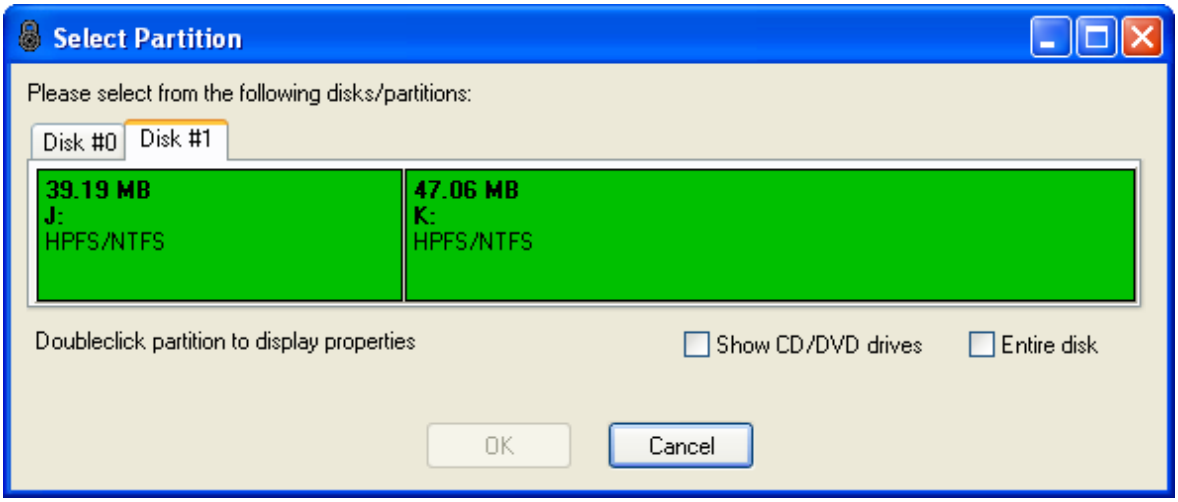
Q: Why does the partition/disk selection display sometimes display less information?

A: Depending on the user's access rights, FreeOTFE may only be able to obtain limited information about the various disk partitions.

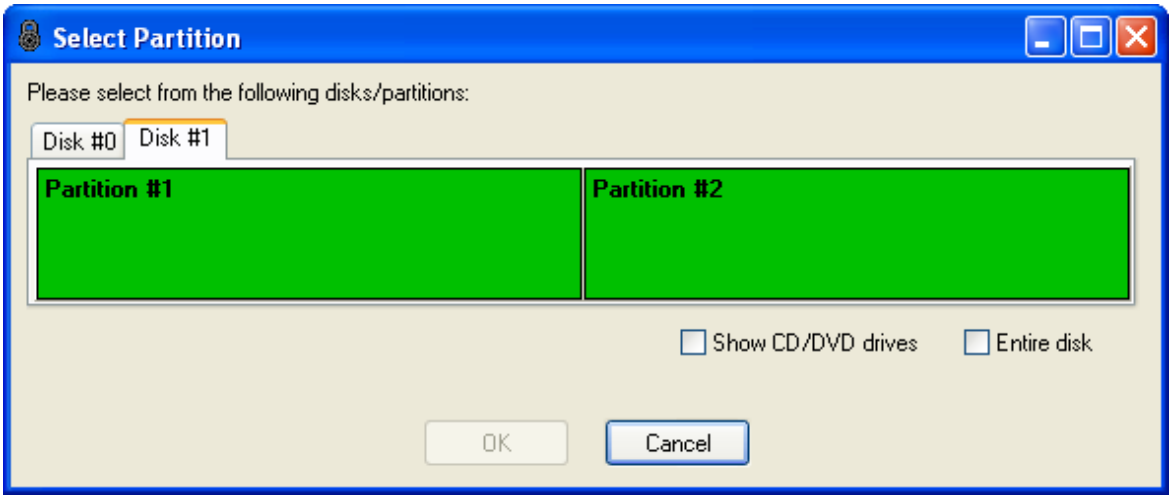
When this happens, FreeOTFE will fallback to displaying a more restricted set of information (e.g. no partition sizes)

Because more information can be displayed if the user is an administrator (or under Windows Vista, the FreeOTFE process has been started with escalated under UAC), it is *highly* recommended that any partition based volumes are created when logged in as an administrator. (Under Vista, FreeOTFE should be launched by rightclicking on the executable, "FreeOTFE.exe", and selecting "Run as administrator".)

By displaying additional information, there is less likelihood of creating a volume on the wrong partition.



Partition selection dialog; full information shown



Partition selection dialog; restricted information shown

Q: I accidentally selected the wrong disk/partition when creating a new volume and now can't see my files! How can I get my data back?

A: The more important thing to do in this kind of situation is **STOP** and **THINK**. Before attempting any kind of recovery, understand what you are going to do and how you are going to do it - *before* doing anything.

For safety reasons, FreeOTFE only writes the initial 512 byte CDB to the start of the disk/partition when creating a new disk/partition based volume (see the Plausible Deniability section for how to initialize a volume by overwriting it). If you haven't yet mounted the volume and started writing data to it, or overwriting it, you have a good chance of getting your files back.

Obviously, if you have written data to the encrypted volume (e.g. by selecting one of the overwrite options or copying files to it), the amount you will be able to recover will decrease.

The recommended approach to recovering the data originally stored on the disk/partition is to:

1. Dismount all mounted volumes.
2. Take an image of the disk/partition the volume was created on (e.g. by using a tool such as USB Flash Tools, or any disk imaging/cloning tool)
3. Use any standard recovery software (e.g. Restorer 2000 Pro) on the *image taken* - not the disk/partition itself - to try to recover your data.

Q: Does FreeOTFE offer whole disk encryption?

A:Yes! FreeOTFE *does* support whole disk encryption, although it does not yet support encrypting the system partition (i.e. the entire disk or partition that the OS boots from)

To encrypt a whole disk, proceed as though creating an encrypted partition and select the "entire disk" checkbox after selecting the drive to be used.

Q: Do I *have* to use a security token/smartcard with FreeOTFE?

A: No! FreeOTFE offers security token/smartcard as an option to provide additional security, they are not necessary to use FreeOTFE.

Q: What is the difference between PKCS#11, Cryptoki, and "tokens"?

A: PKCS#11 and Cryptoki are the same thing; an API for accessing security tokens/smartcards.

"Token" is a generic term to refer to a security token or smartcard.

Q: Does FreeOTFE encrypt my *entire* encrypted volume using my PKCS#11 token?

A: No, just the volume's CDB/keyfile. Encrypting the entire volume would incur significant performance penalties due to the relatively low power of security tokens when compared to a PC, and need to transfer data *twice* over the USB connection (once to sending the encrypted/plaintext data, and again to receive the plaintext/cyphertext)

Q: I've inserted my PKCS#11 (Cryptoki) token, but why is the "PKCS#11 token management..." menuitem disabled?

A: Please ensure that you have configured FreeOTFE to use your token via the "PKCS#11" tab on the Options dialog ("View | Options...")

See the section on Security Token/Smartcard Support for further details

---

Q: How do I change the password on a volume/keyfile which is secured with a PKCS#11 secret key?

A: To change the password on a volume/keyfile which is secured with a PKCS#11 secret key:

1. Decrypt the volume's CDB/keyfile using the token's secret key:
    1. Go to "Tools | PKCS#11 token management..."
    2. Select the "Secret keys" tab
    3. Select the appropriate secret key
    4. Click "Decrypt", and select your volume/keyfile
2. Change the password on it
3. Re-encrypt the keyfile/volume's CDB using the token's secret key, using the "encrypt" function on the PKCS#11 token management dialog

---

Q: Can I use more than one security token with FreeOTFE?

A: Yes! FreeOTFE supports as many security tokens as you've got!

You can even use different tokens to mount different volumes, or the same token to mount multiple volumes, all at the same time if you wish!

The only caveat being that your PKCS#11 library provider may only support up to a certain number of security tokens being plugged in *at the same time* (typically this may allow up to 16 tokens to be used simultaneously)

Q: Why don't all of my volumes automatically dismount when I remove my security token?

A: First, please check that you have configured FreeOTFE to autodismount volumes on token removal by:

1. Go to "View | Options..."
2. Select the PKCS#11 tab
3. Ensure that the "Auto dismount PKCS#11 volumes when associated token is removed" is checked

If you dismount, then remount, your volumes with your PKCS#11 token, they should be dismounted when it is removed.

Please note that *only those volumes which were mounted with the removed token* will be automatically dismounted.

More than one token may be used at the same time; again, only those volumes mounted with the removed token will be automatically dismounted.

Q: (Windows Vista only) Why do I get "unidentified program wants access to your computer" prompts when using FreeOTFE?

(This FAQ is only applicable when running under Windows Vista and later; it is not relevant for other operating systems)

A: Windows Vista incorporates a new security system called "User Access Control" (UAC), which is there to help prevent malicious software from doing things which could be harmful to your computer.

Whenever you attempt to use any part of FreeOTFE's functionality which Windows considers a malicious program could use to cause harm, Windows displays this dialog (called the "consent/credential" dialog), and asks you if you would give your permission for it to continue. You will be shown this dialog even if you are logged on as an Administrator.

The same type of dialog will appear when you attempt to (for example) go to Window's Control Panel, selecting "Date and Time", and then attempting to change the computer's time or date.

Because the FreeOTFE executable does not have a digital signature that Windows recognises, this dialog claims that "An unidentified program wants access to your computer". This is perfectly normal, and part of Vista's system to help protect you. If you would like to check that your copy of FreeOTFE is an original, you may do so by checking the hashes/signatures available from the FreeOTFE WWW site.

These prompts form part of Windows Vista's "User Access Control" (UAC) system, which you can find out more about from the Microsoft WWW site.

Q: (Windows Vista only) Why does FreeOTFE prompt me to enter my Administrator's password?

(This FAQ is only applicable when running under Windows Vista and later; it is not relevant for other operating systems)

A: FreeOTFE doesn't ask you to enter an Administrator's password; it has no use or need for this information. Windows Vista, however, will prompt you to enter an Administrator's password whenever you are logged in as a "standard" (i.e. non-Administrator) user, and attempt to carry out any operation which it deems could be harmful to your computer.

If you are happy for FreeOTFE to carry out the operation you requested of it, you should select the relevant option from the consent/credential dialog, and enter the appropriate Administrator's password to allow FreeOTFE to proceed.

Those operations which require Administrator's explicit approval before Windows Vista will permit you to carry them out are marked in FreeOTFE with a "shield icon".

It should be emphasised that it is Windows Vista itself which is generating these prompts, and not FreeOTFE, which will have no access to the password you type in.

These prompts form part of Windows Vista's "User Access Control" (UAC) system, which you can find out more about from the Microsoft WWW site.

Q: (Windows Vista only) How do I stop the Windows Vista "consent/credential" (UAC) dialog from being displayed?

(This FAQ is only applicable when running under Windows Vista and later; it is not relevant for other operating systems)

A: To prevent the UAC dialogs from being shown when using FreeOTFE (and all other applications), you can disable it by carrying out the following steps:

1. Click on the "Start" button, and then select "Control Panel"
2. Doubleclick "User Accounts"
3. Click on "Turn User Account Control on or off"
4. Make sure that the "Use User Account Control (UAC)" checkbox is unchecked
5. Click "OK"
6. Restart your computer

Q: (Windows Vista only) I have problems starting any of the drivers under the 64 bit version of Windows Vista/Windows 7 - what's wrong?

(This FAQ is only applicable when running under Windows Vista and later; it is not relevant for other operating systems)

A: The 64 bit versions of MS Windows Vista and MS Windows 7 both use driver signing; please see the section on installing FreeOTFE on Windows Vista x64 and Windows 7 x64

Q: (Windows Vista only) What are the little "shield" icons shown next to some menuitems?

(This FAQ is only applicable when running under Windows Vista and later; it is not relevant for other operating systems)

A: Functions marked with a "shield" icon require Administrator privileges in order to use them, for security reasons. This is for your security, and more information can be found on the Microsoft WWW site.

10.4 FreeOTFE4PDA Specific (PDA)

Q: I created my volume file using the PDA version of FreeOTFE and can mount it on my PC - but why does it keep asking if I want to format it?

A: When you created your volume on your PDA, your PDA fully formatted the volume as though it was a new device - not just a partition on a device.

In order for a volume to be mounted and used correctly on both a PDA and PC, it should be created and formatted using a PC as a FAT volume, and subject to the maximum volume size your PDA can support (see FAQ on volume sizes). This will ensure it can be read on all systems.

Q: I created a volume on my PC, and can mount it successfully on my PDA - but can't see any of my files!

A: The chances are that you formatted your volume on your PC using NTFS, and your PDA doesn't support this filesystem.

Volumes which are to be used on a PDA should normally be formatted as FAT or FAT32; this should be carried out on a PC, not a PDA (see related FAQ).

Please note also that FAT32 can only support volumes up to (4GB less one byte)

Q: How can I speed FreeOTFE up when mounting my volumes?

A: For security reasons, FreeOTFE doesn't store any information relating to which hash/cypher combination was used to encrypt a FreeOTFE volume.

As a result, FreeOTFE is forced to cycle through all of its possible hash/cypher combinations in order to determine which one to use. Reducing the number of combinations it has to check can significantly reduce the time this takes.

To reduce the number of combinations, without making any difference to the level of security FreeOTFE offers, simply disable any redundant cypher/hash implementation drivers such as either one of:

- FreeOTFE4PDACypherAES_Gladman.dll
- FreeOTFE4PDACypherAES_ltc.dll

and any two of:

- FreeOTFE4PDACypherTwofish_Gladman.dll
- FreeOTFE4PDACypherTwofish_HifnCS.dll
- FreeOTFE4PDACypherTwofish_ltc.dll

Please see Advanced Topics, "Enabling/Disabling Hash/Cypher Algorithms" section, for instructions on how to enable/disable hash/cypher drivers.

(Please see the FAQ on duplicated drivers for an explanation as to why multiple implementations are included in the release)

The mount time can be reduced even more dramatically by disabling all of the hash/cypher drivers *except* for the ones which you have secured your data with. This however could decrease the level of security offered, as doing so would make it pretty clear to any attacker which combination you've used - though it's debatable whether this loss in security will actually be of any practical value to an attacker.

To speed things up even further, you could drop the number of key iterations your volume is secured with. This isn't particularly recommended, but might help some users...

Q: Does the PDA version support Linux volumes?

A: Yes!

FreeOTFE4PDA v4.0 and later support LUKS volumes.

A front-end interface to allow support for other Linux encrypted volumes is currently being implemented, and will appear in a later release.

Q: Why does FreeOTFE4PDA's version numbering skip from v0.55 to v2.00; what happened to v1.00?

A: FreeOTFE4PDA's version number was incremented to v2.00 in order to match the PC version of FreeOTFE, with which FreeOTFE4PDA shares a fair amount of common code.

A specific "v1.00" was never released, although there were a fair number of non-public versions released between v0.55 and v2.00 to various people to help with testing and confirm compatibility.

Q: Why does FreeOTFE4PDA's version numbering skip from v3.76 to v5.00; what happened to v4.00?

A: FreeOTFE4PDA's version number was incremented to v5.00 in order to better reflect that its level of functionality was on a par with the PC version with the same version number, after support for encrypted Linux volumes and language translations were added in v5.00.

A specific "v4.00" was never released, although there were a fair number of non-public versions released between v3.76 and v5.00 to various people to help with testing and confirm compatibility.

Q: When I use the "open" dialog to select my volume file/keyfile, it doesn't list the file I'm trying to specify - even when I select "All files" - where is it?

A: The standard Windows Mobile "open file" dialog is a little odd; this isn't just restricted to FreeOTFE!

Although FreeOTFE allows you the freedom to use *any* filename you wish, only files which have a filename extension (i.e. the volume's filename has a full stop followed by one or more letters) will be listed in the "open file" dialog; even if you selected the display "All files" option.

Furthermore, this dialog will only display those files located in the following places:

- Any subdirectory on a storage card which is located from the root directory of that storage card.
- Files in the "My Documents" directory on your PDA
- Files in any subdirectory immediately underneath your "My Documents" directory

The simplest solution is to rename your file, and move it into one of the directories indicated above.

Alternatively, you can still specify your file by simply typing its full path and filename into the relevant entry box, instead of clicking "..." and using the "open file" dialog to select it.

Note that you don't *need* a filename extension, and can store volume/key files *anywhere* on your PDA. Conforming to the above restrictions allows you to use the "open file" dialog to select your files, and does not affect FreeOTFE's operation in any way.

Q: How can I reduce the amount of storage space FreeOTFE4PDA takes up when installed?

A: The easiest way of reducing FreeOTFE4PDA's installed "footprint" is to delete its user documentation from your PDA (i.e. everything in the "docs" subdirectory).

You don't (or at least, shouldn't!) *really* need this documentation as FreeOTFE4PDA is a pretty straightforward application to use - and if you do find you want to refer to it occasionally, tapping on "Help | User guide" will take you to the online version if a local copy cannot be found.

It is recommended that you keep a copy *somewhere* though; on your desktop PC, if nowhere else.

You can further reduce the amount of storage taken up by deleting any unused cypher and hash drivers; this will also increase the speed at which FreeOTFE4PDA will mount volumes. (See FAQ: "How can I speed FreeOTFE up when mounting my volumes?" for further details on how to do this)

If you don't need any of the language translations (or only one of them), deleting those translations you *don't* need from the "locale" subdirectory can free off a small amount more storage.

Q: Why do I get the message "Unable to locate local copy of user guide; would you like to see the latest version on the Internet?" when I try to view the user guide by selecting "Help | User guide"?

A: FreeOTFE4PDA attempts to locate a local copy of the user guide stored with the executable. If this is not found, it will fallback to trying to show you the latest version found on the FreeOTFE WWW site.
To prevent this, please place a copy of the "docs" directory included with the release into the same directory as your "FreeOTFE4PDA.exe" executable. (i.e. Such that you have a "docs" subdirectory in the same directory as the "FreeOTFE4PDA.exe" executable on your PDA)

Q: When I try to mount a volume, I sometimes the error: "Mount failed; the virtual storage device could not be activated at this time"

A: If you see this error message, you have correctly entered all details to allow FreeOTFE4PDA to mount your encrypted volume, however Windows Mobile has failed to activate the FreeOTFE4PDA virtual storage device.

This error appears to be related to the wireless functions (mobile phone/wifi/bluetooth) of these particular devices; turning off wireless functionality (and possibly carrying out a soft-reset of the device) can resolve this issue.

This issue has been reported to affect the following devices:

- T-Mobile Vario (WM v5.1.195 (Build 14847.2.0.0))
- MDA Vario II/HTC TyTn (WM v5.1.195 (Build 14955.2.3.0)
- O2 Exec/QTEK 9000 (WM v5.1.195 (Build 1487.2.0.0))
- Fujitsu Siemens Loox T830 (WM v5.1.195)

A version of FreeOTFE4PDA which should resolve this issue is currently under development and will appear in a later release

---

Q: Which PDAs will FreeOTFE4PDA work with?

A: FreeOTFE4PDA has been tested with various Windows Mobile 2003/2005 and Windows Mobile 6 devices, and should work with all Windows Mobile 2003 and later PDAs.

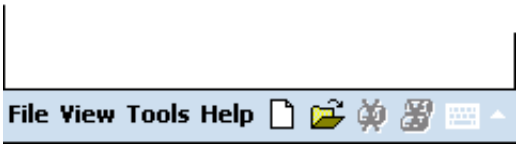
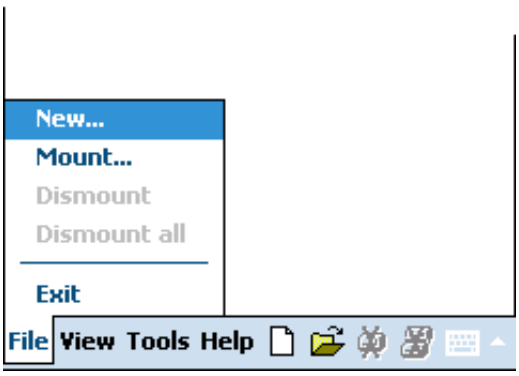
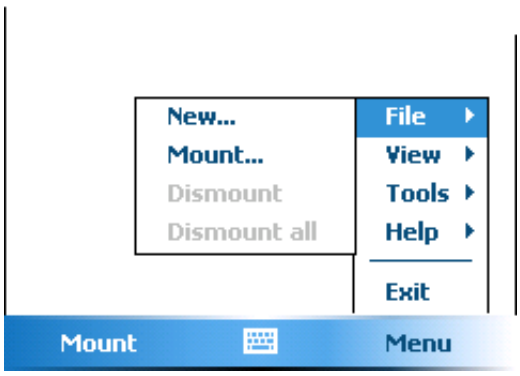
Smartphones *which do not have a touchscreen* may not display FreeOTFE4PDA's interface correctly though. Smartphones which *do* have a touchscreen *can* use FreeOTFE4PDA.

Q: What does the "Support WM 5.0 soft keys" option do?

A: Under Windows Mobile 5.0 and later, you have the option of displaying FreeOTFE's menus and Wizard navigation using the new style two-item "softkey" menus. This is the "Microsoft standard" for Windows Mobile 5 (and later) applications, and is designed to allow users with "softkeys" (i.e. smartphones with two buttons, left and right, below their display) to navigate more quickly and easily.

Alternatively, you can still opt to use the older "menu and toolbar" style used with Windows Mobile 2003 (second edition) and earlier.

Here is a sample of what the different menus look like:

Menu and toolbar style menus	**WM 5.0 soft key menus**
	
	

You can change this setting by going to "Tools | Options".

If you have a PDA which runs Windows Mobile 2003 (second edition) or earlier, your PDA does not support this new style menu.

Q: I don't like the new two-item menus at the bottom of my display - how do I change them back to the older toolbar style menu?

A: The "Microsoft standard" for Windows Mobile 5 (and later) applications is to employ a two-item menu at the bottom of the display, as opposed to using a similar style menu as is found on desktop PCs running MS Windows.

This is to allow users with "softkeys" (i.e. smartphones with two buttons, left and right, below their display) to navigate more quickly and easily.

Of course, as with all user interfaces, there's always someone who doesn't like it! FreeOTFE4PDA does give you the option to change back the older style though; simply tap "Menu | View | Options" and *uncheck* the "Support WM 5.0 soft keys" option.

(See also FAQ: What does the "Support WM 5.0 soft keys" option do?)

Q: I don't like the multi-item menubar/toolbar at the bottom of FreeOTFE4PDA's display - how do I get it to use the newer two-item style (softkey) menus instead?

A: The two-item menu at the bottom of the display is the "Microsoft standard" for Windows Mobile 5 (and later) applications, and is designed to allow users with "softkeys" (i.e. smartphones with two buttons, left and right, below their display) to navigate more quickly and easily.

If you have a PDA which runs Windows Mobile 2003 (second edition) or earlier, your PDA does not support this new style menu.

However, if you are running Windows Mobile 2005 or later, you can enable the two-item style menu by simply tapping "Menu | View | Options" and making sure the "Support WM 5.0 soft keys" option is checked.

(See also FAQ: What does the "Support WM 5.0 soft keys" option do?)

Q: Can I use my PC volumes with the PDA version?

A: Yes!

See FAQ Can I use the same encrypted volumes on *both* my PC and PDA?

Q: I upgraded to the lastest version of FreeOTFE4PDA, can I still mount my old volumes?

A: Yes, you can!

It should be noted that v3.75 and later only have the SHA hash and AES cypher drivers enabled by default. If your volume was secured using a different hash/cypher algorithm, please enable the required drivers by following the procedure described in Advanced Topics, under "Enabling/Disabling Hash/Cypher Algorithms".

Q: I would like to use hash/cypher *XYZ*, why doesn't it appear as an option?

A: The most likely reason is that *XYZ* has been disabled within FreeOTFE4PDA; please see Advanced Topics, "Enabling/Disabling Hash/Cypher Algorithms" section, for how to enable/disable hash/cypher algorithms, and show *XYZ* as an option.

Q: How to I enable hash/cypher *XYZ*?

A: See Advanced Topics, "Enabling/Disabling Hash/Cypher Algorithms" section.

10.5 FreeOTFE Explorer Specific

Q: Does FreeOTFE Explorer support drag and drop with MS Windows Explorer?

A: Yes - FreeOTFE Explorer supports dragging files and folders *from* MS Windows Explorer *to* FreeOTFE Explorer, but doesn't currently support dragging files *from* FreeOTFE Explorer *to* MS Windows Explorer.

Q: What filesystems does FreeOTFE Explorer support?

A: FreeOTFE Explorer supports volumes using the FAT12, FAT16 and FAT32 filesystems. Support for other filesystems is currently under development.

Q: Does FreeOTFE Explorer try to connect to the internet?

A: See FAQ "Does FreeOTFE Explorer try to connect to the internet?"

---

11 Technical Details

11.1 FreeOTFE Volumes and Keyfiles

A FreeOTFE volume (regardless of whether its stored in a file or partition) consists of two things:

1. A critical data block (CDB)
2. An encrypted partition image

The CDB may either form part of the volume, in which case it is prepended to the encrypted partition image, or it may be stored as a separate file, in which case it is referred to as a "keyfile".

Users may create any number of keyfiles for any given volume. To create a new keyfile, the user must supply either:

1. An existing keyfile, and its password, etc
2. A volume file which has a CDB

together with its password, salt length, etc. The keyfile or volume CDB supplied will then be read in, decrypted, and re-encrypted with a new password, salt length, etc (all supplied by the user) before being written out as the new keyfile.

A full definition of the contents of a CDB/keyfile is supplied in this documentation.

11.1.1 Notes

- A FreeOTFE keyfile is nothing more than a CDB, the "volume details block" of which contains the encryption details used for securing the volume it relates to
- A volume may have one or more keyfiles, in which case they all share the same data stored within their respective "volume details block", but each one is encrypted with a different user password, salt, random padding, etc - making each keyfile unique.
- Keyfiles are encrypted with the same cypher/hash that the encrypted partition image they relate to is encrypted with.

11.2 Technical Details: Critical Data Block Layouts

Please select which CDB Format you would like information on:

- CDB Format ID 1 *(obsolete)*
- CDB Format ID 2 *(obsolete)*
- CDB Format ID 3 *(obsolete)*
- CDB Format ID 4

11.2.1 CDB Format ID 1

NOTE: This CDB layout is obsolete; all new volumes should use the latest CDB format.

11.2.1.1 Overview

The following table describes the high-level layout of all FreeOTFE (not Linux) volume files:

<table>
<tr>
<td colspan="8">Critical data block:</td>
<td rowspan="2">Encrypted partition image</td>
</tr>
<tr>
<td>Password salt</td>
<td colspan="6">Encrypted block:
<table><tr><td>Check hash</td><td>Volume details block:
<table><tr><td>Critical data version</td><td>Volume flags</td><td>Encrypted partition image length</td><td>Master key length</td><td>Master key</td><td>Requested drive letter</td><td>Random padding #2</td></tr></table></td></tr></table></td>
<td>Random padding #1</td>
</tr>
</table>

Color key:

Color	Encryption used
	Red items are not encrypted
	Blue items are encrypted with the user’s password, password salt, and user’s chosen hash/cypher algorithms
	Green items are encrypted with the master key and the user’s chosen cypher, with IVs generated with the user’s chosen sector IV method

Seem intimidating? Read on, and all will become clear... When broken down into its component parts, the volume structure is reasonably straightforward to understand.

11.2.1.2 Breakdown: Top level

Critical data block	Encrypted partition image

At the highest level, volume files can be broken down into just two basic parts: A critical data area, and the encrypted partition itself.

Item name	Size (in bits)	Description
Critical data block	4096	This part of the volume file contains "critical data", including the master en/decryption key used for the encrypted partition image, the method used for generating sector IDs, and a check hash. See below for further breakdown.
Encrypted partition image	User defined. (max 2^64 bytes; circa 16777216 TB)	This is a literal partition image which represents the volume. This image is encrypted with: 1. The master key 2. The user’s chosen cypher 3. The user’s chosen method for generating different sector IVs
Total size:	4096 + Encrypted partition image size	

11.2.1.3 Breakdown: Critical data block layout

Password salt	Encrypted block	Random padding #1

Item name	Size (in bits)	Description
Password salt	sl (User specified to a max 512)	This data is appended onto the user’s password, before it’s hashed. The resulting hash may then be used to encrypt/decrypt the "Encrypted block".
Encrypted block	If bs>0 then: ((4096 - salt length) div bs) * bs If bs<=0 then: (4096 - salt length)	This block contains the actual key which is used to encrypt/decrypt the encrypted partition image. See below for further breakdown.
Random padding #1	((4096 - sl) % bs)	Random "padding" data. Required to pad out any remaining, unused, bits in the "critical data block"
Total size:	4096	

11.2.1.4 Breakdown: Encrypted block layout

Check hash	Volume details block

As described above, this entire block is encrypted using the user's password, salt, and chosen hash and cypher algorithms.

As this block is encrypted, its length (in bits) must be a multiple of the cypher's blocksize.

Item name	Size (in bits)	Description
Check hash	hk	This is the hash of the plaintext version of the "Volume details block". If hk is zero, then this hash will be either truncated to 512 bits, or right-padded with 0 bits up to a maximum of 512 bits
Volume details	Total size - *hk*	This stores the details of how to encrypt/decrypt the encrypted partition.
Total size:	If bs>8 then: ((4096 - sl) / bs) * bs If bs<=8 then: (4096 - sl)	Note: "/" represents integer division

---

11.2.1.5 Breakdown: Volume details block layout

Critical data version	Volume flags	Encrypted partition image length	Master key length	Master key	Requested drive letter	Random padding #2

Finally, we reach the details that the critical data block was designed to protect:

Item name	Size (in bits)	Description
Critical data version	8	This is a version ID which identifies the layout of the remainder of the volume details block When this layout format is used, this will always be set to 1.
Volume flags	32	Bitmap flagging various items. Currently, this only identifies which method is used in generating sector IVs, used in encrypting/decrypting the encrypted partition image. Bit - Description 0 - Use different IVs for each sector 0 = Use NULL IV 1 = Use sector ID (possibly hashed) as IV 1 - Sector ID zero is at the start of the file 0 = Sector ID zero is at the start of the encrypted data 1 = Sector ID zero is at the start of the host volume file 2 - <unused> 3 - Hash sector ID before use 0 = Sector ID used as IV 1 = Hash sector ID before using is as an IV
Encrypted partition image length	64	This stores the length (in bytes) of the encrypted partition image
Master key length	32	This will be set to the length of the master key in bits. Not strictly needed, but used as a sanity check.
Master key	*ks* *(max 1536)*	This is set to the random data generated when the volume was created; it represents the en/decryption key used to encrypt the encrypted partition image
Requested drive letter	8	The drive letter the volume should be typically be mounted as. Set to 0x00 if there is no particular drive letter the volume should be mounted as; mount using the first available drive letter.
Random padding #2	All remaining bits to pad out to Total size	Random "padding" data. Required to pad out the encrypted block to a multiple of bs, and to increase the size of this block to the maximum length that can fit within the "critical data block".
Total size:	(((4096 - sl) / bs) * bs) - *hk*	Note: "/" represents integer division

11.2.1.6 Miscellaneous Comments Regarding the FreeOTFE Volume File Layout

The design of the critical data layout eliminates the need for the cypher/hash used to be stored anywhere, denying an attacker this information and increasing the amount of work required to attack a volume file.

The "critical data block" is encrypted.

The "password salt" appears before the "encrypted block", and no indication of the length of salt used is stored anywhere in order to prevent an attacker from even knowing where the "encrypted block" starts within the "critical data block".

The "Check hash" appears before the volume details block. This may seem a little strange since the size of "Check hash" is variable (its actual length is dependant on the hash algorithm chosen), but appears first in order to reduce the amount of "known" (or predictable) data early on in the volume. Theoretically this is desirable as (for example) cyphers operating in CBC (or similar) modes which "whiten" data will do so with data that is less predictable than would occur if the hash appeared towards the end of the block it appears in.

The "Check hash" is limited to 512 bits. This is limited as, in practical terms, some kind of limit is required if the critical data block is to be of a predetermined size. See section on mounting volume files for how multiple matching check hashes is handled.

The "Password salt" is (fairly arbitrarily) limited to 512 bits. Again, this is primarily done for practical reasons.

Although at time of writing (2004) this limit to the length of salt used should be sufficient, the format of the critical data block (with included layout version ID) does allow future scope for modification in order to allow the critical data block to be extended (e.g. from 4096 to 8192 bits), should this limit be determined as insufficient.

The "Encrypted block" does contain a certain amount of data that may be reasonably guessed by an attacker (e.g. the critical data version), however this would be of very limited use to an attacker launching a "known plaintext" attack as the amount of this data is minimal, and as with pretty much any OTFE system, the "Encrypted partition image" can reasonably expected to contain significantly more known plaintext anyway (e.g. the partition’s boot sector)

11.2.1.7 Creating FreeOTFE Volumes

To create a FreeOTFE volume file, a fairly significant amount of information is required due to freedom that FreeOTFE gives you in creating volume files.
Broadly speaking, creating a FreeOTFE volume consists of three distinct stages:

1. Creating a file large enough on the local filesystem
2. Writing the critical data block to the volume file
3. Mounting the volume, formatting it, and "shredding" (overwriting) all free space

Stage 1 is straightforward; write data to the file until is has gained the required size. This stage is skipped in the case of creating a hidden volume.
Stage 2 is more complex; and will be described below.
Stage 3 is required in set the volume up for use, and increase security.

11.2.1.8 Writing the critical data block.

The following procedure is used to build up a FreeOTFE's critical volume block:

1. Obtain all the information which will be stored within the volume's "Volume details block"
2. The following information is determined:
    1. "sl" - The number of "salt" bits required by the user
    2. "ks" - The keysize (in bits) of the user's chosen cypher. If the cypher driver reports that this value is "-1", then "ks" should be assumed to be 8 bits.
    3. "bs" - The blocksize (in bits) of the user's chosen cypher. If the cypher driver reports that this value is "-1", then "bs" should be assumed to be 8 bits.
    4. "hk" - The length (in bits) of hashes generated by the user's chosen hash algorithm. If the hash driver reports that this value is "-1", then "hk" should be assumed to be 512 bits.
3. Build the "Volume details block" in memory. The length of this block (in bits) is calculated as:

    (((4096 - sl) / bs) * bs) - *hk*

    (See the layout breakdown diagrams of the "Critical data area" to understand where this algorithm comes from.)
4. With the user's selection of hash algorithm, hash the "Volume details block" to generate the "Check hash".
5. If the "Check hash" generated has less than "hk" bits, it should be right-padded with zero bits until it is "hk" bits long.
6. If the "Check hash" generated has more than "hk" bits, it should be truncated to the first "hk" bits.
7. Prepend the "Check hash" onto the "Volume details block" to build the plaintext version of the "Encrypted block"
8. Append the salt onto the user's password
9. Hash the salted password with the user's chosen hash algorithm
10. If the hash generated has less than "ks" bits, it should be right-padded with zero bits until it is "ks" bits long.
11. If the hash generated has more than "ks" bits, it should be truncated to the first "ks" bits.
    The truncated/padded hash will be used as the "critical data key"
12. Encrypt the plaintext "Encrypted block" with the user's selection of cypher, an IV of "bs" zeroed bits, and the "critical data key"
13. Prepend the salt onto the cyphertext version of the "Encrypted block"
14. Append "Random padding #1" to the "Encrypted block" in order to pad it out to 4096 bits (512 bytes) and form the complete "Critical data block"
15. Write the "Critical data block" to the volume file, beginning from the start of the file for normal FreeOTFE volumes, or from the user's offset in the case of a hidden volume.

11.2.1.9 Mounting FreeOTFE Volumes

IMPORTANT: Versions of FreeOTFE (v00.00.0x) that used this CDB format had an implementation fault that caused the Volume Details Block to be incorrectly parsed when it was read in. It incorrectly used the 32 bits starting from offset byte 10 within the Volume Details Block (i.e. data storing the part of the encrypted partition image length) as the VolumeFlags. This has been compensated for in later versions of FreeOTFE, which use a later CDB layout in any case.

To mount a FreeOTFE volume, the following information must be determined:

1. The location of the critical data block within the volume file
2. The user's password
3. The length of salt used, in bits (sl)
4. The algorithm used to hash the user's password and salt
5. The cypher used for all en/decryption operations
6. The master key to be used to en/decrypt the encrypted partition image
7. The method used to generate the different IVs for each sector of the encrypted partition image
8. The length of the encrypted partition image

Items 1-3 are obtained by asking the user.
Items 3 and 4 are determined by brute force; by hashing the user's password with the first sl bits of the critical data block with each of the hash algorithms installed, then using that information to decrypt the remainder of the critical data block using each of the installed cyphers in turn. After each decryption, the decrypted critical data block is hashed, and this hash value is compared to the decrypted version of the check hash.
Items 6-8 are then taken from the successfully decrypted critical data block, and the volume is mounted.

In detail, the following procedure is used:

1. The user is prompted for the following information:
   - The volume's password
   - The drive letter they wish to mount the volume as.
   - In case the user opted to change the default amount of "salt" used during volume creation, the user is also be prompted to enter the size of the salt used, in bits (Call this value "sl").
   - The offset within the file where the critical data block is located. Typically this will be zero, unless a hidden volume is to be accessed (see information on "hidden volumes").
2. The first 512 bytes (4096 bits) of the file are read in, beginning from the user specified offset.
3. The first "sl" bits of which are assumed to be the salt, and are appended to the user's password.
4. For each hash algorithm installed on the user's computer:
   *Note: We will use "*hk*" to describe the length of hash values generated by each hash algorithm as it is used in turn. This value is as reported by the hash driver. If the hash driver reports that the hash length "-1", we will use the smaller of 512 bits, and the length of the data that the hash algorithm actually returns, as "*hk*".*
   1. The combined user's password and salt previously generated is hashed, giving an hk bit "critical key".
   2. For each cypher installed on the user's computer:
      *Note: We will use "*ks*" to describe the keysize of each cypher as it is used in turn, the keysize being as reported by the cypher driver. If the cypher driver reports that the cypher's keysize is "-1" (no specific keysize), we will use "*hk*" as "*ks*".*
      *Note: We will use "*bs*" to describe the blocksize of the cypher, as reported by the cypher driver. If the cypher driver reports that the cypher's keysize is "-1" (no blocksize), we will use 8 bits (1 byte) as "*bs*".*
      1. The "Encrypted block" is extracted from the critical data block. The length of the "Encrypted block" is calculated as:

((4096 - sl) / bs) * bs

Alternatively, the length of "Random padding #1" can be viewed as being:

((4096 - sl) % bs)

Where % represents the modulus, and / integer division

2. The "Encrypted block" is decrypted using:
    1. The first "*ks*" bits of the hash value previously generated as the decryption key; if "*hk*" is less than "*ks*", then the hash will have extra zero bits appended to it in order to pad it out to "*ks*"
    2. An IV consisting of a block of "*bs*" bits, all set to 0.
3. Next, the "Check hash" and "Volume details block" are extracted from the decrypted data. In order to do this, the length of the check hash must be known; if the hash algorithm used on the salted user's password is reported to generate hashes "-1" bits long, then the check hash is assumed to be 512 bits long. Otherwise, the actual hash length is used.
4. The "Volume details block" is hashed and right padded with zero bits to make it the same length as the "Check hash", which it is then compared with.
   If the hash generated matches the "Check hash", then the "Encrypted block" is deemed to have been successfully decrypted; a note will be made of the cypher and hash used, together with the contents of the decrypted "Volume details block".
   In either case, all remaining hash/cypher combinations will be processed in the same manner.

5. After all possible hash/cypher combinations have been exhausted:
    - If no cypher/hash combination successfully decrypted the "Volume details block", the user will be informed that they have entered incorrect details.
    - If more than one cypher/hash combination successfully decrypted the "Volume details block", the user should be prompted to choose which of the combinations they wish to use. The system will proceed as if only the user selected combination had been successful in decrypting.
    - If exactly one cypher/hash combination successfully decrypted the "Volume details block", then the "Encrypted partition image" will be decrypted as necessary using this cypher/hash combination, together with the information obtained from the decrypted "Volume details block".

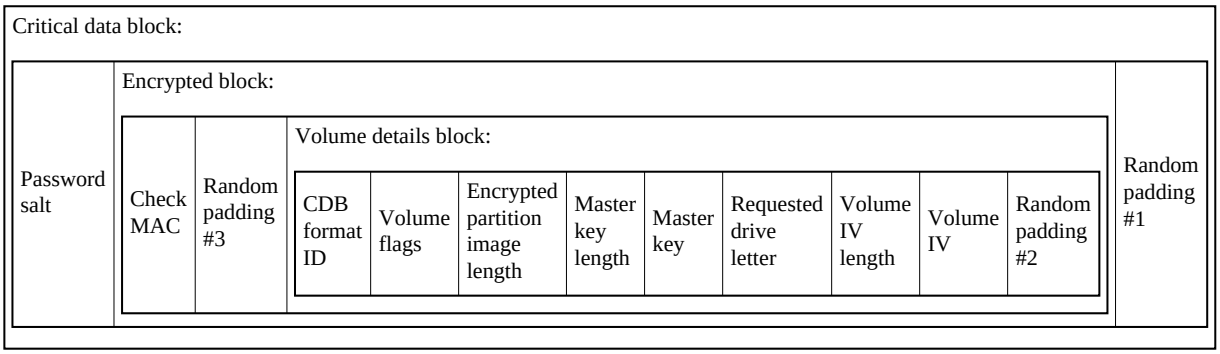
As a result of the volume's layout, it is not necessary to store the cypher or hash algorithm used for en/decryption anywhere. Nor is it necessary to prompt the user for this information since during mounting a FreeOTFE formatted volume, this information can be determined dynamically by attempting the mount using all possible combinations of installed hash/cypher in conjunction with the check hash.

11.2.2 CDB Format ID 2

NOTE: This CDB layout is obsolete; all new volumes should use the latest CDB format.

11.2.2.1 Overview

A FreeOTFE critical data block consists of "CDL" bits of data. The following table describes the high-level layout of a FreeOTFE CDB:



Color key:

Color	Encryption used
	Red items are not encrypted
	Blue items are encrypted with the user's chosen cypher together with a "critical data key" derived from the user's password, salt, and the user's chosen hash algorithm

Seem intimidating? Read on, and all will become clear... When broken down into its component parts, the CDB structure is reasonably straightforward to understand.

Note: Throughout this document, the following definitions apply:

Variable	Definition
CDL	Critical Data Length (in bits) This is defined as 4096 bits.
MML	Maximum MAC Length (in bits) This is defined as 512 bits.
sl	Salt length (in bits) This is the user specified salt length, as specified by the user when the CDB is created
cbs	Cypher Block Size (in bits) The block size of the cypher used to encrypt the volume
cks	Cypher Key Size (in bits) The key size of the cypher used to encrypt the volume. If the cypher accepts variable length keysizes, this is set to a user-specified value up to a maximum of 512.
ml	MAC length (in bits) This is the length of MAC generated

11.2.2.2 Breakdown: CDB layout

Password salt	Encrypted block	Random padding #1

Item name	Size (in bits)	Description
Password salt	sl (User specified to a max 512)	This data is used together with the user's password to derive the "critical data key". This key is then used to encrypt/decrypt the "Encrypted block".
Encrypted block	If cbs>8 then: ((CDL - sl) div cbs) * cbs If cbs<=8 then: (CDL - sl) This size is referred to as "leb"	This block contains the actual key which is used to encrypt/decrypt the encrypted partition image. See below for further breakdown.
Random padding #1	((CDL- sl) - leb)	Random "padding" data. Required to pad out any remaining, unused, bits in the "critical data block"
Total size:	CDL	

11.2.2.3 Breakdown: Encrypted block layout

Check MAC	Random padding #3	Volume details block

As described above, this entire block is encrypted using the user's password, salt, and chosen hash and cypher algorithms.

As this block is encrypted, its length (in bits) must be a multiple of the cypher's blocksize.

Item name	Size (in bits)	Description
Check MAC	ml Up to a maximum of MML bits	This is the MAC of the plaintext version of the "Volume details block". If hk is zero or undefined, then this hash will be either truncated to MML bits, or right-padded with 0 bits up to a maximum of MML bits
Random padding #3	MML - ml	Random "padding" data. Required to pad out the check MAC to a predetermined number of bits.
Volume details	leb - MML	This stores the details of how to encrypt/decrypt the encrypted partition.
Total size:	leb	

11.2.2.4 Breakdown: Volume details block layout

CDB format ID	Volume flags	Encrypted partition image length	Master key length	Master key	Requested drive letter	Volume IV length	Volume IV	Random padding #2

Finally, we reach the details that the critical data block was designed to protect. All of the items within this block have bit order: MSB first.

Item name	Size (in bits)	Description
CDB format ID	8	This is a version ID which identifies the layout of the remainder of the volume details block When this layout format is used, this will always be set to 2.
Volume flags	32	Bitmap flagging various items. Bit - Description 0 - Use different IVs for each sector 0 = Use NULL IV for all sectors 1 = Use sector ID (possibly hashed; see bit 3) as IV 1 - Sector ID zero is at the start of the file 0 = Sector ID zero is at the start of the encrypted data 1 = Sector ID zero is at the start of the host volume file 2 - (unused) 3 - Hash sector ID before use (only valid if bit 0 is set) 0 = Use sector ID as sector IV - do not hash before using 1 = Hash sector ID before using as sector IV

Encrypted partition image length	64	This stores the length of the encrypted partition image in bytes.
Master key length	32	This will be set to the length of the master key in bits.
Master key	*cks*	This is set to the random data generated when the volume was created; and is the en/decryption key used to encrypt the encrypted partition image
Requested drive letter	8	The drive letter the volume should be normally be mounted as. Set to 0x00 if there is no particular drive letter the volume should be mounted as (i.e. mount using the first available drive letter).
Volume IV length	32	This will be set to the length of the Volume IV in bits. If the cypher's blocksize is >= 0, this will be set to the cypher's blocksize. Otherwise, this will be set to 0.

Volume IV	If (cbs > 0), then: cbs
	
	If (cbs <= 0), then
	0
		This is set to the random data generated when the volume was created. When each sector of the encrypted partition is encrypted/decrypted, this value will be XORd with any (hashed or unhashed) sector ID before being used as the sector IV.
		This guarantees that every sector within the encrypted partition has a non-predictable IVs.

Random padding #2		Random "padding" data. Required to pad out the encrypted block to a multiple of bs, and to increase the size of this block to the maximum length that can fit within the "critical data block".
Total size:	(leb - MML)	

11.2.2.5 Miscellaneous Comments Regarding the CDB Layout

The design of the critical data layout eliminates the need for the cypher/hash used to be stored anywhere, denying an attacker this information and increasing the amount of work required to attack a volume file.

The "password salt" appears before the "encrypted block", and no indication of the length of salt used is stored anywhere in order to prevent an attacker from even knowing where the "encrypted block" starts within the CDB.

The "Check MAC" is limited to 512 bits. This is limited for practical reasons as some kind of limit is required if the critical data block is to be of a predetermined size. See section on mounting volume files for how multiple matching MACs are handled.

The "Password salt" is (fairly arbitrarily) limited to 512 bits. Again, this is primarily done for practical reasons.

Although at time of writing (March 2005) this limit to the length of salt used should be sufficient, the format of the critical data block (with included layout version ID) does allow future scope for modification in order to allow the critical data block to be extended (e.g. from 4096 to 8192 bits), should this limit be deemed inadequate..

The "Encrypted block" does contain a certain amount of data that may be reasonably guessed by an attacker (e.g. the CDB format ID), however this would be of very limited use to an attacker launching a "known plaintext" attack as the amount of this data is minimal, and as with pretty much any OTFE system the encrypted partition image can reasonably expected to contain significantly more known plaintext than the CDB anyway (e.g. the partition’s boot sector)

11.2.2.5.1 CDB Encryption

The encrypted data block within a CDB is encrypted using:

- A key derived from the user’s password
- A NULL IV

The key used for this encryption/decryption depends on the CDB format used to create the CDB.

For older (CDB format 1) volumes, the key is derived as follows:

1. The user’s password is appended to the salt bits
2. The result is hashed with the user’s choice of hash algorithm
3. If the cypher used has a fixed keysize, this hash value generated is truncated/right padded with NULLs until it is the same length as the cypher’s keysize

For newer (CDB format 2) volumes, the key is derived as follows:

1. The user’s password is passed through "n" iterations (where "n" is user specified) of PKCS#5 PBKDF2 using HMAC as the PRF, which is turn employs the user’s choice of hash algorithm. In doing so, the user’s password is supplied as the password to PBKDF2, and the salt bits are used as the PBKDF2 salt.

11.2.2.5.2 Check MAC

The manner in which the check bytes within a CDB are calculated depends on the CDB format used.

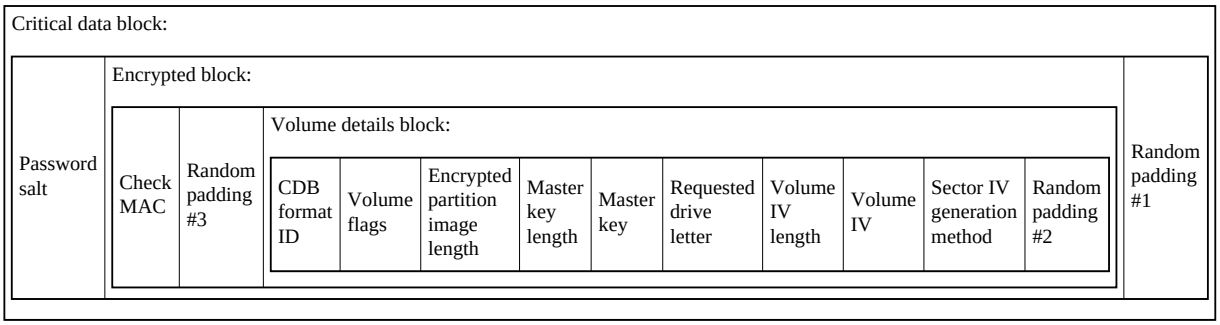
For older (CDB format 1) volumes, the check bytes are calculated by simply hashing the volume details block with the user’s choice of hash algorithm.

For newer (CDB format 2) volumes, the check bytes are calculated by passing the volume details block through HMAC with the user’s choice of hash algorithm. In doing so, the derived key used to encrypt/decrypt the CDB is used as the HMAC key.

11.2.3 CDB Format ID 3

11.2.3.1 Overview

A FreeOTFE critical data block consists of CDL bits of data. The following table describes the high-level layout of a FreeOTFE CDB:



Color key:

Color	Encryption used
	Red items are not encrypted
	Blue items are encrypted with the user's chosen cypher together with a "critical data key" generated using PKCS#5 PBKDF2 (with HMAC PRF) together with the user's password, salt, and chosen hash algorithm

Seem intimidating? Read on, and all will become clear... When broken down into its component parts, the CDB structure is reasonably straightforward to understand.

Note: Throughout this document, the following definitions apply:

Variable	Definition
CDL	Critical Data Length (in bits) This is defined as 4096 bits.
MML	Maximum MAC Length (in bits) This is defined as 512 bits.
sl	Salt length (in bits) This is the user specified salt length, as specified by the user when the CDB is created
cbs	Cypher Block Size (in bits) The block size of the cypher used to encrypt the volume
cks	Cypher Key Size (in bits) The key size of the cypher used to encrypt the volume. If the cypher accepts variable length keysizes, this is set to a user-specified value up to a maximum of 512.
ml	MAC length (in bits) This is the length of MAC generated

11.2.3.2 Breakdown: CDB layout

Password salt	Encrypted block	Random padding #1

Item name	Size (in bits)	Description
Password salt	sl (User specified to a max 512)	This data is used together with the user's password to derive the "critical data key". This key is then used to encrypt/decrypt the "Encrypted block".
Encrypted block	If cbs>8 then: ((CDL - sl) div cbs) * cbs If cbs<=8 then: (CDL - sl) This size is referred to as "leb"	This block contains the actual key which is used to encrypt/decrypt the encrypted partition image. See below for further breakdown.
Random padding #1	((CDL- sl) - leb)	Random "padding" data. Required to pad out any remaining, unused, bits in the "critical data block"
Total size:	CDL	

11.2.3.3 Breakdown: Encrypted block layout

Check MAC	Random padding #3	Volume details block

As described above, this entire block is encrypted using the user's password, salt, and chosen hash and cypher algorithms.

As this block is encrypted, its length (in bits) must be a multiple of the cypher's blocksize.

Item name	Size (in bits)	Description
Check MAC	ml Up to a maximum of MML bits	This is the MAC of the plaintext version of the "Volume details block". If hk is zero or undefined, then this hash will be either truncated to MML bits, or right-padded with 0 bits up to a maximum of MML bits
Random padding #3	MML - ml	Random "padding" data. Required to pad out the check MAC to a predetermined number of bits.
Volume details	leb - MML	This stores the details of how to encrypt/decrypt the encrypted partition.
Total size:	leb	

11.2.3.4 Breakdown: Volume details block layout

CDB format ID	Volume flags	Encrypted partition image length	Master key length	Master key	Requested drive letter	Volume IV length	Volume IV	Sector IV generation method	Random padding #2

Finally, we reach the details that the critical data block was designed to protect. All of the items within this block have bit order: MSB first.

Item name	Size (in bits)	Description
CDB format ID	8	This is a version ID which identifies the layout of the remainder of the volume details block When this layout format is used, this will always be set to 3. Later volume file layouts may have different items in this section, or the layout may change; in which case a different version ID will be used here.

Volume flags	32	Bitmap flagging various items. Bit - Description 0 - (unused) 1 - Sector ID zero is at the start of the file 0 = Sector ID zero is at the start of the encrypted data 1 = Sector ID zero is at the start of the host volume file/partition 2 - (unused) 3 - (unused) 4 - Volume file timestamps normal operation 0 = On dismount, volume file timestamps will be reset to the values they were when mounted 1 = On dismount, volume file timestamps will be left as-is (i.e. will indicate the date/time the volume was last written to) Note: This bit gets ignored by the GUI, which will operate it according to the user options set at the time the volume is mounted
Encrypted partition image length	64	This stores the length of the encrypted partition image in bytes.
Master key length	32	This will be set to the length of the master key in bits.
Master key	*cks*	This is set to the random data generated when the volume was created; and is the en/decryption key used to encrypt the encrypted partition image
Requested drive letter	8	The drive letter the volume should be normally be mounted as. Set to 0x00 if there is no particular drive letter the volume should be mounted as (i.e. mount using the first available drive letter).
Volume IV length	32	This will be set to the length of the Volume IV in bits. If the cypher's blocksize is >= 0, this will be set to the cypher's blocksize. Otherwise, this will be set to 0.

Volume IV	If (cbs > 0), then: cbs
	
	If (cbs <= 0), then 0
		This is set to the random data generated when the volume was created. When each sector of the encrypted partition is encrypted/decrypted, this value will be XORd with any (hashed or unhashed) sector ID before being used as the sector IV.
		This guarantees that every sector within the encrypted partition has a non-predictable IVs.

Sector IV generation method	8	This is set to indicate the method of generating sector IVs. Note that if a volume IV is present, then it will be XORd with the IV generated using this method, before it is used for encryption/decryption. In all cases, the sector IV generated will be right-padded/truncated to the cypher's blocksize. If the cypher's blocksize is <= 0, then this must be set to 0. 0 - No sector IVs (Null sector IV) 1 - Sector IV is the 32 bit sector ID (LSB first) 2 - Sector IV is the 64 bit sector ID (LSB first) 3 - Hash of the 32 bit sector ID (sector ID is LSB first) 4 - Hash of the 64 bit sector ID (sector ID is LSB first) 5 - ESSIV The "Volume flags" item is used to determine the location of sector zero (start of encrypted data, or start of host file/partition)
Random padding #2		Random "padding" data. Required to pad out the encrypted block to a multiple of bs, and to increase the size of this block to the maximum length that can fit within the "critical data block".
Total size:	(leb - MML)	

11.2.3.5 Miscellaneous Comments Regarding the CDB Layout

The design of the critical data layout eliminates the need for the cypher/hash used to be stored anywhere, denying an attacker this information and increasing the amount of work required to attack a volume file.

The "password salt" appears before the "encrypted block", and no indication of the length of salt used is stored anywhere in order to prevent an attacker from even knowing where the "encrypted block" starts within the CDB.

The "Check MAC" is limited to 512 bits. This is limited for practical reasons as some kind of limit is required if the critical data block is to be of a predetermined size. See section on mounting volume files for how multiple matching MACs are handled.

The "Password salt" is (fairly arbitrarily) limited to 512 bits. Again, this is primarily done for practical reasons.

Although at time of writing (March 2005) this limit to the length of salt used should be sufficient, the format of the critical data block (with included layout version ID) does allow future scope for modification in order to allow the critical data block to be extended (e.g. from 4096 to 8192 bits), should this limit be deemed inadequate..

The "Encrypted block" does contain a certain amount of data that may be reasonably guessed by an attacker (e.g. the CDB format ID), however this would be of very limited use to an attacker launching a "known plaintext" attack as the amount of this data is minimal, and as with pretty much any OTFE system the encrypted partition image can reasonably expected to contain significantly more known plaintext than the CDB anyway (e.g. the partition's boot sector)

11.2.3.5.1 CDB Encryption

The encrypted data block within a CDB is encrypted using:

- A key derived from the user's password
- A NULL IV

The key used for this encryption/decryption depends on the CDB format used to create the CDB.

For older (CDB format 1) volumes, the key is derived as follows:

1. The user's password is appended to the salt bits
2. The result is hashed with the user's choice of hash algorithm
3. If the cypher used has a fixed keysize, this hash value generated is truncated/right padded with NULLs until it is the same length as the cypher's keysize

For newer (CDB format 2) volumes, the key is derived as follows:

1. The user's password is passed through "n" iterations (where "n" is user specified) of PKCS#5 PBKDF2 using HMAC as the PRF, which is turn employs the user's choice of hash algorithm. In doing so, the user's password is supplied as the password to PBKDF2, and the salt bits are used as the PBKDF2 salt.

11.2.3.5.2 Check MAC

The manner in which the check bytes within a CDB are calculated depends on the CDB format used.

For older (CDB format 1) volumes, the check bytes are calculated by simply hashing the volume details block with the user's choice of hash algorithm.

For newer (CDB format 2) volumes, the check bytes are calculated by passing the volume details block through HMAC with the user's choice of hash algorithm. In doing so, the derived key used to encrypt/decrypt the CDB is used as the HMAC key.

11.2.4 CDB Format ID 4

CDB layout format 4 is identical to CDB layout format 3, with the one exception that the Master key stored within the Volume details block may include multiple keys, one after the other

For example:

- An LRW encrypted volume's "Master key" will be the encryption key, followed immediately by the tweak key.
- An XTS encrypted volume's "Master key" will be the two XTS encryption keys, one after the other

In all cases, the Master key length will indicate the total length of all keys. It is the responsibility of the cypher driver to determine how the key material is used.

FreeOTFE v3.0 (and later) and FreeOTFE4PDA v3.0 (and later) create volumes using this layout.

11.3 Creating FreeOTFE Volumes

To create a FreeOTFE volume file, a fairly significant amount of information is required due to freedom that FreeOTFE gives you in creating volume files.

Broadly speaking, creating a FreeOTFE volume consists of three distinct stages:

1. Creating a file large enough on the local filesystem to store the encrypted partition image (and CDB, if included as part of the volume).
2. Writing a CDB either to the volume file or a separate keyfile, depending on the user's choice.
3. Mounting the volume, formatting it, and "shredding" (overwriting) all free space.

Stage 1 is straightforward; write data to the file until is has gained the required size. This stage is skipped in the case of creating a hidden volume or volume based on a partition.
Stage 2 is more complex; and is described below.
Stage 3 is required in set the volume up for use, and increase security. This is largely a manual process carried out by the user, depending on their needs.

11.3.1 Writing the CDB/keyfile

The following procedure is used to build up a FreeOTFE CDB/keyfile:

1. Obtain all the information which will be stored within the volume's "Volume details block"
2. Derive the "critical data key" by processing the user's password and salt with PKCS #5 PBKDF2 (using HMAC with the user's choice of hash algorithm).
   The derived key should be ks bits long (i.e. the cypher's keysize). If ks is undefined, then 512 bits will be used. (Note: In this case, the keysize used for encrypting/decrypting the encrypted partition image must be specified by the user. The keysize for the critical data block is fixed at 512 bits if ks is undefined in order to simplify the "mount volume" dialog, and to reduce the potential for user confusion as most cyphers have a fixed ks, and asking users for this information may cause them to think this is more information they have to memorise, which it wouldn't be)
3. Create the plaintext version of the "Volume details block" in memory, including padding the end with random data as appropriate
4. Calculate the check MAC using HMAC, together with the derived key and user's choice of hash algorithm
5. Truncate the MAC to 512 bits if it is longer, or right-pad to 512 bits with random data to if less.
6. Prepend the check MAC (and any random data appended to it) onto the beginning of the volume details block to form a plaintext version of the "Encrypted block"
7. Encrypt the plaintext "Encrypted block" using a null IV and the critical data key.
8. Prepend the salt bytes onto the end of the "Encrypted block", and pad out the end with random data to form the complete CDB
9. Write the CDB to either:
   - The start of the user's volume
   - A keyfile
   - The user specified offset within the host volume, if creating a "hidden volume" which includes a CDB

11.4 Mounting FreeOTFE Volumes

To mount a FreeOTFE volume, the following information must be obtained from the user:

1. The volume to be mounted
2. The user's password
3. If the CDB to be used for mounting is stored in a keyfile, the location of the keyfile
4. The length of salt used in the CDB (sl)
5. The number of key iterations
6. When mounting a hidden volume, the following information is also required:
7. The offset within the host file/volume that the hidden volume resides at
8. If the offset specified gives the location of a CDB, or the "Encrypted partition image"

Although this list may sound as though it places a significant burden on the user to remember a significant amount of information. However, for most users the only information required will be the volume the wish to mount, and their password; all of the other details may be defaulted to sensible values unless the user chooses to take advantage of the more advanced features of FreeOTFE.

From the information supplied by the user, the following can be determined automatically:

1. The cypher used
2. The hash algorithm used
3. All of the details stored within "Volume details block" of the CDB used for mounting

The first two items are determined by brute force; by decrypting the CDB with every possible combination of hash/cypher installed on the user's system until the CDB's "Check MAC" is decrypted successfully.

In detail, the following procedure is used:

1. The information listed above is obtained from the user.
2. The 512 bytes (4096 bits) CDB is read in.
   This will be taken from one of:
   1. The start of the volume being mounted (provided the volume includes a CDB)
   2. A host file, starting from a user specified offset, if mounting a hidden volume (provided the hidden volume includes a CDB)
   3. A keyfile relating to the volume being mounted
3. The first "sl" bits are stripped off the CDB, and are assumed to be the salt
4. For each hash algorithm installed on the user's computer:
   1. For each cypher algorithm installed on the user's computer:
      1. A "critical data key" is derived by processing the user's password and salt with PKCS #5 PBKDF2. The PRF used with PBKDF2 will be HMAC, with the current hash algorithm.
         The derived key should be ks bits long (i.e. the cypher's keysize). If ks is undefined, then 512 bits will be used.
      2. The number of "Random padding #1" bytes is determined, based on the cypher's blocksize. This random padding is then stripped off the CDB to give an "Encrypted block".
      3. The "Encrypted block" is decrypted using the "critical data key" generated above and

the current cypher.

4. The first 512 bytes of the decrypted "Encrypted block" are split off and retained as a plaintext check MAC and "Random padding #3".
5. The remainder of the decrypted "Encrypted block" is assumed to be a "Volume details block". The HMAC of this "Volume details block" is generated, using the current hash algorithm and the "critical data key" just used for decryption.
6. The generated MAC is truncated to its first 512 bytes, if it's longer than this.
7. The resulting "n" bit MAC is compared with the first "n" bits of the "Check MAC"/"Random padding #3"
8. If the MAC is identical to the "Check MAC", it is assumed that a valid cypher/hash combination has been found - the "Volume details block" is parsed and stored together with the hash/cypher combination.
9. Regardless of whether the generated MAC is identical to the "Check MAC", all remaining hash/cypher combinations will be processed in the same manner, to check if there are any other matching hash/cypher combinations.

After all possible hash/cypher combinations have been exhausted:

- If no cypher/hash combination successfully decrypted the "Volume details block", the user will be informed that they have entered incorrect details.
- If more than one cypher/hash combination successfully decrypted the "Volume details block", the user should be prompted to choose which of the combinations they wish to use. The system will proceed as if only the user selected combination had been successful in decrypting.
- If exactly one cypher/hash combination successfully decrypted the "Volume details block", then the "Encrypted partition image" will be decrypted as necessary using this cypher/hash combination, together with the information obtained from the decrypted "Volume details block".

As a result of the CDB layout, it is not necessary to store either the cypher or hash algorithm used for en/decryption anywhere. Nor is it necessary to prompt the user for this information as this information can be determined dynamically by attempting the mount using all possible combinations of installed hash/cypher and testing if the same check MAC can be generated using the decrypted data.

11.4.1 Additional information

The driver API supports passing volume "metadata" to the driver when mounting a volume.
Information passed in this way to the driver is simply stored by the driver against the mounted volume, and is not processed in any way.
When a user application calls DeviceIOControl with IOCTL_FREEOTFE_GET_DISK_DEVICE_STATUS, the number of bytes of volume metadata is returned. By calling with IOCTL_FREEOTFE_GET_DISK_DEVICE_METADATA, the actual data that was stored when mounting can be retrieved.
This is intended for future development; to enable user applications to store information against volumes they mount (e.g. "This is a Linux volume", "This is a FreeOTFE volume"), which can later be retrieved for display to the user.
Volume metadata passed to the driver in this way should be kept to the absolute minimum possible.

11.5 Encrypted Partition Image Encryption/Decryption

The encrypted partition that forms the bulk of an encrypted volume is encrypted on a 512 byte, sector-by-sector basis using:

- The master key stored within the volume's CDB (or keyfile)
- A per-sector IV (provided that the cypher used has a fixed blocksize, greater than zero)

11.5.1 Per-Sector IV Generation

The manner in which per-sector IVs are generated depends on the IV generation method the user selected when creating the volume:

IV Generation Method	Description
Null IV	No IV is used/a null IV is used. i.e. A block of data consisting of with 0x00 (null) characters is used as the IV
32 bit sector ID	The least significant 32 bits of the sector ID is right-padded with 0x00 characters, and used as the IV. These bits are ordered MSB..LSB.
64 bit sector ID	As the 32-bit sector ID method, but a 64 bits sector ID is used. Note: This is unlikely to offer any security advantage over using 32 bit sector IDs, unless used with a volume file ((2^32) * 512) bytes long (2048GB), or greater
Hashed 32 bit sector ID	The least significant 32 bits of the sector ID is hashed with the user's choice of hash algorithm. The resulting hash value will be truncated/right padded with 0x00 characters until it is the same length as the cypher's blocksize.
Hashed 64 bit sector ID	As the hashed 32-bit sector ID method, but a 64 bits sector ID is used. Note: This is unlikely to offer any security advantage over using 32 bit sector IDs, unless used with a volume file ((2^32) * 512) bytes long (2048GB), or greater
ESSIV	This option offers the most security. On mounting the FreeOTFE volume, the master key used for encrypting/decrypting the volume is hashed with the hash algorithm chosen by the user when the volume was created. If the cypher used for encryption/decryption has a fixed keysize, this hash output is truncated/right padded with 0x00 characters until it matches the cypher's keysize and stored as the "ESSIV key" (or "salt"). If the cypher doesn't have a fixed keysize, the full hash output is stored as this key ("salt"). When a per-sector IV is required, the 64 bit sector ID is encrypted using the "ESSIV key". This encrypted sector ID is truncated/right-padded with 0x00 characters until it matches the cypher's blocksize.

In all cases, the sector ID is calculated as:

Sector ID = (Ostart - Soffset) % Ssize

where:

Ostart	The offset within the host volume/partition from where the encrypted partition begins (i.e. after any CDB)
Soffset	The offset from within the encrypted partition from where the sector begins
Ssize	The sector size of the emulated drive (i.e. 512 bytes)
%	is the modulus operator

Putting it another way, this is the sector ID (starting from zero) of the partition as it appears to the host OS after mounting.

If the user opted to additionally use per-volume IVs when the volume was created, IVs generated using the method selected by the user when the volume was created are XORd with a "per-volume" IV. This "per-volume" IV consists of a block of data equal to the blocklength of the cypher used to encrypt the volume and consists of random data generated when the volume was created, and stored within the volume's CDB (or keyfile).

11.6 Registry Entries

The PC version of FreeOTFE doesn't create any registry entries for itself unless the user chooses to associate ".vol" files with the application, in which case only those registry entries which are required to associate the FreeOTFE executable with the filename extension are created. All user options and settings are stored in a ".ini" file located in the FreeOTFE directory.

In addition to this, MS Windows does create a registry entry for each FreeOTFE driver used. This is inevitable; all OTFE systems running under MS Windows are required to do this in order to function correctly.

The following detail the registry entries are typically created by MS Windows:

Registry key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<driver name>

These keys may have the following values under them:

Value	Type	Meaning
ErrorControl	DWORD	0x00000001 - Normal error handling
Start	DWORD	0x00000001 - Driver started at system 0x00000003 - Driver started manually
Type	DWORD	0x00000001
Portable	DWORD	This value is optional, but set to 0x00000001 if present. This value flags that the relevant driver was installed in "portable mode", and should be removed

The PDA version operates in a similar manner, though no registry entries are created automatically by the OS for the drivers used. Instead, FreeOTFE4PDA is required to create a sequence of registry entries when a volume is mounted. These registry entries are deleted once the volume is dismounted.

11.7 Portable Mode Impact

Although no files are copied to your computers hard drive when using portable mode, because part of the manner in which MS Windows manages device drivers, Windows still writes certain details about the portable mode drivers to the registry. Specifically, the full path and filename of the drivers used together with other basic information on the drivers as detailed above.

When portable mode is stopped, most of this information is deleted by Windows automatically. However:

1. Because of the way in which the registry stores data, an attacker may be able to recover that information which has been deleted (this is analogous to deleting a file on your filesystem; although its directory entry may have been marked as "deleted", the data may still be recoverable)
2. When Windows deletes its registry entries, it doesn't delete all of them (e.g. HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_...)

It is not possible to securely delete the relevant registry entries without "going behind Windows' back" - not exactly recommended when working with kernel mode device drivers!

*It should be noted that this applies equally to **all** disk encryption systems that support any kind of "portable mode".*

Should it be a concern that an attacker may discover which FreeOTFE drivers were being used, it is suggested that you either:

1. Change the filenames of the FreeOTFE drivers you will be using in portable mode (e.g. rename "FreeOTFECypherAES.sys" to "FreeOTFECypherTwofish.sys"). This will cause the data written to the registry to reflect this new filename, hopefully convincing an attacker that the driver used was a different one.
2. If you only use one cypher and hash driver in portable mode, store a number (or all) of the other cypher/hash drivers with your "portable" version of FreeOTFE. Even if an attacker can determine which drivers you were using in portable mode, that attacker will not be able to determine which of the portable drivers you were actually using to encrypt/decrypt your data with.

11.8 Random Number Generators (RNGs)

FreeOTFE offers a choice of four different random number generators (RNGs) for use when creating new FreeOTFE volumes:

1. Microsoft CryptoAPI
2. Mouse movement
3. cryptlib
4. PKCS#11 tokens

Whichever one is selected must produce 4096 bits (512 bytes) of cryptographically secure random numbers. This random data is used in three ways:

1. As the master key used for encrypting/decrypting your data
2. For salting
3. As random "padding" to make up otherwise unused space within the FreeOTFE volume's critical data block. (See volume layout documentation for further details)

If more than one RNG is selected, their output will be combined (XOR'd together) and the resulting data used. In this way, the random data generated will never be weaker than the strongest selected RNG.

11.8.1 Microsoft CryptoAPI

The Microsoft CryptoAPI is used to generate random data.

11.8.2 Mouse Movement

This relies on the user "waggling" the mouse in a random fashion to generate random data.

Every 100ms the mouse pointer is checked. If it has moved significantly, then the X and Y coordinates of the mouse pointer are sampled, and the LSB of each is added to the random data collected.

Due to the volume of random data required, and the fact that only 2 bits of random data are collected for each mouse position sampled, this is a relatively slow process.

11.8.3 cryptlib

cryptlib is used to generate random data.

Note: This option is only available if cryptlib (cl32.dll) is installed; see the cryptlib WWW site for further details and download.

11.8.4 PKCS#11 Tokens

If you have a security token or smartcard, this may be used as a RNG.

See the Security Token/Smartcard Support section for more information on setting up and using PKCS#11 tokens.

11.9 Building the Software

FreeOTFE/FreeOTFE4PDA/FreeOTFE Explorer come in a number parts:

- FreeOTFE:
  1. A front-end GUI, written in Delphi
  2. A number of kernel drivers, written in C
- FreeOTFE4PDA:
  1. A front-end GUI, written in C
  2. A number of drivers, written in C
- FreeOTFE Explorer:

1. A front-end GUI, written in Delphi
2. The same drivers as FreeOTFE4PDA, but built for Win32

- A number of command line decryption utilities, also written in C.

11.9.1 FreeOTFE

11.9.1.1 Building the GUI

This is a description for Delphi newbies of the basic steps involved in compiling the FreeOTFE GUI.

To build the GUI, the following software is required:

- Delphi (CodeGear Delphi 2007 or later, though Delphi 2006 should just as well. Delphi v5 - v7 can probably be used with minimal changes, though wouldn't look as nice under Windows Vista)
- The SDeanComponents package (v2.00.00 or later)
- (Optional) GNU gettext for Delphi (dxgettext), available (free) from: http://dybdahl.dk/dxgettext/ (This package adds support for language translations)

The binary release of this software was built with CodeGear Delphi 2007.

1. With each of the packages in the SDeanComponents archive,
   1. Build each package
   2. Install each package
   3. Ensure that the correct path to each package is added to your Delphi environment ("Tools | Environment Options...", "Library" tab)
2. Add the path to the modified Delphi files included in SDeanComponents to fix various bugs relating to Delphi 2006's Windows Vista support to the top of Delphi's standard library paths. (This step probably won't be needed with later versions of Delphi, and shouldn't be carried out with older versions of Delphi, which will have different source)
3. Open the FreeOTFE project ("FreeOTFE.dpr")
4. If you have the dxgettext software installed (see above), ensure that the compiler directive "_DXGETTEXT" is set. Otherwise, make sure that this compiler directive is *not* set.
5. Build the application.
6. You should now find a file called "FreeOTFE.exe" in the directory above the "src" directory

You have now successfully built the GUI frontend!

If required, the compiler definition "FREEOTFE_TIME_CDB_DUMP" may be set, in which case the time taken to dump a CDB ("Tools | Critical data block | Dump to human readable file...") will be shown after the dump completes.

11.9.1.2 Building the Kernel Drivers

The kernel mode drivers implement the actual hash, encryption/decryption and main FreeOTFE drivers.

To build these drivers, the following software is required:

- Microsoft Visual Studio 2008 (older versions may well be used, changing "vcvarsall" to "vcvars32", and similar changes)
  - If using an older version of MS Visual Studio, the MS Windows SDK (February 2003

version) is also needed

- The MS Windows WDK (WDK for Server 2008 v6001.18001)

The binary release of this software was built with Microsoft Visual Studio 2005 Professional Edition.

At time of writing, the MS Windows SDK can be downloaded from the Microsoft WWW site. The MS Windows DDK is not available as a download, but can be ordered from the Microsoft WWW site as a free CD, for the cost of delivery.

It should be noted that if you are unable to source the exact versions listed above, earlier versions may well be substituted, although I cannot guarantee success. Later versions should operate correctly. The above list describes the development environment as used to build the binary release of FreeOTFE.

11.9.1.2.1 Setting up the Build Environment

11.9.1.2.1.1 Installation and Configuration of MS Build Environment

The following list comprehensively describes the configuration used to build the binary release of FreeOTFE. Feel free to adjust according to taste - a number of the options listed are not necessary, and are only included for completeness...

1. Install VC++
2. Put a copy of "vcvarsall.bat" into one of the directories in your path
3. Configure the VC++ editor:
   - To use spaces, not tabs
   - To indent braces
4. Install the MS Windows SDK with the following options:
   - Install in C:\MSSDK
   - Install the "Core SDK"
   - Then install the debugging tools for windows
   - Do not register environment variables (we'll use "Setenv.bat" from the command line)
5. Install the MS Windows DDK with the following options:
   - Install in C:\WINDDK\3790
   - Include the "Illustrative Driver Samples"
   - Include the "Input Samples"
   - Include the "Storage Samples"
   - Include the "Virtual Device Driver Samples"
   - Include the "WDM Samples"
   - Build Environment\Windows Driver Development Kit AMD64 Additional Build Tools
   - Build Environment\Windows Server 2003 AMD64 Libraries
   - Build Environment\Windows XP Headers
   - Build Environment\Windows XP x86 Libraries
   - Build Environment\Windows XP IA86 Libraries
     - Needed if the build .BAT files (see later) use "chk WXP" - if that's skipped it'll default to WNET (windows .NET)
   - Build Environment\Windows 2000 Headers
   - Build Environment\Windows 2000 Build Environment
     - Needed if the build .BAT files (see later) use "chk <something>" - if that's skipped it'll default to WNET (windows .net)

11.9.1.2.1.2 FreeOTFE Build Configuration

1. Edit "setup_env_common.bat" (located under src\drivers\Common\bin), and ensure that the following variables are set appropriately:

Variable	Description	Default value
FREEOTFE_DEBUG	Build type flag; set to 1 for debug build, or 0 for release	0
FREEOTFE_TARGET	Target OS to build for; e.g. WXP/W2K/WNET; note that W2K builds will not operate correctly under Windows XP (e.g. when formatting a volume)	WXP
PROJECT_DRIVE	The drive on which you have stored the FreeOTFE source	E:
PROJECT_DIR	The full drive and path where the "drivers" directory is located	<see file>
MSSDK_DIR	The directory in which you installed the MS SDK	C:\MSSDK
MSDDK_DIR	The directory in which you installed the MS DDK	C:\WINDDK\3790

2. Edit "setup_env_driver.bat" (in the same directory), and ensure that "SETENV.BAT" is called with the parameters appropriate to the type of build you wish to create, and that "FREEOTFE_OUTPUT_DIR" is set to the appropriate directory under the source directories where the build executable places the files it creates (this shouldn't be needed as it will happen automatically if the above are configured correctly)

11.9.1.2.1.3 3rd Party Source Code

Some of the FreeOTFE drivers (the hash/encryptions drivers in particular) are dependant on certain 3rd party software being installed. FreeOTFE's source code comes complete with 3rd party included in the"src\3rd_party" directory and should be preconfigured, ready for use.

Alternatively, you may wish to download this 3rd party source from the original authors in order to verify the integrity of this software. For this reason, details of where this software was obtained from are included in the above directory.

Please note that should choose the latter option, it is important that you review the individual driver notes (see separate driver directories; "_notes.txt" files) to ensure that this software is configured correctly. Additionally, you may well have to modify the "my_build_sys.bat" files, directing them to the location where you installed said 3rd party source code, as the build process requires that certain files are copied over into the FreeOTFE src directories. (Annoying, but this is a requirement of the MS "build.exe" command)

The LibTomCrypt source in particular had minor configuration changes to tomcrypt_cfg.h and tomcrypt_custom.h; please compare the original source (a copy of its release ZIP file is stored under src\3rd_party\libtomcrypt) with the modified version (uncompressed in a directory under this one)

11.9.1.2.2 Building the FreeOTFE Drivers

Either:

1. Open "FreeOTFE.sln" using Visual C++
2. Rightclick on each project in turn, and select "Build"

or:

1. Enter each of the separate driver directories in turn and launch each project's "my_build_sys.bat"

In either case, a copy of the binary which is built will be copied into the directory above your "src" directory.

After reaching this stage, you should have successfully built your own version of the FreeOTFE drivers!

Notes:

1. If FREEOTFE_TARGET is set to W2K, the resulting binary may not operate correctly under MS Windows XP as a number of functions what are only needed under Windows XP and later are #ifdef'd out. As a result, a "W2K" binary may not operate correctly under Windows XP (e.g. trying to format a volume may result in... Nothing happening). If you want a binary which will operate under both Windows 2000 and Windows XP, set this to WXP.
2. Windows XP migrated a couple of the previous Windows 2000 macros to be functions. In order to allow the above "WXP" builds to work under Windows 2000, "IFSRelated.h" includes a copy of these macros, and uses them regardless - see comments in code for an explanation.

11.9.2 FreeOTFE4PDA

To build FreeOTFE4PDA (both the drivers and GUI):

1. Open "FreeOTFE4PDA.sln" using Visual C++
2. Set the build configuration within Visual C++ to "Release" - "Pocket PC 2003 (ARM4)"
3. Rightclick on each project in turn, and select "Rebuild"

A copy of the binary which is built will be copied into the directory above your "src" directory.

11.9.3 FreeOTFE Explorer

11.9.3.1 Building the GUI

This is a description for Delphi newbies of the basic steps involved in compiling the FreeOTFE Explorer GUI.

To build the GUI, the following software is required:

- Delphi (CodeGear Delphi 2007 or later, though Delphi 2006 should just as well. Delphi v5 - v7 can probably be used with minimal changes, though wouldn't look as nice under Windows Vista)
- The SDeanComponents package (v2.00.00 or later)
- (Optional) GNU gettext for Delphi (dxgettext), available (free) from: http://dybdahl.dk/dxgettext/ (This package adds support for language translations)

The binary release of this software was built with CodeGear Delphi 2007.

1. With each of the packages in the SDeanComponents archive,
    1. Build each package
    2. Install each package
    3. Ensure that the correct path to each package is added to your Delphi environment ("Tools | Environment Options...", "Library" tab)
2. Add the path to the modified Delphi files included in SDeanComponents to fix various bugs relating to Delphi 2006's Windows Vista support to the top of Delphi's standard library paths. (This step probably won't be needed with later versions of Delphi, and shouldn't be carried out with older versions of Delphi, which will have different source)
3. Open the FreeOTFE Explorer project ("FreeOTFEExplorer.dpr")
4. If you have the dxgettext software installed (see above), ensure that the compiler directive "_DXGETTEXT" is set. Otherwise, make sure that this compiler directive is *not* set.
5. Build the application.
6. You should now find a file called "FreeOTFEExplorer.exe" in the directory above the "src" directory

You have now successfully built the GUI frontend!

11.9.3.2 Building the DLL drivers

To build the DLLs used by FreeOTFE Explorer:

1. Open "FreeOTFE4PDA.sln" using Visual C++
2. Set the build configuration within Visual C++ to "Release" - "Win32"
3. Rightclick on each project in turn, and select "Rebuild". Note: Don't bother building the "GUI" project; at present, this can only be built for the Windows Mobile platform.

A copy of the binary which is built will be copied into the directory above your "src" directory.

11.9.4 Building the Command Line Decryption Utilities

Note: The development of the command line decryption utilities has ceased. This functionality has been superceded with the development of FreeOTFE Explorer

To build the command line decryption utilities, the following software is required:

- A C compiler (Visual C++ .NET was used to write and test this software)

Please follow the following steps:

1. Install and configure up the build environment, as described as per building the backend drivers, you may omit the SDK and DDK.
2. Modify the software as appropriate for your test
    - Please see the command line decryption utility documentation
3. Launch the relevant "my_build_exe.bat" file

The executable should be built in the same directory.

11.9.5 Signing the Binaries

To sign the FreeOTFE binary files (.exe, .dll and .sys files), the procedure is pretty much as described at: Pantaray Research WWW site

At present, FreeOTFE is signed using a self-signed certificate; the full procedure used is as follows:

1. Install Visual Studio
2. From a command prompt, run "vcvarsall" (all commands detailed below should be executed from this command prompt)
3. Create a private certificate:

   ```
   makecert.exe -sv sdean12.pvk -n "E=sdean12@sdean12.org,CN=Sarah Dean" sdean12.cer
   ```

   this should create two files: sdean12.pvk and sdean12.cer
4. Create a test software publisher certificate (SPC):

   ```
   cert2spc.exe sdean12.cer sdean12.spc
   ```

   to create sdean12.spc. (This file would normally be supplied by a CA, if purchased)
5. Create a personal information file:

   ```
   pvk2pfx -pvk sdean12.pvk -spc sdean12.spc -pfx sdean12.pfx -f /pi <pvk password> /po <pfx password>
   ```

   Where:
   - <*pvk password*> is the password used when generating the .pvk file with makecert.txt
   - <*pfx password*> is the password you wish to use for securing the new .pfx file
6. Sign each of binary files:

   ```
   signtool.exe sign /f sdean12.pfx /p <pfx password> /v /t http://timestamp.verisign.com/scripts/timstamp.dll <filename>
   ```

   Where:
   - <*pfx password*> is the password used when generating the .pfx file with pvk2pfx

   The URL specified is a time stamping service (Verisign's in this case).

11.9.6 Additional Notes

When building the C code, FreeOTFEPlatform.h automatically #defines one of the following:

- FreeOTFE_PC_DRIVER
- FreeOTFE_PC_DLL
- FreeOTFE_PDA

depending on what is being built.

This header file should be #included at the start of *every* file which uses any of these defines. (Yes, this is obvious - but easily overlooked!)

11.10 Creating a New Hash/Cypher Driver

These instructions specify in general terms, the steps in taking an existing cypher driver (Blowfish, in this case), and modifying it to create a new cypher driver. The procedure for creating a new hash driver is practically identical, but one of the existing hash drivers should be used as a base, instead of the Blowfish cypher.

1. Create a new directory to contain your new cypher driver
   - Make a copy of the directory containing the Blowfish cypher driver ("CYPHER_BLOWFISH")
   - Rename this directory to reflect the new cypher's name
2. Modify "my_build_sys.bat"
   - Edit this file using a text editor to change:
     - All instances of "Blowfish" to reflect new cypher's name
     - The files needed
3. Modify "CYPHER_BLOWFISH.vcproj"
   - Rename this file to reflect the new cypher's name
   - Edit this file using a text editor to change:
     - All instances of "Blowfish" to reflect new cypher's name
     - What files are needed, as per my_build_sys.bat
4. Modify "src/sources"
   - Edit this file using a text editor to change:
     - All instances of "Blowfish" to reflect new cypher's name
     - What files are needed, as per my_build_sys.bat
5. In the "src" directory, rename all the "*Blowfish*" files to reflect the new cypher's name
6. Start VC++ and load the "FreeOTFE.sln" Visual Studio Solution.
7. Add the "....vcproj" project file you modified above into the solution.
8. Within VC++, modify the new FreeOTFECypherXXX.rc file to change all instances of "Blowfish" to reflect new cypher's name
9. Within VC++, modify FreeOTFECypherXXX.h
   - Change:
     - All instances of "Blowfish" to reflect new cypher name
     - Add/remove any "DRIVER_CIPHER_TITLE_XXX" entries as required
     - All of the GUID values (this is important! The FreeOTFE GUI uses these values to differentiate between the different cypher implementations)
     - Set the definition of "CYPHERS_SUPPORTED" to reflect the number of different cyphers the driver will provide (i.e. the number of "DRIVER_CIPHER_TITLE_XXX" definitions you have)
10. Within VC++, modify FreeOTFECypherXXX.c
    - Change:
      - The cipher descriptions returned
      - The initialization and encrypt/decrypt routines to check for, and use the correct cypher
11. Modify "clean.bat", in the top level "src" directory to clean up any object, garbage, etc files that are created when your driver is built
12. You should now be able to build your new cypher driver, which may then be installed as per any other cypher or hash driver.

11.10.1 Hash Length/Blocksize

When your hash driver is asked for details of the hash algorithms it provides, the hash length returned for each hash algorithm must be one of the following:

Hash length	Validity	Meaning	Comments
0	Valid	Hash values returned are 0 bits long.	Hashes which return zero length hash values cannot be used with FreeOTFE volumes. (FreeOTFE volumes use PKCS#5 PBKDF2 (HMAC), which requires that the length of hash values returned is greater than zero.)
Less than 0	Valid	Hash values returned are of variable length (e.g. the "NULL" hash, which returns its input as the generated hash value.)	Hashes which return variable length hash values cannot be used with FreeOTFE volumes. (FreeOTFE volumes use PKCS#5 PBKDF2 (HMAC), which requires that the length of the hash values used is fixed.)
Greater than 0	Valid	Hash values returned have a fixed, defined length (e.g. SHA-512's hash length is 512 bits)	Must be a multiple of 8.

When your hash driver is asked for details of the hash algorithms it provides, the blocksize returned for each hash algorithm must be one of the following:

Hash blocksize	Validity	Meaning	Comments
0	Valid	Hash algorithm does not process input data.	Hashes may only have a blocksize of 0 bits if the length of the hash values they output is also 0 bits long, or if they ignore their input. Hashes which use zero length blocksizes cannot be used for FreeOTFE volumes. (FreeOTFE volumes use HMAC, which requires that the blocksize of hashes used is greater than zero.)
Less than 0	Valid	Hash algorithm processes input data in variable-length blocks.	Hashes which use variable length blocksizes cannot be used for FreeOTFE volumes. (FreeOTFE volumes use HMAC, which requires that the blocksize of hashes is a fixed size.)
Greater than 0	Valid	Hash algorithm processes input data in defined, fixed blocks (e.g. SHA-512's block size is 1024 bits)	Must be a multiple of 8.

11.10.2 Cypher Keysize/Blocksize

When your hash driver is asked for details of the cyphers it provides, the keysize returned for each cypher must be one of the following:

Cypher keysize	Validity	Meaning	Comments
0	Valid	No key is used during encryption (e.g. if the cypher doesn't encrypt data, just returns the plaintext as the cyphertext; or if the cypher uses a hardcoded key)	
Less than 0	Valid	The cypher takes variable keylengths (e.g. the "XOR" cypher)	
Greater than 0	Valid	The cypher accepts only a specific keysize (e.g. full-strength DES only accepts 64 bit keys)	Must be a multiple of 8.

When your hash driver is asked for details of the cyphers it provides, the blocksize returned for each cypher must be one of the following:

Cypher blocksize	Validity	Meaning	Comments
0	Valid	Cypher does not process input data. (e.g. the "NULL" cypher, which just returns the supplied plaintext as cyphertext)	If the blocksize is 0, then no IVs will be used for encrypting/decrypting.
Less than 0	Valid	Cypher processes input data in variable-length blocks. (e.g. XOR processes data in blocks with the same length as the key being used)	If the blocksize isn't fixed, then no IVs will be used for encrypting/decrypting.
Greater than 0	Valid	Cypher processes input data in defined, fixed blocks (e.g. AES has a block size is 128 bits)	Must be a multiple of 8.

11.10.3 Miscellaneous Comments: Cypher Drivers

- When called upon to encrypt/decrypt data, if the "IVLength" passed in is 0, then "IV" should be ignored (it may be set to NULL)
- When writing a cypher driver, the encrypt/decrypt implementation should not write to the input buffer; only the output buffer (i.e. when encrypting, do not write to the plaintext buffer passed in; when decrypting, do not write to the cyphertext buffer). An optimisation in the driver involves the use of a single buffer for input/output. You may find that you'll need to create a temporary buffer equal to your blocksize when implementing some modes of operation (e.g. CBC)
- **Important:** Don't read/write to the *input* plaintext/cyphertext buffer! Only the *output* ones!

11.11 Filename Extensions

Volume files, keyfiles, and all other files created and used by FreeOTFE can have ***any*** file extension you wish to give them (if any).

By default, FreeOTFE uses the following:

Extension	Description
.vol	Volume file
.cdb	Keyfile (aka Critical Data Block)
.cdbBackup	Critical Data Block Backup

12 Known Bugs

- LUKS volumes which use the Tiger hash cannot be mounted correctly
- Certain combined mobile phone/PDA devices have issues mounting volumes. See FAQ for details of the specific models affected.

Both of the above issues are currently under investigation, and should be fixed in a later version

If you believe that you have found a bug in FreeOTFE Explorer, it would be very much appreciated if you could get in touch and report it; please see the section on: Fault/Bug Reporting

13 Fault/Bug Reporting

Although FreeOTFE Explorer should be pretty stable and have no faults, it's always possible that you may find something not quite right. In these cases, it would be very much appreciated if you could email a fault report the address shown in the contact details.

If you have recently upgraded to a newer version of FreeOTFE Explorer, please could you ensure that you have followed the upgrade procedure exactly before reporting a fault. (See Installation and Upgrading from a Previous Version section)

When reporting a fault with FreeOTFE Explorer, please include as much detail as possible, preferably including as much of the following as possible (Note: Not all of the following may apply):

1. What OS you're running under.
2. Any service packs which have been applied to your OS.
3. The FreeOTFE Explorer executable version (See the "Help | About..." dialog).
4. The main FreeOTFE Explorer driver version (See the "Help | About..." dialog).
5. Details of how the volume in question was created (e.g. the summary shown on the last stage of creating a FreeOTFE volume).
6. The size of any volume file/partition involved.
7. If using a volume file, the filesystem used on the drive the volume file is stored on (e.g. NTFS/FAT/FAT32).
8. The filesystem the volume is formatted as (e.g. NTFS/FAT/FAT32).
9. A copy of the CDB dump taken from any volume file involved (See "Tools | Critical Data Block | Dump to human readable file...").
10. If a keyfile is being used, a copy of the keyfile and password used.
11. A small test volume (e.g. 1MB) which can replicate the problem found (Note: Please do not email volume files, unless asked to! An FTP site is available for uploading these)

Some of the items listed above may include potentially sensitive data. In which case, feel free to omit that information - or better still, create a simple test case which replicates the problem, but doesn't include any such data.

Additionally, if you are having problems with the PDA version of FreeOTFE, please could you also include:

1. The make and model of device you are using
2. The exact version number of your OS, which can be found by:
    1. Tapping the "Start" button at the top left of your screen
    2. Tapping "Settings"
    3. Tapping on the "System" tab of the dialog displayed
    4. Tapping on the "About" icon; this will give full details of your OS version (e.g. v5.1.1702 (Build 14366.1.0.1))

14 TODO List

- Additional hash and cypher drivers (suggestions welcome)
- GPG integration; support for using GPG to generate random data and Linux AES multikey support (this only really need the GPG interface finishing off)
- Support for Linux key iterations (-C option)
- Other MAC algorithms (e.g. PMAC, OMAC)
- Command line utility written in C to carry out the main functions of the GUI.
- The source code could do with a little tidying up...
- (Constructive) suggestions welcome.

15 Appendix A: Version History

- v3.00 (22nd July 2009)
  - Optimised mount performance
  - Added copy and move functionality
  - Added rename functionality
  - Added support for dropping keyfiles onto the password entry dialog to simplify keyfile filename entry
  - Added XTS mode to Gladman cyphers (AES, MARS, RC-6, Serpent and Twofish)
  - Added RIPEMD-256 and RIPEMD-320 hash algorithms
  - Made numerous cosmetic and usability changes to main window, including:
    - Added "Date Created" and "Date Accessed" columns. (Note: As with MS Windows Explorer, these are not shown by default - rightclick on the column header and select to show them)
    - Added ability to move list view columns around
    - Added option to hide file extensions of known file types
    - Added option to hide the folders treeview (see menuitem "View | Explorer bar | Folders")
    - Added option to store/restore window layout on startup/exit (if settings are being saved)
  - Added support for forward/back/refresh multimedia keys
  - Added option to associate ".vol" files with FreeOTFE Explorer (Note that you are still free to use *any* filename with *any* filename extension)
  - Added "/create" command line option for creating new volumes
  - Added "/mount" command line option for mounting volumes
  - Added "/settings" command line parameter to read options from user specified location
  - Added automatic check for updates (turned *off* by default)
  - Created U3 and portableapps.com specific installers, simplifying installation for users of these particular systems
  - Changed to display more helpful message when attempting to mount NTFS volumes (as opposed to the more cryptic "FAT count 1 <= X <= 0"!)
  - Toolbar flickering eliminated when navigating the directory structure, etc
  - Removed restriction requiring hidden volumes to begin at an offset which is a multiple of 512 bytes
  - Improved support for FAT12/FAT16 filesystems
  - Fixed intermittant bug which could cause an exception on some systems when run from a non-root directory on slow storage devices (e.g. USB flash drives)
  - Fixed bug which left spurious LFNs on deletion
  - Fixed bug which prevented access to data after the first 4GB in a volume
  - Fixed minor bug which prevented hidden volumes from being automatically mounted and formatted after being created
- v2.10 (19th April 2009)
  - Added functionality to create simple plaintext disk images, as well as encrypted ones
  - Added LUKS support
  - Improved detection of different FAT types
  - Improved filesystem compatibility when formatting new volumes as FAT12/FAT16/FAT32, including:
    - Default filesystem now defaults to FAT16/FAT32, depending on volume size

        - Changed filesystem OEM name for more common one
        - Volume serial number randomized
        - Added FSInfo sector
        - Added filesystem backup sectors
    - Fixed bug in LFN handling, causing it to pad filenames when not needed, potentially resulting in "duplicate" directory entries
    - Fixed trivial bug introduced in v2.00, causing CDB dumps to report the hash driver used twice, instead of the hash driver and cypher driver used.
    - Fixed bug introduced in v2.00, which prevented per-volume IVs from being passed to the backend driver. Only affects CBC based volumes using per-volume IVs.
- v2.0 (4th April 2009)
    - Radical overhaul of user interface to provide much of the functionality provided by FreeOTFE
    - Combined FreeOTFE Explorer GUI code with existing FreeOTFE GUI code
    - Improved store and extract functionality
    - Added support for PKCS#11
    - Added support for language translations
    - Added ability to mount Linux volumes
    - Added ability to create new volumes
    - Added toolbar with most often used functions
    - Added ability to copy FreeOTFE Explorer to USB drive (or other removable media) via user interface
    - Added ability to create keyfiles
    - Added ability to change volume/keyfile passwords
    - Added ability to backup/restore/dump to human readable file volume CDBs
    - Added options dialog, and ability to save settings
    - Added forward/back navigation
    - Added properties dialog for items selected
    - Added context menu onto tree view, providing all functionality
    - Added drag/drop interface, allowing files and directories to be dropped onto FreeOTFE Explorer to store them
    - Added "mount readonly" option when mounting plaintext images
    - Added installer version
    - Added optional MRU list of FreeOTFE volumes mounted
    - Corrected FAT implementation to mark clusters in FAT as unused when deleting items.
    - Prevent user from deleting the root directory of mounted volume(!)
- v1.0 (1st March 2009)
    - Initial public release

16 Appendix B: Credits

Thanks go to:

- Tom St Denis (tomstdenis@iahu.ca) for the LibTomCrypt library (http://libtomcrypt.org/).
- Hi/fn and Counterpane Systems for the optimised reference implementation of Twofish (http://www.schneier.com/twofish.html).
- Oliver Taylor for the PKCS#11 and partition display implementation, and his work on the drivers
- Peter Gutmann (pgut001@cs.auckland.ac.nz) for the cryptlib library which (if installed) FreeOTFE can take advantage of an RNG.
- Clemens Fruhwirth (http://clemens.endorphin.org/), author of the LUKS specification and ESSIV.
- David Saunders (dsaunders@gawab.com) for setting up the FreeOTFE.org domain registration and WWW site.
- Dr. Brian R. Gladman (gladman@seven77.demon.co.uk) for his implementation of the 2nd round AES candidates
- Lars B. Dybdahl (Lars@dybdahl.dk) and Peter Thörnqvist (peter3@peter3.com) for dxgettext, an Delphi port of GNU's gettext

---

16.1 Translations

Language	Translator
German	Volkmar Brandmaier (brandmaier@gmx.net)
Italian	Daniele (fuzzee@virgilio.it)
French	Daniel Berthereau (Daniel.fr@Berthereau.net)
Spanish	Luis Santalla (luis.santalla@hotmail.com)

17 Appendix C: Licence

17.1 Preamble

The following licence covers FreeOTFE, FreeOTFE4PDA and FreeOTFE Explorer (including source, executables, documentation and all related) with the exception of the files stored in the source release under the "src\3rd_party" directory which are covered by the separate licences included with them (e.g. libtomcrypt is public domain software, Twofish is uncopyrighted and license-free):

The primary purposes of this licence are:

1. To ensure that any improvements made to FreeOTFE/FreeOTFE4PDA/FreeOTFE Explorer can always be incorporated into the main releases
2. To ensure proper credit is given to these projects
3. To protect against modified versions of the software claiming to be FreeOTFE/FreeOTFE4PDA/FreeOTFE Explorer, or be named such that they may be confused with them.

17.2 FreeOTFE Explorer Licence (v1.1)

1. Definitions
    1. Original Author: The individual, organization or legal entity identified in Exhibit A
    2. Original Work: The Source Code, documentation and built executables relating to the software identified in Exhibit B
    3. Derivative Works: Any modified version of the Original Work (or product which uses any part of the Original Work) and any subsequent Derivative Works.
    4. The Software: The Original Work or any Derivative Works
    5. You: Any individual, organization or legal entity wishing to use The Software for any purpose.
    6. Source Code: The complete source code including all related build scripts, images and components required to build the source code into an executable version
2. Grant of license
    1. Subject to the terms and conditions of this licence You are granted a world wide, royalty free, non exclusive licence to use, copy, distribute, display and modify The Software released under this licence.
3. Modifications and Derivative Works
    1. The Software may be modified in order to produce Derivative Works subject to the following:
        1. Derivative Works in executable form must prominently display the notice detailed at Exhibit C whenever the Derivative Work is started and in any display and documentation where copyright is asserted for the Derivative Works
        2. Derivative Works may not be named or be identified using any of the names specified in Exhibit D nor use any similar sounding or potentially confusing name
        3. Derivative Works may not use any of the names specified at Exhibit D or that of the Original Author to promote or endorse the Derivative Works without the prior written permission of the Original Author.
    2. Derivative Works in executable form may only be distributed provided that:

        1. Such distributions include a copy of this licence
        2. The Source Code used to produce the Derivative Works must be released under this licence
        3. The Source Code used to produce the Derivative Works must be made freely available to anyone who wishes a copy (including but not limited to the Original Author any recipient of the Derivative Works) for a minimum period starting from the time at which the Derivative Works are initially distributed to at least two years after the Derivative Works cease to be distributed.
4. Termination
    1. This licence terminates automatically should You:
        1. Fail to comply with any of the term of this license
        2. Commence any action including claim, cross-claim or counterclaim against the Original Author
5. Disclaimer of Warranty
    1. THE SOFTWARE COMES WITH NO WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED

17.2.1 Exhibit A

The Original Author is:

- Sarah Dean

17.2.2 Exhibit B

The works covered by this licence are:

- FreeOTFE
- FreeOTFE4PDA
- FreeOTFE Explorer

17.2.3 Exhibit C

The notice to be shown is:

> This software is based on FreeOTFE and/or FreeOTFE4PDA, the free disk encryption system for PCs and PDAs, available at www.FreeOTFE.org

17.2.4 Exhibit D

The restricted names are:

- FreeOTFE
- FreeOTFE4PDA
- FreeOTFE Explorer

18 Appendix D: Glossary

CDB
Acronym: Critical Data Block
See "Critical Data Block"

Critical Data Block
AKA "CDB"
A block of data holding data vital to the correct mounting and use of an encrypted partition (volume). Among other things, a volume’s CDB contains the master key used for encrypting/decrypting data as it is written/read from a the volume. CDBs are encrypted. A full description of what FreeOTFE stores in its CDBs can be found in the Technical Details section.

LES file
Linux Encryption Settings file; a text file in which encryption settings for Linux volumes are held. Created by using the "Load..."/"Save..." buttons on the password entry dialog when mounting Linux volumes.

OTFE
Acronym: On The Fly Encryption
Any of a number of encryption systems where data is stored on disk in encrypted format. When it is read in from disk, it is transparently decrypted "on the fly" before user applications receive it. In a similar manner, when applications, etc write data back to the disk, it is automatically encrypted before being written.

19 Appendix E: PKCS#11 Driver Libraries

If you have a token which supports the PKCS#11 standard, as most do, your token can be used by FreeOTFE Explorer. Below is a list of well known smartcards/tokens which do support this standard, and the suggested library filename to use.

Please note that:

1. Token manufacturers may change their driver filenames without notice; check with your supplier if the information listed below doesn't work.
2. This list is ***not exhaustive*** - many more tokens are supported than are listed here.

If you are using a token which isn't mentioned on the list below, please check with your token supplier as to what to enter, and get in touch to have it added to the list.

Manufacturer	Device	Library	WWW Site	Notes
ACS	ACOS5 smartcards	acospkcs11.dll		For use with ACOS5 smartcards. ACOS5 smartcards unable to create data objects, and contain no mechanisms for creating secret keys (except public/private keypairs)
AET	Rainbow iKey 3000 series and G&D StarCos 2.3 SPK cards	aetpkss1.dll	(WWW)	aka StarSign Middleware. Also installs PKCS#11 drivers aetpksse.dll (Entrust PKCS#11 lib) and aetpkssw.dll (PKCS #11 library wrapper that detects the need for automatic login on aetpkss1.dll)
Aladdin	eToken PRO	etpkcs11.dll	(WWW)	Confirmed working with FreeOTFE
	eToken R2	etpkcs11.dll	(WWW)	
Algorithmic Research	MiniKey	sadaptor.dll		

Aloaha	Smart Card Connector	aloaha_pkcs11.dll	(WWW)	Not fully tested, but from initial checks, this library appears *badly broken* - it reports tokens present when no card reader is installed. When creating data objects on the phantom token, it reports that the objects were *successfully created*, but doesn't actually *do anything*! Possible that it works correctly if both hardware and token present?
A-Sign	A-Sign premium cards	psepkcs11.dll		Reputedly v1.0 of this library is incomplete and has many bugs??
Athena	Athena Smartcard System ASE Card	asepkcs.dll		
A-Trust	a-sign	asignp11.dll	(WWW)	
Belgian Government	Belgian Electronic Identity (eID) Card	Belgium Identity Card PKCS11.dll		
Chrysalis		cryst32.dll	(WWW)	WWW site no longer exists
	LUNA	cryst201.dll	(WWW)	WWW site no longer exists
Dallas Semiconductors	iButton	dspkcs.dll		

Eracom	(hardware)	cryptoki.dll		Full path to library may be required, e.g: C:\ Program Files\ ERACOM\ Cprov SDK\ bin\ csa\ cryptoki.dll On 1st December2005 Eracom Technologies AG was acquired by SafeNet
	(software emulation)	cryptoki.dll		Full path to library may be required, e.g.: C:\ Program Files\ ERACOM\ Cprov SDK\ bin\ sw\ cryptoki.dll On 1st December2005 Eracom Technologies AG was acquired by SafeNet
Estonian Government	Estonian Electronic Identity (eID) Card	opensc-pkcs11.dll		
Eutron	Crypto Identity	sadaptor.dll	(WWW)	
Feitain technologys Co.,Ltd	ePass 1000	EP1PK111.DLL	(WWW)	
	ePass 2000	ep2pk11.dll	(WWW)	
	ePass 2000_FT11	ngp11v211.dll	(WWW)	Confirmed working with FreeOTFE
	ePass 3000	ngp11v211.dll	(WWW)	
	ePass 3003	ShuttleCsp11_3003.dll	(WWW)	Confirmed working with FreeOTFE
Gemplus		gclib.dll		
		pk2priv.dll		
GemPlus/GemSoft	GemPlus/GemSoft Smartcard	w32pk2ig.dll		
GemSafe		gclib.dll		GemSafe new version
		pk2priv.dll		GemSafe old version

IBM	IBM 4758	cryptoki.dll		
	IBM Digital Signature for the Internet (DSI) for MFC cards	CccSigIT.dll		
	IBM Embededded Security Subsystem	csspkcs11.dll		
	IBM Netfinity PSG Chip1	ibmpkcss.dll		
	IBM SecureWay Smartcard	w32pk2ig.dll		
		cryptoki.dll		
ID2		id2cbox.dll		Full path to library may be required, e.g.: C:\ Data\ Development\ SmartCardIntegration\ PKCS11wrapper\ JavaToPKCS11\ demo\ test\ old\ id2cbox.dll
Mozilla/Netscape	Mozilla or Netscape crypto module	softokn3.dll		Cannot be used with FreeOTFE; requires additional parameters to initailize
nCipher	nFast	cknfast.dll	(WWW)	
	nShield	cknfast.dll	(WWW)	
Nexus		nxpkcs11.dll		
OpenSC	(multiple)	opensc-pkcs11.dll	(WWW)	Free, open source library
Orga Micardo		micardoPKCS11.dll		

Rainbow	CryptoSwift Accelerator	Cryptoki22.dll		
	CryptoSwift HSM	iveacryptoki.dll		
	Ikey 1000	cryptoki22.dll		From wiki.cacert.org: for USB use Datakey driver
	iKey 1000/1032	k1pk112.dll	(WWW)	
	iKey 2000 series and for DataKey cards	dkck201.dll		
	iKey 2000/2032	dkck232.dll	(WWW)	
	iKey 2032	dkck201.dll		
		cryptoki22.dll		From wiki.cacert.org: for USB use Datakey driver
Safelayer	HSM	p11card.dll		From wiki.cacert.org: for USB use Datakey driver
Schlumberger	Cryptoflex	acpkcs.dll	(WWW)	
	Cryptoflex	slbck.dll	(WWW)	
	Cyberflex Access	slbck.dll	(WWW)	
SeTec	SeTokI cards	SetTokI.dll		Full path to library may be required, e.g.: C:\ Program Files\ Setec\ SetTokI\ SetTokI.dll
Siemens	HiPath SIcurity Card	siecap11.dll		
	Some Siemens Card OS cards	eTpkcs11.dll		
SmartTrust		smartp11.dll		
Spyrus		SpyPK11.dll	(WWW)	
Utimaco	Cryptoki for SafeGuard	pkcs201n.dll		

?	ActivCard cards	acpkcs.dll		
	ActivClient	acpkcs211.dll		
	Datakey	dkck201.dll		From wiki.cacert.org: for Entrust
	Datakey	pkcs201n.dll		
	Datakey CIP	dkck201.dll		
	Datakey/iKey	dkck232.dll		From wiki.cacert.org: NB: buggy, use 201
	Fortezza Module	fort32.dll		
	Oberthur AuthentIC	AuCryptoki2-0.dll		
	SCW PKCS 3GI 3-G International	3gp11csp.dll		
	TeleSec	pkcs11.dll		

The information listed above was compiled from multiple sources, including:

- CAcert Wiki: Pkcs11TaskForce
- StrongDisk Server (Russian documentation)
- cryptlib v3.1 testlib.c
- IAIKPkcs11.properties
- Using the IAIK JCE Provider for PKCS#11

20 Appendix F: Command Line Decryption Utilities

20.1 Overview

Note: The development of the command line decryption utilities has ceased. This functionality has been superceded with the development of FreeOTFE Explorer

FreeOTFE is relatively unique in that comes complete with software which may be used to decrypt encrypted volumes (provided the correct decryption key is known!).

This software is designed to fulfil two main objectives:

1. To increase and encourage peer review of FreeOTFE
2. To act as a "security blanket" for users - should development of FreeOTFE ever be dropped, it will still be possible for users to recover their data, regardless of the state of the FreeOTFE project.

Functionally, this software has one task: to decrypt the encrypted partition area of FreeOTFE volume files and to write out the plaintext version for examination.

This software is considerably easier to understand than the kernel mode drivers, and does not require the Microsoft SDK/DDK to be present. As a result, any competent software engineer should be able to modify the software as appropriate and confirm that data is being encrypted correctly by the FreeOTFE system.

This software is not intended for general public use, but by those who understand and can write C. In order to use it, modifications to the source code will most probably be required (to change the decryption keys used, if nothing else). For this purpose, the command line decryption utilities are not released in binary form, only as source code which must be compiled by the user.

20.2 Operation

Each of the command line decryption utilities is designed to operate in the following manner:

1. Open the (input) encrypted volume file.
   - The filename used is hard coded to "inFile.dat"; obviously this may be changed as required.
2. Open/Create the (output) plaintext volume file.
   - The filename used is hard coded to "outFile.dat"; obviously this may be changed as required.
3. Generate an IV, if required
   - The method of generating the IV may vary, dependant on how the volume was encrypted
4. Read in a sector’s worth of data from the input (encrypted) file
5. Decrypt the sector, block by block
   - The key used here is hard coded in the source, and must be the actual key that was used to encrypt the data (obviously!)
   - The way in which decryption is carried out is cypher, and cypher implementation dependant
6. Write the decrypted sector to the output (plaintext) file
7. Repeat steps 3-6 until all data has been decrypted
8. Close the output file

9. Close the input file

Please note:

1. This software is focussed only on decrypting data. They do not hash user keys, etc
2. The hard coded keys must represent the actual encryption keys. In the case of Linux volumes, this is the user’s password hashed as appropriate. In the case of FreeOTFE volumes, this is the "master key" stored in the volume’s "critical data block"

At time of writing, although a separate command line decryption utility to decode a FreeOTFE volume’s CDB/keyfiles has not been implemented, the FreeOTFE GUI does incorporate this functionality allowing developers to extract all of the information required contained within a CDB/keyfile. (Note: For obvious reasons, this requires the volume’s password and all other details that are required to use the CDB are known - it is simply not possible to decrypt this information otherwise)

21 Appendix G: Uninstalling

To uninstall FreeOTFE Explorer, please carry out the steps detailed in either of the sections below:

21.1 Automatic Uninstall

If installed via the installation wizard, FreeOTFE Explorer may be uninstalled by either:

1. Using the "Add and Remove Programs" control panel applet.
2. Running "uninstall.exe", found in the directory FreeOTFE Explorer was installed in

---

21.2 Manual Uninstall

1. Exit FreeOTFE Explorer, if running.
2. Delete the directory in which you installed/uncompressed FreeOTFE Explorer, and any shortcuts you may have created.

22 Appendix H: Contact Details

Please send any comments, feedback, etc to: sdean12@sdean12.org

If you would like to send PGP encrypted email, feel free to use the following public key block (download):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGP 8.1 - not licensed for commercial use: www.pgp.com

mQGiBDekay4RBADuX/QEq7W9POZ3V5xIGj4lpO4+LB+LhY2+ZMi0/jMtsMQg6niJ
ctPBjyKCmm0LwleojZyLo7ArJ1Bkw4dswYbaULEDC+nB4KEwOsJzfdceJ8jI063g
KFZylqpuhAdEKr0am1UScp91O9HndO/TvW+Tz26MGdpHZMSR6ppAISyhSQCg/+eD
Me3ocRdWgJRK3QFPqJ8sZ4sD/jOzuQoWEjcCS+9RCW3ciCbBdsdpeZ+JTSwwiurj
tQ+vOZVmDxx+rBuN2f20BqeXGA1ySaZBkO3FTEcctxN37v7lh7LVa2Has+RZyNL1
P35sTUgJW0/v6ZcevG4pTMhWjQWPQsUGKHVIvHekEDTy+aeYb3kUvZBWJVnQXgJC
3oc/A/9OzHJMDyyezEDdElqAggilPatwIM61WdtX6r0fWF+JG0KrrNCD3PS6FR3O
EVryJdhgqUnmgjUYap50w2IeAgcRronaI8rxaQJHpp2v213Tt8b2Bs/FkZ77AAzI
xLgvQUQkhL8GNf3gtEO8ATJXVB6yswTWgmOFzCHSMH4zYJziprQgU2FyYWggRGVh
biA8c2RlYW4xMkBzZGVhbjEyLm9yZz6JAEsEEBECAAsFAkPGE8oECwMCAQAKCRAt
2N+++s56l4HlAKC32vR75faqWiy9+pcaDkcsiWoy1wCdFl5khqK1wDDnscmFEeWM
7jPhxoe5Ag0EN6RrLhAIAPZCV7cIfwgXcqK61qlC8wXo+VMROU+28W65Szgg2gGn
VqMU6Y9AVfPQB8bLQ6mUrfdMZIZJ+AyDvWXpF9Sh01D49Vlf3HZSTz09jdvOmeFX
klnN/biudE/F/Ha8g8VHMGHOfMlm/xX5u/2RXscBqtNbno2gpXI61Brwv0YAWCvl
9Ij9WE5J280gtJ3kkQc2azNsOA1FHQ98iLMcfFstjvbzySPAQ/ClWxiNjrtVjLhd
ONM0/XwXV0OjHRhs3jMhLLUq/zzhsSlAGBGNfISnCnLWhsQDGcgHKXrKlQzZlp+r
0ApQmwJG0wg9ZqRdQZ+cfL2JSyIZJrqrol7DVekyCzsAAgIH/2eFQW6w68YNeovi
EvJt1i41orLsBIvbR8M3ql6ly75zdIScRCREtORSb2JNRXkfiqc7Fca6gnmqUBNU
woo4NJAjyEz7ADyUxcA+uAU57anyqZeXUhQUZ5s/wEU77nYBFx0fNh+SlsNF4yL8
z5vJQvwUGJLvLZs1Ap1RSGxYMWJk0kdBu2j5wS0el6FrpqoXzrJZpPUpiegUPOqY
hg54Q0ddn9ksGO4LJUClhIjX/y54Pu9cWrcO+mKJ2kSZOM6zCXqErIek3J4xYy/W
CxJHtXjLEg5y2XMam45qKC6CmKcFwJJiVufWSWSCqqVYG30f9g1w44AYmn/VCIXx
t6ygcr6JAEYEGBECAAYFAjekay4ACgkQLdjfvvrOepdPnACg1HGvpsdlEQkb8VZc
rWbmoZyMLoYAoPBjVMIiYkdWD2t0cKaR/de8Rjct
=Hejr
-----END PGP PUBLIC KEY BLOCK-----
```