CryptoTE / Enctain File Format v1.0

Introduction

This HTML document describes the file format used by CryptoTE. The editor saves text documents in an encrypted container file using the libenctain library. Enctain is short for ENCrypted conTAINer. An encrypted container can hold a set of enumerated binary subfiles with associated application-defined metadata.

Format Overview

Each encrypted container file consists of the following sections:

1	Unencrypted Header1	Fixed-length header data. Contains a magic signature to identify file format and version. Implicitly specifies the metadata encryption cipher and method.
2	Unencrypted Metadata	Variable-length key-value properties. They are application-defined and publicly readable. Thus they can be displayed e.g. in file listings without requiring the encryption key.
3	Encrypted Master KeySlots	Encryption context parameters: contains a master key which is encrypted using different user password-keys. Any valid user password can be used to open the container. Lots of parameters are needed to make the encryption key derivation secure.
4	Encrypted Header3	Fixed-length header containing metadata length and CRC32 checksum.
5	Encrypted Compressed Metadata	Variable-length key-value properties. Contains two distinct parts: global metadata properties and (local) SubFile metadata properties. Both global and SubFile-local allow any number of application-defined properties. Besides the application-defined properties, the metadata also contains some fixed properties: the SubFile's compressed and uncompressed size, encryption and compression methods, CRC32 checksum and possibly encryption key and initialization vector.
6	SubFile Data	SubFile binary data, possibly encrypted and compressed. No separators are needed anymore.

Detailed Format v1.0

All binary numbers are stored in little-endian encoding. uint is an abbreviation for unsigned integer.

Unencrypted Header1

Header1 is an unencrypted fixed-length header at offset 0 to identify the file format and version.

1	8 bytes	Signature	An eight byte string or binary magic signature to identify the file's type. The CryptoTE editor uses "CryptoTE" (without NULL string termination). The value used by Enctain can be changed using the function `SetSignature()`.
2	16 bit uint	Version Major	Currently (major = 1, minor = 0) which means v1.0. This version number also implicitly defines the following sections including encryption cipher and compression methods.
3	16 bit uint	Version Minor	
4	32 bit uint	Unencrypted Metadata Length	Length of the following unencrypted metadata in bytes.

16 bytes Total

Unencrypted Metadata

The first variable-length section of the file contains application-defined unencrypted metadata properties. These key-value pairs can be set, retrieved and enumerated using the functions `SetGlobalUnencryptedProperty()`,

SetGlobalUnencryptedProperty(), EraseGlobalUnencryptedProperty() and EraseGlobalUnencryptedProperty(). This section can also be omitted (Header1.MetadataLength == 0), if no properties are defined by the application.

Application-defined metadata properties are key-value pairs of opaque binary strings. Enctain will store any data values (including NULLs) and any amount (< 4GB in total). This way data like current window position and other dialog settings can be stored using some private binary structure.

These key-value lists are stored by Enctain using the following format. Each properties key-value consists of two (possibly binary) strings, the key and the value. The (binary) string data is prefixed with it's length encoded in a single byte:

Example:
The value "string" is encoded into the hexbytes 06 73 74 72 69 6e 67. Note the 06 string length at the beginning.

This encoding makes the functions to read and write variable-length strings very simple. Furthermore binary NULL (0 bytes) can also be stored in the strings, as they are not NULL-terminated.

If a string is longer than 255 bytes the length does not fit into the prefix byte. Therefore an "escape length" is introduced: 0xFF in the length field means "long string". The 0xFF is then followed by a 32-bit unsigned integer specifying the full length of the string. Therefore all strings with 255 bytes or longer have 5 prefix bytes specifying their length.

Example:
A string containing 'a' 1022 times is encoded: FF FE 03 00 00 61 61 *(1018(decimal) more 61s)* 61 61.

The variable-length unencrypted metadata section contains a list of concatenated key-value pairs, which represent the global properties of the container. The number of key-value pairs (!) is stored as a 32-bit uint as the beginning of the variable length structure. The total length of this section is defined by Header1.UnencryptedMetadataLength. The actual key-value pairs used are completely left up to the application.

CryptoTE currently uses the following unencrypted key-value properties:

Subject	User-defined subject text string from the container properties dialog.
Author	User-defined author text string from the container properties dialog. Initialized with the user's login name for new containers.
Description	User-defined description multi-line string from the container properties dialog.

Encrypted KeySlots Header2

The KeySlots header section contains all information needed to decrypt and validate the master key material when a valid user secret password is provided. Ultimately the master key is used to decrypt the following two encrypted header sections. This section is the Achilles' heel of the file's encryption security.

The container format supports multiple user password slots. Each slot can be used to decrypt the file. As the user password data is hashed (repeatedly), the original passwords cannot be reconstructed from the information in the container. They are also not stored.

For password hashing Enctain currently uses the PBKDF2 (Password-Based Key Derivation Function) from PKCS#5 v2 [RFC2898] with HMAC(SHA256) as hash function. All three algorithms are implemented by the mini-Botan library contained in Enctain.

PBKDF2 requires an arbitrary amount of random salt data and an iteration count. Random salt data in libenctain is always 32 bytes long and stored in the file. Iteration count is also stored and randomized in the range 1000 - 11000.

The "master key" material is 64 bytes of random data. It is generated by mini-Botan's random number generator modules. The master key material is generated when the first user key slot is added to the container.

Three PBKDF2-derivations of the master key material are calculated by Enctain. For all derivations the salt data and iterations count is stored in the header. The first derivation is 32 bytes of digest data which is stored in the container and used by Enctain to detect that the entered user password is correct. The second and third derivation is not stored: they initialize the encryption cipher of the following metadata section. The second derivation is 32 bytes long and is used as encryption key for the Serpent cipher; the third derivation, only 16 bytes long, is used as CBC-IV (Initialization Vector).

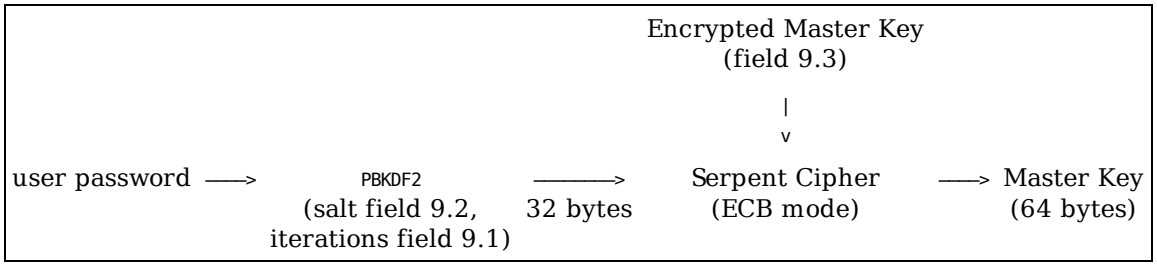
Structure

1		4 bytes uint	Master Key PBKDF2 Iterations for Digest	These three fields (1, 2, 3) are used for a "digest check" of the master key: PBKDF2(masterkey, salt, iterations) is used to calculate 32 bytes of digest data. This digest must be equal to the field "digest value". While loading a container this digest is compared to detect a valid user password.
2		32 bytes	Master Key PBKDF2 Salt for Digest	
3		32 bytes	Master Key PBKDF2 Digest Value	
4		4 bytes uint	Master Key PBKDF2 Iterations for Metadata Key	These two fields (4 and 5) are the parameters for another calculation of PBKDF2(masterkey, salt, iterations). Again a 32 bytes key is derived and is used as the Serpent cipher key for the following encrypted metadata section.
5		32 bytes	Master Key PBKDF2 Salt for Metadata Key	
6		4 bytes uint	Master Key PBKDF2 Iterations for Metadata IV	These two fields (6 and 7) are the parameters for yet another calculation of PBKDF2(masterkey, salt, iterations). This time 16 bytes of key material is derived and is used as the CBC initialization vector for the following encrypted metadata section.
7		32 bytes	Master Key PBKDF2 Salt for Metadata IV	
8		4 bytes uint	Number of KeySlots	Number of user KeySlots: number of times block 9 is repeated. Must be >= 1, otherwise container is not decryptable.
9	1	4 bytes uint	User KeySlot PBKDF2 Iterations	Lastly the most interesting part: The two fields (9.1 and 9.2) again initialize PBKDF2, but this time the password is the user-given "password string" entered in the dialog box. This PBKDF2(user-password, salt, iterations) calculates 32 bytes of key data. This key is used to initialize the Serpent cipher (256 bits) which is then used to decrypt the following encrypted copy of the master key (in ECB mode). After decryption the possibly-correct master key can be verified using the PBKDF2 Digest Iterations, Salt and Values (field 2,3 and 4).
	2	32 bytes	User KeySlot PBKDF2 Salt	
	3	64 bytes	User KeySlot Encrypted Master Key	Encrypted copy of the master key material.

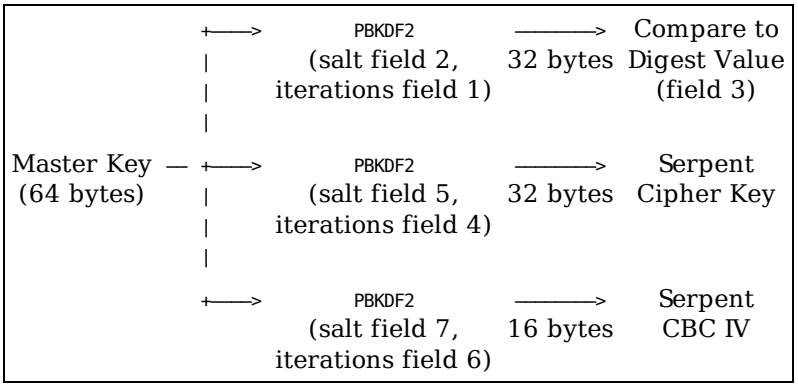
160 + n * 100 bytes Total

Diagrams

The following diagram shows how the user pasword string is used to decrypt the master key material.



And once the (possibly incorrect) master key material is decrypted, it must be checked against the digest value. If it is detected to be correct, then the Serpent cipher parameters are derived from it.



Encrypted Header3

Following the KeySlots is the first encrypted header. This header is 16-bytes long, exactly the block-size of the Serpent cipher which is used to encrypt it. Thus to read this block, the user must be queried for the encryption key, the Serpent cipher must be initialized with CBC-mode filter and the correct initialization vector set. The encryption key (256 bits) and CBC-IV are derived from the master key as described in the previous section.

1	4 bytes uint	Metadata Compressed Length	The length of the following variable metadata block. Because this metadata block is compressed using zlib, this value specifies the compressed length.
2	4 bytes uint	Metadata CRC32	CRC32 of the following variable metadata block. This is actually duplicated by zlib at the end of the compressed stream, though zlib uses the Adler32 checksum algorithm.
3	4 bytes uint	Padding 1	Zero.
4	4 bytes uint	Padding 2	Zero.

Encrypted Metadata

Following the encrypted header is a variable-length block of properties. The length of this section is defined in Header3. This section is encrypted using the Serpent cipher and compressed using zlib. The CBC-IV context continues from the the header. The compressed metadata stream is padded to the block length of the cipher; this is no problem because zlib ignores all data beyond the compressed image's end.

The compressed metadata contains two main parts: global metadata properties and (local) subfile metadata properties. These are combined into one section to make compression more efficient. Furthermore SubFile metadata is detached from the file data itself so that it is possible to read and display the metadata properties of all subfiles without reading the complete subfile data.

Special about this section is that fixed and variable data is mixed.

1		4 bytes uint	Number of Global Properties	Number of global property key-value pairs in the following variable length section.
2		variable	Global Metadata	Section holding all global encrypted properties. They are simply concatenated and their number is known from the previous field. These properties are completely application-defined.
3		4 bytes uint	Number of SubFiles	Number of SubFiles in the container, also number of times block 4 is repeated.
4	1	4 bytes uint	SubFile Storage Size	Size of the SubFile as stored in the container. This includes eventual encryption padding. Used to read the concatenated subfiles.
	2	4 bytes uint	SubFile Real Size	Size of the SubFile after decryption and decompression.
	3	4 bytes uint	SubFile Flags	Compound field holding the subfile's encryption cipher number and compression algorithm.
	4	4 bytes uint	SubFile CRC32	CRC32 value of the SubFile's real data. Used to verify decryption and decompression.
	5	4 bytes uint	Length of SubFile Cipher Parameters	Length of the following field 4.6.
	6	variable	SubFile Cipher Parameters	This field contains the encryption key data and CBC-IV for the subfile's cipher context. This field fully defines

8-bit uint	Compression Algorithm: 0 = none 1 = ZLib 2 = BZ2
8-bit uint	Encryption Algorithm: 0 = none 1 = Serpent
16-bit uint	reserved

			the encryption cipher parameters and is usually initialized with random data. Thus a change in the master key / user key slots does not effect the encryption parameters of SubFiles. The length is stored in field 4.5 and is dependent on the encryption/compression flags: For NONE encryption the length is 0 byte. For Serpent encryption the length is 48 bytes: first 32 bytes of random key data, followed by 16 bytes random CBC-IV.
7	4 bytes uint	Number of SubFile Properties	Number of (local) subfile properties.
8	variable	SubFile Properties	Variable length block holding all subfile properties. They are simply concatentated. Again these properties are completely application-defined.

CryptoTE currently uses the following global key-value properties:

CTime	Creation Time of the container. Stored as 4-byte time_t value.
MTime	Last Modification Time of the container. Stored as 4-byte time_t value.
DefaultCompression	Default compression algorithm for new SubFiles.
DefaultEncryption	Default encryption cipher for new SubFiles.
FileListDisplayMode	Private binary structure used to save the display mode of the file list.
FileListColumns	Private binary structure used to save the currently displayed columns in report file list mode.
RestoreView	Flag from global properties whether to restore text editor display settings.
SubFilesOpened	Array of SubFiles indexes opened in editor when the container was saved. Used to re-open the SubFiles on container reload.
KeySlot-*number*-Description	User-defined description of KeySlot *number*. Note that the Enctain format does not directly support "usernames" or other metadata for key slots. This is emulated by setting global properties.
KeySlot-*number*-CTime	Creation Time of the user KeySlot *number*. Stored as 4-byte time_t value.
KeySlot-*number*-ATime	Last Match Time of the user KeySlot *number*. Stored as 4-byte time_t value.

CryptoTE currently uses the following local SubFile key-value properties:

Name	Filename as displayed in file list. Note that this is a just a property and no lookup key. Thus file names are not required to be unique and cannot be searched for directly.
CTime	Creation Time of the SubFile. Stored as 4-byte time_t value.
MTime	Last Modification Time of the SubFile. Stored as 4-byte time_t value.
Filetype	Currently either "text" or anything else. If it is "text" the SubFile is opened using a text editor page, otherwise it is shown using a simple hexdump.
Author	Used-defined string in properties. Initialized with the login name upon SubFile creation.
Subject	Used-defined string in properties.
Description	Used-defined multi-line string in properties.
WTextPageSettings	Private binary structure used by the text editor page to save various display options like line-wrapping and line-numbers.

SubFile Data

Finally after all headers and metadata the actual SubFile data is appended. Each SubFile's storage length, encryption cipher and compression algorithm are defined in the compressed and encrypted metadata section. Thus no additional structuring is required. All SubFiles are simply stored concatenated.

To locate a specific SubFile's data it is necessary to know the beginning offset of all SubFile Data. To this

offset all preceding SubFile's StorageSize field must be added.

Example

The following table contains a hexadecimal dump of the beginning of an encrypted container file. The different header fields are color coded for better distinction in the explanation below:

Offset	Hexadecimal		ASCII
00000000	43 72 79 70 74 6f 54 45	01 00 00 00 41 00 00 00	CryptoTE....A...
00000016	03 00 00 00 06 41 75 74	68 6f 72 02 54 42 0b 44	.....Author.TB.D
00000032	65 73 63 72 69 70 74 69	6f 6e 11 53 6f 6d 65 20	escription.Some
00000048	6c 6f 6e 67 65 72 20 74	65 78 74 2e 07 53 75 62	longer text..Sub
00000064	6a 65 63 74 0c 54 65 73	74 20 45 78 61 6d 70 6c	ject.Test Exampl
00000080	65 ac 04 00 00 86 b4 6c	c7 0f 4c 30 36 f7 73 69	e......l..L06.si
00000096	e2 0a 76 64 99 e1 55 c7	04 17 a3 43 d6 a3 37 2b	..vd..U....C..7+
00000112	f7 f7 5c 26 89 f9 74 93	00 ef d1 4c 6b 68 73 11	..\&..t....Lkhs.
00000128	f8 54 e9 42 e4 b1 83 06	10 7c 35 47 3f 22 00 2c	.T.B.....|5G?".,
00000144	7f 82 a7 39 0a 89 0e 00	00 20 dd c6 99 74 b2 a0	...9..... ...t..
00000160	64 6c 01 e0 db 9c af 20	41 e9 1e 9a 8c 95 46 1d	dl..... A.....F.
00000176	8a 2b d8 87 16 70 00 8a	20 e1 16 00 00 c7 97 dc	.+...p.. ....
00000192	24 82 6f a4 4a 18 45 a4	71 7a 78 7f e7 56 9d 13	$.o.J.E.qzx..V..
00000208	56 6b f8 0b a3 c8 ed 52	72 d8 51 4e 2a 01 00 00	Vk.....Rr.QN*...
00000224	00 a0 0c 00 00 a9 6b de	d6 f1 59 0b 99 d1 f8 40	......k...Y....@
00000240	50 aa 2e 55 0e d1 9b b0	5b da 52 82 35 02 aa 10	P..U....[.R.5...
00000256	98 60 0d 51 03 ac b3 09	cb 96 41 ee 12 dc e4 c3	.`.Q......A.....
00000272	dc 37 6a 04 4c 13 ed e7	d5 3d 2c 59 f4 4f f4 f5	.7j.L....=,Y.O..
00000288	f5 9f 21 ec 8f e0 6b 10	e9 b5 5a 95 0e f5 6f 5b	..!...k...Z...o[
00000304	14 83 12 d3 b2 d6 6d 0b	df ad a0 dd e9 9d 77 2d	......m.......w-
00000320	f6 4e 1a 60 81 41 73 ad	71 0a 1d 0b d0 51 ff 73	.N.`.As.q....Q.s
00000336	40 f9 ca a2 09 b5 41 5e	e0 2b 91 c1 d1 b4 8a 7c	@.....A^.+.....
00000352	cc de 74 95 4c 21 49 fd	cb 24 92 61 4e e0 06 33	..t.L!I..$.aN..3
00000368	1c 52 54 53 24 1a 7e 6e	09 dc 14 10 27 0a 89 0d	.RTS$.~n....'...

Header1 shows the signature and version number v1.0. Following Header1 are 65 bytes of unencrypted metadata.

The unencrypted metadata contains the following 3 key-value pairs:

Subject	Test Example
Author	TB
Description	Some longer text.

After the unencrypted metadata the KeySlots header2 begins. The master key digest PBKDF2 iterations count is 1196 followed by 32 bytes of random salt and another 32 bytes which are the value of the digest. After the digest the PBKDF2 parameters for Serpent key and CBC-IV are stored: 4 bytes iteration and 32 bytes salt. Iterations count for the key are 3721 and for the CBC-IV 5827.

Last block in the keyslots header holds 1 user key slot: it contains an iteration count of 334, 32 bytes of random salt and the encrypted master key copy.

Now the encrypted part begins. Here the data is not random salt and keys, but actual data though unreadable. The first 16 bytes are the encrypted header3 followed by encrypted and compressed metadata. The example dump stops here, because all following data is encrypted and thus incomprehensible.